Deep Analysis of a QBot Campaign – Part I | FortiGuard Labs

By Xiaopeng Zhang

Published: 2020-06-11 · Archived: 2026-04-05 22:49:00 UTC

FortiGuard Labs Threat Research Report

Affected platforms: Microsoft Windows
Impacted parties: Windows Users
Impact: Collects sensitive information from victims' computers
Severity level: High

QBot is a Trojan, also known as QakBot, which has been active for years. It was originally known as a financial malware designed to target governments and businesses for financial fraud by stealing user credentials and keystrokes. It was observed by threat researchers at the time that it was delivered through phishing campaigns, or by another malware, such as Emotet.

FortiGuard Labs recently captured an MS Office Word document in the wild that was spreading a variant of QBot. Normally, such Word documents are only delivered in a phishing email designed to deceive the victim into opening it. Unfortunately, we only captured the Word file, so we do not know how it is being delivered.

I performed a deep analysis on this sample file. QBot uses complicated techniques and a framework designed for it to run covertly on a victim's system. In this post, I will explain how it works on a victim's machine, as well as what techniques it uses.

Opening the Word Document Containing QBot

As you may have expected, the Word document includes a malicious Macro. Once the file is opened in the Word program, it asks the victim to click a yellow button, as shown in Figure 1.1 (on the left). The image on the right part shows what it looks like after the "Enable Content" button has been clicked. It spoofs the victim into thinking it is working hard on loading data.

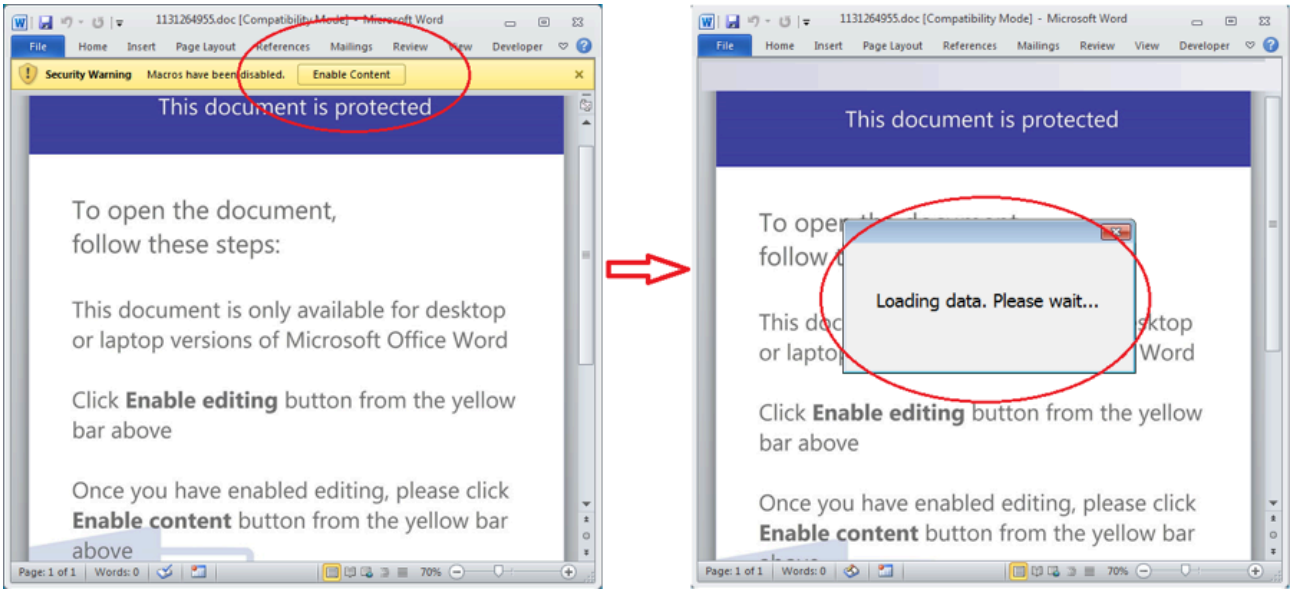

Figure 1.1. Word document is opened in Word program

However, what's actually happening is that the malicious Macro (VBA code) is executing in the background. It has a function called Document_Open() that is automatically called while the file is opening.

The Macro creates a folder named "tmpdir" in "C:\Users\Public\". It then tries to download the QBot payload into this folder. The attacker puts the QBot payload file in five places. These are:

hxxp://pickap[.]io/wp-content/uploads/2020/04/evolving/888888[.]png

hxxp://decons[.]vn/wp-content/uploads/2020/04/evolving/888888[.]png

hxxp://econspiracy[.]se/evolving/888888[.]png

hxxp://enlightened-education[.]com/wpcontent/uploads/2020/04/evolving/888888[.]png

hxxp://kslanrung[.]com/evolving/888888[.]png

The URLs are decoded from five Base64-encoded strings with PowerShell code. During its execution, the malware shows the victim the information shown in Figure 1.1 (the image on the right.)

The PowerShell code repeats picking one of the five URLs once within a loop to download the payload file 888888.png (EXE file) into "C:\Users\Public\tmpdir\". It then renames it to "file*.exe" that finally gets executed. It stops the loop when the first payload file has been downloaded. (Note: the "*"symbol used here can be 1, 2, 3, 4, or 5. Therefore, the downloaded payload file will be referred as "file1.exe" in this analysis.)

Figure 1.2 shows the powershell code decoded by Macro to download QBot payload file.

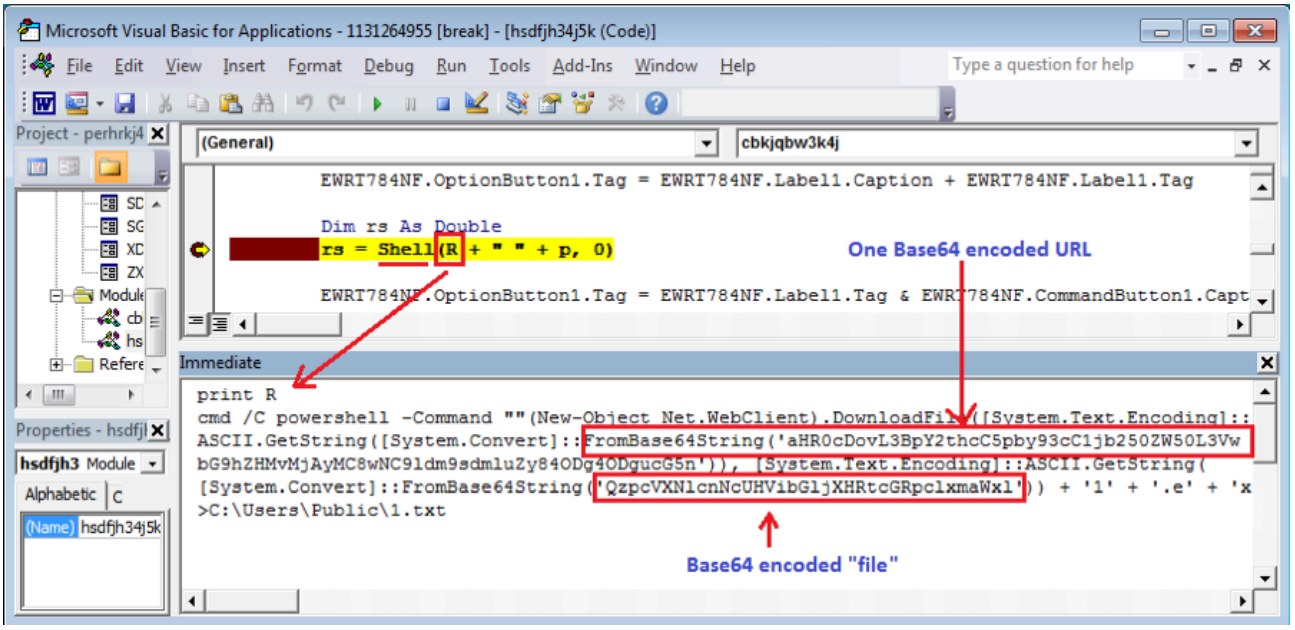

Figure 1.2. PowerShell code to download QBot payload and execute it

Looking at the five URLs, you may notice that they were probably built with the same website builder, which may have vulnerability that allows the uploading of EXE file onto it with a PNG extension name.

Executing the Downloaded Payload

“file1.exe” is the downloaded payload that is protected in a packer. When it starts, the packer extracts the protected QBot into memory and then overrides the packer’s code. Once that is completed its entry point gets called.

QBot provides some command line parameters, like “/C”, “/W”, “/I”, “/P”, “/Q” and so on, for performing different features. When it is started by the PowerShell code, no parameter is provided. It goes to a non-parameter branch, which first spawns a normal child process of itself with the command line parameter of “/C”. Figure 2.1 shows it about to create a child process with that command line parameter.

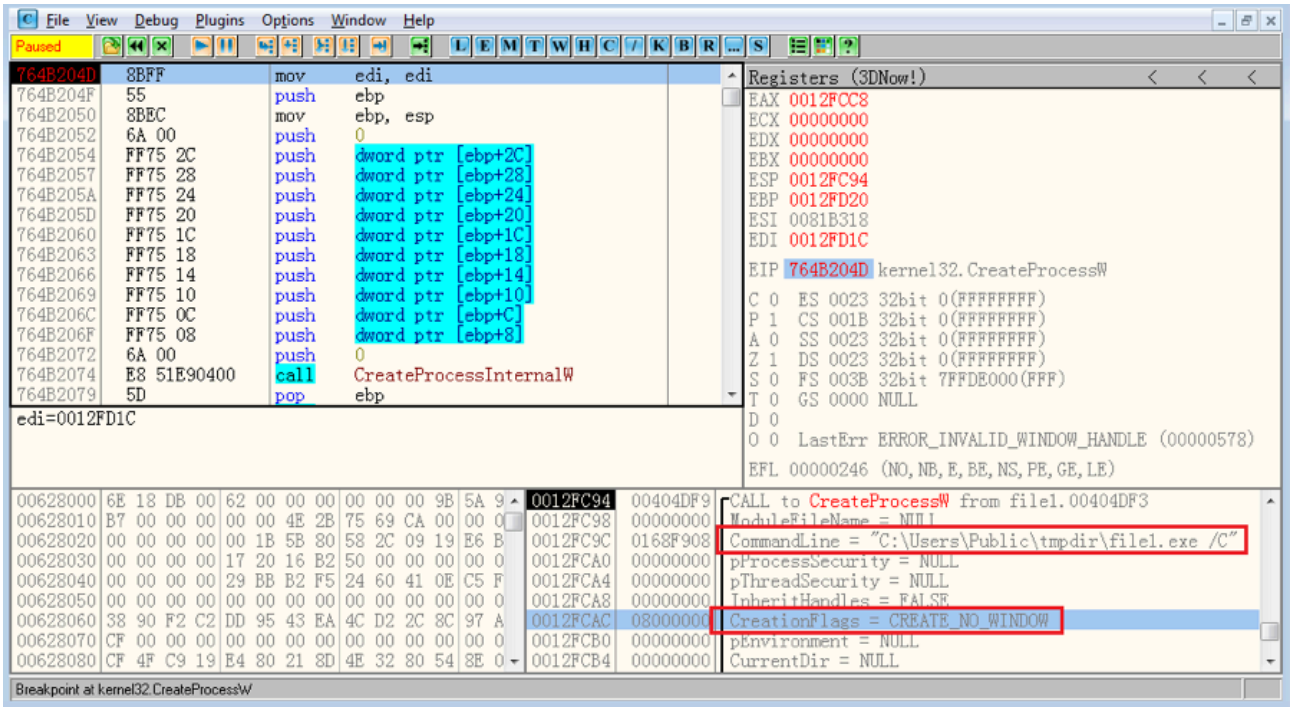

Figure 2.1. Spawning a child process with parameter “/C”.

The “/C” feature is for checking whether or not it is running in an analysis environment. The following are the ways that it performs that detection.

- It executes ASM code with the keyword “VMXh”, where it will trigger an exception if it is in a Virtual Machine. The exception handler can capture the exception and returns 1, otherwise it returns 0. Below is the ASM code snippet.

```
[…]
.text:00403452          push    ebx
.text:00403453          push    ecx
.text:00403454          push    edx
.text:00403455          mov     dx, 5658h
.text:00403459          mov     ecx, 564D5868h  ;; "VMXh".
.text:0040345E          mov     eax, ecx
.text:00403460          mov     ecx, 14h
.text:00403465          in      eax, dx
.text:00403466          mov     [ebp+var_1C], eax
.text:00403469          pop     edx
.text:0040346A          pop     ecx
[…]
```

- It checks if it runs in Virtual Machine environment by calling the API function SetupDiEnumDeviceInfo() to enumerate the device information. It then checks to see if that device information contains the below texts, which are keywords for Virtual Machine software, like “VMware”, “VirtualBox”, “CwSandbox”, “Red Hat Virtualization”, “QEMU”, and so on.

"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"

- It checks to see if any analysis tools are running, like “VMware Tools Service”, “VMware Activation Helper”, “Metasploit Metsvc Backdoor”, and “Windump”, whose process names are *"vmtoolsd.exe","vmacthlp.exe", "metsvc-server.exe"*, and *"windump.exe"*.
- It determines if the current process is running in “Sandboxie” by determining whether a special Dll file has been loaded, and also if the current process name contains the string “sample”, “mlwr_smpl”, or“artifact.exe”. It does this because some sandbox tools may change the sample file name into them.
- Besides the above methods, it also checks the CPU information by calling the ASM instruction *cpuid*.

The constant strings that appear in the detections are decrypted. In fact, not only these strings, but also all constant strings are encrypted by default, and they are decrypted before referring to them.

After all the detections, the child process exits with an exit code of 1 if any of the above parameters is triggered, and with a 0 if nothing is triggered.

Back to the Parent Process

The parent process can call the API GetExitCodeProcess() to obtain the exit code. When it detects that QBot is running in an analysis device, it does not exit the process immediately, but secretly sets a global variable. As a result, it goes to a different code branch where it does some irrelevant things but load the core module and exit the process at last. I will explain this later when we reach the core module.

If it is not running in an analysis device, it continues to create a home folder under the “%AppData%\Microsoft\” folder for saving QBot’s process and data. The home folder’s name is randomly generated. In my device, it is “Vhdktrbeex”. It may differ on different devices. It checks if the current QBot process is from its home folder. Of course, it is not for this first time, as it is in the folder “C:\Users\Public\tmpdir\”.

It then copies file1.exe into the home folder and renames it as “mavrihvu.exe”. The file name is generated from the user name of the victim. In Figure 2.2, below, you can see the ASM code snippet to compare the two folder names.

```
00401BC3
00401BC3 loc_401BC3:                             ; CODE XREF: start+191↑j
00401BC3             push    10h                 ; int
00401BC5             lea     eax, [ebp+String1]
00401BC8             push    eax                 ; int
00401BC9             push    offset String       
00401BCE             call    sub_404847          ; it generates home folder name "Vuasakvgntoj"
00401BD3             add     esp, 0Ch
00401BD6             mov     eax, offset word_410E40 ; "C:\Users\Public\tmpdir"
00401BDB             call    sub_401E80
00401BE0             push    eax                 ; String1 = "Vuasakvgntoj"
00401BE0
00401BE1             lea     eax, [ebp+String1] ; String2 = "tmpdir"
00401BE4             push    eax
00401BE5             call    ds:lstrcmpiW        ; it compares two folder names.
00401BEB             test    eax, eax
00401BED             jz      short loc_401C6C
00401BEF             cmp     dword_411C50, ebx
00401BF5             jnz     short loc_401C60
00401BF7             push    6                   ; dwCoInit
00401BF9             push    edi                 ; pvReserved
00401BFA             call    ds:CoInitializeEx
00401C00             push    edi
00401C01             mov     eax, offset asc_40B740 ; "\""
00401C06             push    eax
00401C07             push    offset Filename ; UNICODE "C\Users\Public\tmpdir\file1.exe"
00401C0C             push    eax
00401C0D             push    offset aC           ; "/c "
00401C12             call    sub_4020AD
```

Figure 2.2. ASM code snippet of comparing home folder name.

In this code branch, it continues to load a resource named “307” from the current process. This is the core module of QBot. The string “307” is decrypted. If it detects being in an analysis device, according to the exit code from the child process called with parameter “/C”, then the “307” string decryption will fail with no error alert. It then does nothing and soon exits the process. It can be treated as an anti-analysis technique.

The content of “307” is an encrypted PE file. However, it does not really load the core module “307” to perform its work here. Instead, it loads another resource, named “308”, from the decrypted “307” module.

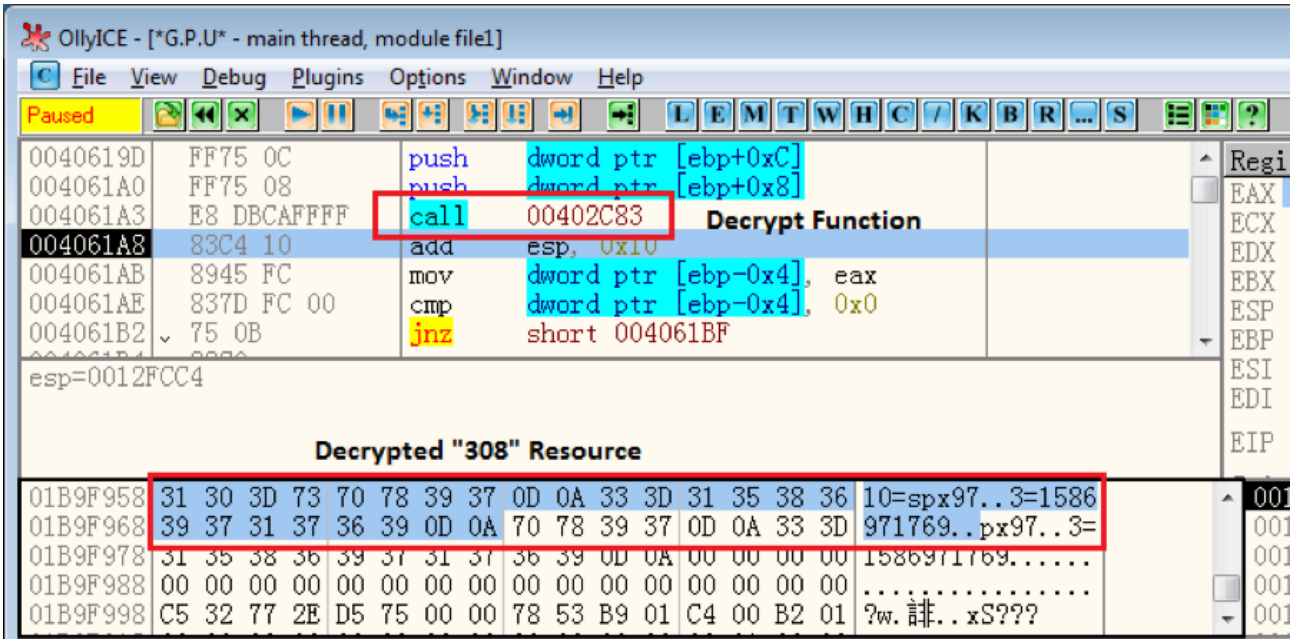

Figure 2.3 Display decrypted data of “308” and decryption function

From the decrypted data of “308”, as shown in Figure 2.3, “spx97” in “10=spx97” is the variant identification of QBot and 3=1586971769, is a unix epoch time, is the creation time of resource “307”. They are used when communicating with the C2 server to reveal its version. Based on that information, the C2 server determines if it needs to be upgraded.

It then creates a file named “mavrihvu.dat” to save the encrypted configuration data. Below is its content before encryption.

```
01AFFB60  D8 88 6C 71 57 93 A7 1D D2 B8 97 4F 1B FC C1 E3  ??lqW“§ò?—Oüá?
01AFFB70  A2 D0 F7 C0 31 31 3D 32 0D 0A 31 3D 32 32 2E 34   ¢ D÷à11=2..1=22.4
01AFFB80  31 2E 35 37 2D 31 35  2F 30  35 2F  32 30  32 30 0D  1.57-15/05/2020.
01AFFB90  0A 32  3D 31 35 38 39  36 30  37 37  31 37  0D 0A      .2=1589607717..
```

It contains several pieces of basic information. The first 14H bytes are the SHA1 value of the rest content, 11=2 records the type of hard-drive, 1=22.41.57-15/05/2020 is the time and date when the QBot was installed on the victim’s device, 2=1589607717 is the Unix time of the installation time. This “mavrihvu.dat” file will be frequently used later to load and save other configuration data for QBot.

It also creates a WMI (Windows Management Instrumentation) Object to execute the process of “%AppData%\Microsoft\Vhdktrbeex\mavrihvu.exe” without a parameter. To do so, it calls the ConnectServer() API with the WMI namespace “ROOT\\CIMV2”, CoSetProxyBlanket(), and GetObject() with the “Win32_Process". Finally, it calls Put() with the command line “%AppData%\Microsoft\Vhdktrbeex\mavrihvu.exe” and ExecMethod() to run it.

Figure 2.4 shows how the ASM code snippet calls Put() and ExecMethod().

```
00409A2F
00409A2F loc_409A2F:                             ; CODE XREF: sub_40994A+DA↑j
00409A2F          mov     eax, [ebp+var_18] ; pInParams
00409A32          mov     ecx, [eax]
00409A34          lea     edx, [ebp+var_C] ; pClassInstance
00409A37          push    edx
00409A38          push    esi
00409A39          push    eax             ; pInParams
00409A3A          call    dword ptr [ecx+3Ch] ; hr = pInParams->SpawnInstance(0, &pClassInstance);
00409A3D          push    8
00409A3F          pop     eax
00409A40          push    esi
00409A41          mov     [ebp+var_2C], ax
00409A45          mov     eax, [ebp+arg_0] ;
00409A48          lea     edx, [ebp+var_2C]
00409A4B          push    edx             ; &command
00409A4C          push    esi
00409A4D          mov     [ebp+var_24], eax ; "%AppData%\Microsoft\Vhdktrbeex\mavrihvu.exe"
00409A50          mov     eax, [ebp+var_C] ; pClassInstance
00409A53          mov     ecx, [eax]
00409A55          push    offset aCommandline ; "CommandLine"
00409A5A          push    eax             ; pClassInstance
00409A5B          call    dword ptr [ecx+14h] ; hr = pClassInstance->Put(L"CommandLine", 0, &command, 0);
00409A5E          mov     eax, [ebp+pProxy]
00409A61          mov     ecx, [eax]
00409A63          push    esi
00409A64          lea     edx, [ebp+var_1C]
00409A67          push    edx
00409A68          push    [ebp+var_C]     ; pClassInstance
00409A6B          push    esi
00409A6C          push    esi
00409A6D          push    edi             ; "Create"
00409A6E          push    ebx             ; "Win32_Process"
00409A6F          push    eax             ; pProxy
00409A70          call    dword ptr [ecx+60h] ; hr = pSvc->ExecMethod("Win32_Process", "Create", WBEM_FLAG_RETURN_[M
00409A73          test    eax, eax
00409A75          jz      short loc_409A7E
00409A77          mov     [ebp+var_4], 0FFFFFFF8h
00409A7E
00409A7E loc_409A7E:                             ; CODE XREF: sub_40994A+E3↑j
00409A7E                                         ; sub_40994A+12B↑j
00409A7E          pop     ebx
```

Figure 2.4, WMI object to execute QBot process

I think using a WMI object to run QBot is a better way than directly calling CreateProcess for protecting the process. As we know, the WMI object is handled by the Windows process “wmiprvse.exe”, which then executes the mavrihvu.exe process. Below, Figure 2.5 displays a screenshot showing that the process tree starts running “file1.exe”, and that “mavrihvu.exe” is started by “wmiprvse.exe” (WMI Provider Host).

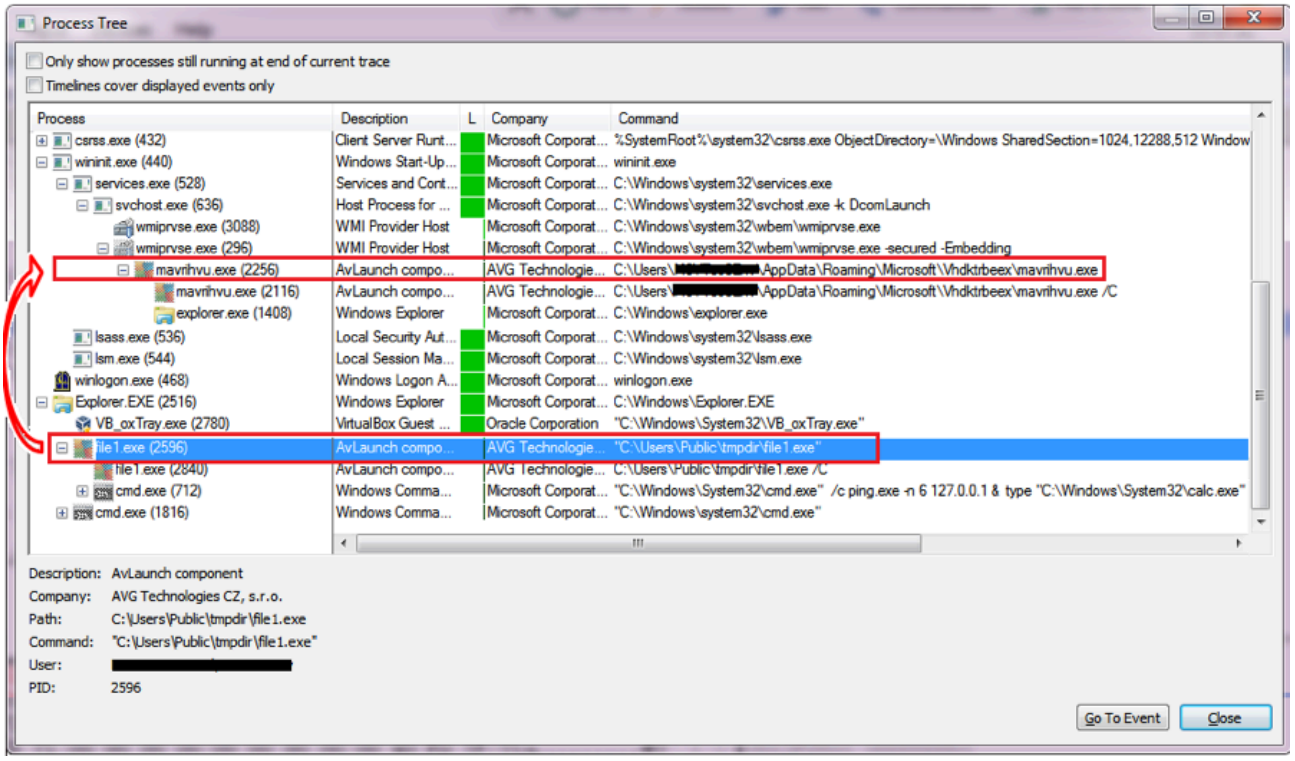

Figure 2.5, Process Tree

I will explain in detail how “mavrihvu.exe” is started by “wmiprvse.exe” in the next section of this blog.

"file1.exe” then continues to create a one-time run task in the task schedule. It uses the command “C:\Windows\system32\schtasks.exe /Create /RU \"NT AUTHORITY\SYSTEM\" /tn **qyuoeflyq** /tr \"C:\Users\Public\tmpdir\file1.exe\" /I qyuoeflyq /SC ONCE /Z /ST 22:48 /ET 23:00”. The created task name is “qyuoeflyq”, which executes the command “C:\Users\Public\tmpdir\file1.exe /I qyuoeflyq”. “/I qyuoeflyq” is the command line parameter. The code branch then replaces the content of “file1.exe” with “calc.exe” to destroy the “file.exe” and then delete this one-time-run task of its name “qyuoeflyq” that passes with “/I”.

Figure 2.6 shows that it calls the API CreateProcessW() in the “/I” handling code branch to execute commands to replace the content of “file1.exe”.

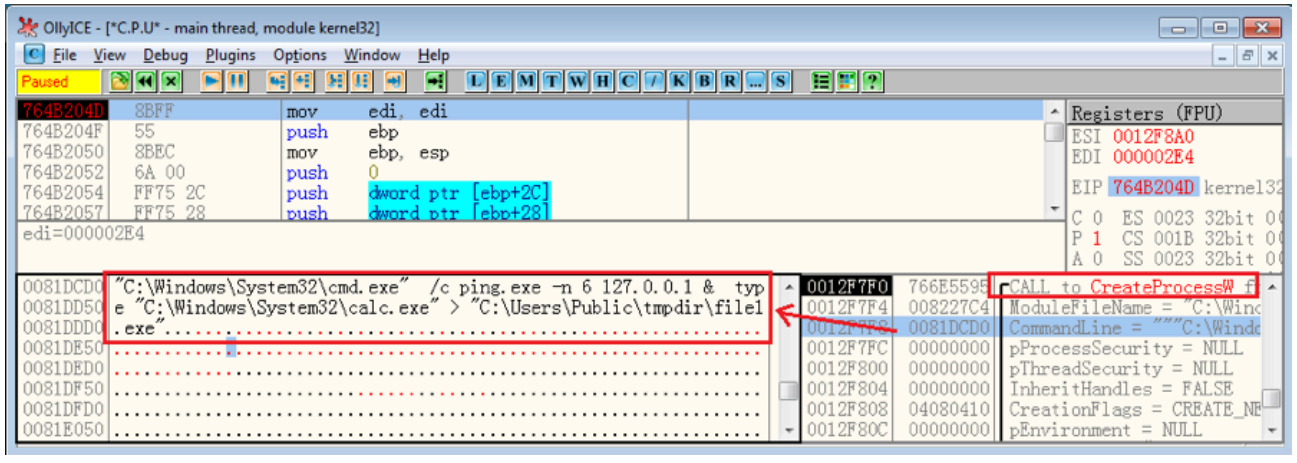

Figure 2.6, “/I” parameter handler executes a command

At this point, “file1.exe”’s task is completed. It then calls the API ExitProcess() to exit the process.

WMI Provider Host Executes QBot

In Figure 2.5, you can see that QBot (“mavrihvu.exe”) is started by the WMI Provider Host (“wmiprvse.exe”) with no parameter. It performs all the work that “file1.exe” does, like checking to see if it is in an analysis device (parameter “/C”), which I explained earlier, and then checks to see if it is from its home folder, “%AppData%\microsoft\Vhdktrbeex\”. This time the result is apparently yes, therefore, it will go to different branch than where “file1.exe” goes.

Next, it creates a suspended process from one of a number of common processes, including "C:\Windows\explorer.exe", "C:\Windows\System32\mobsync.exe", and "C:\Program Files\Internet Explorer\iexplore.exe". Which one is used depends on which one works first. QBot then moves onto the selected common process to execute its malicious code to protect it from being identified by the victim. The three common process strings are encrypted by default.

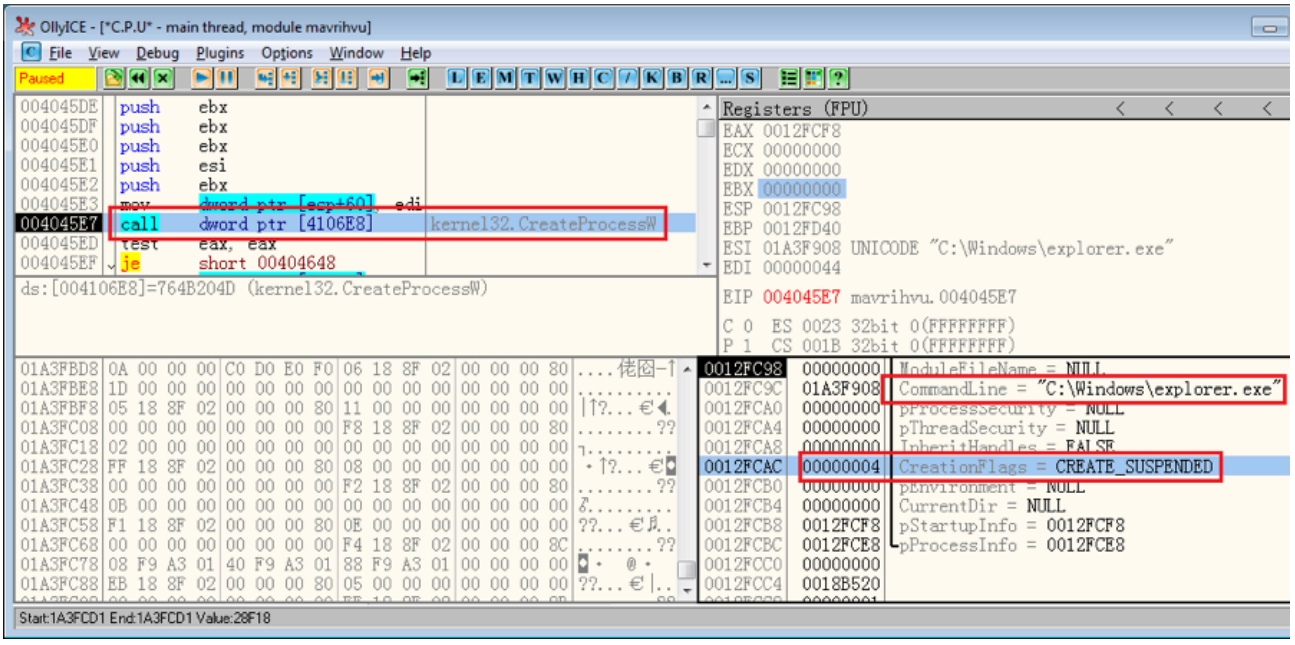

Figure 3.1, It creates suspended explorer.exe

Figure 3.1 is a screenshot of when it creates “explorer.exe” with a CREATE_SUSPENDED flag. This way, QBot can modify the memory data of “explorer.exe” and then resume its running.

QBot copies its entire data from its memory into explorer.exe’s memory. To do this, it calls API ZwCreateSection(), ZwMapViewOfSection(), and memcpy() to copy the data. It then reads relocation data from the PE structure and adjusts the relocation offsets within the copied code in “explorer.exe”. Lastly, it calls the API GetThreadContext() to get the current entry point of “explorer.exe” and then modifies it so it is able to jump to the copied QBot’s code (entry point). It then calls ResumeThread() to resume “explorer.exe” running from its entry point. Now, with all the work of “mavrihvu.exe” with no parameter is done, it calls API ExitProcess() as exiting is the last thing for it to do. Now, QBot is perfectly running within explorer.exe process.

QBot Executes in Explorer.exe

The code that runs in “explorer.exe” has a new entry point that is first called. Its main task is to load and decrypt resource “307”. It calls the APIs FindResourceA(), SizeofResource(), and LoadResource() to load resource “307” into memory. Next, it gets the “307” data decrypted by calling a RC4 function. Below, in Figure 4.1, is the just decrypted data of “307”, which is a PE file.

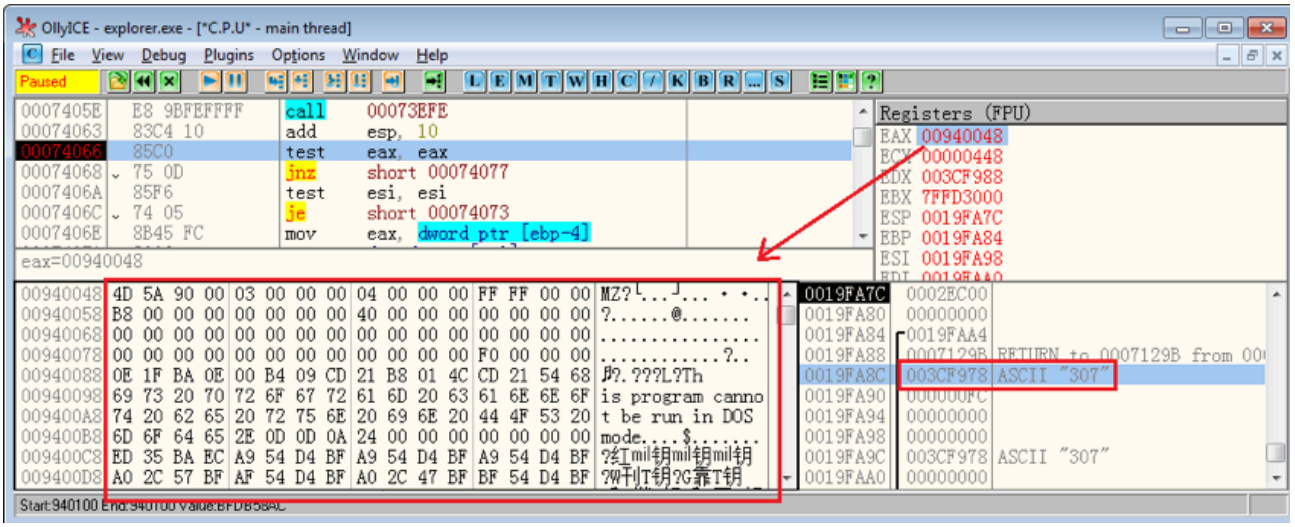

Figure 4.1, Decrypted resource “307”

I dumped and analyzed the PE file. It is a Dll file, which will be the core module of QBot. It contains three resources, “308”, “310” and “311”, which are used by the core module; I’ve explained the content of “308” before. For others, I will walk you through their contents when they are decrypted.. Figure 4.2 shows the three resources of the dumped resource “307” in a PE analysis tool.

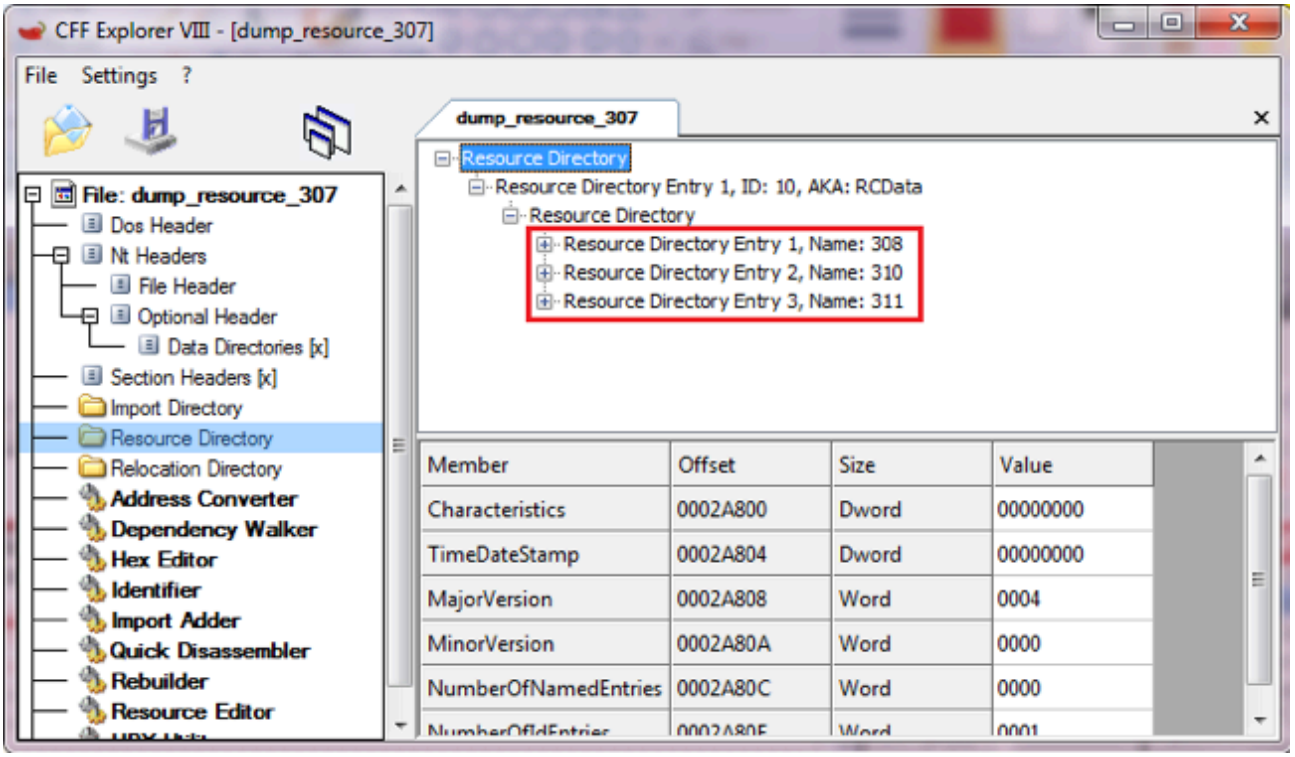

Figure 4.2, Three resources of dumped resource “307”

It continues loading each section from the “307” PE structure into newly allocated memory by calling the API VirtualAllocate(). It then goes to repair the relocation data and import the neessary APIs for getting the core module ready to execute in “explorer.exe”, which is the same way a PE Loader does when creating a process.

The core module’s Entry Point is called when above steps are complete. Figure 4.3 shows a snippet of ASM code to call the Entry Point that is saved in var_10.

```
00402673 ; ---------------------------------------------------------------------------
00402673
00402673 loc_402673:                             ; CODE XREF: sub_40242F+240↑j
00402673                cmp     [ebp+arg_8], 0
00402677                jz      short loc_402681
00402679                mov     eax, [ebp+arg_8]
0040267C                mov     ecx, [ebp+var_10]
0040267F                mov     [eax], ecx
00402681
00402681 loc_402681:                             ; CODE XREF: sub_40242F+248↑j
00402681                mov     eax, [ebp+var_C]
00402684                mov     ecx, [ebp+arg_0]
00402687                mov     [eax+34h], ecx
0040268A                push    [ebp+arg_4]
0040268D                push    1
0040268F                push    [ebp+arg_0]
00402692                call    [ebp+var_10]     ; ;;; It calls the Entry Point of resource "307".
00402692
00402695 locret_402695:                          ; CODE XREF: sub_40242F+176↑j
00402695                leave
00402696                retn
00402696 sub_40242F     endp
00402696
```

Figure 4.3, The Entry Point in var_10 of core module resource “307”is called.

I’ll continue to analyze what the core module does in explorer.exe in the next part of this analysis. For example, I will look at how QBot connects to its C2 server and what data it steals from victim’s device and sends to its C2 server. Stay tuned.

Conclusion

In the first part of this report, I provided a detailed explanation of how this variant of QBot is downloaded by an Office Word document by using a malicious Macro, and how it uses complicated techniques to hide and protect itself from being recognized by the victim.

During my analysis, QBot kept upgrading its payload file, almost once a day. I’ll keep tracking its actions and posting more analysis for it when some new features have been added.

Solution

The most common vector for delivery of these threats is email. FortiMail deployed on-premises or in the cloudcan be used to detect malicious phishing content and can be configured to send attachments to the FortiSandbox solution (ATP), to determine if a file displays malicious behavior. To mitigate against file based threats on PDF and Office documents, FortiMail and FortiGate support Content Disarm and Reconstruction (CDR) to remove active threats such as QBot from these file formats.

Fortinet customers running FortiGate and FortiMail are already protected from this QBot variant by FortiGuard's Web Filtering service, AntiVirus service, and CDR feature as follows:

The downloading URLs are rated as "**Malicious Websites**" by the FortiGuard Web Filtering service.

The Word document and downloaded file1.exe are detected as "**VBA/Qbot.CC!tr.dldr**" and "**W32/QBOT.CC!tr**" and blocked by the FortiGuard AntiVirus service.

IOCs:

URLs

hxxp://pickap[.]io/wp-content/uploads/2020/04/evolving/888888.png
hxxp://decons[.]vn/wp-content/uploads/2020/04/evolving/888888.png
hxxp://econspiracy[.]se/evolving/888888.png
hxxp://enlightened-education[.]com/wp-content/uploads/2020/04/evolving/888888.png
hxxp://kslanrung[.]com/evolving/888888.png

Sample SHA-256

[Original Word Document]
432B6D767539FD5065593B160128AA7DCE271799AD2088A82A16542E37AD92B0

[file1.exe or 888888.png]
D3B38681DBC87049022A3F33C9888D53713E144A277A7B825CF8D9628B9CA898

References:

https://malware.wikia.org/wiki/Qakbot

https://docs.microsoft.com/en-us/powershell/scripting/samples/getting-wmi-objects--get-ciminstance-?view=powershell-7

Learn more about FortiGuard Labs threat research and the FortiGuard Security Subscriptions and Services portfolio. Sign up for the weekly Threat Brief from FortiGuard Labs.

Learn more about Fortinet's free cybersecurity training initiative or about the Fortinet Network Security Expert program, Network Security Academy program, and FortiVet program.
