

Qakbot Returns - K7 Labs

 labs.k7computing.com/index.php/qakbot-returns/

By Saikumaravel

January 4, 2024

The Qakbot malware has reappeared just four months after law enforcement disrupted its distribution in the “Duck Hunt” operation. Lately, various security companies have noticed the malware spreading through phishing emails. Microsoft, which discovered [this](#), described it as a small-scale campaign starting on December 11, 2023, specifically targeting the hospitality industry. Although the number of these emails is currently low, given Qakbot’s past persistence, it’s anticipated that the volume will rise soon. We got our hands on one such sample by this [tweet](#).

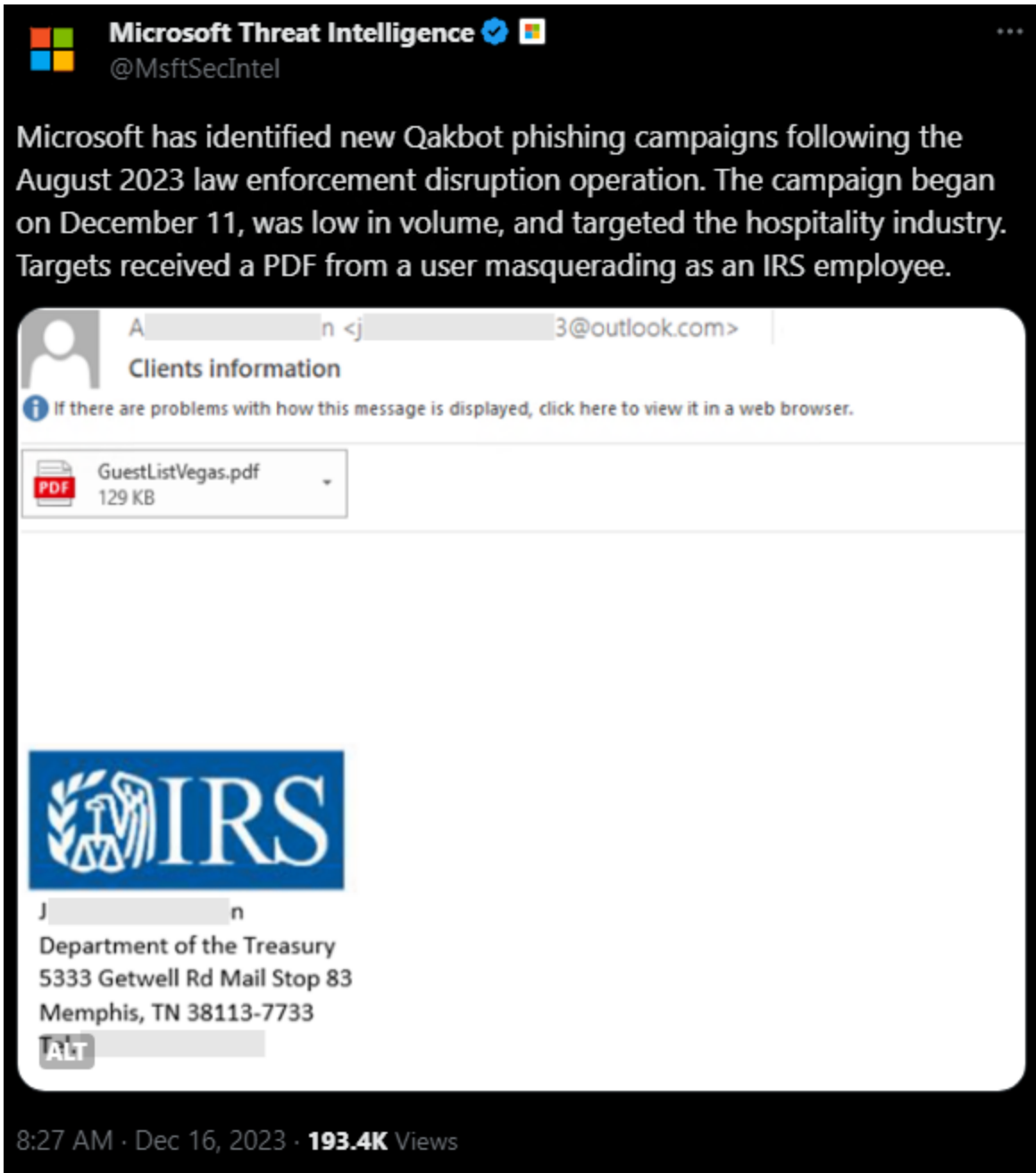


Figure 1: Microsoft discovery of Qakbot resurface

Binary Analysis

As per Microsoft's tweet, in the recent campaign, an MSI file is being downloaded to the user's machine from the malicious PDFs which were spread through phishing mails.

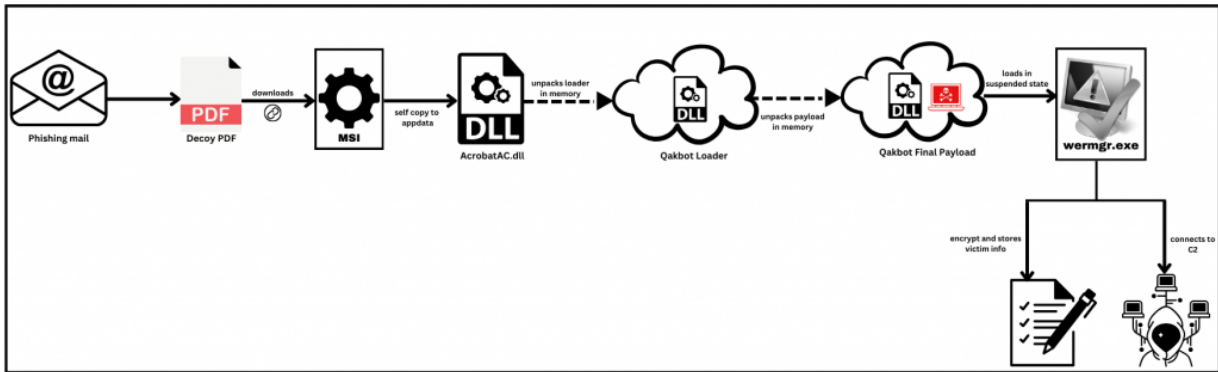


Figure 2: Execution Flow

On analysing the MSI file, we found that the suspicious DLL compressed inside was a patched IDM (Internet Download Manager) DLL with Qakbot inside.

Name	Size	Modified	Attributes	Method	Block
t.dll	919 552	2023-12-19 17:20	A	MSZip	0

Qakbot inside IDM DLL

Name	Offset	Size	Hex
idmcchandler7_64.dll	000c	DWORD	000a6394
	Base	0010	DWORD
			00000001

Ordinal	RVA	Name
0003	00039800	000a64b9 hostFile
0004	0003be20	000a6404 EditOwnerInfo

Figure 3: Qakbot inside IDM DLL

We found that this DLL was packed with a custom packer. Usually unpacking the Qakbot DLL is quite simple. It uses **VirtualAlloc()** to allocate memory to unpacked code and **VirtualProtect()** to change the protection on a memory region. We set breakpoints on both of those APIs to unpack. We first got the dump of the PE file without the MZ header. Later, we found that it was the Qakbot second stage loader by manually adding the MZ header. The threat actor employs this method to avoid detection by EDR, as it scans memory regions for MZ headers to identify potential process injection methods.

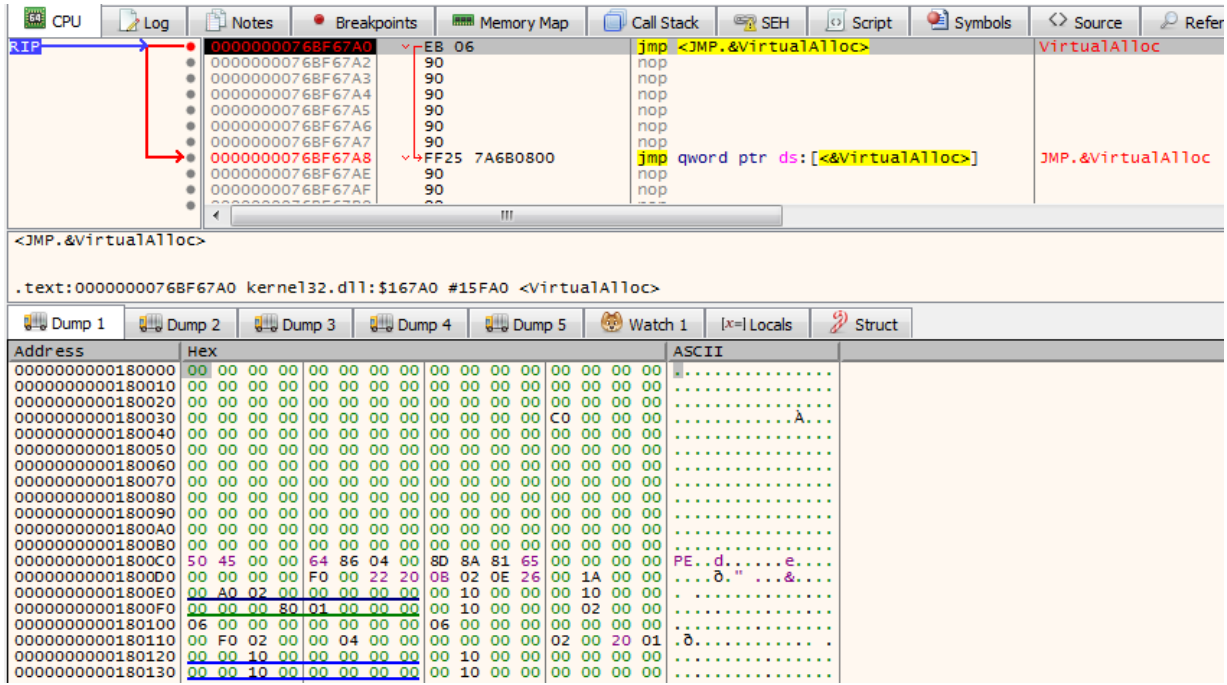


Figure 4: Unpacking Qakbot

On further unpacking, we got the final Qakbot payload which loads from memory while executing. Some security researchers found that in the new campaign, Qakbot uses AES encryption to encrypt and store victim information but the final payload we got was the usual Qakbot payload with the same RC4 encryption.

Name	000c DWORD	00025ad2	Hex	acledit.dll
Base	0010 DWORD	00000001		
<input type="checkbox"/> Show valid				
Ordinal	RVA	Name		
0001	00001000	00025ade	EditOwnerInfo	

Address	Hex	ASCII
0000000001C00000	4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00	MZ.....ÿÿ..
0000000001C00010	B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00e.....
0000000001C00020	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000001C00030	00 00 00 00 00 00 00 00 00 00 00 00 00 01 00 00
0000000001C00040	0E 1F BA 0E 00 84 09 CD 21 B8 01 4C CD 21 54 68	...°.!.!..L!Th
0000000001C00050	69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F	is program canno
0000000001C00060	74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20	t be run in DOS
0000000001C00070	6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00	mode...\$.
0000000001C00080	6D EB 1A 9F 29 8A 74 CC 29 8A 74 CC 29 8A 74 CC	më..).ti).ti.ti
0000000001C00090	62 F2 72 CD 28 8A 74 CC 62 F2 75 CD 25 8A 74 CC	böri(.tibbu!%.ti
0000000001C000A0	32 17 E8 CC 2A 8A 74 CC 29 8A 75 CC 49 8A 74 CC	z.èi=.ti).uII.ti
0000000001C000B0	62 F2 70 CD 2A 8A 74 CC 11 0A 70 CD 3B 8A 74 CC	böpi=.ti).pi;.ti
0000000001C000C0	11 0A 71 CD 2D 8A 74 CC 11 0A 7D CD 6A 8A 74 CC	..qi-.ti).}ij.ti
0000000001C000D0	11 0A 74 CD 28 8A 74 CC 11 0A 76 CD 28 8A 74 CC	..ti(.ti).vi(.ti
0000000001C000E0	52 69 63 68 29 8A 74 CC 00 00 00 00 00 00 00 00	Rich).ti.....
0000000001C000F0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000001C00100	50 45 00 00 64 86 05 00 37 59 81 65 00 00 00 00	PE..d...7Y.e...
0000000001C00110	00 00 00 00 F0 00 22 20 08 02 0E 25 00 F8 01 00	...ö."...%.ö..
0000000001C00120	00 A8 00 00 00 00 00 00 20 10 00 00 00 10 00 00	...ö.....
0000000001C00130	00 00 C0 01 00 00 00 00 00 10 00 00 00 02 00 00	..Ä.....
0000000001C00140	06 00 00 00 00 00 00 00 06 00 00 00 00 00 00 00
0000000001C00150	00 E0 02 00 00 04 00 00 00 00 00 00 02 00 20 00	..ä.....
0000000001C00160	00 00 10 00 00 00 00 00 00 10 00 00 00 00 00 00
0000000001C00170	00 00 10 00 00 00 00 00 00 10 00 00 00 00 00 00
0000000001C00180	00 00 00 00 10 00 00 00 A0 5A 02 00 4C 00 00 00Z.L...
0000000001C00190	EC 5A 02 00 8C 00 00 00 00 00 00 00 00 00 00 00	iz.....
0000000001C001A0	00 B0 02 00 DC 14 00 00 00 00 00 00 00 00 00 00 00	..Ü.....
0000000001C001B0	00 D0 02 00 44 00 00 00 10 45 02 00 38 00 00 00	..D..D...E..8...
0000000001C001C0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000001C001D0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000001C001E0	00 00 00 00 00 00 00 00 00 10 02 00 90 02 00 00
0000000001C001F0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000001C00200	00 00 00 00 00 00 00 00 2E 74 65 78 74 00 00 00text...
0000000001C00210	AD F6 01 00 00 10 00 00 00 F8 01 00 00 04 00 00	..ö.....ö.....
0000000001C00220	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Figure 5: Final Qakbot payload

On dynamic analysis, the MSI drops an installer temp file which passes the command line to invoke rundll32.exe and hides the window to run in background.

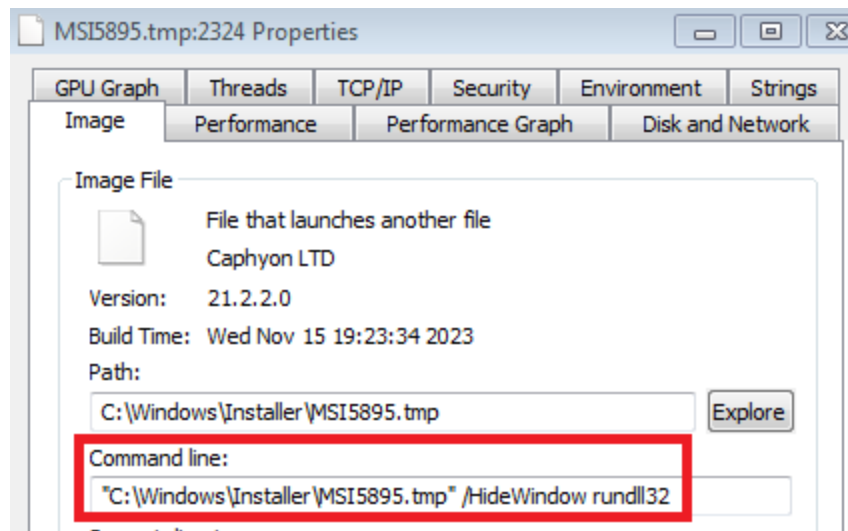


Figure 6: MSI installer

Since the threat actor uses PDF in their kill chain, the malicious DLL self copies itself in the name of **AcrobatAC.dll** and passes the command line arguments to execute the DLL with Qakbot export function **EditOwnerInfo**.

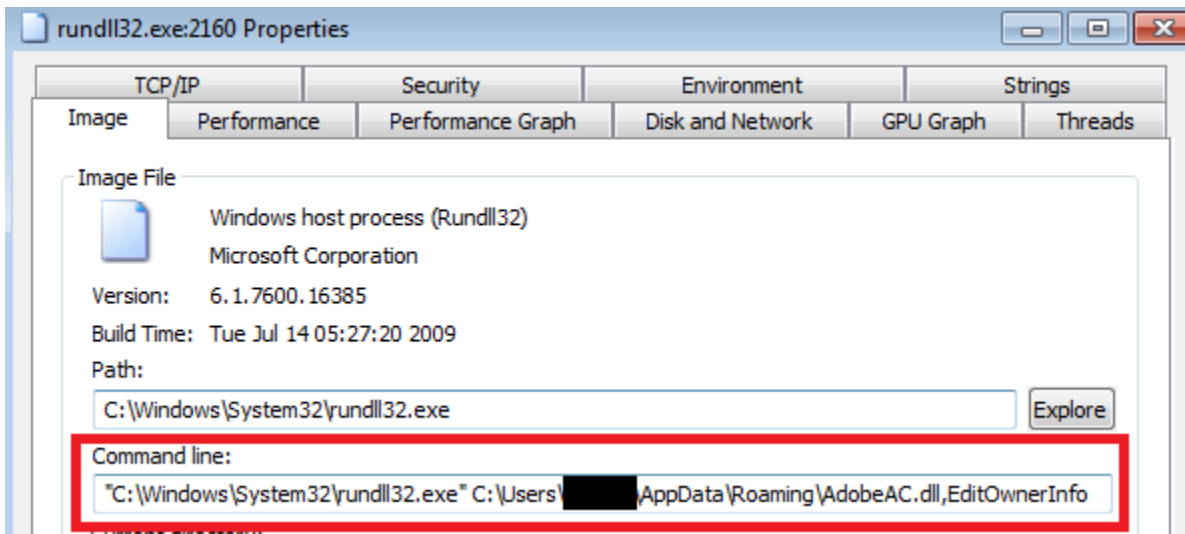


Figure 7: Malicious DLL running on background

It showed the dummy Acrobat window and fake error window as a decoy. Further we found that the malicious DLL invokes the *wermgr.exe* – Windows Error Manager in suspended state to pursue its kill chain.

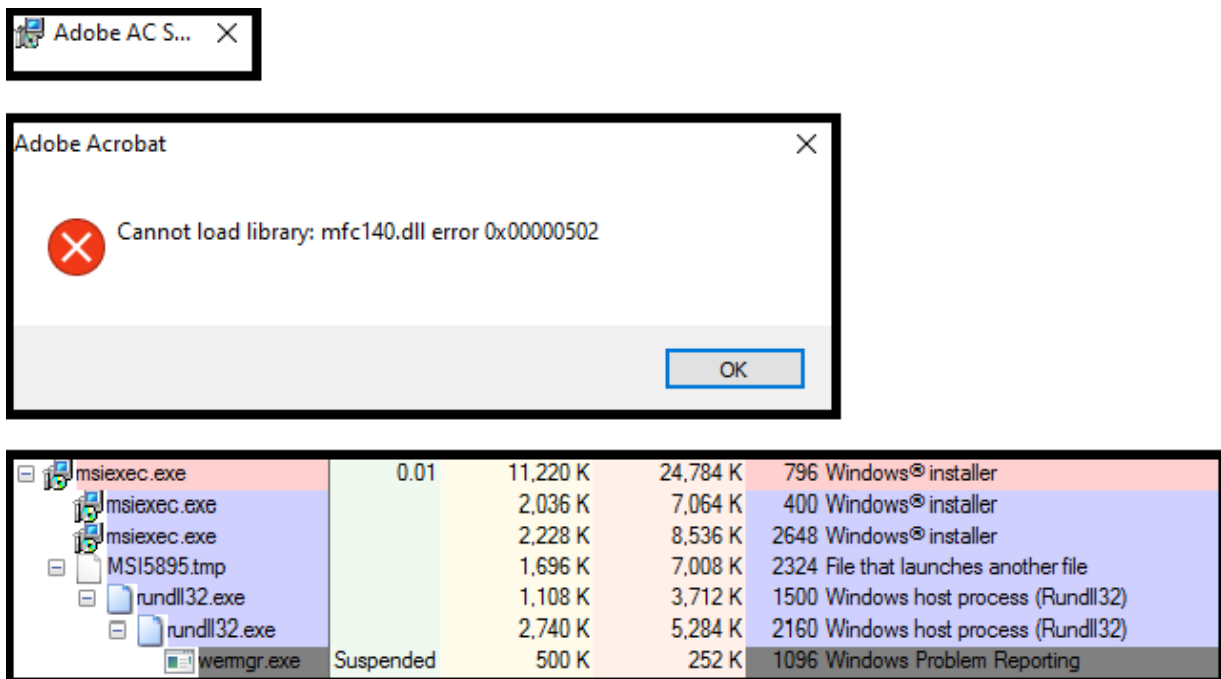


Figure 8: Decoy and invoking wermgr.exe

We dumped the PE file from *wermgr.exe* which was our previously unpacked final Qakbot payload. The threat actor implied **Process Hollowing** technique to inject malicious code into the suspended process of Windows Error Manager.

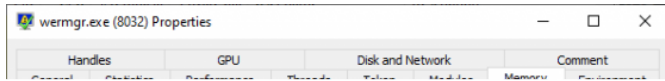


Figure 9: Process Hollowing wermgr.exe

As mentioned earlier, the *wermgr.exe* creates a registry key with RC4 encrypted data of victim system information, timestamp of installation and C2 information which is a usual Qakbot TTP.

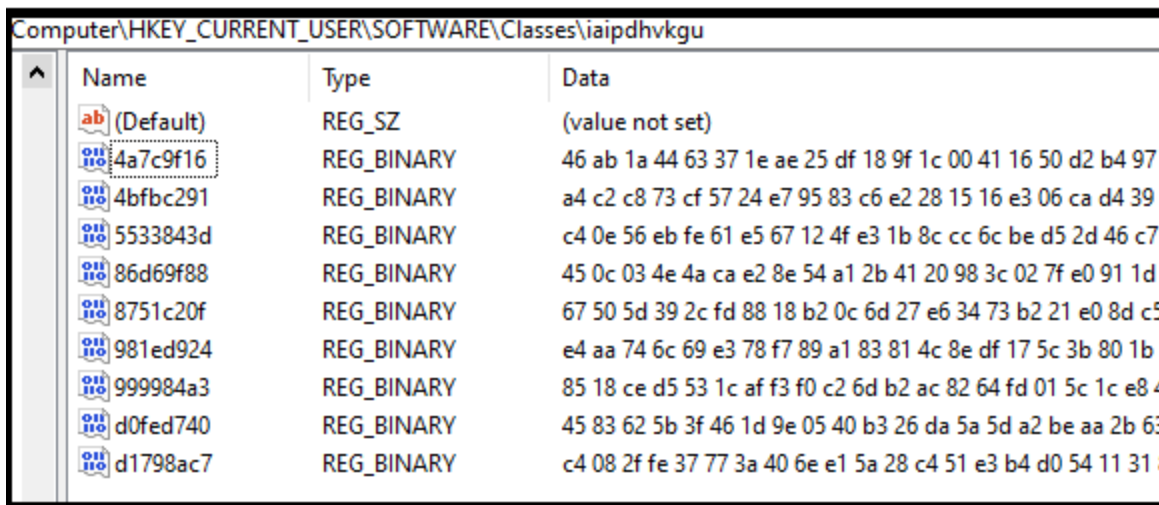
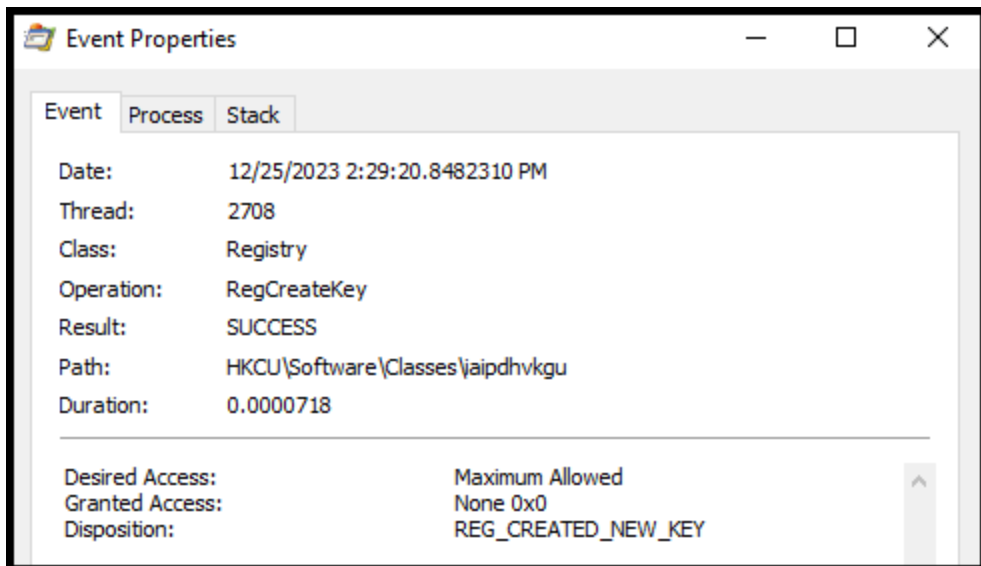


Figure 10: Creates registry key

Qakbot tries to make a C2 connection in the background when the victim believes *wermgr.exe* is running. Since the C2 was down at the time of analysis, it was unable to establish a connection for carrying out any further malicious activity.

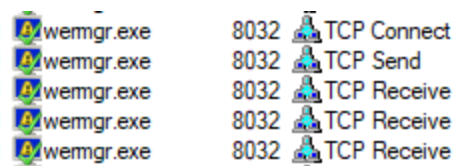


Figure 11: C2 connection by wermgr.exe

We at K7 Labs provide detection against latest threats and also for this newer variant of Qakbot. Users are advised to use a reliable security product such as “K7 Total Security” and keep it up-to-date so as to safeguard their devices.

IoCs

Hash**Detection Name**

723DAE8ED3F157E40635681F028328E6	Backdoor (005af9cf1)
88BBF2A743BAAF81F7A312BE61F90D76	Backdoor (005af9cf1)

References
