
MUNICH CYBER TACTICS
TECHNIQUES AND PROCEDURES
23
MCTTP
14. & 15. September 2023
mcttp.de | #mcttp23
VOGEL IT AKADEMIE

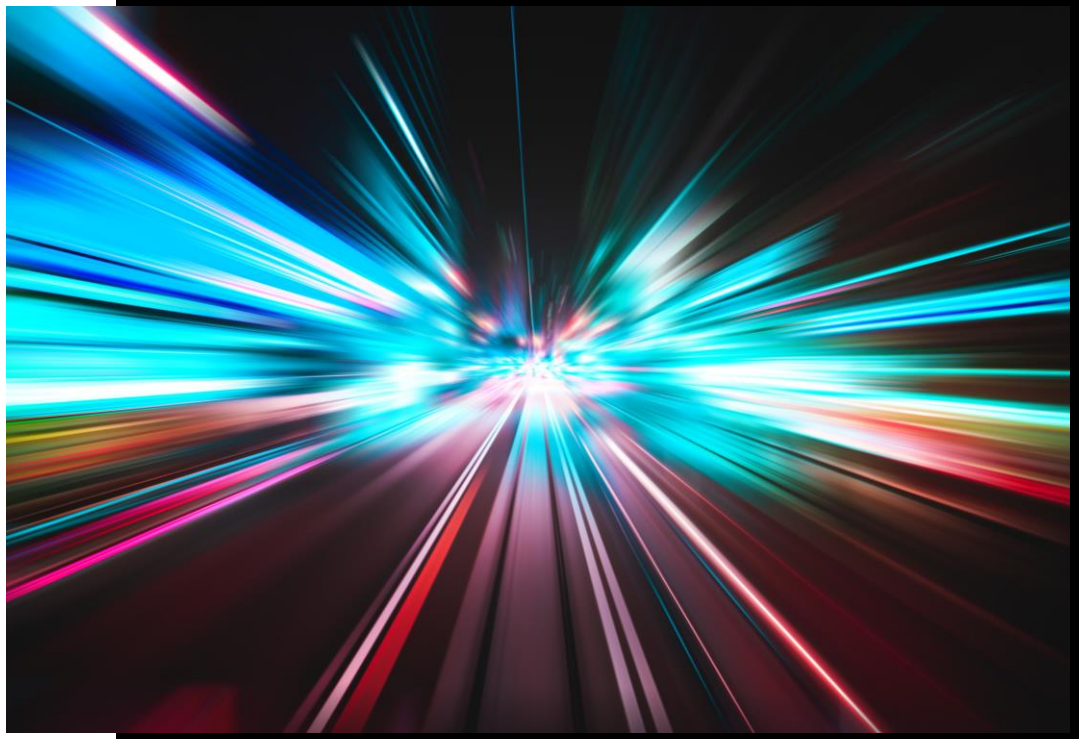
MCTTP 2023

PLAYING CHESS AS RED TEAMS

Fabian Mosch

00

AGENDA

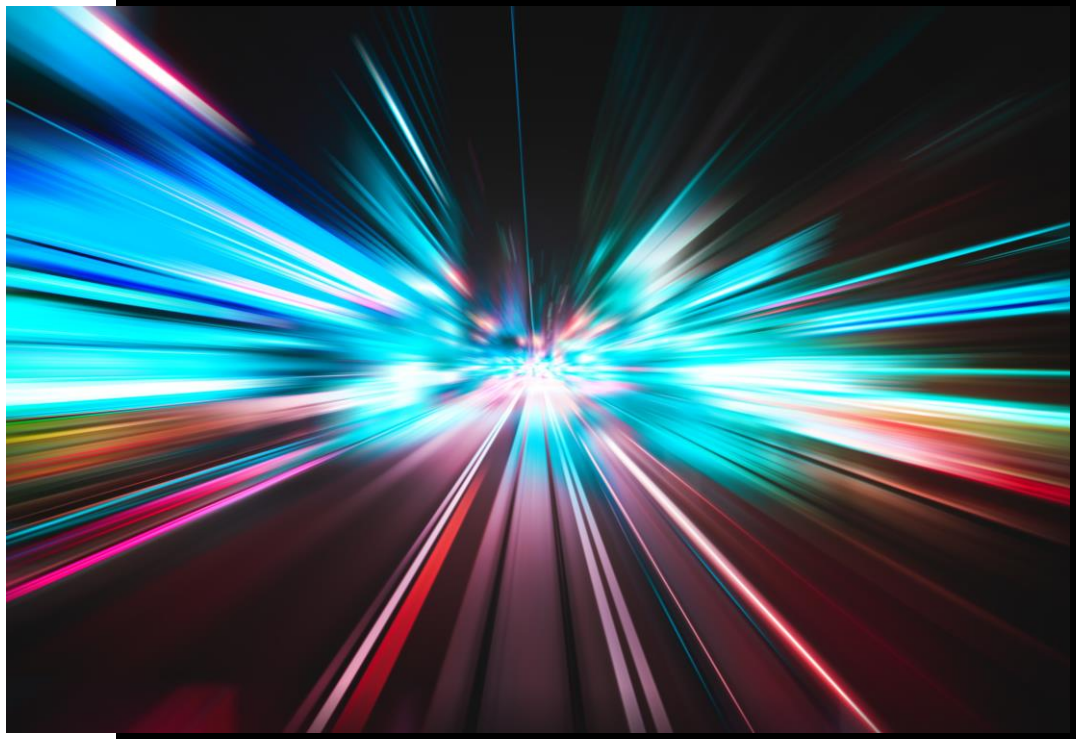


AGENDA

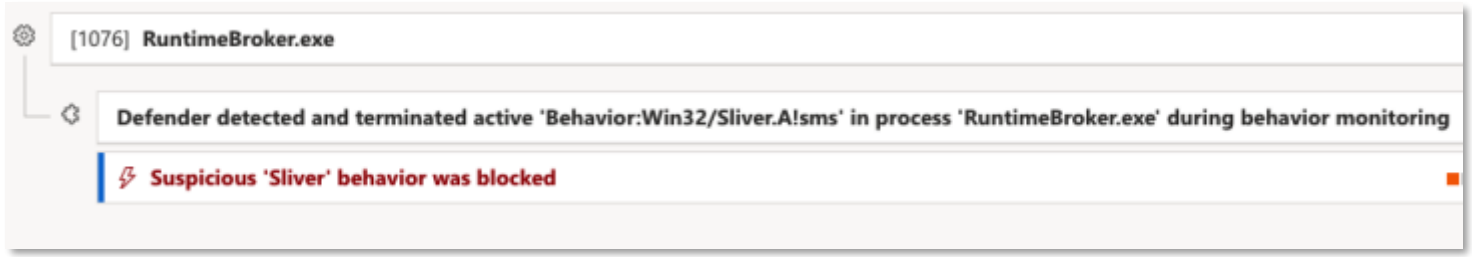
- The need for strategic decisions – avoid EDR detections
- Custom tooling and obfuscation
- Diving into evasion techniques
- Avoiding detections coming from kernelland
- Proof of Concept

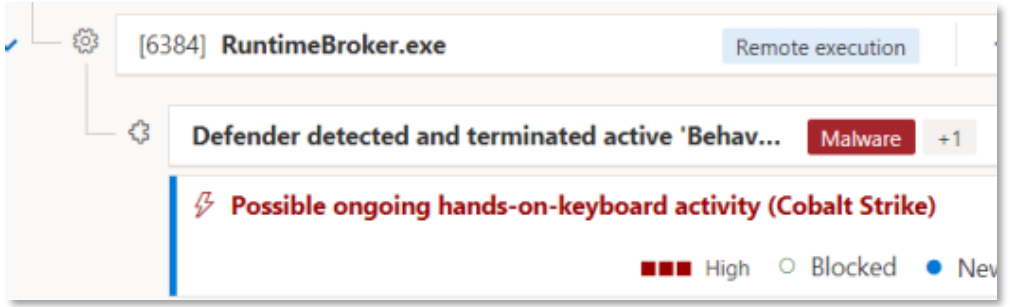
01

THE NEED FOR STRATEGIC DECISIONS – AVOID EDR DETECTIONS



AVOID EDR DETECTIONS

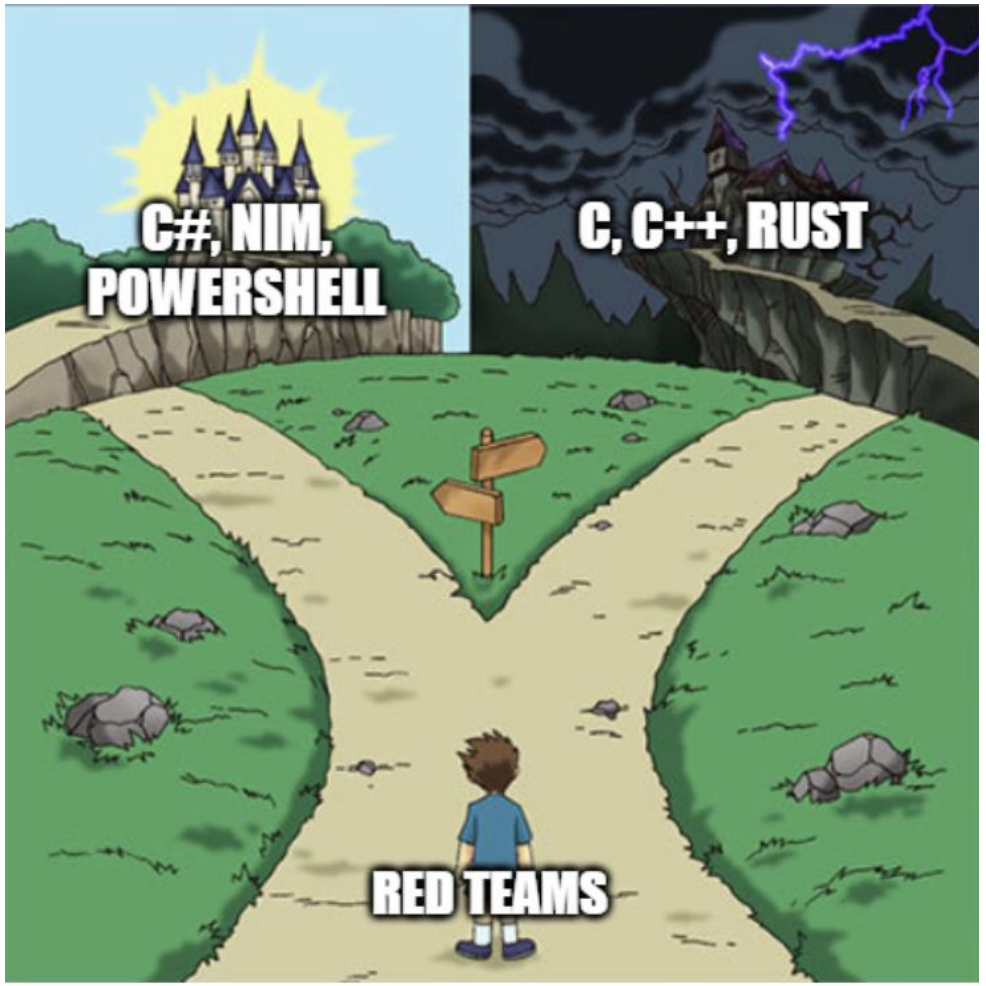




THE NEED FOR STRATEGIC DECISIONS

Programming language:

- In any case *something* has to be privately developed (C2, Packer, …)
  - Public tooling is not feasible anymore
- What is your main goal:
  - Most control
  - Simplicity & less effort
- What is the team capable of and how much budget is there for innovation/tooling



THE NEED FOR STRATEGIC DECISIONS

Execution technique:

- Pack Payloads to execute them from memory
  - Downside of being detectable by memory scans
- Obfuscate Payloads
  - May break functionality
  - May not evade signatures
- Test Payloads (!)
  - If you know about the target EDR
  - Otherwise against any

THE NEED FOR STRATEGIC DECISIONS

Where to go

- Stay in the local process
  - Initial EDR detection rate is lower
  - Easier to detect when doing manual analysis
- Inject into another process
  - Got much harder without detections
  - New evasion approaches needed

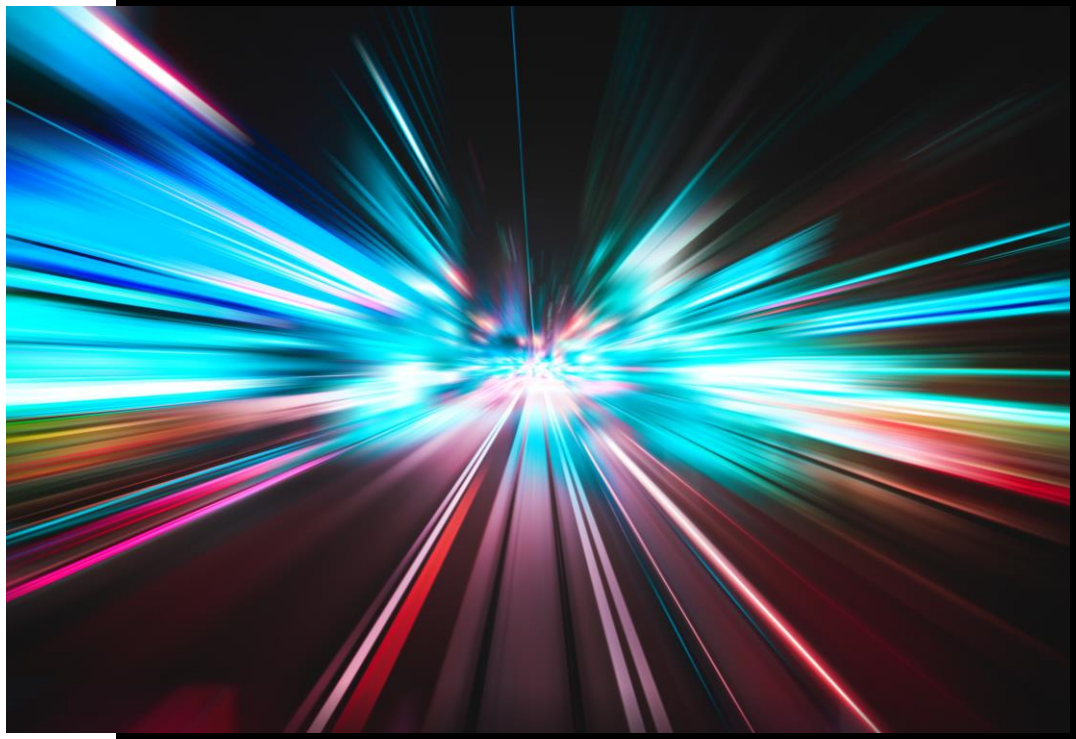
THE NEED FOR STRATEGIC DECISIONS

<u>In the very end it´s still about signatures:</u>

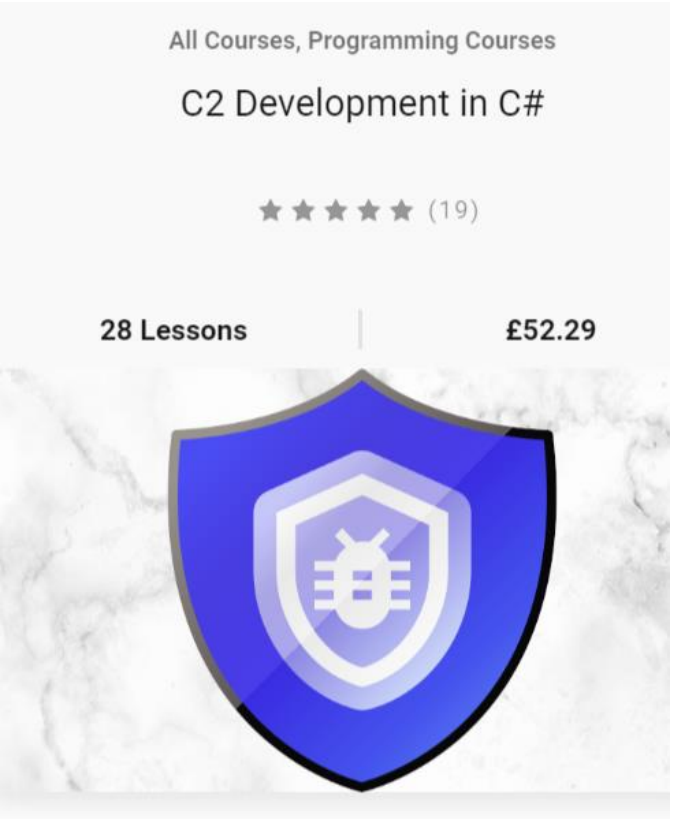
- Although often memory based nowadays
- Custom tooling would evade most signatures
  - Much effort
  - Many resources needed
- Evasion techniques as alternative
  - Wide selection
  - Techniques itself could get detected

02

CUSTOM TOOLING AND OBFUSCATION



CUSTOM TOOLING AND OBFUSCATION



<u>Exemplary C2 design decision:</u>

- C# C2-Framework - less effort and simplicity
  - ZeroPoint Security C2 development course for beginners

<u>Avoid signature based detections:</u>

- Minimalistic agents
  - All Namespaces, Class-Names, variables, […] random for each agent
  - All modules loaded on runtime - also randomized per agent

CUSTOM TOOLING AND OBFUSCATION

Tooling execution approach:

- Public C# tooling for
  - Lateral Movement
  - Credential Harvesting
  - Persistence
  - …

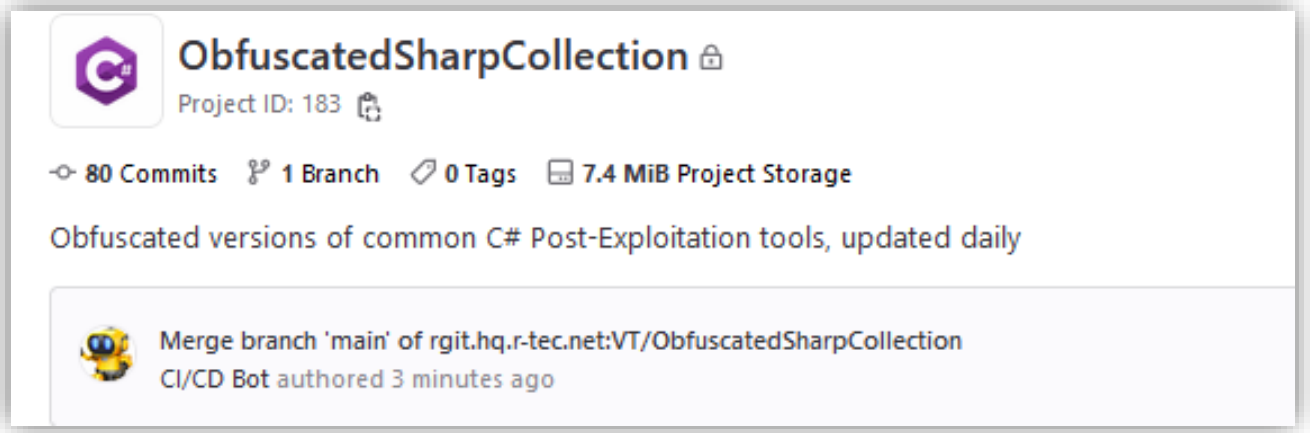
Detections based on:

- AMSI
- ETW

CUSTOM TOOLING AND OBFUSCATION

Exemplary solution:

- Azure Obfuscation Pipeline
  - Renaming for each tool
  - GUID change
  - String obfuscation
  - …

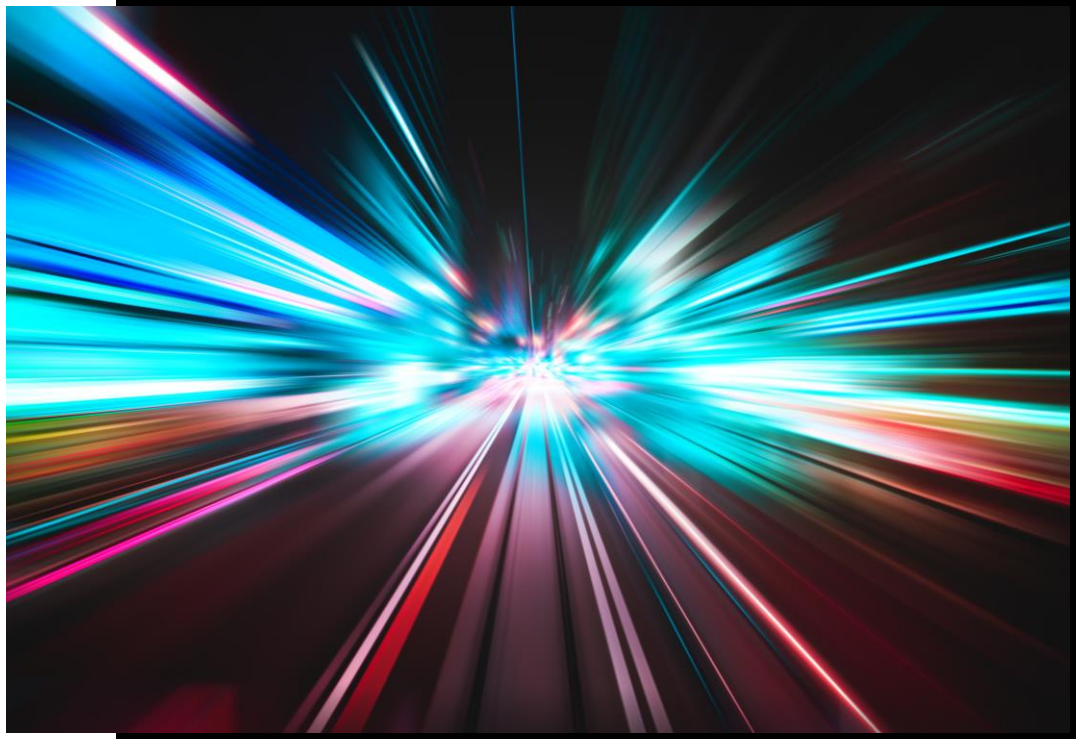


No necessary need for:

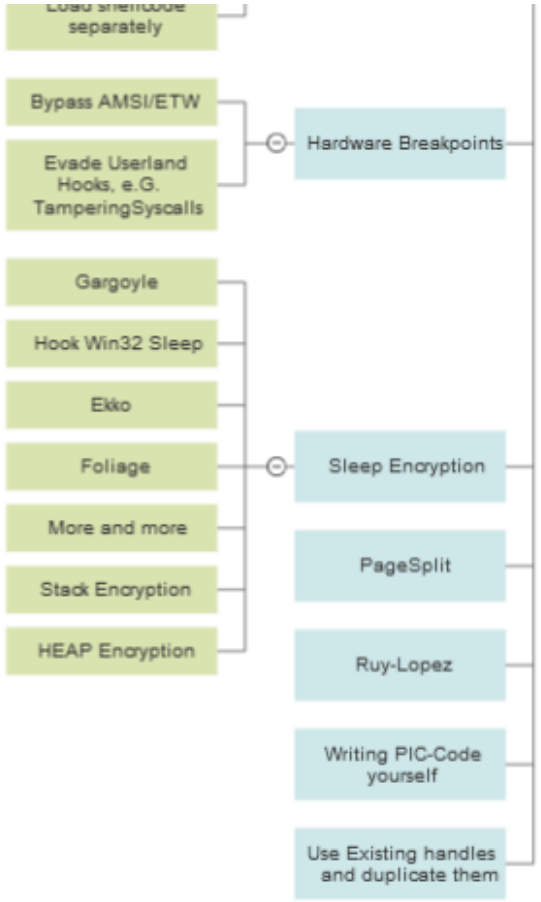
- AMSI bypasses
- ETW tampering

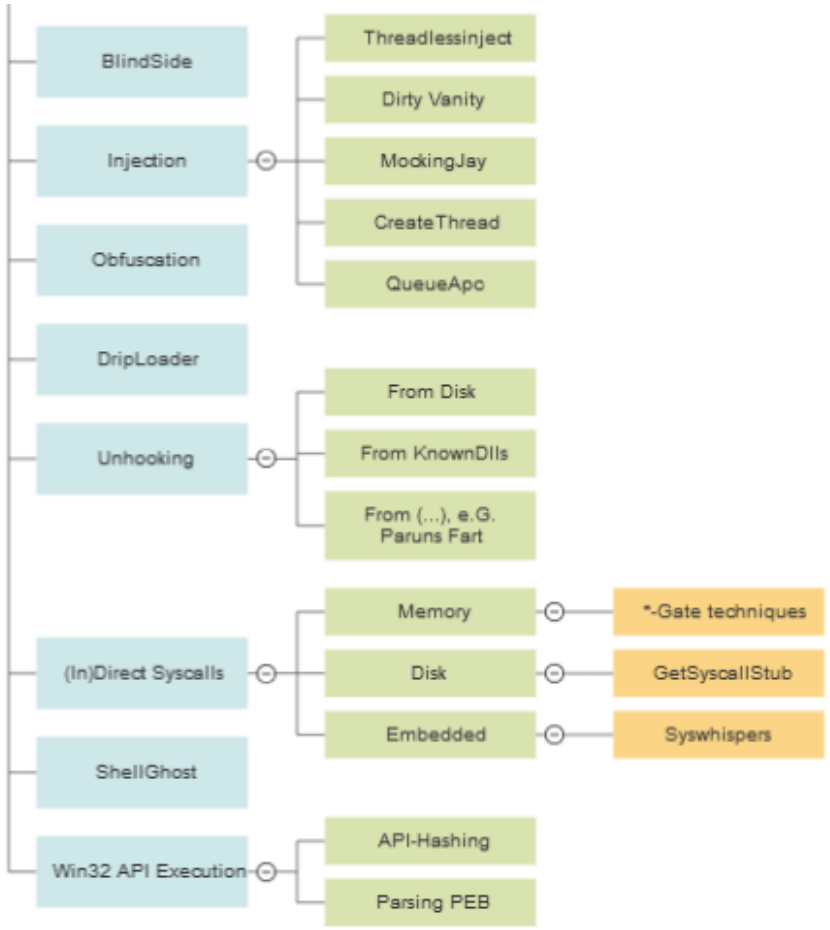
03

DIVING INTO EVASION TECHNIQUES



DIVING INTO EVASION TECHNIQUES

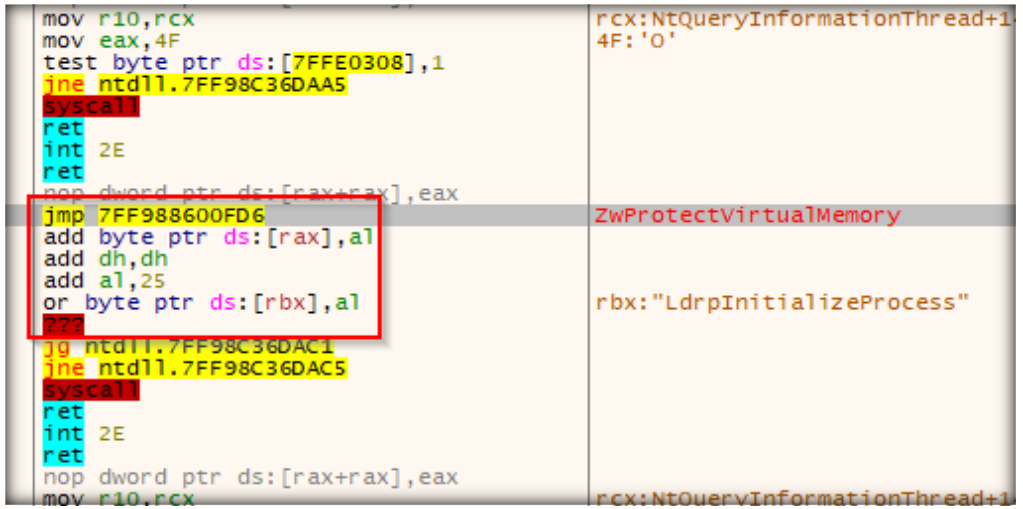




DIVING INTO EVASION TECHNIQUES

Userland Hooks

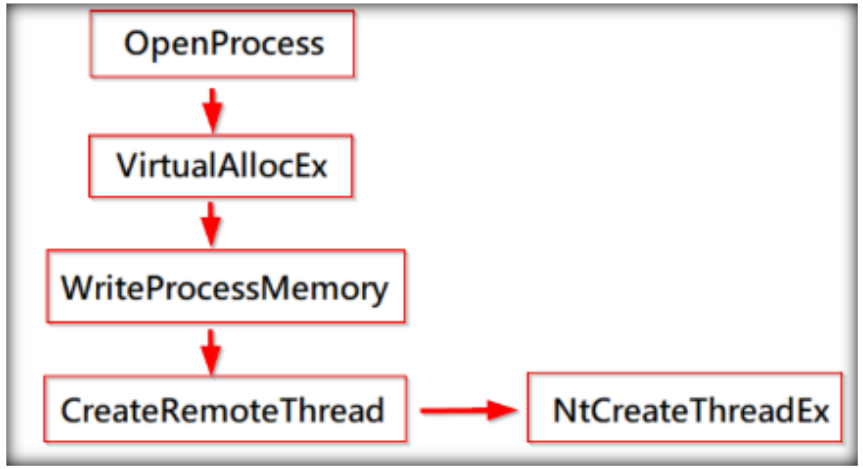
- Memory Windows API patching
  - The `jmp` goes to the EDR DLL
- Input arguments analysis
- Malicious Payloads can be detected on runtime



DIVING INTO EVASION TECHNIQUES

Userland Hooks – simple Example

- EDR checks the `startAddress` on runtime
  - A memory Scan for its memory location is done
  - Yara rule finds Cobaltstrike/Sliver/Covenant Shellcode and verifies that as known malicious
  - The Process is killed

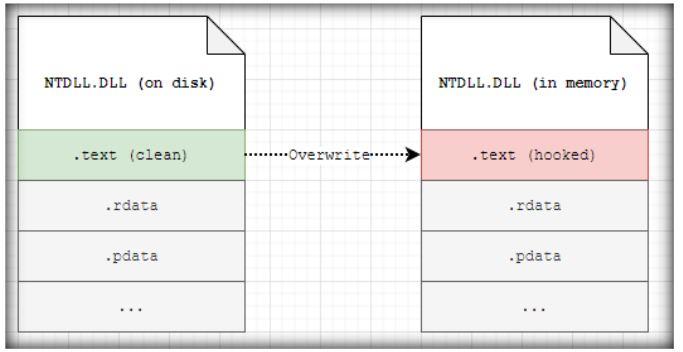



```
def NtCreateThreadEx(
    ref threadHandle as IntPtr
    desiredAccess as UInt32,
    objectAttributes as IntPtr
    processHandle as IntPtr,
    startAddress as IntPtr,
    parameter as IntPtr,
    inCreateSuspended as bool,
    stackZeroBits as Int32,
    sizeOfStack as Int32,
    maximumStackSize as Int32,
    attributeList as IntPtr) as UInt32:
    pass
```

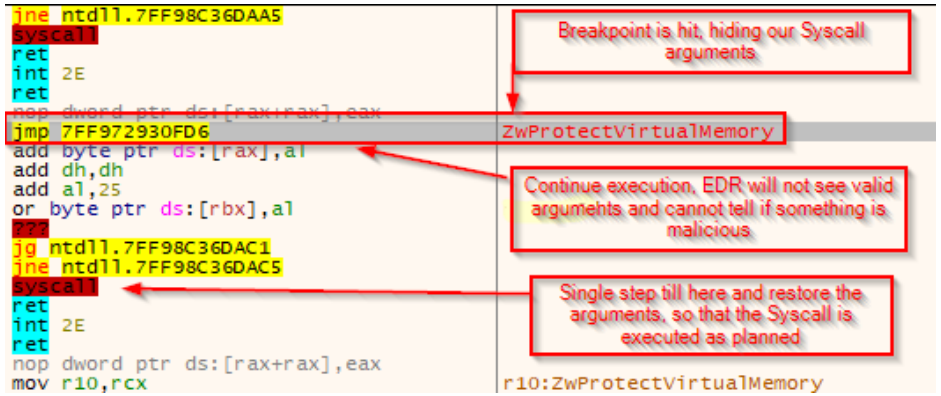
BYPASSING USERLAND HOOKS

Techniques with PoCs published in the last years:

- Unhooking
- Using (In)Direct Syscalls
- Using Hardware Breakpoints
- DLL Entrypoint Patching
- Ruy-Lopez



https://www.ired.team/offensive-security/defense-evasion/how-to-unhook-a-dll-using-c++

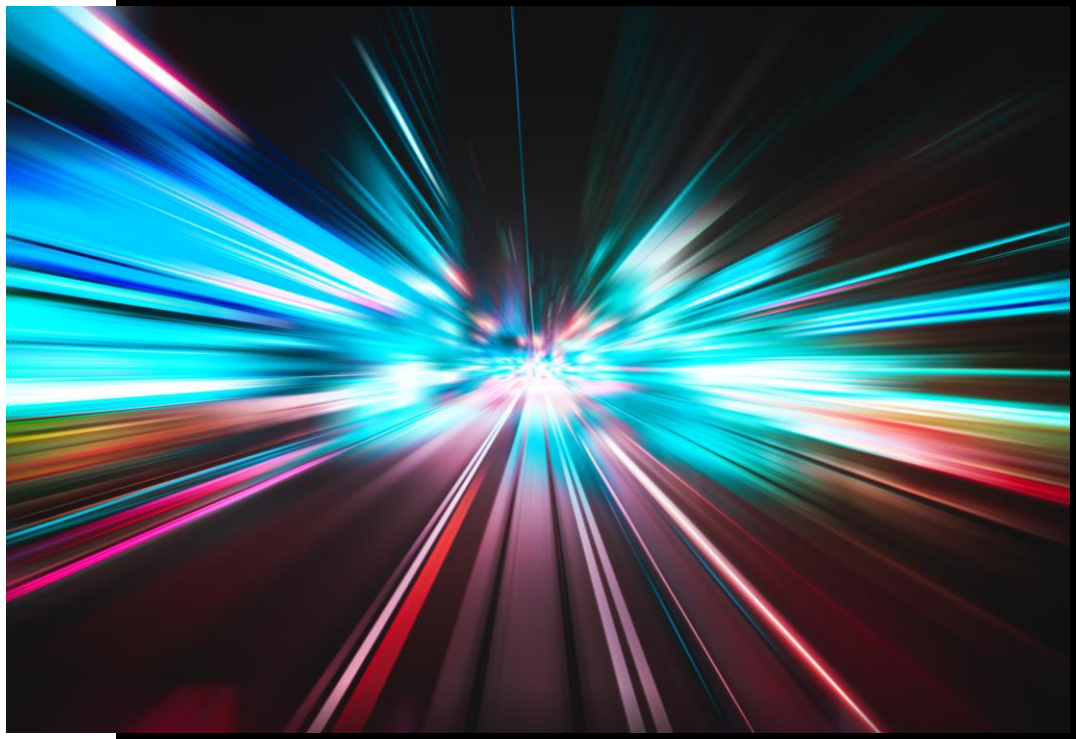


TamperingSyscalls approach

04

AVOIDING DETECTIONS

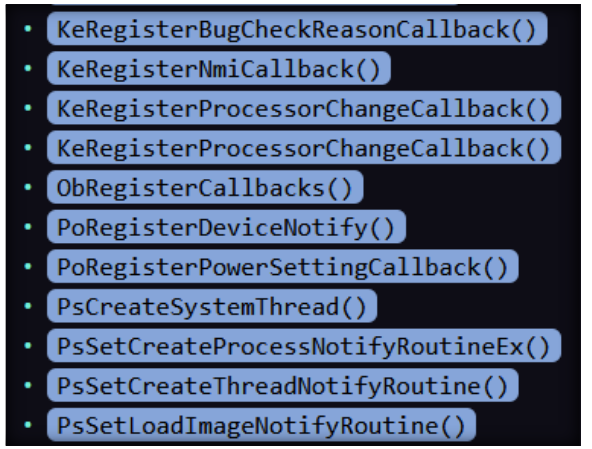
COMING FROM KERNELLAND



AVOIDING DETECTIONS COMING FROM KERNELLAND

Kernel Callbacks

- Live interception / interaction
- Imaginable like Hooks but from Kernel land



Excerpt Kernel Callbacks[1]

ETW threat intelligence

- Event based subscriptions
- Interaction after event capture
  - Stack Trace analysis
  - Memory Scans

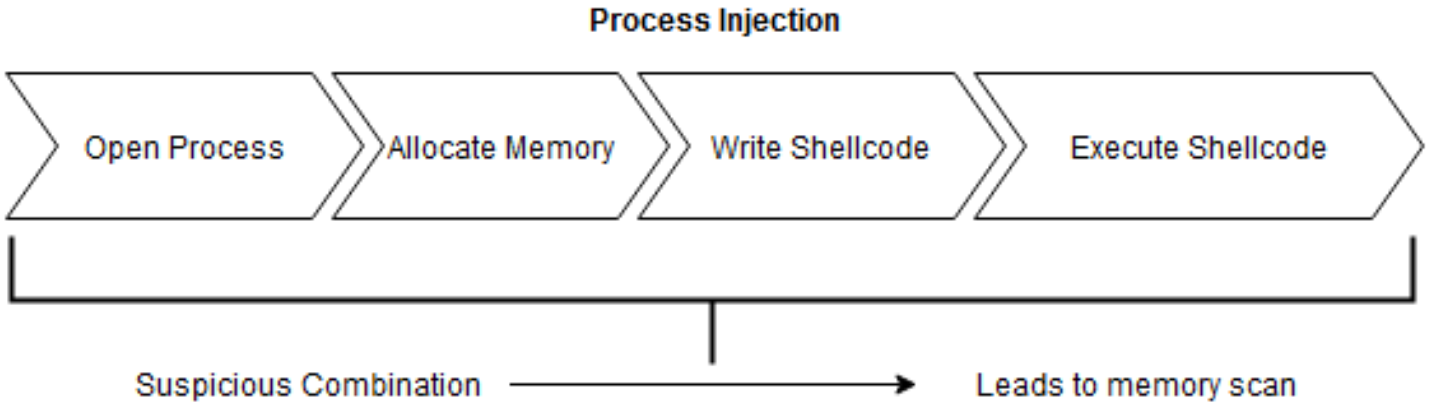
EventId	Event Description
1	THREATINT_ALLOCVM_REMOTE
2	THREATINT_PROTECTVM_REMOTE
3	THREATINT_MAPVIEW_REMOTE
4	THREATINT_QUEUEUSERAPC_REMOTE
5	THREATINT_SETTHREADCONTEXT_REMOTE
6	THREATINT_ALLOCVM_LOCAL
7	THREATINT_PROTECTVM_LOCAL
8	THREATINT_MAPVIEW_LOCAL
11	THREATINT_READVM_LOCAL
12	THREATINT_WRITEVM_LOCAL
13	THREATINT_READVM_REMOTE
14	THREATINT_WRITEVM_REMOTE
15	THREATINT_SUSPEND_THREAD
16	THREATINT_RESUME_THREAD
17	THREATINT_SUSPEND_PROCESS

Excerpt TI Provider events[2]

[1] https://pre.empt.dev/posts/maelstrom-edr-kernel-callbacks-hooks-and-callstacks/#Kernel_Callbacks
[2] https://posts.specterops.io/uncovering-windows-events-b4b9db7eac54

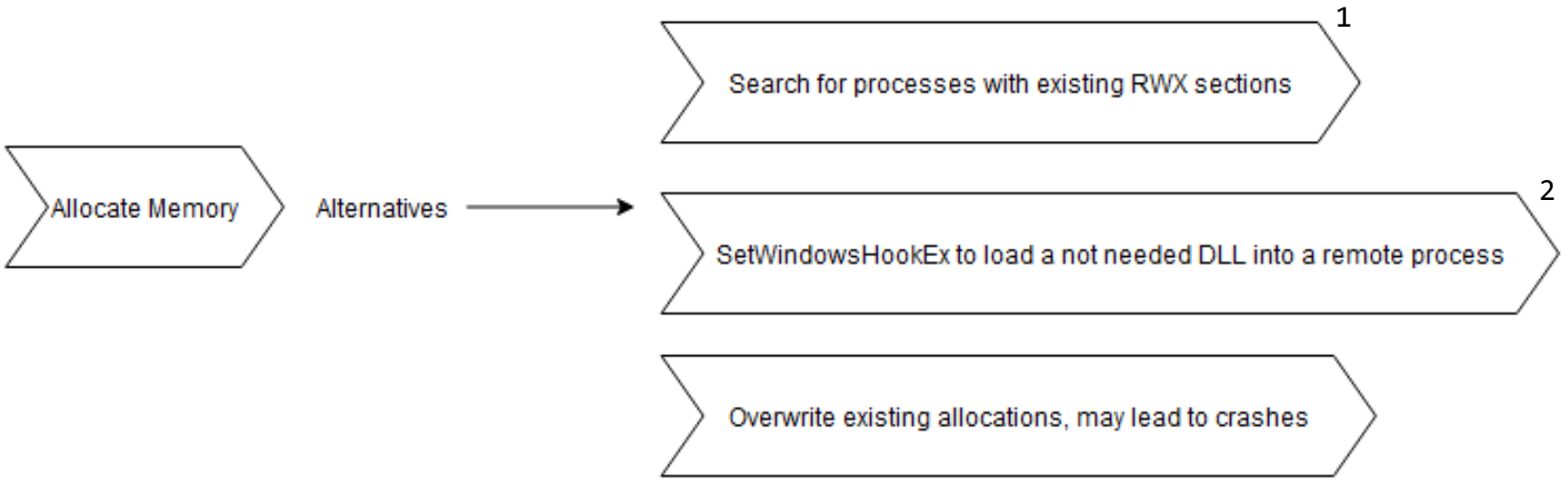
AVOIDING DETECTIONS COMING FROM KERNELLAND

Triggers for specific Windows API combinations



- | Cobalt Strike C2 | ■■■ High | Cobalt Strike C2 on one endpoint | New |
- unencrypted in the remote process memory

AVOIDING DETECTIONS COMING FROM KERNELLAND

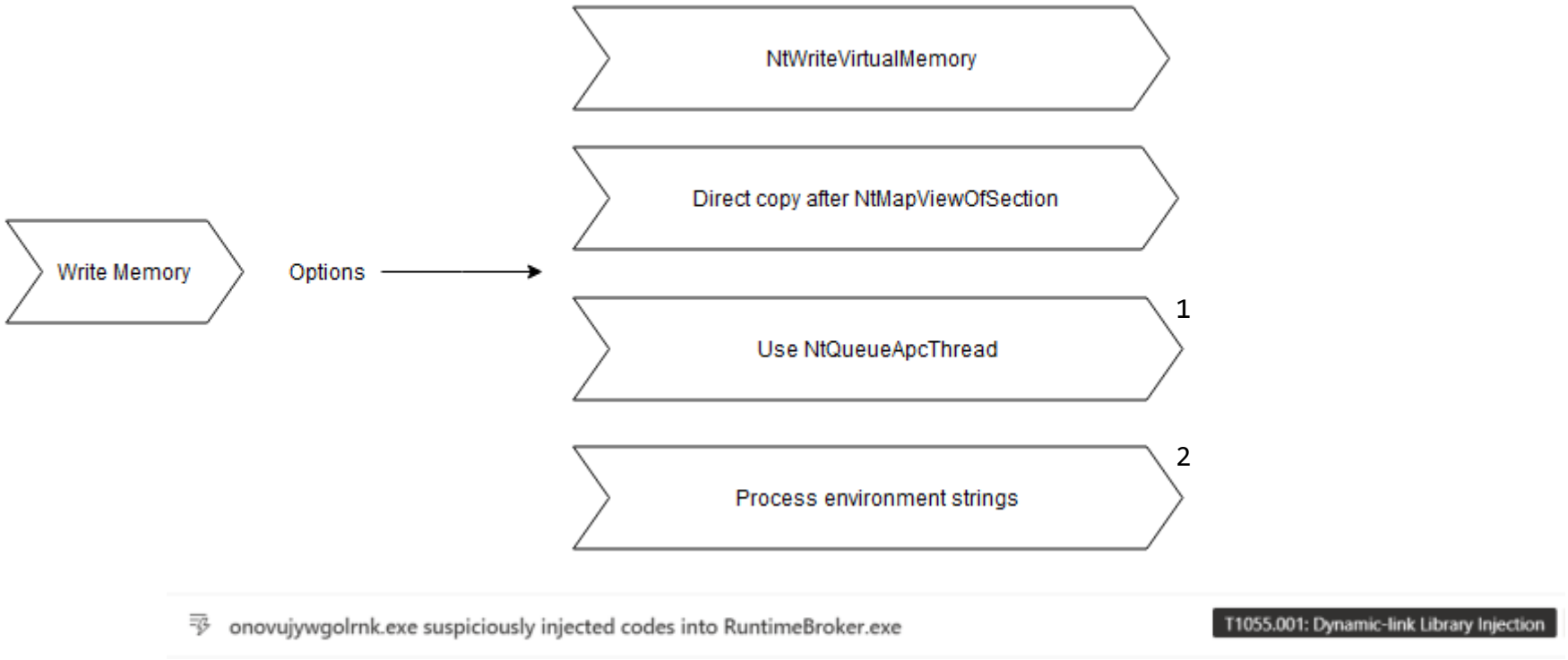


Aug 2, 2023 5:17:48.528 PM lswzjcxhbab.exe allocated memory in the address space of notepad.exe T1055.002: Portable Executable Injection

[1] https://www.ired.team/offensive-security/defense-evasion/finding-all-rwx-protected-memory-regions

[2] https://twitter.com/MrUn1k0d3r/status/1627678626584883200

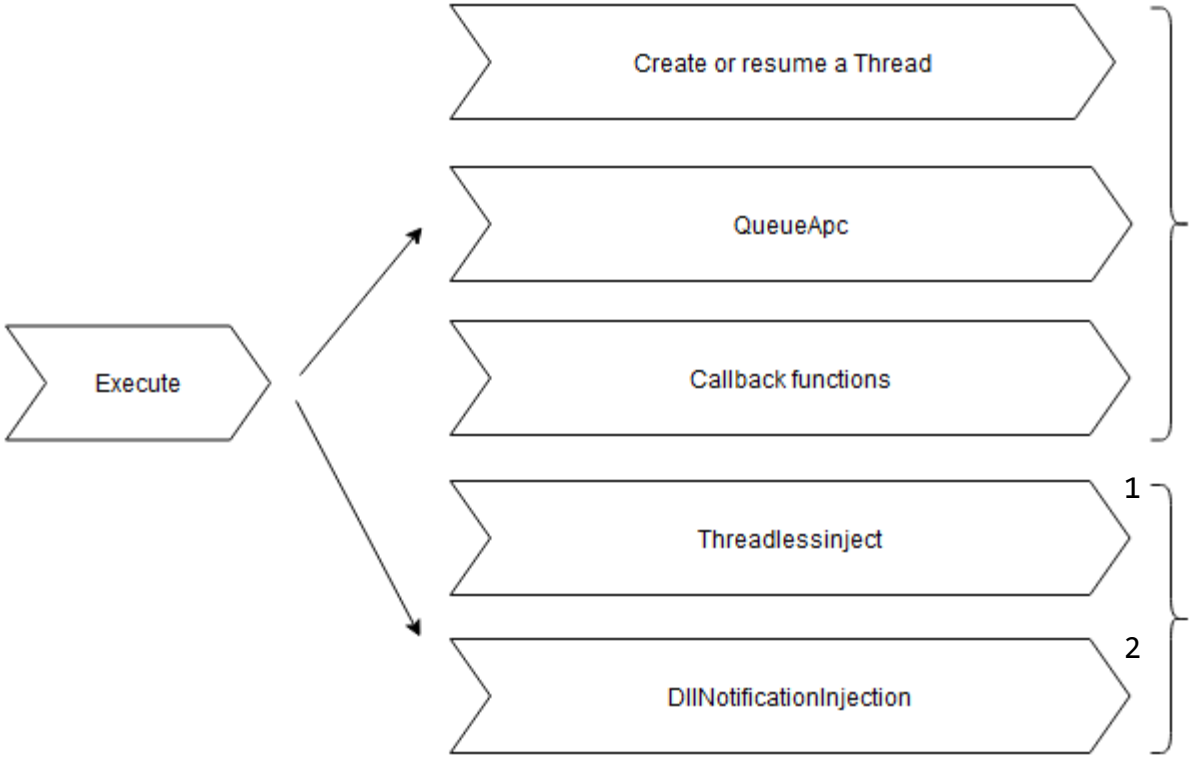
AVOIDING DETECTIONS COMING FROM KERNELLAND



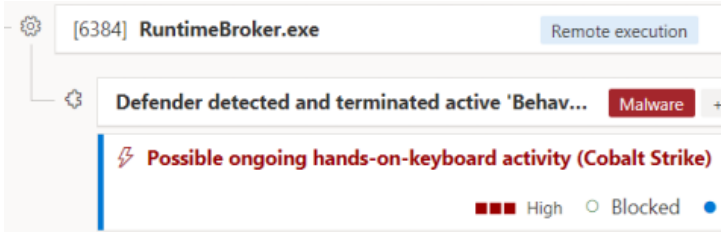
[1] https://www.x86matthew.com/view_post?id=writeprocessmemory_apc

[2] https://www.x86matthew.com/view_post?id=proc_env_injection

AVOIDING DETECTIONS COMING FROM KERNELLAND



- Easy to detect from Kernel
- Especially when using unbacked memory
- Memory Scan == Game lost



- Much harder to detect – due to no
  - Kernel Callback
  - ETWti notification

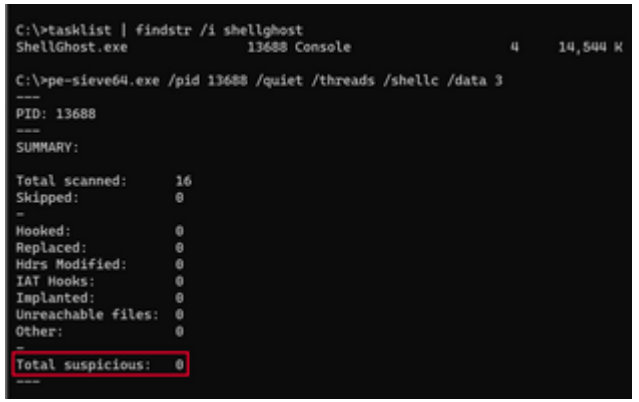
[1] https://github.com/CCob/ThreadlessInject

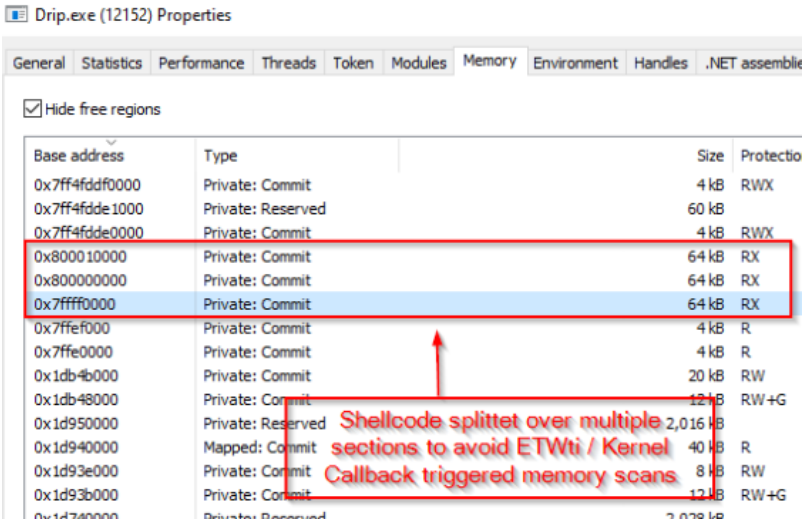
[2] https://github.com/ShorSec/DllNotificationInjection

AVOIDING DETECTIONS COMING FROM KERNELLAND

Think different - Preventing the memory scanner from finding malicious code:

- Split Payload over multiple sections – DripLoader [1]
  - Split it over non concatenated sections - PageSplit [2]
- Hiding it with Hardware-Breakpoints - ShellGhost [3]





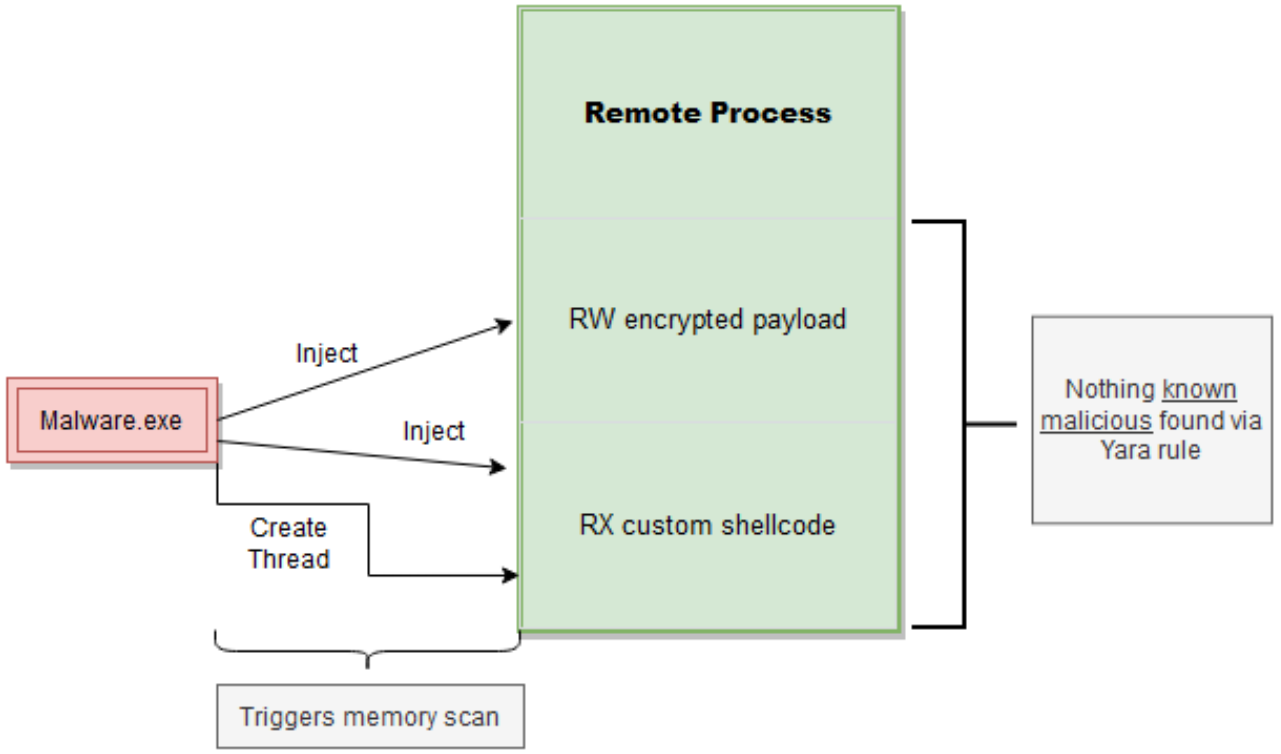
[1] https://github.com/xuanxuan0/DripLoader

[2] https://github.com/x0reaxeax/PageSplit

[3] https://github.com/lem0nSec/ShellGhost

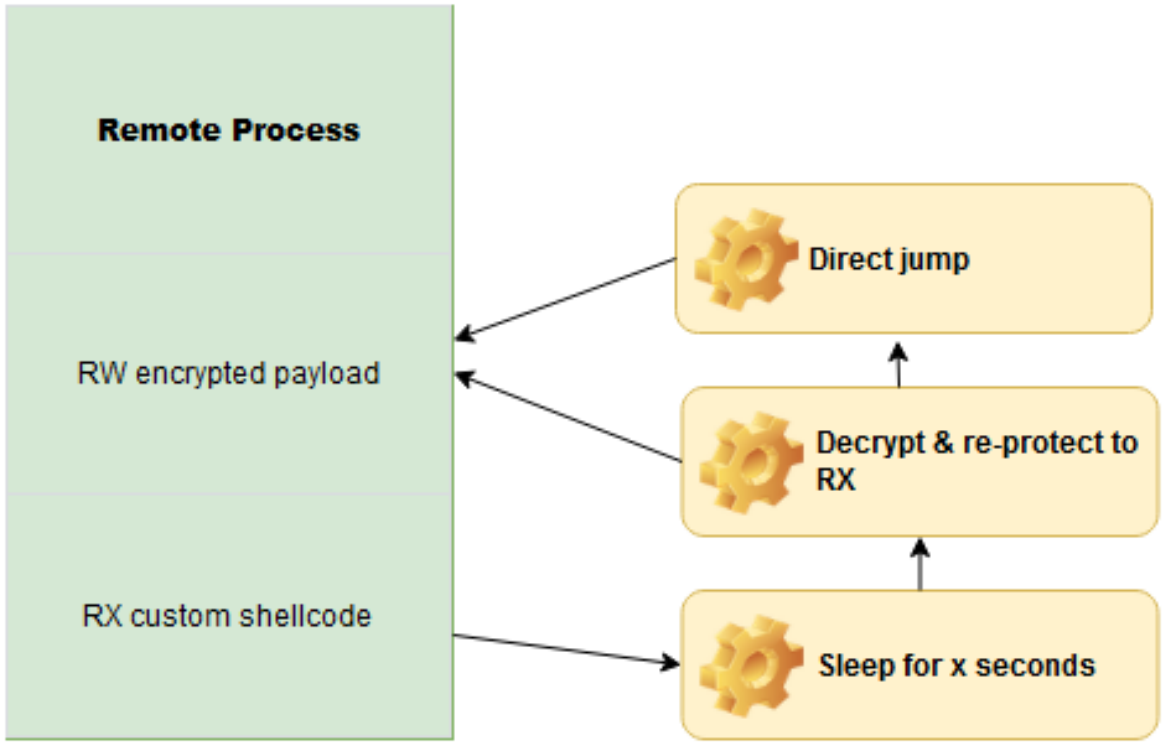
AVOIDING DETECTIONS COMING FROM KERNELLAND

Idea for another approach



AVOIDING DETECTIONS COMING FROM KERNELLAND

Idea for another approach



AVOIDING DETECTIONS COMING FROM KERNELLAND

Difference to existing tooling

- Encoders such as SGN - https://github.com/EgeBalci/sgn
  - Real shellcode is still in the memory region which get's executed
  - Can potentially be unpacked/debugged on runtime
  - Not effective anymore against some EDRs

05

PROOF OF CONCEPT

PROOF OF CONCEPT – CARO KANN

```
[*] Target Process: 15076
[*] pHandle: 168
[*] Writing shellcode into remote process memory: 000001C1A8390000
[+] WriteProcessMemory: 1
    \-- bytes written: 201820

[*] Encrypted shellcode was written into the remote process!
-----------------------------------------------------------------
[*] Allocating memory for our custom shellcode, which will decrypt and re-protect
[+] VirtualAllocEx success!
[*] Second Shellcode will be written to: 000001C1A83D0000
-----------------------------------------------------------------
[*] Looking for the egg, which will be filled with the first shellcodes memory address
[*] Found egg at index: 1888
[*] Writing allocated memory address into egg
[*] Done.
-----------------------------------------------------------------
[*] Looking for the second egg, which will be filled the same address but to jump there at the end
[*] Found egg at index: 521
[*] Writing memory address into the jump at the end
-----------------------------------------------------------------
[*] Looking for the third egg, which will be filled with the shellcodes size
[*] Found egg at index: 1904
[*] Writing shellcode length into egg: 201820
[+] Successfully allocated remote process memory for the custom shellcode
-----------------------------------------------------------------
[+] WriteProcessMemory: 1
    \-- bytes written: 2015

-----------------------------------------------------------------
[*] Creating remote Thread for the second custom shellcode, which will sleep, decrypt and deprotect plus jump to the first one.
```

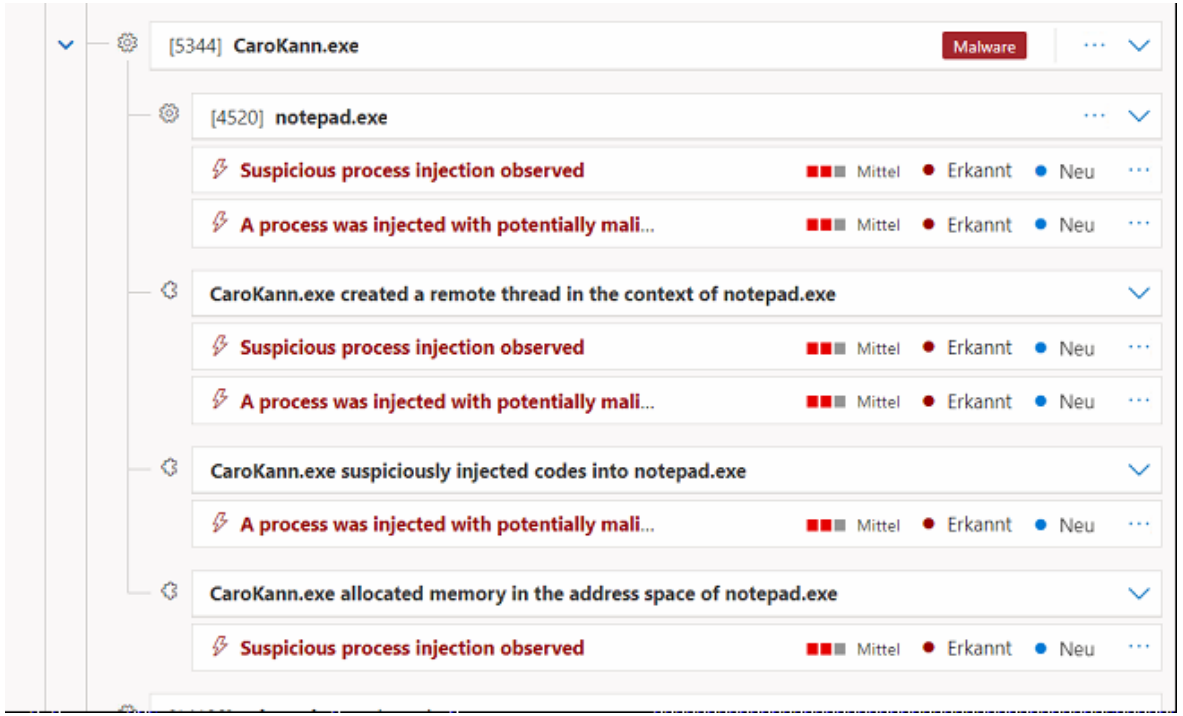
https://www.wikidata.org/wiki/Q192538

PROOF OF CONCEPT – CARO KANN

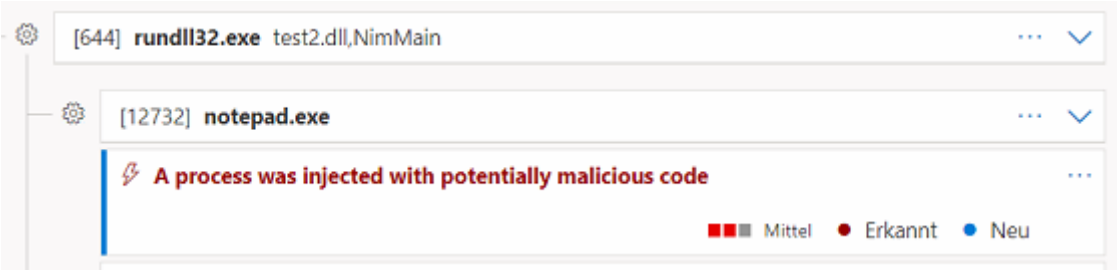
Plain Injection:

PROOF OF CONCEPT – CARO KANN

CaroKann PoC:



Direct Syscall usage:



PROOF OF CONCEPT – CARO KANN

- https://github.com/S3cur3Th1sSh1t/Caro-Kann

PROOF OF CONCEPT – CARO KANN

OPSec improvements:

- Bypass Userland-Hooks for Injection
- Back Payload(s) by legitimate DLL (Module Stomping)
- Load C2-Dlls via the first Shellcode
  - Avoid memory scans triggered by library loads
- Use ThreadlessInject or DLLNotificationInjection instead of Thread Creation
- Write Payload in 4kB chunks

OPSec considerations for C2-Payloads:

- Sleep encryption
- Unhooking | Direct Syscalls[1]
- Proxy module loading

[1] https://github.com/rad9800/misc/blob/main/bypasses/WorkItemLoadLibrary.c

THANK YOU FOR YOUR ATTENTION!

QUESTIONS?

Fabian Mosch