An old enemy – Diving into QBot part 1

malwareandstuff.com/an-old-enemy-diving-into-qbot-part-1/

March 30, 2020



Published by hackingump on March 30, 2020

While checking out the Triage Sandbox[1] I stumbled across upon QBot which I've seen already plenty of times at work at GData Cyberdefense AG[2]. This time I wanted to take a closer look at the sample myself.

The first part of this blog article dives deep into how the packer works.



Triage sandbox overview of the analysed sample

Quick summary

The packer used by this sample first allocates virtual memory and fills it with chunks of bytes from its .text section.

After jumping into this allocated area, the address of <u>GetProcAddress</u> [3] is determined by looping over the export table of <u>KernelBase.dll</u>. This function is then used to load further dependencies.

Next another temporary memory is allocated, filled with decrypted code and replaces the code we started with. Finally the sample jumps back to the now decrypted payload and executes it.

1 – Allocating VirtualAlloc



The first step itself does not decrypt any code, however it writes bytes in 0×64 chunks into virtual memory 2304 times ($0\times38400 / 0\times64$). The position of these chunks are calculated loop after loop and do not lie linear in the memory.



2 – Loading dependencies

Once the virtual memory is allocated we can dump the code and load it into IDA to analyse it. After returning the base address of the KernelBase.dll, the offset to the GetProcAddress function is determined by iterating over the export table.

Ordinal	Function RVA	Name Ordinal	Name RVA	Name	
(nFunctions)	Dword	Word	Dword	szAnsi	
0000010D	00030D34	010C	0003D62E	GetNumberFormatEx	
0000010E	00030CDC	010D	0003D640	GetNumberFormatW	
0000010F	0002DA56	010E	0003D651	GetOEMCP	
00000110	000075E2	010F	0003D65A	GetOverlappedResult	
00000111	0000EA14	0110	0003D66E	GetPriorityClass	
00000112	0001C9AA	0111	0003D67F	GetPrivateObjectSecurity	
00000113	00011180	0112	0003D698	GetProcAddress	
00000114	0001469A	0113	0003D6A7	GetProcessHeap	
00000115	000146AC	0114	0003D6B6	GetProcessHeaps	Some
00000116	0000E67D	0115	0003D6C6	GetProcessId	
00000117	00012B5C	0116	0003D6D3	GetProcessIdOfThread	
00000118	00031811	0117	0003D6E8	GetProcessPreferredUILanguages	
00000119	0000EA7A	0118	0003D707	GetProcessTimes	
0000011A	0000EEA2	0119	0003D717	GetProcessVersion	
0000011B	0002296D	011A	0003D729	GetPtrCalData	
0000011C	000229A6	011B	0003D737	GetPtrCalDataArray	
0000011D	00007693	011C	0003D74A	GetQueuedCompletionStatus	
0000011E	00007723	011D	0003D764	GetQueuedCompletionStatusEx	
0000011F	0001C640	011E	0003D780	GetSecurityDescriptorControl]
00000120	0001C6CD	011F	0003D79D	GetSecurityDescriptorDacl	1

exported functions of KernelBase.dll

Explaining this behaviour in pseudo code makes it clearer:

😐 CPU	🍨 Graph 🛛 💆 Log	🗿 Notes	• Breakpoints	🛲 Menory Map	Call Stack	9 <mark>2</mark> SEH	🖉 Script	🔮 Symbols	Source	₽ References	😒 Threads	🔒 Handa	es 👔 Tr	ace
	 006175E5 006175E8 006175E8 	8955 F0 8845 F8	5	nov dword ptr nov eax,dword	ss: ebp-10, e ptr ss: ebp-8	dx	acts: "At	tius tTokenGr	0.07	-	Show FPU			
	 0061/5EE 0061/5EE 0061/5EE 	034D 08 894D F0	5	add ecx, dword nov dword ptr	ptr ss: ebp+8 ss: ebp-4].ec	X	CCA. A	ujustrukenar	oups		EAX 74F	BB128 DE000	kernelba	se.74FBB128
	 006175F4 006175F8 	C745 F4 * EB 09	4 00000000	nov dword ptr jnp 617606	ss: ebp-C 0	_					EDX 001	8C228 8F870	"GetProc	Address"
	006175FD 00617600 00617600	8855 F4 83C2 01	4 1 4	add edx,1	ptr ss: ebp-C		edx: "G	etProcAddres	s"		ESP 001	8F85C	å"Adjust	TokenGroups"
	→ 00617606 00617609	8845 F8 8840 F4	5	nov eax, dword nov ecx, dword	ptr ss: ebp-8 ptr ss: ebp-C	Î					EDI 000	00000		
	00617600 0061760F	3848 18 v 73 41	8	cnp ecx,dword jae 617652	ptr ds:[eax+1	8]	ecx: "A	djustTokenGr	oups"		EIP 006	17620		
	 00617611 00617613 00617613 	6A 0E 8D55 D8	8	push E lea edx,dword	ptr ss:[ebp-2	8]	edv: "G	at Broc Addres			EFLAGS ZF 0 PF	00000304 1 AF 0		
	 00617617 0061761A 	8845 F(8808	0	nov eax, dword nov ecx, dword	ptr ss: ebp-1 ptr ds:[eax]	0]	ecx: "A	djustTokenGr	oups"		OF 0 SF CF 0 TF	0 DF 0 1 IF 1		
-	00617610 0061761F	034D 08 51	5	add ecx,dword push ecx	ptr ss:[ebp+8	1	ecx: "A	- djustTokenGr	oups"		LastError	0000000	0 (ERROR_	SUCCESS)
81.B	00617625	83C4 00 85C0	50000	add esp,C test eax,eax							GS 0028	FS 0053	(SIRIUS,	_30((255)
	 0061762A 0061762C 	 75 18 8855 Fé 	4	jne 617647 nov edx, dword	ptr ss: ebp-C	L					ES 002B 0 CS 0023	OS 0028		
	 0061762F 00617632 00617632 	8845 E8 0FB70C9	50	nov eax, dword novzx ecx, word nov dword atc	ptr ss: ebp-1 ptr ds:[eax+	8 edx*2]	ecx: "A	djustTokenGr	oups" ASE_d11"		DR.0 00000	000	svm	bols
	 00617639 00617630 	8855 E0 8845 F0		nov edx, dword nov eax, dword	ptr ss: ebp-1 ptr ss: ebp-4	41	[ebp-1-	4]:L"KERNELB	ASE.d11"		DR1 00000 DR2 00000	000	3911	0010
	 0061763F 00617642 	880490 0345 08	5	nov eax,dword add eax,dword	ptr ds:[eax+e ptr ss:_ebp+8	dx=4]					DR3 00000	000		
	1	. co nh	11	100 617624			Dumo 2				Default (etdeal	n		
00617A10		expor	rt table	٦			Address	ley.			4	SCI1		
00617620					74F6C228 41 64 64 75 73 74 54 6F 68 65 66 47 72 6F 75 70 #djustTokenGroup					Í				
📢 Dump 1 👹 Dump 3 👹 Dump 4 👹 Dump 5 👹 Watch 1 🖉 Locals 🎾 Struct				74FBC248 76 69 6C 65 67 65 73 00 41 6C 6C 6F 63 61 74 65 vileges.Allocate 74FBC258 41 6E 64 49 6E 69 74 69 61 6C 69 7A 65 53 69 64 AndInitializeSid										
address Hey ASCTT 7					74FBC268 0 74FBC278 5	0 41 6C 6C 1	6F 63 61 74 75 65 49 64	65 4C 6F 63 61 00 41 72 65 41	6C 6C 79 . 6C 6C 41 U	AllocateL niqueId.A	ocally reAllA			
74F88128 28 C2 03 00 34 C2 03 00 50 C2 03 00 69 C2 03 00 (ÅÅPÅ1Å 74F88138 81 C2 03 00 97 C2 03 00 40 C2 03 00 50 C2 03 00 (ÅÅÅÅÅ.					74FBC2BB @	3 63 65 73	73 65 73 47	72 61 6E 74 65	64 00 41 c	cessesGra	nted.A			
74F88148 D3 C2 03 00 E9 C2 03 00 FF C2 03 00 15 C3 03 00 0Å. ėÅ. yÅÅ.					74FBCZ98 72 65 41 6E 79 41 63 63 65 73 73 65 73 47 72 61 PEANYACCESSESGRA 74FBC2A8 6E 74 65 64 00 41 72 65 46 69 6C 65 41 70 69 73 nted.AreFileApis									
74FBB158 74FBB168	36 C3 03 00 58 C3 94 C3 03 00 A5 C3	03 00 75 C3 03 00 BB C3	03 00 BF C3 0 03 00 CE C3 0	03 00 6A. [A. u			74FBC2BB 4	1 4E 53 49	00 42 61 73	65 44 6C 6C 46	72 65 65 A	NSI.BaseD	11Free	
74FBB178	DE C3 03 00 EF C3	03 00 02 C4	03 00 00 C4 0	03 00 pÅ1Å	4Ă		74FBC2C8 5 74FBC2D8 6	2 65 73 6F	75 72 63 65 70 52 65 73	49 64 00 42 61 6F 75 72 63 65	49 54 57 1	esourceId MagResou	.BaseD	
74FBB188 74FBB198	3F C4 03 00 53 C4 96 C4 03 00 A2 C4	03 00 69 C4 03 00 C6 C4	03 00 81 C4 0 03 00 FE C4 0	03 00 ?A5Ai/ 03 00 .ĢĢ	АрА		74FBC2EB 0	0 42 61 73	65 47 65 74	50 72 6F 63 65	73 73 44	BaseGetPr	ocessD	

Searching for GetProcAddress in the debugger

With **GetProcAddress** the location of **LoadLibrary** is returned. By using these two functions the packer is now able to write offsets of needed library functions into memory.

3 – Decrypt the code

In the third step the actual payload is being prepared. VirtualAlloc [4] sets up another memory area which is used to hold decrypted code temporarily. After the decryption is finished a fully unpacked PE file lies now in memory. The PE sections we started with are zero'ed and replaced with the new decrypted sections.

Some exported functions are still missing. In order to determine their position the same trick is used which I already explained in the second step. This time though, different libraries are used.



Determining position of final dependencies

4 - Returning to the payload

All that is left now is to return to the unpacked sample via return instruction because the return address is still written onto the stack.

00401A1C	50	push eax	
00401A1D	E8 E5080000	call gbot.402307	
00401A22	59	pop ecx	ecx: "PE"
00401A23	59	pop ecx	ecx: "PE"
00401A24	SE	pop edi	
00401A25	SE	non esi	
00401426	58	non ehx	
00401427	69	leave	
00401428	63	ret	
00401429	55	nush ebn	
00401424	8BEC	mov ebp.esp	
00401420	81EC 30020000	sub esp. 230	
00401432	53	nuch ehv	
00401432	55	push esi	
00401433	50	push adi	
00401434	2255	vor adi adi	
00401435	33FF 50	Nor eur, eur	
00401437	8970 F8	mov dword per ss: eop-s, edi	
00401A3A	FF15 68814000	call oword per os:[<adetcommandcinews]< th=""><th></th></adetcommandcinews]<>	
00401A40	8040 F8	Tea ecx, dword ptr ss:[ebp-8]	
00401A43	51	push ecx	ecx: "PE"
00401A44	50	push eax	
00401A45	FF15 B0B14000	call dword ptr ds:[<&CommandLineToArgvw	
00401A4B	8BF0	mov esi,eax	
00401A4D	3BF7	cmp_es1,ed1	
00401A4F	✓ 75 0C	jne qbot.401A5D	
00401A51	C745 FC 01000000	mov dword ptr ss:[ebp-4],1	
00401A58	E9 EA020000	jmp qbot.401D47	
00401A5D	57	push edi	
00401A5E	68 00000800	push 80000	
00401A63	57	push edi	
00401A64	FF15 B0B04000	call dword ptr ds: call dword ptr ds:	
00401A6A	6A 03	push 3	
00401A6C	5 B	pop ebx	

Return back to where we started at





Graph overview of start func packed



5 – loCs

Sample c23c9580f06fdc862df3d80fb8dc398b666e01a523f06ffa8935a95dce4ff8f4 SHA256