Botnet

Fodcha Is Coming Back, Raising A Wave of Ransom DDoS

Alex.Turing, **Hui Wang**, **YANG XU**

Oct 31, 2022 • 16 min read

Background

On April 13, 2022, 360Netlab first disclosed the `Fodcha` botnet. After our article was published, Fodcha suffered a crackdown from the relevant authorities, and its authors quickly responded by leaving `"Netlab pls leave me alone I surrender"` in an updated sample.No surprise, Fodcha's authors didn't really stop updating after the fraudulent surrender, and soon a new version was released.

In the new version, the authors of Fodcha redesigned the communication protocol and started to use `xxtea` and `chacha20` algorithms to encrypt sensitive resources and network communications to avoid detection at the file & traffic level; at the same time, a dual-C2 scheme with `OpenNIC domain` as the primary C2 and `ICANN domain` as the backup C2 was adopted.

Relying on the strong N-day vulnerability integration capabilities, the comeback of Focha is just as strong as the previous ones. In our data view, in terms of scale, Fodcha has once again developed into a massive botnet with more than `60K` daily active bots and `40+ C2 IPs`, we also observed it can easily launch more than 1Tbps DDos traffic; in terms of attacks, Fodcha has an average of `100+` daily attack targets and more than `20,000` cumulative attacks, on October 11, Fodcha hit its record and attacked `1,396` unique targets in that single day.

While Fodcha was busy attacking various targets, it has not forgot to mess with us, we saw it using `N3t1@bG@Y` in one of it scan payload.

Timeline

Backed by our `BotMon` systems, we have kept good track of Fodcha's sample evolution and DDoS attack instructions, and below are the sample evolution and some important DDoS attack events we have seen. (Note: The Fodcha sample itself does not have a specific flag to indicate its version, this is the version number we use internally for tracking purposes)

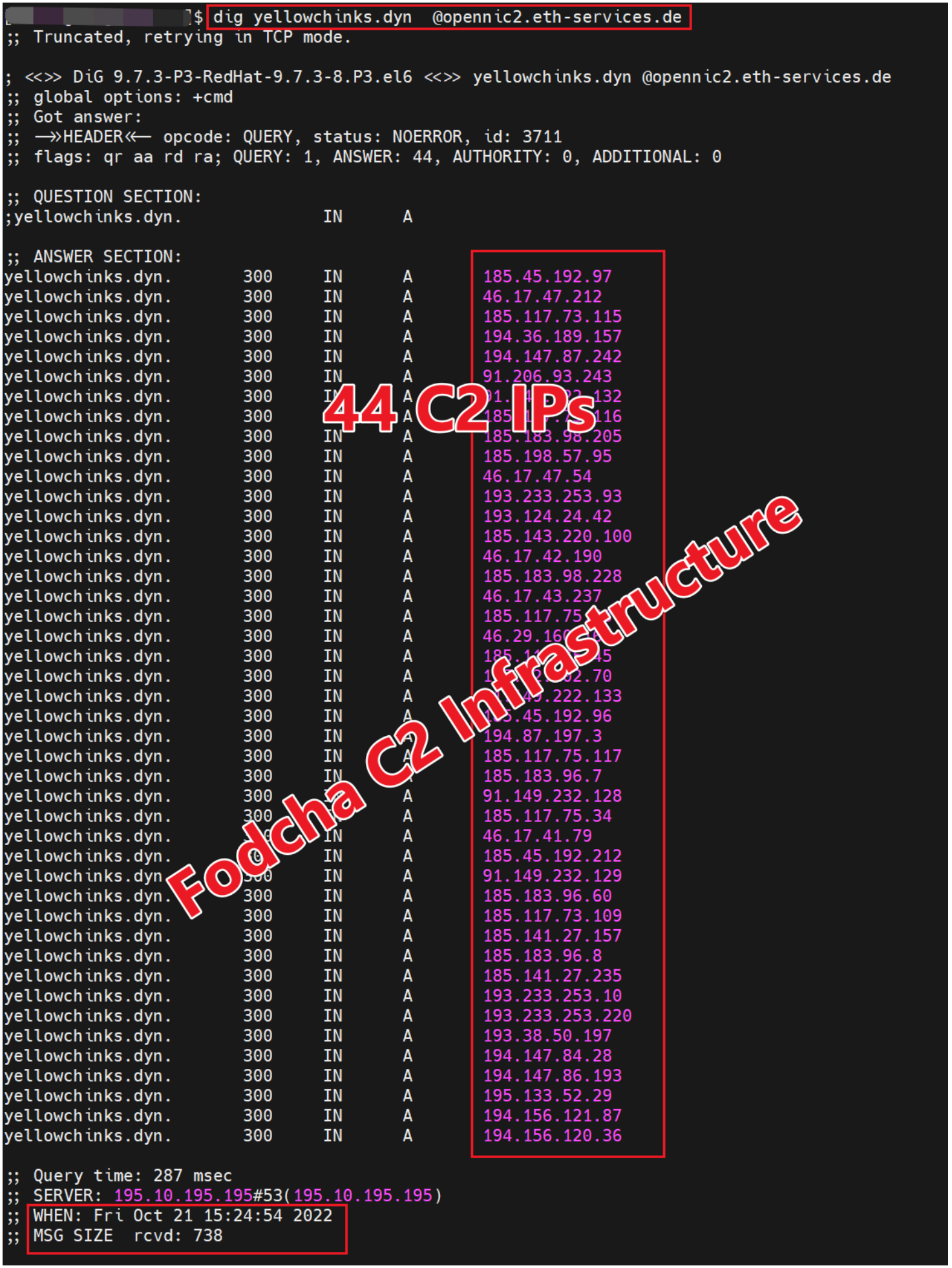
- On January 12, 2022, the first Fodcha botnet sample was captured.
- April 13, 2022, Disclosure of the Fodcha botnet, containing versions V1, V2.
- April 19, 2022, captured version V2.x, using `OpenNIC's TLDs` style C2
- April 24, 2022, version V3, using xxtea algorithm to encrypt configuration information, adding `ICANN's TLDs` style C2, adding `anti-sandbox` & `anti-debugging` mechanism.
- June 5, 2022, version V4, using structured configuration information, `anti-sandboxing` & `anti-debugging` mechanism were removed.
- July 7, 2022, version V4.x with an additional set of `ICANN C2`.
- On September 21, 2022, a well-known cloud service provider was attacked with traffic exceeding `1Tbps`.

Botnet Size

In April, we confirmed that the number of Fodcha's global daily live bots was about `60,000` (refer to our other article). We don't have accurate number of the current

size, but suspect that the number of current active bots has not dropped, maybe more than `60,000` now.

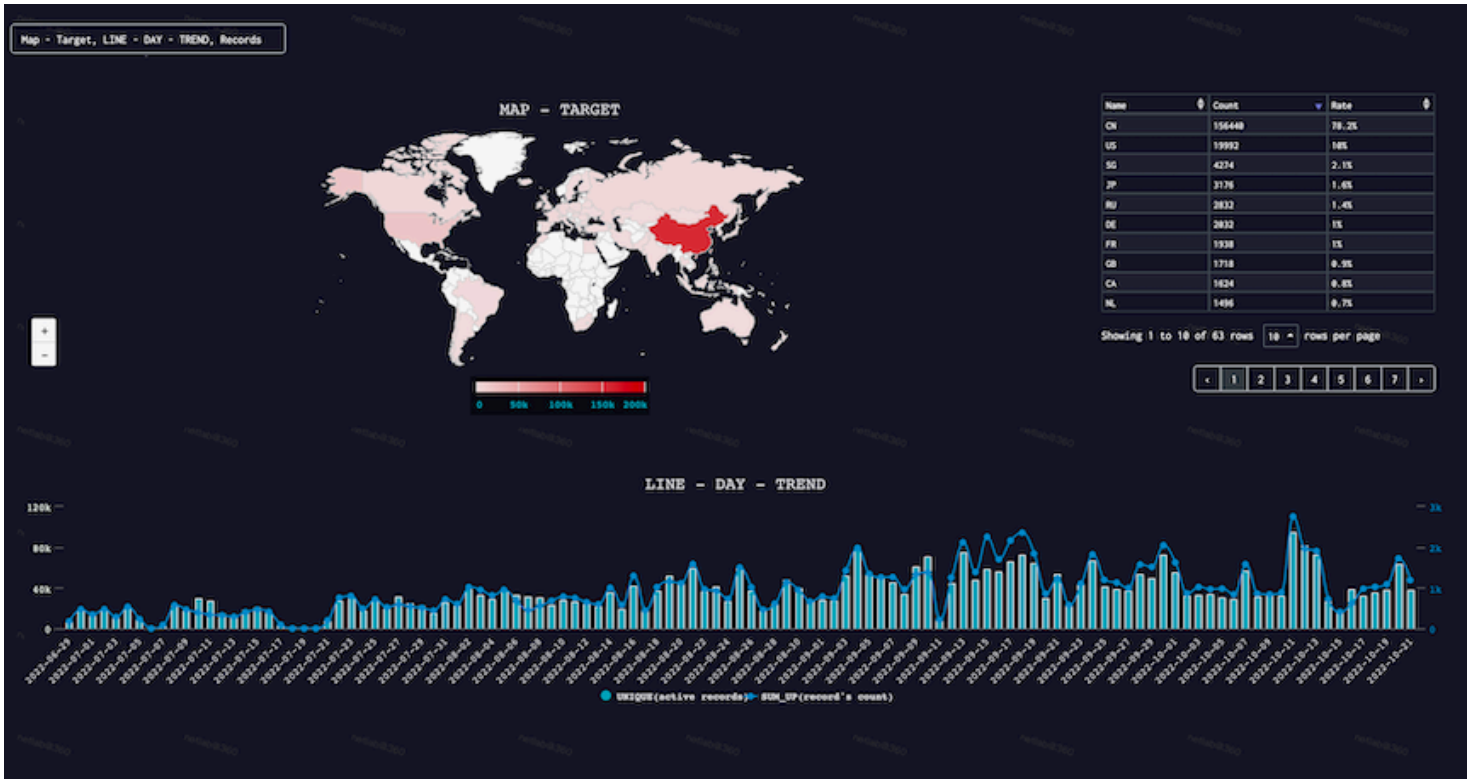
There is a positive relationship between the size of a botnet and the number of C2 IPs, and the most parsimonious view is that "the larger the botnet, the more C2 infrastructure it requires. In April, there were `10 c2s` to control the `60,000` bots; After that, we observed that the IPs corresponding to its C2 domains continued to increase. Using a simple dig command to query the latest C2 domain name `yellowchinks.dyn`, we can see it resolves to `44 IPs`.



One likely reason for this is that their botnet is so large that they need to invest more IP resources in order to have a reasonable ratio between Bots and C2s to achieve load balancing.

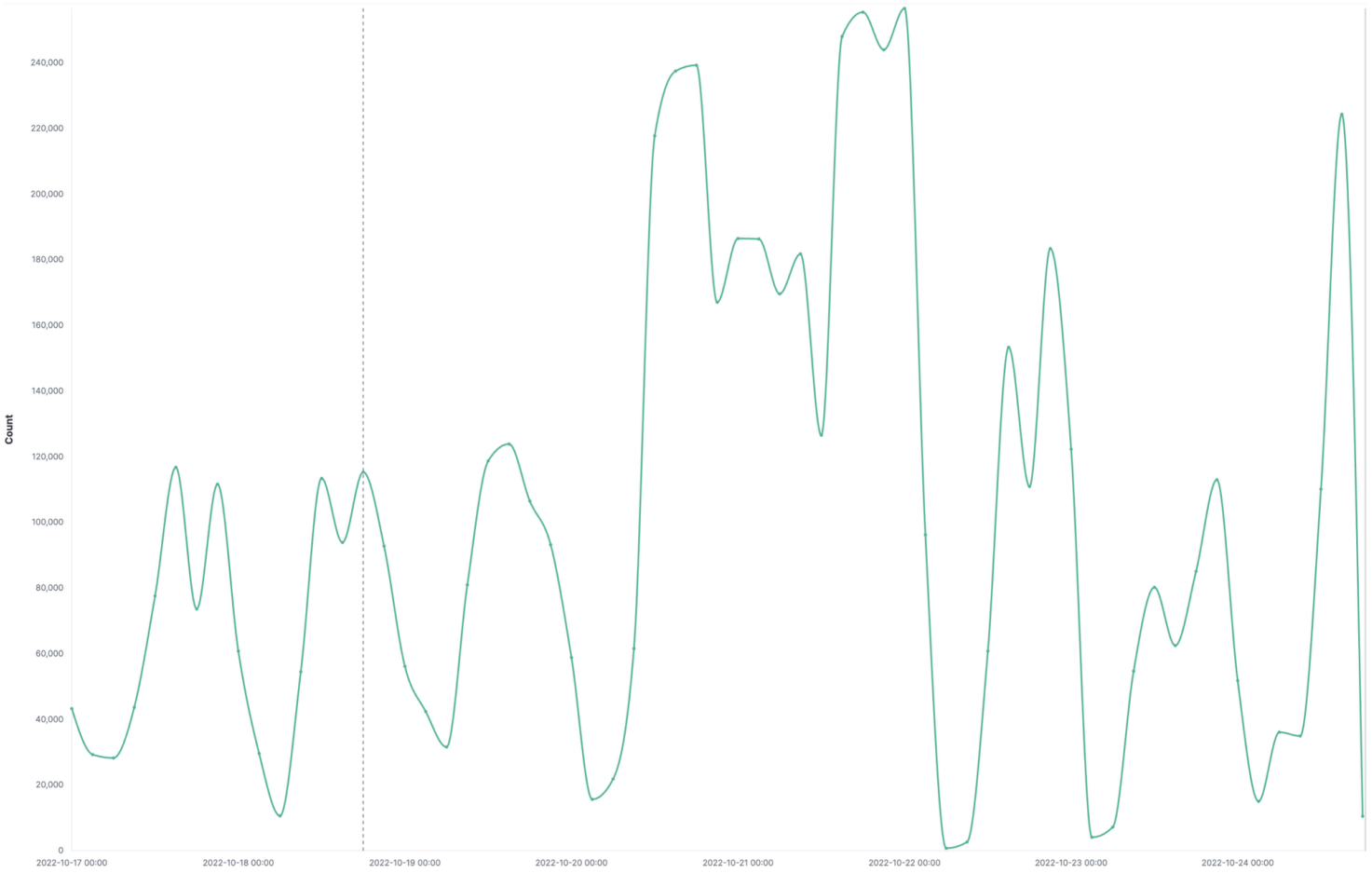
DDoS Statistics

More C2 IPs cost more money, and it seems that the business is good as it has been very active launching ddos attacks.We have excerpted the data from June 29, 2022 to the present, and the attack trends and target area distribution are as follows.



We can see that the ddos attacks has been non-stop, and China and US have the most targets.

The time distribution of the attack instructions within 7 days is shown below, which shows that Fodcha launched DDoS attacks throughout 7 * 24 hours, without any obvious working time zone.



Sample Analysis

We have divided the captured samples into four major versions, of which V1and `V2` have been analyzed in the previous blog, here we select the latest `V4` series samples as the main object of analysis, their basic information is shown below.

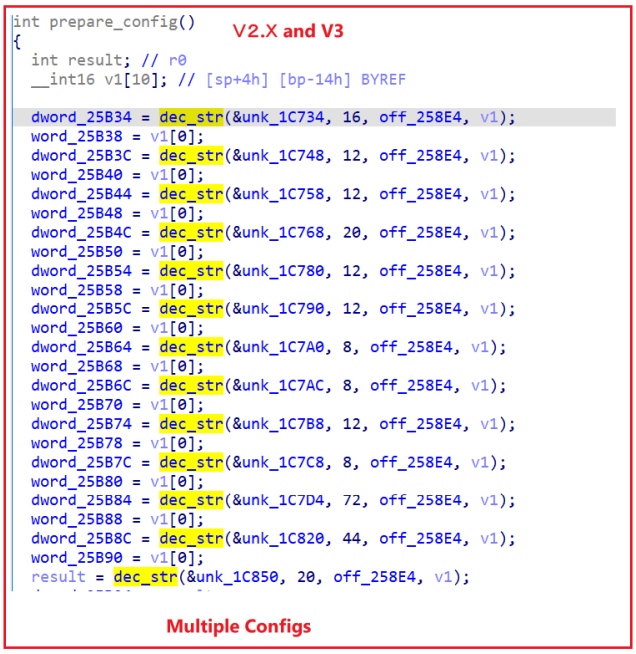
```
MD5: ea7945724837f019507fd613ba3e1da9
ELF 32-bit LSB executable, ARM, version 1, dynamically linked (uses shared libs), st
LIB: uclibc
PACKER: None
version: V4

MD5: 899047ddf6f62f07150837aef0c1ebfb
ELF 32-bit LSB executable, ARM, version 1 (SYSV), statically linked, stripped
Lib: uclibc
Packer: None
Version: V4.X
```

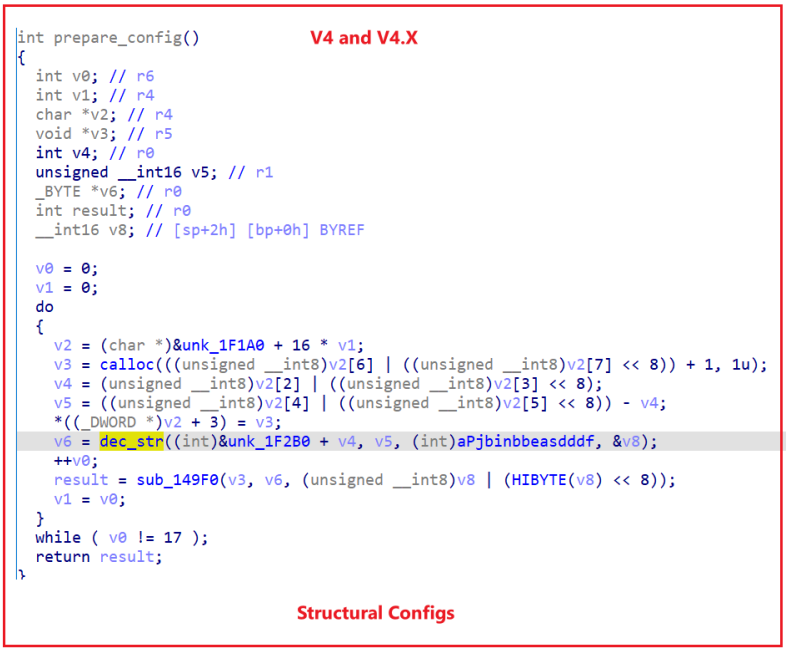
When Fodcha's Bot executes, it will first check `the operating parameters`, `network connectivity`, `whether the "LD_PRELOAD" environment variable is set`, and `whether it is debugged`. These checks can be seen as a simple countermeasure to the typical emulator & sandbox.

When the requirements are met, it first decrypts the configuration information and print “snow slide” on the Console, then there are some common host behaviors, such as single instance, process name masquerading, manipulating watchdog, terminating specific port processes, reporting specific process information, etc. The following will focus on the decryption of configuration information, network communication, DDoS attacks and other aspects of Fodcha.

Decrypting configuration information (Config)

Fodcha uses a side-by-side Config organization in `V2.X` and `V3`, and a structured Config organization in `V4` and `V4.X`. The following figure clearly shows the difference.





Although the organization of Config is different, their encryption methods are the same. As we can see by the constants referenced in the code snippet below, they use the xxtea algorithm with the key `PJbiNbbeasddDfsc`.

```
if ( (unsigned __int16)v4 != 1 )
{
  v21 = 0x9E3779B9 * (0x34u / (unsigned __int16)v4 + 6);
  if ( v21 )
  {
    v22 = &v13[v4];
    do
    {
      v23 = v39;
      v24 = (v21 >> 2) & 3;
      v25 = v22 - 2;
      do
      {
        v20 = v25[1]
            - ((((4 * v20) ^ (*v25 >> 5)) + ((16 * *v25) ^ (v20 >> 3))) ^ ((v20 ^ v21)
                                                                          + (*v25 ^ v17[v24 ^ v23-- & 3])));
        v25[1] = v20;
        --v25;
      }
      while ( v23 );
      v26 = (((4 * v20) ^ (v13[v39] >> 5)) + ((16 * v13[v39]) ^ (v20 >> 3))) ^ ((v20 ^ v21) + (v13[v39] ^ v17[v24]));
      v21 += 0x61C88647;
      v20 = *v13 - v26;
      *v13 = v20;
    }
    while ( v21 );
  }
}
```

After inversion, we wrote the following `IDAPYTHON` script to decrypt the configuration information.

```
# md5: ea7945724837f019507fd613ba3e1da9
# requirement: pip install xxtea-py
# test: ida7.6_python3

import ida_bytes
import xxtea

BufBase=0x1F2B0
ConfBase=0x0001F1A0
key=b"PJbiNbbeasddDfsc"
for i in range(17):
    offset=ida_bytes.get_word(i*16+ConfBase+2)
    leng=ida_bytes.get_word(i*16+ConfBase+4)-offset
    buf=ida_bytes.get_bytes(BufBase+offset,leng)
    print("index:%d, %s" %(i,xxtea.decrypt(buf,key)))
```

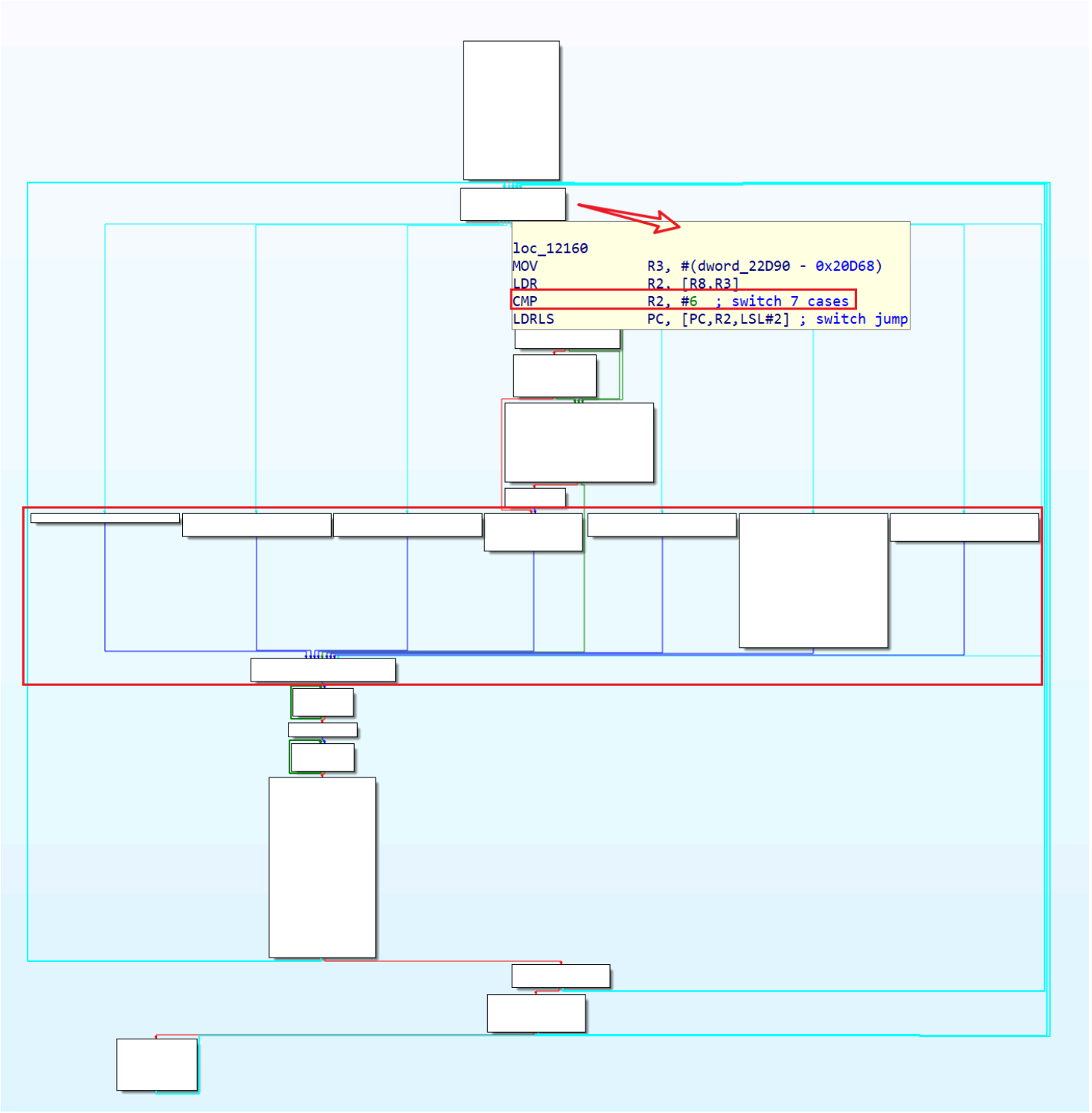
The decrypted Config information is shown in the following table. You can see that index 11 still retains the aforementioned "surrender" egg, index 12 is worth mentioning, as it is the reporter server address to which Fodcha reports some process-specific information.

INDEX	VALUE
0	snow slide

INDEX	VALUE
1	/proc/
2	/stat
3	/proc/self/exe
4	/cmdline
5	/maps
6	/exe
7	/lib
8	/usr/lib
9	.ri
10	GET /geoip/?res=10&r HTTP/1.1\r\nHost: 1.1.1.1\r\nConnection: Close\r\n\r\n
11	Netlab pls leave me alone I surrender
12	kvsolutions.ru
13	api.opennicproject.org
14	watchdog
15	/dev/
16	TSource Engine Query

Network communication

Fodcha's network communication has a very fixed feature at the code level: a perpetual While loop, with switch-case processing at each stage, so that the CFG graphs generated by each version of Fodcha's network protocol processing functions are highly similar in IDA.

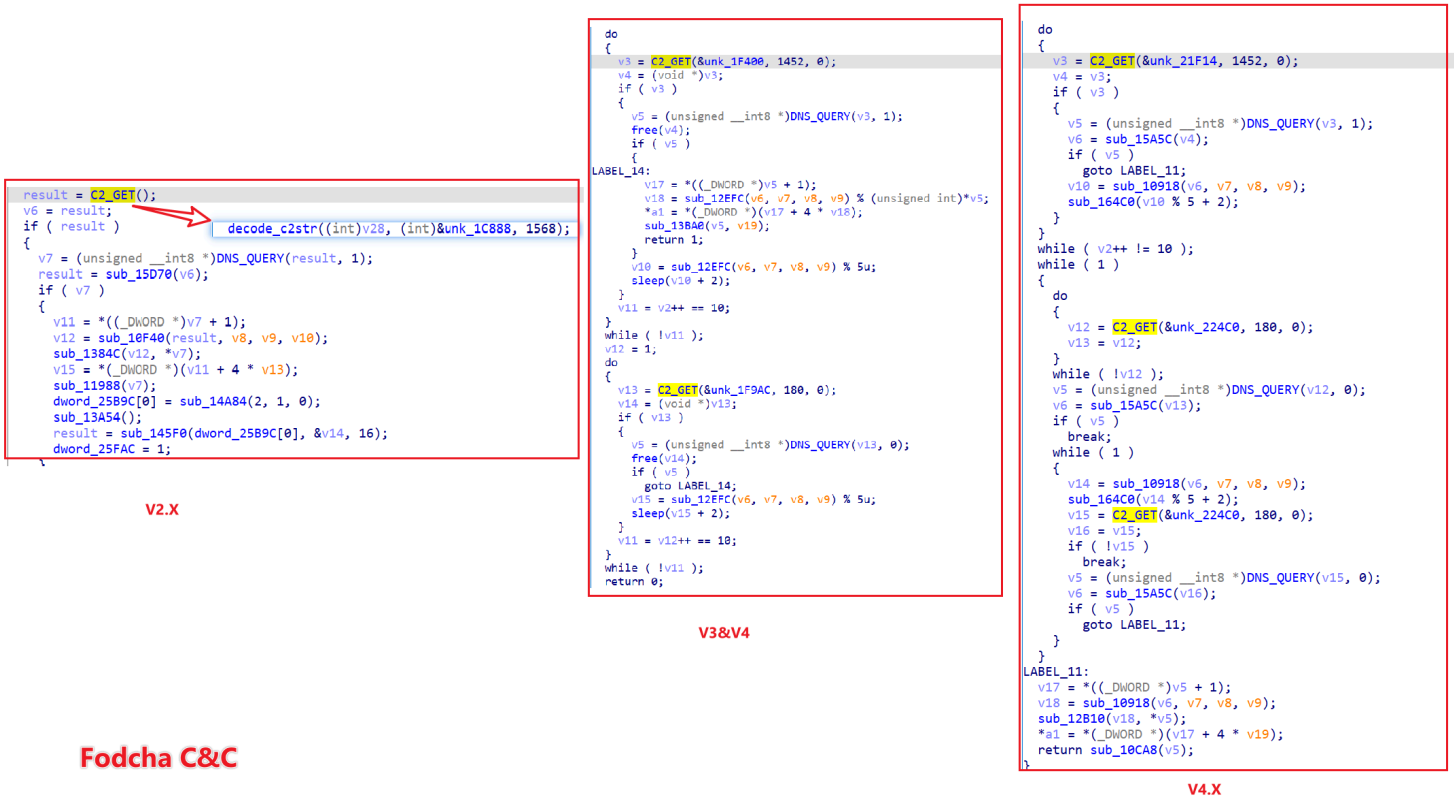


In summary, Fodcha's network communication goes through the following four steps.

1. Decryption C2
2. DNS query
3. Establishing communication
4. Execute the command

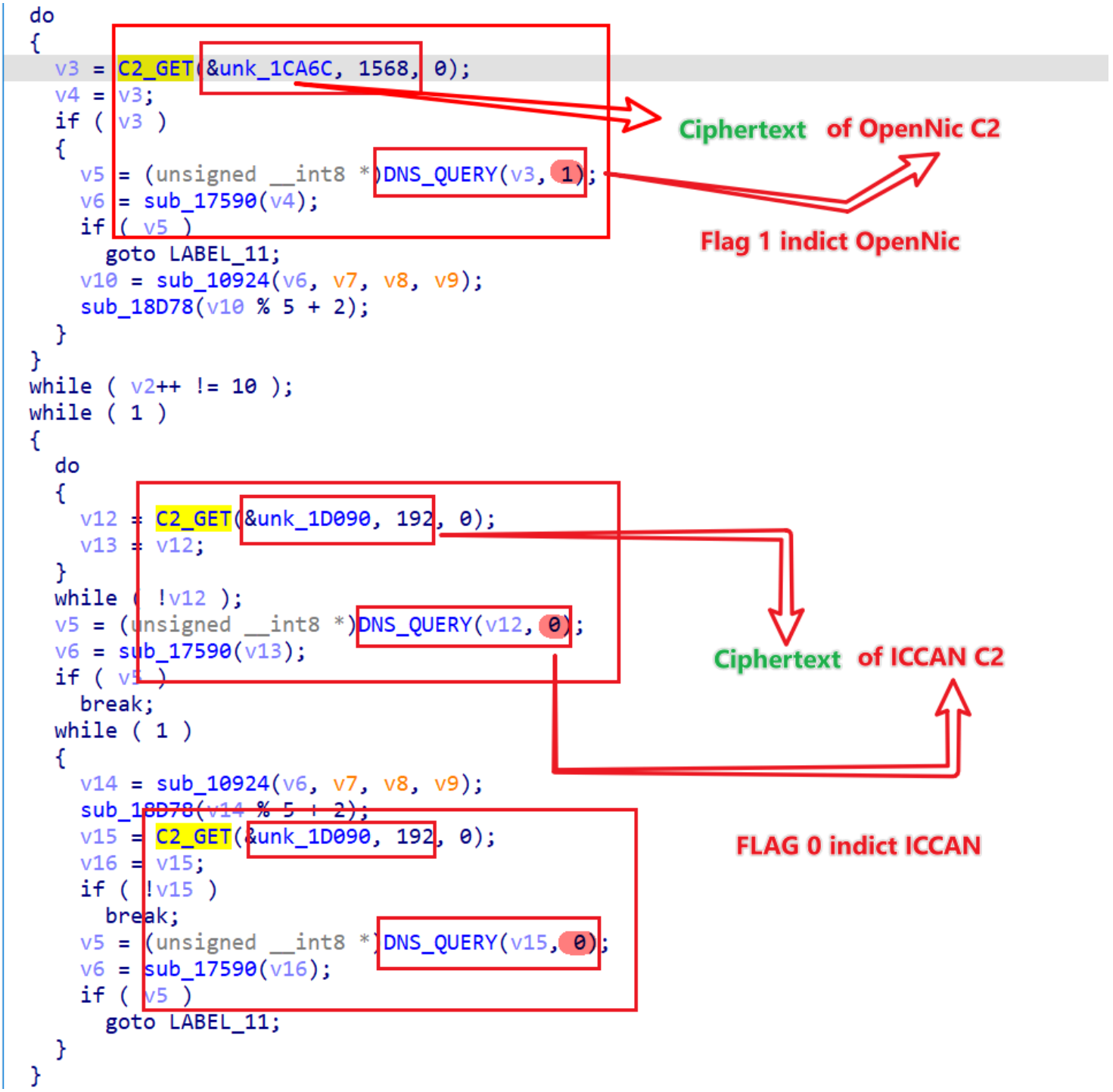
0x1: Decrypting C2

Different versions of Fodcha support different types of C2. V2.X has only 1 group of OpenNIC C2; V3 & V4 have 1 group of OpenNIC C2 and 1 group of ICANN C2; and V4.X has the most, 1 group of OpenNIC C2 and 2 groups of ICANN C2, the following diagram shows the difference very clearly.



Although the types & numbers of C2 are different, their processing logic is almost the same as shown in the figure below.

Firstly, a C2 domain name is obtained through the C2_GET function, and then the corresponding IP of C2 is obtained through the DNS_QUERY function, where the first parameter of C2_GET is the C2 ciphertext data, and the second parameter is the length, while the second parameter of DNS_QUERY implies the type of C2.



A valid C2 domain name can be obtained through C2_GET, and its internal implementation can be divided into 2 steps.

- First, the C2 ciphertext data must be decrypted.
- Then they are constructed into a legitimate domain name.

Decrypting C2 ciphertext data

The C2 ciphertext data uses the same encryption method as the configuration information, i.e. xxtea, and the key is also

`PJbiNbbeasddDfsc`. The OpenNic C2 data can be decrypted by the following simple IDAPYTHON script.

```
#md5: 899047DDF6F62F07150837AEF0C1EBFB
import xxtea
import ida_bytes
import hexdump
key=b"PJbiNbbeasddDfsc"
buf=ida_bytes.get_bytes(0x0001CA6C,1568)  # Ciphertext of OpenNic C2
plaintext=xxtea.decrypt(buf,key)
print(plaintext)
```

The decrypted C2 data is shown below, you can see that the C2 data consists of 2 parts, the front is the domain names, the back is the TLDs, they are separated by the "/" symbol in the red box.


```
"14\nwehateyellow,s3x4fun,techsupporthelpars,s3x4fun,cookiedough,s3x4fun,parasjha,botnetskid,parasjha,cookiedough,parasjha,cookiedough,parasjha,parasjha,parasjha,cookiedough,cookiedough,yellowchinks,cookiedough,s3x4fun,wehateyellow,s3x4fun,s3x4fun,botnetskid,s3x4fun,parasjha,wehateyellow,s3x4fun,botnetskid,wehateyellow,yellowchinks,parasjha,s3x4fun,botnetskid,botnetskid,s3x4fun,cookiedough,s3x4fun,botnetskid,botnetskid,cookiedough,wearelegal,s3x4fun,botnetskid,parasjha,s3x4fun,wehateyellow,parasjha,cookiedough,s3x4fun,funnyyellowpeople,wehateyellow,botnetskid,s3x4fun,parasjha,wehateyellow,botnetskid,chinksdogeaters,wehateyellow,parasjha,s3x4fun,parasjha,blackpeeps,botnetskid,cookiedough,pepperfan,botnetskid,parasjha,parasjha,cookiedough,wehateyellow,botnetskid,botnetskid,parasjha,wehateyellow,parasjha,wehateyellow,cookiedough,parasjha,wehateyellow,chinkchink,cookiedough,cookiedough,botnetskid,cookiedough,wehateyellow,cookiedough,s3x4fun,parasjha,parasjha,wehateyellow,s3x4fun,botnetskid,peepeepoo,botnetskid,botnetskid,botnetskid,wehateyellow,parasjha,wehateyellow,cookiedough,s3x4fun,s3x4fun,botnetskid,respectkkk,s3x4fun,wehateyellow,wehateyellow,parasjha,botnetskid,s3x4fun,wehateyellow,wehateyellow,bladderfull,wehateyellow,botnetskid,parasjha,cookiedough,botnetskid,cookiedough,tsengtsing,cookiedough,botnetskid,wehateyellow,parasjha,obamalover,/pirate,at,geek,libre,ozz,geek,indy,libre,at,geek,libre,oss,libre,geek,libre,libre,libre,at,geek,oss,libre,dyn,oss,geek,dyn,at,ozz,libre,at,geek,geek,geek,oss,dyn,libre,oss,geek,oss,oss,oss,at,oss,"
```


Constructing a domain name

Fodcha has a specific domain name construction method, and the equivalent Python implementation is shown below.

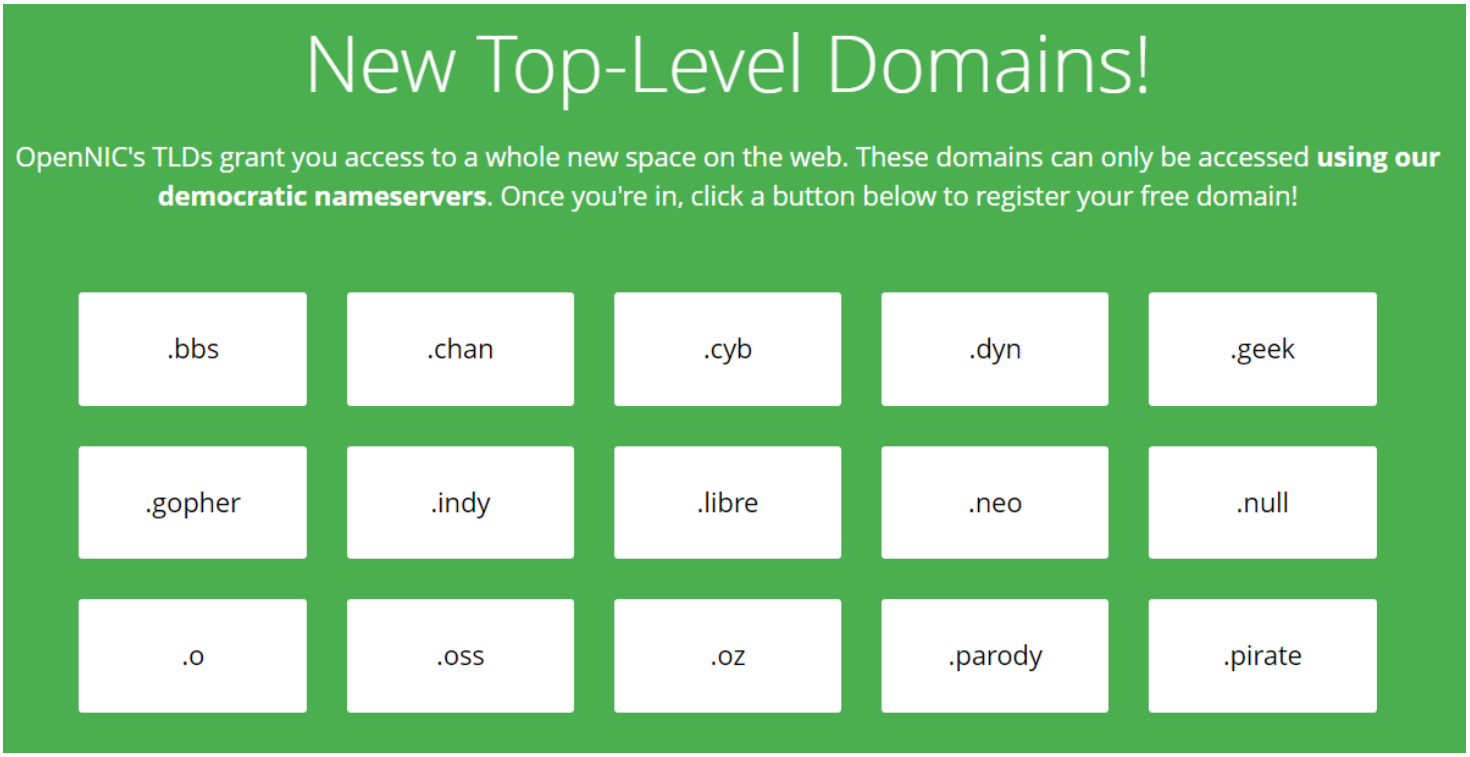
```
# md5: 899047ddf6f62f07150837aef0c1ebfb
# requirement: pip install xxtea-py
# test: ida7.6_python3

import xxtea
import ida_bytes

def getcnt(length):
    cnt=1
    while True:
        cnt +=1
        calc=2
```

```
        for i in range(1,cnt):
            calc+=2+12*i%cnt

        if calc +cnt==length-1:
            return cnt


key=b"PJbiNbbeasddDfsc"
buf=ida_bytes.get_bytes(0x0001CA6C,1568)  # Ciphertext of OpenNic C2
plaintext=xxtea.decrypt(buf,key)

domains,tlds=plaintext.split(b'/')
domainList=domains.split(b',')
tldList=tlds.split(b',')

cnt=getcnt(len(domainList))

print("------------There're %d C2------------" %cnt)
coff=2
for i in range(0,cnt):
    if i ==0:
        c2Prefix=domainList[i+coff]
    else:
        coff+=12*i %cnt+2
        c2Prefix=domainList[i+coff]
    c2Tld=tldList[(cnt-i-1)*3]
    print(c2Prefix + b'.' + c2Tld)
```

Taking the C2 data obtained above as input, the following 14 OpenNIC C2s are finally constructed.

```
techsupporthelpars.oss
yellowchinks.geek
yellowchinks.dyn
wearelegal.geek
funnyyellowpeople.libre
chinksdogeaters.dyn
blackpeeps.dyn
pepperfan.geek
chinkchink.libre
peepeepoo.libre
respectkkk.geek
bladderfull.indy
tsengtsing.libre
obamalover.pirate
```

Readers familiar with the ICANN domain name system may think at first glance that our decryption is wrong, because the ICANN domain name system does not support these TLDs, they would be "unresolvable", but in fact they are the domain names under the OpenNIC system, OpenNIC is independent of the OpenNIC, which supports the TLDs shown in the figure below. The domain names of OpenNIC cannot be resolved by common DNS (such as `8.8.8.8`, `101.198.198.198`) and must use the specified NameServer. Readers can check out OpenNIC's official website for more details.

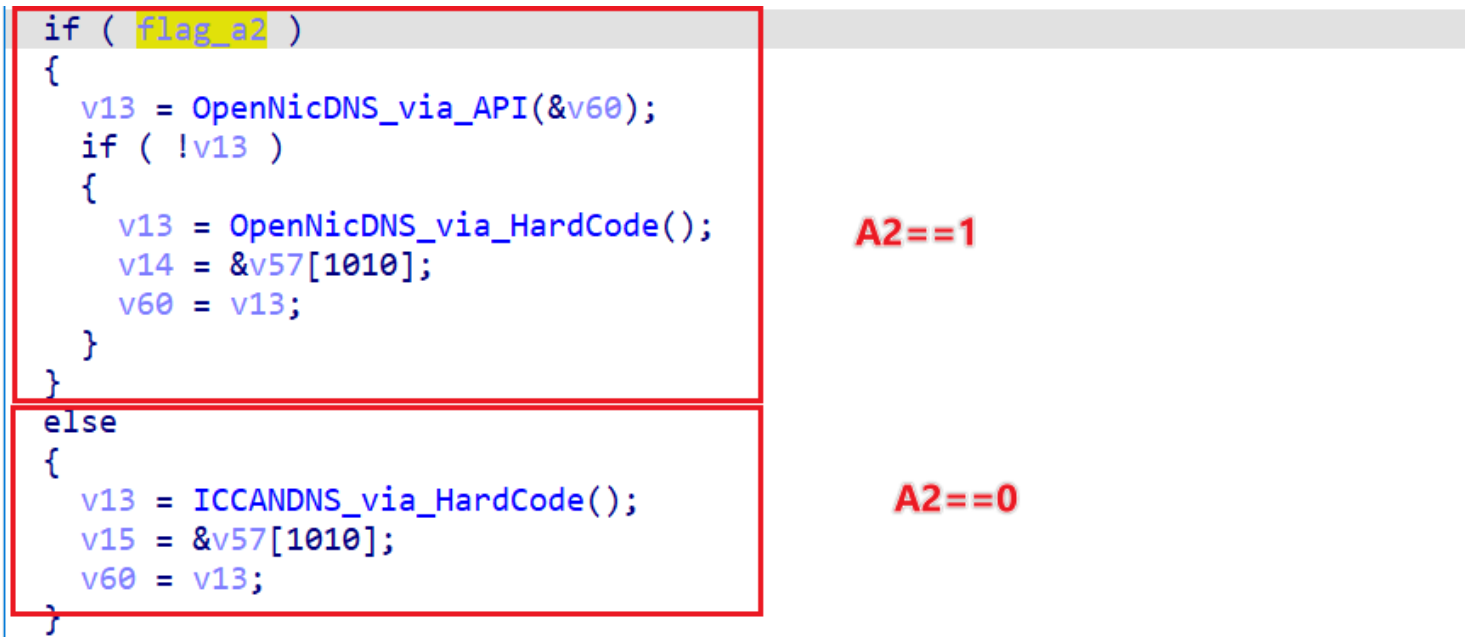


Using the same method, we can get the following 4 ICANN C2s.

```
cookiemonsterboob[.]com
forwardchinks[.]com
doodleching[.]com
milfsfors3x[.]com
```

0X2: DNS lookup

When the C2 domain name is successfully obtained, Bot performs the domain name resolution through the function `DNS_QUERY`, its 2nd parameter is a FLAG, which implies the different processing of OpenNIC/ICANN C2, and the corresponding code snippet is shown below.



It can be seen that there are 2 options for the resolution of OpenNIC C2.

- Option 1: Request from `api.opennicproject.org` through API interface to get nameserver dynamically

```
GET /geoip/?res=10&r HTTP/1.1
Host: 1.1.1.1
Connection: Close

HTTP/1.1 200 OK
Server: nginx
Date: Thu, 21 Jul 2022 18:01:52 GMT
Content-Type: text/plain;charset=UTF-8
Content-Length: 143
Connection: close
Vary: Accept-Encoding
Strict-Transport-Security: max-age=15768000
Allow: GET, HEAD
X-Upstream-Cache-Status: HIT
X-Cache-Key: geoip 180.163.225.13 res=10&r

144.24.181.253
91.217.137.37
94.247.43.254
194.36.144.87
103.1.206.179
130.61.117.123
51.77.149.139
195.10.195.195
94.16.114.254
94.16.114.254
```

Fake

- Option 2: Use the hard-coded nameserver shown in the figure below

```
int __fastcall OpenNicDNS_via_HardCode()
{
  int result; // r0

  switch ( sub_134C8() % 7u )
  {
    case 0u:
      result = 0xC3C30AC3;
      break;
    case 1u:
      result = 0x579024C2;
      break;
    case 2u:
      result = 0x4FDE5097;
      break;
    case 3u:
      result = 0x57C7854F;
      break;
    case 4u:
      result = 0x7B753D82;
      break;
    case 5u:
      result = 0x25E2B35F;
      break;
    default:
      result = 0x8B954D33;
      break;
  }
  return result;
}
```

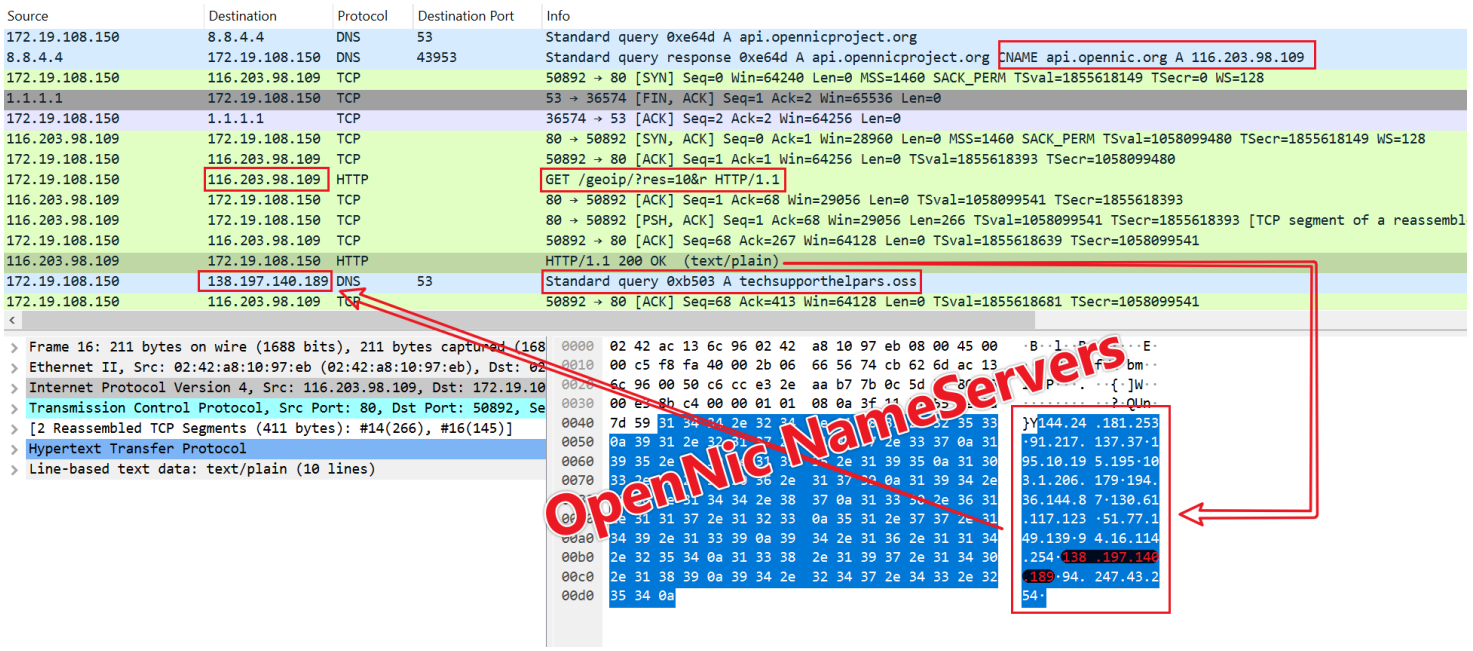
```
195.10.195.195
194.36.144.87
151.80.222.79
79.133.199.87
130.61.117.123
95.179.226.37
51.77.149.139
```

For ICANN C2, there is only one option, i.e., use the hard-coded nameserver shown in the figure below.

```
int __fastcall ICCANDNS_via_HardCode()
{
  int result; // r0

  switch ( sub_134C8() % 7 )
  {
    case 1u:
      result = 0x4040808;
      break;
    case 2u:
      result = 0x1010101;
      break;
    case 3u:
      result = 0x2010101;
      break;
    case 4u:
      result = 0xDEDE43D0;
      break;
    case 5u:
      result = 0xDCDC43D0;
      break;
    case 6u:
      result = 0x9090909;
      break;
    default:
      result = 0x8080808;
      break;
  }
  return result;
}
```

```
8.8.4.4
1.1.1.1
1.1.1.2
208.67.222.222
208.67.220.220
9.9.9.9
8.8.8.8
```

The actual resolution of C2 `techsupporthelpars.oss`, for example, is reflected in the network traffic as follows.



Source	Destination	Protocol	Destination Port	Info
172.19.108.150	8.8.4.4	DNS	53	Standard query 0xe64d A api.opennicproject.org
8.8.4.4	172.19.108.150	DNS	43953	Standard query response 0xe64d A api.opennicproject.org CNAME api.opennic.org A 116.203.98.109
172.19.108.150	116.203.98.109	TCP		50892 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=1855618149 TSecr=0 WS=128
1.1.1.1	172.19.108.150	TCP		53 → 36574 [FIN, ACK] Seq=1 Ack=2 Win=65536 Len=0
172.19.108.150	1.1.1.1	TCP		36574 → 53 [ACK] Seq=2 Ack=2 Win=64256 Len=0
116.203.98.109	172.19.108.150	TCP		80 → 50892 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM TSval=1058099480 TSecr=1855618149 WS=128
172.19.108.150	116.203.98.109	TCP		50892 → 80 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=1855618393 TSecr=1058099480
172.19.108.150	116.203.98.109	HTTP		GET /geoip/?res=10&r HTTP/1.1
116.203.98.109	172.19.108.150	TCP		80 → 50892 [ACK] Seq=1 Ack=68 Win=29056 Len=0 TSval=1058099541 TSecr=1855618393
116.203.98.109	172.19.108.150	TCP		80 → 50892 [PSH, ACK] Seq=1 Ack=68 Win=29056 Len=266 TSval=1058099541 TSecr=1855618393 [TCP segment of a reassembl
172.19.108.150	116.203.98.109	TCP		50892 → 80 [ACK] Seq=68 Ack=267 Win=64128 Len=0 TSval=1855618639 TSecr=1058099541
116.203.98.109	172.19.108.150	HTTP		HTTP/1.1 200 OK (text/plain)
172.19.108.150	138.197.140.189	DNS	53	Standard query 0xb503 A techsupporthelpars.oss
172.19.108.150	116.203.98.109	TCP		50892 → 80 [ACK] Seq=68 Ack=413 Win=64128 Len=0 TSval=1855618681 TSecr=1058099541

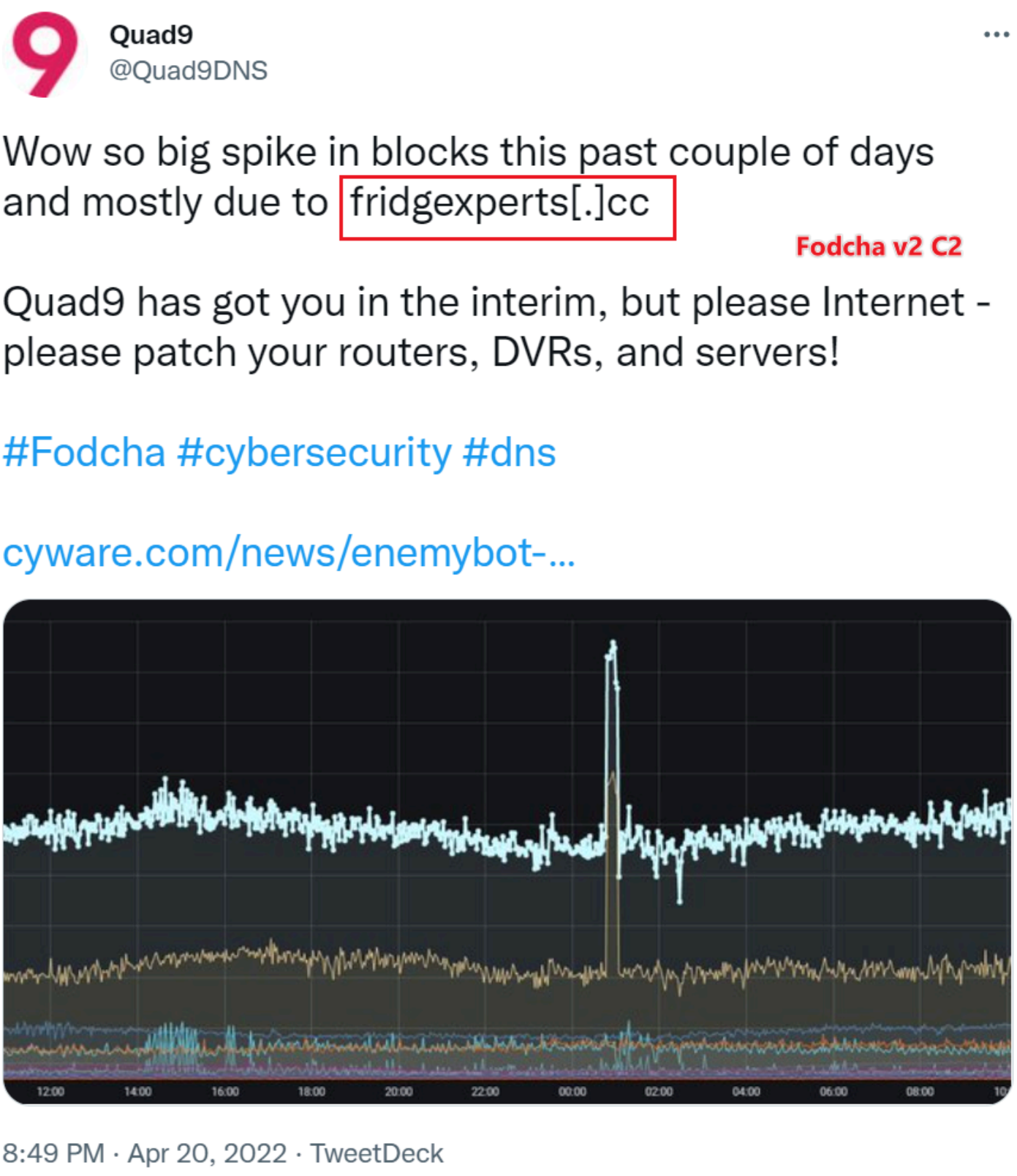
```
Frame 16: 211 bytes on wire (1688 bits), 211 bytes captured (168
Ethernet II, Src: 02:42:a8:10:97:eb (02:42:a8:10:97:eb), Dst: 02
Internet Protocol Version 4, Src: 116.203.98.109, Dst: 172.19.10
Transmission Control Protocol, Src Port: 80, Dst Port: 50892, Se
[2 Reassembled TCP Segments (411 bytes): #14(266), #16(145)]
Hypertext Transfer Protocol
Line-based text data: text/plain (10 lines)
```

```
0000  02 42 ac 13 6c 96 02 42  a8 10 97 eb 08 00 45 00   ·B··l··B ····E·
0010  00 c5 f8 fa 40 00 2b 06  66 56 74 cb 62 6d ac 13   ····@·+· fVt·bm··
0020  6c 96 00 50 c6 cc e3 2e  aa b7 7b 0c 5d 80 18 01   l··P···· ··{·]···
0030  00 e5 8b c4 00 00 01 01  08 0a 3f 11 55 [illegible]  ·····?·U
0040  7d 59 31 34 34 2e 32 34  [illegible] 2e 32 35 33   }Y144.24 .181.253
0050  0a 39 31 2e 32 31 37 2e  [illegible] 2e 33 37 0a 31   ·91.217. 137.37·1
0060  39 35 2e 31 30 [illegible] 2e 31 39 35 0a 31 30   95.10.19 5.195·10
0070  33 2e [illegible] 2e 31 37 39 0a 31 39 34 2e   3.1.206. 179·194.
0080  [illegible] 31 34 34 2e 38 37 0a 31 33 36 2e 36 31   36.144.8 7·130.61
0090  [illegible] 31 31 37 2e 31 32 33 0a 35 31 2e 37 37 2e 31   .117.123 ·51.77.1
00a0  34 39 2e 31 33 39 0a 39 34 2e 31 36 2e 31 31 34   49.139·9 4.16.114
00b0  2e 32 35 34 0a 31 33 38 2e 31 39 37 2e 31 34 30   .254·138 .197.140
00c0  2e 31 38 39 0a 39 34 2e 32 34 37 2e 34 33 2e 32   .189·94. 247.43.2
00d0  35 34 0a   54·
```

Why use OpenNIC / ICANN dual C2?

Fodcha's author has built a redundant OpenNIC / ICANN dual-C2 architecture, why did he do so?

From a C2 infrastructure perspective, after Fodcha was exposed, its C2 was added to various security lists. `Quad9DNS (9.9.9.9)`, for example, had sent a tweet about Fodcha domain traffic spike



After Fodcha was cracked down, its author, when reselecting the C2 infrastructure, looked at the `DNS Neutrality` feature touted by OpenNIC to eliminate the

possibility of C2 being regulated & taken over.

At the same time, OpenNIC based C2 has it own problems, such as the NameServer of OpenNIC may not be accessible in some regions, or there are efficiency or stability problems in domain name resolution. For the sake of robustness, Fodcha authors re-added ICANN C2 as the backup C2 after V3 to form a redundant structure with the main C2.

0x3: Establishing communication

Fodcha Bot establishes a connection to C2 through the following code snippet, which has a total of 22 ports.

```
int __fastcall establist_connect(int a1, int a2, int a3, int a4)
{
  unsigned int v4; // r0
  int result; // r0
  sockaddr_in v6; // [sp+0h] [bp-20h] BYREF

  memset(&v6.sin_port, 0, 14);
  v6.sin_family = 2;
  v4 = sub_10924();
  v6.sin_port = HIBYTE(portlist[v4 % 0x15]) | ((unsigned __int8)portlist[v4 % 0x15] << 8);
  sub_FC54(&v6.sin_addr.s_addr);
  dword_26CB8[0] = socket(2, 1, 0);
  sub_15068((void *)dword_26CB8[0], (void *)4, (void *)0x800);
  result = connect(dword_26CB8[0], &v6, 16);
  dword_28C90 = 1;
  return result;
}
```

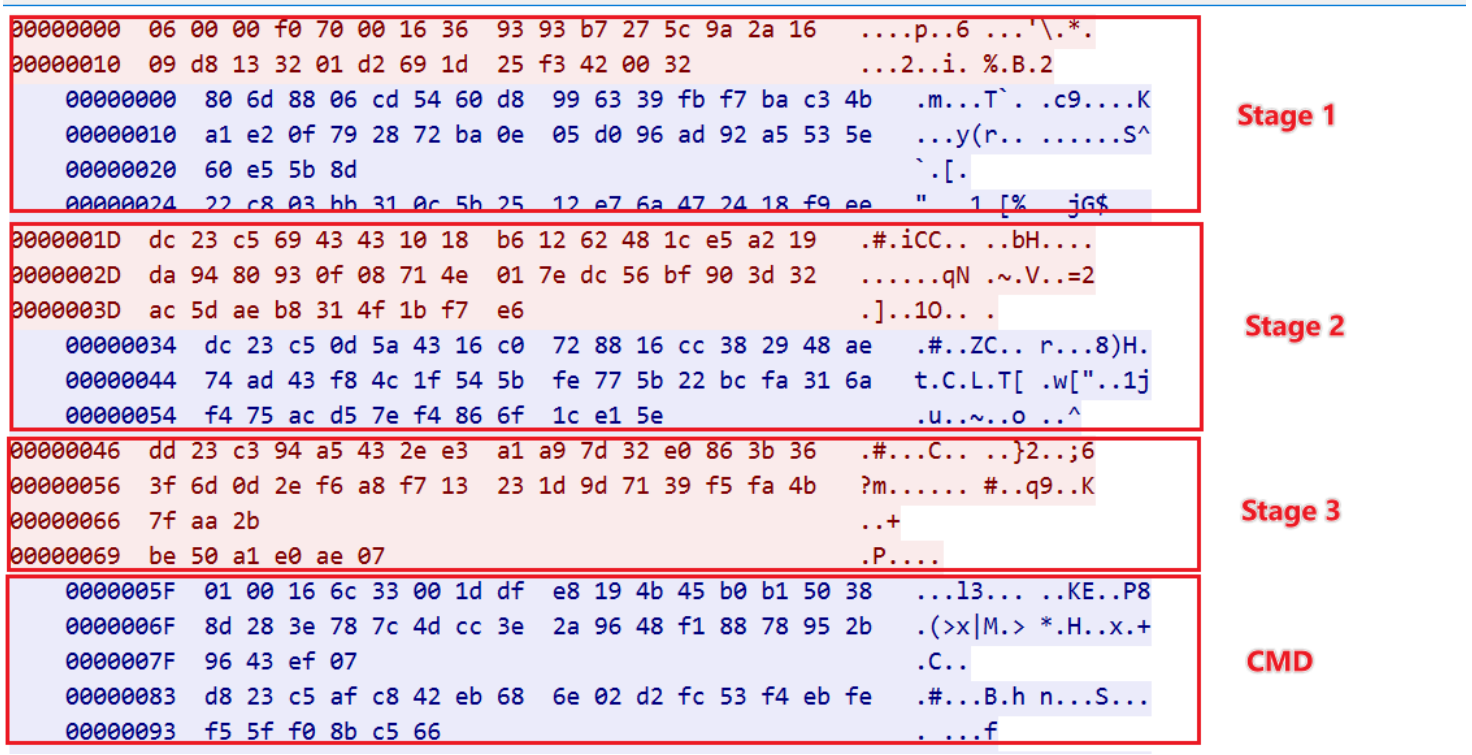
```
; _WORD portlist[22]
portlist        DCW 2348, 12381, 8932, 8241, 38441, 23845, 8745, 6463
                                        ; DATA XREF: sub_FD94+50↑o
                                        ; .text:off_FE58↑o
                DCW 7122, 1114, 6969, 1337, 4200, 3257, 7214, 2474, 4444
                DCW 2222, 3333, 5555, 24811, 0
```

Once the connection with C2 is successfully established, the Bot and C2 must go through 3 phases of interaction before communication is actually established.

- Stage 1: Bot requests the key&nonce of the chacha20 encryption algorithm from C2.
- Stage 2: Bot and C2 use the key&nonce from stage 1 for identity confirmation.

- Stage 3: Bot sends the encrypted upline & group information to C2.

To aid in the analysis, we ran the Bot sample within a restricted environment and used `fsdsaD` as the packet string to generate the network traffic shown in the figure below, and the details of how this traffic was generated are described below.



Stage 1: Bot ---> C2 ,formatted as head(7 bytes) + body(random 20-40 bytes)

Bot actively sends an initialization message with `netstage=6` to C2, this message has the format of head+body, and the meaning of each field is shown below.

```
00000000  06 00 00 f0 70 00 16 36  93 93 b7 27 5c 9a 2a 16   ....p..6 ...'\.*.
00000010  09 d8 13 32 01 d2 69 1d  25 f3 42 00 32            ...2..i. %.B.2
```

head

The length of head is 7 bytes, and the format is shown below.

```
06              ---->netstage,1byte,06 means init
f0 70       ---->tcpip checksum, 2byte,
00 16           ---->length of body, 2 bytes
```

checksum

checksum

The checksum in head uses the tcp/ip checksum, which is calculated for the whole payload, and the original value of the checksum offset is `\x00\x00`, and the python implementation of the checksum is as follows.

```
def checksum(data):
    s = 0
    n = len(data) % 2
    for i in range(0, len(data)-n, 2):
        s+= ord(data[i]) + (ord(data[i+1]) << 8)
    if n:
        s+= ord(data[-1])
    while (s >> 16):
        s = (s & 0xFFFF) + (s >> 16)
        s = ~s & 0xffff
    return s

buf="\x06\x00\x00\x00\x00\x00\x16\x36\x93\x93\xb7\x27\x5c\x9a\x2a\x16\x09\xd8\x13\x32
print(hex(checksum(buf)))

#hex(checksum(buf))
#0x70f0
```

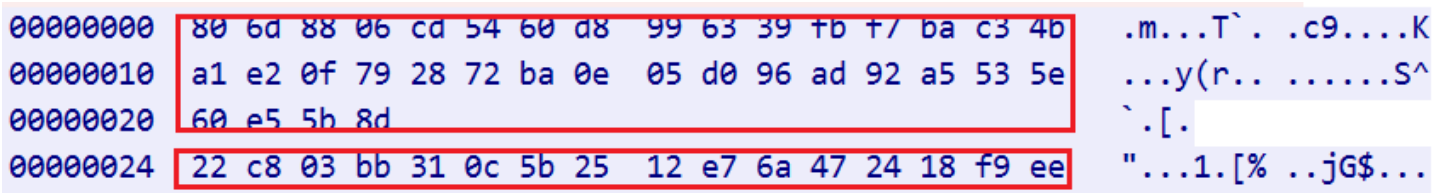
body

body is a randomly generated content, meaningless.

```
00000000  36 93 93 b7 27 5c 9a 2a 16 09 d8 13 32 01 d2 69
00000010  1d 25 f3 42 00 32
```

Stage 1: C2-->Bot, 2 rounds

When C2 receives the message `netstage=6` from Bot, it will send 2 rounds of data to BOT.



- The first round, 36 bytes , the original message is encrypted by xxtea and decrypted as the key of chacha20, with the length of 32bytes

```
import hexdump
import xxtea
key=b"PJbiNbbeasddDfsc"
keyBuf=bytes.fromhex("806d8806cd5460d8996339fbf7bac34ba1e20f792872ba0e05d096
chaKey=xxtea.decrypt(keyBuf,key)
hexdump.hexdump(chaKey)

#chaKey
00000000: E6 7B 1A E3 A4 4B 13 7F  14 15 5E 99 31 F2 5E 3A
00000010: D7 7B AB 0A 4D 5F 00 EF  0C 01 9F 86 94 A4 9D 4B
```

- Second round, 16 bytes , the original message is encrypted by xxtea, decrypted as the nonce of chacha20, length 12bytes

```
import hexdump
import xxtea
key=b"PJbiNbbeasddDfsc"
nonBuf=bytes.fromhex("22c803bb310c5b2512e76a472418f9ee")
chaNonce=xxtea.decrypt(nonBuf,key)
hexdump.hexdump(chaNonce)

#chaNonce
00000000: 98 79 59 57 A8 BA 7E 13  59 9F 59 6F
```

Stage 2: Bot--->C2, chacha20 encryption

Once Bot receives the key and nonce of chacha20, it sends the message `netstage=4` to C2, this time the message is encrypted using chacha20, the key&nonce is obtained from the previous stage, the number of rounds encrypted is 1.

```
0000001D  dc 23 c5 69 43 43 10 18  b6 12 62 48 1c e5 a2 19   .#.iCC.. ..bH....
0000002D  da 94 80 93 0f 08 71 4e  01 7e dc 56 bf 90 3d 32   ......qN .~.V..=2
0000003D  ac 5d ae b8 31 4f 1b f7  e6                       .]..1O.. .
```

We can decrypt the above traffic using the following python code that

```
from Crypto.Cipher import ChaCha20
cha=ChaCha20.new(key=chaKey,nonce=chaNonce)
cha.seek(64)
tmp=bytes.fromhex('dc23c56943431018b61262481ce5a219da9480930f08714e017edc56bf903d32ac
rnd3=cha.decrypt(tmp)
```

The decrypted traffic is shown below, it still has the format of head (7 bytes) + body as described before, where the value of the netstage field of head is 04, which represents the authentication.

```
00000000: 04 00 00 FA 8C 00 22 64  4B 11 90 B6 4F 11 4B E6  ......"dK...O.K.
00000010: 94 CA 70 1A CA 6F 81 38  FB 3F 9B 16 7B 92 5A 06  ..p..o.8.?..{.Z.
00000020: B9 DB 7E DD 93 1B 81 FD  5E                       ..~.....^
```

Stage 2: C2 ---> Bot, chacha20 encryption

After receiving the authentication message from Bot, C2 also sends a message with `netstage=4` to Bot's data, also using chacha20 encryption, and the key,nonce,round number is the same as that used by Bot.

```
00000034  dc 23 c5 0d 5a 43 16 c0  72 88 16 cc 38 29 48 ae   .#..ZC.. r...8)H.
00000044  74 ad 43 f8 4c 1f 54 5b  fe 77 5b 22 bc fa 31 6a   t.C.L.T[ .w["..1j
00000054  f4 75 ac d5 7e f4 86 6f  1c e1 5e                  .u..~..o ..^
```

Using the same code as Bot to decrypt the traffic, we can see that its format is also head+body, and the value of netstage is also 04.

```
00000000: 04 00 00 9E 95 00 24 BC  8F 8B E4 32 6B DD A1 51  ......$....2k..Q
00000010: 3A F3 B3 71 89 78 A4 2D  04 36 1C 62 78 F8 56 5E  :..q.x.-.6.bx.V^
00000020: E1 F3 7C B0 DC A0 1C 65  A4 1B B0                 ..|....e...
```

After Bot and C2 send each other the message `netstage=4`, the chacha20 key&nonce representing stage 1 is recognized by both parties, and the authentication of each other is completed, and Bot enters the next stage to prepare to go online.

Stage 3: Bot--->C2, 2 rounds, chacha encryption

Bot sends netstage=5 message to C2 to indicate that it is ready to go online, and then reports its own grouping information to C2, these 2 rounds of messages also use chacha20 encryption.

- First round

```
00000046  dd 23 c3 94 a5 43 2e e3  a1 a9 7d 32 e0 86 3b 36  .#...C.. ..}2..;6
00000056  3f 6d 0d 2e f6 a8 f7 13  23 1d 9d 71 39 f5 fa 4b  ?m...... #..q9..K
00000066  7f aa 2b                                          ..+
```

- Second round

```
00000069  be 50 a1 e0 ae 07                                 .P....
```

After the above two rounds of data decryption, we can see that the content of the group is exactly the preset `fsdsaD`, which means our analysis is correct, so the Bot is successfully online and starts to wait for the execution of the command sent by C2.

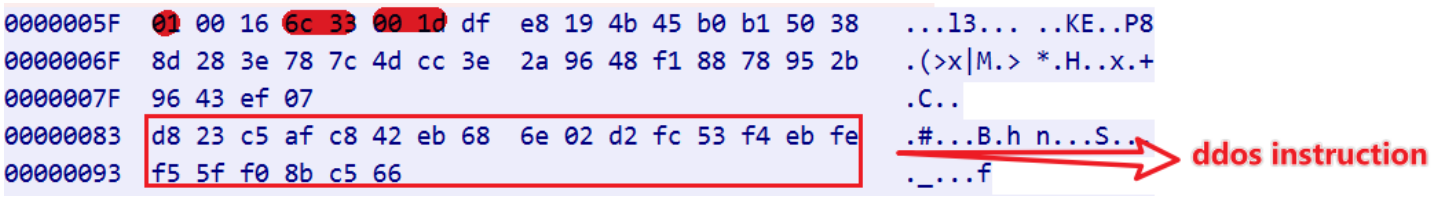
```
00000000: 05 00 06 07 6A 00 1C 9F  5C AA 8F CC B3 72 D2 C9  ....j...\....r..
00000010: 71 33 FD A7 33 CF 07 65  D9 5C DA 31 FD F7 9D 7F  q3..3..e.\.1....
00000020: 6A 2C FB                                          j,.

00000000: 66 73 64 73 61 44                                 fsdsaD
```

0x4: Execute command

Bot successfully online, support the netstage number, as shown in the figure below, the most important is the `netstage = 1` on behalf of the DDoS task, Fodcha reuse a large number of Mirai attack code, a total of `17` kinds of attack methods are supported.

```
  if ( (unsigned __int8)netstage == 2 )
    exit(1);
  if ( (unsigned __int8)netstage == 3 )
    return heartbeat(dword_20D68);
  if ( (unsigned __int8)netstage != 1 )          if netstage == 1
  {
    v5 = dword_20D68[0];
    dword_22D90 = 0;
LABEL_14:
    close(v5);
    v6 = rand() % 10;
    return sleep(v6 + 3);
  }
  return ddos_task(dword_20D68, &unk_20D6C, v4);
}
```

Take the following `DDos_Task` traffic `(netstage=01)` as an example.



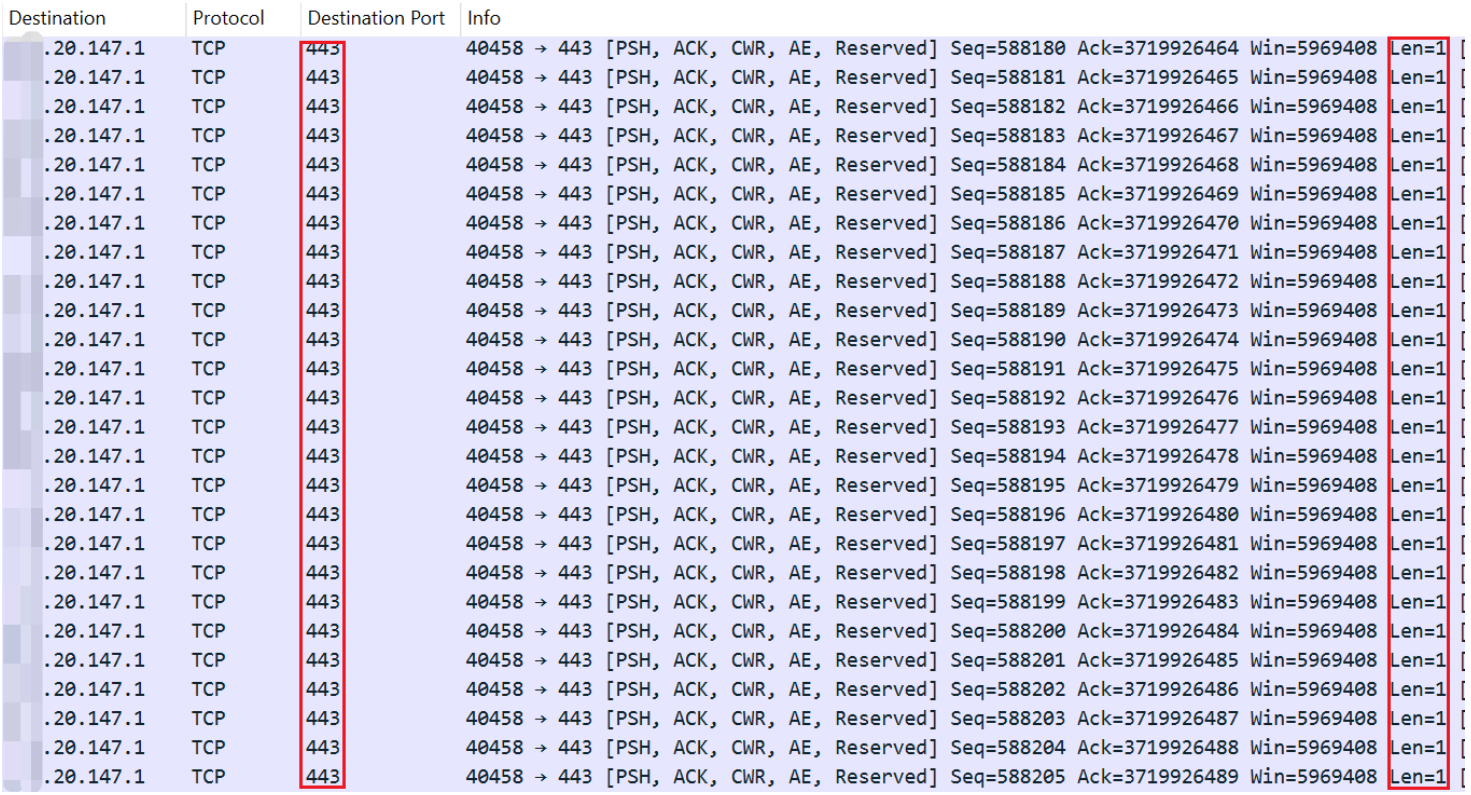
The attack instructions are still encrypted using `chacha20`, and the decrypted instructions are shown below, which might ring a bell for readers who are familiar with `Mirai`.

```
00000000: 00 00 00 3C 07 01 xx 14  93 01 20 02 00 00 02 01
00000010: BB 01 00 02 00 01
```

The format and parsing of the above attack instructions are shown in the following table.

OFFSET	LEN (BYTES)	VALUE	MEANING
0x00	4	00 00 00 3c	Duration
0x04	1	07	Attack Vector，07
0x05	1	1	Attack Target Cnt
0x06	4	xx 14 93 01	Attack Target，xx.20.147.1
0x0a	1	20	Netmask
0x0b	1	02	Option Cnt
0x0c	5	00 00 02 01 bb	OptionId 0，len 2, value 0x01bb ---> (port 443)
0x11	5	01 00 02 00 01	OptionId 1, len 2, value 0x0001---> (payload len 1 byte)

When Bot receives the above instruction, it will use the tcp message with a payload of 1 byte to conduct a DDoS attack on the target `xx.20.147.1:443`, which corresponds to the actual packet capture traffic.



Destination	Protocol	Destination Port	Info
.20.147.1	TCP	443	40458 → 443 [PSH, ACK, CWR, AE, Reserved] Seq=588180 Ack=3719926464 Win=5969408 Len=1 [
.20.147.1	TCP	443	40458 → 443 [PSH, ACK, CWR, AE, Reserved] Seq=588181 Ack=3719926465 Win=5969408 Len=1 [
.20.147.1	TCP	443	40458 → 443 [PSH, ACK, CWR, AE, Reserved] Seq=588182 Ack=3719926466 Win=5969408 Len=1 [
.20.147.1	TCP	443	40458 → 443 [PSH, ACK, CWR, AE, Reserved] Seq=588183 Ack=3719926467 Win=5969408 Len=1 [
.20.147.1	TCP	443	40458 → 443 [PSH, ACK, CWR, AE, Reserved] Seq=588184 Ack=3719926468 Win=5969408 Len=1 [
.20.147.1	TCP	443	40458 → 443 [PSH, ACK, CWR, AE, Reserved] Seq=588185 Ack=3719926469 Win=5969408 Len=1 [
.20.147.1	TCP	443	40458 → 443 [PSH, ACK, CWR, AE, Reserved] Seq=588186 Ack=3719926470 Win=5969408 Len=1 [
.20.147.1	TCP	443	40458 → 443 [PSH, ACK, CWR, AE, Reserved] Seq=588187 Ack=3719926471 Win=5969408 Len=1 [
.20.147.1	TCP	443	40458 → 443 [PSH, ACK, CWR, AE, Reserved] Seq=588188 Ack=3719926472 Win=5969408 Len=1 [
.20.147.1	TCP	443	40458 → 443 [PSH, ACK, CWR, AE, Reserved] Seq=588189 Ack=3719926473 Win=5969408 Len=1 [
.20.147.1	TCP	443	40458 → 443 [PSH, ACK, CWR, AE, Reserved] Seq=588190 Ack=3719926474 Win=5969408 Len=1 [
.20.147.1	TCP	443	40458 → 443 [PSH, ACK, CWR, AE, Reserved] Seq=588191 Ack=3719926475 Win=5969408 Len=1 [
.20.147.1	TCP	443	40458 → 443 [PSH, ACK, CWR, AE, Reserved] Seq=588192 Ack=3719926476 Win=5969408 Len=1 [
.20.147.1	TCP	443	40458 → 443 [PSH, ACK, CWR, AE, Reserved] Seq=588193 Ack=3719926477 Win=5969408 Len=1 [
.20.147.1	TCP	443	40458 → 443 [PSH, ACK, CWR, AE, Reserved] Seq=588194 Ack=3719926478 Win=5969408 Len=1 [
.20.147.1	TCP	443	40458 → 443 [PSH, ACK, CWR, AE, Reserved] Seq=588195 Ack=3719926479 Win=5969408 Len=1 [
.20.147.1	TCP	443	40458 → 443 [PSH, ACK, CWR, AE, Reserved] Seq=588196 Ack=3719926480 Win=5969408 Len=1 [
.20.147.1	TCP	443	40458 → 443 [PSH, ACK, CWR, AE, Reserved] Seq=588197 Ack=3719926481 Win=5969408 Len=1 [
.20.147.1	TCP	443	40458 → 443 [PSH, ACK, CWR, AE, Reserved] Seq=588198 Ack=3719926482 Win=5969408 Len=1 [
.20.147.1	TCP	443	40458 → 443 [PSH, ACK, CWR, AE, Reserved] Seq=588199 Ack=3719926483 Win=5969408 Len=1 [
.20.147.1	TCP	443	40458 → 443 [PSH, ACK, CWR, AE, Reserved] Seq=588200 Ack=3719926484 Win=5969408 Len=1 [
.20.147.1	TCP	443	40458 → 443 [PSH, ACK, CWR, AE, Reserved] Seq=588201 Ack=3719926485 Win=5969408 Len=1 [
.20.147.1	TCP	443	40458 → 443 [PSH, ACK, CWR, AE, Reserved] Seq=588202 Ack=3719926486 Win=5969408 Len=1 [
.20.147.1	TCP	443	40458 → 443 [PSH, ACK, CWR, AE, Reserved] Seq=588203 Ack=3719926487 Win=5969408 Len=1 [
.20.147.1	TCP	443	40458 → 443 [PSH, ACK, CWR, AE, Reserved] Seq=588204 Ack=3719926488 Win=5969408 Len=1 [
.20.147.1	TCP	443	40458 → 443 [PSH, ACK, CWR, AE, Reserved] Seq=588205 Ack=3719926489 Win=5969408 Len=1 [

Misc

0x01: Racism

From some of the OpenNIC C2 constructs, it seems that the author of Fodcha is more hostile to people from some regions.

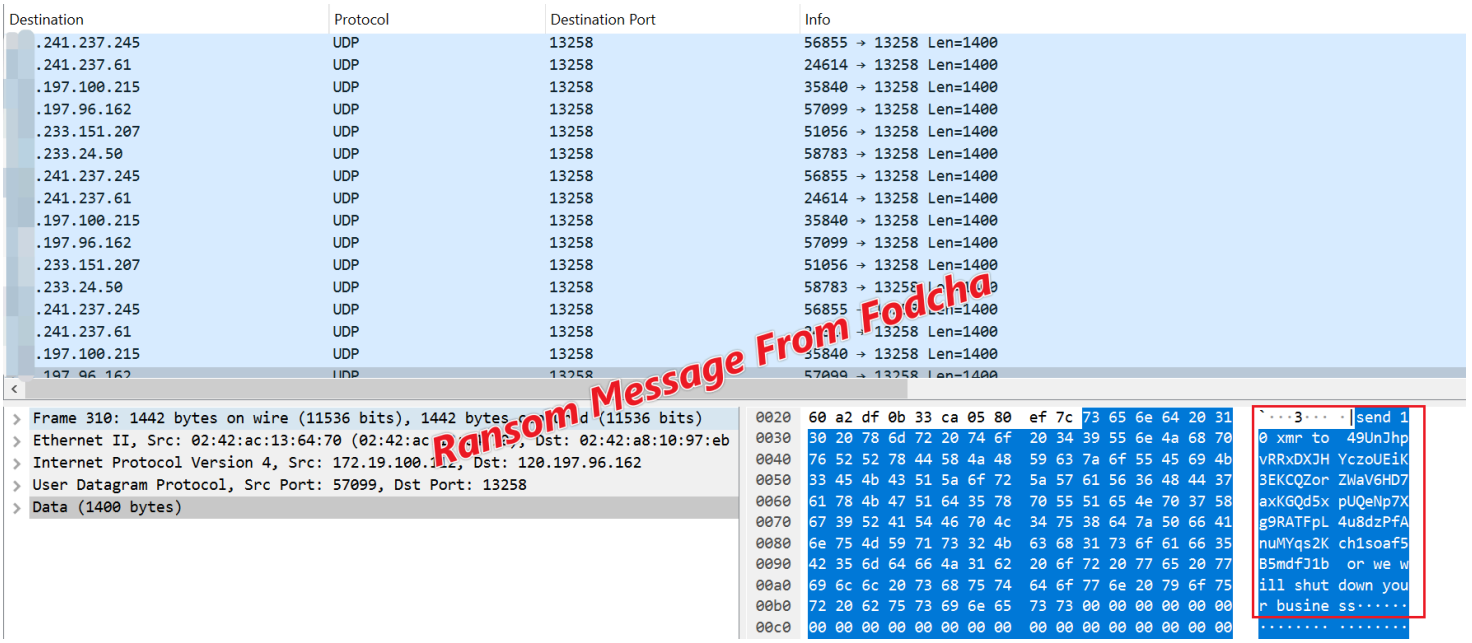
```
yellowchinks.geek
wearelegal.geek
funnyyellowpeople.libre
chinksdogeaters.dyn
blackpeeps.dyn
bladderfull.indy

wehateyellow
```

0x02: Ransom DDoS

Fodcha had the following string attached to the UDP attack command it issued.

```
send 10 xmr to 49UnJhpvRRxDXJHYczoUEiK3EKCQZorZWaV6HD7axKGQd5xpUQeNp7Xg9RATFpL4u8dzP
```

The corresponding attack traffic is shown below, the wallet address appears to be illegal, perhaps the operators behind Fodcha are experimenting with the attack-as-ransom business model, we will see how it evolve.



Contact us

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

IoC

C2

```
v1,v2:
folded[.]in
fridgexperts[.]cc

ICANN C2:
forwardchinks[.]com
doodleching[.]com
cookiemonsterboob[.]com
milfsfors3x[.]com
```

OpenNIC C2:

yellowchinks.geek
yellowchinks.dyn
wearelegal.geek
tsengtsing.libre
techsupporthelpars.oss
respectkkk.geek
pepperfan.geek
peepeepoo.libre
obamalover.pirate
funnyyellowpeople.libre
chinksdogeaters.dyn
chinkchink.libre
bladderfull.indy
blackpeeps.dyn
91.206.93.243
91.149.232.129
91.149.232.128
91.149.222.133
91.149.222.132
67.207.84.82
54.37.243.73
51.89.239.122
51.89.238.199
51.89.176.228
51.89.171.33
51.161.98.214
46.17.47.212
46.17.41.79
45.88.221.143
45.61.139.116
45.41.240.145
45.147.200.168
45.140.169.122
45.135.135.33
3.70.127.241
3.65.206.229
3.122.255.225
3.121.234.237
3.0.58.143
23.183.83.171
207.154.206.0
207.154.199.110
195.211.96.142
195.133.53.157
195.133.53.148
194.87.197.3
194.53.108.94
194.53.108.159
194.195.117.167
194.156.224.102

194.147.87.242
194.147.86.22
193.233.253.93
193.233.253.220
193.203.12.157
193.203.12.156
193.203.12.155
193.203.12.154
193.203.12.151
193.203.12.123
193.124.24.42
192.46.225.170
185.45.192.96
185.45.192.227
185.45.192.212
185.45.192.124
185.45.192.103
185.198.57.95
185.198.57.105
185.183.98.205
185.183.96.7
185.143.221.129
185.143.220.75
185.141.27.238
185.141.27.234
185.117.75.45
185.117.75.34
185.117.75.119
185.117.73.52
185.117.73.147
185.117.73.115
185.117.73.109
18.185.188.32
18.136.209.2
178.62.204.81
176.97.210.176
172.105.59.204
172.105.55.131
172.104.108.53
170.187.187.99
167.114.124.77
165.227.19.36
159.65.158.148
159.223.39.133
157.230.15.82
15.204.18.232
15.204.18.203
15.204.128.25
149.56.42.246
139.99.166.217
139.99.153.49
139.99.142.215
139.162.69.4

```
138.68.10.149
137.74.65.164
13.229.98.186
107.181.160.173
107.181.160.172
```

Reporter

```
kvsolutions[.]ru
icarlyfanss[.]com
```

Samples

```
ea7945724837f019507fd613ba3e1da9
899047ddf6f62f07150837aef0c1ebfb
0f781868d4b9203569357b2dbc46ef10
```

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

Botnet

僵尸网络911 S5的数字遗产

Heads up! Xdr33, A Variant Of CIA's HIVE Attack Kit Emerges

警惕：魔改后的CIA攻击套件Hive进入黑灰产领域

See all 114 posts →

Import 2022-11-30 11:16

P2P 僵尸网络：回顾·现状·持续监测

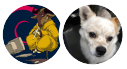
缘起 P2P结构的网络比传统的C/S结构具有更好的可扩展性和健壮性，这些优点很早就为botnet的作者所认识到并被用到他们的僵尸网络中。从时间上看，2007年出现的Storm可以算是这方面的鼻祖，那时botnet这种网络威胁刚为大众所知。Storm之后，陆续又有Karen、ZeroAccess、GameOver、Hijime、mozi等20来种P2P botnet先后出现...


Nov 2, 2022 · 16 min read

Botnet

卷土重来的DDoS狂魔：Fodcha僵尸网络再次露出獠牙

背景 2022年4月13日，360Netlab首次向社区披露了Fodcha僵尸网络，在我们的文章发表之后，Fodcha遭受到相关部门的打击，其作者迅速做出回应，在样本中留下Netlab pls leave me alone I surrender字样向我们投降。本以为Fodcha会就此淡出江湖，没想到这次投降只是一个不讲武德的假动作，Fodcha的作者在诈降之后并没有停下更新的脚...

Oct 27, 2022 · 23 min read