A Window into Ring0

Sam Brown

Securi-Tay 2017
Please stop using Windows 7, what year is this? Why are you doing that?
++

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
Applications

System DLL’s

Ring 3

Ntdll.dll
User32.dll
Gdi32.dll

Win32u.dll

System Calls

Ntoskrnl.exe
~449 system calls

Win32k.sys
~1138 system calls

Drivers

Ring 0

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

- Font’s 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 <3’s kernel fuzzing
+ [https://github.com/mwrlabs/KernelFuzzer](https://github.com/mwrlabs/KernelFuzzer)

---

**Windows Kernel Fuzzing**

Nils presented Windows Kernel Fuzzing at T2.fi 2015

---

**Platform Agnostic Kernel Fuzzing**

James Loureiro and Georgi Geshev presented ‘Platform Agnostic Kernel Fuzzing’ at Def Con 24.

---

**Fuzzing the Windows kernel**

Yong Chuan Koh presented ‘Windows kernel fuzzing’ HITB GSEC, Singapore
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:
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 GWLP_ID on a window handle with GWL_STYLE set to WS_CHILD. Chrome’s sandbox blocks win32k.sys system calls using
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
  - NCC Group’s: 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
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]**

<table>
<thead>
<tr>
<th>Test</th>
<th>ebx, ebx</th>
</tr>
</thead>
<tbody>
<tr>
<td>jz</td>
<td>short loc_BF937F73</td>
</tr>
<tr>
<td>cmp</td>
<td>ebx, 0FFFFFFFH</td>
</tr>
<tr>
<td>jz</td>
<td>short loc_BF937F73</td>
</tr>
<tr>
<td>push</td>
<td>dword ptr [ebp-4]</td>
</tr>
<tr>
<td>push</td>
<td>dword ptr [ebp+10h]</td>
</tr>
<tr>
<td>push</td>
<td>1EFh</td>
</tr>
<tr>
<td>push</td>
<td>cbx</td>
</tr>
</tbody>
</table>

```plaintext
3214130781
3214130784
3214130789
3214130791
3214130793
3214130795
3214130796
3214130799
3214130804
3214130805
```

- push cbx
- call IsMFMwFPWindowA, IsMFMwFPWindow(x)
- test eax, eax
- jz short loc_BF93DE7F
- push [ebp+Address]; Address
- push dword ptr [ebp+UnicodeDString]; UnicodeDString
- push 1EFh; MBString
- push cbx; P
- call _xxxSendMessage@16; xxxSendMessage(x,x,x,x)
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.*

Ben Hawkes, USENIX Enigma 2016 - What Makes Software Exploitation Hard?
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...
Just have a friendly driver disable it...

Double KO! Capcom's Street Fighter V installs hidden rootkit on PCs

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

```assembly
lea  rax, disable_sme
lea  rcx, [rsp+48h+var_28]
call rax; disable_sme
mov  rcx, [rsp+48h+var_18]
call [rsp+48h+var_20]; execute shellcode \o/
lea  rax, enable_sme
lea  rcx, [rsp+48h+var_28]
call rax; enable_sme
```
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 Address</th>
<th>Handle Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>3072</td>
<td>0x84B50AE8</td>
<td>0xB8</td>
</tr>
<tr>
<td>3072</td>
<td>0x845DFF00</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>0x8EE9F838</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>0x98265898</td>
<td>0x18</td>
</tr>
<tr>
<td>264</td>
<td>0x82AA7030</td>
<td>0x1C</td>
</tr>
</tbody>
</table>
KASLR – Address Leaks

SystemModuleInformation

<table>
<thead>
<tr>
<th>Module Name</th>
<th>Base Address</th>
<th>Module Name</th>
<th>Base Address</th>
<th>Module Name</th>
<th>Base Address</th>
<th>Module Name</th>
<th>Base Address</th>
<th>Module Name</th>
<th>Base Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>\SystemRoot\System32\Drivers\svr.sys</td>
<td>0x96F85000</td>
<td>Module name \SystemRoot\System32\Drivers\spsys.sys</td>
<td>0xA2E02000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>\SystemRoot\System32\Drivers\BTHUSB.sys</td>
<td>0xA2E6C000</td>
<td>Module name \SystemRoot\System32\Drivers\bthport.sys</td>
<td>0xA2E7E000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>\SystemRoot\system32\Drivers\rfcomm.sys</td>
<td>0xA2E2000</td>
<td>Module name \SystemRoot\system32\DRIVERS\BthEnum.sys</td>
<td>0xA2F06000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>\SystemRoot\System32\Drivers\bthpan.sys</td>
<td>0xA2F13000</td>
<td>Module name \Windows\System32\ntdll.dll</td>
<td>0x77810000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>\Windows\System32\smss.exe</td>
<td>0x47AF0000</td>
<td>Module name \Windows\System32\apisetschema.dll</td>
<td>0x77A50000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>\Windows\System32\autochk.exe</td>
<td>0x330000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
++

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
NtQuerySystemInformation failed with error code 0xC0000022
<table>
<thead>
<tr>
<th>Technique</th>
<th>Windows 7</th>
<th>Windows 8</th>
<th>Windows 8.1 Low Integrity</th>
<th>Windows 8.1 Medium Integrity</th>
<th>Windows 10 Low Integrity</th>
<th>Windows 10 Medium Integrity</th>
</tr>
</thead>
<tbody>
<tr>
<td>NtQuerySystemInformation</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>
</tr>
<tr>
<td>NtQuerySystemInformation</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>
</tr>
<tr>
<td>NtQuerySystemInformation</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>
</tr>
<tr>
<td>NtQuerySystemInformation</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>
</tr>
<tr>
<td>NtQuerySystemInformation</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>
</tr>
<tr>
<td>System Call Return Values</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>
</tr>
<tr>
<td>Descriptor Tables</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>
</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
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 its 6th 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

+ Call SetWindowText to write arbitrary data to 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
+ Call SetWindowText to write arbitrary data to it

<table>
<thead>
<tr>
<th>tagWND</th>
</tr>
</thead>
<tbody>
<tr>
<td>HANDLE h = 0xFFFFFFFF</td>
</tr>
<tr>
<td>PVOID pSelf = 0xFFFFFFFFFFFFFFFF</td>
</tr>
<tr>
<td>PVOID spwndParent = 0xFFFFFFFFFFFFFFFF</td>
</tr>
<tr>
<td>unsigned int cbwndExtra = 0x04000000</td>
</tr>
<tr>
<td>200 byte gap</td>
</tr>
<tr>
<td>strName.Buffer = 0x4141414141414141</td>
</tr>
<tr>
<td>unsigned int cbwndExtra = 0x0</td>
</tr>
</tbody>
</table>

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

```c
tagWND
HANDLE h = 0xFFFFFFFF
PVOID pti = 0xFFFFFFFF????????
PVOID pSelf = 0xFFFFFFFFFFFFFFFF
....
PVOID spwndParent = 0xFFFFFFFFFFFFFFFF
....
unsigned int cbwndExtra = 0x0
...
```
Exploitation – Privesc

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

**tagTHREAD**
- PVOID pETHREAD = 0xFFFFF????????????
- ...

Exploitation – Privesc

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

**tagTHREAD**
- PVOID pETHREAD = 0xFFFFFFFF???????????

**ETHREAD**
- PVOID pKAPC_STATE = 0xFFFFFFFF???????????
Exploitation – Privesc

```c
ETHREAD

...

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

KAPC_STATE

...

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

ETHREAD

... PVOID pKAPC_STATE = 0xFFFF???????????
...

KAPC_STATE

... PVOID pKPROCESS = 0xFFFF???????????
...

KPROCESS

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

<table>
<thead>
<tr>
<th>KPROCESS</th>
<th>KPROCESS</th>
<th>KPROCESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>UINT UniqueProcessId</td>
<td>UINT UniqueProcessId</td>
<td>UINT UniqueProcessId</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>PVOID ActiveProcessLinks</td>
<td>PVOID ActiveProcessLinks</td>
<td>PVOID ActiveProcessLinks</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>PVOID Token</td>
<td>PVOID Token</td>
<td>PVOID Token</td>
</tr>
</tbody>
</table>
Exploitation – Privesc

KPROCESS

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

4?
Exploitation – Privesc
Exploitation – Privesc

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

4?

CTRL + C

CTRL + V
Exploitation – Privesc
Exploitation – Privesc

A screenshot of a Task Manager window showing various processes and their details. The process 'calc.exe' is running with a PID of 2976 and a CPU usage of 0%. Other processes listed include 'cmd.exe', 'conhost.exe', 'csrss.exe', and 'CVE-2016-7255.exe'.
Caveats

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
Conclusions

- Windows kernel has a massive complex attack surface
- Exploit development rapidly becoming harder
- Not going away anytime soon
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/