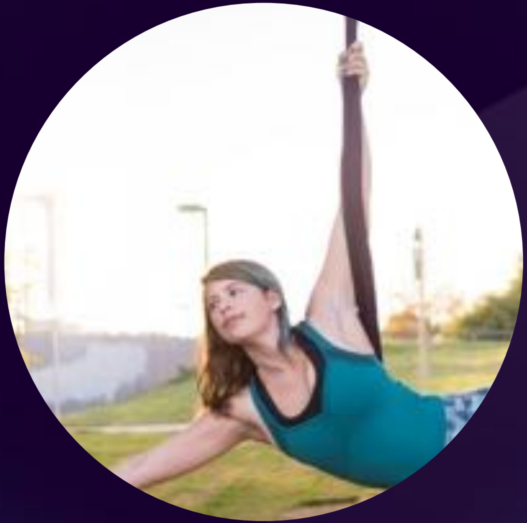
DEEP HOOKS

MONITORING NATIVE EXECUTION IN WOW64 APPLICATIONS

Assaf Carlsbad
@assaf_carlsbad

Yarden Shafir
@yarden_shafir



Yarden

- I started dancing at the age of 7 and later competed with a rhythmic gymnastics team.
- After my military service I practiced dancing and aerial acrobatics.
- Today I teach aerial acrobatics and perform on silks and lyra.

- In my spare time, I'm a security researcher at SentinelOne.

SentinelOne

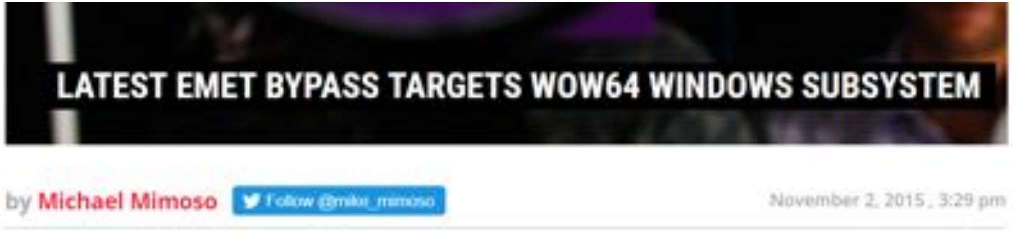
Assaf

- Click to add picture

- Click to add text

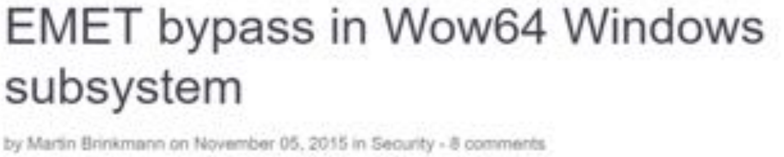
BACKGROUND

- AVs (EDR/EPP/NGAV) do tons of user-mode hooking
  - Used to intercept and potentially block process’ actions
- User-mode hooks can be (and are) bypassed by malicious techniques
  - Some techniques are unique to WoW64 processes



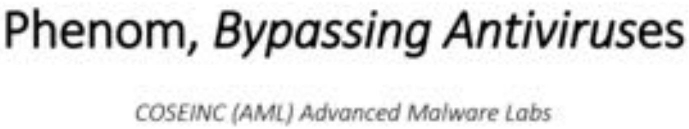
LATEST EMET BYPASS TARGETS WOW64 WINDOWS SUBSYSTEM

by Michael Mimoso Follow @mike_mimoso November 2, 2015 , 3:29 pm



EMET bypass in Wow64 Windows subsystem

by Martin Brinkmann on November 05, 2015 in Security - 8 comments



Phenom, *Bypassing Antiviruses*

COSEINC (AML) Advanced Malware Labs

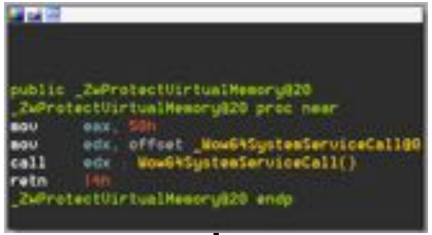
WOW64

- Windows on Windows 64
  - 32-bit apps running on 64-bit systems
- Filesystem & registry redirection (out of scope)
- Syscall handling
  - 2 versions of NTDLL in the process – 32-bit and 64-bit
  - API calls from the app reach the 32-bit NTDLL

WoW64 system call overview

x86 mode

32-bit ntdll



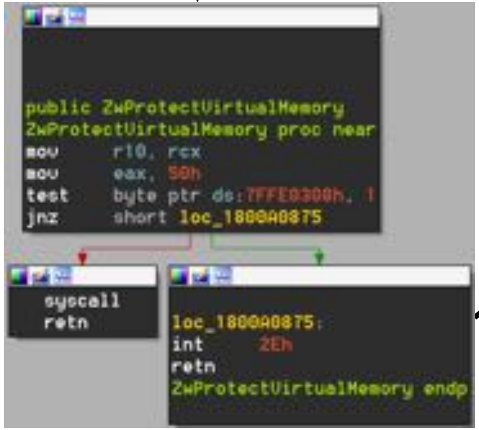
```
public _ZwProtectVirtualMemory@20
_ZwProtectVirtualMemory@20 proc near
mov     eax, 50h
mov     edx, offset _Wow64SystemServiceCall@0
call    edx  Wow64SystemServiceCall()
retn    14h
_ZwProtectVirtualMemory@20 endp
```

Transition to 64-bit

Jmp 0x33:xxxxxxxx

x64 mode

Pointer expansion
Calling convention adjustments
Etc.



```
public ZwProtectVirtualMemory
ZwProtectVirtualMemory proc near
mov     r10, rcx
mov     eax, 50h
test    byte ptr ds:7FFE0308h, 1
jnz     short loc_1800A0875
```

```
syscall
retn
```

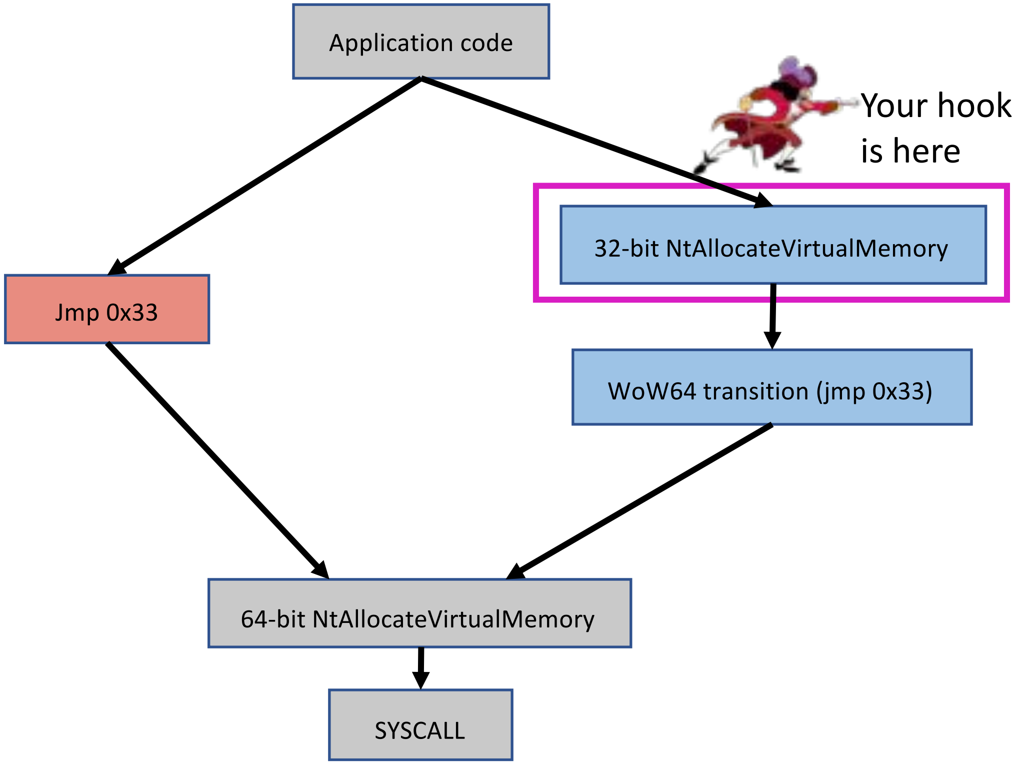
```
loc_1800A0875:
int     2Eh
retn
ZwProtectVirtualMemory endp
```

64-bit ntdll

OS kernel

HEAVEN'S GATE

- A technique for calling the 64-bit API function without going through the 32-bit API
- Abuses the JMP 0x33 control transfer
- Used ITW by various malware


Application code
Your hook is here
32-bit NtAllocateVirtualMemory
Jmp 0x33
WoW64 transition (jmp 0x33)
64-bit NtAllocateVirtualMemory
SYSCALL

http://rce.co/knockin-on-heavens-gate-dynamic-processor-mode-switching/

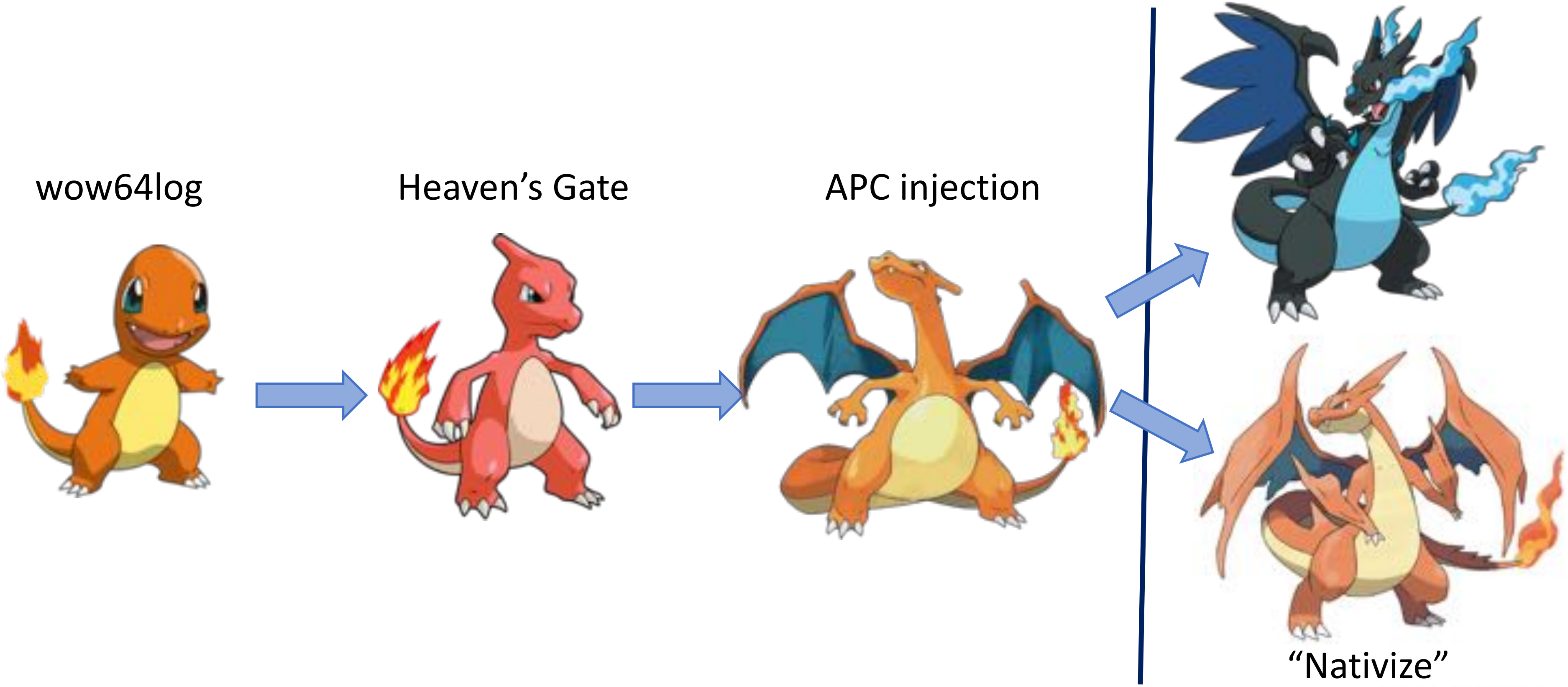
THE SOLUTION

- Hook 64-bit APIs in WoW64 processes!
- But…
  - Need to inject 64-bit code into the process
  - That code should run in a difficult environment
  - No hooking library we are aware of can do this out-of-the-box

INJECTION

- Lots of injection methods exist out there
- Most can only inject a DLL which has the same bitness as the target process
- We need to do something unique - inject a 64-bit DLL into a WoW64 process

INJECTION CONT.



wow64log

Heaven’s Gate

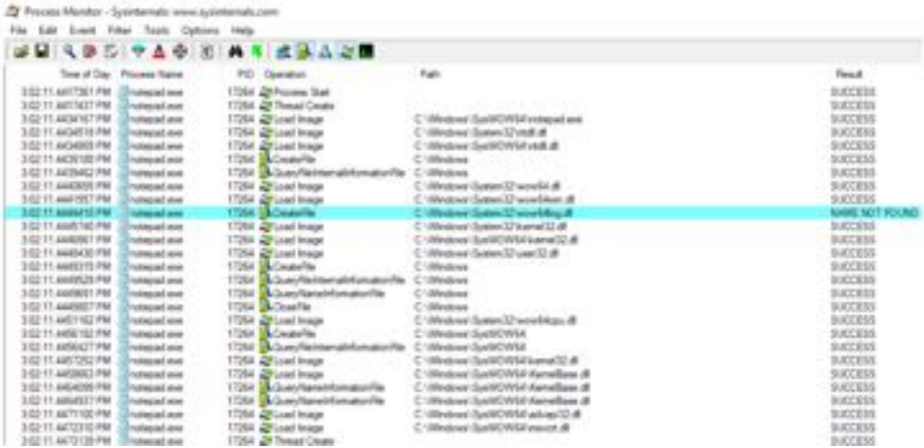
APC injection

“Thunkless”

“Nativize”

INJECTION #1 – WOW64LOG.DLL

- Wow64log.dll
  - A DLL loaded automatically during WoW64 initialization
  - Not shipped by default, so can be created in system32
  - Easy – just name your DLL wow64log.dll

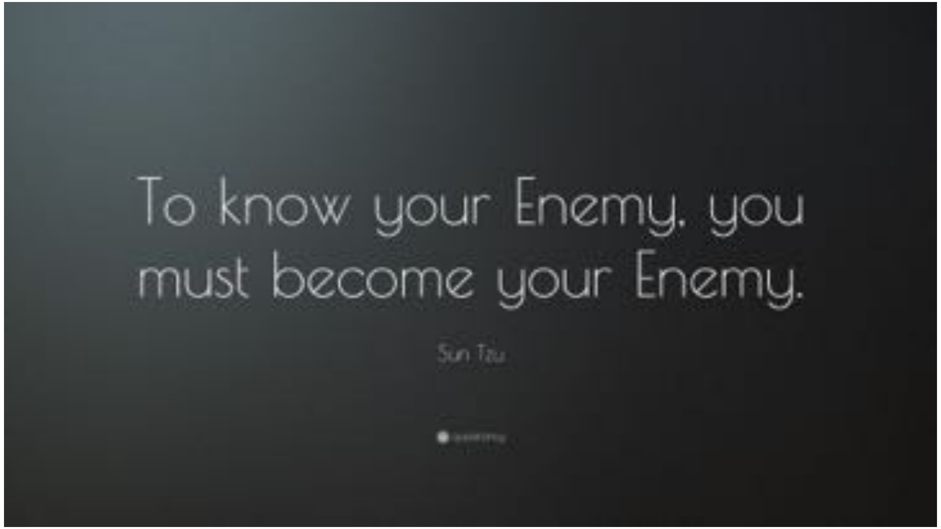


Time of Day	Process Name	PID	Operation	Path	Result
3:02:11.[illegible] PM	notepad.exe	17264	Process Start		SUCCESS
3:02:11.[illegible] PM	notepad.exe	17264	Thread Create		SUCCESS
3:02:11.[illegible] PM	notepad.exe	17264	Load Image	C:\Windows\SysWOW64\notepad.exe	SUCCESS
3:02:11.[illegible] PM	notepad.exe	17264	Load Image	C:\Windows\System32\ntdll.dll	SUCCESS
3:02:11.[illegible] PM	notepad.exe	17264	Load Image	C:\Windows\SysWOW64\ntdll.dll	SUCCESS
3:02:11.[illegible] PM	notepad.exe	17264	CreateFile	C:\Windows	SUCCESS
3:02:11.[illegible] PM	notepad.exe	17264	QueryFileInternalInformationFile	C:\Windows	SUCCESS
3:02:11.[illegible] PM	notepad.exe	17264	Load Image	C:\Windows\System32\wow64.dll	SUCCESS
3:02:11.[illegible] PM	notepad.exe	17264	Load Image	C:\Windows\System32\wow64win.dll	SUCCESS
3:02:11.[illegible] PM	notepad.exe	17264	CreateFile	C:\Windows\System32\wow64log.dll	NAME NOT FOUND
3:02:11.[illegible] PM	notepad.exe	17264	Load Image	C:\Windows\System32\kernel32.dll	SUCCESS
3:02:11.[illegible] PM	notepad.exe	17264	Load Image	C:\Windows\SysWOW64\kernel32.dll	SUCCESS
3:02:11.[illegible] PM	notepad.exe	17264	Load Image	C:\Windows\System32\user32.dll	SUCCESS
3:02:11.[illegible] PM	notepad.exe	17264	CreateFile	C:\Windows	SUCCESS
3:02:11.[illegible] PM	notepad.exe	17264	QueryFileInternalInformationFile	C:\Windows	SUCCESS
3:02:11.[illegible] PM	notepad.exe	17264	QueryNameInformationFile	C:\Windows	SUCCESS
3:02:11.[illegible] PM	notepad.exe	17264	CloseFile	C:\Windows	SUCCESS
3:02:11.[illegible] PM	notepad.exe	17264	Load Image	C:\Windows\System32\wow64cpu.dll	SUCCESS
3:02:11.[illegible] PM	notepad.exe	17264	CreateFile	C:\Windows\SysWOW64	SUCCESS
3:02:11.[illegible] PM	notepad.exe	17264	QueryFileInternalInformationFile	C:\Windows\SysWOW64	SUCCESS
3:02:11.[illegible] PM	notepad.exe	17264	Load Image	C:\Windows\SysWOW64\kernel32.dll	SUCCESS
3:02:11.[illegible] PM	notepad.exe	17264	Load Image	C:\Windows\SysWOW64\KernelBase.dll	SUCCESS
3:02:11.[illegible] PM	notepad.exe	17264	QueryNameInformationFile	C:\Windows\SysWOW64\KernelBase.dll	SUCCESS
3:02:11.[illegible] PM	notepad.exe	17264	QueryNameInformationFile	C:\Windows\SysWOW64\KernelBase.dll	SUCCESS
3:02:11.[illegible] PM	notepad.exe	17264	Load Image	C:\Windows\SysWOW64\advapi32.dll	SUCCESS
3:02:11.[illegible] PM	notepad.exe	17264	Load Image	C:\Windows\SysWOW64\msvcrt.dll	SUCCESS
3:02:11.[illegible] PM	notepad.exe	17264	Thread Create		SUCCESS

http://waleedassar.blogspot.com/2013/01/wow64logdll.html

INJECTION #2 – HEAVEN’S GATE

- To detect the malware, you must become the malware

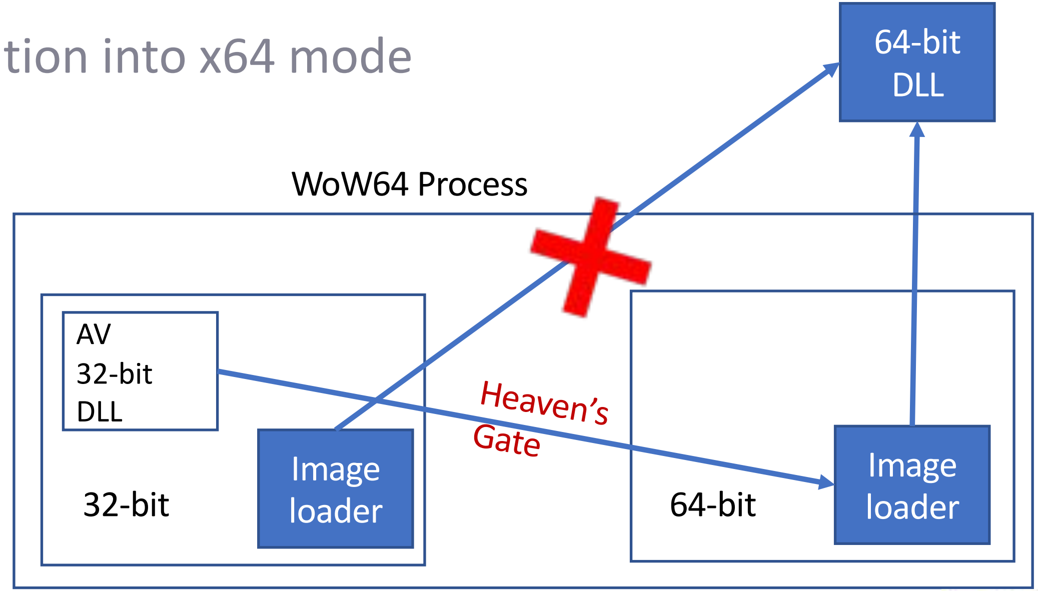


To know your Enemy, you must become your Enemy.

Sun Tzu

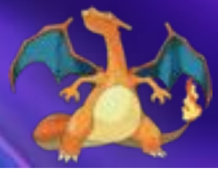
INJECTION #2 – HEAVEN'S GATE

- 2 image loaders: 32-bit and 64-bit
- Requires injection of 32-bit code first
- Use Heaven's Gate to transition into x64 mode
- Call 64-bit *LdrLoadDll()*



SentinelOne

INJECTION #3 - APC



- Asynchronous Procedure Call
- Kernel mechanism that provides a way to execute a custom routine in the context of a particular thread
- User-mode APCs
  - Runs with user-mode permissions
  - Target thread must enter alertable wait state
  - Handled by *ntdll!KiUserApcDispatcher*
  - Usually queued from a kernel driver

https://wikileaks.org/ciav7p1/cms/page_7995519.html

INJECTION #3 - APC

- Popular among AVs and intelligence agencies
  - Queue an APC to *LdrLoadDll()/LoadLibrary()*
  - Used to inject a DLL with the same bitness as the target process
- In WoW64 processes – APCs can run in 32-bit or 64-bit mode
  - Can call 64-bit loader functions!

Vault 7: CIA Hacking Tools Revealed

Navigation: » Directory » Remote Development Branch (RDB) » RDB Home » Umbrage » Component Library » Kernel Land

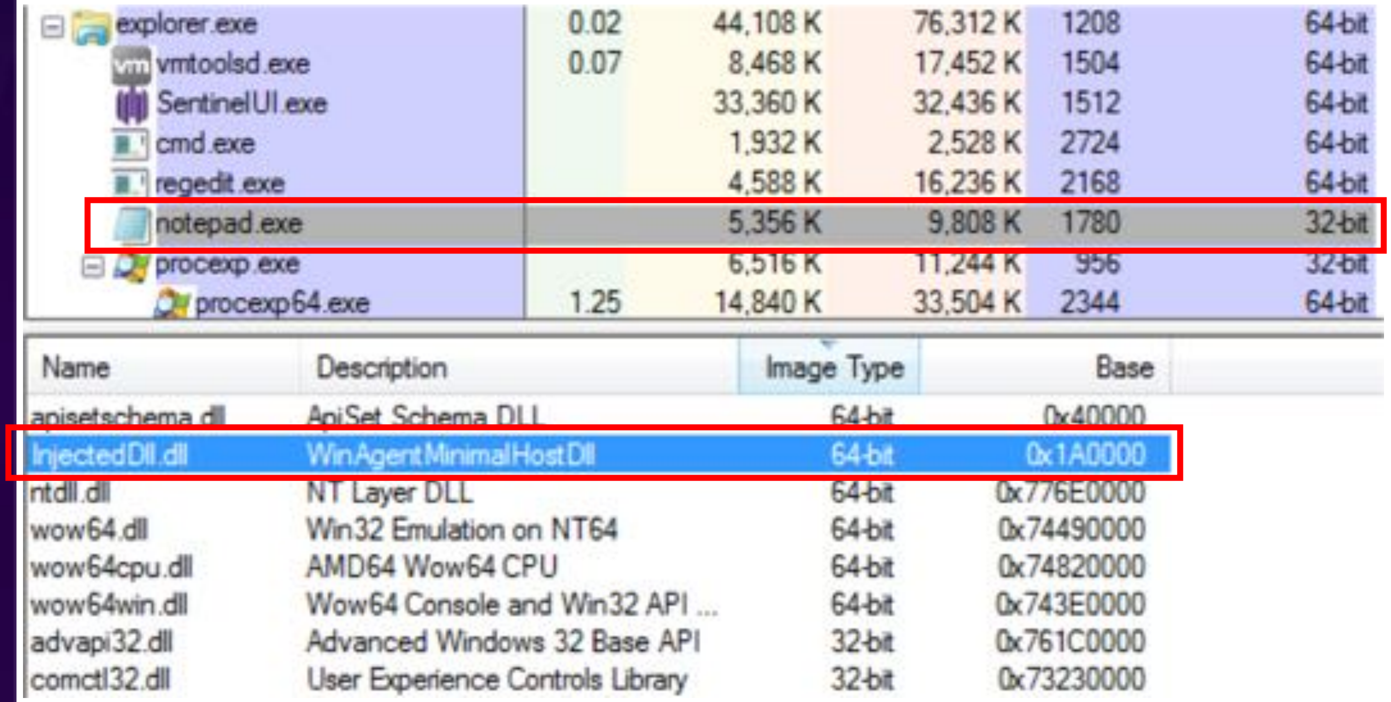
Kernel to User land: APC injection

Overview

When running in Kernel mode, it may be necessary to inject code into a User-land process. There are two ways that Asynchronous Procedure Calls (APCs) can be used to accomplish this goal.

```
void InjectDllByApc(_In_ PKTHREAD pTargetThread)
{
    ZwAllocateVirtualMemory(ZwCurrentProcess(), &ctx, 0, &ctxSize, MEM_COMMIT | MEM_RESERVE,
PAGE_READWRITE);
    ctx->pLdrLoadDll64 = pLdrLoadDll;

    RtlInitEmptyUnicodeString(&ctx->DllName, ctx->Buffer, sizeof(ctx->Buffer));
    RtlUnicodeStringCopyString(&ctx->DllName, L“injectedDll.dll”);

    ZwAllocateVirtualMemory(ZwCurrentProcess(), &pUserApcCode, 0, &apcRoutineSize, MEM_COMMIT | MEM_RESERVE,
PAGE_EXECUTE_READWRITE);

    // Copy the code of 'AdapterThunk' into user-space.
    RtlCopyMemory(pUserApcCode, AdapterThunk, AdapterThunkSize());
    KeInitializeApc(pKapcObj, pTargetThread, OriginalApcEnvironment, KernelApcRoutine, NULL, pUserApcCode,
UserMode, ctx);
    KeInsertQueueApc(pKapcObj, NULL, NULL, 0);
}

void AdapterThunk(_In_ PVOID NormalContext, _In_ PVOID Sysarg1, _In_ PVOID Sysarg2)
{
    HANDLE hModule;
    PINJECTION_CONTEXT ctx = (PINJECTION_CONTEXT)NormalContext;
    ctx->pLdrLoadDll64(0, 0, &ctx->DllName, &hModule);
}
```

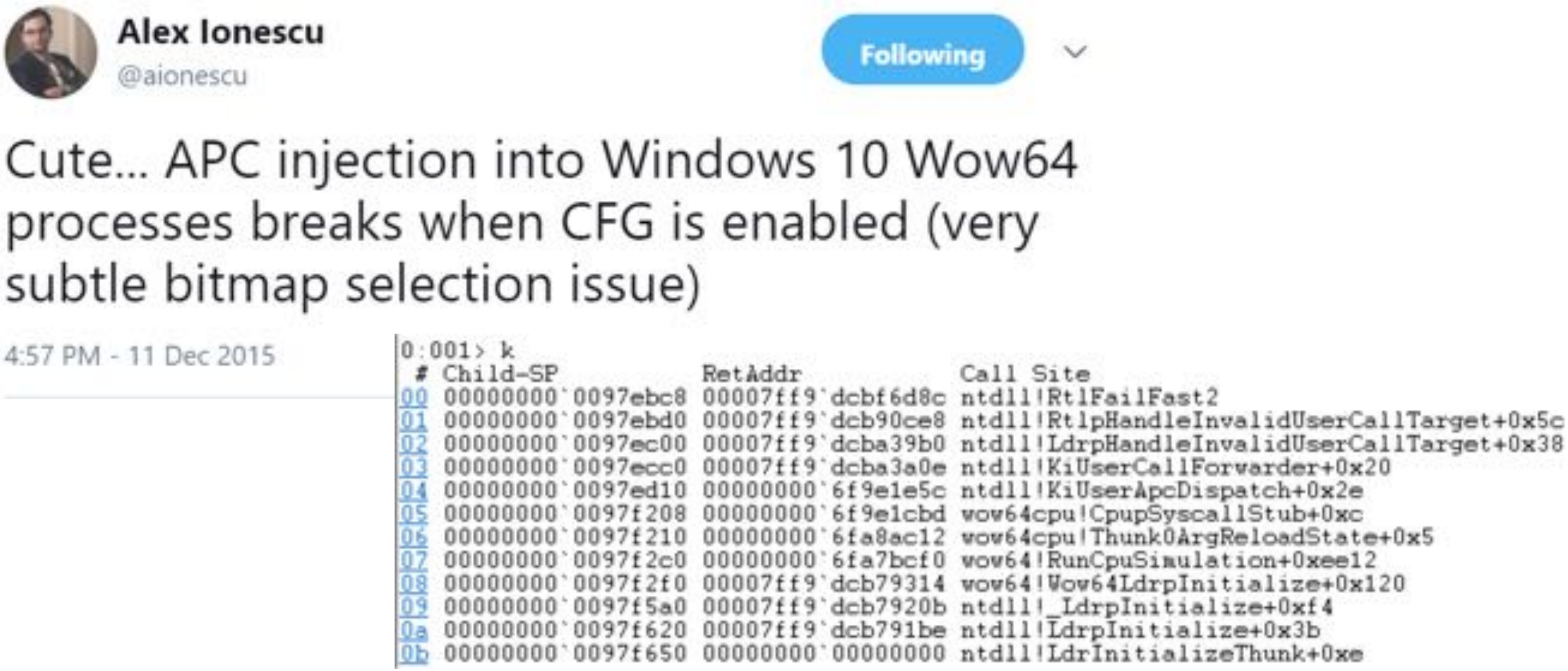
SUCCESS!



explorer.exe	0.02	44,108 K	76,312 K	1208	64-bit
vmtoolsd.exe	0.07	8,468 K	17,452 K	1504	64-bit
SentinelUI.exe		33,360 K	32,436 K	1512	64-bit
cmd.exe		1,932 K	2,528 K	2724	64-bit
regedit.exe		4,588 K	16,236 K	2168	64-bit
notepad.exe		5,356 K	9,808 K	1780	32-bit
procexp.exe		6,516 K	11,244 K	956	32-bit
procexp64.exe	1.25	14,840 K	33,504 K	2344	64-bit

Name	Description	Image Type	Base
apisetschema.dll	ApiSet Schema DLL	64-bit	0x40000
InjectedDll.dll	WinAgentMinimalHostDll	64-bit	0x1A0000
ntdll.dll	NT Layer DLL	64-bit	0x776E0000
wow64.dll	Win32 Emulation on NT64	64-bit	0x74490000
wow64cpu.dll	AMD64 Wow64 CPU	64-bit	0x74820000
wow64win.dll	Wow64 Console and Win32 API ...	64-bit	0x743E0000
advapi32.dll	Advanced Windows 32 Base API	32-bit	0x761C0000
comctl32.dll	User Experience Controls Library	32-bit	0x73230000

OR NOT?



Alex Ionescu
@aionescu

Cute... APC injection into Windows 10 Wow64 processes breaks when CFG is enabled (very subtle bitmap selection issue)

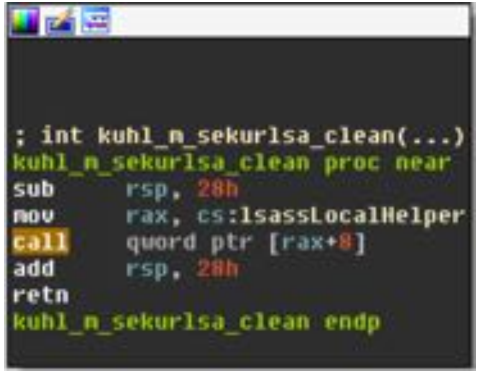
4:57 PM - 11 Dec 2015

```
0:001> k
 # Child-SP          RetAddr           Call Site
00 00000000`0097ebc8 00007ff9`dcbf6d8c ntdll!RtlFailFast2
01 00000000`0097ebd0 00007ff9`dcb90ce8 ntdll!RtlpHandleInvalidUserCallTarget+0x5c
02 00000000`0097ec00 00007ff9`dcba39b0 ntdll!LdrpHandleInvalidUserCallTarget+0x38
03 00000000`0097ecc0 00007ff9`dcba3a0e ntdll!KiUserCallForwarder+0x20
04 00000000`0097ed10 00000000`6f9e1e5c ntdll!KiUserApcDispatch+0x2e
05 00000000`0097f208 00000000`6f9e1cbd wow64cpu!CpupSyscallStub+0xc
06 00000000`0097f210 00000000`6fa8ac12 wow64cpu!Thunk0ArgReloadState+0x5
07 00000000`0097f2c0 00000000`6fa7bcf0 wow64!RunCpuSimulation+0xee12
08 00000000`0097f2f0 00007ff9`dcb79314 wow64!Wow64LdrpInitialize+0x120
09 00000000`0097f5a0 00007ff9`dcb7920b ntdll!_LdrpInitialize+0xf4
0a 00000000`0097f620 00007ff9`dcb791be ntdll!LdrpInitialize+0x3b
0b 00000000`0097f650 00000000`00000000 ntdll!LdrInitializeThunk+0xe
```

CFG – CONTROL FLOW GUARD

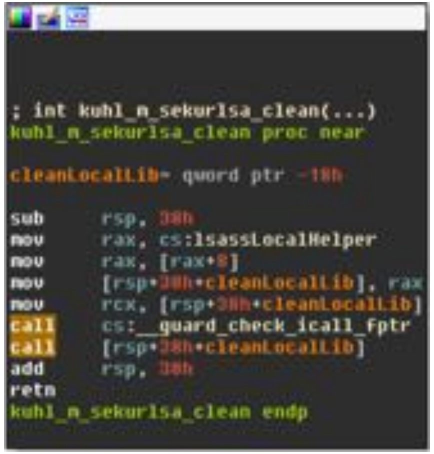
- Exploit mitigation feature introduced in Windows 8.1
- Only allows indirect calls to valid call targets
- Indirect calls to invalid targets will crash the process

WITHOUT CFG



```
; int kuhl_m_sekurlsa_clean(...)
kuhl_m_sekurlsa_clean proc near
sub     rsp, 28h
mov     rax, cs:lsassLocalHelper
call    qword ptr [rax+8]
add     rsp, 28h
retn
kuhl_m_sekurlsa_clean endp
```

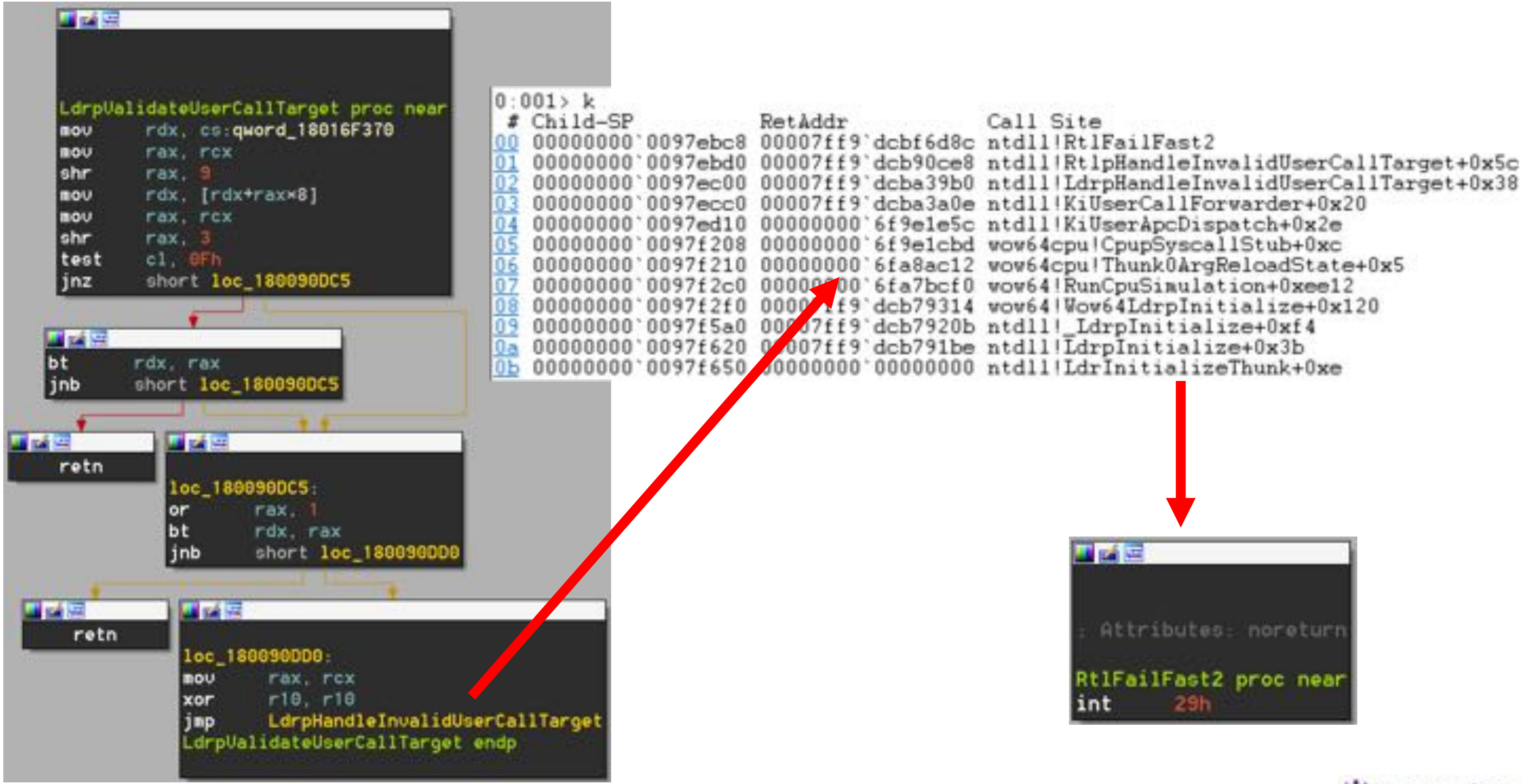
WITH CFG



```
; int kuhl_m_sekurlsa_clean(...)
kuhl_m_sekurlsa_clean proc near

cleanLocalLib= qword ptr -18h

sub     rsp, 38h
mov     rax, cs:lsassLocalHelper
mov     rax, [rax+8]
mov     [rsp+38h+cleanLocalLib], rax
mov     rcx, [rsp+38h+cleanLocalLib]
call    cs:__guard_check_icall_fptr
call    [rsp+38h+cleanLocalLib]
add     rsp, 38h
retn
kuhl_m_sekurlsa_clean endp
```

VALID CALL TARGETS

- Valid call targets:
  - For images – Start addresses of functions
  - For executable private memory allocations – All of the buffer
- CFG uses a bitmap to mark valid executable addresses
- Each bit represents 8 bytes in the process' address space
- Valid call targets are marked in the bitmap whenever new executable memory is introduced into the process



```
LdrpValidateUserCallTarget proc near
mov     rdx, cs:qword_18016F370
mov     rax, rcx
shr     rax, 9
mov     rdx, [rdx+rax*8]
mov     rax, rcx
shr     rax, 3
test    cl, 0Fh
jnz     short loc_180090DC5
```

```
bt      rdx, rax
jnb     short loc_180090DC5
```

```
retn
```

```
loc_180090DC5:
or      rax, 1
bt      rdx, rax
jnb     short loc_180090DD0
```

```
retn
```

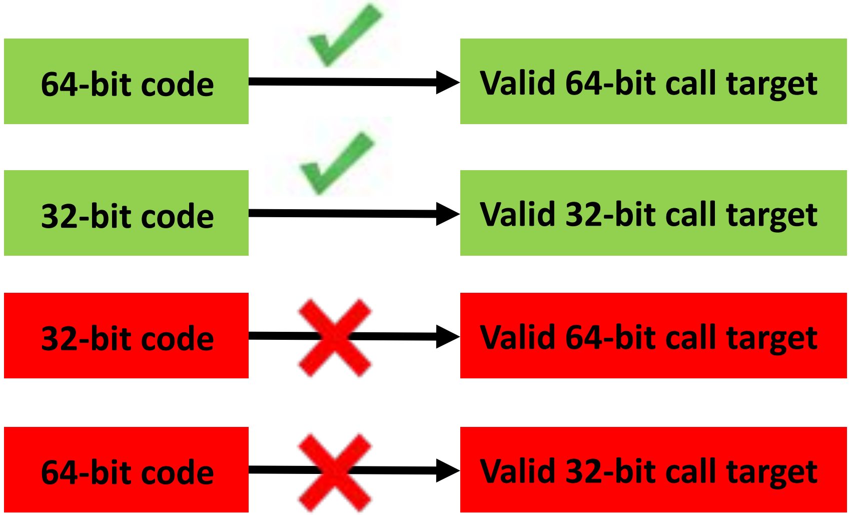
```
loc_180090DD0:
mov     rax, rcx
xor     r10, r10
jmp     LdrpHandleInvalidUserCallTarget
LdrpValidateUserCallTarget endp
```

```
0:001> k
 # Child-SP          RetAddr           Call Site
00 00000000`0097ebc8 00007ff9`dcbf6d8c ntdll!RtlFailFast2
01 00000000`0097ebd0 00007ff9`dcb90ce8 ntdll!RtlpHandleInvalidUserCallTarget+0x5c
02 00000000`0097ec00 00007ff9`dcba39b0 ntdll!LdrpHandleInvalidUserCallTarget+0x38
03 00000000`0097ecc0 00007ff9`dcba3a0e ntdll!KiUserCallForwarder+0x20
04 00000000`0097ed10 00000000`6f9e1e5c ntdll!KiUserApcDispatch+0x2e
05 00000000`0097f208 00000000`6f9e1cbd wow64cpu!CpupSyscallStub+0xc
06 00000000`0097f210 00000000`6fa8ac12 wow64cpu!Thunk0ArgReloadState+0x5
07 00000000`0097f2c0 00000000`6fa7bcf0 wow64!RunCpuSimulation+0xee12
08 00000000`0097f2f0 00007ff9`dcb79314 wow64!Wow64LdrpInitialize+0x120
09 00000000`0097f5a0 00007ff9`dcb7920b ntdll!_LdrpInitialize+0xf4
0a 00000000`0097f620 00007ff9`dcb791be ntdll!LdrpInitialize+0x3b
0b 00000000`0097f650 00000000`00000000 ntdll!LdrInitializeThunk+0xe
```

```
; Attributes: noreturn

RtlFailFast2 proc near
int     29h
```

CFG IN WOW64

- WoW64 processes have 2 CFG bitmaps:
  - Native bitmap – for 64-bit code
  - WoW64 bitmap – for 32-bit code
- 2 NTDLLs = 2 validation functions
  - 64-bit NTDLL - native bitmap
  - 32-bit NTDLL - WoW64 bitmap

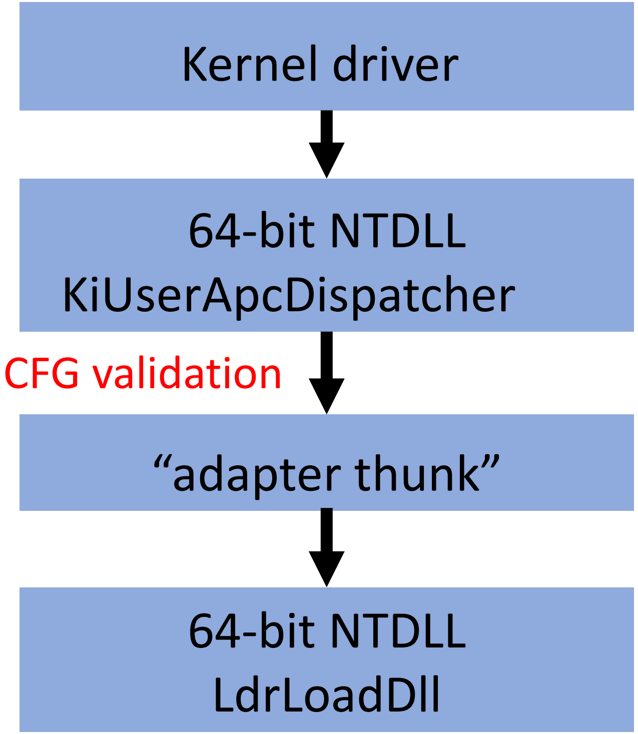


http://www.alex-ionescu.com/?p=300

```
PVOID * MiSelectCfgBitmap(PEPROCESS CurrentProcess, PVOID Address, PSEGMENT Segment)
{
    if ( (CurrentProcess.WoW64Process != NULL) &&
        (Address < _4gb) &&
        ( (Segment == NULL) || (MiSelectBitMapForImage(Segment) == DynamicBaseBitMap32) ) )
    {
        return Wow64CFGBitmap;
    }
    else
    {
        return NativeCFGBitmap;
    }
}
```

BACK TO APC INJECTION

- APC target contains 64-bit code
- Handled by 64-bit *KiUserApcDispatcher*
- Adapter thunk is never called
- Adapter thunk is not considered a valid call target by CFG validation routine


Kernel driver

64-bit NTDLL
KiUserApcDispatcher

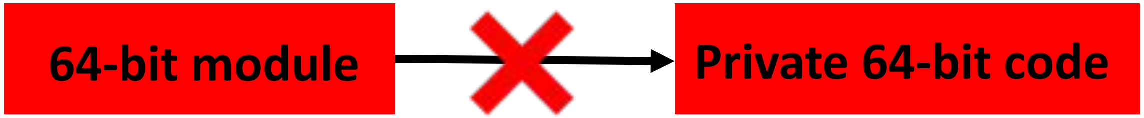
CFG validation

"adapter thunk"

64-bit NTDLL
LdrLoadDll

SO WHERE’S THE PROBLEM?

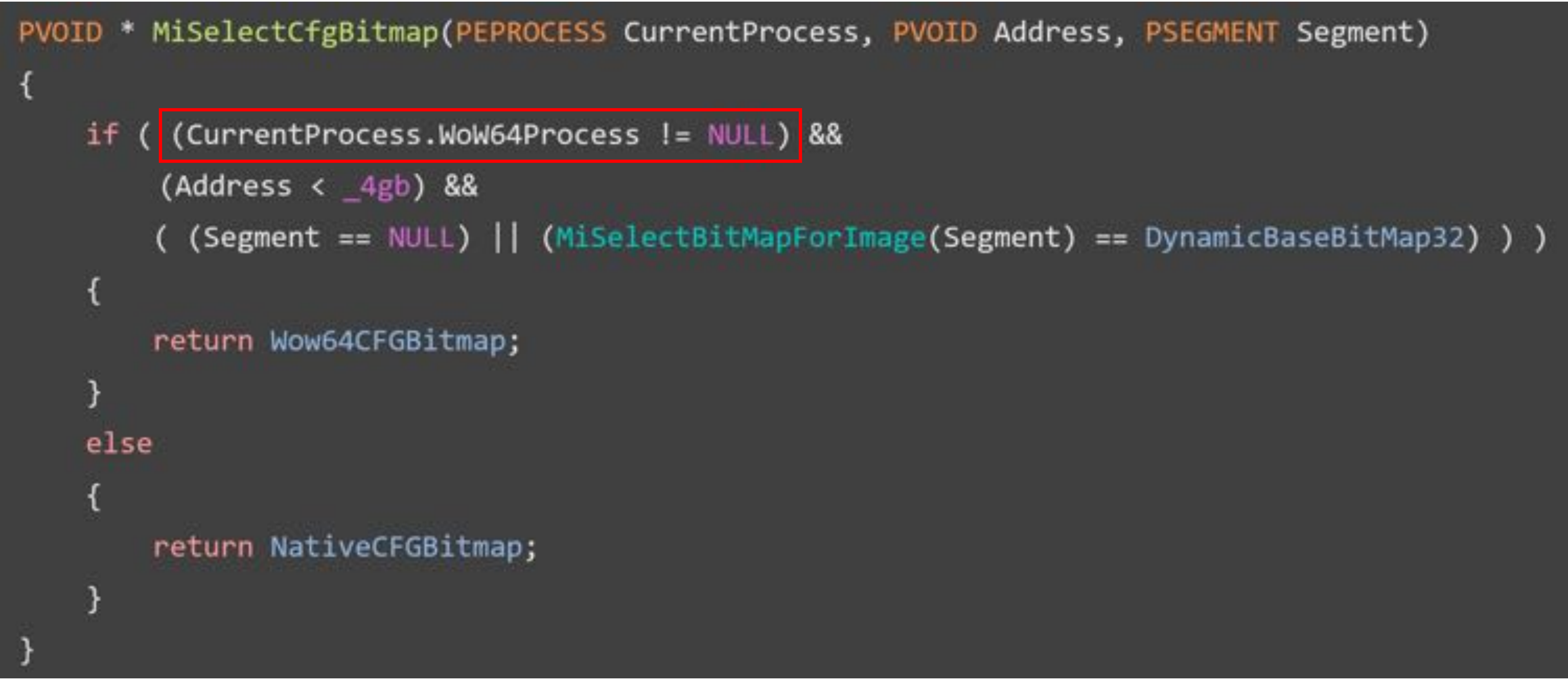
- Our “adapter thunk” is not marked in the native CFG bitmap
- Only 64-bit modules are marked in the native CFG bitmap
- Private memory allocations are always marked in the WoW64 bitmap



64-bit module

Private 64-bit code

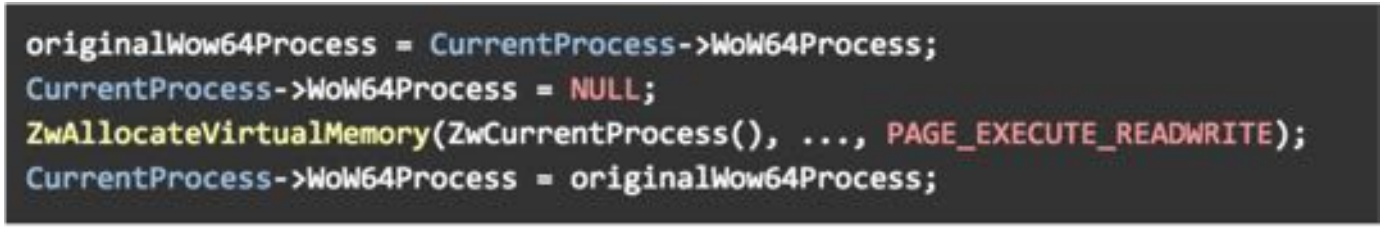
OPTION #1 – “NATIVIZE” THE PROCESS



```
PVOID * MiSelectCfgBitmap(PEPROCESS CurrentProcess, PVOID Address, PSEGMENT Segment)
{
    if ( (CurrentProcess.WoW64Process != NULL) &&
        (Address < _4gb) &&
        ( (Segment == NULL) || (MiSelectBitMapForImage(Segment) == DynamicBaseBitMap32) ) )
    {
        return Wow64CFGBitmap;
    }
    else
    {
        return NativeCFGBitmap;
    }
}
```

OPTION #1 – “NATIVIZE” THE PROCESS

- To check if the process is native, the kernel uses the WoW64Process member of the EPROCESS
- If we set *EPROCESS->WoW64Process* to NULL, *MiSelectCfgBitmap* will:
    - Assume that the process is a native one
    - Mark the adapter thunk in the native bitmap



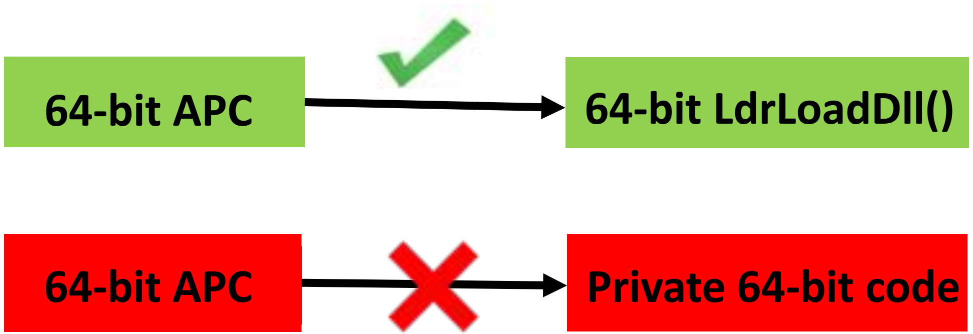
```
originalWow64Process = CurrentProcess->WoW64Process;
CurrentProcess->WoW64Process = NULL;
ZwAllocateVirtualMemory(ZwCurrentProcess(), ..., PAGE_EXECUTE_READWRITE);
CurrentProcess->WoW64Process = originalWow64Process;
```

“NATIVIZE” THE PROCESS – DOWNSIDES

- The EPROCESS structure is undocumented and changes often
- Need to search for WoW64Process heuristically
- Dynamically changing WoW64Process could cause unexpected side effects

OPTION #2 – “THUNKLESS” APC INJECTION

- Private memory is marked in the WoW64 CFG bitmap
- No private memory = no problem
- Can we call the 64-bit *LdrLoadDll()* directly?

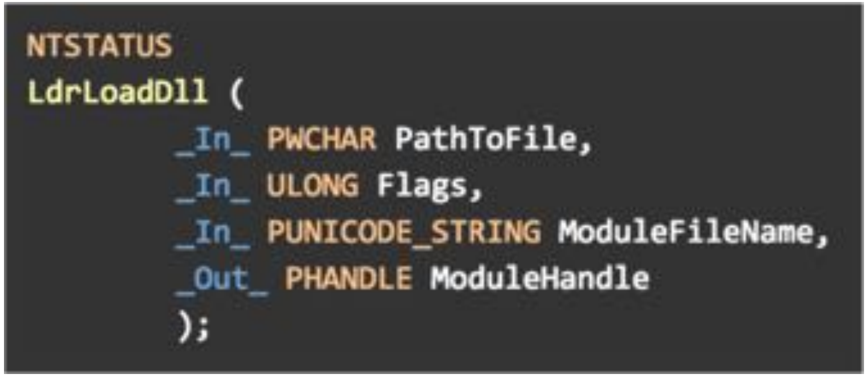


64-bit APC ✓ 64-bit LdrLoadDll()

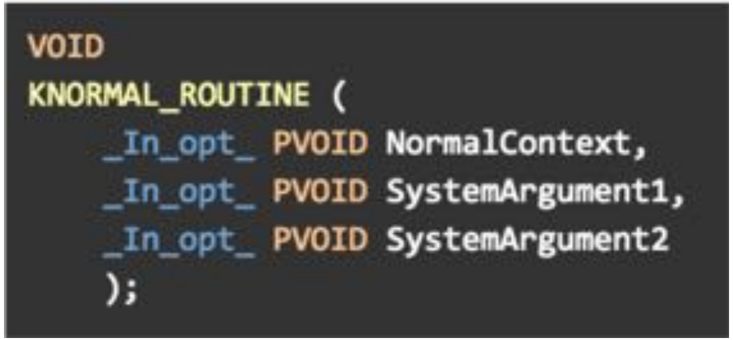
64-bit APC ✗ Private 64-bit code

OOPS...

- An APC routine receives 3 arguments
- *LdrLoadDll()* expects 4 arguments



```
NTSTATUS
LdrLoadDll (
        _In_ PWCHAR PathToFile,
        _In_ ULONG Flags,
        _In_ PUNICODE_STRING ModuleFileName,
        _Out_ PHANDLE ModuleHandle
        );
```



```
VOID
KNORMAL_ROUTINE (
    _In_opt_ PVOID NormalContext,
    _In_opt_ PVOID SystemArgument1,
    _In_opt_ PVOID SystemArgument2
    );
```

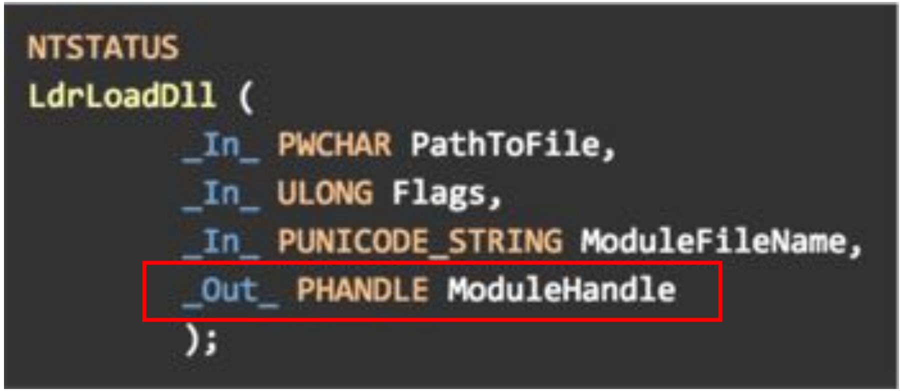
NOT A FAILURE YET

- Because of x64 calling convention, every function implicitly receives 4 arguments
  - First 4 arguments are passed in registers: rcx, rdx, r8, r9
  - We can control the first 3 parameters passed by the APC
  - Whatever value is in r9 will be interpreted as the fourth

```
push p5
mov r9, p4
mov r8, p3
mov rdx, p2
mov rcx, p1
call func
```

REQUIREMENTS

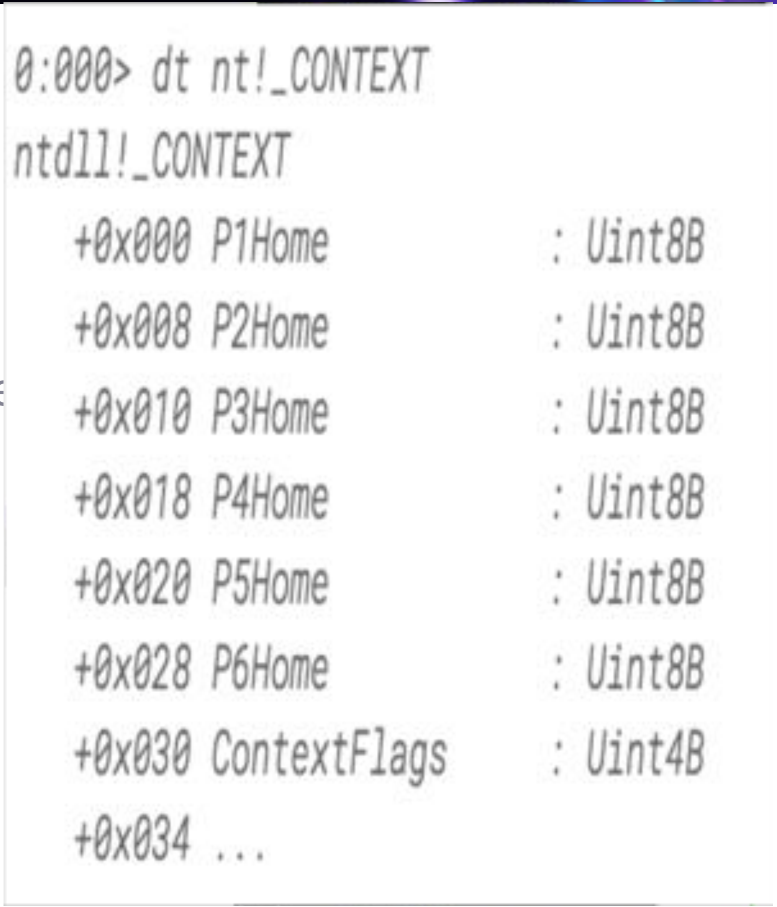
- Fourth parameter is an output parameter
- Needs to be a pointer to writable memory
- Needs to be memory we can overwrite without messing things up



```
NTSTATUS
LdrLoadDll (
          _In_ PWCHAR PathToFile,
          _In_ ULONG Flags,
          _In_ PUNICODE_STRING ModuleFileName,
          _Out_ PHANDLE ModuleHandle
          );
```

WHAT'S IN R9?

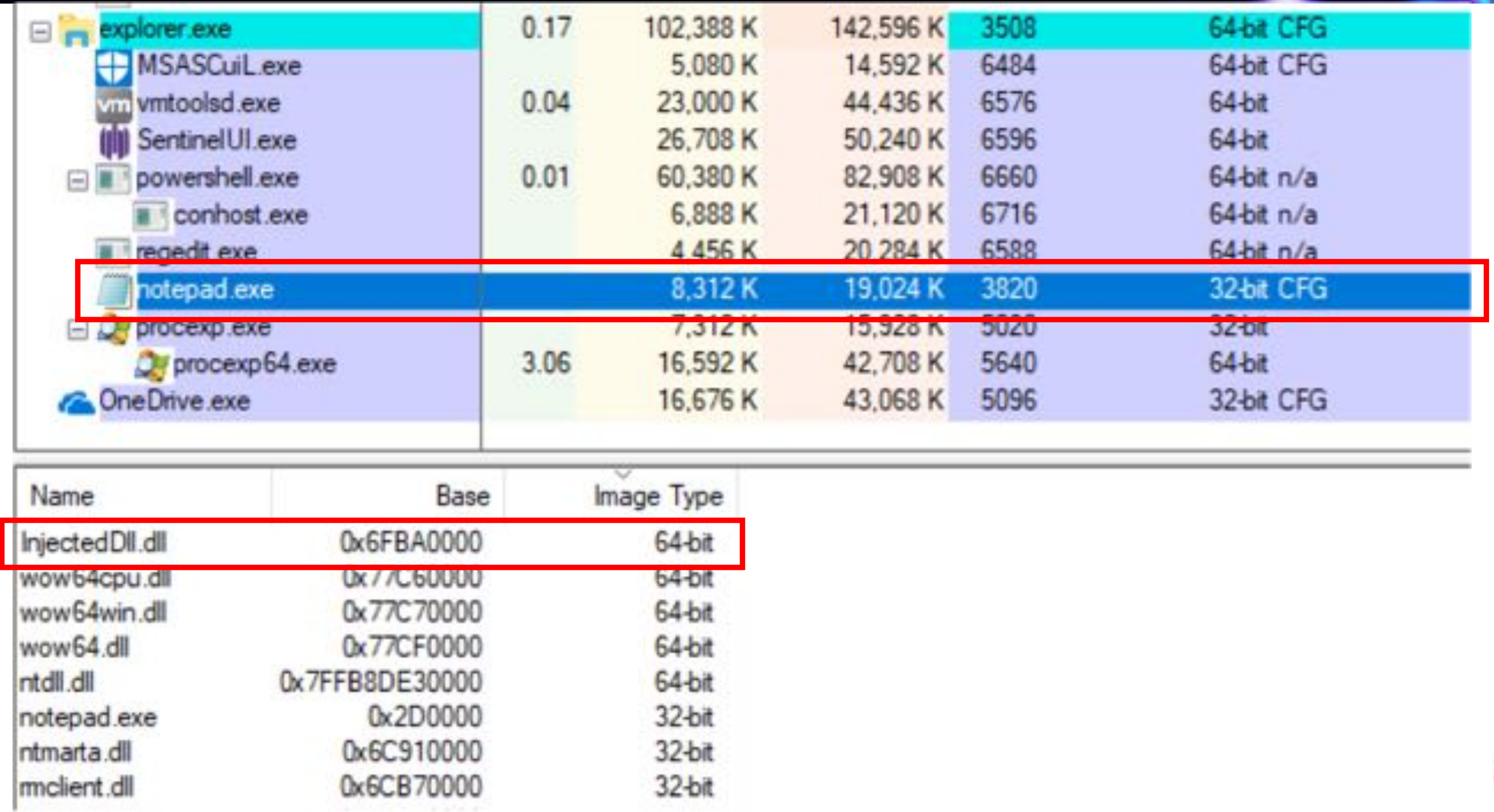
- R9 holds a *CONTEXT* structure
- Will be used to resume the thread after APC dispatch (via *NtContinue*)
- First few members don't hold CPU-relate data and can be overwritten



```
0:000> dt nt!_CONTEXT
ntdll!_CONTEXT
   +0x000 P1Home           : Uint8B
   +0x008 P2Home           : Uint8B
   +0x010 P3Home           : Uint8B
   +0x018 P4Home           : Uint8B
   +0x020 P5Home           : Uint8B
   +0x028 P6Home           : Uint8B
   +0x030 ContextFlags     : Uint4B
   +0x034 ...
```

http://www.nynaeve.net/?p=202

SUCCESS!

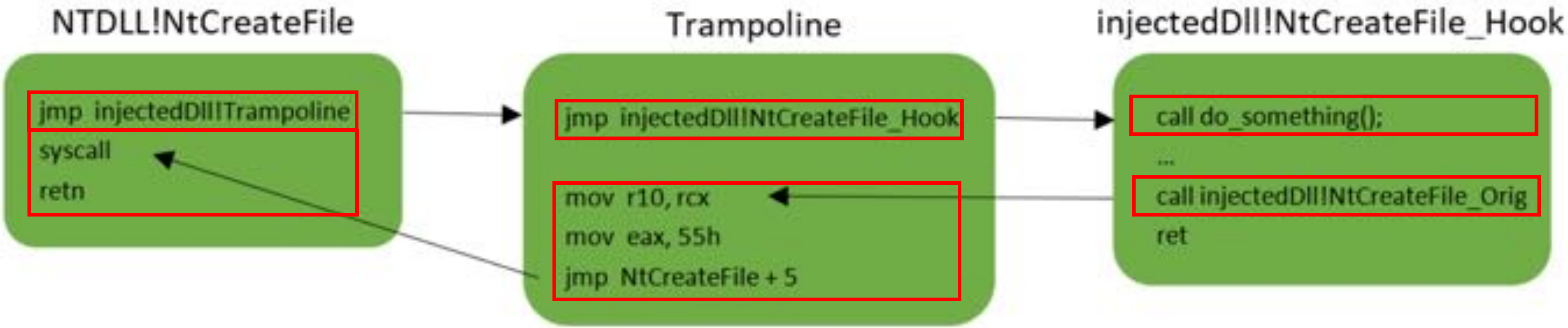


explorer.exe	0.17	102,388 K	142,596 K	3508	64-bit CFG
MSASCuiL.exe		5,080 K	14,592 K	6484	64-bit CFG
vmtoolsd.exe	0.04	23,000 K	44,436 K	6576	64-bit
SentinelUI.exe		26,708 K	50,240 K	6596	64-bit
powershell.exe	0.01	60,380 K	82,908 K	6660	64-bit n/a
conhost.exe		6,888 K	21,120 K	6716	64-bit n/a
regedit.exe		4,456 K	20,284 K	6588	64-bit n/a
notepad.exe		8,312 K	19,024 K	3820	32-bit CFG
procexp.exe		7,312 K	15,928 K	5020	32-bit
procexp64.exe	3.06	16,592 K	42,708 K	5640	64-bit
OneDrive.exe		16,676 K	43,068 K	5096	32-bit CFG

Name	Base	Image Type
InjectedDll.dll	0x6FBA0000	64-bit
wow64cpu.dll	0x77C60000	64-bit
wow64win.dll	0x77C70000	64-bit
wow64.dll	0x77CF0000	64-bit
ntdll.dll	0x7FFB8DE30000	64-bit
notepad.exe	0x2D0000	32-bit
ntmarta.dll	0x6C910000	32-bit
mclient.dll	0x6CB70000	32-bit

INLINE HOOKS 101

- We want our DLL to hook the 64-bit NTDLL
- Most hooking engines use “inline hooks”



NTDLL!NtCreateFile

```
jmp injectedDll!Trampoline
syscall
retn
```

Trampoline

```
jmp injectedDll!NtCreateFile_Hook

mov r10, rcx
mov eax, 55h
jmp NtCreateFile + 5
```

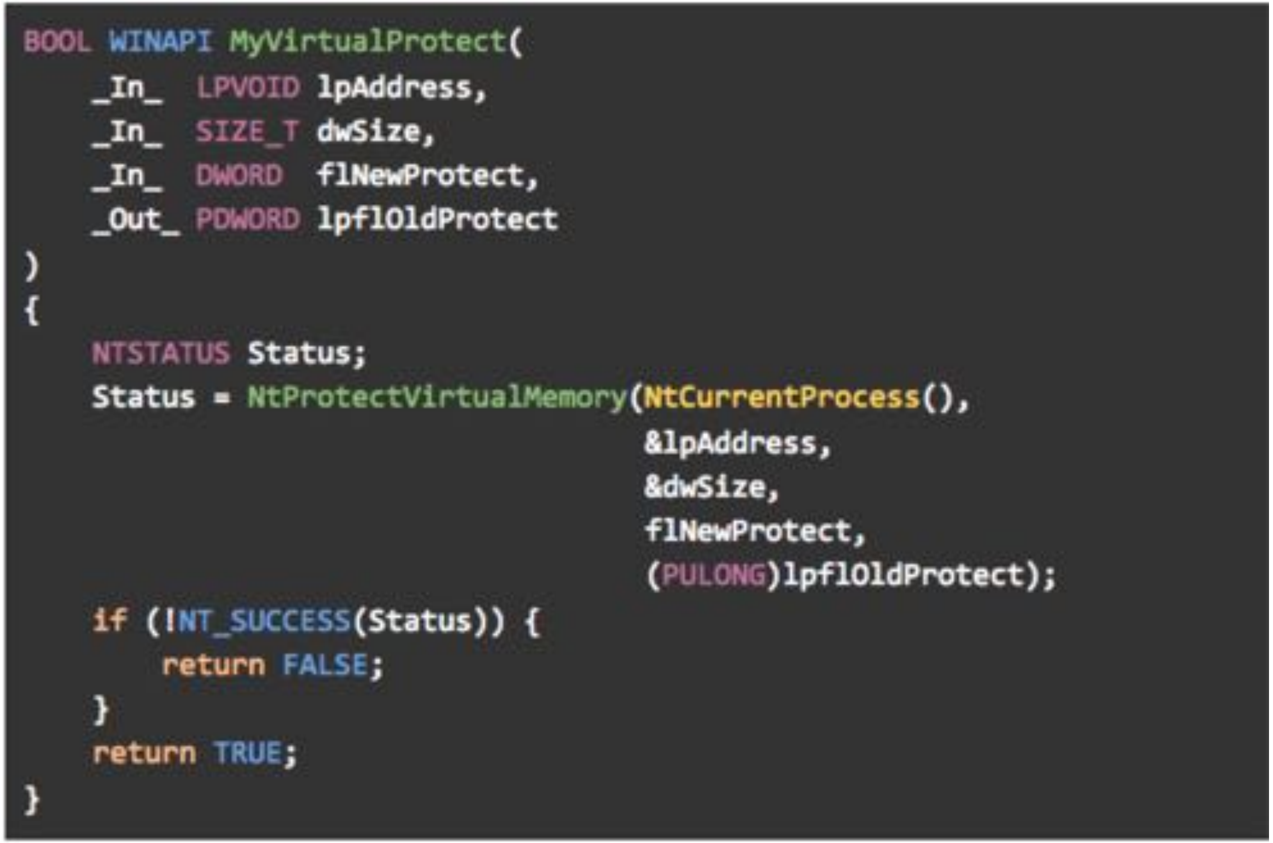
injectedDll!NtCreateFile_Hook

```
call do_something();
...
call injectedDll!NtCreateFile_Orig
ret
```

CONSTRAINTS

- No hooking engine can hook 64-bit APIs in WoW64 apps
- Major limitation - no core Win32 DLLs
  - Kernelbase.dll
  - Kernel32.dll
  - user32.dll
  - msvcrt.dll
- Strip all dependencies other than 64-bit NTDLL
  - Re-implement WIN32 APIs
  - Disable some security and runtime checks
  - Replace some functions (memset, memcpy) implemented in CRT

API RE-IMPLEMENTATION

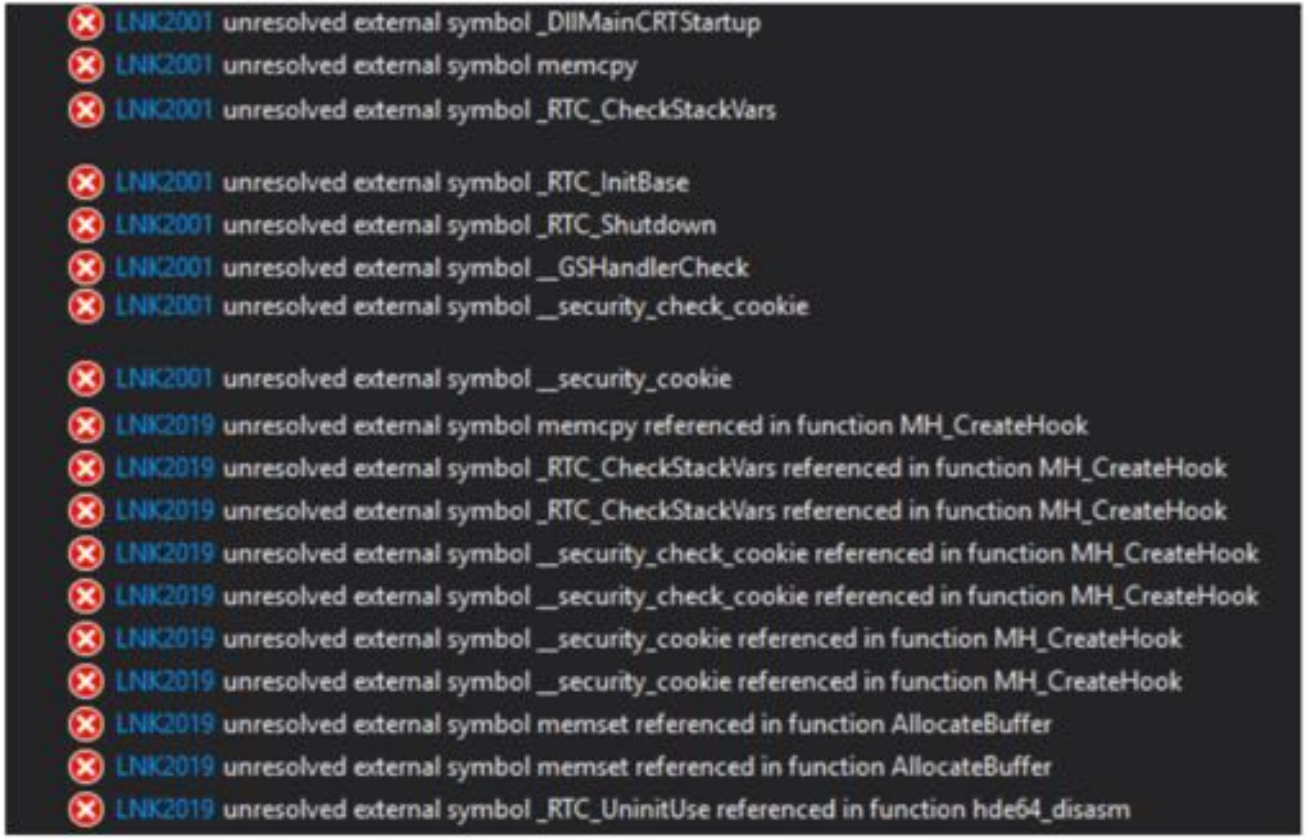


```
BOOL WINAPI MyVirtualProtect(
    _In_  LPVOID lpAddress,
    _In_  SIZE_T dwSize,
    _In_  DWORD  flNewProtect,
    _Out_ PDWORD lpflOldProtect
)
{
    NTSTATUS Status;
    Status = NtProtectVirtualMemory(NtCurrentProcess(),
                                    &lpAddress,
                                    &dwSize,
                                    flNewProtect,
                                    (PULONG)lpflOldProtect);
    if (!NT_SUCCESS(Status)) {
        return FALSE;
    }
    return TRUE;
}
```

ReactOS

ReactOS to the rescue!

SOLVING SOME MORE ERRORS

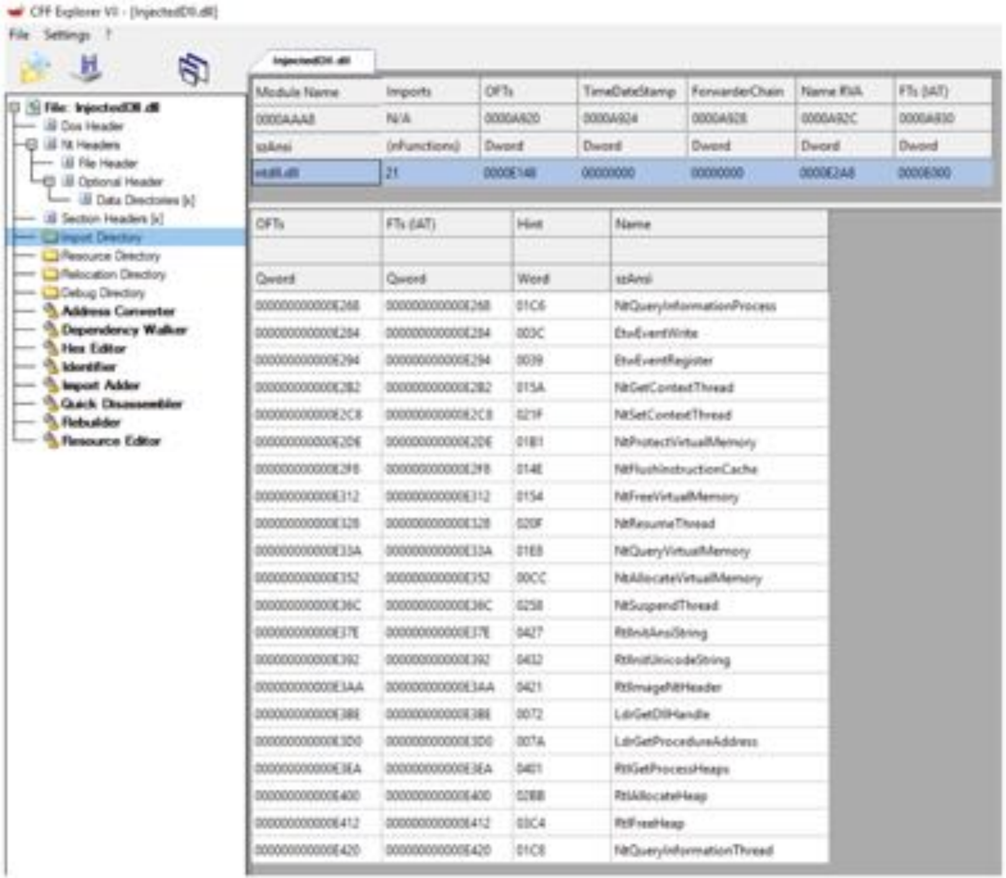


```
LNK2001 unresolved external symbol _DllMainCRTStartup
LNK2001 unresolved external symbol memcpy
LNK2001 unresolved external symbol _RTC_CheckStackVars

LNK2001 unresolved external symbol _RTC_InitBase
LNK2001 unresolved external symbol _RTC_Shutdown
LNK2001 unresolved external symbol __GSHandlerCheck
LNK2001 unresolved external symbol __security_check_cookie

LNK2001 unresolved external symbol __security_cookie
LNK2019 unresolved external symbol memcpy referenced in function MH_CreateHook
LNK2019 unresolved external symbol _RTC_CheckStackVars referenced in function MH_CreateHook
LNK2019 unresolved external symbol _RTC_CheckStackVars referenced in function MH_CreateHook
LNK2019 unresolved external symbol __security_check_cookie referenced in function MH_CreateHook
LNK2019 unresolved external symbol __security_check_cookie referenced in function MH_CreateHook
LNK2019 unresolved external symbol __security_cookie referenced in function MH_CreateHook
LNK2019 unresolved external symbol __security_cookie referenced in function MH_CreateHook
LNK2019 unresolved external symbol memset referenced in function AllocateBuffer
LNK2019 unresolved external symbol memset referenced in function AllocateBuffer
LNK2019 unresolved external symbol _RTC_UninitUse referenced in function hde64_disasm
```

These mostly require configuration changes.

They are not at all interesting so we will not talk about them.

SUCCESS!

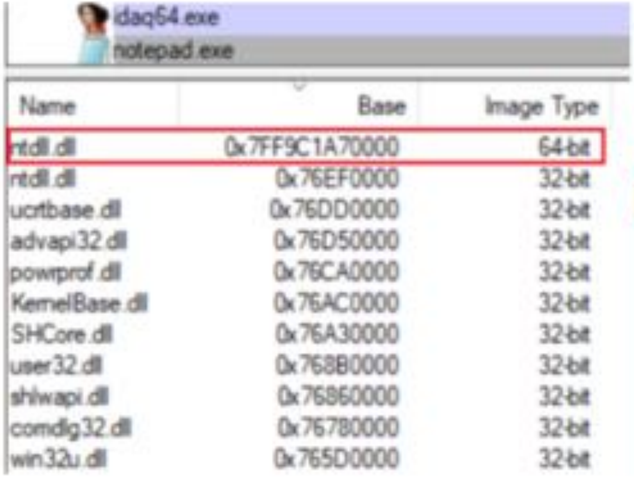


Module Name	Imports	OFTs	TimeDateStamp	ForwarderChain	Name RVA	FTs (IAT)
0000AAA8	N/A	0000A920	0000A924	0000A928	0000A92C	0000A930
szAnsi	(nFunctions)	Dword	Dword	Dword	Dword	Dword
ntdll.dll	21	0000E148	00000000	00000000	0000E2A8	0000E000

OFTs	FTs (IAT)	Hint	Name
Qword	Qword	Word	szAnsi
000000000000E268	000000000000E268	01C6	NtQueryInformationProcess
000000000000E284	000000000000E284	003C	EtwEventWrite
000000000000E294	000000000000E294	0039	EtwEventRegister
000000000000E2B2	000000000000E2B2	015A	NtGetContextThread
000000000000E2C8	000000000000E2C8	021F	NtSetContextThread
000000000000E2DE	000000000000E2DE	01B1	NtProtectVirtualMemory
000000000000E2F8	000000000000E2F8	014E	NtFlushInstructionCache
000000000000E312	000000000000E312	0154	NtFreeVirtualMemory
000000000000E328	000000000000E328	020F	NtResumeThread
000000000000E33A	000000000000E33A	01E8	NtQueryVirtualMemory
000000000000E352	000000000000E352	00CC	NtAllocateVirtualMemory
000000000000E36C	000000000000E36C	0258	NtSuspendThread
000000000000E37E	000000000000E37E	0427	RtlInitAnsiString
000000000000E392	000000000000E392	0432	RtlInitUnicodeString
000000000000E3AA	000000000000E3AA	0421	RtlImageNtHeader
000000000000E3BE	000000000000E3BE	0072	LdrGetDllHandle
000000000000E3D0	000000000000E3D0	007A	LdrGetProcedureAddress
000000000000E3EA	000000000000E3EA	0401	RtlGetProcessHeaps
000000000000E400	000000000000E400	02BB	RtlAllocateHeap
000000000000E412	000000000000E412	03C4	RtlFreeHeap
000000000000E420	000000000000E420	01C8	NtQueryInformationThread

MAYBE NOT?

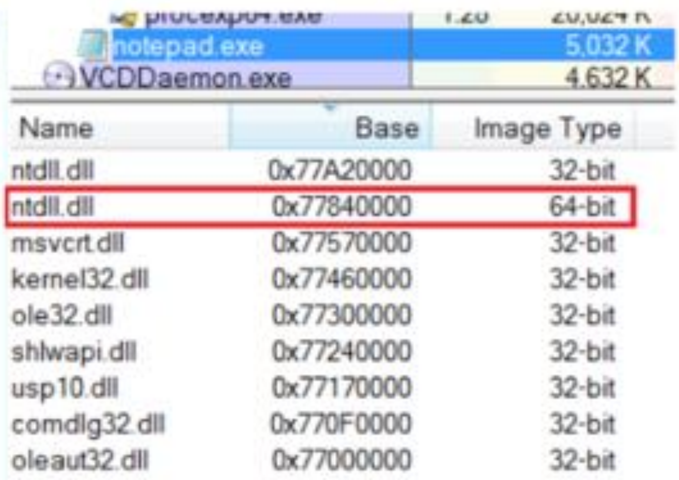
- Until Windows 8 all the modules in WoW64 processes resided below 4GB
- In Windows 8.1 the 64-bit NTDLL was moved above 4GB



idaq64.exe
notepad.exe

Name	Base	Image Type
ntdll.dll	0x7FF9C1A70000	64-bit
ntdll.dll	0x76EF0000	32-bit
ucrtbase.dll	0x76DD0000	32-bit
advapi32.dll	0x76D50000	32-bit
powrprof.dll	0x76CA0000	32-bit
KernelBase.dll	0x76AC0000	32-bit
SHCore.dll	0x76A30000	32-bit
user32.dll	0x768B0000	32-bit
shlwapi.dll	0x76860000	32-bit
comdlg32.dll	0x76780000	32-bit
win32u.dll	0x765D0000	32-bit

Windows 10



procexp64.exe	1.20	20,024 K
notepad.exe		5,032 K
VCDDaemon.exe		4,632 K

Name	Base	Image Type
ntdll.dll	0x77A20000	32-bit
ntdll.dll	0x77840000	64-bit
msvcrt.dll	0x77570000	32-bit
kernel32.dll	0x77460000	32-bit
ole32.dll	0x77300000	32-bit
shlwapi.dll	0x77240000	32-bit
usp10.dll	0x77170000	32-bit
comdlg32.dll	0x770F0000	32-bit
oleaut32.dll	0x77000000	32-bit

Windows 7

BACK TO THE DRAWING BOARD #1

- The JMP used by inline hooks (0xE9) only allows jumping 2GB or less into the trampoline
- No other code can be allocated above 4GB
- Distance between the trampoline and the hooked function is much greater than 2GB => STATUS_EPIC_FAILURE

BACK TO THE DRAWING BOARD #1

1

```
JMP $+      23344
```

2

```
MOV rax, qword p      000]
JMP rax
```

3

```
MOV rax,0x11223      788
JMP rax
```

4

```
PUSH   0x55667788
MOV    dword ptr [rsp+4],0x11223344
RET
```

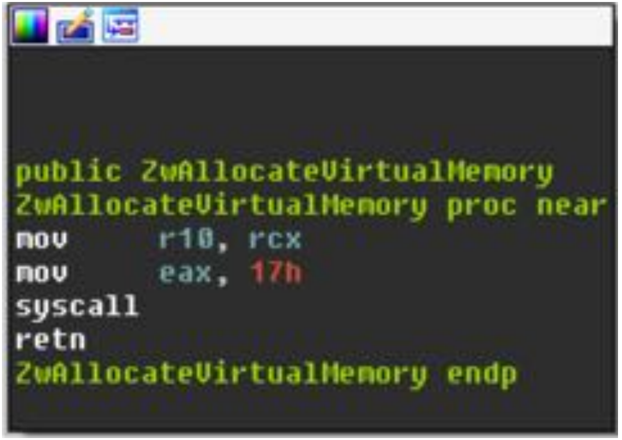
5

```
JMP qword ptr [rip+0]
0x1122334455667788
```

http://www.ragestorm.net/blogs/?p=107

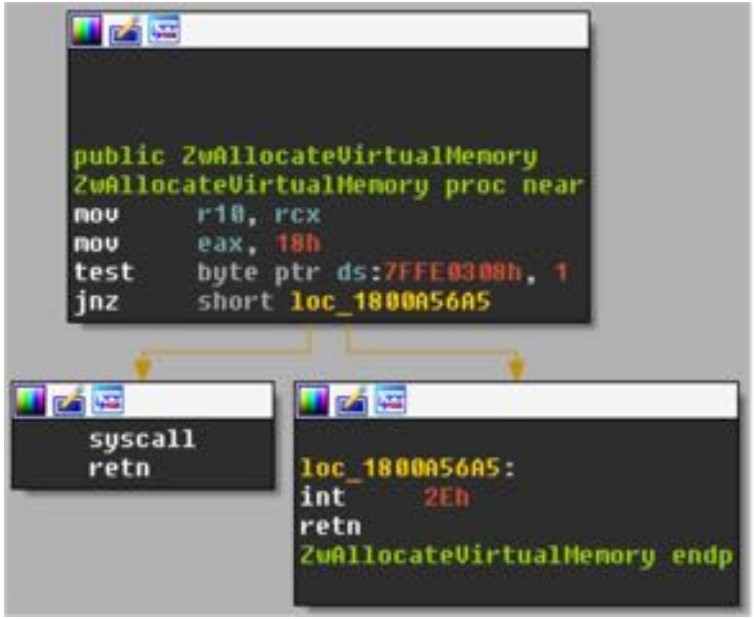
WORKS ON WINDOWS 10 (BUT ONLY THERE…)

- Instruction is too long for environments older than Windows 10



```
public ZwAllocateVirtualMemory
ZwAllocateVirtualMemory proc near
mov     r10, rcx
mov     eax, 17h
syscall
retn
ZwAllocateVirtualMemory endp
```

Windows 7



```
public ZwAllocateVirtualMemory
ZwAllocateVirtualMemory proc near
mov     r10, rcx
mov     eax, 18h
test    byte ptr ds:7FFE0308h, 1
jnz     short loc_1800A56A5
```

```
        syscall
        retn
```

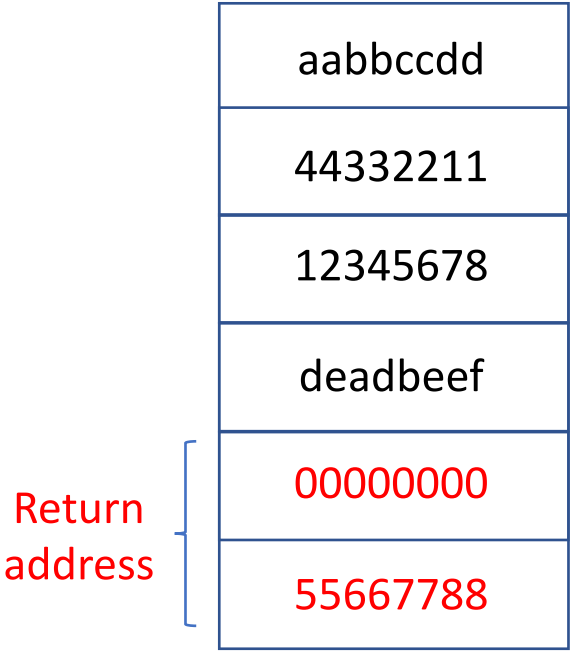
```
loc_1800A56A5:
int     2Eh
retn
ZwAllocateVirtualMemory endp
```

Windows 10

BACK TO THE DRAWING BOARD #2

```
PUSH   0x55667788
~~MOV    dword ptr [rsp+4],0x11223344~~
RET
```

- Memory can only be allocated in lower 4GB
  - Trampoline will be in lower 4GB
- 64-bit addresses under 4GB – 0x00000000xxxxxxxx
- Push imm32 zero-extends the value to 64-bits



aabbccdd

44332211

12345678

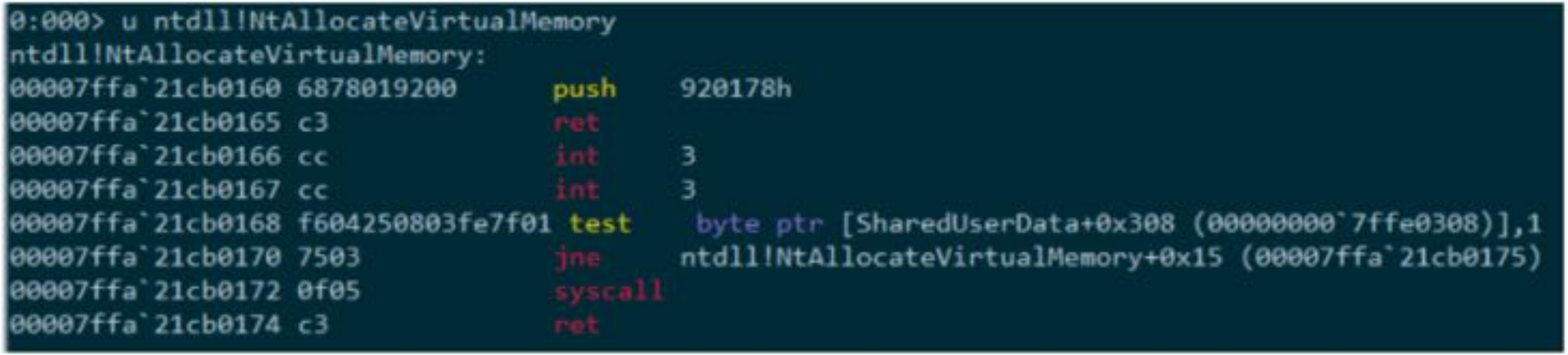
deadbeef

00000000

55667788

Return address

SUCCESS!



```
0:000> u ntdll!NtAllocateVirtualMemory
ntdll!NtAllocateVirtualMemory:
00007ffa`21cb0160 6878019200       push    920178h
00007ffa`21cb0165 c3               ret
00007ffa`21cb0166 cc               int     3
00007ffa`21cb0167 cc               int     3
00007ffa`21cb0168 f604250803fe7f01 test    byte ptr [SharedUserData+0x308 (00000000`7ffe0308)],1
00007ffa`21cb0170 7503             jne     ntdll!NtAllocateVirtualMemory+0x15 (00007ffa`21cb0175)
00007ffa`21cb0172 0f05             syscall
00007ffa`21cb0174 c3               ret
```

DEEP HOOKS - RECAP

- Injection of 64-bit DLL to WoW64 process:
  - 3 (relatively) known injection methods
  - 2 (new) variations to APC injection
- Modified hooking engine
  - Re-implemented Win32 APIs
  - Project configuration changes
  - Replaced the JMP instruction from the hooked function to the detour

REFERENCES

- https://www.sentinelone.com/blog/deep-hooks-monitoring-native-execution-wow64-applications-part-1/
- https://www.sentinelone.com/blog/deep-hooks-monitoring-native-execution-wow64-applications-part-2/
- https://www.sentinelone.com/blog/deep-hooks-monitoring-native-execution-wow64-applications-part-3/
- https://github.com/Sentinel-One/minhook

QUESTIONS?