Bypassing Qakbot Anti-Analysis

lab52.io/blog/bypassing-qakbot-anti-analysis-tactics/

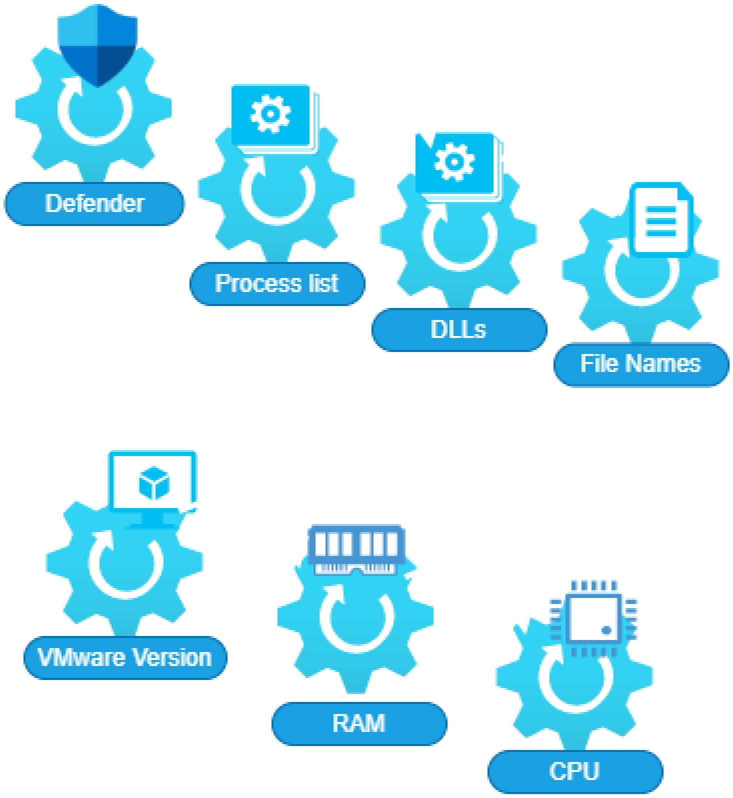
QakBot is a banking trojan that has been evolving since its first version was discovered in 2008. According to the 2022 report published by CISA, it was one of the most active variants in 2021, and during 2022 and so far in 2023 it has remained quite active. Taking a brief look at the latests news of QakBot it has been updating its tactics constantly, for example, using a Windows zero-day to avoid displaying the MoTW or the most recent one, using OneNote files to drop QakBot.

In this case we are particularly interested in the **anti-analysis techniques used by QakBot during the early stages of its execution**. These techniques can make malware analysis harder if they are not known, so learning to identify and bypass them is essential to get to see the malware's operation at its full potential. Furthermore, there are techniques that can replicate / adopt different types of malware, so knowking them opens the door to the study of different samples.

This article is structured according to the verifications carried out using the following sample, focusing of those aspects that are most remarkable.

md5	58e1c32eeb0130da19625e55ee48cf1e
sha1	00ae1c5066f67e5e71285de99bea8d8b67085743
sha256	f5ff6dbf5206cc2db098b41f5af14303f6dc43e36c5ec02604a50d5cfecf4790

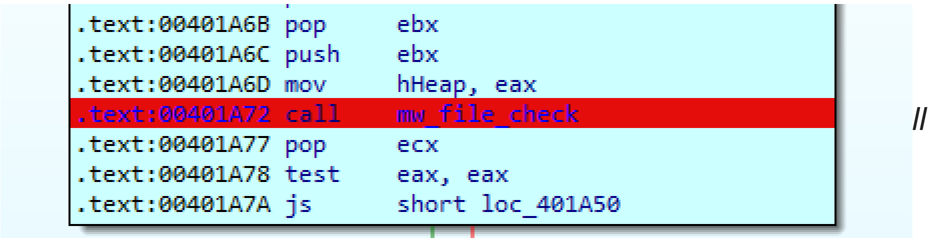
The following image summarizes the checks performed by QakBot before executing its payload. This article is structured following this chain of checks, which corresponds to the anti-analysis techniques used by the sample.



Anti-analyis checks performed by Qakbot

Windows Defender

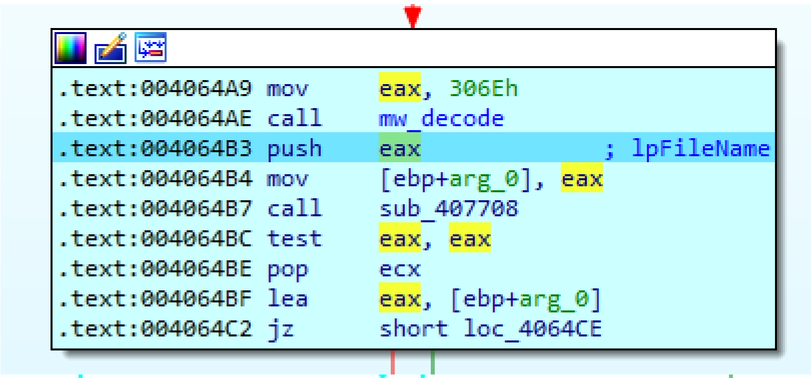
At the beginning of the program execution, QakBot will perform a first inevitable check since this sample is intended for Windows systems: to verify if Windows Defender is active. QakBot will perform this check by searching for **representative files**.



```
.text:00401A6B pop     ebx
.text:00401A6C push    ebx
.text:00401A6D mov     hHeap, eax
.text:00401A72 call    mw_file_check
.text:00401A77 pop     ecx
.text:00401A78 test    eax, eax
.text:00401A7A js      short loc_401A50
```

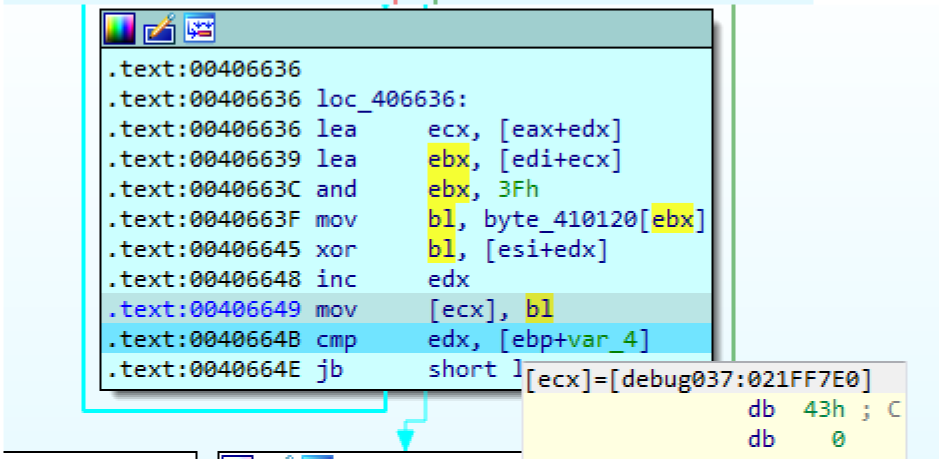
llustration 1 Call to the first check function

Inside the function we can observe a mov to the EAX register and then a call to a function used recurrently during the whole execution of the program. This function has been renamed to **mw_decode** since its objective is to decode text strings, taking the EAX register as parameter and performing the XOR operation.



```
.text:004064A9 mov     eax, 306Eh
.text:004064AE call    mw_decode
.text:004064B3 push    eax              ; lpFileName
.text:004064B4 mov     [ebp+arg_0], eax
.text:004064B7 call    sub_407708
.text:004064BC test    eax, eax
.text:004064BE pop     ecx
.text:004064BF lea     eax, [ebp+arg_0]
.text:004064C2 jz      short loc_4064CE
```

Illustration 2 Call to mw_decode

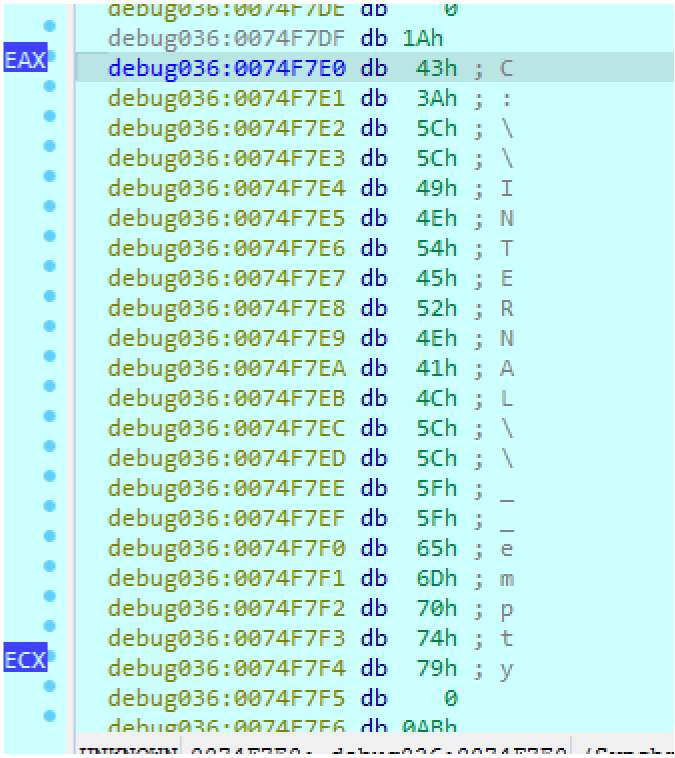


```
.text:00406636
.text:00406636 loc_406636:
.text:00406636 lea     ecx, [eax+edx]
.text:00406639 lea     ebx, [edi+ecx]
.text:0040663C and     ebx, 3Fh
.text:0040663F mov     bl, byte_410120[ebx]
.text:00406645 xor     bl, [esi+edx]
.text:00406648 inc     edx
.text:00406649 mov     [ecx], bl
.text:0040664B cmp     edx, [ebp+var_4]
.text:0040664E jb      short l[ecx]=[debug037:021FF7E0]
                                 db  43h ; C
                                 db    0
```

Illustration 3 mw_decode content

After performing all iterations of the loop, the decrypted string is visible when looking at the address of the ECX register. During all the checks performed by QakBot, this behavior can be seen.

In this case, the string refers to Windows Defender, since it is part of the empty files created by this utility.

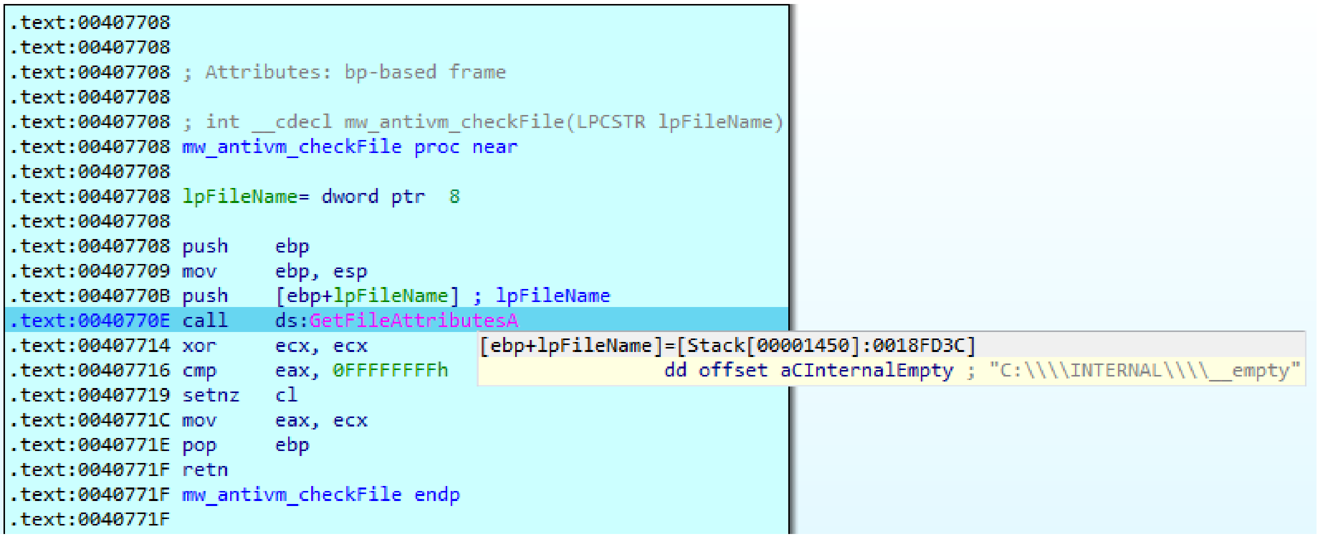


```
debug036:0074F7DE db    0
debug036:0074F7DF db 1Ah
debug036:0074F7E0 db  43h ; C
debug036:0074F7E1 db  3Ah ; :
debug036:0074F7E2 db  5Ch ; \
debug036:0074F7E3 db  5Ch ; \
debug036:0074F7E4 db  49h ; I
debug036:0074F7E5 db  4Eh ; N
debug036:0074F7E6 db  54h ; T
debug036:0074F7E7 db  45h ; E
debug036:0074F7E8 db  52h ; R
debug036:0074F7E9 db  4Eh ; N
debug036:0074F7EA db  41h ; A
debug036:0074F7EB db  4Ch ; L
debug036:0074F7EC db  5Ch ; \
debug036:0074F7ED db  5Ch ; \
debug036:0074F7EE db  5Fh ; _
debug036:0074F7EF db  5Fh ; _
debug036:0074F7F0 db  65h ; e
debug036:0074F7F1 db  6Dh ; m
debug036:0074F7F2 db  70h ; p
debug036:0074F7F3 db  74h ; t
debug036:0074F7F4 db  79h ; y
debug036:0074F7F5 db    0
debug036:0074F7F6 db 0ABh
```

Illustration 4 Decrypted string related to Windows Defender: C:\INTERNAL_empty

From here, taking the value **C:\INTERNAL_empty** as a parameter, it makes a call to the function GetFileAttributesA of the Windows API. Then, checks if this file already exists in the system.

This check is made to know if Windows Defender is present in the system, since the file C:\INTERNAL_empty is part of the files that Windows Defender creates.



```
.text:00407708
.text:00407708
.text:00407708 ; Attributes: bp-based frame
.text:00407708
.text:00407708 ; int __cdecl mw_antivm_checkFile(LPCSTR lpFileName)
.text:00407708 mw_antivm_checkFile proc near
.text:00407708
.text:00407708 lpFileName= dword ptr  8
.text:00407708
.text:00407708 push    ebp
.text:00407709 mov     ebp, esp
.text:0040770B push    [ebp+lpFileName] ; lpFileName
.text:0040770E call    ds:GetFileAttributesA
.text:00407714 xor     ecx, ecx
.text:00407716 cmp     eax, 0FFFFFFFFh
.text:00407719 setnz   cl
.text:0040771C mov     eax, ecx
.text:0040771E pop     ebp
.text:0040771F retn
.text:0040771F mw_antivm_checkFile endp
.text:0040771F
```

```
[ebp+lpFileName]=[Stack[00001450]:0018FD3C]
                dd offset aCInternalEmpty ; "C:\\\\INTERNAL\\\\__empty"
```

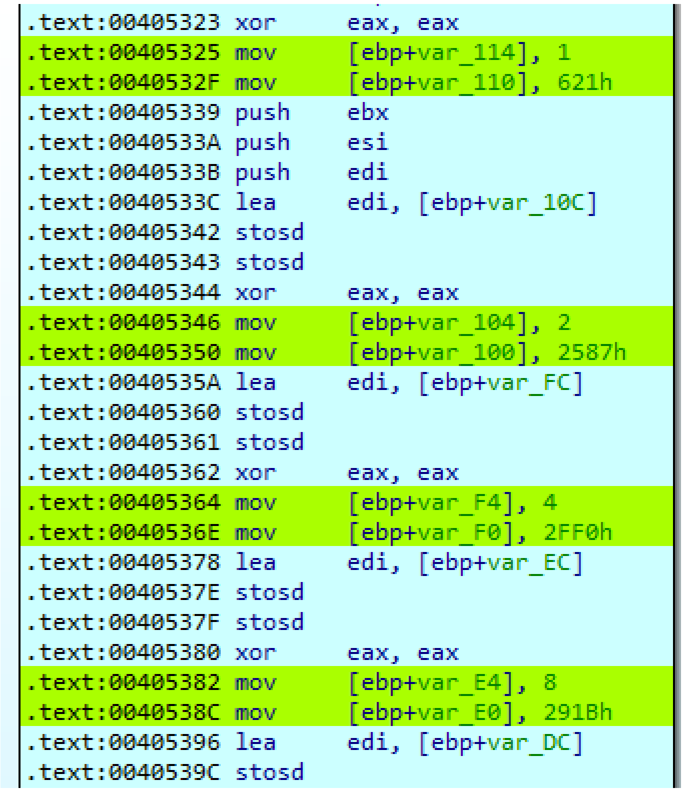
Illustration 5 Call to GetFileAttributesA with representative string

In case, after making the API call, it detects that the Windows Defender-related file is present in the system, the sample execution will be stopped. Otherwise, QakBot will continue with its execution, to continue with the checks.

Representative processes in execution

The next check is on the system processes. The main objective is to evaluate if there is any security application that can be used to detect or to anlyse malware, such as antivirus applications or applications used by reserarchers, or in sandboxes. In order to do so, Qakbot analyses the list of process and compares it with known representative names of processes.

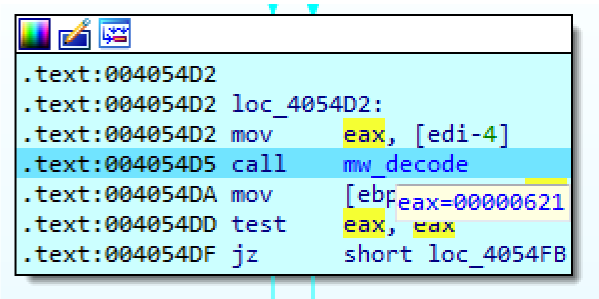
The first thing Qakbot will do is to load several hexadecimal values.



```
.text:00405323 xor      eax, eax
.text:00405325 mov      [ebp+var_114], 1
.text:0040532F mov      [ebp+var_110], 621h
.text:00405339 push     ebx
.text:0040533A push     esi
.text:0040533B push     edi
.text:0040533C lea      edi, [ebp+var_10C]
.text:00405342 stosd
.text:00405343 stosd
.text:00405344 xor      eax, eax
.text:00405346 mov      [ebp+var_104], 2
.text:00405350 mov      [ebp+var_100], 2587h
.text:0040535A lea      edi, [ebp+var_FC]
.text:00405360 stosd
.text:00405361 stosd
.text:00405362 xor      eax, eax
.text:00405364 mov      [ebp+var_F4], 4
.text:0040536E mov      [ebp+var_F0], 2FF0h
.text:00405378 lea      edi, [ebp+var_EC]
.text:0040537E stosd
.text:0040537F stosd
.text:00405380 xor      eax, eax
.text:00405382 mov      [ebp+var_E4], 8
.text:0040538C mov      [ebp+var_E0], 291Bh
.text:00405396 lea      edi, [ebp+var_DC]
.text:0040539C stosd
```

Illustration 6 Loading values in hexadecimal

As mentioned before, **mw_decode** will continue to be used to decode the strings used by the malware, so the hexadecimal value 0x621, seen before at the start of the function, is saved in the EAX register.

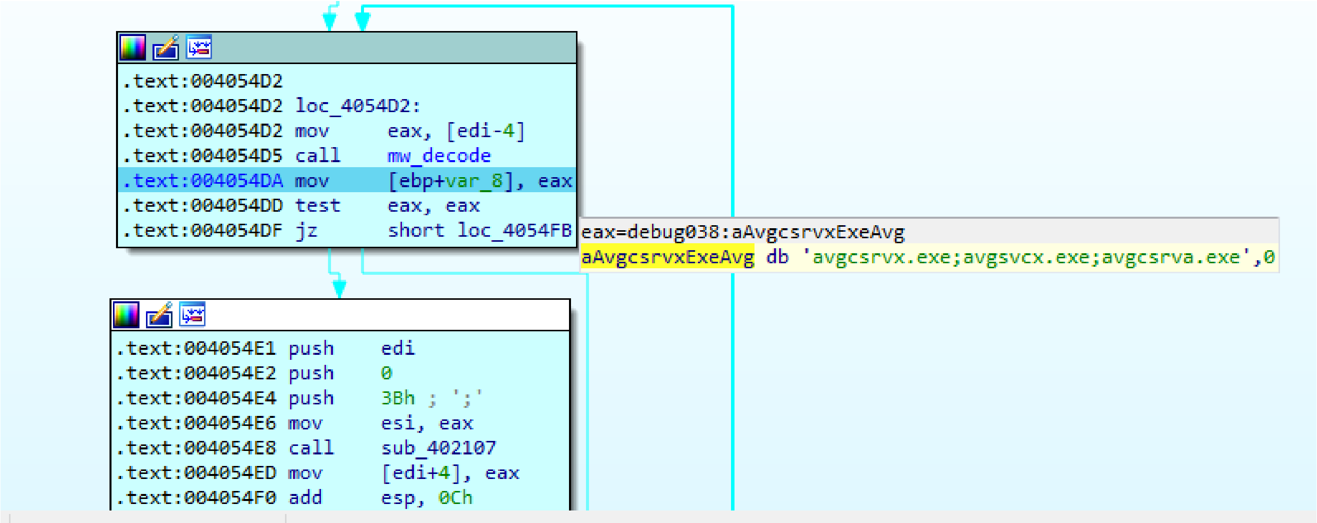


```
.text:004054D2
.text:004054D2 loc_4054D2:
.text:004054D2 mov      eax, [edi-4]
.text:004054D5 call     mw_decode
.text:004054DA mov      [ebp eax=00000621
.text:004054DD test     eax, eax
.text:004054DF jz       short loc_4054FB
```

Illustration 7 Call to mw_decode with value 0x621 entered as a parameter

After calling the function in charge of decrypting the strings, it will start a loop to obtain all the processes names for which it will check their existence in the system.

For example, the following image shows a list of processes subject to check with the names: avgcsrvx.exe, avgsvcx.exe and avgcsrva.exe. These are representative processes of AVG Free Antivirus.



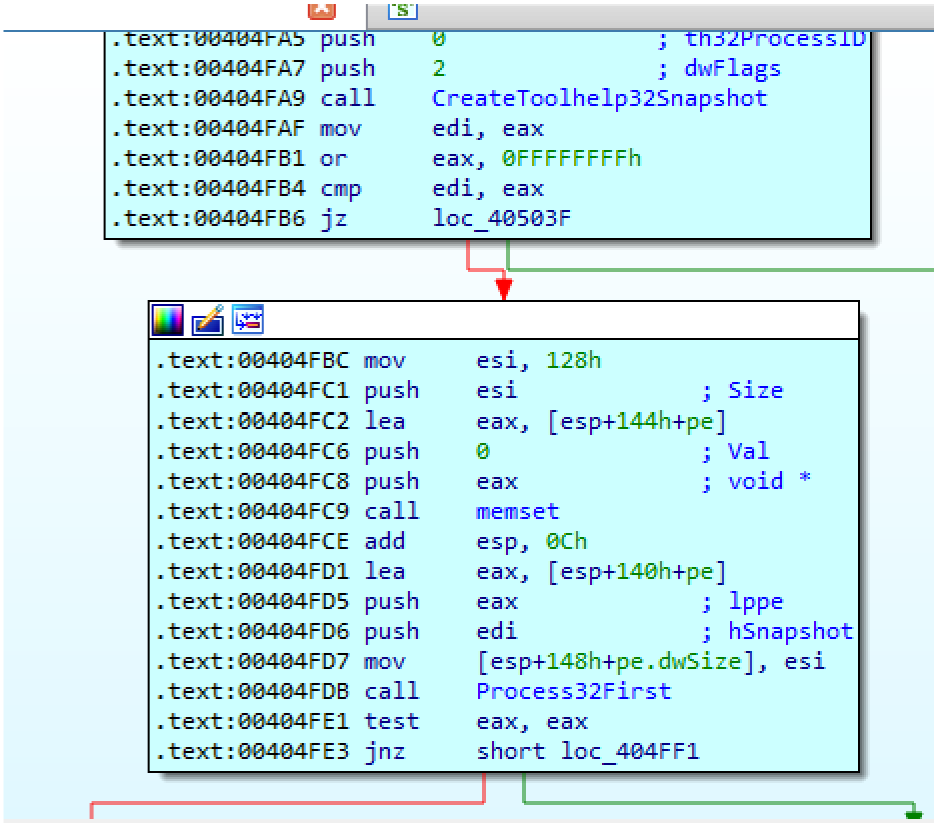
```
.text:004054D2
.text:004054D2 loc_4054D2:
.text:004054D2 mov     eax, [edi-4]
.text:004054D5 call    mw_decode
.text:004054DA mov     [ebp+var_8], eax
.text:004054DD test    eax, eax
.text:004054DF jz      short loc_4054FB
```

```
eax=debug038:aAvgcsrvxExeAvg
aAvgcsrvxExeAvg db 'avgcsrvx.exe;avgsvcx.exe;avgcsrva.exe',0
```

```
.text:004054E1 push    edi
.text:004054E2 push    0
.text:004054E4 push    3Bh ; ';'
.text:004054E6 mov     esi, eax
.text:004054E8 call    sub_402107
.text:004054ED mov     [edi+4], eax
.text:004054F0 add     esp, 0Ch
```

Illustration 8 Some names of processes that will be checked

Once it has the strings to check, to obtain the first running process in the system it proceeds with calls to the CreateToolhelp32Snapshot and Process32First functions.

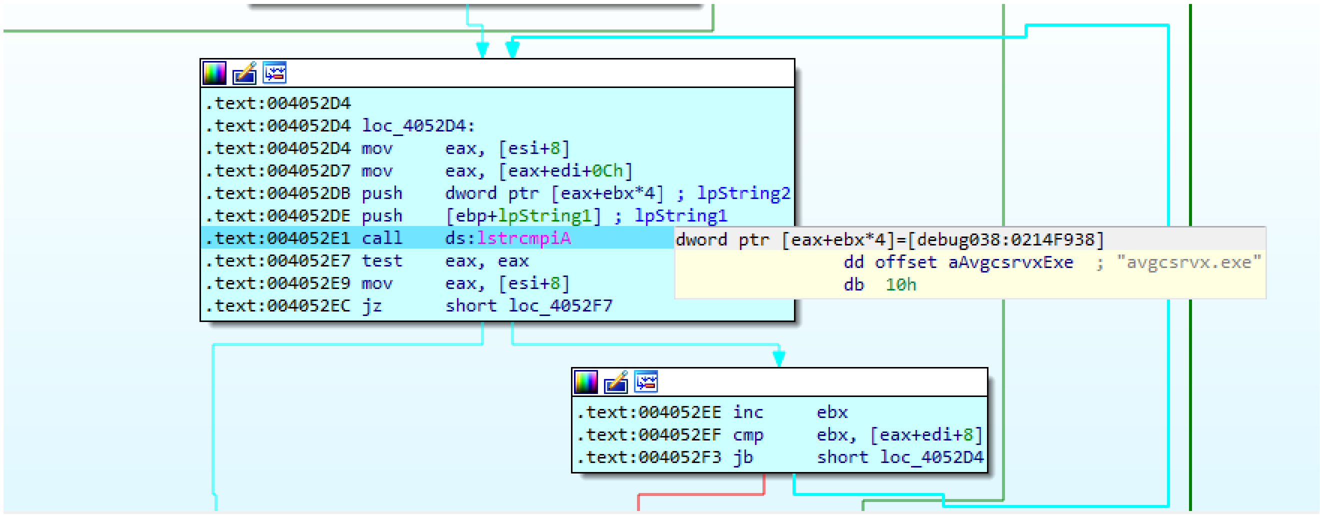


```
.text:00404FA5 push    0                  ; th32ProcessID
.text:00404FA7 push    2                  ; dwFlags
.text:00404FA9 call    CreateToolhelp32Snapshot
.text:00404FAF mov     edi, eax
.text:00404FB1 or      eax, 0FFFFFFFFh
.text:00404FB4 cmp     edi, eax
.text:00404FB6 jz      loc_40503F
```

```
.text:00404FBC mov     esi, 128h
.text:00404FC1 push    esi                ; Size
.text:00404FC2 lea     eax, [esp+144h+pe]
.text:00404FC6 push    0                  ; Val
.text:00404FC8 push    eax                ; void *
.text:00404FC9 call    memset
.text:00404FCE add     esp, 0Ch
.text:00404FD1 lea     eax, [esp+140h+pe]
.text:00404FD5 push    eax                ; lppe
.text:00404FD6 push    edi                ; hSnapshot
.text:00404FD7 mov     [esp+148h+pe.dwSize], esi
.text:00404FDB call    Process32First
.text:00404FE1 test    eax, eax
.text:00404FE3 jnz     short loc_404FF1
```

Illustration 9 Calls to CreatToolhelp32Snapshot and Process32First

Qakbot then checks if the processes names obtained above match any currently active process in the system.



```
.text:004052D4
.text:004052D4 loc_4052D4:
.text:004052D4 mov     eax, [esi+8]
.text:004052D7 mov     eax, [eax+edi+0Ch]
.text:004052DB push    dword ptr [eax+ebx*4] ; lpString2
.text:004052DE push    [ebp+lpString1] ; lpString1
.text:004052E1 call    ds:lstrcmpiA
.text:004052E7 test    eax, eax
.text:004052E9 mov     eax, [esi+8]
.text:004052EC jz      short loc_4052F7
```

```
dword ptr [eax+ebx*4]=[debug038:0214F938]
                dd offset aAvgcsrvxExe  ; "avgcsrvx.exe"
                db  10h
```

```
.text:004052EE inc     ebx
.text:004052EF cmp     ebx, [eax+edi+8]
.text:004052F3 jb      short loc_4052D4
```

Illustration 10 Iteration to compare processes names

It will perform this operation with all the processes, if any of them is equal to the ones it has defined, it will terminate the execution. In particular, the following processes names have been found to be subject to analysis. They are ordered with relation to the type of application in the following table.

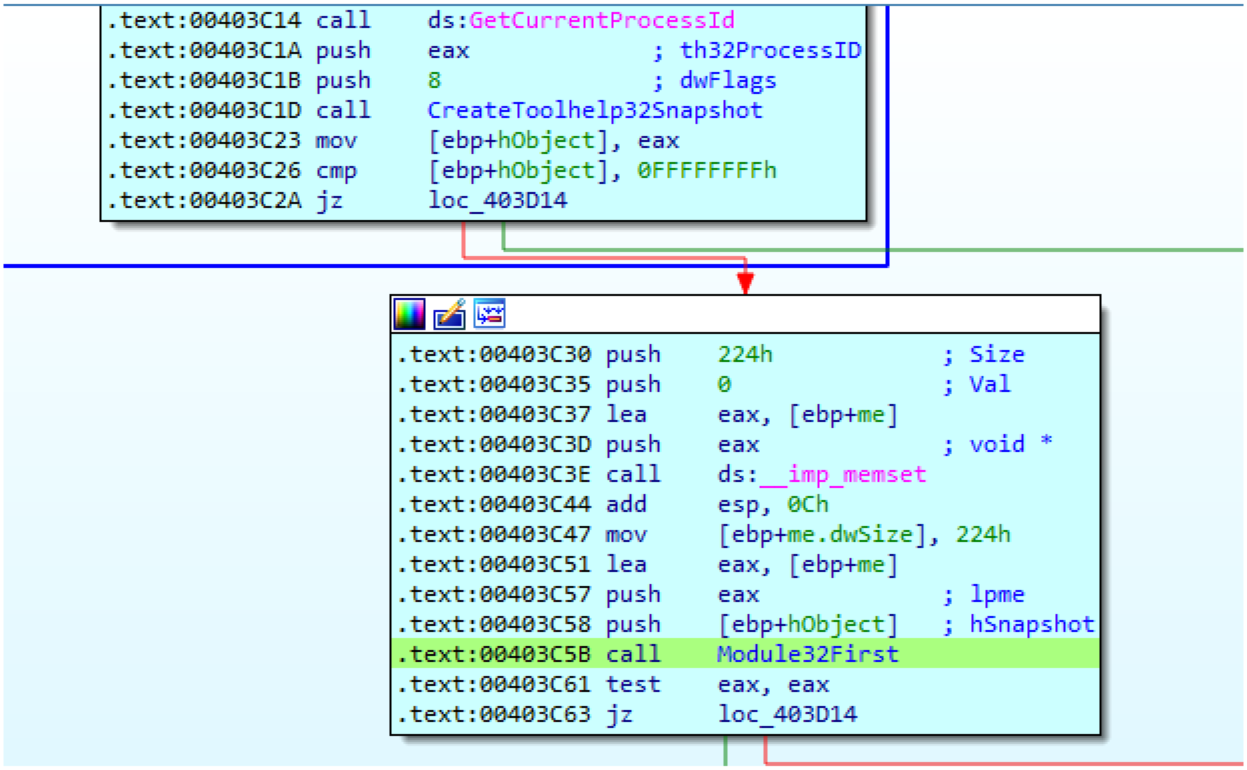
Type	Name of process
Antivirus	Avgcsrvx.exe Avgsvcx.exe Avgcsrva.exe ccSvcHst.exe MsMpEng.exe mcshield.exe Avp.exe kavtray.exe Egui.exe ekrn.exe Bdagent.exe Vsserv.exe vsservppl.exe AvastSvc.exe coreServiceShell.exe PccNTMon.exe NTRTScan.exe SAVAdminService.exe SavService.exe fshoster32.exe WRSA.exe Vkise.exe Isesrv.exe cmdagent.exe ByteFence.exe MBAMService.exe mbamgui.exe fmon.exe Dwengine.exe Dwarkdaemon.exe dwwatcher.exe bds-vision-agent-nai.exe bds-vision-apis.exe bds-vision-agent-app.exe
Malware Analysis	Fiddler.exe lordpe.exe regshot.exe Autoruns.exe Dsniff.exe HashMyFiles.exe ProcessHacker.exe Procmon.exe Procmon64.exe Netmon.exe pr0c3xp.exe ProcessHacker.exe CFF Explorer.exe dumpcap.exe Wireshark.exe idaq.exe Idaq64.exe ResourceHacker.exe MultiAnalysis_v1.0.294.exe x32dbg.exe Tcpview.exe OLLYDBG.EXE windbg.exe samp1e.exe sample.exe runsample.exe
Virtualization Environments	VBoxTray.exe vmtoolsd.exe vm3dservice.exe VGAuthService.exe TPAutoConnect.exe vmacthlp.exe VBoxTray.exe VboxService.exe

As anticipated, this point groups together checks involving both user protection and analysis tools. It is to be expected that successive versions of QakBot will update the previous list.

If QakBot does not find any process with the above names, it continues its execution with the next check.

Modules

If it passes the above check, it will make use of the Module32First and Module32Next APIs to get all the modules for each of the processes in the system.

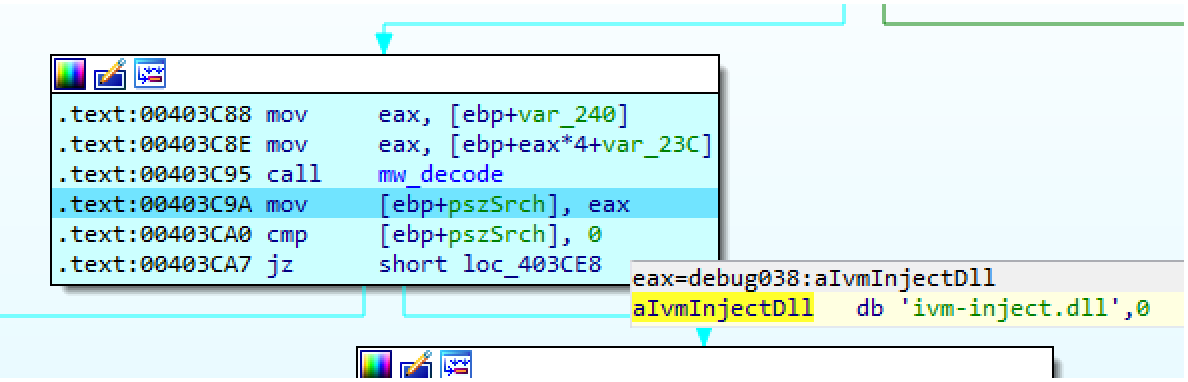


```
.text:00403C14 call    ds:GetCurrentProcessId
.text:00403C1A push    eax             ; th32ProcessID
.text:00403C1B push    8               ; dwFlags
.text:00403C1D call    CreateToolhelp32Snapshot
.text:00403C23 mov     [ebp+hObject], eax
.text:00403C26 cmp     [ebp+hObject], 0FFFFFFFFh
.text:00403C2A jz      loc_403D14
```

```
.text:00403C30 push    224h            ; Size
.text:00403C35 push    0               ; Val
.text:00403C37 lea     eax, [ebp+me]
.text:00403C3D push    eax             ; void *
.text:00403C3E call    ds:__imp_memset
.text:00403C44 add     esp, 0Ch
.text:00403C47 mov     [ebp+me.dwSize], 224h
.text:00403C51 lea     eax, [ebp+me]
.text:00403C57 push    eax             ; lpme
.text:00403C58 push    [ebp+hObject]   ; hSnapshot
.text:00403C5B call    Module32First
.text:00403C61 test    eax, eax
.text:00403C63 jz      loc_403D14
```

Illustration 11 Use of Module32First

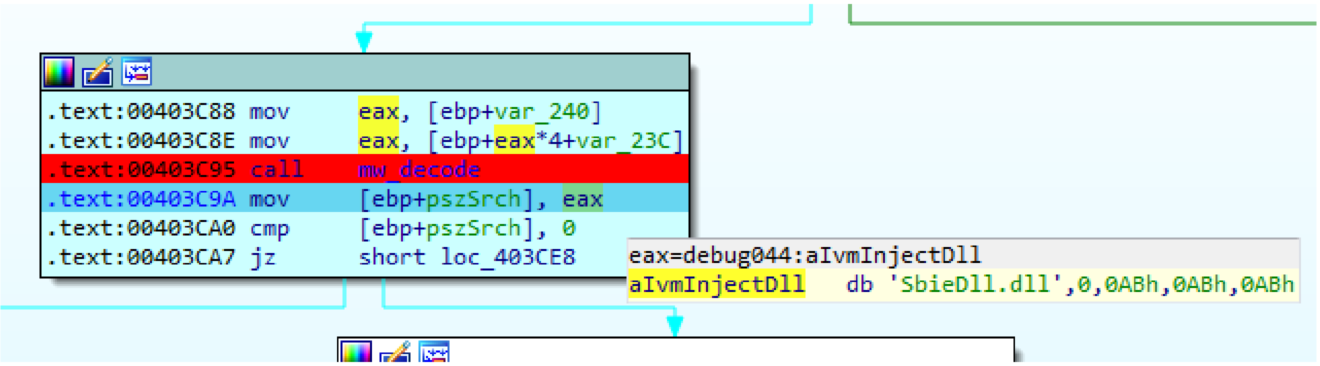
If any of the system modules contain the string **ivm-inject.dll** or **SbieDll.dll** it will terminate its execution.



```
.text:00403C88 mov     eax, [ebp+var_240]
.text:00403C8E mov     eax, [ebp+eax*4+var_23C]
.text:00403C95 call    mw_decode
.text:00403C9A mov     [ebp+pszSrch], eax
.text:00403CA0 cmp     [ebp+pszSrch], 0
.text:00403CA7 jz      short loc_403CE8
```

```
eax=debug038:aIvmInjectDll
aIvmInjectDll    db 'ivm-inject.dll',0
```

Illustration 12 String ivm-inject.dll



```
.text:00403C88 mov     eax, [ebp+var_240]
.text:00403C8E mov     eax, [ebp+eax*4+var_23C]
.text:00403C95 call    mw_decode
.text:00403C9A mov     [ebp+pszSrch], eax
.text:00403CA0 cmp     [ebp+pszSrch], 0
.text:00403CA7 jz      short loc_403CE8
```

```
eax=debug044:aIvmInjectDll
aIvmInjectDll    db 'SbieDll.dll',0,0ABh,0ABh,0ABh
```

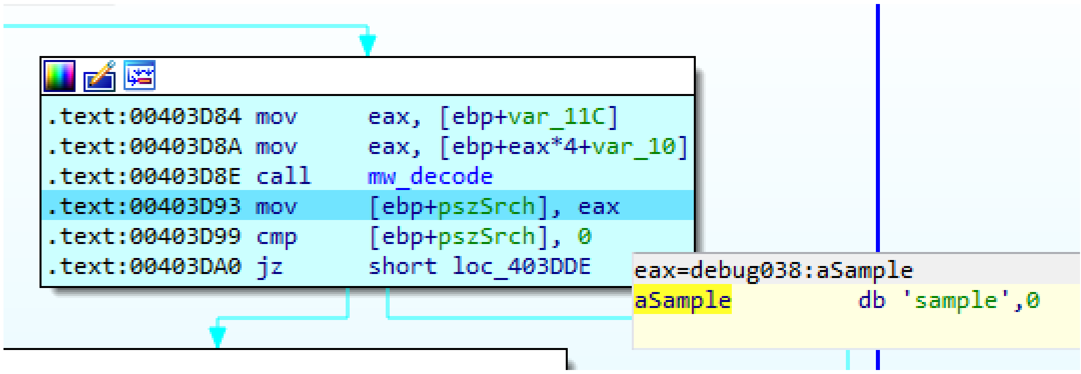
Illustration 13 String SbieDll.dll

The names of the DLLs have been identified as part of the Sandboxie program, used to run programs in isolated environments. If any program uses these modules, it could be an indication that this analysis tool is on the system, and QakBot would stop its execution.

It is worth noting, for example, that the Sandboxie-Plus version could incorporate utilities to hide the presence of SbieDll.dll.

Characteristic names given to the sample

Analysts have some habits that QakBot will check. In this case, it will check if in the name of the binary itself (the malware) is present any of the characteristic strings that could be used by analysts to rename the sample, before its execution, such as “sample”, “mlwr_sm”, “artifact.exe”. Again, these strings will be observed after the execution of mw_decode.



```
.text:00403D84 mov     eax, [ebp+var_11C]
.text:00403D8A mov     eax, [ebp+eax*4+var_10]
.text:00403D8E call    mw_decode
.text:00403D93 mov     [ebp+pszSrch], eax
.text:00403D99 cmp     [ebp+pszSrch], 0
.text:00403DA0 jz      short loc_403DDE
```

```
eax=debug038:aSample
aSample         db 'sample',0
```

Illustration 14

String sample

If any of these strings are found as part of the filename, it will stop the execution of the program. In addition, this check is not case-sensitive, i.e. it does not distinguish between upper and lower case.

It is curious, for example, that it does not also check that the name of the binary may correspond to a sha256 pattern, since samples downloaded from platforms such as VirusTotal or other systems retain in their name the hash of the binary, which the analyst may or may not rename.

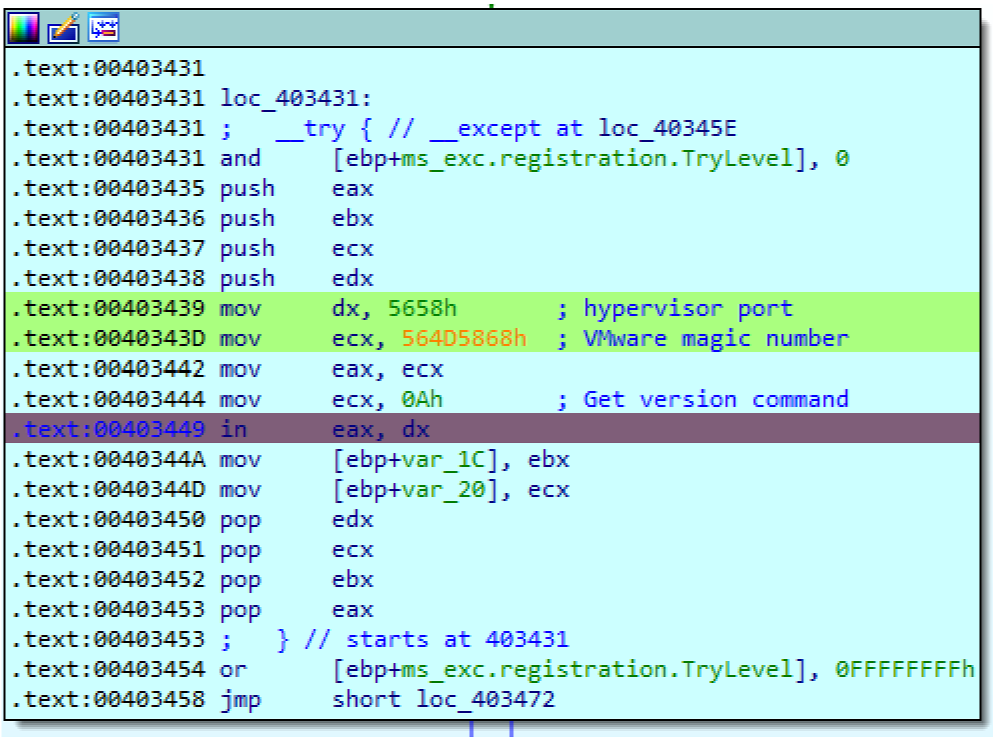
Anti-VM Techniques

QakBot performs specific checks to determine if it is running in a virtual environment. These checks are described below.

VMware version

The malware will evaluate whether it is running within a VMWare virtual machine. To do that, QakBot will make use of a special VMWare I/O port. In particular, the verification at this point focuses on the port used by the official VMWare tools to perform the communications. VMWare uses I/O port **0x5658** to communicate internally with the deployed virtual machines, so the first step executed by QakBot is to save in the **DX** register the value corresponding to

the I/O port. After this step, the value **0x564D5868** is stored in EAX. This value corresponds to the string ‘**VMXh**‘, which is the VMWare magic number.
Finally, the internal VMWare command is specified. In this case **0x0A** is used, which corresponds to the command to obtain information from VMWare.



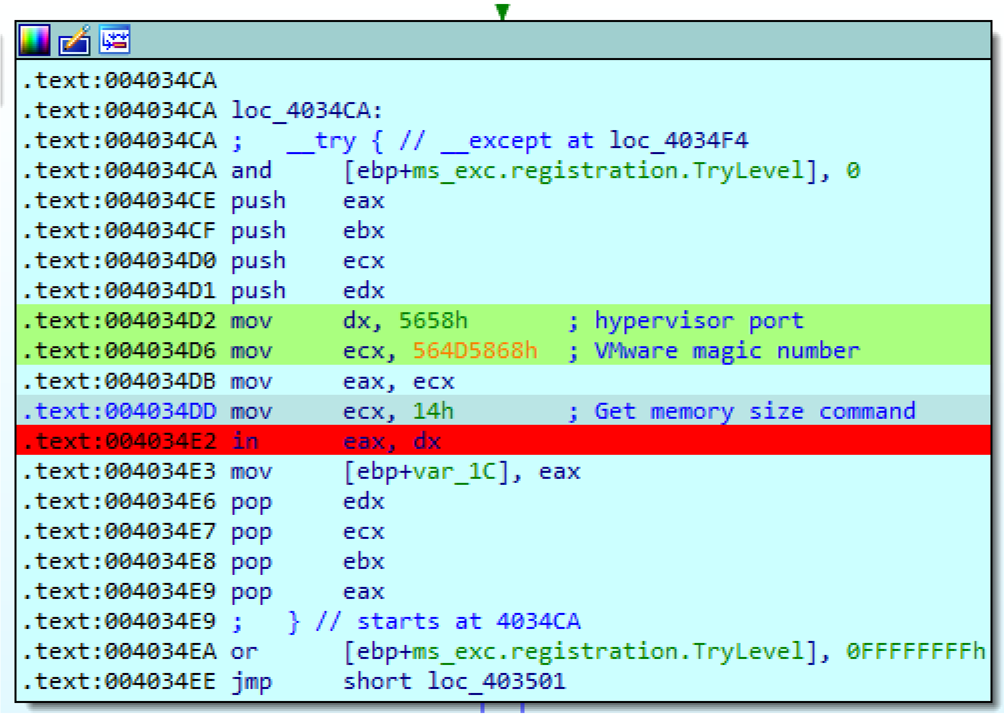
```
.text:00403431
.text:00403431 loc_403431:
.text:00403431 ;   __try { // __except at loc_40345E
.text:00403431 and     [ebp+ms_exc.registration.TryLevel], 0
.text:00403435 push    eax
.text:00403436 push    ebx
.text:00403437 push    ecx
.text:00403438 push    edx
.text:00403439 mov     dx, 5658h        ; hypervisor port
.text:0040343D mov     ecx, 564D5868h   ; VMware magic number
.text:00403442 mov     eax, ecx
.text:00403444 mov     ecx, 0Ah         ; Get version command
.text:00403449 in      eax, dx
.text:0040344A mov     [ebp+var_1C], ebx
.text:0040344D mov     [ebp+var_20], ecx
.text:00403450 pop     edx
.text:00403451 pop     ecx
.text:00403452 pop     ebx
.text:00403453 pop     eax
.text:00403453 ;   } // starts at 403431
.text:00403454 or      [ebp+ms_exc.registration.TryLevel], 0FFFFFFFFh
.text:00403458 jmp     short loc_403472
```

Illustration 15 Check code: VMware

After performing the “**in**” instruction, the EBX and ECX registers will be modified.
In the EBX register the magic number of Vmware will be written, while in the ECX register the value corresponding to VMWare products will be stored. The following values are known:
– 01h = Express
– 02h = ESX Server
– 03h = GSX Server
– 04h = Workstation

RAM memory size

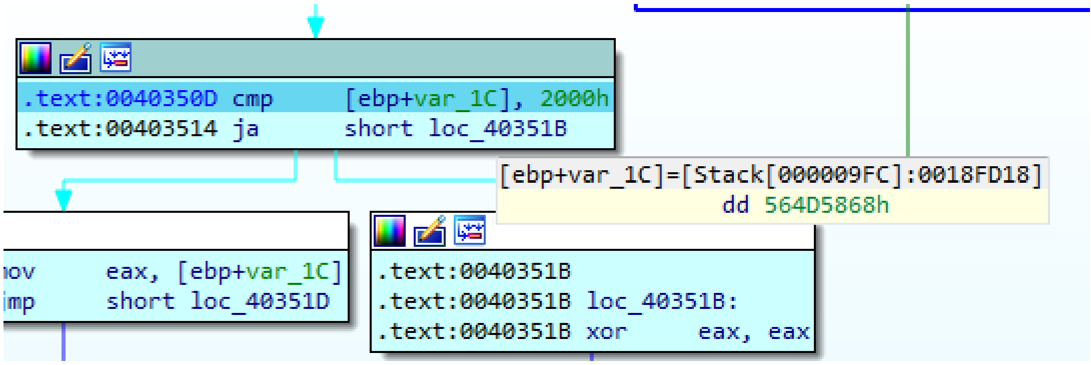
If the previous check is passed, QakBot proceeds to obtain the size of the memory allocated to the system. This check is performed, like the previous check, using the I/O port, but in this case it uses the value **0x14** as the command. The resulting value will be stored in the EAX register, to later perform a move to EBP. It is important to note that, if the previous check does not detect that it is running in a VM and passes to this check, here it makes again use of the I/O port, which would be a contradiction.



```
.text:004034CA
.text:004034CA loc_4034CA:
.text:004034CA ;   __try { // __except at loc_4034F4
.text:004034CA and     [ebp+ms_exc.registration.TryLevel], 0
.text:004034CE push    eax
.text:004034CF push    ebx
.text:004034D0 push    ecx
.text:004034D1 push    edx
.text:004034D2 mov     dx, 5658h        ; hypervisor port
.text:004034D6 mov     ecx, 564D5868h   ; VMware magic number
.text:004034DB mov     eax, ecx
.text:004034DD mov     ecx, 14h         ; Get memory size command
.text:004034E2 in      eax, dx
.text:004034E3 mov     [ebp+var_1C], eax
.text:004034E6 pop     edx
.text:004034E7 pop     ecx
.text:004034E8 pop     ebx
.text:004034E9 pop     eax
.text:004034E9 ;   } // starts at 4034CA
.text:004034EA or      [ebp+ms_exc.registration.TryLevel], 0FFFFFFFFh
.text:004034EE jmp     short loc_403501
```

Illustration 16 Check code: PC memory

QakBot will decide if it is inside a VM at this point by comparing the value stored in the EBP register, which contains the size of the machine's RAM, against the value 0x2000, which is equivalent to 8192 in decimal. It means that, if the machine has less than 8 Gbytes of RAM, QakBot will decide that it is in a virtual machine.



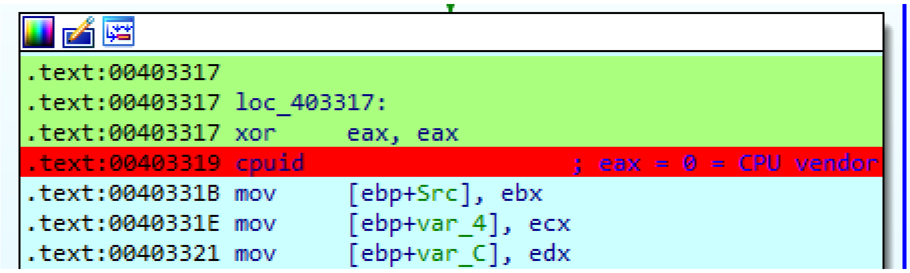
```
.text:0040350D cmp     [ebp+var_1C], 2000h
.text:00403514 ja      short loc_40351B

[ebp+var_1C]=[Stack[000009FC]:0018FD18]
             dd 564D5868h

ov      eax, [ebp+var_1C]
mp      short loc_40351D

.text:0040351B
.text:0040351B loc_40351B:
.text:0040351B xor     eax, eax
```

Illustration 17 RAM size check

Note that QakBot only performs this check if it has previously detected that it is not running in a virtual machine using the VMWare I/O port. However, it is curious that the malware uses the VMWare I/O port again during this check, as it should not be able to obtain a valid RAM value when it is not running in a VMWare environment.

CPU Characteristics

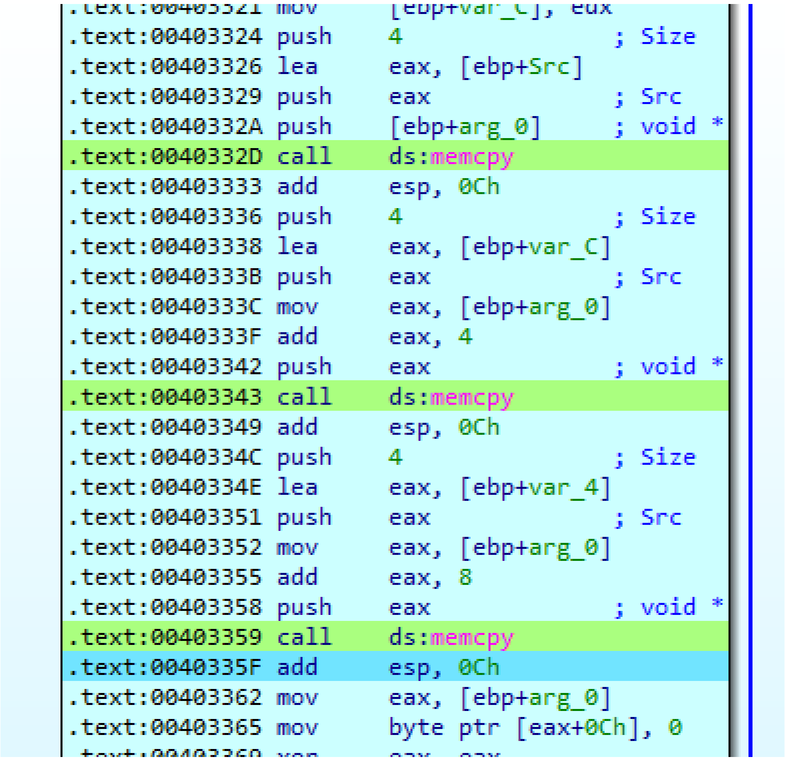
For the last check QakBot will make use of the **cpuid** instruction. This instruction returns different values based on the value stored in EAX. In this case an EAX xor operation is performed on EAX, which results in a 0 always.



```
.text:00403317
.text:00403317 loc_403317:
.text:00403317 xor     eax, eax
.text:00403319 cpuid                      ; eax = 0 = CPU vendor
.text:0040331B mov     [ebp+Src], ebx
.text:0040331E mov     [ebp+var_4], ecx
.text:00403321 mov     [ebp+var_C], edx
```

Illustration 18 cpuid instruction

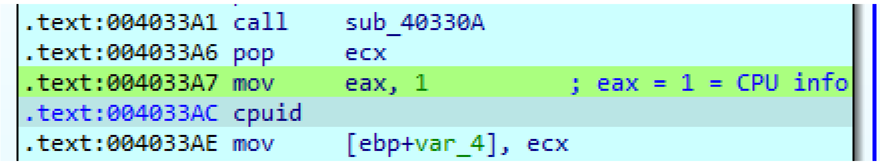
When cpuid has a 0 as EAX value, it returns the CPU manufacturer, which is precisely the target pursued by the malware in this step. Then, it performs three memcpy operations to reorder the resulting string.



```
.text:00403321 mov     [ebp+var_C], edx
.text:00403324 push    4               ; Size
.text:00403326 lea     eax, [ebp+Src]
.text:00403329 push    eax             ; Src
.text:0040332A push    [ebp+arg_0]     ; void *
.text:0040332D call    ds:memcpy
.text:00403333 add     esp, 0Ch
.text:00403336 push    4               ; Size
.text:00403338 lea     eax, [ebp+var_C]
.text:0040333B push    eax             ; Src
.text:0040333C mov     eax, [ebp+arg_0]
.text:0040333F add     eax, 4
.text:00403342 push    eax             ; void *
.text:00403343 call    ds:memcpy
.text:00403349 add     esp, 0Ch
.text:0040334C push    4               ; Size
.text:0040334E lea     eax, [ebp+var_4]
.text:00403351 push    eax             ; Src
.text:00403352 mov     eax, [ebp+arg_0]
.text:00403355 add     eax, 8
.text:00403358 push    eax             ; void *
.text:00403359 call    ds:memcpy
.text:0040335F add     esp, 0Ch
.text:00403362 mov     eax, [ebp+arg_0]
.text:00403365 mov     byte ptr [eax+0Ch], 0
.text:00403369 xor     eax, eax
```

Illustration 19 memcpy instructions

After the operations the final string will correspond to the CPU manufacturer of the system. Once it has obtained this data, it moves the value 1 to EAX to call cpuid again. When cpuid is called with EAX value 1, this operation returns the processor information.

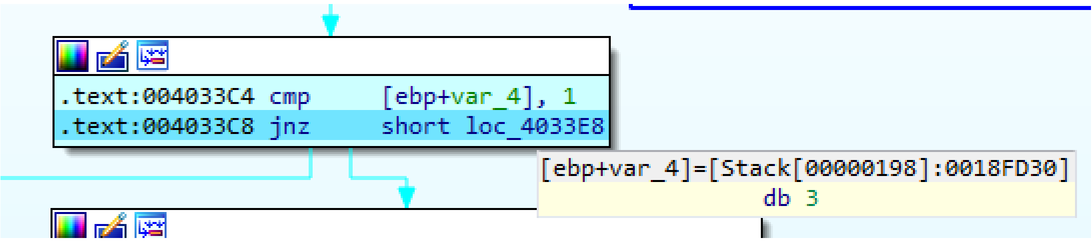


```
.text:004033A1 call    sub_40330A
.text:004033A6 pop     ecx
.text:004033A7 mov     eax, 1          ; eax = 1 = CPU info
.text:004033AC cpuid
.text:004033AE mov     [ebp+var_4], ecx
```

Illustration 20 Processor information request

The information received in ECX after the execution of the cpuid instruction will always end with a value of 0 in the case of a physical machine, but in the case of a virtual machine it will be 1.

It should be noted at this point that for both VMware and VirtualBox system execution a value of 3 is received, so that for both platforms it would be possible to bypass this check.



```
.text:004033C4 cmp     [ebp+var_4], 1
.text:004033C8 jnz     short loc_4033E8
```

```
[ebp+var_4]=[Stack[00000198]:0018FD30]
            db 3
```

Illustration 21

Return value for VMware

Conclusions

This analysis has focused on the anti-analysis capabilities employed by QakBot in order to help overcome these obstacles before starting the analysis. The anti-analysis techniques detailed here can be used by different malware, so it is very important to be aware of them. However, it is important to note that this analysis is based on a specific sample of QakBot malware, and there are various other families of malware that employ different anti-analysis techniques that have not been covered in this report. These techniques may be explored in future posts.

Regarding the analysis performed, it is also interesting to highlight the checks made by Qakbot to detect if it is under a virtualized environment, as these checks only applyies to VMWare software when using VMWare’s own I/O port, and searching by its unique magic number.

References

VMware Backdoor I/O Port

CPUID instruction reference

Windows Defender DB dump and VDLL’s