A chain is only as strong as its weakest Win32k

Sam Brown

SteelCon 2017
whoami

+ Sam Brown – @_samdb_

+ Consultant in the research practice @ MWR


+ Research/home time – poking at Windows/driver internals, playing with Angr and Z3
Introduction

- Survey style – no 1337 0day
- Focused on concepts
- Based off past year of reading, reversing and poking at kernel/driver bugs
- References at end but all of the things here: https://github.com/sam-b/windows_kernel_resources
Outline
1. Motivation
2. The Attack Surface
3. Bug Hunting
4. Mitigations
5. CVE-2016-7255
6. Conclusions & Questions
**Motivation – Sandboxes**

“a virtual space in which new or untested software or coding can be run securely.”
Motivation – Sandboxes

+ Started appearing in 2006 with IE 7 protected mode
+ Low Integrity processes
+ Increasingly prevalent

Firefox takes the next step towards rolling out multi-process to everyone

Firefox gets closer to offering the same security and stability as competition.

PETER BRIGHT (US) - 22/12/2016, 05:15
Motivation – Sandbox Escapes

+ Compromised a client but sandbox containing us
+ EoP exploit required
+ Sandbox broker exploit – limited attack surface but possible
++ Motivation – Sandbox Escapes

+ Kernel – straight to the core, massive attack surface

Background

- We want to escalate our privileges
- Low Integrity to SYSTEM
- How?
Background

- Windows has Access Token objects
- Think cookies for users
- Many methods of privescing
- Steal the Access Token from a process running as SYSTEM
- Modify users token to have permissions to inject code into a process running as SYSTEM
- Overwrite a SYSTEM processes security descriptor with NULL
Outline
1. Motivation
2. The Attack Surface
3. Bug Hunting
4. Mitigations
5. CVE-2016-7255
6. Conclusions & Questions
The Attack Surface

- System calls
- Drivers
- Font Parsing
Ntdll.dll
User32.dll
Gdi32.dll
Win32u.dll
Ntoskrnl.exe
~449 system calls
Win32k.sys
~1138 system calls
Applications
System DLL’s
Drivers
Ring 0
Ring 3
System Calls
https://github.com/sam-b/windows_syscalls_dumper
++ win32k
+ Main Windows graphics driver
+ Lots of complex functionality
+ Written in the 90’s
+ All in kernel mode
+ “How bad design decisions created the least secure driver on Windows” by Thomas Garnier[1]
ntoskrnl

+ Windows kernel executive

+ Implements core functionality:
  • Processes, Threads
  • Virtual Memory
  • The Registry
++

ntoskrnl

• A fraction of the system call count Win32k has
• Less than half the number of CVE’s
• Still lots of bugs to be found
++ Drivers

- Interact with hardware
- Firmware updaters
- Antivirus
- Anti-Cheat
Driver Communications

- Many ways, bugs mostly in...

- IOCTL codes – triggers a function within the driver, identified by a number – input buffer pointer and size and output buffer pointer and size sent

- Shared memory – mapped memory shared between user mode and kernel mode, allows for fast data exchange
Third party drivers do terrible things

RTCore64.sys
RivaTuner[5]

ASMMAP.sys – ASUS[6]

NTIO.sys - MSI[5]

Winlo.sys - internals.com[5]
Font Parsing

- Fonts are actually super complex
- Include small instruction sets
- Win32k is responsible for parsing TrueType and OpenType fonts
Outline

1. Motivation
2. The Attack Surface
3. Bug Hunting
4. Mitigations
5. CVE-2016-7255
6. Conclusions & Questions
++

Kernel Fuzzing

+ MWR ♥’s kernel fuzzing
+ https://github.com/mwrlabs/KernelFuzzer
Kernel Fuzzing – general work flow:

1. Select library/system call from catalogue
2. Generate fuzzed values for primitives
3. Grab random Handles from HandleDB if needed
4. Log arguments and call
5. Execute
6. Saves any returned Handles in HandleDB
7. GOTO 1;
Kernel Fuzzing

- All of the bugs:

<table>
<thead>
<tr>
<th>Bug Description</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows kernel: use-after-free in bitmap handling</td>
<td>CCPProjectZeroMembers</td>
</tr>
<tr>
<td>Windows kernel: NULL pointer dereference with window station and clipboard</td>
<td>CCPProjectZeroMembers</td>
</tr>
<tr>
<td>Windows kernel: use-after-free in WindowsStation</td>
<td>CCPProjectZeroMembers</td>
</tr>
<tr>
<td>Windows kernel: Brush object Use-after-free vulnerability</td>
<td>CCPProjectZeroMembers</td>
</tr>
<tr>
<td>Windows kernel: use-after-free in bitmap handling #2</td>
<td>CCPProjectZeroMembers</td>
</tr>
<tr>
<td>Windows kernel: possible NULL pointer dereference of a SURFOBJ</td>
<td>CCPProjectZeroMembers</td>
</tr>
<tr>
<td>Windows kernel: buffer overflow in win32kSolidFillRect</td>
<td>CCPProjectZeroMembers</td>
</tr>
<tr>
<td>Windows kernel: use-after-free in HmgAllocateObjectAttr</td>
<td>CCPProjectZeroMembers</td>
</tr>
<tr>
<td>Windows kernel: pool buffer overflow drawing caption bar</td>
<td>CCPProjectZeroMembers</td>
</tr>
<tr>
<td>Windows kernel: use-after-free with UserCommitDesktopMemory</td>
<td>CCPProjectZeroMembers</td>
</tr>
<tr>
<td>Windows kernel: DeferWindowPos use-after-free</td>
<td>CCPProjectZeroMembers</td>
</tr>
<tr>
<td>Windows kernel: pool buffer overflows in NtGdiStretchBit</td>
<td>CCPProjectZeroMembers</td>
</tr>
<tr>
<td>Windows kernel: use-after-free with printer device contexts</td>
<td>CCPProjectZeroMembers</td>
</tr>
<tr>
<td>Windows kernel: use-after-free with cursor object</td>
<td>CCPProjectZeroMembers</td>
</tr>
<tr>
<td>Windows kernel: use-after-free in bGetRealizedBrush</td>
<td>CCPProjectZeroMembers</td>
</tr>
<tr>
<td>Windows kernel: buffer overflow in NtGdiBltBlt</td>
<td>CCPProjectZeroMembers</td>
</tr>
<tr>
<td>Windows kernel: FlashWindowEx memory corruption</td>
<td>CCPProjectZeroMembers</td>
</tr>
<tr>
<td>Windows kernel: use-after-free with device contexts and NtGdiSelectBitmap</td>
<td>CCPProjectZeroMembers</td>
</tr>
<tr>
<td>Windows kernel: NtUserScrollDC memory corruption</td>
<td>CCPProjectZeroMembers</td>
</tr>
<tr>
<td>Windows race condition leading to use after free in DestroySMVP</td>
<td>CCPProjectZeroMembers</td>
</tr>
<tr>
<td>Windows Cursor object potential memory leak</td>
<td>CCPProjectZeroMembers</td>
</tr>
<tr>
<td>Windows ndis.sys IOCTL 0x170034 (ndisIndisNsiGetNIFNameForIndex) pool buffer overflow</td>
<td>CCPProjectZeroMembers</td>
</tr>
<tr>
<td>win32k clipboard Bitmap use-after-free vulnerability</td>
<td>CCPProjectZeroMembers</td>
</tr>
<tr>
<td>win32k null pointer dereference with Desktop and Clipboard</td>
<td>CCPProjectZeroMembers</td>
</tr>
<tr>
<td>Windows kernel null pointer dereference in win32k!OffsetChildren</td>
<td>CCPProjectZeroMembers</td>
</tr>
<tr>
<td>Windows kernel: NtGdiGetTextExtentExW out-of-bounds memory read</td>
<td>CCPProjectZeroMembers</td>
</tr>
<tr>
<td>Windows kernel: bitmap use-after-free</td>
<td>CCPProjectZeroMembers</td>
</tr>
<tr>
<td>Windows kernel: DrawMenuBarTemp wild-write on 64-bit</td>
<td>CCPProjectZeroMembers</td>
</tr>
<tr>
<td>Windows 7 win32k bitmap use-after-free (1)</td>
<td>CCPProjectZeroMembers</td>
</tr>
<tr>
<td>Windows 7 win32k bitmap use-after-free (2)</td>
<td>CCPProjectZeroMembers</td>
</tr>
</tbody>
</table>
Code Review

+ Generally everything's closed source
+ A few exceptions…

The Windows vulnerability is a local privilege escalation in the Windows kernel that can be used as a security sandbox escape. It can be triggered via the win32k.sys system call NtSetWindowLongPtr() for the index GWP_ID on a window handle with GWL_STYLE set to WS_CHILD. Chrome's sandbox blocks win32k.sys system calls using…

Alex Ionescu @aionescu 2 Nov 2016

Here's your Microsoft "actively exploited" Google-disclosed bug right here. Source: GitHub, where people post NT4 source (try Google Search) pic.twitter.com/touOvjqac
Reverse Engineering

- Supports other techniques
- A lot of Windows binaries have debugging symbols on Microsoft’s symbol server which helps
- ReactOS helps
- Narrowly targeted might be successful
- Kernel is huge, fuzzers still easily find bugs, why bother?
Reverse Engineering

- Reversing Third Party drivers has been a good source of bugs
- Much smaller binaries, lower code quality
- Tools to help:
  - My IDA plugin: [https://github.com/mwrlabs/win_driver_plugin](https://github.com/mwrlabs/win_driver_plugin)
  - NCC Group’s: [https://github.com/nccgroup/DriverBuddy](https://github.com/nccgroup/DriverBuddy)
Driver Fuzzing

- Reverse driver to find IOCTL codes
- Randomly fuzz them
- iSEC’s driver fuzzer:
  [https://github.com/iSECPartners/DIBF](https://github.com/iSECPartners/DIBF)
++

Font Fuzzing/j00ru is a machine

+ J00ru has been hitting this heavily for years[2]
+ Specs are publically available
+ Targeted fuzzing with custom fuzzers
**Patch Diffing**

- One day bugs
- Diff kernel code before/after patch Tuesday
- CVE details and patch notes give hints[7]

---

CVE-2014-4113

New pointer check
Outline

1. Motivation
2. The Attack Surface
3. Bug Hunting
4. Mitigations
5. CVE-2016-7255
6. Conclusions & Questions
Mitigations

- **Type 0 - Strong Mitigation**
  End a bug class.

- **Type 1 - Weak Mitigation**
  End an exploitation technique.

- **Type 2 - Attack Surface Reduction**
  Remove a set of exposed functionality.

- **Type 3 - Chain Extension**
  Increase the number of bugs required in an exploit.
Mitigations

- Many mitigations in modern Windows
- Only covering a few key/interesting ones
- Being added to Windows 10 rapidly
Once upon a time…

- Kernel memory marked NX
- Map shellcode in usermode
- Control flow hijacking exploit? Jump to it
- Write–What–Where? Overwrite an entry in a function table to point at it
++ SMEP

+ Supervisor Mode Execution Prevention
+ Introduced with Intel Ivy Bridge Processors ~April 2012
+ First supported in Windows 8
+ Causes a BSOD on kernel mode attempting to execute user mode memory
+ Type 1 Mitigation
Bypasses

- Data only attacks
- Return Oriented Programming
- Or...
Double KO! Capcom's *Street Fighter V* installs hidden rootkit on PCs

Fatality – wait, no, what? That's the other game

```asm
lea   rax, disable_smep
lea   rcx, [rsp+48h+var_28]
call  rax; disable_smep
mov   rcx, [rsp+48h+var_18]
call  [rsp+48h+var_20]; execute shellcode \o/
lea   rax, enable_smep
lea   rcx, [rsp+48h+var_28]
call  rax; enable_smep
```
KASLR

+ Kernel Address Space Layout Randomisation
+ Randomizes addresses objects are loaded at
+ Introduced in Vista, potentially a type 3 mitigation
+ Randomness++ since
++

KASLR – Address Leaks

+ NtQuerySystemInformation

+ Undocumented function for getting information about the system
KASLR – Address Leaks

SystemHandleInformation

<table>
<thead>
<tr>
<th>PID</th>
<th>Object</th>
<th>Handle</th>
</tr>
</thead>
<tbody>
<tr>
<td>3072</td>
<td>0x84B50AE8</td>
<td>0xB8</td>
</tr>
<tr>
<td>3072</td>
<td>0x845DFF0</td>
<td>0xBC</td>
</tr>
<tr>
<td>3072</td>
<td>0x847E68B8</td>
<td>0xC0</td>
</tr>
<tr>
<td>3072</td>
<td>0x85A65C18</td>
<td>0xC4</td>
</tr>
<tr>
<td>3072</td>
<td>0xA519F558</td>
<td>0xC8</td>
</tr>
<tr>
<td>3072</td>
<td>0x963047E0</td>
<td>0xCC</td>
</tr>
<tr>
<td>3072</td>
<td>0x8463B8A0</td>
<td>0xD0</td>
</tr>
<tr>
<td>264</td>
<td>0xEE9F838</td>
<td>0x4</td>
</tr>
<tr>
<td>264</td>
<td>0x8544DAF8</td>
<td>0x8</td>
</tr>
<tr>
<td>264</td>
<td>0x85A9DD90</td>
<td>0xC</td>
</tr>
<tr>
<td>264</td>
<td>0x85A9F00</td>
<td>0x10</td>
</tr>
<tr>
<td>264</td>
<td>0x84A9B038</td>
<td>0x14</td>
</tr>
<tr>
<td>264</td>
<td>0x78265898</td>
<td>0x18</td>
</tr>
<tr>
<td>264</td>
<td>0xB2AA7030</td>
<td>0x1C</td>
</tr>
</tbody>
</table>
# KASLR – Address Leaks

## SystemModuleInformation

<table>
<thead>
<tr>
<th>Module name</th>
<th>Base Address</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>\SystemRoot\System32\DRIVERS\srv.sys</code></td>
<td>0x96F85000</td>
</tr>
<tr>
<td><code>\SystemRoot\System32\drivers\spsys.sys</code></td>
<td>0xA2E02000</td>
</tr>
<tr>
<td><code>\SystemRoot\System32\Drivers\BTHUSB.sys</code></td>
<td>0xA2E6C000</td>
</tr>
<tr>
<td><code>\SystemRoot\System32\Drivers\bthport.sys</code></td>
<td>0xA2E7E000</td>
</tr>
<tr>
<td><code>\SystemRoot\System32\Drivers\rfcomm.sys</code></td>
<td>0xA2EE2000</td>
</tr>
<tr>
<td><code>\SystemRoot\System32\Drivers\BthEnum.sys</code></td>
<td>0xA2F06000</td>
</tr>
<tr>
<td><code>\SystemRoot\System32\Drivers\bthpan.sys</code></td>
<td>0xA2F13000</td>
</tr>
<tr>
<td><code>\Windows\System32\ntdll.dll</code></td>
<td>0x77810000</td>
</tr>
<tr>
<td><code>\Windows\System32\smss.exe</code></td>
<td>0x47AF0000</td>
</tr>
<tr>
<td><code>\Windows\System32\apisetschema.dll</code></td>
<td>0x77A50000</td>
</tr>
<tr>
<td><code>\Windows\System32\autochk.exe</code></td>
<td>0x330000</td>
</tr>
</tbody>
</table>

C:\Users\sam\Documents\Visual Studio 2015\Projects\NtQuerySysInfo_SystemModuleInformation\Debug\NtQuerySysInfo_SystemModuleInformation.exe
KASLR – Address Leaks

Windows 8.1, Low Integrity 😞

C:\Users\sam\Desktop\windows_kernel_address_leaks\windows_kernel_address_leaks\NtQuerySysInfo_SystemLockInformation\x64\Debug\NtQuerySysInfo_SystemLockInformation.exe
NtQuerySystemInformation failed with error code 0xC0000022
<table>
<thead>
<tr>
<th>Technique</th>
<th>7</th>
<th>8</th>
<th>8.1</th>
<th>10 - 1511</th>
<th>10 - 1607</th>
<th>10 - 1703</th>
<th>10 - 1703 + VBS</th>
</tr>
</thead>
<tbody>
<tr>
<td>NtQuerySystemInformation:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SystemHandleInformation</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SystemLockInformation</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SystemModuleInformation</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SystemProcessInformation</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SystemBigPoolInformation</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>System Call Return Values</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Win32k Shared Info User Handle Table</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Descriptor Tables</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>HMValidateHandle</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GdiSharedHandleTable</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>DesktopHeap</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
</tbody>
</table>

NULL Page Mapping

- NULL pointer deference's
- Super common C/C++ coding error
- Map the NULL page from user mode
- Manipulate kernel control flow by customising the data you control
- Gone as of Windows 7 64 bit
- Type 0 mitigation
NULL Security Descriptor Protection

- SecurityDescriptor field header == NULL?
- Is it a process object?
- SecurityRequired flag set?
- Nettitude did an awesome writeup[3]
- Type 1 mitigation
Moving Font Parsing out of the kernel

- Windows 10 anniversary update
- Font parsing now done in an AppContainer[4][9]
- Type 2 mitigation
Win32k Lockdown

- Stop processes using win32k[8]
- Type 2 mitigation
Virtualisation Based Security (VBS)

- Delivered with Device Guard
- kCFG, W^X, HVCI, KMCI, NPIEP
NPIEP

From the Hyper-V functional specification:

Non-Privileged Instruction Execution (NPIEP) is a feature that limits the use of certain instructions by user-mode code. Specifically, when enabled, this feature can block the execution of the SIDT, SGDT, SDLT, and STR instructions. Execution of these instructions results in a #GP fault.

Key OS data structures

Example: Overwrite IDT entries to hook keystrokes, network packets, etc

SGDT is only useful in operating-system software; however, it can be used in application programs without causing an exception to be generated.
++

NPIEP

+ Trap on targeted instructions and return hardcoded or sanitized data
+ Intel implementing this in hardware

User-Mode Instruction Prevention (bit 11 of CR4) — When set, the following instructions cannot be executed if CPL > 0: SGDT, SIDT, SLDT, SMSW, and STR. An attempt at such execution causes a general-protection exception (#GP).

https://github.com/sam-b/windows_kernel_address_leaks/blob/master/notes/NPIEP.md
Outline

1. Motivation
2. The Attack Surface
3. Bug Hunting
4. Mitigations
5. CVE-2016-7255
6. Conclusions & Questions
++ CVE-2016-7255/MS16-135

Disclosing vulnerabilities to protect users
October 31, 2016

Posted by Neel Mehta and Billy Leonard, Threat Analysis Group

On Friday, October 21st, we reported 0-day vulnerabilities — previously publicly-unknown vulnerabilities — to Adobe and Microsoft. Adobe updated Flash on October 26th to address CVE-2016-7855; this update is available via Adobe’s updater and Chrome auto-update.

After 7 days, per our published policy for actively exploited critical vulnerabilities, we are today disclosing the existence of a remaining critical vulnerability in Windows for which no advisory or fix has yet been released. This vulnerability is particularly serious because we know it is being actively exploited.


++

Primitives

+ One kernel structure leak

+ One kernel memory corruption vulnerability - ‘or’ any value with 4

+ Combined for SYSTEM code exec on Windows 7 to 10, 32 + 64 bit

+ Source: https://github.com/mwrlabs/CVE-2016-7255
Data Leak

+ void* HMValidateHandle(HANDLE h, int type);
+ Undocumented/unexported function in user32
+ Copies entire tagWND structure into user memory
+ Helpfully tagWND includes a pointer to itself :D
Corruption Primitive

- Window object
- NtUserSetWindowLongPtr, can modify spmenu with no checks
- xxxNextWindow takes this value and uses it as a pointer to a tagMenu
- Sets a single bit the address + 0x28 using an ‘or’ with 4
- Allows a byte at any address in memory to have it’s 6th least significant bit set
++

**Exploitation – setup**

+ Create 0x100 Window objects
+ HMValidateHandle to leak locations in kernel memory
+ Find two that are < 0x3fd00 apart
+ Destroy spares
++

Exploitation – Initial corruption

+ Extra memory after a tagWND
+ Size == cbwndExtra
++

Exploitation – Initial corruption

+ Use the corruption primitive to ‘or’ highest byte of cbWndExtra with 4

+ 0 -> 0x04000000

+ Extra memory now includes the secondary tagWND structure
++

Exploitation – Read primitive

+ Corrupt tagWND -> any address read
+ spwndParent field – pointer to parent window
+ NtUserGetAncestor reads 32 bit int at spwndParent
+ End of tagWND 1 – start of tagWND 2
  spwndParent

++
Exploitation - Read primitive

+ Call NtUserSetWindowLongPtr(primaryWindow, diff, TARGET_ADDRESS)
+ NtUserGetAncestor to read it
Exploitation – Read primitive

+ Call NtUserSetWindowLongPtr(primaryWindow, diff, TARGET_ADDRESS)

+ NtUserGetAncestor to read it
++

Exploitation – Write primitive

+ Turn corrupting a tagWND into an any address write

+ tagWND has a name field – overwrite it’s buffer pointer with the address we want to write

---

HANDLE h = 0xFFFFFFFF

...  
PVOID pSelf = 0xFFFFFFFFFFFFFFFF

...  
PVOID spwndParent = 0xFFFFFFFFFFFFFFFF

...  
unsigned int cbwndExtra = 0x0

...

200 byte gap

strName.Buffer = 0xFFFFFFFFFFFFFFFF
++

Exploitation – Write primitive

+ Turn corrupting a tagWND into an any address write

+ tagWND has a name field – overwrite it’s buffer pointer with the address we want to write

+ Call SetWindowText to write arbitrary data to it

```c
HANDLE h = 0xFFFFFFFF
....
PVOID spwndParent = 0xFFFFFFFFFFFFFFFF
....
PVOID pSelf = 0xFFFFFFFFFFFFFFFF
.....
unsigned int cbwndExtra = 0x0
...
200 byte gap
strName.Buffer = 0x4141414141414141
```
Exploitation – Privesc

<table>
<thead>
<tr>
<th>tagWND</th>
</tr>
</thead>
<tbody>
<tr>
<td>HANDLE h = 0xFFFFFFFF</td>
</tr>
<tr>
<td>PVOID pti = 0xFFFFFFFF???????????</td>
</tr>
<tr>
<td>PVOID pSelf = 0xFFFFFFFFFFFFFFFF</td>
</tr>
<tr>
<td>....</td>
</tr>
<tr>
<td>PVOID spwndParent = 0xFFFFFFFFFFFFFFFF</td>
</tr>
<tr>
<td>....</td>
</tr>
<tr>
<td>unsigned int cbwndExtra = 0x0</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>
Exploitation – Privesc

**tagWND**
- HANDLE h = 0xFFFFFFFF
- PVOID pti = 0xFFFFFFFF
- PVOID pSelf = 0xFFFFFFFF
- PVOID spwndParent = 0xFFFFFFFF
- unsigned int cbwndExtra = 0x0

**tagTHREAD**
- PVOID pETHREAD = 0xFFFFFFFF
Exploitation – Privesc

tagWND

HANDLE h = 0xFFFFFFFF
PVOID pti = 0xFFFFFFFF
PVOID pSelf = 0xFFFFFFFF
PVOID spwndParent = 0xFFFFFFFF
unsigned int cbwndExtra = 0x0
...
PVOID pti = 0xFFFFF

PVOID pETHREAD = 0xFFFFFFFF

ETHREAD

PVOID pKAPC_STATE = 0xFFFFFFFF
Exploitation – Privesc

ETHREAD

...  
PVOID pKAPC_STATE = 0xFFFFF????????????
...  

KAPC_STATE

...  
PVOID pKPROCESS = 0xFFFFF????????????
...
Exploitation – Privesc

ETHREAD

... 

PVOID pKAPC_STATE = 0xFFFFF????????????
...

KAPC_STATE

...

PVOID pKPROCESS = 0xFFFFF?????????????
...

KPROCESS

...

UINT UniqueProcessId
...

PVOID ActiveProcessLinks
...

PVOID Token
...
MWR Labs

++

Exploitation – Privesc

KPROCESS
...
UINT UniqueProcessId
...
PVOID ActiveProcessLinks
...
PVOID Token
...
Exploitation – Privesc

KPROCESS
...
UINT UniqueProcessId
...
PVOID ActiveProcessLinks
...
PVOID Token
...

4?
++

Exploitation – Privesc

KPROCESS
...
UINT UniqueProcessId
...
PVOID ActiveProcessLinks
...
PVOID Token
...

CTRL + C

4?

CTRL + V
Exploitation – Privesc
Exploitation – Demo

+ The demo’s are all recorded
+ Because: https://youtu.be/DLND8bKv27w?t=35m21s
Exploitation - Anniversary update

WIN32K_CRITICAL_FAILURE (164)
Win32k has encountered a critical failure.
Arguments:
Arg1: 0000000000000006, DESKTOP_HEAP_POINTER_OUT_OF_RANGE
Unspecified
Arg2: ffffffff811a3763d0, Unspecified
Arg3: ffffffff80000000, Unspecified
Arg4: 0000000001400000, Unspecified
Hardening Windows 10 with zero-day exploit mitigations


January 30, 2017

Hardening Windows 10 With Zero Day Exploit Mitigations Under The Microscope

https://improsec.com/blog/hardening-windows-10-with-zero-day-exploit-mitigations-under-the-microscope
Anniversary Update – Read
## Anniversary Update – Read

<table>
<thead>
<tr>
<th>Line</th>
<th>Address</th>
<th>Name</th>
<th>Address 2</th>
<th>Name 2</th>
<th>Ratio</th>
<th>BBlocks 1</th>
<th>BBlocks 2</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00495</td>
<td>1c00c8b6c</td>
<td>_GetParent</td>
<td>1c00df16c</td>
<td>_GetParent</td>
<td>1.000</td>
<td>5</td>
<td>5</td>
<td>Function hash</td>
</tr>
<tr>
<td>00131</td>
<td>1c00915ec</td>
<td>GetMessageWindow</td>
<td>1c0039880</td>
<td>GetMessageWindow</td>
<td>1.000</td>
<td>3</td>
<td>3</td>
<td>Function hash</td>
</tr>
<tr>
<td>01449</td>
<td>1c00504c0</td>
<td>UserSetLastError</td>
<td>1c003269c</td>
<td>UserSetLastError</td>
<td>1.000</td>
<td>10</td>
<td>10</td>
<td>Perfect match, same name</td>
</tr>
<tr>
<td>00455</td>
<td>1c005df50</td>
<td>_GetDesktopWindow</td>
<td>1c008774c</td>
<td>_GetDesktopWindow</td>
<td>1.000</td>
<td>4</td>
<td>4</td>
<td>Function hash</td>
</tr>
</tbody>
</table>
Anniversary Update – Read

https://www.youtube.com/watch?v=nPR22lS0-oY
Anniversary Update – Write

<table>
<thead>
<tr>
<th>Line</th>
<th>Address</th>
<th>Name</th>
<th>Address 2</th>
<th>Name 2</th>
<th>Ratio</th>
<th>BBlocks 1</th>
<th>BBlocks 2</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00610</td>
<td>1c0074938</td>
<td>DefSetText</td>
<td>1c0038530</td>
<td>DefSetText</td>
<td>0.970</td>
<td>27</td>
<td>29</td>
<td>Perfect match, same name</td>
</tr>
<tr>
<td>01497</td>
<td>1c00eb3b0</td>
<td>NtUserDefSetText</td>
<td>1c010b700</td>
<td>NtUserDefSetText</td>
<td>0.960</td>
<td>21</td>
<td>20</td>
<td>Perfect match, same name</td>
</tr>
</tbody>
</table>

```
119loc_1c00386a8:
120    lea    rdx, [rbx+0D8h]
121    mov    rcx, rsi
122    call   DesktopVerifyHeapLargeUnicodeString
123    mov    ecx, [rsp+58h+arg_10]
124    jretn  1c00205000
```
Anniversary Update – Write

- Verifies the Window Name is within the Desktop Heap
- By comparing to details in tagDESKTOP
- The pointer from which comes from tagWND
- And it isn’t verified...
- ...and we can modify tagWND
Anniversary Update – Write

1. Read the tagDESKTOP pointer from tagWND
2. Overwrite tagDESKTOP pointer to point to user mode memory
Anniversary Update – Write

1. Read the tagDESKTOP pointer from tagWND

2. Overwrite tagDESKTOP pointer to point to user mode memory
++

Anniversary Update – Write

1. Read the tagDESKTOP pointer from tagWND
2. Overwrite tagDESKTOP pointer to point to user mode memory
3. Fake required values to pass checks
4. Overwrite strName
Anniversary Update – Write

1. Read the tagDESKTOP pointer from tagWND

2. Overwrite tagDESKTOP pointer to point to user mode memory

3. Fake required values to pass checks

4. Overwrite strName

5. Call NtUserDefSetText

```c
HANDLE h = 0xFFFFFFFF
PVOID pSelf = 0xFFFFFFFFFFFFFFFF
...
...
PVOID spwndParent = ...
unsigned int cbwndExtra = 0x00000000
...
...
strName.Buffer = 0x4141414141414141
PVOID rpDesktop = 0x000000002
```
Anniversary Update – Write

1. Read the tagDESKTOP pointer from tagWND
2. Overwrite tagDESKTOP pointer to point to user mode memory
3. Fake required values to pass checks
4. Overwrite strName
5. Call NtUserDefSetText
6. Replace tagDESKTOP
Anniversary Update – Write

1. Read the tagDESKTOP pointer from tagWND
2. Overwrite tagDESKTOP pointer to point to user mode memory
3. Fake required values to pass checks
4. Overwrite strName
5. Call NtUserDefSetText
6. Replace tagDESKTOP
7. Replace strName

HANDLE h = 0xFFFFFFFF
PVOID pSelf = 0xFFFFFFFFFFFFFFFF
...
...
PVOID spwndParent = ...
unsigned int cbwndExtra = 0x00000000
...
...
strName.Buffer = 0xFFFFFFFFFFFFFFFF
PVOID rpDesktop = 0xFFFFFFFFFFFFFFFF
Anniversary Update – Write

1. Read the tagDESKTOP pointer from tagWND
2. Overwrite tagDESKTOP pointer to point to user mode memory
3. Fake required values to pass checks
4. Overwrite strName
5. Call NtUserDefSetText
6. Replace tagDESKTOP
7. Replace strName

Details @ https://improsec.com/blog/hardening-windows-10-with-zero-day-exploit-mitigations-under-the-microscope
MWR Labs

Anniversary Update – w00t

https://www.youtube.com/watch?v=a_R11Fzpk_c
Conclusions

- Windows kernel has a massive complex attack surface
- Exploit development rapidly becoming harder
- Not going away anytime soon
- OEM’s will destroy your security
Questions?
References

1. https://medium.com/@mxatone/how-bad-design-decisions-created-the-least-secure-driver-on-windows-33e662a502fe#.a527m4bvt
2. https://googleprojectzero.blogspot.co.uk/2016/06/a-year-of-windows-kernel-font-fuzzing-1_27.html
7. https://whitehatters.academy/diffing-with-kam1n0/