Latest Trickbot Variant has New Tricks Up Its Sleeve

By sharon

Published: 2018-08-14 · Archived: 2026-04-05 15:12:46 UTC

Trickbot is well-known financial Trojan that targets customers of large banks and steals their credentials." It is a modular malware that includes different modules for its malicious activities. It has been around since 2016 and since then new variants have appeared on an ongoing basis, each time updated with new tricks and modules.

Trickbot includes modules for stealing data from browsers, stealing data from Microsoft Outlook, locking the victim's computer, system information gathering, network information gathering and domain credentials stealing.

Cyberbit malware research team researched the latest Trickbot variant to understand how it differs from previous variants and fully understand its new behaviors and tricks. The new variant of Trickbot comes with a stealthy code injection technique that performs process hollowing using direct system calls, anti-analysis techniques and disabling of security tools. The behavior patterns of this new Trickbot variant somewhat resemble those of the Flokibot banking Trojan.

In this report we analyze the new variant and its infection vector – a malicious Word document.

Zooming in to the infection vector

This variant of Trickbot is downloaded by a Word document (SHA256: aef2020534f81bbebec6c9b842b3de6fd4f5b55fe5aeb3d9e79e16c0a3ff37ab) which contains a macro code. We first learned about it from a Twitter post. However, this document will not execute its macro until the user had both clicked "enable content" to enable execution of macros and zoomed in/out of the document. While this will probably evade sandboxes, it may also evade humans who don't zoom the document.

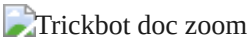

Figure 1 – The zoom in/out bar is highlighted in red

Figure 2 – Macro executes only if the window was resized. The method InkPicture1_Resize triggers when the user zooms in/out.

The macro is obfuscated as are most malicious macros and ends up executing a PowerShell script that downloads and executes Trickbot. After deobfuscation and renaming, the PowerShell script looks like this:

Figure 3 – PowerShell script download and executes Trickbot

Payload analysis – Trickbot variant

SHA256: 1c81272ffc28b29a82d8313bd74d1c6030c2af1ba4b165c44dc8ea6376679d9f

By initial examination of the malware, we can see the debug path is "c:\users\exploitdb\desktop\esetfuck\release\esetfuck.pdb"

The malware's author probably dislikes ESET – A well-known IT security company.

Anti-analysis, resource decryption & execution

Upon execution, the malware sleeps for 30 seconds to evade sandboxes by calling Sleep(30000). Then it decrypts its resource using the RSA algorithm.


Trickbot main

Figure 4 – Sleep of 30 seconds is marked in red. The call to the resource decryption function is marked in purple

Decryption procedure description

1. The function at 0x405680 receives a key (private_key), key size (16), a pointer to the encrypted buffer and its size
2. A handle to a cryptographic service provider is acquired by CryptAcquireContextW, with chosen provider PROV_RSA_FULL
3. The public key is imported from a key BLOB that is embedded within the executable
4. the private_key is copied right after a BLOB header in the memory, to form a key BLOB
5. CryptImportKey is called again, with the key BLOB being the one formed at step 4, and the decryption key as the public key from step 3. The output key is saved (output_decryption_key)
6. CryptEncrypt is called with the output_decryption_key saved from step 5 for the decryption of the resource
7. The public key and the output_decryption_key are destroyed using CryptDestroyKey

The decrypted resource is a DLL (SHA256: 31A4065460CEF51C8B4495EFC9827926A789F602F5AD5C735EA1D88CAFAC135A) with an exported function named "shellcode_main"


Trickbot decryption function

Figure 5 – Inside the resource decryption function

Then, there are lots of calls to:

- CreateWindowEx with garbage-looking class names and windows names
- SendMessageW with undefined message code (0x64 and 0xfa) and non-existent window
- GetLastError
- InSendMessage

The calls to CreateWindowEx are never executed and the logic of this code leads eventually to 27 calls to SendMessageW with undefined message code – 0x64 and nonexistent window. The calls to InSendMessage are

also never executed.


Trickbot createwindowex

Figure 6 – These are only part of the calls, there are many more below

After jumping from one call to another (which seems to have no purpose at all other than confusing the analyst), the malware continues its malicious intent: the decrypted DLL is mapped to a buffer located at starting address 0x10000000.

Then it calls Sleep(3) 3890 times which results in 11 seconds of delay in execution. The large amount of calls with a short sleep time might be good for sandbox evasion, because a short sleep time doesn't look suspicious. Finally, the exported shellcode_main function, located at 0x10001900, is executed.


Trickbot tonsofsleep

Figure 7 – Sleeping and waking up every 3 milliseconds

Now let's skip to the Trickbot's process hollowing which is implemented differently in this variant.

Trickbot process hollowing using direct system calls

This new variant makes use of process hollowing (a.k.a RunPE) – as in older samples of Trickbot. The process hollowing technique is used for unpacking.

We observed that in this case the process hollowing is done using direct system calls, in a way that is very similar to the Flokibot malware. We suspect that some piece of code is shared between these malwares.

As in the Flokibot malware – not all the functions used for the process hollowing were directly called using system calls – some of them were called from the functions addresses that were saved on the stack earlier.

At the end of this section we show some similarities and differences between Flokibot and Trickbot.

Let's dive into Trickbot to see the hollowing in action.

As usual, a suspended process is created using CreateProcessW. The chosen process is the malware's process itself. The thread context structure of the main thread of that process is saved using GetThreadContext.


Trickbot createprocess1

Figure 8 – Creating the suspended process

Then, the malware uses CreateFileW to obtain a handle to ntdll.dll it copies it a buffer allocated by VirtualAlloc using ReadFile, and then allocates another buffer for mapping it to the memory from its raw copy:


Trickbot createfileNTDLL

Figure 9 – Reading ntdll.dll from the disk


Trickbot mapntdll

Figure 10 – Manually mapping ntdll.dll

The last call instruction in the above figure is a call to a function that receives a pointer to the mapped ntdll.dll buffer and a CRC32 value of the function's name.

This function performs CRC32 on each function name from the mapped ntdll.dll and compares it to the input CRC32 value. If it matches – it returns the offset in that buffer where the desired function starts.

For example, in the following screenshot, the address of UnmapViewOfSection is resolved:


Trickbot umapviewofsection

Figure 11 – A pointer to a buffer containing the mapped ntdll.dll (0x1cd0000) and 0x90483ff6 = CRC32(NtUnmapViewOfSection)

One piece of evidence in the code for the use of the CRC32 algorithm is the constant value 0xedb88320, the hex representation of the reverse CRC32 polynomial, being used.


Trickbot crc32

Figure 12 – Calculating the CRC32 value of the string NtAllocateVirtualMemory. The value 0xedb88320 is the binary representation of the CRC32 polynomial.

Later, the system call number is extracted, the parameters are placed on the stack and the appropriate function is called by placing the system call number on EAX, making a transition to the kernel with the sysenter instruction.


Trickbot syscall_extraction

Figure 13 – system call number extraction

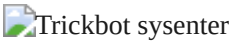
Trickbot sysenter

Figure 14 – sysenter *command. EAX contains the system call number and the stack contains the appropriate parameters*

The procedure above (including re-reading and re-mapping ntdll.dll for each function) is performed for the following functions:

- NtUnmapViewOfSection – Unmapping the original malware module
- NtCreateSection – Creating a section to write the malicious code to
- NtMapViewOfSection – Mapping the above section to the hollowed process
- NtWriteVirtualMemory – Writing the ImageBaseAddress of the current process to the ImageBaseAddress of the hollowed process (in its PEB)
- NtResumeThread – Resume the suspended process and starts execution

As mentioned above, these are not all the functions used for the hollowing. Some functions are called from the addresses saved on the stack earlier. The complete hollowing sequence looks as follows. Functions called using

direct system calls are marked in red. Functions called from saved addresses on the stack are marked in blue.

CreateProcessW → GetThreadContext → NtUnmapViewOfSection → NtCreateSection → NtMapViewOfSection → NtWriteVirtualMemory → SetThreadContext → NtResumeThread

After running the malware, we can see, as in previous variants, it copied itself and its encrypted modules to C:\Users\%USERNAME%\AppData\Roaming\msnet

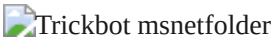
Trickbot msnetfolder

Figure 15 – Trickbot (1c9_patched.exe) & its modules

This variant also disables and deletes the Windows Defender service via the following commands:

- exe /c sc stop WinDefend
- exe /c sc delete WinDefend
- exe /c powershell Set-MpPreference -DisableRealtimeMonitoring $true

The last one being a PowerShell command for disabling Windows Defender real time monitoring.

Trickbot: Similarities/Differences to Flokibot

Malware	Functions called via direct system calls	Functions called from kernel32.dll	Command used for entering kernel mode	Suspended process
Flokibot	NtCreateSection NtMapViewOfSection NtResumeThread	CreateProcessW GetThreadContext SetThreadContext	int 2e	explorer.exe
Trickbot	(same as above) + NtUnmapViewOfSection NtWriteVirtualMemory	(Same as above)	sysenter	The malware itself

You can see here that Trickbot called two additional functions via direct system calls. However, it didn't implement the direct system calls to the 3 functions mentioned in the table above – and could have been stealthier if it did. Oddly enough – these are the exact same functions that didn't have this implementation in Flokibot either.

Another similarity to note is the use of the CRC32 algorithm for hashing the function names. In Flokibot the CRC32 is used in conjunction with XOR of 2-bytes value while in Trickbot the CRC32 is used without any

additional XORing.

Summary of Trickbot Evolution

Trickbot is constantly evolving, adopting new tricks and becoming stealthier. It still has some way to go since it didn't implement all its process hollowing function calls via direct system calls. To avoid being analyzed, it added some very simple and ineffective techniques such as sleep (for a long/short time) and useless function calls. To avoid detection, it disabled and deleted the Windows defender service.

Organizations should be aware of this new trend to directly call functions via system calls. This technique bypasses security tool hooks and therefore most security products will not detect this threat. Cyberbit EDR detects this kind of threat by using pure-behavioral detection which surfaces Trickbot and other threats regardless of indicators of compromise (IoCs).

Hod Gavriel is a Malware Analyst at Cyberbit.

Learn more about Cyberbit EDR Kernel-Based Endpoint Detection vs. Whitelisting

---