Floki Bot and the stealthy dropper

blog.malwarebytes.com/threat-analysis/2016/11/floki-bot-and-the-stealthy-dropper/

Malwarebytes Labs

November 10, 2016



Floki Bot, described recently <u>by Dr. Peter Stephenson from SC Magazine</u>, is yet another bot based on the leaked Zeus code. However, the author came up with various custom modifications that makes it more interesting.

According to the advertisements announced on the black market, this bot is capable of making very stealthy injections, evading many mechanisms of detection. We decided to take a look at what are the tricks behind it. It turned out, that although the injection method that the dropper uses is not novel by itself, but it comes with few interesting twists, that are not so commonly used in malware.

Analyzed sample

5649e7a200df2fb85ad1fb5a723bef22 - dropper <- main focus of this analysis

- <u>e54d28a24c976348c438f45281d68c54</u> core module bot 32bit
- <u>d4c5384da41fd391d16eff60abc21405</u> core module bot 64bit

NOTE: The core modules depend on a data prepared by the dropper and they crash while run independently.

The Floki Dropper

The Floki dropper looks simple and it has been found in wild without any outer protection layer. It has 3 resources with descriptive names – *bot32*, *bot64*, and *key*:

5												,						
🕅 Resource Hacker - 5e1967	7db286d886b	87d1	ec6555	59b9	af69	4fc6	=002fe	ea3a6	c7fd	13c6	b0b4	19ea	бе					
File Edit View Action	Help																RCData : BC	DT32:0
		5	Jo			Ĉ		P		7		ł	Dia Mer			>	1	
🔺 🕕 RCData	000062D8	C6	C8 96	AA	58	98 8	33 47	CE	0E	22	8B	BA	8D	8A	FE		XG"	
👾 😭 🙀 👷 👷	000062E8	5F	9D D8	Α9	A1	46 8	81 A8	F1	32	34	AD	76	B2	16	48		F 24	v H
👾 😭 BOT64 : 0	000062F8	D8	81 9A	E8	41	E9 1	L3 D2	2 78	48	25	20	22	52	1D	AC		A xH%	"R
🔤 🙀 KEY : 0	00006308	D3	35 17	B5	2B	33 2	24 59	41	80	FB	8D	B8	5A	42	40		5 +3\$YA	ZB@
	00006318	44	DE AD	61	24	74 4	11 90	: 21	1F	BE	55	47	D4	FD	84		D a\$tA ! [JG
	00006328	8F	16 28	B6	72	A7 I	08 CC) 2A	75	36	8F	99	25	32	F1		(r *u6	% 2
	00006338	34	1E 83	6F	DB	72 3	BE 58	96	6D	C4	FF	F9	75	9A	DC		4 or>X m	u
	00006348	62	14 FB	88	6D	38 I	DE 07	85	14	F5	AD	C2	Β4	E6	96		b m8	
	00006358	18	98 70	3E	17	BD 2	45 41	. 1B	8A	04	C1	A5	2B	CA	67		p> A	+ g
	00006368	CE	86 55	FF	89	01 ()2 EC	; D0	74	Α6	C8	E6	FB	7B	42		U t	{B
	00006378	30			78		78 46			B6	58	4E		5A				KN;Z
	00006388		25 31		EE		77 76						_	19		-		L 2
	00006398		57 3A			B3 1	F. F.5	6 07	F1	F9	52	BC	ΠD	DE	73		W: 1	X 3
94 731			Off. 0	, Sel	. 0													

When we try to observe its activity, we can see it making an injection into explorer.

Image:	87.41	47 184 K	29 228 K	3372	
axplorer.exe	22.66	2 296 K	5 484 K	2120 Windows Explorer	Microsoft Corporation

Indeed, when we attach the debugger to the newly created explorer process, we can see some alien code implanted – it is written on three additional memory areas with full permissions (RWE):

2025-2020 20201202 2026-2020 02031202 2025-2020 02031202 2025-2020 02031202 2025-2020 02031202 2025-2020 02031202 02120202 02032 02120202 02031202 02120202 02031202 02120202 02031202	Pri	(Volume2\Windows\System32\locale.nls (Volume2\Windows\en-US\explorer.exe.mui
00181000 00080000 explorer .text c 00281000 00008000 explorer .data d 00284000 00108000 explorer .rsrc r	Esources locations Ima Pri D Dump - 0006000000060FFF	□ □
Dump 00080000.00080FFF Dagsbaga 55 Dagsbaga 51 Dagsbaga 52 Dagsbaga 52 Dagsbaga 52 Dagsbaga 53 Dagsbaga 50 Dagsbaga 50 <	Image: SP PTR SS:[EBP-4], kernel32.Sleep S0000 JE 0 PTR SS:[EBP-4], kernel32.Sleep 30000 JE 0 PTR SS:[EBP-4], kernel32.Sleep 30000 JE 0 PTR SS:[EBP-4] MOV 30 0 HOV 30 MOV 30	EBP EBP,ESP ESP.FFFFFF0 ESP.FFFFF0 ESP.568 ESI EDI EAX.DAX 00669204 DU0RD PTR DS:LEAX+4],0 00669204 DU0RD PTR DS:LEAX+4],0 00669204 S.X110.X110 ECX.ECX PS DeWORD PTR SS:LESP+20],X110 PS DeWORD PTR SS:LESP+20],X110 PS DeWORD PTR SS:LESP+40],X110 EAX.ECX E

However, when we trace the API calls, we cannot find any reference to a function that will write the code into the explorer process. Fragment of the trace:

```
[...]
28a8;called module: C:\Windows\system32\kernel32.dll:CreateProcessW
210f;called module: C:\Windows\system32\kernel32.dll:IsWow64Process
1d94;called module: C:\Windows\SYSTEM32\ntdll.dll:ZwClose
210f;called module: C:\Windows\system32\kernel32.dll:IsWow64Process
1d94;called module: C:\Windows\SYSTEM32\ntdll.dll:ZwClose
292c;called module: C:\Windows\system32\kernel32.dll:DuplicateHandle
210f;called module: C:\Windows\system32\kernel32.dll:IsWow64Process
1d94;called module: C:\Windows\SYSTEM32\ntdll.dll:ZwClose
2a1e;called module: C:\Windows\system32\kernel32.dll:GetThreadContext
2a37;called module: C:\Windows\system32\kernel32.dll:SetThreadContext
210f;called module: C:\Windows\system32\kernel32.dll:IsWow64Process
2aa1;called module: C:\Windows\system32\kernel32.dll:WaitForSingleObject
1818;called module: C:\Windows\system32\kernel32.dll:IsBadReadPtr
182a;called module: C:\Windows\SYSTEM32\ntdll.dll:RtlFreeHeap
2aad;called module: C:\Windows\system32\kernel32.dll:ExitProcess
```

We can see that a new process is created, and it's context is being changed – that suggests manipulation – but where is the write? In order to find an answer to this question, we will take a deep dive inside the code.

Inside

At the beginning, the dropper dynamically loads some of the required imports:

```
00402679 push
                 ebp
0040267A mov
                 ebp, esp
                 esp, ØFFFFFF8h
0040267C and
                 esp, 634h
0040267F sub
00402685 push
                 ebx
00402686 push
                 esi
00402687 push
                 edi
                 load imports by hashes
00402688 call
                 ebx, ebx
0040268D xor
0040268F push
                                  ; CRC("ndll.dll") ^ 0x58E5
                 84C006A5h
00402694 mov
                 syscalls_array, ebx
0040269A mov
                 syscalls_num, ebx
                 search and open ntdll
004026A0 call
004026A5 mov
                 [esp+640h+var_62C], eax
004026A9 cmp
                 eax, ØFFFFFFFh
```

The used approach depicts, that the author was trying not to leave any artifacts that could allow for easy detection of what modules and functions are going to be used. Instead of loading DLLs by their names, it picks them enumerating all the DLLs in the system32 directory:

004013CB	Ι.	CMP EAX,ESI	
004013CD	·~	JE SHORT dropper.004013DC	
004013CF	Ι.	PUSH dropper.00401068	UNICODE "*.dll"
004013D4		LEA EAX, [LOCAL.2]	
00401307		CALL dropper.00401998	
004013DC	5	LEA EAX, [LOCAL. 151]	
004013E2	Ľ	PUSH EAX	wininet.760D0000
004013E3	1:	PUSH [LOCAL.2]	****************
004013E6	11	CALL DWORD PTR DS: [0x407FB0]	kernel32.FindFirstFileW
004013EC	11	MOV EDI.EAX	wininet.760D0000
004013EE	11	CMP EDI, -0x1	****************
004013F1		JE SHORT dropper,00401445	
004013F3	5	LEA EBX, [LOCAL.140]	
004013F9	1.	CALL dropper.004018BE	
004013FE	11	MOV ESI.EAX	wininet.760D0000
00401400	11	TEST ESI,ESI	****************
00401402	1	JE SHORT dropper.00401421	
00401404	1:	MOV ECX,ESI	
00401406	1:	CALL dropper.004019E4	
0040140B	11	PUSH EAX	wininet.760D0000
0040140C		PUSH ESI	
0040140D		CALL dropper.00401C9C	crc32
00401412		XOR EAX.0x58E5	
00401417		XOR EAX,0x58E5 CMP EAX, LARG.1]	
0040141A	.~	JE SHORT dropper.00401435	
0040141C	Ι.	CALL dropper.00401811	
00401421	× 1	LEA EAX, [LOCAL.151]	
00401427	Ι.	PUSH EAX	wininet.760D0000
00401428	Ι.	PUSH EDI	
00401429	Ι.	CALL DWORD PTR DS:[0x407F10]	kernel32.FindNextFileW
0040142F	Ι.	TEST EAX,EAX	wininet.760D0000
00401431	- ·	JNZ SHORT dropper.004013F3	
00401433	.×.	JMP SHORT dropper.00401445	
00401435	× 1	LEA EAX, [LOCAL. 140]	
0040143B	Ι.	PUSH EAX	wininet.760D0000
0040143C	Ι.	CALL DWORD PTR DS: [0x408030]	kernel32.LoadLibraryW
00401442	1.	MOV [LOCAL.3], EAX	wininet.760D0000
00401445	>	PUSH EDI	
00401446	I.	CALL DWORD PTR DS:[0x407F40]	kernel32.FindClose
0040144C	I.	MOV ESI, [LOCAL.2]	
	-		

For the sake of obfuscation, it doesn't use string comparison. Instead, it calculates a checksum of each found name. The checksum is created by CRC32 from the name XORed with some hardcoded value, that is constant for a particular sample (in the described sample it is 0x58E5):

00401404	mov	ecx, esi
00401406	call	str_len
0040140B	push	eax
0040140C	push	esi
0040140D	call	crc32
00401412	xor	eax, 58E5h

The resulting checksums are compared with the expected value, till the appropriate module is found and loaded. In similar way the export table of a particular module is enumerated and the required functions are being resolved.

After the initial imports load, exactly the same method is used to search NTDLL.DLL.

As we know, NTDLL.DLL provides an interface to execute native system calls. Every version of Windows may use a different number of a syscall in order to do the same thing. That's why it is recommended to use them via wrappers, that we can find among functions exported by NTDLL. For example, this is how the implementation of the *NtAllocateVirtualMemory* may look on Windows 7:

	Hex	Disasm
452D8	B81300000	MOV EAX, 0X13
452DD	BA0003FE7F	MOV EDX, 0X7FFE0300
452E2	FF12	CALL DWORD NEAR [EDX]
452E4	C21800	RET 0X18
452E7	90	NOP

Another variant, from Windows 8 looks a bit different:

	Hex			Disasm
6C1D0	B89B010000		MOV EAX, 0X19B	
6C1D5	E80300000	V	CALL 0X6A26C1DD	
6C1DA	C21800		RET 0X18	
6C1DD	8BD4		MOV EDX, ESP	
6C1DF	0F34		SYSENTER	
6C1E1	C3		RET	

The common part is, that the number of the syscall to be executed is moved into the EAX register.

The dropper loads NTDLL into the memory and extracts syscalls from selected functions:

- 0 : NtCreateSection
- 1 : NtMapViewOfSection
- 2 : ZwAllocateVirtualMemory
- 3 : ZwWriteVirtualMemory
- 4 : NtProtectVirtualMemory
- 5 : NtResumeThread
- 6 : ZwOpenProcess
- 7 : NtDuplicateObject
- 8 : NtUnmapViewOfSection

It checks a beginning of each function's code by comparing it with *0xB8*, that is a bytecode for moving a value into EAX:

	T
🗾 🗹 🖼	
00402003 movz	zx edx, word ptr [ebx]
00402006 mov	esi, [edi+1Ch]
00402009 lea	edx, [esi+edx*4]
0040200C mov	esi, [edx+eax]
0040200F add	esi, eax
00402011 cmp	byte ptr [esi], OB8h ; MOV EAX,imm32
00402014 jnz	short loc_40206E

If the check passed, the syscall value, that was moved into EAX, is extracted and stored in a buffer:

```
00402045 and
                 [ebp+syscall_buf], 0
00402049 push
                 ь
                                  ; 4 bytes - syscall value length
                                  ; move pointer by 1 byte
0040204B lea
                 ecx, [esi+1]
0040204E push
                 ecx
0040204F lea
                 ecx, [ebp+syscall buf]
00402052 push
                 ecx
00402053 call
                 copy bytes
                 ecx, [ebp+syscall buf]
00402058 mov
0040205B inc
                 [ebp+counter]
```

Then, when the dropper wants to call some of the functions, it uses those extracted values. The number of the syscall is fetched from the array where it was saved, and copied to EAX. Parameters of the function are pushed on the stack. The pointer to the parameters is loaded into EDX – and the syscall is triggered by with the help of an interrupt – *INT 0x2E:*

```
0040212E
0040212E make syscall proc near
0040212E
0040212E arg_4= byte ptr_8
0040212E
0040212E lea
                  esp, [esp]
00402131 lea
                  esp, [esp]
00402134 <mark>lea</mark>
                  edx, [esp+arg 4]
                                    ; DOS 2+ internal - EXECUTE COMMAND
00402138 int
                  2Eh
                                    ; DS:SI -> counted CR-terminated command string
00402138
0040213A retn
0040213A make syscall endp
```

That's how the functions *NtCreateSection*, *NtMapViewOfSection* and *NtResumeThread* are being called. Those were the missing elements of the API calls' trace, so it explains a lot!

Example 1 – dropper makes a call that is the equivalent of calling the function *NtCreateSection*:

C *G.P.U* - main thread, module dymasa										
00402710 00402713 LEA EDX, DWORD PTR SS:[ESP] 00402713 LEA EDX, DWORD PTR SS:[ESP+0x8] 00402719 00402719 SETN 00402711 SPUSH 0x0 00402721 00402721 00402721 CALL dymasa.004026FD 00402720 00402720 CALL dymasa.0040270D 00402720 CALL dymasa.0040270D 00402720 CALL dymasa.0040270D 00402720 CALL dymasa.0040270D	NtCreateSection	4	Registers (FPU) EAX 00000054 ECX 002500F8 EDX 0012F874 EBX 0000000 ESP 0012F86C EBF 0012F884 ESI 0000000 ESI 0000000 EDI 0012F914							
0012F874 0012F914 0012F878 0000000E 0012F87C 00000000 0012F880 0012F900			EIP 00402717 dymasa.00402717							
0012F884 00000040 0012F882 0800000 0012F882 0800000 0012F890 00000000 0012F894 00000000 0012F894 00000000 0012F898 0000000										

Example 2 – the dropper mapped a section by using a syscall – it is an equivalent of calling the function *NtMapViewOfSection*:

Odd402713 00402717 00402719 INT 0x2E NtMapUiewOfSecton EDX FFFF EBX 00121 ESX 00121 ESX 00121 00402719 00402719 . RETN . <	C *G.P.U*	* - main th	read, modu	ile dymasa	1							
Address Size Owner Section Contains Type Access Initial acc 00120000 00067000 Map 00041002 R R 00230000 00090000 Map 00041002 R R 00250000 00090000 Map 00041002 R R 00250000 00003000 Map 00041002 R R 00300000 00000000 Priv 00021004 RW RW	0040270D 00402710 00402713 00402713 00402717 00402719	LEA LEA LEA LEA INT	ESP, DWORD ESP, DWORD EDX, DWORD 0x2E	PTR SS:[] PTR SS:[] PTR SS:[]	ESP] ESP] ESP+Øx8]	NtMapViewOfSec	ton			EAX 00 ECX 00 EDX FF EBX 00 ESP 00	000000 EA0001 FFFFFF 12F910 12F844	
001C0000 00067000 Map 00041002 R R 00220000 00009000 Map 00041002 R R 002F0000 00003000 Map 00041002 R R 00250000 00003000 Map 00041002 R R 00250000 00000000 Map 00041002 R R 00300000 Priv 00021004 RW RW			0				T	0	1	1		M-
00430000 000010000 dynasa .rstc resources Imag 01001002 R RWE 00440000 00001000 .reloc Imag 01001002 R RWE 00550000 00001000 .reloc Imag 01001002 R RWE 00550000 00001000 .reloc Map 00041002 R RWE 00550000 00001000 .reloc Map 00041040 RWE RWE 00550000 00003000 .reloc Map 00041040 RWE RWE 000570000 00003000 .reloc Map 00041040 RWE RWE 000570000 00003000 .reloc .reloc Map 00041040 RWE RWE 000570000 00003000 .reloc .reloc .reloc .reloc .reloc .reloc .reloc 000570000 00003000 .reloc .reloc .reloc .reloc .reloc 000570000 .reloc .reloc .reloc .reloc .reloc .reloc .reloc	001C0000 00250000 002F0000 00400000 00401000 00405000 00405000 00405000 0043D000 0043D000 0043D000 0043D000 00550000 00550000 005570000		dymasa dymasa dymasa dymasa dymasa	.text .data .rsrc	PE header SFX,code,im	nports	Map 00041002 Map 00041002 Map 00041002 Priv 00021004 Imag 01001002 Imag 01001002 Imag 01001002 Imag 01001002 Imag 01001002 Priv 00021004 Map 00041040 Priv 00021004	R R R R R R R R R R R R R R R R R R R			access	

Once the memory is prepared, the shellcode is copied there:

00401502	· FUOR EHA	
004015C3	. PUSH DWORD PTR SS:[ESP+0x58]	
004015C7	DUOU DUODD DTD OO STODYOUDOD	ntdll.777C6570
	. POSH DWORD PTR SSILESP+08281	ntall.rrrbbro
004015CB	. PUSH -0x1	
004015CD	. CALL DWORD PTR DS:[0x4071B4]	kernel32.DuplicateHandle
004015D3	. PUSH 0x2	
004015D5	PUSH EBX	
004015D6		
004015D7	LEA EAX, DWORD PTR SS:[ESP+0xC8]	
004015DE	. PUSH EAX	
004015DF	PUSH DWORD PTR SS:[ESP+0x58]	
004015E3	. PUSH DWORD PTR SS:[ESP+0x70]	
004015E7	. PUSH -0x1	
004015E9	2011 DU0DD DTD D0 50 4071043	kernel32.DuplicateHandle
	DUOL OUT	kernetsz.bupticatenanute
004015EF	. PUSH_0x4	
004015F1	. POP EAX	
004015F2	. PUSH EAX	
004015F3	. PUSH dymasa.00407170	
004015F8	. LEA ECX, DWORD PTR SS: [ESP+0x27]	
	DUCH ECY	kernel32 771BEBE7
004015FC	DUCH ECY	kernel32.771BEBF7
004015FC 004015FD	DUCH ECY	kernel32.771BEBF7 the hook content
004015FC 004015FD 00401605	DUCH ECY	
004015FC 004015FD 00401605 00401605	DUCH ECY	
004015FC 004015FD 00401605 0040160D 0040160D 00401615	DUCH ECY	
004015FC 004015FD 00401605 0040160D 0040160D 00401615	DUCH ECY	
004015FC 004015FD 00401605 00401605 00401605 00401615 00401619	DUCH ECY	
004015FC 004015FD 00401605 00401605 00401615 00401615 00401619 00401621	DUCH ECY	
004015FC 004015FD 00401605 00401605 00401615 00401619 00401621 00401629	 PUSH ECX MOV DWORD PTR SS: [ESP+0x24],0x51EC8855 MOV DWORD PTR SS: [ESP+0x28],0xFC45C7 MOV DWORD PTR SS: [ESP+0x20],0x68000000 MOV DWORD PTR SS: [ESP+0x30],EBX MOV DWORD PTR SS: [ESP+0x30],0xC7FC55FF MOV DWORD PTR SS: [ESP+0x30],0xFC45 MOV DWORD PTR SS: [ESP+0x30],0x680000 	
004015FC 004015FD 00401605 00401605 00401615 00401619 00401621 00401629 00401621	 PUSH ECX MOV DWORD PTR SS:[ESP+0x24],0x51EC8855 MOV DWORD PTR SS:[ESP+0x28],0xFC45C7 MOV DWORD PTR SS:[ESP+0x2C],0x68000000 MOV DWORD PTR SS:[ESP+0x34],0xC7FC55FF MOV DWORD PTR SS:[ESP+0x38],0xFC45 MOV DWORD PTR SS:[ESP+0x38],0xFC45 MOV DWORD PTR SS:[ESP+0x30],0x650000 MOV DWORD PTR SS:[ESP+0x30],0xFC45 MOV DWORD PTR SS:[ESP+0x30],0xFC45 MOV DWORD PTR SS:[ESP+0x30],0xFF040000 	
004015FC 004015FD 0040160D 0040160D 00401615 00401619 00401621 00401629 00401631 00401639	<pre>PUSH ECX MOV DWORD PTR SS:[ESP+0x24],0x51EC8855 MOV DWORD PTR SS:[ESP+0x28],0xFC45C7 MOV DWORD PTR SS:[ESP+0x20],0x68000000 MOV DWORD PTR SS:[ESP+0x30],EBX MOV DWORD PTR SS:[ESP+0x30],0xFC45 MOV DWORD PTR SS:[ESP+0x30],0xFC45 MOV DWORD PTR SS:[ESP+0x40],0xFF000000 MOV DWORD PTR SS:[ESP+0x44],0xC483FC55</pre>	
004015FC 004015FD 00401605 00401605 00401605 00401615 00401619 00401621 00401629 00401639 00401639	 PUSH ECX MOV DWORD PTR SS: [ESP+0x24],0x51EC8855 MOV DWORD PTR SS: [ESP+0x28],0xFC45C7 MOV DWORD PTR SS: [ESP+0x20],0x68000000 MOV DWORD PTR SS: [ESP+0x34],0xC7FC55FF MOV DWORD PTR SS: [ESP+0x38],0xFC45 MOV DWORD PTR SS: [ESP+0x30],0xFC45 MOV DWORD PTR SS: [ESP+0x40],0xFF000000 MOV DWORD PTR SS: [ESP+0x40],0xFF000000 MOV DWORD PTR SS: [ESP+0x44],0xC483FC55 MOV DWORD PTR SS: [ESP+0x44],0x5DE58804 	
004015FC 004015FD 0040160D 0040160D 00401615 00401619 00401621 00401629 00401631 00401639	 PUSH ECX MOV DWORD PTR SS: [ESP+0x24],0x51EC8855 MOV DWORD PTR SS: [ESP+0x28],0xFC45C7 MOV DWORD PTR SS: [ESP+0x20],0x68000000 MOV DWORD PTR SS: [ESP+0x34],0xC7FC55FF MOV DWORD PTR SS: [ESP+0x38],0xFC45 MOV DWORD PTR SS: [ESP+0x30],0xFC45 MOV DWORD PTR SS: [ESP+0x40],0xFF000000 MOV DWORD PTR SS: [ESP+0x40],0xFF000000 MOV DWORD PTR SS: [ESP+0x44],0xC483FC55 MOV DWORD PTR SS: [ESP+0x44],0x5DE58804 	
004015FC 004016P5 00401605 00401605 00401615 00401615 00401619 00401621 00401629 00401631 00401631 00401641 00401649	<pre>PUSH ECX MOV DWORD PTR SS:[ESP+0x24],0x51EC8B55 MOV DWORD PTR SS:[ESP+0x28],0xFC45C7 MOV DWORD PTR SS:[ESP+0x30],EBX MOV DWORD PTR SS:[ESP+0x30],EBX MOV DWORD PTR SS:[ESP+0x33],0xC7FC55FF MOV DWORD PTR SS:[ESP+0x32],0xFC45 MOV DWORD PTR SS:[ESP+0x40],0xFF000000 MOV DWORD PTR SS:[ESP+0x44],0xC483FC55 MOV DWORD PTR SS:[ESP+0x44],0xC483FC55 MOV DWORD PTR SS:[ESP+0x42],0xC3 MOV DWORD PTR SS:[ESP+0x42],0xC3 MOV DWORD PTR SS:[ESP+0x42],0xC3 MOV DWORD PTR SS:[ESP+0x42],0xC483FC55 MOV DWORD PTR SS:[ESP+0x42],0xC3 MOV DWORD PTR SS:[ESP+0x42],0xC3 MOV DWORD PTR SS:[ESP+0x42],0xC3</pre>	
004015FC 004015FD 00401605 00401605 00401615 00401619 00401621 00401629 00401631 00401639 00401639 00401649 00401649 00401649	<pre>PUSH ECX MOV DWORD PTR SS:[ESP+0x24],0x51EC8855 MOV DWORD PTR SS:[ESP+0x28],0xFC45C7 MOV DWORD PTR SS:[ESP+0x20],0x68000000 MOV DWORD PTR SS:[ESP+0x34],0xC7FC55FF MOV DWORD PTR SS:[ESP+0x38],0xFC45 MOV DWORD PTR SS:[ESP+0x30],0xF040000 MOV DWORD PTR SS:[ESP+0x40],0xF040000 MOV DWORD PTR SS:[ESP+0x44],0xC483FC55 MOV DWORD PTR SS:[ESP+0x44],0xC483FC55 MOV DWORD PTR SS:[ESP+0x42],0xC38000 MOV DWORD PTR SS:[ESP+0x40],0xC3 MO</pre>	
004015FC 004015FD 00401605 00401605 00401605 00401615 00401621 00401621 00401621 00401631 00401639 00401641 00401645 00401645 00401653	<pre>. PUSH ECX . MOV DWORD PTR SS: LESP+0x24], 0x51EC8B55 . MOV DWORD PTR SS: LESP+0x28], 0xFC45C7 . MOV DWORD PTR SS: LESP+0x30], EBX . MOV DWORD PTR SS: LESP+0x30], EBX . MOV DWORD PTR SS: LESP+0x34], 0xC7FC55FF . MOV DWORD PTR SS: LESP+0x33], 0xFC45 . MOV DWORD PTR SS: LESP+0x3C], 0x6500000 . MOV DWORD PTR SS: LESP+0x40], 0xFF000000 . MOV DWORD PTR SS: LESP+0x44], 0xC483FC55 . MOV DWORD PTR SS: LESP+0x40], 0xF5000000 . MOV DWORD PTR SS: LESP+0x40], 0x5DE58B04 . MOV DWORD PTR SS: LESP+0x40], 0xC3 . CALL dymasa, 00401E10 . PUSH EAX</pre>	
004015FC 00401605 00401605 00401605 00401615 00401619 00401629 00401629 00401631 00401639 00401641 00401649 00401645 00401653 00401654	<pre>PUSH ECX MOV DWORD PTR SS: LESP+0x24], 0x51EC8B55 MOV DWORD PTR SS: LESP+0x28], 0xFC45C7 MOV DWORD PTR SS: LESP+0x30], EBX MOV DWORD PTR SS: LESP+0x30], EBX MOV DWORD PTR SS: LESP+0x34], 0xC7FC55FF MOV DWORD PTR SS: LESP+0x30], 0xFC45 MOV DWORD PTR SS: LESP+0x30], 0xFF000000 MOV DWORD PTR SS: LESP+0x40], 0xFF000000 MOV DWORD PTR SS: LESP+0x40], 0xFF000000 MOV DWORD PTR SS: LESP+0x40], 0xF5055804 MOV DWORD PTR SS: LESP+0x40], 0xC3 MOV DWORD PTR SS: LESP+0x40], 0xC3 CRLL dymasa, 00401E10 PUSH EAX LEA ECX, DWORD PTR SS: LESP+0x48] PUSH EAX</pre>	the hook content
004015FC 004015FD 00401605 00401605 00401615 00401619 00401621 00401621 00401621 00401631 00401631 00401639 00401649 00401649 00401654 00401654	<pre>PUSH ECX MOV DWORD PTR SS:[ESP+0x24],0x51EC8855 MOV DWORD PTR SS:[ESP+0x28],0xFC45C7 MOV DWORD PTR SS:[ESP+0x20],0x68000000 MOV DWORD PTR SS:[ESP+0x30],EBX MOV DWORD PTR SS:[ESP+0x30],0xC7FC55FF MOV DWORD PTR SS:[ESP+0x40],0xFC45 MOV DWORD PTR SS:[ESP+0x40],0xFF000000 MOV DWORD PTR SS:[ESP+0x40],0xFF000000 MOV DWORD PTR SS:[ESP+0x40],0xC483FC55 MOV DWORD PTR SS:[ESP+0x40],0xC582804 MOV BYTE PTR SS:[ESP+0x40],0xC3 CALL dymasa.00401E10 PUSH EAX LEA ECX,DWORD PTR SS:[ESP+0x48] PUSH ECX</pre>	
004015FC 004016P5 00401605 00401605 00401615 00401615 00401621 00401629 00401631 00401631 00401639 00401641 00401645 00401645 00401655	<pre>PUSH ECX MOV DWORD PTR SS: LESP+0x24], 0x51EC8B55 MOV DWORD PTR SS: LESP+0x28], 0xFC45C7 MOV DWORD PTR SS: LESP+0x30], EBX MOV DWORD PTR SS: LESP+0x30], EBX MOV DWORD PTR SS: LESP+0x34], 0xC7FC55FF MOV DWORD PTR SS: LESP+0x30], 0xFC45 MOV DWORD PTR SS: LESP+0x30], 0xFF000000 MOV DWORD PTR SS: LESP+0x40], 0xFF000000 MOV DWORD PTR SS: LESP+0x40], 0xFF000000 MOV DWORD PTR SS: LESP+0x40], 0xF5055804 MOV DWORD PTR SS: LESP+0x40], 0xC3 MOV DWORD PTR SS: LESP+0x40], 0xC3 CRLL dymasa, 00401E10 PUSH EAX LEA ECX, DWORD PTR SS: LESP+0x48] PUSH EAX</pre>	the hook content

After the preparations, those sections are mapped into the context of the explorer process, that has been created as suspended. Using *SetThreadContext*, it's Entry Point is being redirected to the injected memory page. When the explorer process is being resumed, the new code executes and proceeds with unpacking the malicious core.

At this point of the injection, it's malicious core is not yet revealed – it's decryption process takes place inside the shellcode implanted in the *explorer*. This is also additional countermeasure that this dropper takes against detection tools.

Another trick that this bot uses, is a defense against inline hooking – a method utilized by various monitoring tools. All the mapped DLLs are compared with their raw versions, read from the disk by the dropper. If any anomaly is detected, the dropper overwrites the mapped DLL by the code copied from it's raw version. As a results, the functions are getting

"unhooked" and the monitoring programs are loosing the trace on the executed calls. Example from Cuckoo – the unhooking procedure was executed after calling *NtGetThreadContext* – as a result the sandbox lost control over executed calls:

2016-11-07 04:39:06,453	CreateProcessInternalW	ApplicationName: C:\WINDOWS \explorer.exe ProcessId: 1924 CommandLine: ThreadHandle: 0x000000c4 ProcessHandle: 0x000000c0 ThreadId: 580 CreationFlags: 0x0800004	SUCCESS
2016-11-07 04:39:06,453	NtGetContextThread	ThreadHandle: 0x000000c4	success
2016-11-07 04:39:06,674	anomaly	ThreadIdentifier: 584 Subcategory: unhook Message: Function was unhooked/restored! FunctionName: LdrLoadDll	success
2016-11-07 04:39:06,674	anomaly	ThreadIdentifier: 584	success

Conclusion

The illustrated concept is not novel, however it was utilized in an interesting way. Many programs detect malicious activity by monitoring API calls, that are most often misused by malware. Also, applications used for automated analysis hooks API functions, in order to monitor where and how they are being used. The presented method allows to bypass them – at the same time being relatively easy to implement.

In this case, the author didn't use the full potential of the technique, because he could have implement all the injection-related functions via direct syscalls – instead, he chose to use only some subset, related to writing into remote memory area. Some other syscalls has been loaded but not used – it may suggest that the product is still under development. Creation of the new process and changing it's context still could be detected via API monitoring – and it was enough to rise alerts and make the dropper less stealthy than it was intended.

Appendix

<u>https://www.evilsocket.net/2014/02/11/on-windows-syscall-mechanism-and-syscall-numbers-extraction-methods</u>/ – On Windows Syscall Mechanism and Syscall Numbers Extraction Methods

This was a guest post written by Hasherezade, an independent researcher and programmer with a strong interest in InfoSec. She loves going in details about malware and sharing threat information with the community. Check her out on Twitter @hasherezade and her personal blog: <u>https://hshrzd.wordpress.com</u>.