Botnet

The Mostly Dead Mozi and Its' Lingering Bots

Alex.Turing, Hui Wang, Genshen Ye

Aug 30, 2021 • 10 min read

Background

It has been nearly 2 years since we (**360NETLAB**) first disclosed the Mozi botnet in December 2019, and in that time we have witnessed its development from a small-scale botnet to a giant that accounted for `an extremely high percentage of IOT traffic at its peak`.

Now that **Mozi's authors have been taking custody by law enforcement agencies**, in which we provided technical assistance throughout, we don't think it will continue to be updated for quite some time to come. But we know that Mozi uses a P2P network structure, and one of the "advantages" of a P2P network is that it is robust, so even if some of the nodes go down, the whole network will carry on, and the remaining nodes will still infect other vulnerable devices, that is why we can still see Mozi spreading.

Many security vendors have tracked and analyzed Mozi, but from our point of view, there are some omissions and even mistakes. So here is our provide some updates to complement the security community's analysis; and to conclude our ongoing focus on the Mozi botnet.

This article will answer the following questions.

1: Does Mozi have any functional nodes other than the Bot node?

2: Are there any new features in the Mozi Bot module?

3: Is the Mozi botnet still being updated?

What are the other functional nodes in the Mozi botnet besides the Bot?

As we all know, each node in the Mozi botnet is driven by a configuration file called Config issued by the Botnet Master to perform specific tasks. The following figure is a classic Config file, where the `[ss]` field describes the function of the node, in this case the `Bot node`, the main function is DDoS attacks.

```
[ss]botv2[/ss][dip]192.168.2.100:80[/dip][hp]88888888[/hp][count]http://ia.51.la/go1
[/count]..........................................................................
```

What puzzled us was that, in addition to the Bot node's Config, the following forms of Config files were captured as well, indicating that there were also nodes named **sk,ftp,sns,ssh** in the Mozi botnet.

```
[ss]sk[/ss][hp]88888888[/hp][count]http://ia.51.la/go1?id
[ss]ftp[/ss][hp]88888888[/hp][count]http://ia.51.la/go1?id
[ss]sns[/ss][hp]88888888[/hp][count]http://ia.51.la/go1?id
[ss]ssh[/ss][hp]88888888[/hp]
```

So what exactly are they?

0x1: FTP node

On January 20, 2020, a Windows PE file named "photo.scr" (a9d4007c9419a6e8d55805b8f8f52de0) generated network traffic that matched our Mozi signature. At first we thought it was a false alarm, but after analyzing it, we determined that it was exactly **the Mozi ftp node sample we had in mind**.

In order to distinguish the samples from the different functional nodes in the Mozi botnet, we started to use the `Mozi_"ss value"` internally, so this sample was named `Mozi_ftp`.

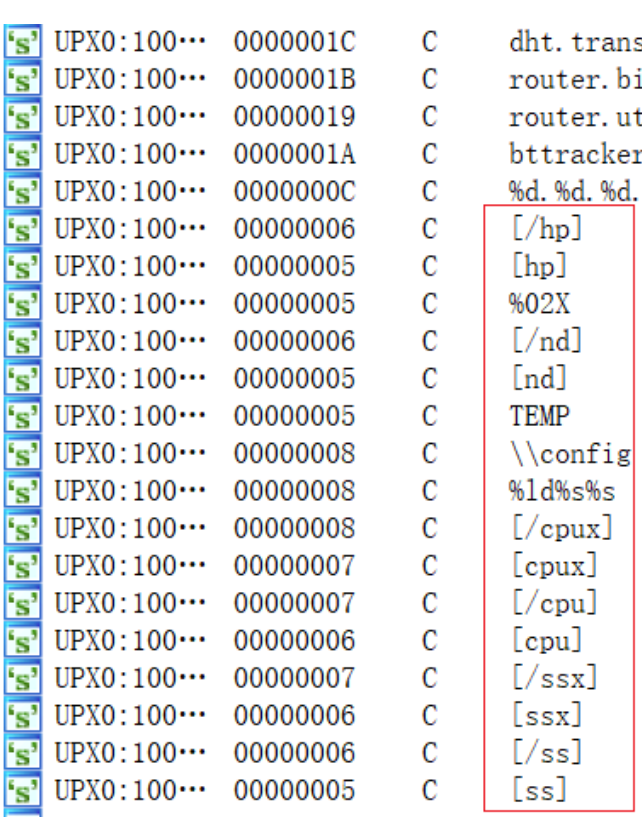
In short, Mozi_ftp is a pyinstaller-packaged mining trojan that spreads through FTP weak password, and it joins the Mozi P2P network and waits to execute the Config issued by Botnet Master. the wallet address is shown below:

```
47BD6QNfkWf8ZMQSdqp2tY1AdG8ofsEPf4mcDp1YB4AX32hUjoLjuDaNrYzXk7cQcoPBzAuQrmQTgNgpo6XP
```

The module named **back.jpg** is responsible for joining the Mozi network as well as pulling the Config file, and its basic information is shown as follows.

> *Filename:back.jpg*
> *MD5:4ae078dd5085e97d3605f20dc079412a*
> *PE32 executable for MS Windows (DLL) (console) Intel 80386*
> *Packer: upx*

Some of the tags supported by Mozi_ftp Config can be clearly seen in the unpacked sample.



Like Mozi_bot, Mozi_ftp als

o h a s a n e n c r y p t e d r a w C o n f i g f i l e e m b e d d e d , w h i c h i s d e c r

```
[ss]ftp[/ss][cpu].w[/cpu]
[hp]88888888[/hp].....
.....
......pw.OÙß¬WÀCüÌýý>¼..ov2%M.ãR..K.j.¥..dJ.§aâÓ·b<.R{
.µj..þ.Ñ.ncÀ3µ.'¨..Ô²¶m.ð¡(è.qz´ó¤ðÿnuð..(|..ìÕOo~
```

ypted by yXORas follows Aswith Mozi_bot, Mozi_ftpchecksth

e
si
g
n
a
t
u
r
e
o
f
t
h
e
C
o
n
fi
g
w
it
h
t
h
e
f
ol
lo
w
in
g
c
o
d
e
s
ni
p
p
e
t

```
loc_100098BE:
mov     dword ptr [esp+4], 31h
mov     dword ptr [esp], offset public_key
call    _Z7xor_encPhi
mov     dword ptr [esp+4], 31h
mov     dword ptr [esp], offset public_key2
call    _Z7xor_encPhi
mov     dword ptr [esp+8], 210h ; size_t
mov     dword ptr [esp+4], offset configserver ; void *
mov     dword ptr [esp], offset dword_10022450 ; void *
call    memcpy
mov     dword ptr [esp+8], 4 ; size_t
mov     dword ptr [esp+4], offset dword_1002265C ; void *
lea     eax, [ebp+var_1D0]
mov     [esp], eax      ; void *
call    memcpy
mov     dword ptr [esp+8], 60h ; size_t
mov     dword ptr [esp+4], offset dword_1000C4D0 ; void *
mov     dword ptr [esp], offset signature ; void *
call    memcpy
mov     dword ptr [esp+8], offset signature
mov     dword ptr [esp+4], offset configserver
mov     dword ptr [esp], offset public_key
call    ecdsa_verify
mov     [ebp+var_34], eax
cmp     [ebp+var_34], 1
jnz     short loc_10009975
```

The XOR key used, and the two public_keys are as follows.

```
 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
```

Their values are the same as those used by Mozi_bot. According to the characteristics of the ECDSA384 elliptic algorithm, this means that **Mozi_ftp and Mozi_bot use the same private key**, and excluding the possibility of private key leakage, we can conclude that **they are from the same author**.

In `back.jpg`, we can see that Mozi_ftp's Config supports the following basic tags.

```
[hp]
[cpu]
[cpux]
[ss]
[ssx]
[nd]
```

In the script of `ftpcrack.py`, there is the following code snippet.

```
[Code]
    File Name: ftpcrack.py
    Object Name: config
    Arg Count: 1
    Locals: 19
    Stack Size: 12
    Flags: 0x00000043 (CO_OPTIMIZED | CO_NEWLOCALS | CO_NOFREE)
    [Constants]
        None
        '.w'
        0x0L
        ''
        '\\config'
        0
        1
        '[mdf]'
        '[/mdf]'
        '|'
        8
        '/'
        -1
        '\\'
        'wb'
        ' '
        '[mdr]'
        '[/mdr]'
        2
        'win32'
        'taskkill /F /IM '
        '.exe'
        'killall -9 '
        'chmod +x %s'
        'no'
        '[mud]'
        '[/mud]'
        '\\updater.exe'
        '\\updater'
        '[mrn]'
        '[/mrn]'
        '$tmp'
        60
```

This shows that Mozi_ftp also implements the following 4 special tags of its own.

```
[mdf]
[mdr]
```

```
[mud]
[mrn]
```

0x2: SSH node

Mozi uses the 51la, public service platform for its own statistics, In September 2020, we were able to tape into Mozi's backend statistics, on which we see not only the statistics of the Mozi_bot node, but also a set of unseen reporting entries as shown below.

```
http://www.so.com//tmp/.work/work64/125.136.165.99:8014/.x64
http://www.so.com//usr/.work/work32/200.73.130.219:8016/.x86
http://www.so.com//tmp/.work/work64/121.169.34.252:8012/.x64
http://www.so.com//tmp/.work/work64/121.139.167.24:8012/.x64
http://www.so.com//tmp/.work/work64/121.149.39.183:8011/.x64
http://www.so.com//tmp/.work/work64/220.88.103.27:8017/.x64
http://www.so.com//usr/.work/work64/121.178.197.45:8017/.x64
http://www.so.com//tmp/.work/work64/220.88.103.30:8011/.x64
```

On August 18, 2021, security vendor **QiAnxin** and **Sangfor** issued threat reports describing a mining Trojan named **WorkMiner** was spreading through weak SSH password, and it has P2P network behavior. We took a look and were surprised to find that this is exactly the SSH node in the Mozi botnet, and it has direct link to the aforementioned 51la urls. Here we will call it `Mozi_ssh`.

The basic information of the sample we selected for analysis is shown below.

> *Filename:work64*
> *MD5:4292582700689668l3aa77efd5834a94*
> *ELF 64-bit LSB executable, x86-64, version 1 (GNU/Linux), statically linked, stripped*
> *Packer:upx*

In brief, Mozi_ssh is a mining trojan that spreads worm-like through SSH weak password, and became active around October 2020 (based on the sample's time on VT, which may not be accurate), with the wallet address shown below, which shows that Mozi_ssh and Mozi_ftp use the same wallet.

```
47BD6QNfkWf8ZMQSdqp2tY1AdG8ofsEPf4mcDp1YB4AX32hUjoLjuDaNrYzXk7cQcoPBzAuQrmQTgNgpo6XP
```

Mozi_ssh is compiled from a mix of GO code and C code. The GO code is responsible for SSH-related propagation and the handling of Config, while the C code handles joining the Mozi P2P network and pulling Config.

```
--------------------------------------------------------------------------------
Source File paths(Total number: 525, default print results are user-defind files):

/home/haha/work/go/sshworm/work/gossh.go
/home/haha/work/go/sshworm/work/pool.go
/home/haha/work/go/sshworm/work/http.go
/home/haha/work/go/sshworm/work/ip.go
/home/haha/work/go/sshworm/work/scp.go
/home/haha/work/go/sshworm/main.go
--------------------------------------------------------------------------------
```

Mozi_ssh is implemented by the following code snippet calling the C code (dht_task) to add to the P2P network.

```
main__Cfunc_GetConf proc near           ; CODE XREF: .text:0000000006C870D↓j
                                        ; main_main+865↓p
                                        ; DATA XREF: ...

var_28          = qword ptr -28h
var_20          = qword ptr -20h
var_10          = qword ptr -10h
var_8           = qword ptr -8
arg_0           = qword ptr  8
arg_8           = dword ptr  10h

                mov     rcx, fs:0FFFFFFFFFFFFFFC0h
                cmp     rsp, [rcx+10h]
                jbe     short loc_6C8708
                sub     rsp, 28h
                mov     [rsp+28h+var_8], rbp
                lea     rbp, [rsp+28h+var_8]
                mov     [rsp+28h+arg_8], 0
                lea     rax, [rsp+28h+arg_0]
                mov     [rsp+28h+var_10], rax
                mov     rax, cs:off_CFD118
                mov     [rsp+28h+var_28], rax
                mov     rax, [rsp+28h+var_10]
                mov     [rsp+28h+var_20], rax
                call    loc_403BC0
                cmp     cs:byte_F26336, 0
                jnz     short loc_6C86EC

loc_6C86E2:                             ; CODE XREF: main__Cfunc_GetConf+76↓j
                mov     rbp, [rsp+28h+var_8]
                add     rsp, 28h
                retn
```

```
off_CFD118      dq offset sub_6CD3F0
```

```
sub_6CD3F0 proc near
; __unwind {
push    r12
push    rbp
push    rbx
mov     rbx, rdi
call    _cgo_topofstack
mov     rdi, [rbx]
mov     r12, rax
call    dht_task
mov     ebp, eax
call    _cgo_topofstack
sub     rax, r12
mov     [rbx+rax+8], ebp
pop     rbx
pop     rbp
pop     r12
retn
; } // starts at 6CD3F0
sub_6CD3F0 endp
```

The function dht_task handles the same logic as Mozi_bot, and the embedded Config is decrypted as shown below.

```
[ss]ssh[/ss][cpu].x[/cpu]
[hp]88888888[/hp].....
.....
.....?Ä¯þf.êê.±=×6cç+.0.ùô´Ha¼ª..ºE..9A=Ö
.õÔ......Ü ü.B×¸8Ói G.~.p¯.Aû..æ
Õ..ßEÉ¼¨Eª9õ5û...êþnîÐKÜëoÕOo~
```

Like Mozi_ftp, the XOR key used to decrypt the Config, and the two public_keys used to check the signature of the Config are the same as the ones used in Mozi_bot, which means that **Mozi_ssh and Mozi_bot come from the same author**.

In the `dht_task` function, it can be seen that the Config of Mozi_ssh supports the following basic tags.

```
[hp]
[ver]
[cpu]
[ss]
[sv]
[nd]
```

For the Config that passes the test, Mozi_ssh uses to process it through the function main_deal_conf, for example, the following code snippet is processing the swan tag. Compared to Mozi_bot, Mozi_ssh supports not only basic tags, but also implements many of its own special tags.

The special tags supported by Mozi_ssh are shown below.

```
[slan]
[swan]
[spl]
[sdf]
[sud]
[ssh]
[srn]
[sdr]
[scount]
```

0x3: Summary

The discovery of Mozi_ftp, Mozi_ssh gives us clear evidence that the Mozi botnet is also trying to profit from mining. From the samples of `bot, ftp, and ssh nodes`, we can see that their authors have used the "DHT+Config" model as a basic module, and by reusing this module and designing different special tag commands for different functional nodes, they can quickly develop the programs needed for new functional nodes, which is very convenient. This convenience is one of the reasons for the rapid expansion of the Mozi botnet.

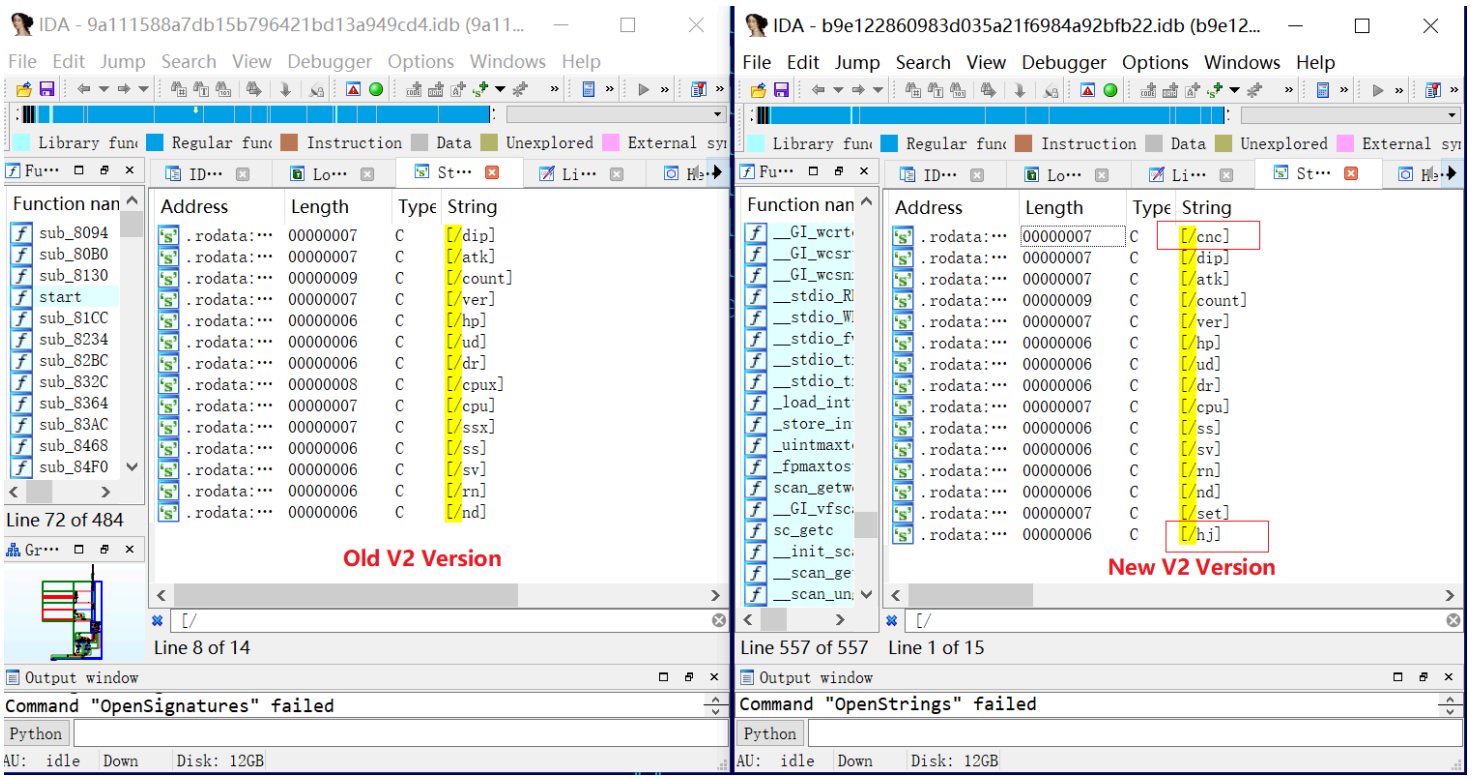
What are the changes in Mozi bot v2s?

The Mozi botnet was mostly made of mozi_bot nodes. On January 07, 2020, we captured the bot sample with a version number v2s (1bd4f62fdad18b0c140dce9ad750f6de), and this version has been active since then and has attracted a lot of attention from the security community, and although many security vendors have analyzed it, we found that there are still missing parts.

According to statistics, the samples of mozi v2s bot are mainly ARM and MIPS CPU architectures, and the samples of ARM architecture are selected below for analysis.

> *MD5:b9e122860983d035a21f6984a92bfb22*
> *ELF 32-bit LSB executable, ARM, version 1 (ARM), statically linked, stripped*
> *Lib: uclibc*
> *Packer:UPX*

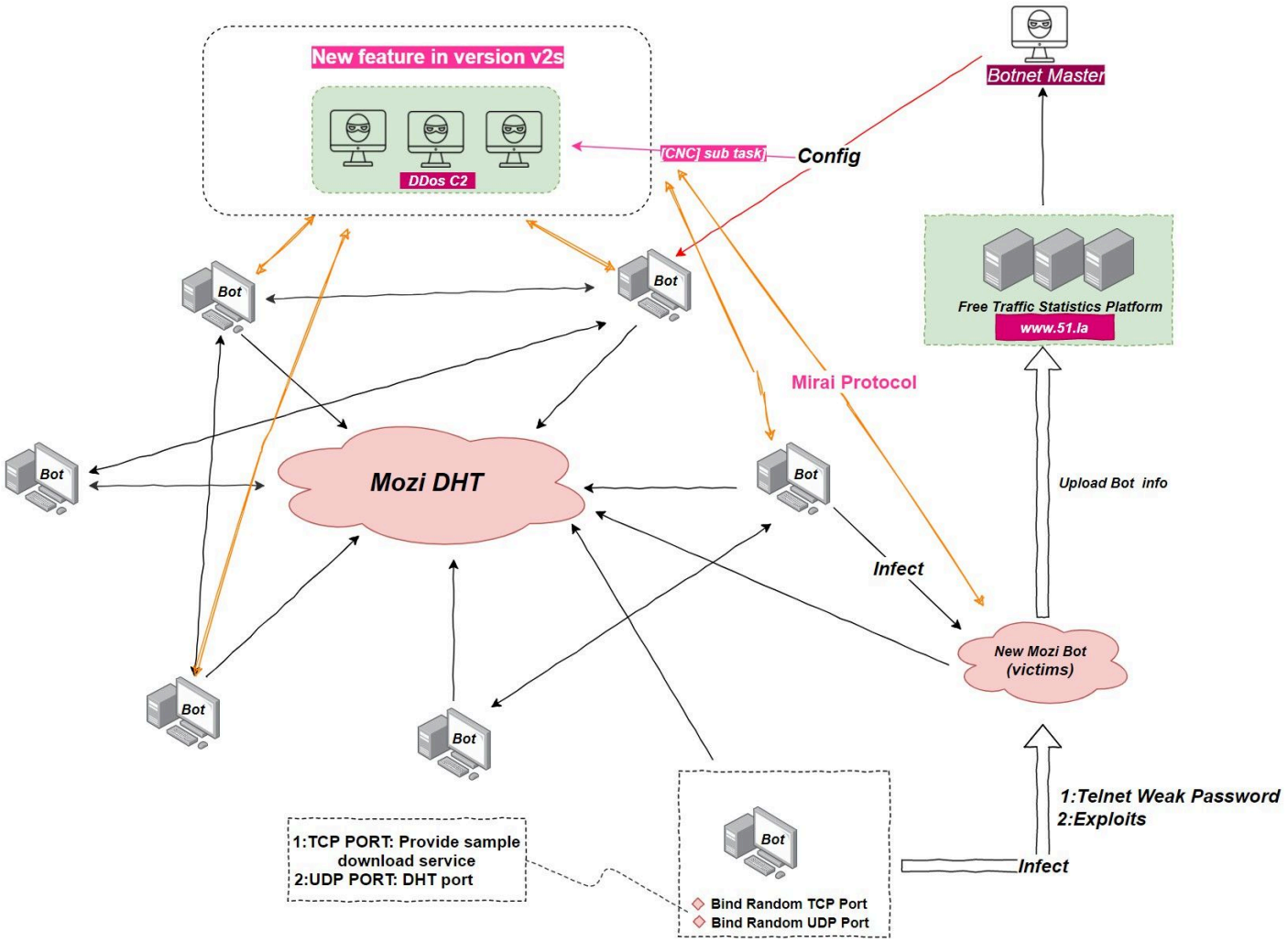
The v2s bot sample has many changes from the v2 sample we initially analyzed, the most intuitive of which is reflected in the tags supported by Config. v2s has added two new tags **[cnc], [hj]**, in addition to the new external network address acquisition, upnp port mapping and other features, the following section we will go over the changes brought by these features to Mozi_bot, note Microsoft published an article on the `[hj]` tag on August 19, 2021 here.



0x1: [cnc] tag

The Mozi botnet's "DHT+Config" design has its convenience, but it also has a drawback that all Bot nodes have `inefficiency in synchronizing Config`, which indirectly leads to inefficiency in DDoS. To solve this problem, Mozi authors designed the tag [cnc], which corresponds to the new DDoS attack subtask.

The whole subtask reuses the code of Mirai, specifying C2 by the [cnc] keyword, and the Bot node waits for the command sent by C2 to perform DDos attack after establishing communication with C2 through Mirai protocol. After adding this subtask, when Mozi wants to carry out an attack, it no longer needs to obtain the attack target by synchronizing Config one at a time, but only needs to synchronize Config once to get the specified C2, which greatly improves the attack efficiency of Bot nodes, and the corresponding network structure is shown as follows.



The following is the code snippet to obtain C2:PORT.

```
v19 = splittwo(&unk_54438, 625, "[cnc]", "[/cnc]", &v56);
if ( v19 )
{
  c2 = _GI_strtok(v19, (int)":");
  if ( !c2 )
    goto LABEL_33;
  v20 = _GI_strtok(0, (int)":");
  v21 = v20;
  if ( v20 && is_digit(v20) )
  {
    lib_atol(v21);
    dword_4A1C8 = v22;
  }
}
```

The following is the code snippet for sending online packets.

```
  _GI_sleep(v15 + 1);
}
else
{
  v29 = lib_util_strlen((unsigned __int8 *)&v46);
  LOBYTE(v53) = v29;
  dword_7115C = lib_util_local_addr(v29);
  _libc_send(cnc_socket, (const char *)&unk_36A00, 4u, 0x4000, &v50);// init msg
  _libc_send(cnc_socket, (const char *)&v53, 1u, 0x4000);
  if ( (_BYTE)v53 )
    _libc_send(cnc_socket, (const char *)&v46, (unsigned __int8)v53, 0x4000);
}
```

```
unk_36A00       DCB     0
                DCB     0
                DCB     0
                DCB     1
                DCB     0
```

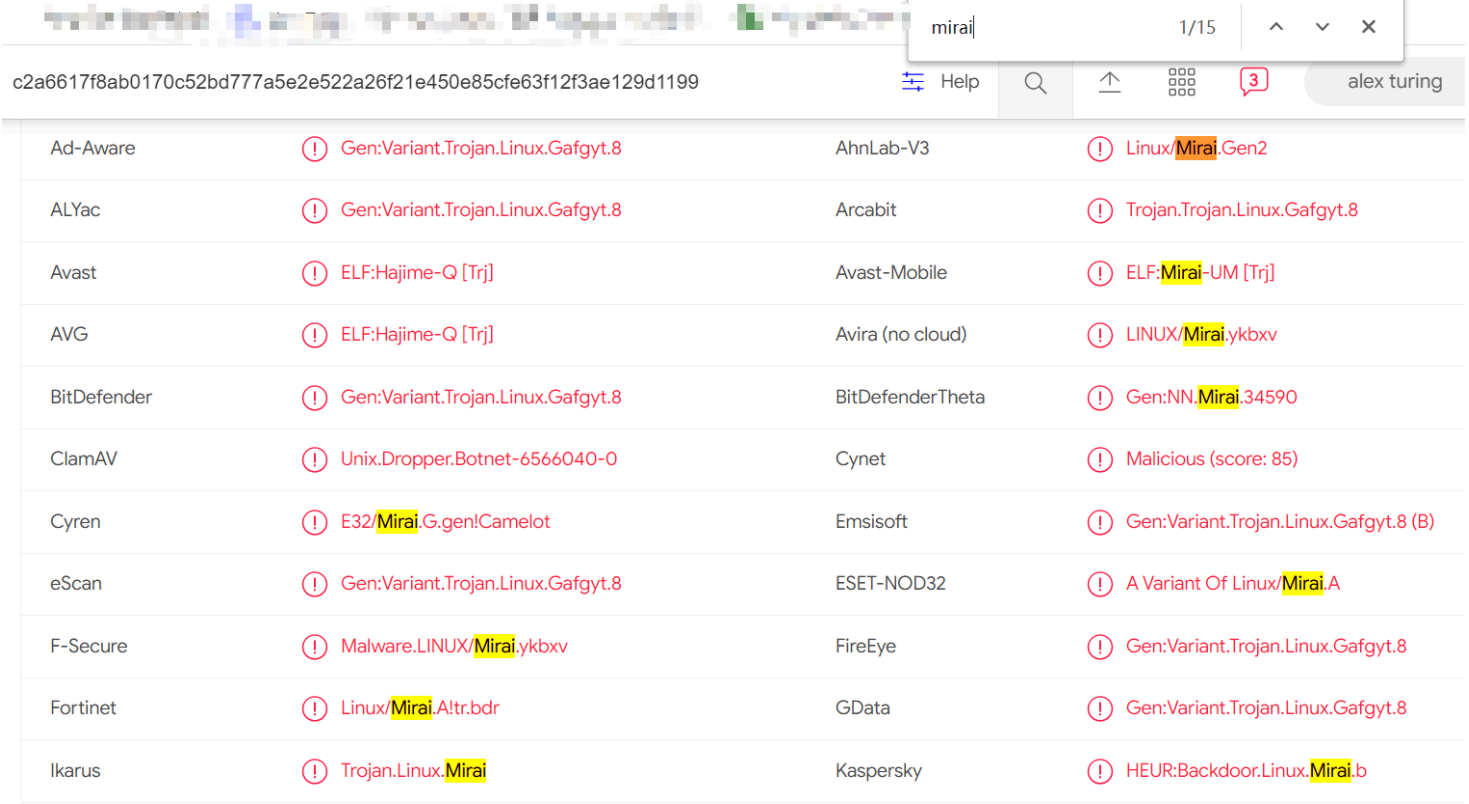
The following is the code snippet for sending heartbeats.

```
v9 = _libc_select(v8 + 1, &v44, &v43, 0);
if ( v9 != -1 )
{
  if ( !v9 )
  {
    v53 = 0;
    v10 = lib___modsi3(v41++, 6);
    if ( !v10 )
      _libc_send(cnc_socket, (const char *)&v53, 2u, 0x4000, &v48);// heartbeat
  }
```

The following is the supported attack vector, there are 12 methods, the number 11 is written twice, so in reality Mozi's Bot node only supports 11 attack methods.

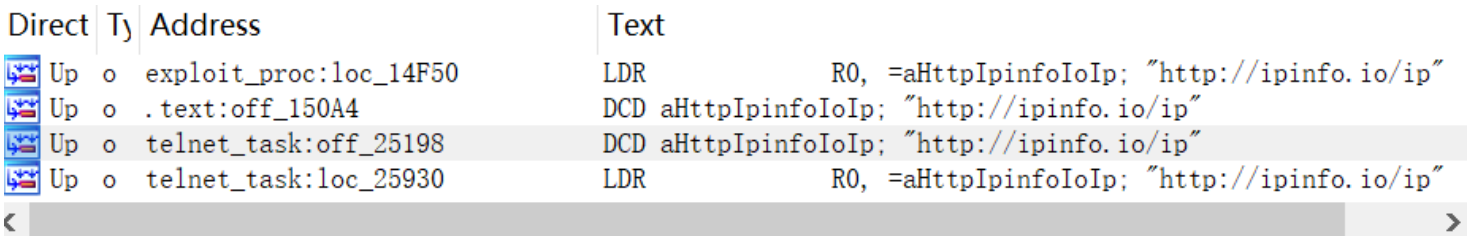
```
add_attack(0, (int)attack_udp_generic);
add_attack(1, (int)attack_udp_vse);
add_attack(2, (int)attack_udp_dns);
add_attack(9, (int)attack_udp_plain);
add_attack(3, (int)attack_tcp_syn);
add_attack(4, (int)attack_tcp_ack);
add_attack(5, (int)attack_tcp_stomp);
add_attack(6, (int)attack_gre_ip);
add_attack(7, (int)attack_gre_eth);
add_attack(10, (int)attack_app_http);
add_attack(11, (int)attack_udp_53port);
add_attack(11, (int)attack_app_cfnull);
return 1;
```

If you are familiar with Mirai, you will smile when you see the following screenshot. Because of the extensive use of Mirai code, this batch of Mozi samples are marked as Mirai by a large number of antivirus producers.

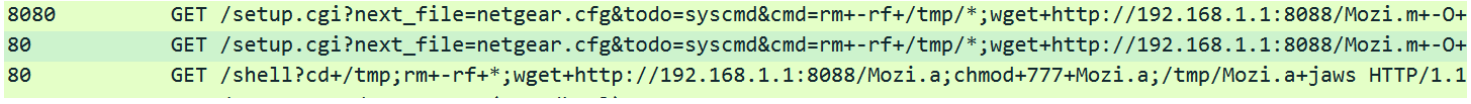


0x2: Extranet address acquisition

`http://ipinfo.io/ip` is called to get the internet address during the telnet &exploit procdures.



After adding this feature there will be no more cases where intranet ip is being used to spread the payload.



0x3: upnp port mapping

When the infected device is accessing the network through NAT, the HTTP sample download service opened by Mozi_bot on the device through the listening port is not directly accessible by the external network. The new version of Mozi implements port mapping on the router through upnp's `AddPortMapping` to ensure normal access to the service.

0x4: Summary

We can see that these updates to Mozi bot are all about efficiency: efficiency of DDoS attacks, efficiency of spreading infections. The abandonment of Gafgyt's attack code in favor of the more efficient Mirai. the separation of control nodes through [cnc] subtasking, refactoring DDoS attack functionality, and achieving separation of control nodes from bot nodes greatly increases Mozi's network resilience. This separation means that the botnet's control function is decoupled from the actual bot functions, allowing Mozi's authors to not only conduct DDos attacks themselves, but also make it possible to lease the network to other groups.

Are Mozi bots still being updated?

The Mozi botnet samples have stopped updating for quite some time, but this does not mean that the threat posed by Mozi has ended. Since the parts of the network that are already spread across the Internet have the ability to continue to be infected, new devices are infected every day. Overall we expect it to oscillate downward in size on a weekly basis but might keep alive for a long time, just like several other botnets that have been terminated by law enforcement agencies in the past.

0-day

Mirai_ptea_Rimasuta变种正在利用RUIJIE路由器在野0DAY漏洞传播

版权 版权声明：本文为Netlab原创，依据 CC BY-SA 4.0 许可证进行授权，转载请附上出处链接及本声明。 概述 2021年7月我们向社区公布了一个通过KGUARD DVR未公开漏洞传播的僵尸网络Mirai_ptea，一开始我们认为这是一个生命短暂的僵尸网络，不久就会消失不见，因此只给了它一个通用的名字。但很显然我们小看了这个家族背后的团伙，事实上该...

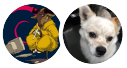
Botnet

Mozi已死，余毒犹存

背景 360NETLAB于2019年12月首次披露了Mozi僵尸网络，到现在已有将近2年时间，在这段时间中，我们见证了它从一个小规模僵尸网络发展为巅峰时占据了极高比例IOT流量巨无霸的过程。 现在Mozi的作者已经被执法机关处置，其中我们也全程提供了技术协助，因此我们认为后续在相当长的一段时间内它都不会继续更新。但我们知道，Mozi采用了P2P网...

See all 114 posts →

Sep 28, 2021 · 14 min read



Aug 27, 2021 · 14 min read