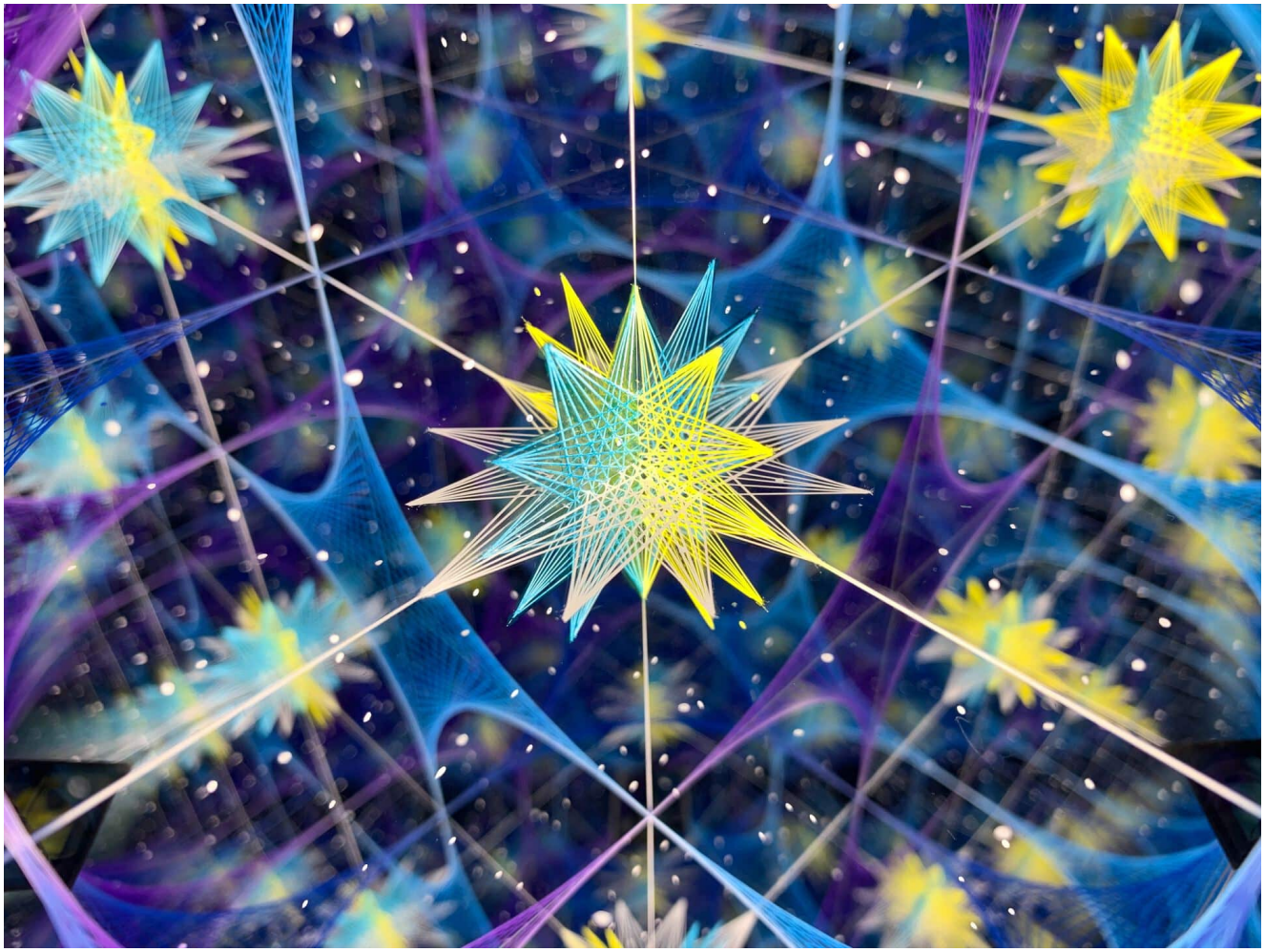
Blog.



2019-11-12

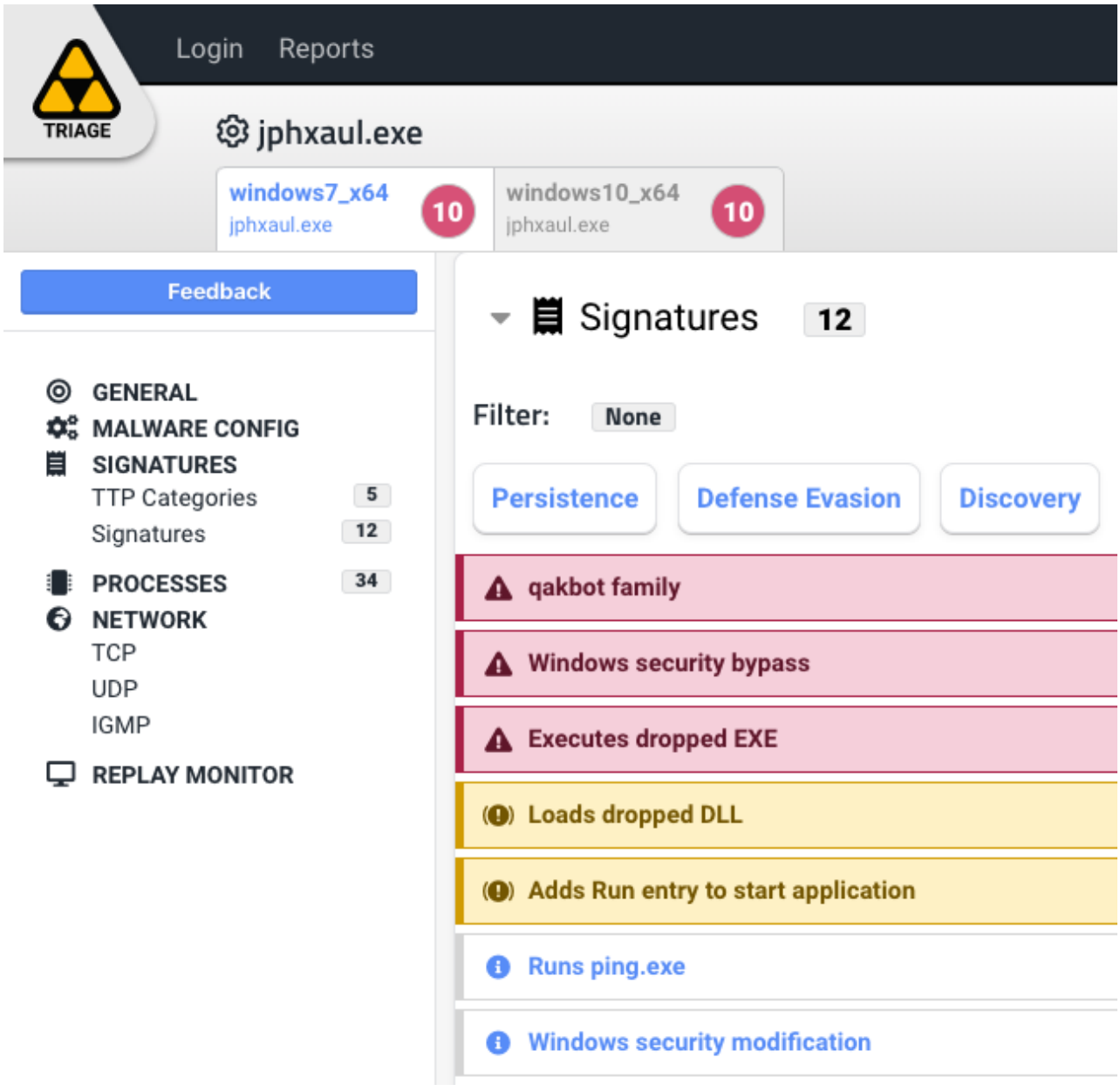
- triage
- malware
- sandbox
- banker

Written by
Markel Picado (d00rt)

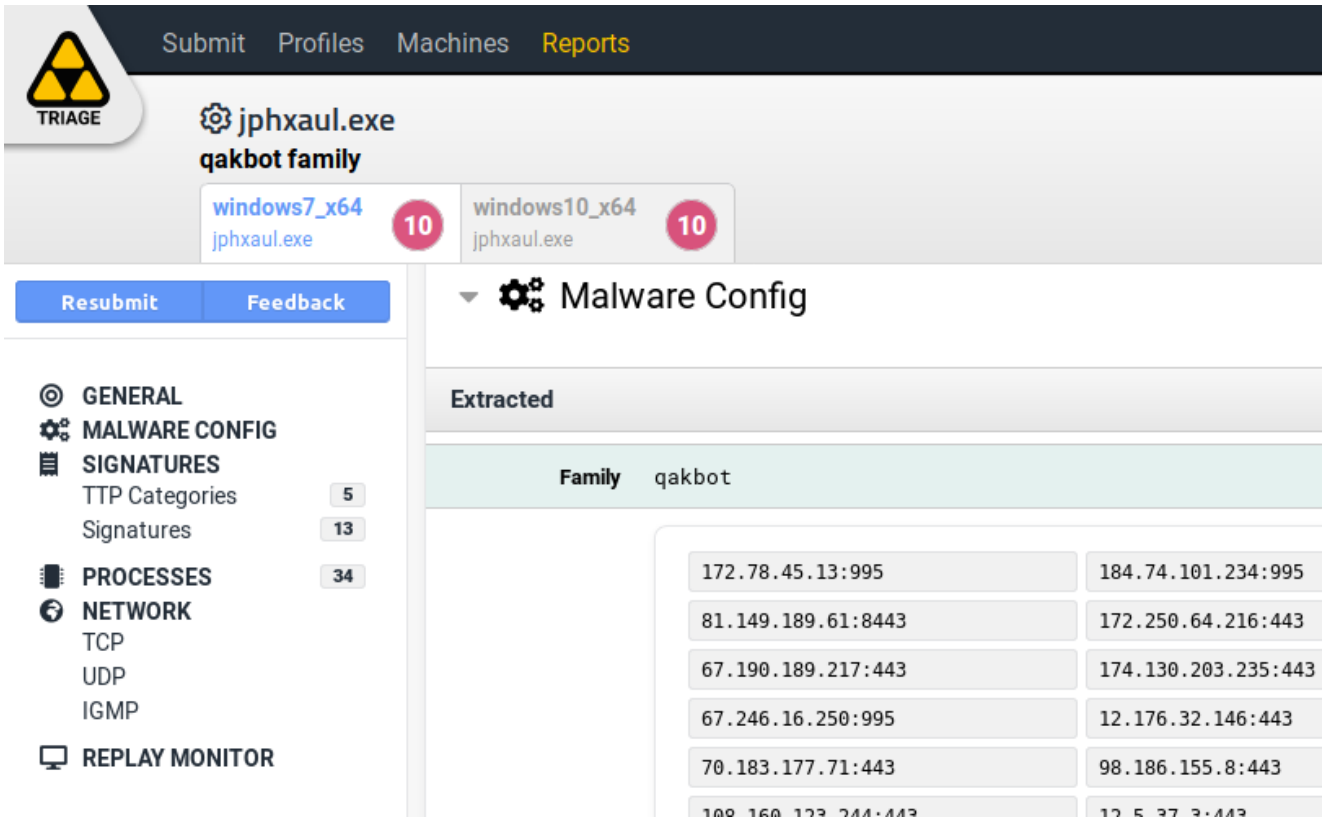
Summary

Qbot or Qakbot is a sophisticated worm with banking capabilities. This malware family has been infecting computers since 2009, utilizing a number of techniques (some of them quite advanced) which make it difficult to detect. It has a packing layer, anti-VM techniques, anti-debug techniques, and anti-sandbox techniques which make the analysis of this threat difficult. Qakbot is capable of updating itself and this also makes this threat more complex to detect since it is constantly changing on disk.

Using Triage we analyzed the most recent variant of this malware, and we added a new module to support the detection and configuration extraction of Qakbot samples as shown in the image below. A tool to deobfuscate the Qakbot payload is also included qakbuscator.py.



Qakbot family detection in Hatching Triage

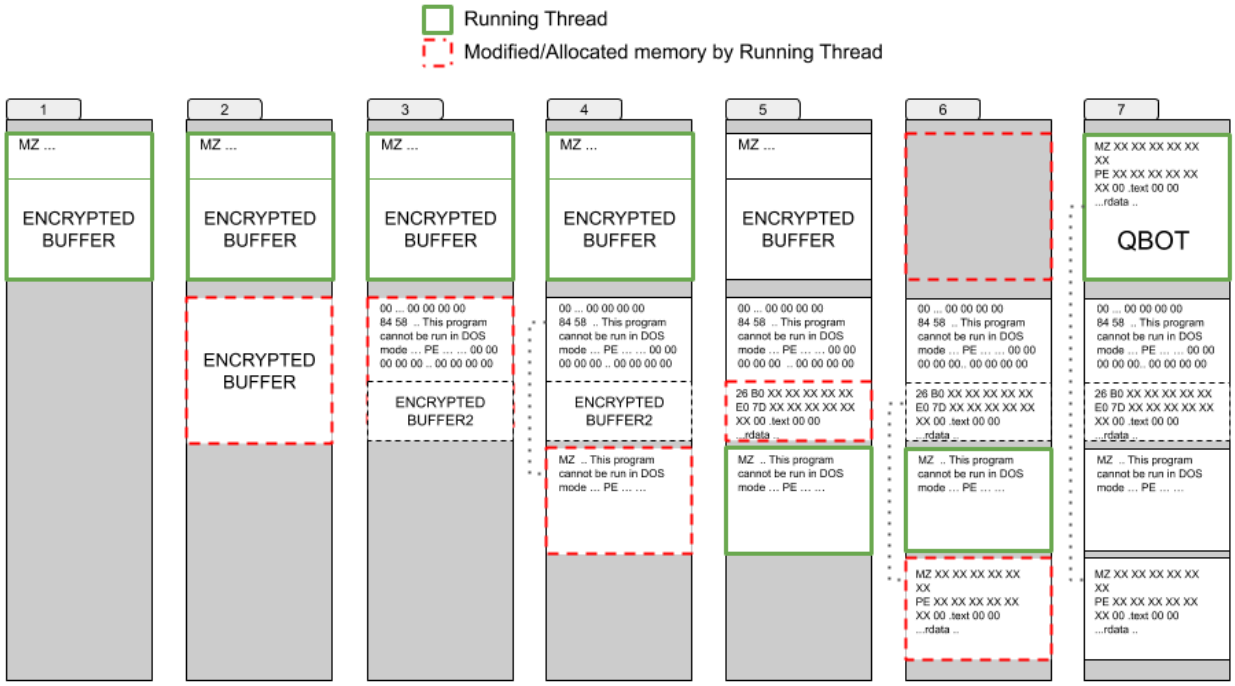


Qakbot config extraction Hatching Triage

Unpacking process

Qakbot has a custom packer. There are probably other versions of Qakbot in the wild with different packers, but this section is based on analysis of the packer for the sample: `e736cf964b998e582fd2c191a0c9865814b632a315435f80798dd2a239a5e5f5`.

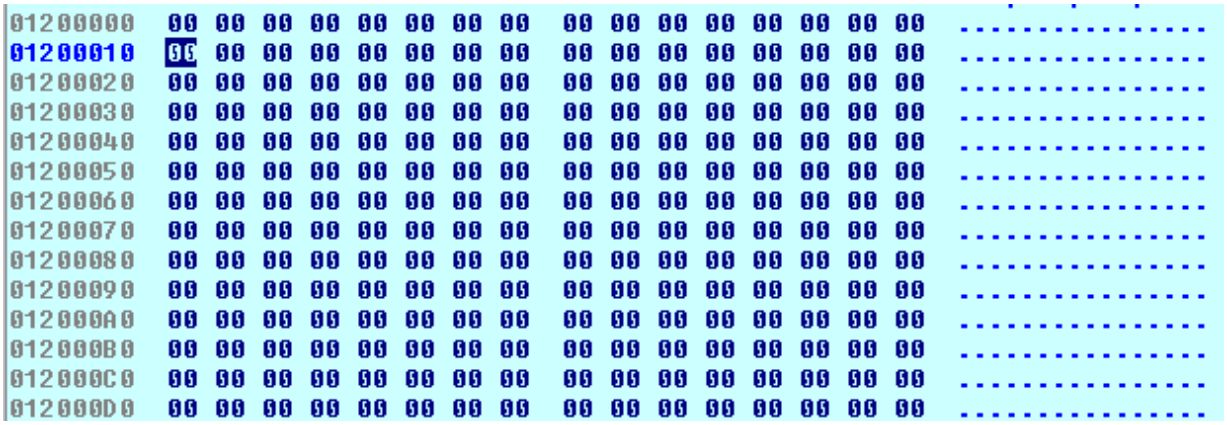
In summary, the unpacking process is as follows:



The unpacking process

The packer allocates memory and then drops an encrypted buffer there **(Step 1-2)**.

The dropped buffer is decrypted and the decrypted data contains a PE file **(Step 3)**. This PE file is not at the beginning of the buffer but starts at offset `0x427`. From the beginning to the PE file offset is filled with `0x00` bytes.



Offset is filled with `0x00` bytes

This could be a trick to make analysts think that this function is “freeing memory” or that it’s a memset-like function.

The PE header is modified - this can also confuse analysts or memory dumping tools that look for PE file signatures since they can’t find the “MZ” magic number. This is shown in the image below.

```
012003F7  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
01200407  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
01200417  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
01200427  84 58 07 42 1B BD 00 00  04 00 00 00 FF FF 00 00  äX.B.¢......¨¨..
01200437  B8 00 00 00 00 00 00 00  40 00 00 00 00 00 00 00  ©.......@.......
01200447  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
01200457  00 00 00 00 00 00 00 00  00 00 00 00 71 13 E1 A3  ............q.ßú
01200467  0E 1F BA 0E 00 B4 09 CD  21 B8 01 4C CD 21 54 68  ..¦..¦.-¶©.L-¶Th
01200477  69 73 20 70 72 6F 67 72  61 6D 20 63 61 6E 6E 6F  is·program·canno
01200487  74 20 62 65 20 72 75 6E  20 69 6E 20 44 4F 53 20  t·be·run·in·DOS·
01200497  6D 6F 64 65 2E 0D 0D 0A  24 00 00 00 00 00 00 00  mode....$.......
012004A7  A3 0F F6 0B E7 6E 98 58  E7 6E 98 58 E7 6E 98 58  ú.÷.þnÿXþnÿXþnÿX
012004B7  BA 4C 93 58 E4 6E 98 58  64 72 96 58 F1 6E 98 58  ¦LôXõnÿXdrûX±nÿX
012004C7  BA 4C 92 58 BE 6E 98 58  85 71 8B 58 EA 6E 98 58  ¦LÆX¥nÿXàqïXÛnÿX
012004D7  E7 6E 99 58 90 6E 98 58  E7 6E 98 58 E0 6E 98 58  þnÖX.nÿXþnÿXónÿX
012004E7  B8 4C 93 58 E8 6E 98 58  20 68 9E 58 E6 6E 98 58  ©LôXþnÿX·h×XµnÿX
```

Modified PE header

The decrypted PE file image size is calculated to allocate memory for it. The PE file is copied (mapped as a windows loader would do) from the decrypted buffer to the newly allocated memory **(Step 3-4)**.

```
01290000  84 58 07 42 1B BD 00 00  04 00 00 00 FF FF 00 00  äX.B.¢......¨¨..
01290010  B8 00 00 00 00 00 00 00  40 00 00 00 00 00 00 00  ©.......@.......
01290020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
01290030  00 00 00 00 00 00 00 00  00 00 00 00 71 13 E1 A3  ............q.ßú
01290040  0E 1F BA 0E 00 B4 09 CD  21 B8 01 4C CD 21 54 68  ..¦..¦.-¶©.L-¶Th
01290050  69 73 20 70 72 6F 67 72  61 6D 20 63 61 6E 6E 6F  is·program·canno
01290060  74 20 62 65 20 72 75 6E  20 69 6E 20 44 4F 53 20  t·be·run·in·DOS·
01290070  6D 6F 64 65 2E 0D 0D 0A  24 00 00 00 00 00 00 00  mode....$.......
01290080  A3 0F F6 0B E7 6E 98 58  E7 6E 98 58 E7 6E 98 58  ú.÷.þnÿXþnÿXþnÿX
01290090  BA 4C 93 58 E4 6E 98 58  64 72 96 58 F1 6E 98 58  ¦LôXõnÿXdrûX±nÿX
012900A0  BA 4C 92 58 BE 6E 98 58  85 71 8B 58 EA 6E 98 58  ¦LÆX¥nÿXàqïXÛnÿX
UNKNOWN 01290000: debug022:01290000
```

Hex View-1

```
01200427  84 58 07 42 1B BD 00 00  04 00 00 00 FF FF 00 00  äX.B.¢......¨¨..
01200437  B8 00 00 00 00 00 00 00  40 00 00 00 00 00 00 00  ©.......@.......
01200447  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
01200457  00 00 00 00 00 00 00 00  00 00 00 00 71 13 E1 A3  ............q.ßú
01200467  0E 1F BA 0E 00 B4 09 CD  21 B8 01 4C CD 21 54 68  ..¦..¦.-¶©.L-¶Th
01200477  69 73 20 70 72 6F 67 72  61 6D 20 63 61 6E 6E 6F  is·program·canno
01200487  74 20 62 65 20 72 75 6E  20 69 6E 20 44 4F 53 20  t·be·run·in·DOS·
01200497  6D 6F 64 65 2E 0D 0D 0A  24 00 00 00 00 00 00 00  mode....$.......
012004A7  A3 0F F6 0B E7 6E 98 58  E7 6E 98 58 E7 6E 98 58  ú.÷.þnÿXþnÿXþnÿX
```

The PE is copied over

```
20      do
21      {
22        v6 = v7;
23        v10 = v8;
24        if ( !v8 )
25          memcpy(DECRYPTEDBUFFER, NEWALLOCATEDMEMORY, *(_DWORD *)(NEWALLOCATEDMEMORY + 396));
26        ++v8;
27        v7 += 40;
28        memcpy(*(_DWORD *)(v6 + 12) + DECRYPTEDBUFFER, *(_DWORD *)(v6 + 20) + NEWALLOCATEDMEMORY,
29      }
30      while ( v10 != 3 );
```

Stack view

```
0012FCB8  01294000
0012FCBC  01202E27
0012FCC0  00000200
0012FCC4  001E0000
0012FCC8  01291000
0012FCCC  001E0000
0012FCD0  00000000
0012FCD4  605380EF
0012FCD8  0012FD9C
```

Mapping file to Allocated Memory

Once the file is mapped to the newly allocated memory, the header is fixed as shown in the following image. Once the PE file is mapped its entry point is called. **(Step 4)**

```
*(_WORD *)NEWALLOCATEDMEMORY_1 = 'ZM';
*(_DWORD *)(NEWALLOCATEDMEMORY_1 + 0x3C) = 0x138;
*(_DWORD *)(NEWALLOCATEDMEMORY_1 + 0x138) = 'EP';
```

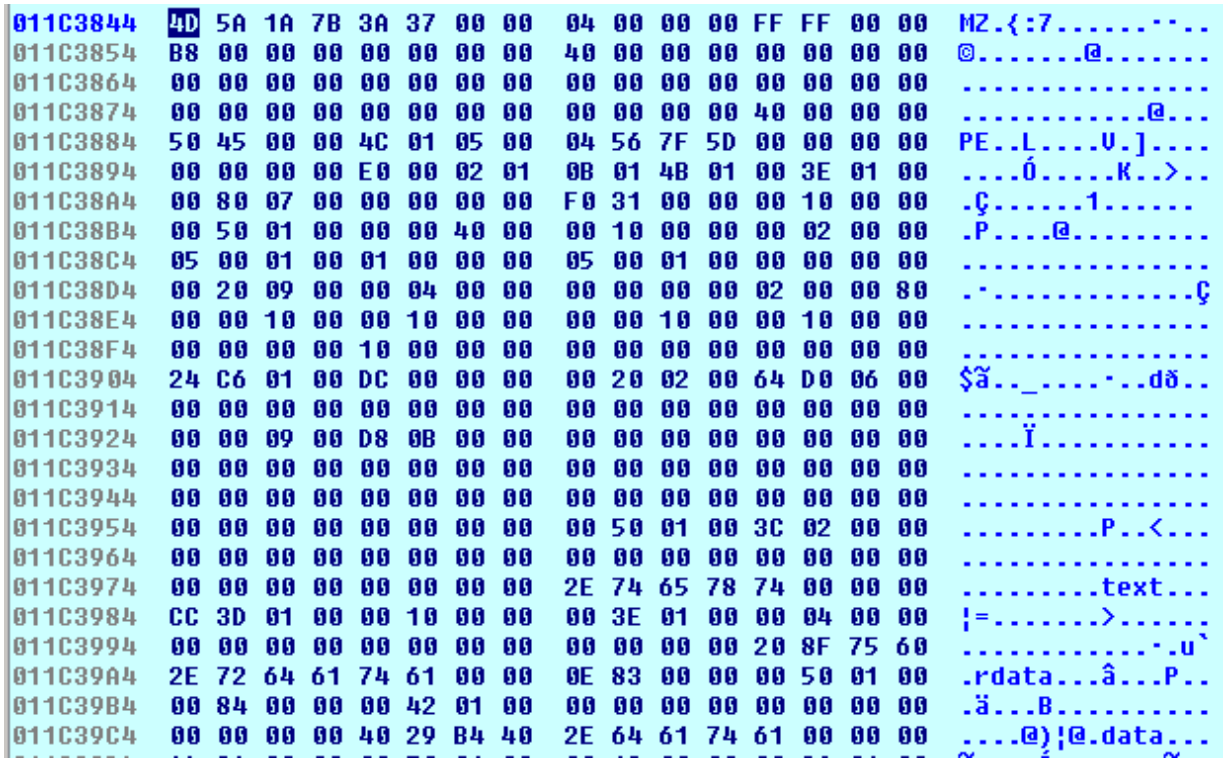
Calling the entry point

This PE is going to read the rest of the previous decrypted buffer since there is still some encrypted data. Once the data is decrypted a new PE file can be found. **(Step 4-5)**

```
011C3844  26 B0 1A 7B 3A 37 00 00  04 00 00 00 FF FF 00 00  &¦.{:7......¨¨..
011C3854  B8 00 00 00 00 00 00 00  40 00 00 00 00 00 00 00  ©.......@.......
011C3864  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
011C3874  00 00 00 00 00 00 00 00  00 00 00 00 74 0B 32 4A  ............t.2J
011C3884  E0 7D 1F B5 08 EA 05 00  04 56 7F 5D 00 00 00 00  Ó}.Á.Û...V.]....
011C3894  00 00 00 00 D2 A0 02 01  C4 74 4B 01 00 3E 01 00  ....Êá..-tK..>..
011C38A4  00 80 07 00 00 00 00 00  70 C3 00 00 00 10 00 00  .Ç......p+......
011C38B4  00 50 01 00 00 00 40 00  00 10 00 00 00 02 00 00  .P....@.........
011C38C4  05 00 01 00 01 00 00 00  05 00 01 00 00 00 00 00  ................
011C38D4  00 20 09 00 00 04 00 00  00 00 00 00 02 00 00 80  .·.............Ç
011C38E4  00 00 10 00 00 10 00 00  00 00 10 00 00 10 00 00  ................
011C38F4  00 00 00 00 10 00 00 00  00 00 00 00 00 00 00 00  ................
011C3904  24 C6 01 00 DC 00 00 00  00 20 02 00 64 D0 06 00  $ã.._....·..dð..
011C3914  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
011C3924  00 00 09 00 D8 0B 00 00  00 00 00 00 00 00 00 00  ....Ï...........
011C3934  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
011C3944  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
011C3954  00 00 00 00 00 00 00 00  00 50 01 00 3C 02 00 00  .........P..<...
011C3964  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
011C3974  00 00 00 00 00 00 00 00  2E 74 65 78 74 00 00 00  .........text...
011C3984  CC 3D 01 00 00 10 00 00  00 3E 01 00 00 04 00 00  ¦=.......>......
011C3994  00 00 00 00 00 00 00 00  00 00 00 00 20 8F 75 60  ............·.u`
011C39A4  2E 72 64 61 74 61 00 00  0E 83 00 00 00 50 01 00  .rdata...â...P..
011C39B4  00 84 00 00 00 42 01 00  00 00 00 00 00 00 00 00  .ä...B..........
011C39C4  00 00 00 00 40 29 B4 40  2E 64 61 74 61 00 00 00  ....@)¦@.data...
```

The new PE file

This time the header is also modified and is fixed before mapping it. **(Step 5-6)**

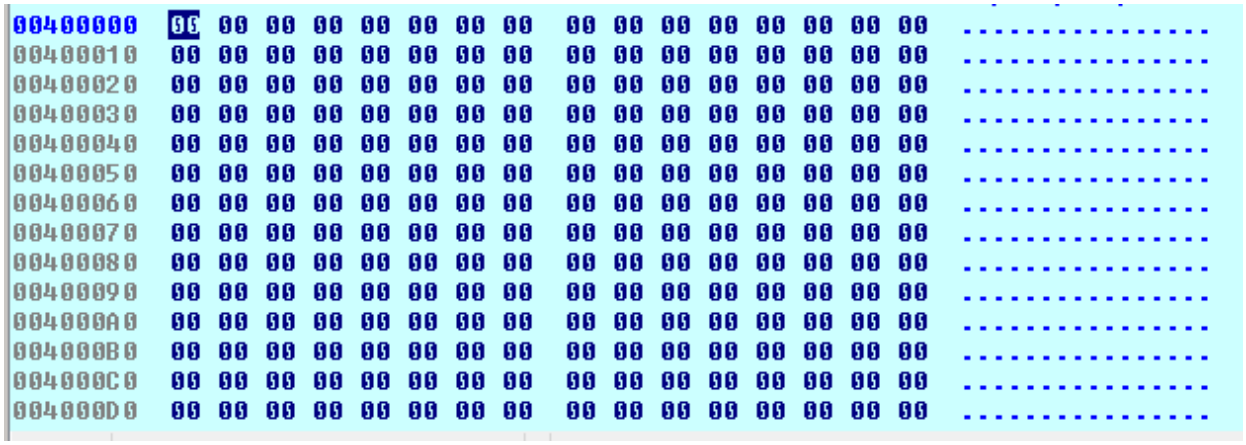


New PE file with new header

The decrypted PE is Qakbot itself. In this case the PE header doesn't have the well-known string "This program cannot be run in DOS mode", because the DOS-Stub was deleted.

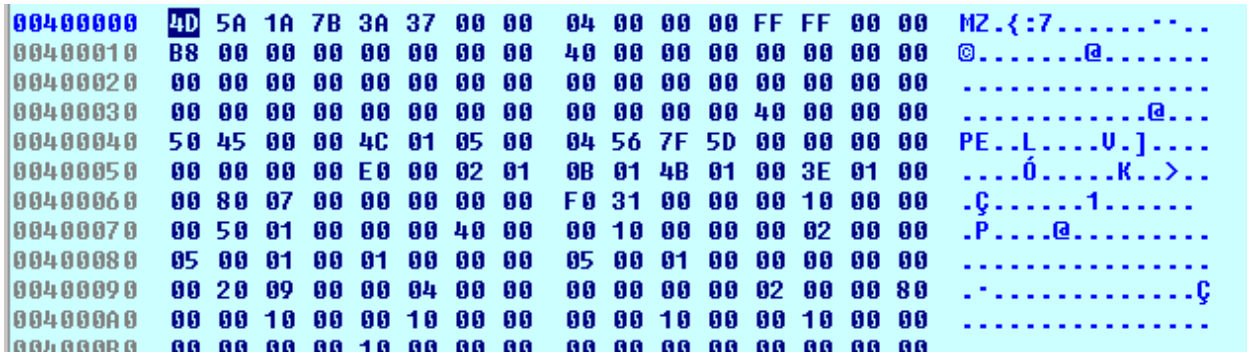
This PE file is the final payload of Qakbot so finally the PE file is mapped to the ImageBaseAddress of the original file **(Step 6)**.

The original image loaded at address `0x400000` is wiped.



The address 0x400000 is wiped

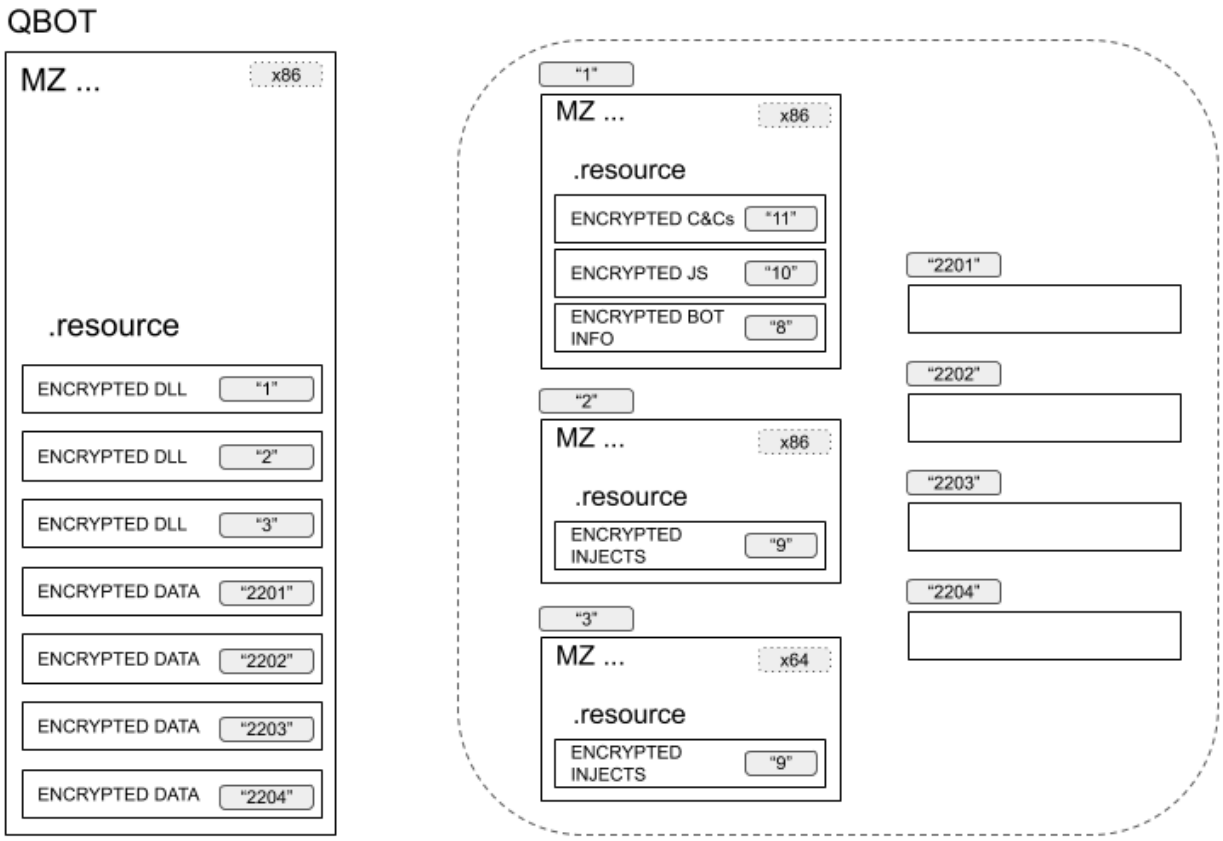
The newly unpacked PE (Qakbot) is copied to the original image base address `0x400000`.



Qakbot copied over to 0x400000

So, after mapping the Qakbot binary the execution flow goes to the EntryPoint of this file. **(Step 7)**.

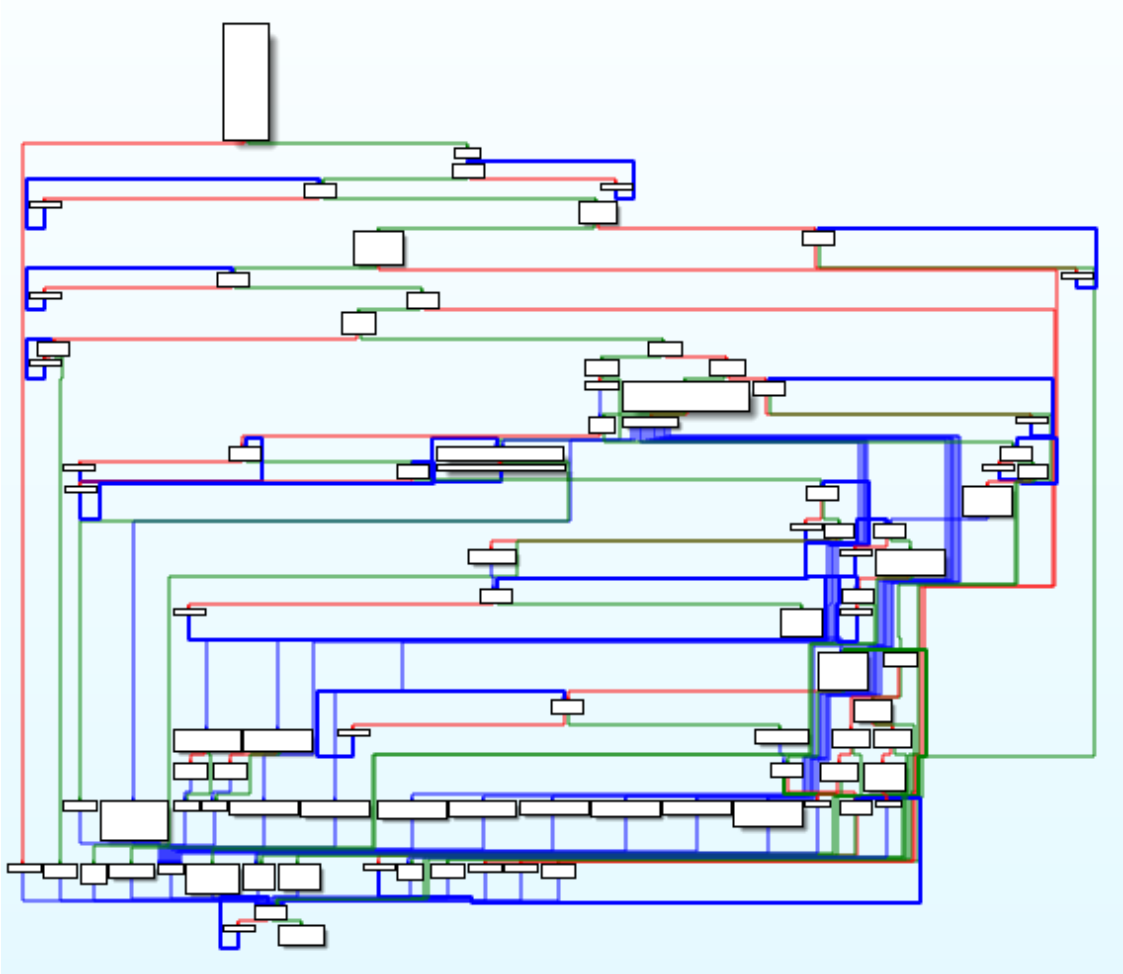
The unpacked sample hash of the file we ran in Triage:
`850ff92b7f3badda4bd4eca0a54fbdea410667db1ea27db8069337bf451078d1`



Overview

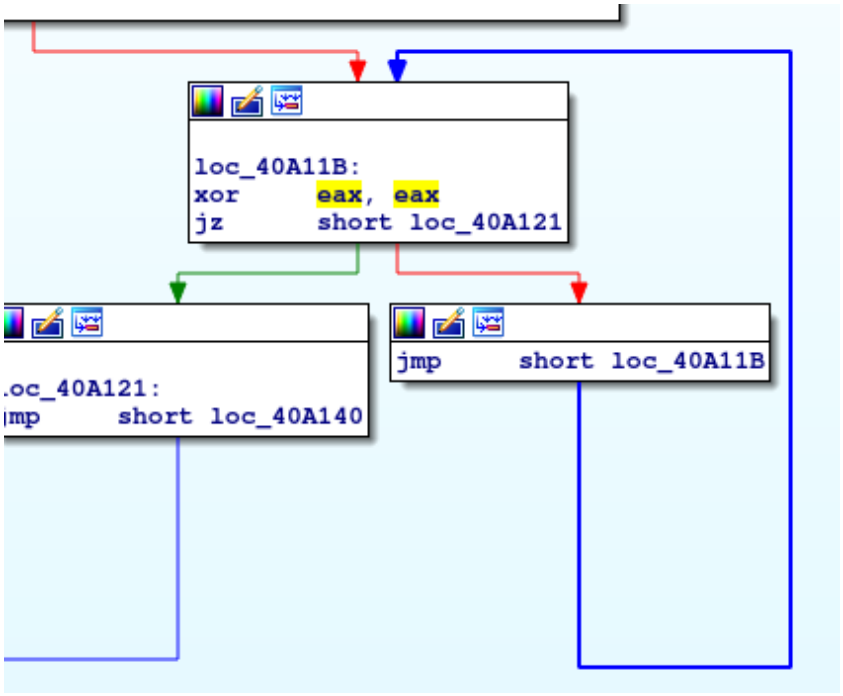
Obfuscation

Once the sample is unpacked, Qakbot itself also implements an obfuscation layer in its code. This obfuscation makes the analysis a bit harder. The flow graph of the main function is the following:



Qakbot's obfuscation

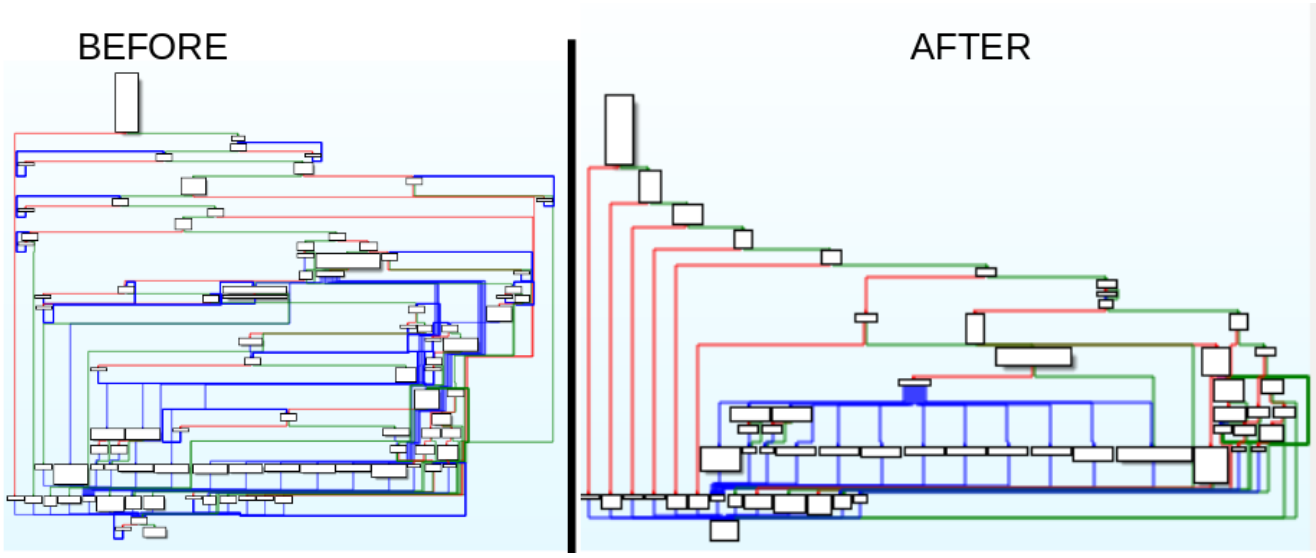
The obfuscation basically consists of adding unused loops with an empty body. Like the following:



Unused loops

As is shown in the image above, it does a “XOR EAX, EAX” operation and then decides to loop or not depending on the Z flag which is set with the previous instruction (so the loop will never happen). The goal of these small loops is to make a less comprehensive flow graph and to make the analysis harder. There are more than 600 loops like this throughout the code.

At Hatching, we implemented a tool qakbuscator.py to deobfuscate the code and make the analysis much easier. This tool is provided with this analysis to allow all researchers to use it.



Qakbot's deobfuscated

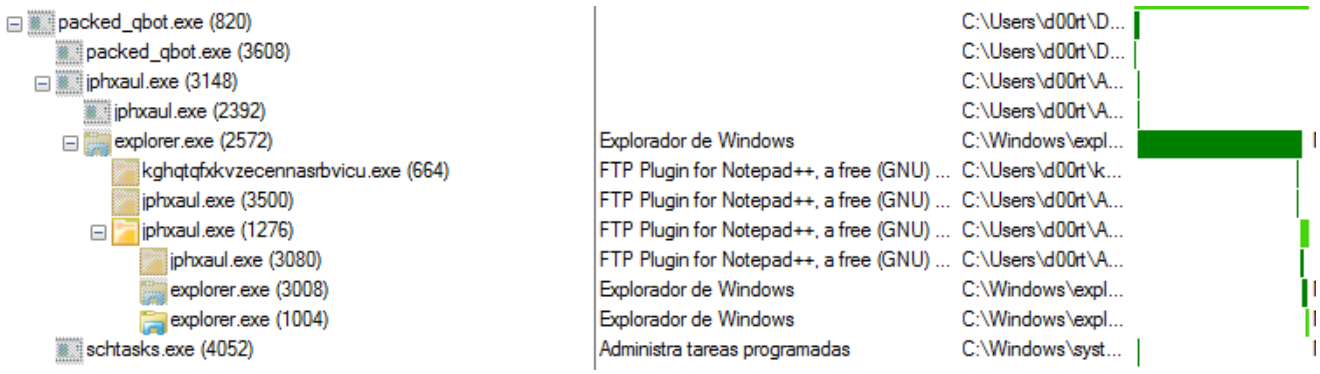
The DLLs that are in the Qakbot resources also have this obfuscation layer - you can use the script to deobfuscate them.

Behavioral analysis

The sample used to perform the behavioral analysis is the deobfuscated sample using our deobfuscator tool explained in the previous section.

SAMPLE: `3bd468d29868bb3f198530ef2426668efe30a8330bf3835a4f3a941d534ef2df`

This is how a process tree of a Qakbot infection looks like:



Process tree after Qakbot infection

Regardless of the input vector, the first time Qakbot runs it tries to install itself.

Anti-VM/Anti-analysis tricks

First of all, it checks if it is running in a virtualized environment or not. Qakbot executes itself with the option `"/C"`. Qakbot admits parameters, in this case the parameter `"/C"` is to make anti-VM and anti-sandbox checks like the following ones:

Reading from the virtual port in order to detect VMWare

```
signed int __usercall antivm_trick1@<eax>(int a1@<ebx>)
{
  unsigned __int32 v1; // eax
  signed int result; // eax

  v1 = __indword('VX');                         // Read virtual I/O port
  if ( a1 == 'VMXh' )
    result = 110;
  else
    result = 0;
  return result;
}
```

VMWare detection

Check the CPUID

```
  v15 = 0;
  lpString2 = 0;
  cpuid_(&String1);
  _EAX = 1;
  __asm { cpuid }
  v18 = _ECX;
  lpString2 = (LPCSTR)decrypt_string(0x181Au);
  if ( lpString2 )
  {
    if ( v18 == 1 && !lstrcmpiA(&String1, lpString2) )
      v17 = 1;
    w_freemem(&lpString2);
  }
  return 0;
}
```

CPUID check

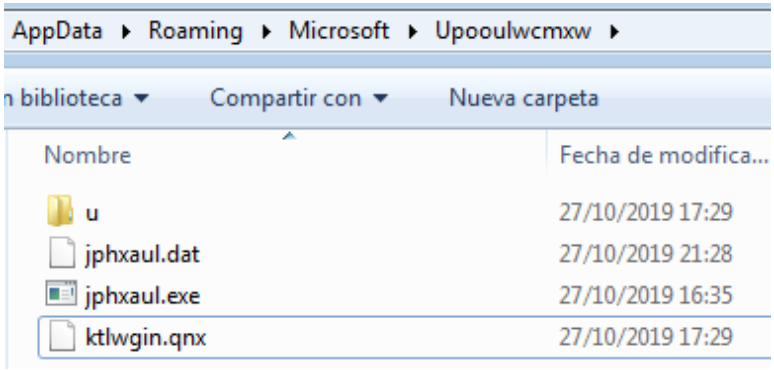
There are also other techniques used by Qakbot to know if it is running in an emulated environment like checking the sample name - in order to see if it is set to some default name like “sample.exe” or “malware.exe”; or checking running processes in order to detect any related to a virtual environments, anti-virus, debuggers etc.

Among the different options that Qakbot accepts we can find the following:

Accepted parameters	Description
/C	Anti-VM checks
/I [name]	Disable Windows SpyNey and delete scheduled task [name]
/P[file]	Decrypt [file] and load it
/Q	Set exit status to `0x6F`
/T	Sync related stuff
/V	Debug/Testing option
/W	Debug/Testing option
/i [name]	Install itself and delete scheduled task [name]
/s	Create service
/t	Send Window Message
/A [1] [2]	Unknown

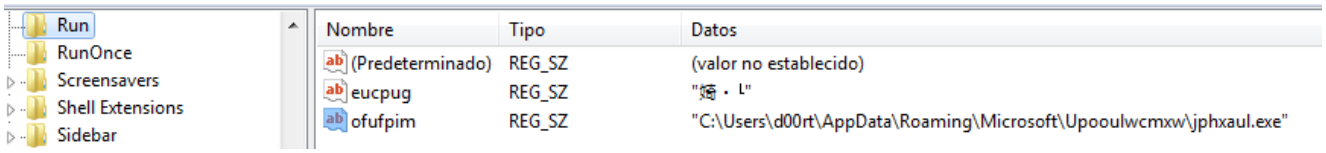
Installation

If a VM is detected it exits. Otherwise, it copies itself into `%APPDATA%` under a randomly generated folder with a randomly generated name. Those names are unique for each infected machine since they are created using some characteristics from infected host.



Copying to %APPDATA%

It also creates the following registry key in order to be run when the system reboots **"HKCU\Software\Microsoft\Windows\CurrentVersion\Run"**.



Run on system boot

Also, it drops a `.dat` file that has configuration information, like botnet name, timestamp, etc. This file contains encrypted data which is decrypted in memory during run time. Once this file is decrypted it looks like is shown in the image below.

```
11=2
1=17.23.33-27/10/2019
2=1572193413
50=1
39=127.23.255.255
45=47.202.98.230
46=443
49=1
43=1572193843
6=54.36.108.120:65400
38=1572207957
5=VgBCAE8AWABTAFYAUgA7ADIA
```

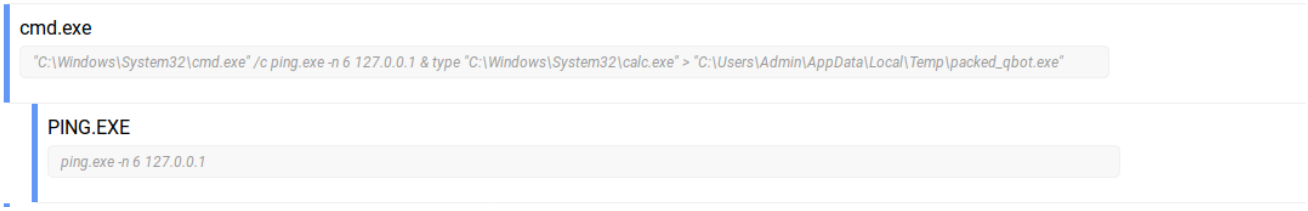
Decrypted file

The following table, from a blog post by the security researcher Vitaly Kremez (link) , shows the meanings of some of these config values:

Qakbot Config
11 = 2 (number of hardcoded C2)
1 = date of qbot install in HH:MM:ss-dd/mm/yyyy
2 = victim qbot install
45 = C2 IP

Qakbot Config
46 = C2 Port
39 = victim external IP
38 = last victim call to C2 (time in Unix)
43 = time of record ((time in Unix)
5 = victim network shares

Finally, the copied file is executed and the original file is overwritten with `calc.exe`. Some malware deletes the file directly, but Qakbot has decided to overwrite it with a legitimate binary. This way it doesn't leave traces.



Overwriting with legitimate binary

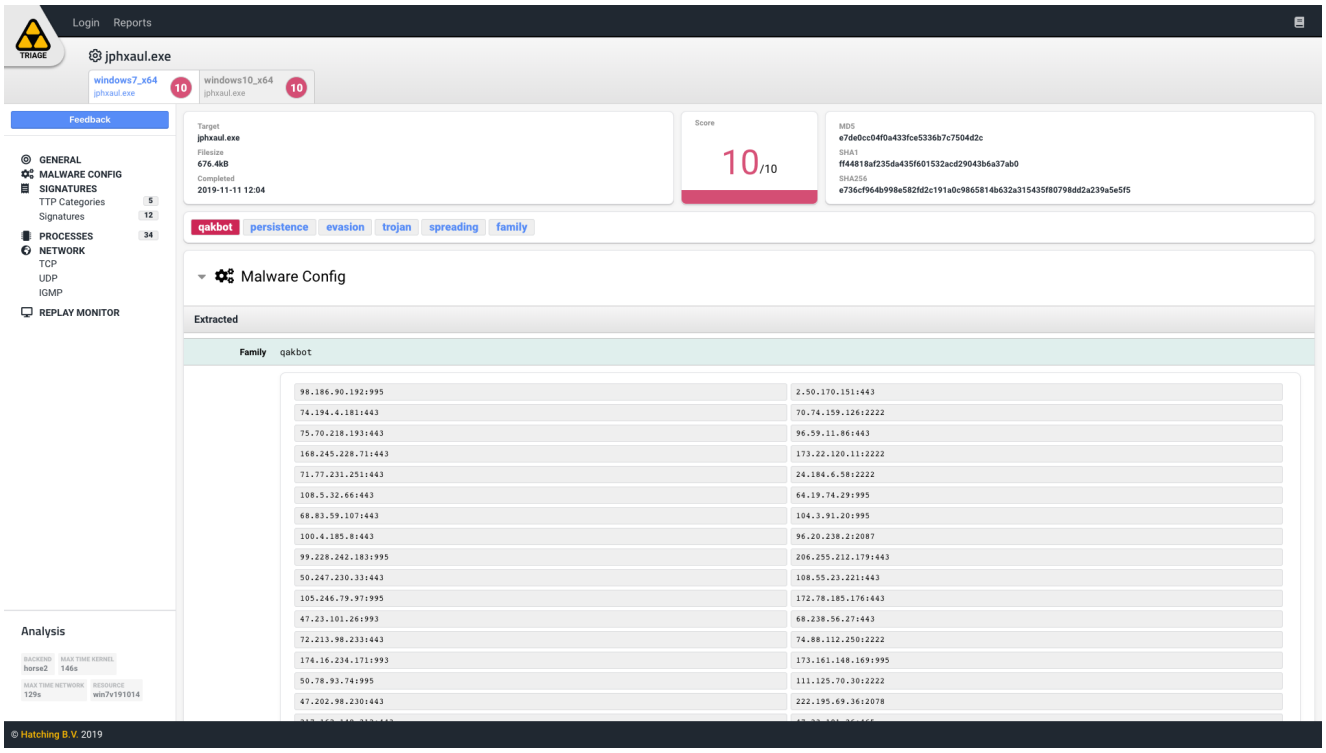
When Qakbot is installed, its behavior is different. In this case, it is going to create an instance of the `explorer.exe` process in order to inject itself into it.

Once injected into explorer, the main .dll is loaded. At this point, different things could happen since the communication with the control panel begins. As shown in the process tree above, the explorer process executes an update of Qakbot directly downloaded from the C&C. Also, it can exfiltrate data, or infect browsers in order to get banking information from the victim system.

Qakbot update sample: https://tria.ge/reports/191104-athqk1tjxn/task2

Triage

In Triage we've just added support for this family, meaning you can detect Qakbot as well as get its configuration directly after the analysis.



Qakbot in Triage

Samples

The Triage report for the sample that was used for this blog can be found (here).

Sample state	SHA256
Packed Qakbot	e736cf964b998e582fd2c191a0c9865814b632a315435f80798dd2a239a5e5f5
Qakbot	850ff92b7f3badda4bd4eca0a54fbdea410667db1ea27db8069337bf451078d1
Deobfuscated Qakbot	3bd468d29868bb3f198530ef2426668efe30a8330bf3835a4f3a941d534ef2df
Qakbot resource 1 (main.dll)	83273809a35ba26c2fb30cba58ba437004483ae754babad63c5d168113efa430
Deobfuscated Qakbot resource 1 (main.dll)	74f8907acfd070d2590895523433a8c85b5ef87f4e1a5ef7ccd356f5562b7a6b
Qakbot resource 2 (injects dll x86)	b7d9a462bd105193e998b6324f3343b84f11ceb21ab24e60e2580a26d95e4494

Sample state	SHA256
Qakbot resource 3 (injects dll x64)	8c7a43002ee6105fc37fcdfc00a192239639f7c08bf28e06ca1432551fe21b3f

Here is a list of related samples and their corresponding Triage reports.

SHA256	Triage Report
f614a06748251107a34fa7e44c7652fd88e61fd958df724455e14ec88040abf9	1.bin https://tria.ge/reports/191111-ypg95xvrwj_
7d4d207fb5258f504d3f9ef60d431332d1e7320d5849c0b0acf624612b01c8f0	2.bin https://tria.ge/reports/191111-mgrgp545yx_
357b4979324e2065adc8e6bd11cd7161f830250cae30f50fb13edd70fd2b506b	3.bin https://tria.ge/reports/191111-sbsq7xbqea_
29754f0caa9576eba6b9c351d20549e7e19216c6e72c2963da33450719a51277	4.bin https://tria.ge/reports/191111-57yf3bdh4j_
304a01a339d86ccbba7b1f671839624d446e6ea86474912bf976837df779bad2	5.bin https://tria.ge/reports/191111-38qmrk62q2_
d2f8a61e8cfc9a6c983fc40d2b7ac33e2a686872d0136dce2f66466c044f246c	6.bin https://tria.ge/reports/191111-p6cqne7cwn_
2b9ef4a9f47402d171eec28acadf3753cbb33c9bc6ec26d99aa060127a470e95	7.bin https://tria.ge/reports/191111-zl9l5y6lp2_
eb17935cf972d90be92c9b39fff8b3d760ecda78a6f602cb2b8bbaf3d87e6b61	8.bin https://tria.ge/reports/191111-7tn19rbh9x_
6b88260f4c4da4651a82bb62761cd23ee9ad6662a2a0abbec017e7193668397b	9.bin https://tria.ge/reports/191111-hb6qpeaars_
256967605423fea1e00368078eea1cdb52d391aa0091e0798db797ab337d1567	10.bin https://tria.ge/reports/191111-m8tm8zqbrs_

SHA256	Triage Report
13c2f4b6fb80500884a4ea9d2fe8077412 4f46ebfd80de3e1dfcfb9e167aee08	11.bin https://tria.ge/reports/191111-7cpggrpxts_

References