Botnet

卷土重来的DDoS狂魔：Fodcha僵尸网络再次露出獠牙

Alex.Turing, Hui Wang, YANG XU

Oct 27, 2022 • 23 min read

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步，很快就推出了新版本。

在新版本中，Fodcha的作者重新设计了通信协议，并开始使用xxtea和chacha20算法对敏感资源和网络通信进行加密，以躲避文件&流量层面的检测；同时引入了**OpenNIC 域名做为主选C2，ICANN 域名做为后备C2的双C2方案**。这种冗余机制，既能防止C2被接管，又有良好的健壮性，能够维持其主控网络的稳定。

依托于背后团队强大的N-day漏洞整合能力，卷土重来的Focha与之前对比可谓有过之而无不及。在我们的数据视野中，**从规模来看**，Fodcha再次发展成日活Bot节点数超过60K，C2域名绑定40+IP，可以轻松打出超过**1Tbps**流量的大规模僵尸网络；**就活跃程度而言**，Fodcha日均攻击目标100+，累计攻击目标2万多，在10月11日到达了攻击的巅峰，单日“**丧心病狂**”的攻击了**1396**个目标。

在极短的时间内重回巅峰，Fodcha的作者似乎忘了闷声发大财的道理，竟然又开始主动"招惹”我们，在某次扫描的脚本中使用`N3t1@bG@Y`字样的leetspeak，翻译过来就是"`NETLABGAY`"，这么明目张胆的黑Netlab，让我们觉得它多多少少有些“皮痒”了。

鉴于Fodcha的规模&活跃程度带来的巨大危险性，以及非常嚣张的挑衅，我们决定撰写本文向社区分享我们的发现，一起打击Fodcha的嚣张气焰，共同维护网络安全。

时间线

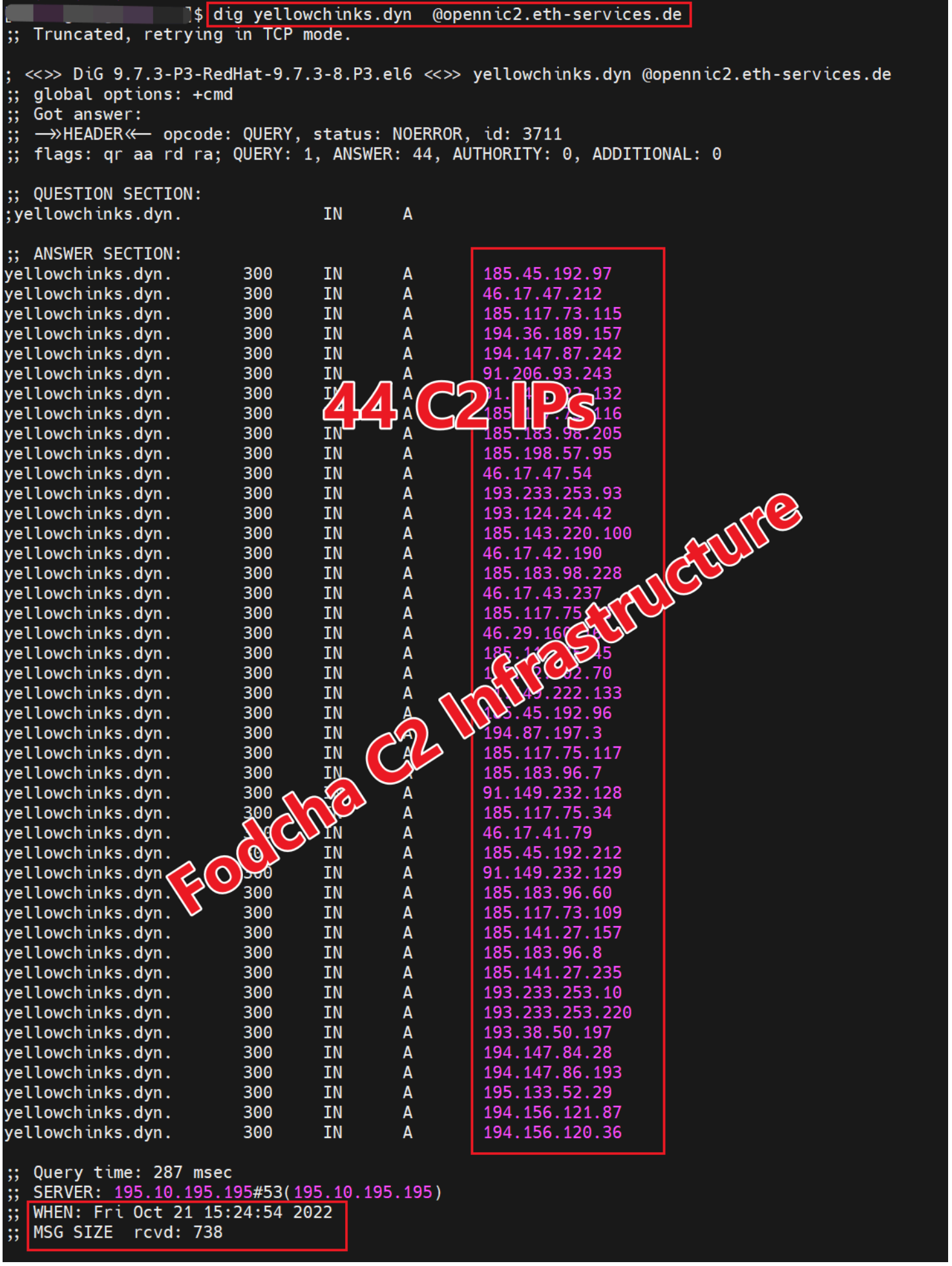
依托于360Netlab强大的BotMon和DDoSMon系统，我们对Fodcha的样本演变和DDoS攻击指令一直保持着良好跟踪，下面是我们看到的样本演变以及一些重要的DDoS攻击事件。（注：Fodcha样本本身没有特定的标志表明其版本，这是我们内部为了跟踪方便而定的版本号）

- 2022年1月12日，首次捕获到Fodcha僵尸网络样本。
- 2022年4月13日，首次向外披露Fodcha僵尸网络，包含版本V1，V2。
- 2022年4月19日，捕获版本V2.x，使用**OpenNIC's TLDs风格的C2**（全文简称OpenNIC C2）。
- 2022年4月24日，捕获版本V3，使用xxtea算法加密配置信息，新增**ICANN's TLDs风格的C2**（全文简称ICANN C2），和OpenNIC C2构成冗余机制；新增反沙箱&反调试机制。
- 2022年6月5日，捕获版本V4，使用结构化的配置信息，去除反沙箱&反调试机制。
- 2022年6月7&8日，监控到Fodcha对**某国的某地的健康码机构**进行了DDoS攻击。
- 2022年7月7日，捕获版本V4.x，额外新增一组ICANN C2。
- 2022年9月X日，在协助**某国的某执法机构**固定某公司语音业务被DDoS攻击的证据链过程中，发现攻击背后有Fodcha的影子。
- 2022年9月21日，**某知名云服务商**就一起流量**超过1Tbps**的攻击事件向我们咨询，经过数据的交叉比对，确定攻击方为Fodcha。

规模推测

国外合作伙伴的数据表明Fodcha 4月份时全球日活Bot的数量为6W(参考我们另一篇文章)，关于Fodcha僵尸网络的目前规模，我们没有确切的数字，但通过对比Fodcha 4月和10月在C2 IP数量上的差异，我们从技术上出发，有个未经验证的猜测：目前Fodcha的日活Bot数量超过6W。

推测过程如下：
僵尸网络的规模与C2 IP的数量存在一个正向关系，最朴素的观点是：“僵尸网络规模越大，所需要的C2基础设施也越多”。在4月份，Fodcha被处置之前，其作者为维持6W的规模，投入了10个C2 IP；随后Fodcha开始了自己的复活之旅，我们观察到一个现象，随着Fodcha的复苏，其C2域名对应的IP在持续增加。时至今日，Fodcha的作者投入了多少C2 IP呢？使用dig命令查询最新的C2域名`yellowchinks.dyn`的绑定IP，可以看到数量是44。



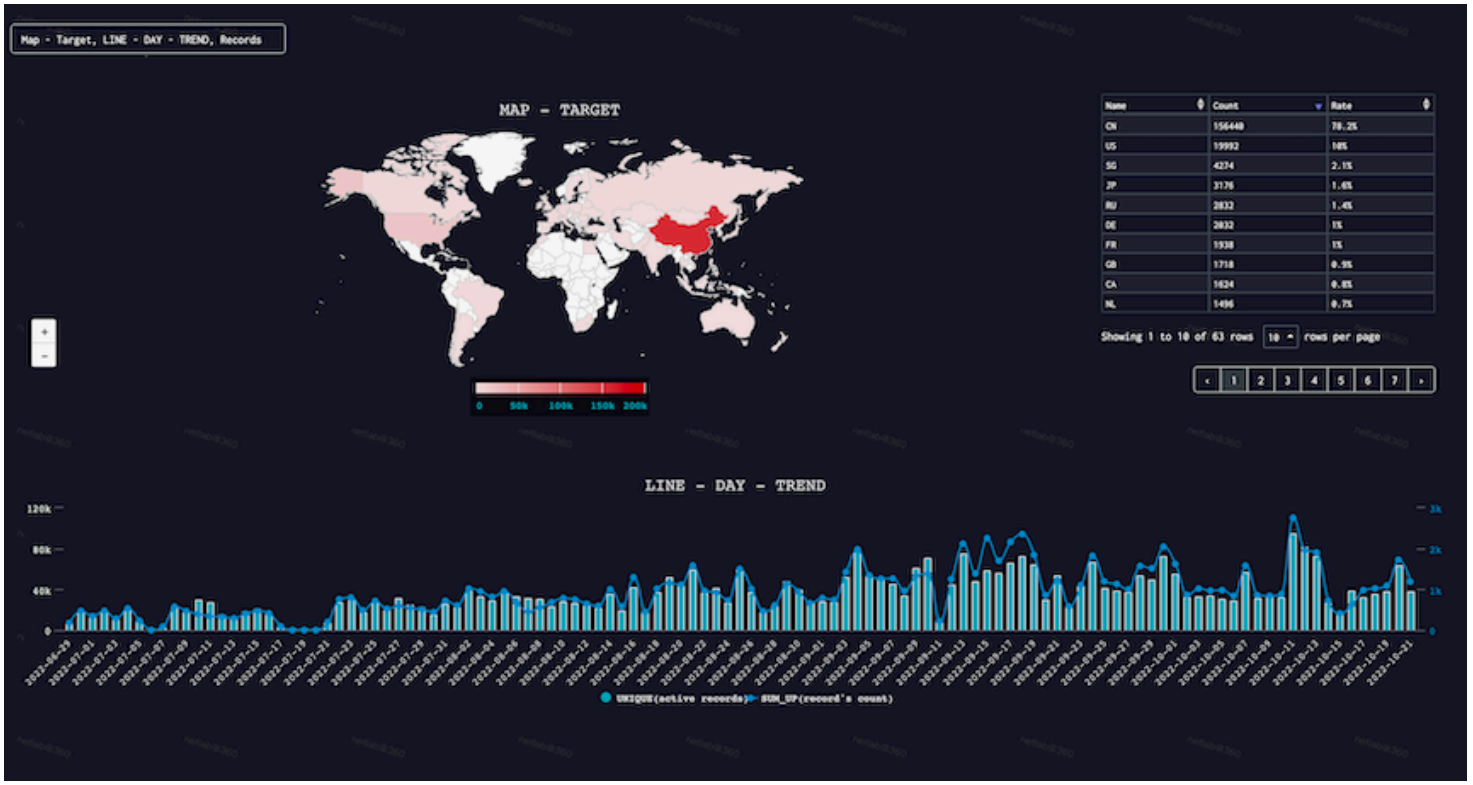
可以说我们见证了Fodcha的C2 IP一步步从几个增长到今天的40+，可能的解释是作者人傻钱多无脑上资源，但结合其迅猛的传播以及历史上曾看到的万级规模，他们

增加C2 IP更可能的原因是因为其僵尸网络规模太大，需要投入更多的IP资源，以使Bot与C2之间在数量上有一个合理比例，达到负载均衡。

综上，我们从C2 IP数量上大幅度的增长，推测目前Fodcha的规模大于4月份，日活Bot数量超过6W。当然再合理的推测也还是假设，欢迎有视野的社区伙伴不吝指正。

DDoS统计

回到C2 IP 44这个数字本身，纵然我们和僵尸网络battle多年见多识广，但这个数字依然让我们感到惊讶。世上没有无缘故的爱，光是这些IP资源，就得花费不少的，Fodcha的作者为什么愿意花这个钱呢？答案是DDoS攻击让他赚到了钱。我们节选了2022年6月29至今的数据，其攻击趋势和目标区域分布如下：

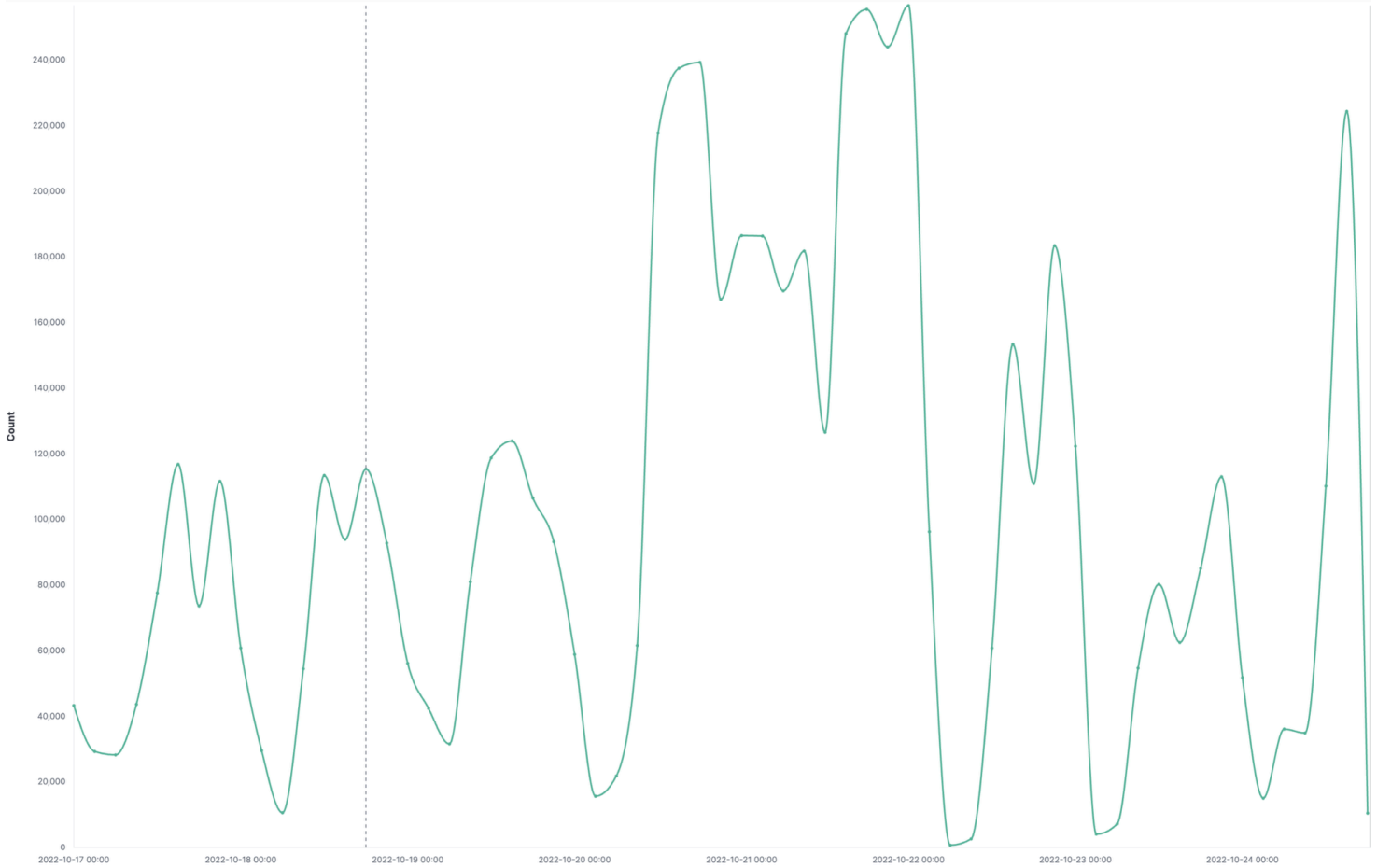


可以看出：

- 无愧于DDoS狂魔的称号，攻击几乎没有停歇，几乎打遍全球，日均攻击事件1K+。
- 中美两国颜色较深，说明两国累计被攻击目标及次数较多，综合考虑到两国在互联网上业务的比重原本就比较大，这里的“看起来多”是一种正常状

况。

攻击指令在7天内的时间分布如下所示，可以看出Fodcha发起的DDoS攻击遍及7 * 24小时，没有明显的时区性，我们倾向Fodcha是一个商业驱动的僵尸网络。



样本分析

我们将捕获的样本分成了 4 个大版本，其中在上一篇blog中已经分析过V1V2，此处就不再赘述了，本文选取最新的V4系列样本为主要分析对象，它们的基本信息如下所示：

```
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
```

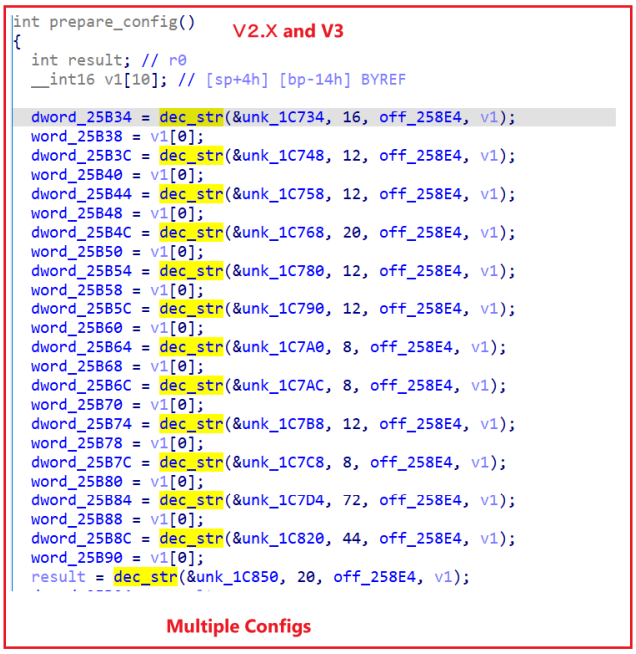
```
Packer: None
Version: V4.X
```

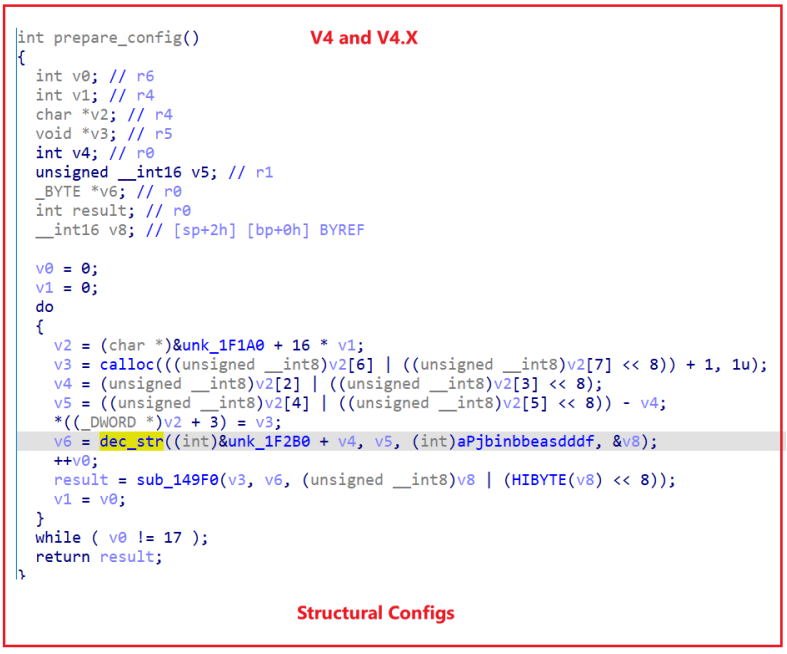
Fodcha的Bot在被侵入设备运行时，首先会从`运行参数`，`网络的连通性`，`是否设置“LD_PRELOAD”环境变量`，`自身是否被调试`等方面进行检查，如果不满足要求就直接退出，这些检查可以看成是一种对通过模拟器&沙箱提取IOC的简单对抗。

当满足要求运行要求时，则首先解密出配置信息，在Console上输出**snow slide**，然后就是一些常见的主机行为，如单一实例，进程名伪装，操控watchdog，清空特定端口进程，上报特定进程信息等，我们认为这些主机侧的功能没有太多亮点，因此不再展开分析，下文将着重从解密配置信息，网络通信，DDoS攻击等方面对Fodcha进行剖析。

解密配置信息(Config)

Fodcha在V2.X，V3使用并列的Config组织方式，而在V4,V4.X中则使用结构化的Config组织方式，下图非常清楚的显示了它们的区别。





虽然Config的组织方法不一样，但它们的加密方法是一样的，通过下面代码片段引用的常量可知，它们使用的是xxtea算法，密钥为`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 );
  }
}
```

经过逆向，我们编写了以下IDAPYTHON脚本来解密配置信息。

```
# 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)))
```

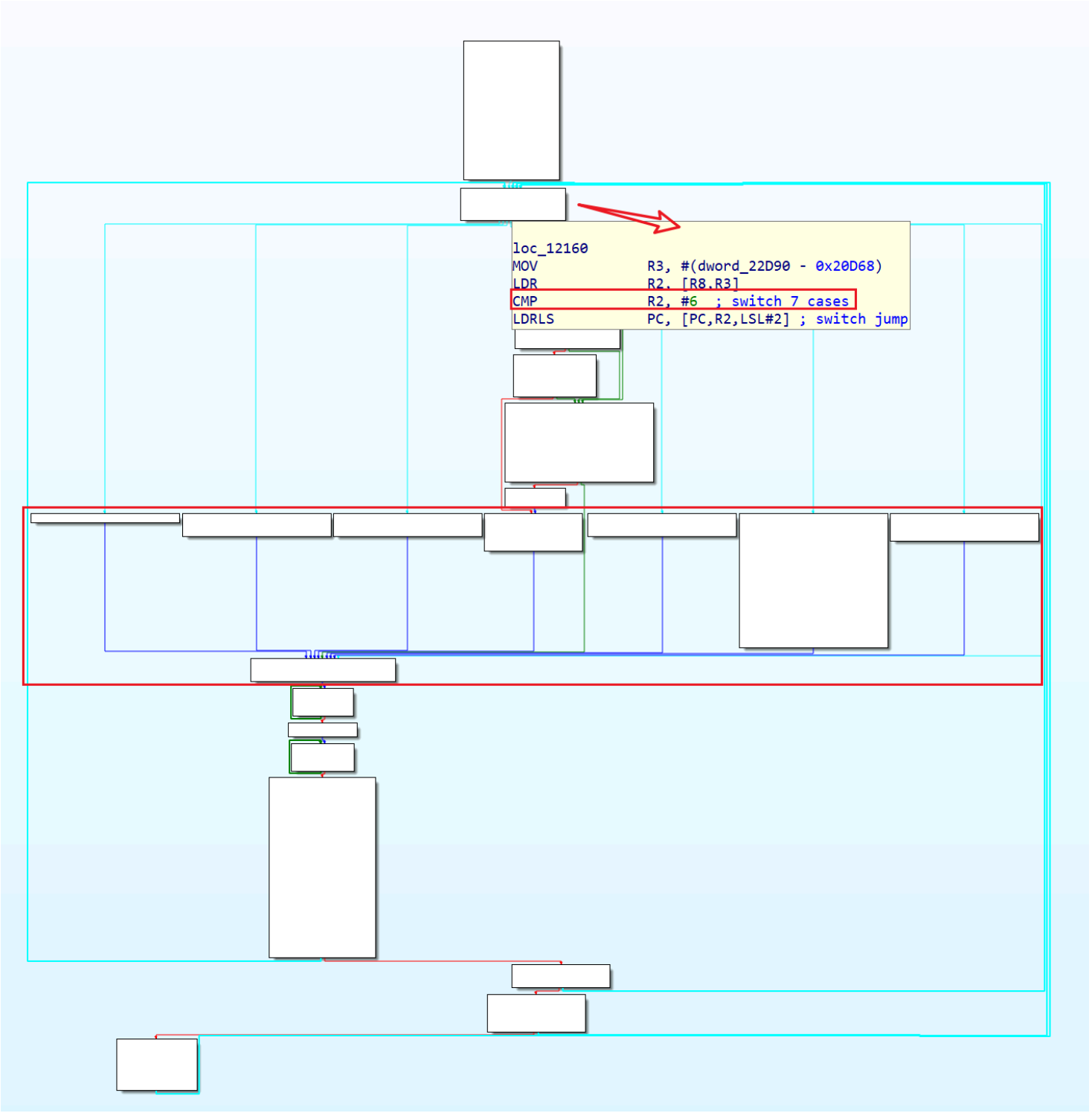
解密后的Config信息如下表所示，可以看到index 11还保留着“投降”的彩蛋，另外值得一提的是index 12，它是reporter服务器地址，Fodcha会将一些特定进程的信息上报给它。

INDEX	VALUE
0	snow slide
1	/proc/
2	/stat

INDEX	VALUE
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

网络通信

Fodcha的网络通信在代码层面有一个非常固定的特点：一个永真的While循环，通过switch-case进行各个阶段的处理，因此Fodcha各个版本的网络协议处理函数在IDA中产生的CFG图高度相似，这个特点可以帮助我们对样本进行辨别，对功能快速定位。

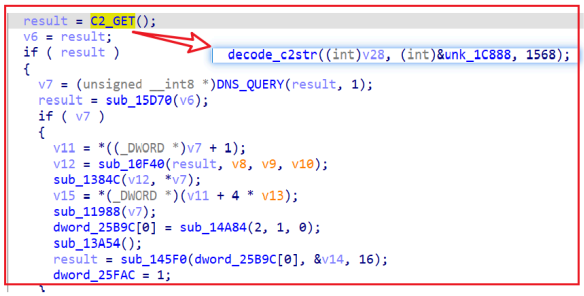


总的来说，Fodcha的网络通信要经过以下4个步骤：

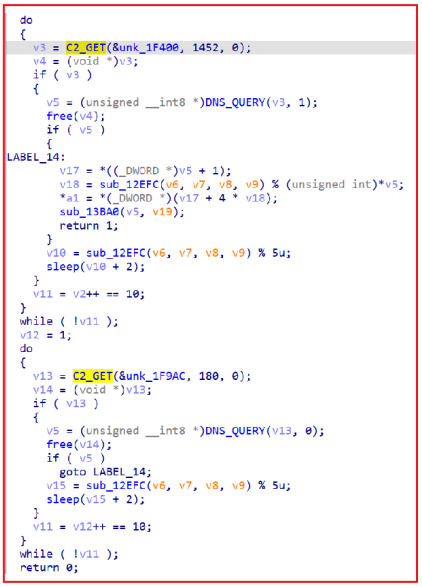
1. 解密C2
2. DNS查询
3. 建立通信
4. 执行指令

0x1: 解密C2

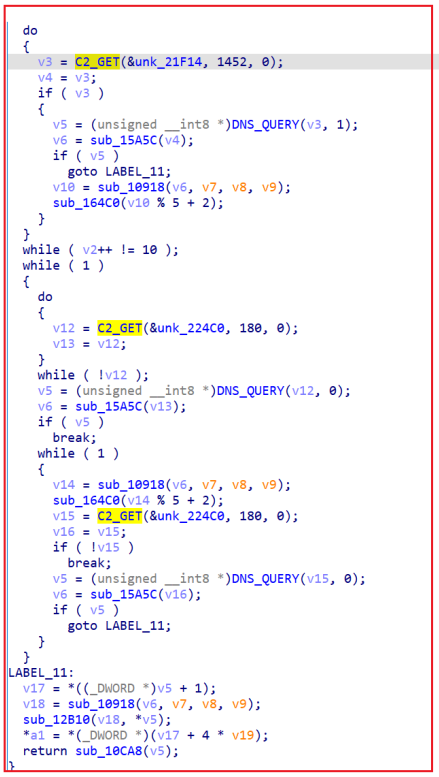
Fodcha的不同版本支持的C2种类是不一样的，V2.X只有1组OpenNIC C2；V3&V4拥有1组OpenNIC C2，1组ICANN C2；而V4.X则是最多的，1组OpenNIC C2，2组ICANN C2，下面的图非常清楚的显示了它们的区别。



V2.X



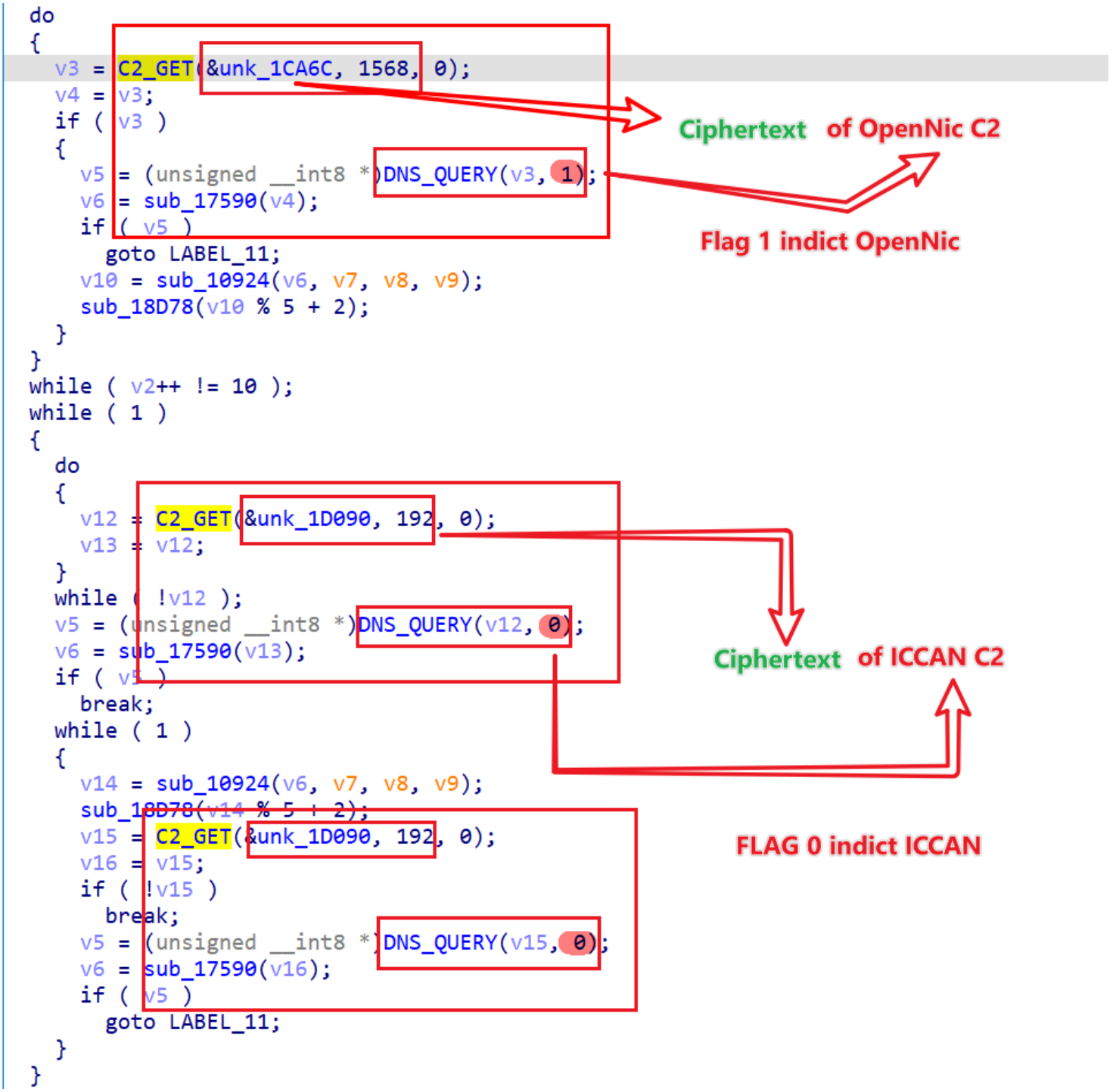
V3&V4



V4.X

Fodcha C&C

虽然C2种类&数量不一样，但是它们的处理逻辑如下图所示，几乎是一样的，首先通过C2_GET函数获得一个C2域名，然后通过DNS_QUERY函数获得C2对应的IP，其中C2_GET的第一个参数为C2密文数据，第2个参数为长度，而DNS_QUERY的第2个参数则暗示了C2的类型。



通过C2_GET可以获得一个有效的C2域名，它内部的实现可以分成2步：

- 首先得解密C2密文数据。
- 然后将它们构造成一个合法的域名。

解密C2密文数据

C2的密文数据使用了配置信息一样的加密方式，即xxtea，密钥也是**PJbiNbbeasddDfsc**，通过下面简单的IDAPYTHON脚本，即可解密出OpenNic C2数据。

```
#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)
```

解密后的C2数据如下图所示，可以看出C2数据由2部分组成，前面的是domain names，后面是TLDs，它们通过红框中的“/”符号分隔。


```
"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,"
```


构造域名

Fodcha有一个特定的域名构造方法，等效的Python实现如下所示：

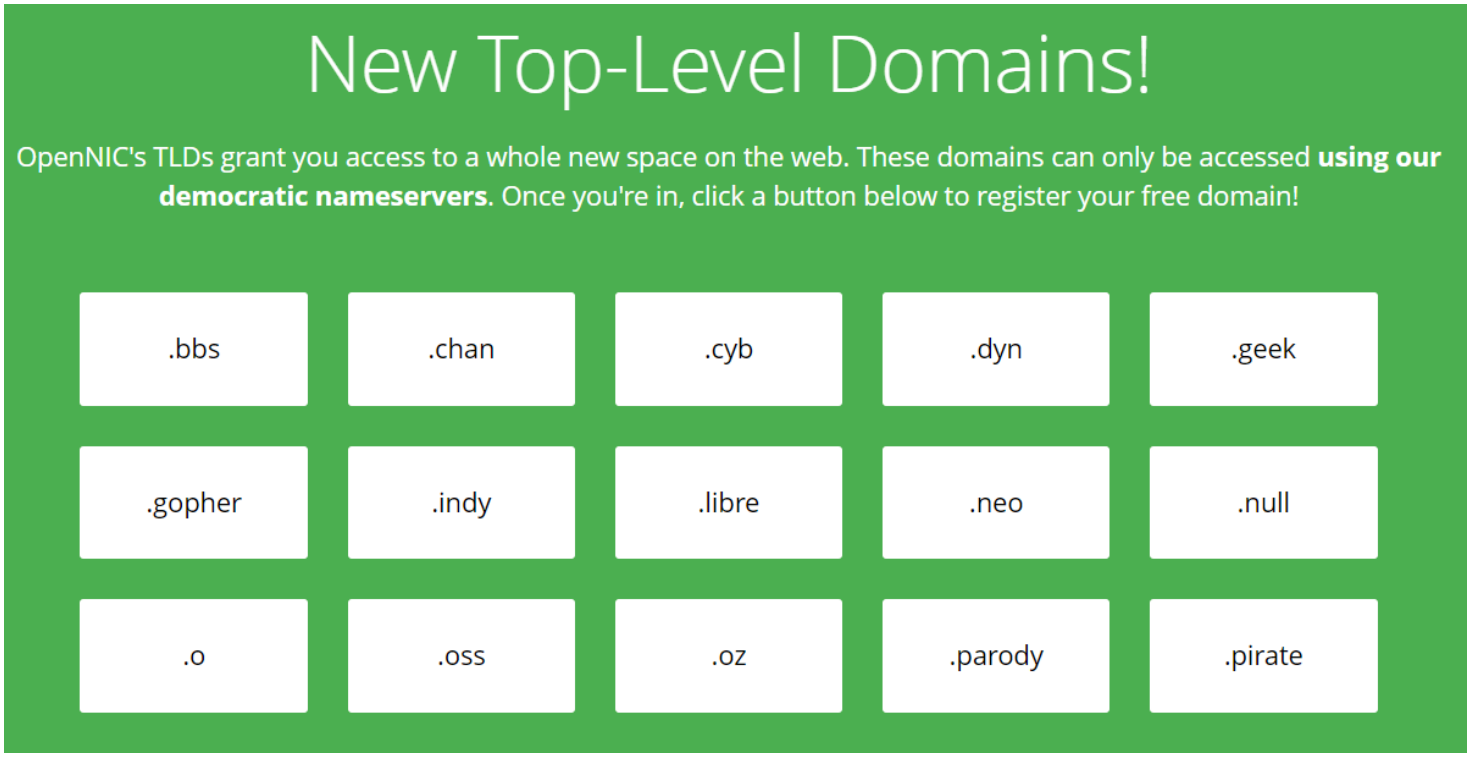
```
# md5: 899047ddf6f62f07150837aef0c1ebfb
# requirement: pip install xxtea-py
# test: ida7.6_python3

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)
```

将上文得到的C2数据做为输入，最终构造出以下14个OpenNIC C2。

```
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
```

对ICANN域名体系熟悉的读者，或许会在第一眼就认为我们的解密是错误的，因为ICANN的域名体系并不支持这些TLDs，它们肯定“无法解析”，事实上它们正是OpenNIC体系下的域名，OpenNIC是独立于ICANN的另一套域名体系，它支持下图所示的TLDs，OpenNIC的域名无法通过常见的DNS(如8.8.8.8，101.198.198.198)解析，必须使用指定的NameServer，更多的细节就不再展开，感兴趣的读者自行到其官方网站了解。

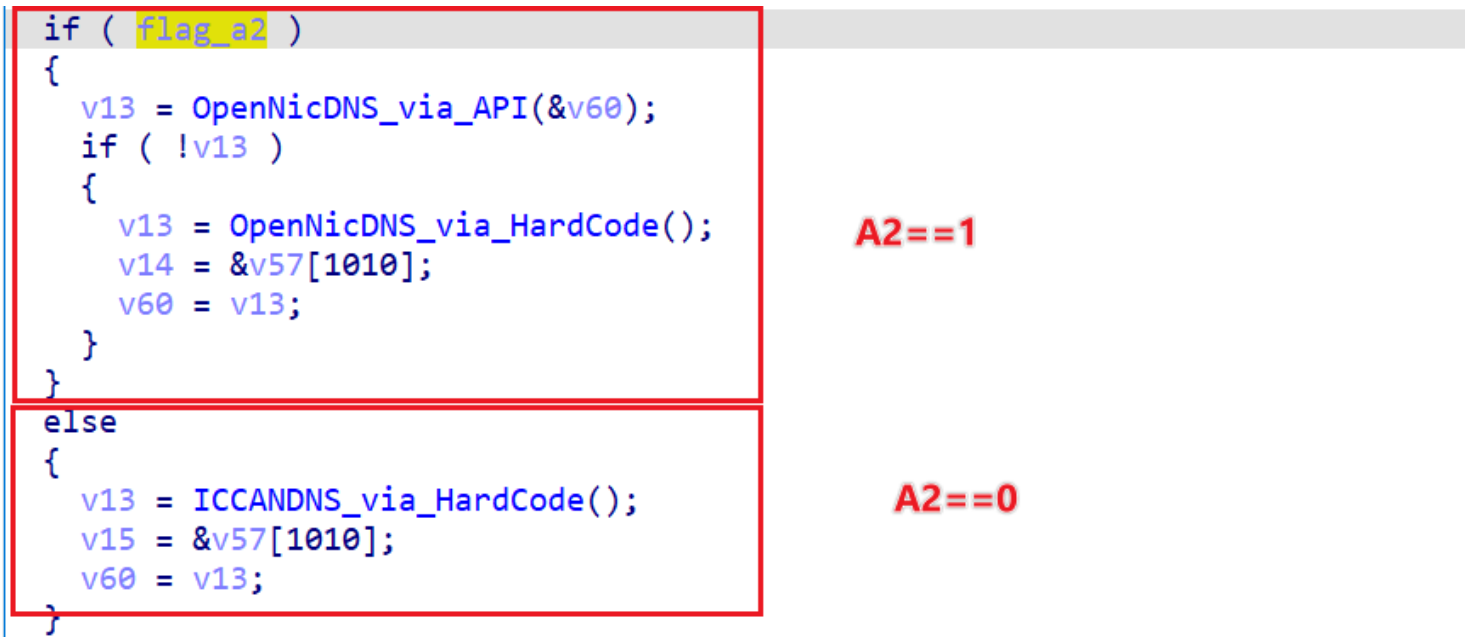


用同样的方法，我们可以得到以下4个ICANN C2。

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

0X2: DNS查询

当成功获得C2域名后，Bot通过函数**DNS_QUERY**进行域名解析，它的第2个参数是一个FLAG，暗示了OpenNIC/ICANN C2的不同处理过程，相应的代码片段如下所示：



可以看出对于OpenNIC C2的解析有2个选择：

- 选择1：通过API接口向**api.opennicproject.org**请求，动态的获取nameserver

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

Fake

```
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
```

- 选择2：使用下图所示的硬编码nameserver

```
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;
}
```

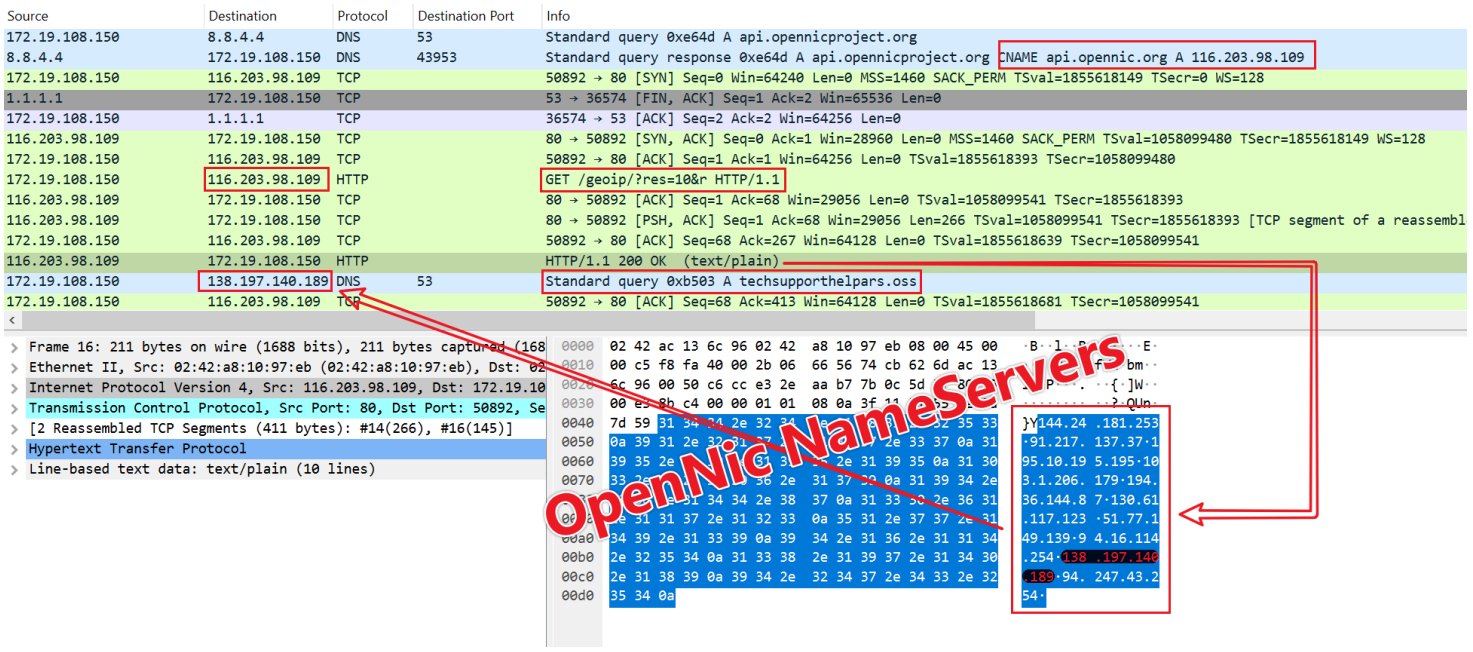
```
195.10.195.195
194.36.144.87
151.80.222.79
79.133.199.87
130.61.117.123
95.179.226.37
51.77.149.139
```

而对于ICANN C2则只有一个选择，即使用下图中的硬编码nameserver。

```
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

以实际解析 C2“techsupporthelpars.oss”为例，它的解析过程在网络流量中的体现如下所示：



为什么使用OpenNIC / ICANN 双C2?

Fodcha作者构建了一套OpenNIC / ICANN 双C2的冗余结构，他为什么要这么做呢?

从C2基础设施的角度出发，Fodcha被曝光后，其C2被一些服务商加入到了监控列表，进行拦截。例如Quad9DNS（9.9.9.9）就曾发过一个关于Fodcha域名流量spike的Twitter

在Fodcha被打击之后，其作者在重新选择C2基础设施时，看中了OpenNIC宣传的"DNS Neutrality"特性，通过绕开ICANN的域名体系，从根本上消除C2被监管&

接管的可能性，因此Fodcha在V2.X引入OpenNic C2，并将其做为主C2。

与此同时，OpenNIC C2可能存在一些问题，比如OpenNIC的NameServer在某些地区可能无法访问，或者域名解析上存在效率或稳定性的问题。出于健壮性的考虑，Fodcha作者在V3之 后重新加入ICANN C2作为后备C2，与主C2构成冗余结构。

0x3: 建立通信

Fodcha Bot通过又下代码片段和C2建立连接，一共有个22个端口。

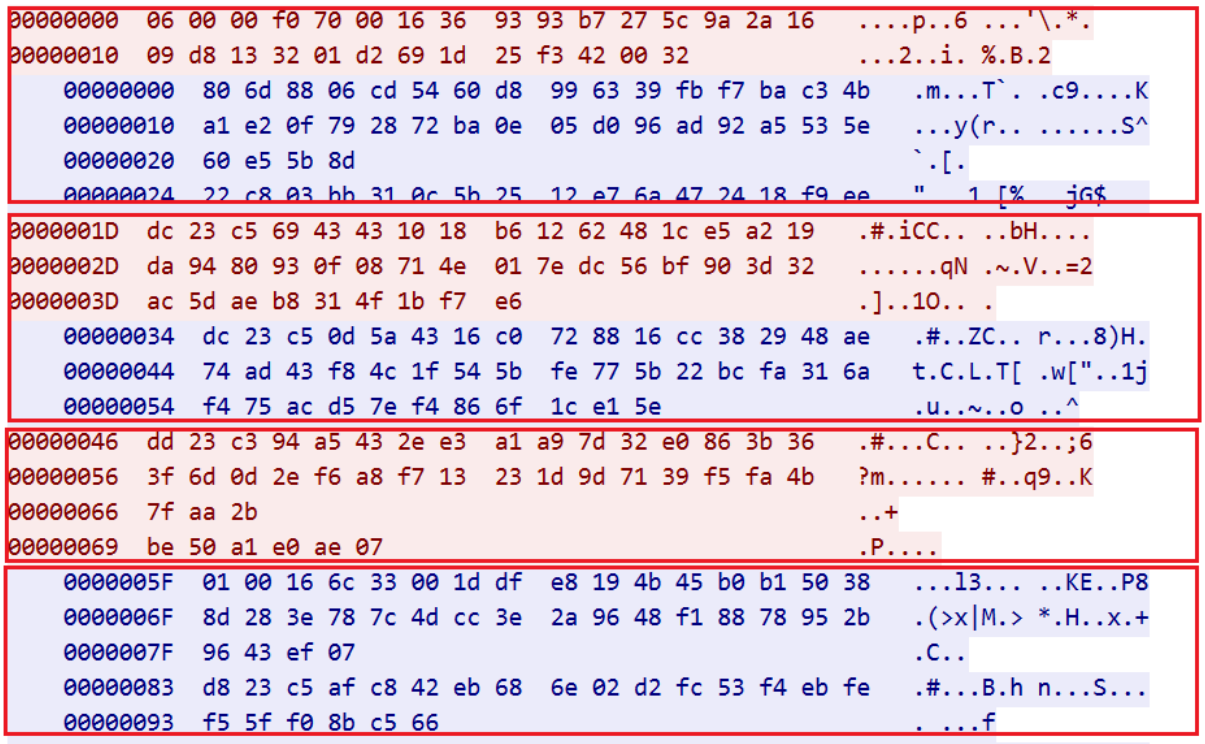
```
int __fastcall establist_connect(int a1, int a2, int a3, int a4)
{
  unsigned int v4; // r0
  int result; // r0
  sockaddr_in v6; // [sp+0h] [bp-20h] BYREF

  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
```

当成功和C2建立连接后，Bot与C2必须经过3个阶段的交互，才能真正建立通信。

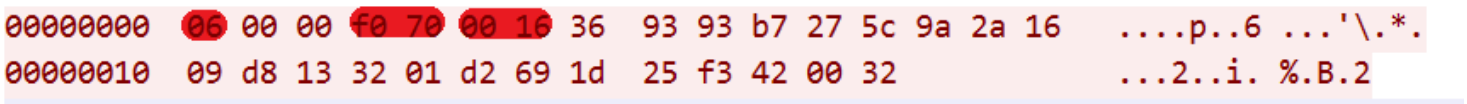
- 阶段1：Bot向C2请求chacha20加密算法的的key&nonce。
- 阶段2：Bot与C2使用阶段1的key&nonce进行身份确认。
- 阶段3：Bot将加密后的上线&分组信息发往C2。

为了辅助分析，我们在受限的环境内运行了Bot样本，并使用`fsdsaD`做为分组字串，产生了下图所示的网络流量，下文将详细介绍此流量是如何生成的。



阶段1：Bot ---> C2 ,格式为head(7 bytes) + body(random 20-40 bytes)

Bot主动向C2发送**netstage=6**的初始化消息，这个消息的格式为head+body，各字段含义如下所示：



head

head的长度为7 bytes，格式如下所示：

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

checksum

head中的checksum使用的是tcp/ip的checksum，它计算对象为整个payload，checksum所在偏移的原始值为"\x00\x00"，checksum的python实现如下所示：

```
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为随机生成的内容，无意义。

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

阶段1：C2--->Bot，共2轮

当C2收到Bot的**netstage=6**的消息后，就会向BOT发送2轮的数据。

```
00000000  80 6d 88 06 cd 54 60 d8  99 63 39 fb f7 ba c3 4b  .m...T`. .c9....K
00000010  a1 e2 0f 79 28 72 ba 0e  05 d0 96 ad 92 a5 53 5e  ...y(r.. ......S^
00000020  60 e5 5b 8d                                      `.[.
00000024  22 c8 03 bb 31 0c 5b 25  12 e7 6a 47 24 18 f9 ee  "...1.[% ..jG$...
```

- 第一轮，36 bytes , 原信息被xxtea加密，解密后作为chacha20的key，长度为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
```

- 第二轮，16 bytes，原信息被xxtea加密，解密后作为chacha20的nonce，长度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
```

阶段2：Bot--->C2，chacha20加密

Bot收到chacha20的key和nonce后，就向C2发送**netstage=4**的消息，此次消息使用chacha20加密，key&nonce由上一阶段获得，加密的轮数为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.. .
```

我们可以使用下面的python代码可以解密上面的流量，

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

解密后的流量如下所示，它的格式依然是前文所述的head（7 bytes）+body，其中head的netstage字段的值为04，代表身份认证。

```
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                       ..~.....^
```

阶段2：C2--->Bot，chacha20加密

C2在收到Bot的身份认证消息后，也向Bot的数据发送**netstage=4**的消息，同样使用chacha20加密，且key,nonce,轮数和Bot使用的是一样的。

```
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 ..^
```

使用和Bot相同的代码解密流量，可以看出它的格式也是head+body，netstage的值也为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...
```

在Bot和C2互发**netstage=4**的消息之后，代表阶段1的chacha20 key&nonce被双方认可，彼此的身份认证完成，Bot进入下一阶段准备上线。

阶段3：Bot--->C2，共2轮，chacha加密

Bot向C2发送**netstage=5**的消息，表示准备上线，接着再自己的分组信息上报给C2，这2轮消息也使用chacha20加密。

- 第一轮

```
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                                          ..+
```

- 第二轮

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

上述2轮的数据解密后如下所示，可以看出分组的内容正是预设的"`fsdsaD`"，这代表我们的分析是正确的，至此Bot成功上线，开始等待执行C2下发的指令。

```
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:执行指令

Bot成功上线后，支持的netstage编号，如下所图所示，其中最重要的就是**netstage=1**代表DDoS任务，Fodcha复用了大量Mirai的攻击代码，一共支持17种攻击方法。

```
  if ( (unsigned __int8)netstage == 2 )
    exit(1);
  if ( (unsigned __int8)netstage == 3 )
    return heartbeat(dword_20D68);
  if ( (unsigned __int8)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);
}
```

if netstage == 1

以下图的DDos_Task流量（netstage=01）为例：

```
0000005F  01 00 16 6c 33 00 1d df  e8 19 4b 45 b0 b1 50 38   ...l3... ..KE..P8
0000006F  8d 28 3e 78 7c 4d cc 3e  2a 96 48 f1 88 78 95 2b   .(>x|M.> *.H..x.+
0000007F  96 43 ef 07                                       .C..
00000083  d8 23 c5 af c8 42 eb 68  6e 02 d2 fc 53 f4 eb fe   .#...B.h n...S...
00000093  f5 5f f0 8b c5 66                                 ._...f
```

ddos instruction

攻击指令依然采用chacha20加密，解密后的指令如下所示，相信熟悉Mirai的读者看到此处肯定会心一笑。

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

上述攻击指令的格式和解析方式如下表所示：

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

OFFSET	LEN (BYTES)	VALUE	MEANING
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)

当Bot接收到上述指令，就会使用payload为1字节的tcp报文对目标xx.20.147.1:443进行DDoS攻击，这和实际抓包的流量是能对应上的。

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 [

花絮

0x01: 种族歧视

从某些OpenNIC C2的构词上来说，Fodcha的作者似乎对黄种人，黑人有比较大的敌意。

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

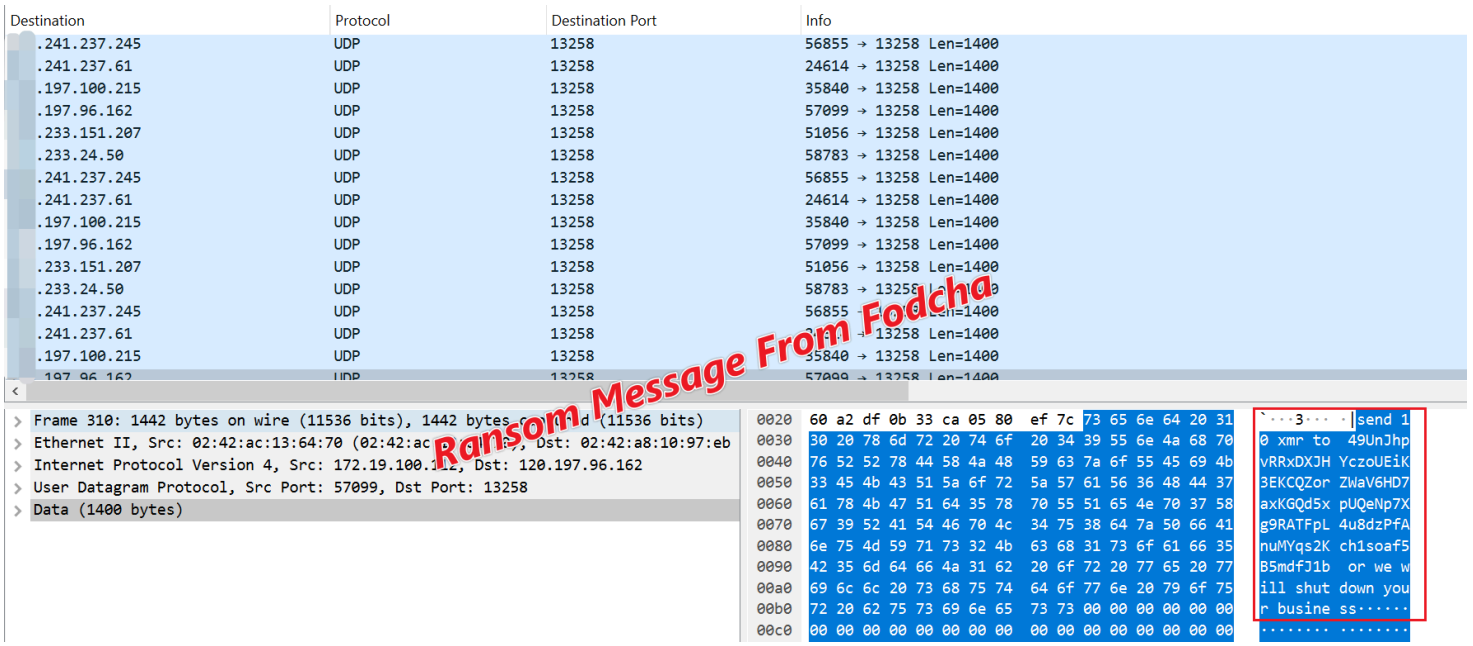
```
wehateyellow
```

0x02: 攻击即勒索

Fodcha曾在其下发的UDP攻击指令中，附带以下字串：

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

Bot打出的攻击流量如下所示，该钱包地址似乎是非法的，没能给我们更多的线索，但从这一行为出发，或许Fodcha背后的运营者正在尝试攻击即勒索这种商业模式。



联系我们

感兴趣的读者，可以在 twitter 或者通过邮件netlab[at]360.cn联系我们。

解决方案

基于Netlab多年研究工作孵化的360全系列DNS安全产品均已支持文中远控服务器的拦截和检测，同时内置多种算法可有效发现和拦截各种未知威胁，建议企业客户

接入360 DNS安全SaaS平台或部署本地360DNS安全产品，及时防范此类新型威胁，避免企业资产失陷。联系人: wangkun-bd@360.cn

IoC

C2

```
yellowchinks.geek
yellowchinks.dyn
wearelegal.geek
tsengtsing.libre
techsupporthelpars.oss
respectkkk.geek
pepperfan.geek
peepeepoo.libre
obamalover.pirate
milfsfors3x[.]com
funnyyellowpeople.libre
fridgexperts[.]cc
forwardchinks[.]com
folded[.]in
doodleching[.]com
cookiemonsterboob[.]com
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

Botnet

Fodcha Is Coming Back, Raising A Wave of Ransom DDoS

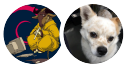
Background On April 13, 2022, 360Netlab first disclosed the Fodcha botnet. After our article was published, Fodcha suffered a crackdown from the relevant authorities, and its authors quickly responded by leaving "Netlab pls leave me alone I surrender" in an updated sample.No surprise, Fodcha's authors

Botnet

PureCrypter is busy pumping out various malicious malware families

In our daily botnet analysis work, it is common to encounter various loaders.Compared to other types of malware, loaders are unique in that they are mainly used to "promote", i.e., download and run other malware on the infected machine. According to our observations, most loaders are

See all 114 posts →



Oct 31, 2022 · 16 min read

Aug 29, 2022 · 12 min read