FUNCTION REROUTING FROM KERNEL LAND

WITH

"HADES"

Jason Raber
Director of Cyber Research Lab
Riverside Research

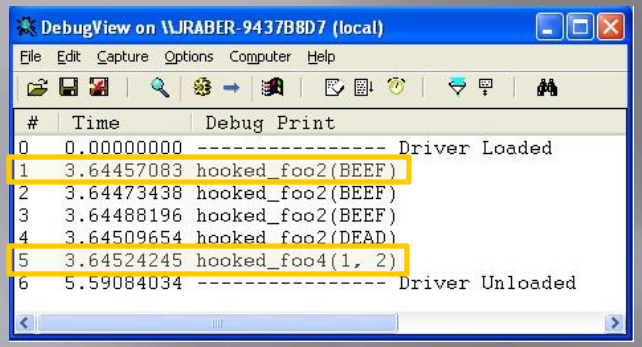
Overview

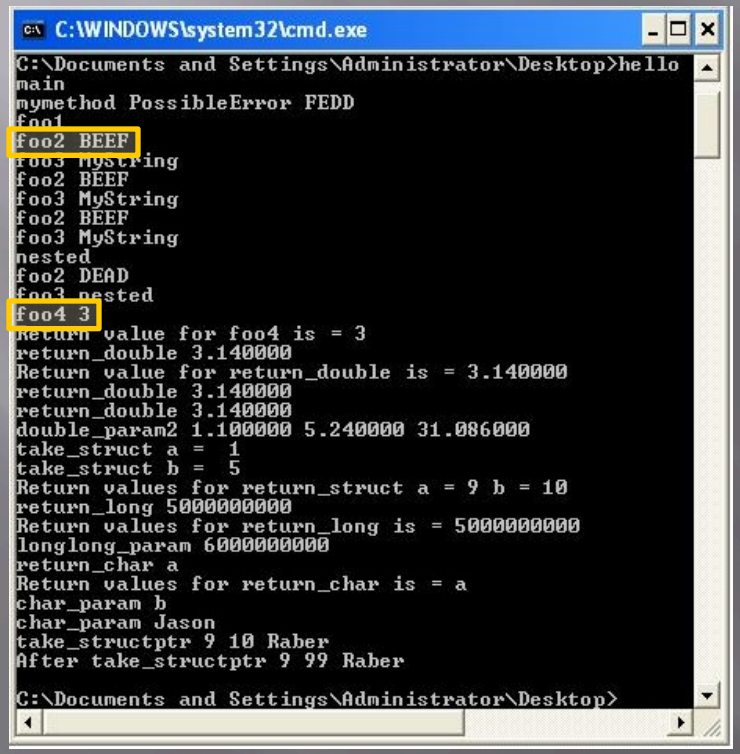
- Why it's cool
- Output
- How it works
- Poor man's tutorial
- It slices, it dices, …
- Where to get it

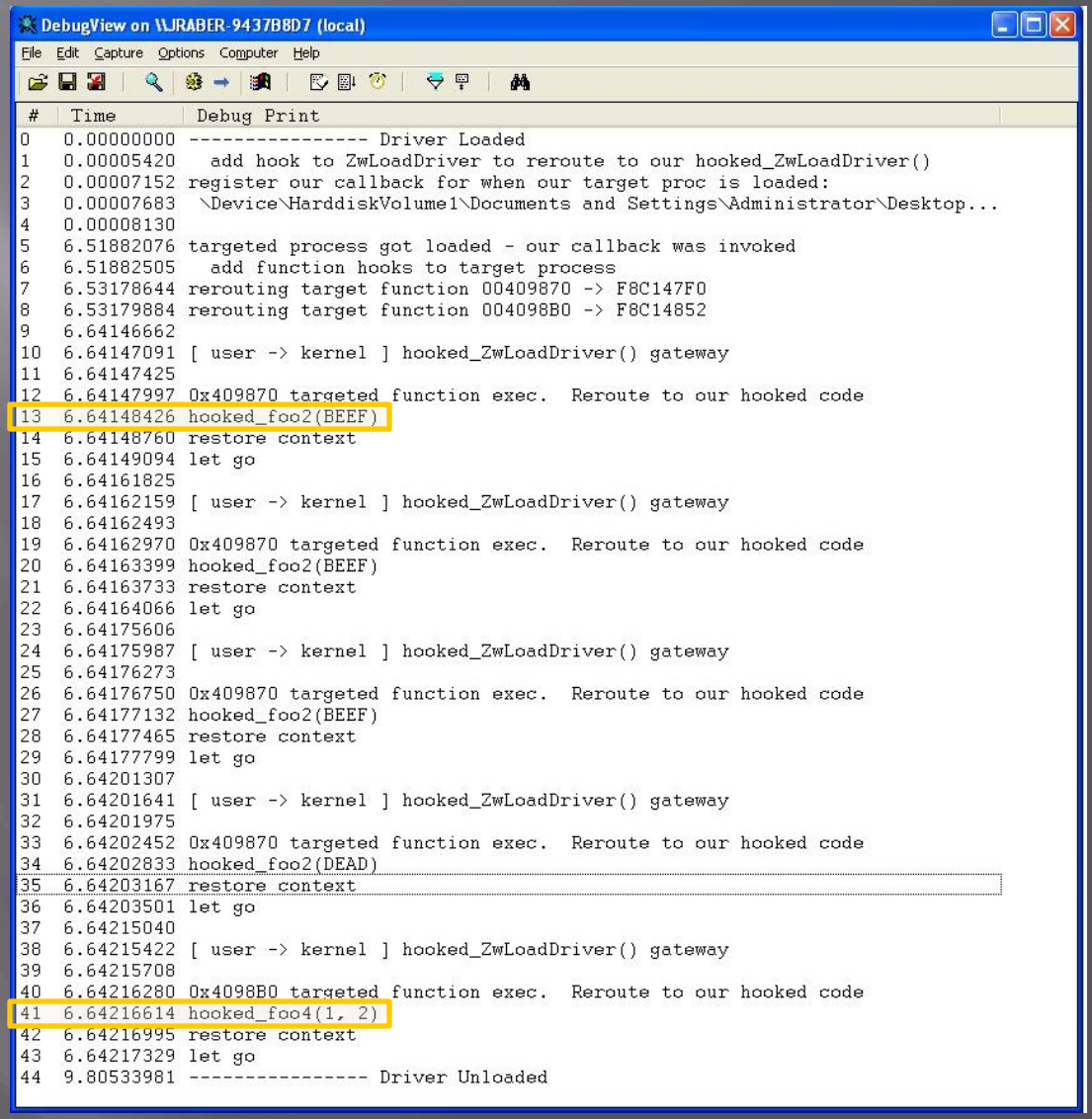
Why it's cool

- "Some folks call it Hell, I call it Hades."
- Hooks both:
  - DLL APIs
  - Internal functions
- Motivation:
  - Detours, WinAPIOverride without the weight
- Gets around a lot of anti-debugging tricks
- Free source code

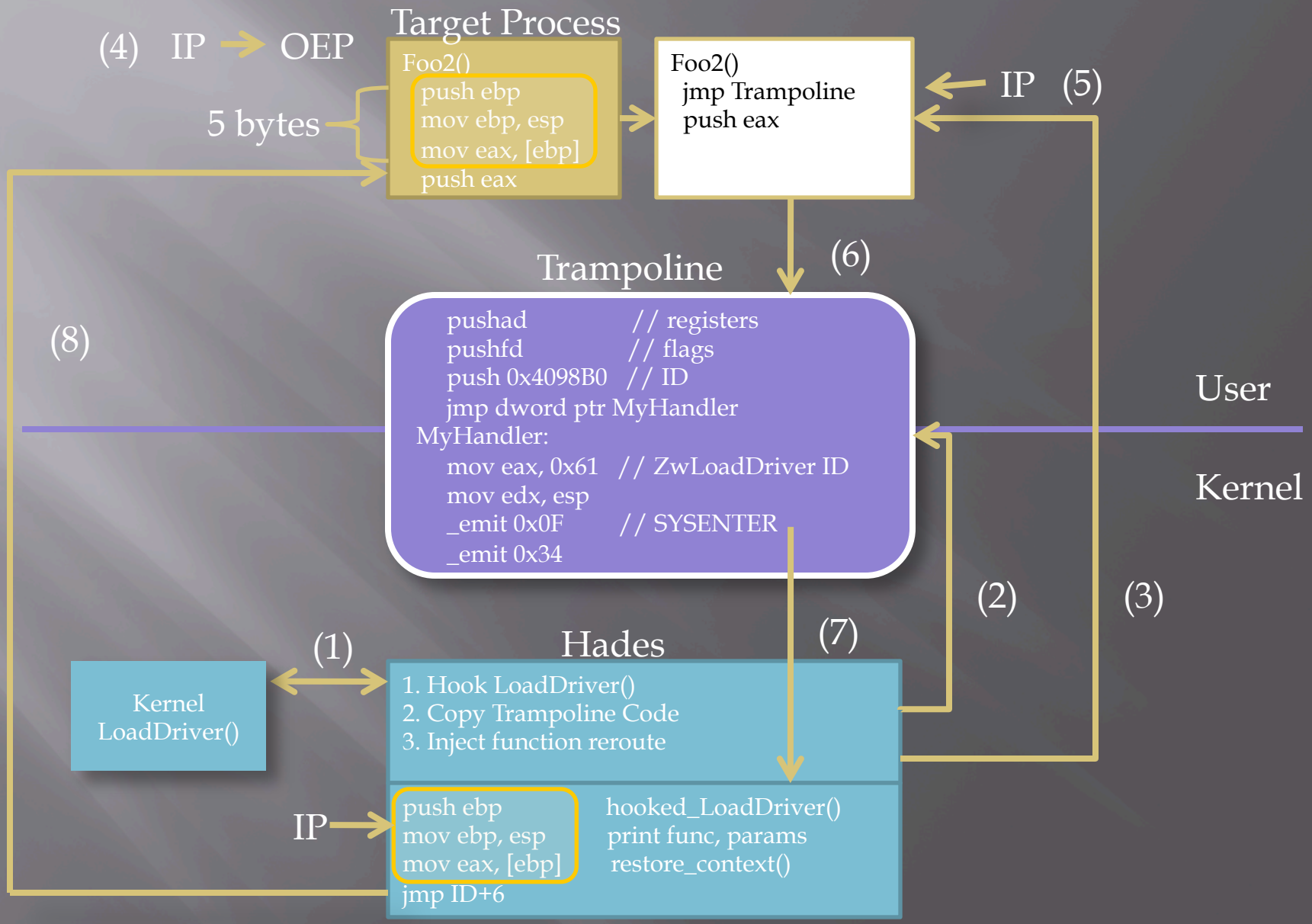
Output


DebugView on \\JRABER-9437B8D7 (local)
File Edit Capture Options Computer Help
Time Debug Print
0 0.00000000 ---------- Driver Loaded
1 3.64457083 hooked_foo2(BEEF)
2 3.64473438 hooked_foo2(BEEF)
3 3.64488196 hooked_foo2(BEEF)
4 3.64509654 hooked_foo2(DEAD)
5 3.64524245 hooked_foo4(1, 2)
6 5.59084034 ---------- Driver Unloaded


C:\WINDOWS\system32\cmd.exe
C:\Documents and Settings\Administrator\Desktop>hello
main
mymethod PossibleError FEDD
foo1
foo2 BEEF
foo3 MyString
foo2 BEEF
foo3 MyString
foo2 BEEF
foo3 MyString
nested
foo2 DEAD
foo3 nested
foo4 3
Return value for foo4 is = 3
return_double 3.140000
Return value for return_double is = 3.140000
return_double 3.140000
return_double 3.140000
double_param2 1.100000 5.240000 31.086000
take_struct a = 1
take_struct b = 5
Return values for return_struct a = 9 b = 10
return_long 5000000000
Return values for return_long is = 5000000000
longlong_param 6000000000
return_char a
Return values for return_char is = a
char_param b
char_param Jason
take_structptr 9 10 Raber
After take_structptr 9 99 Raber
C:\Documents and Settings\Administrator\Desktop>

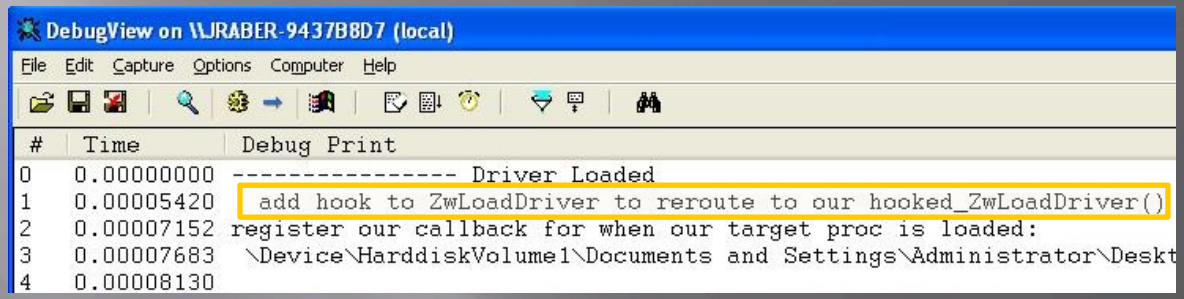

DebugView on \\JRABER-9437B8D7 (local)
File Edit Capture Options Computer Help
Time Debug Print
0 0.00000000 ---------- Driver Loaded
1 0.00005420 add hook to ZwLoadDriver to reroute to our hooked_ZwLoadDriver()
2 0.00007152 register our callback for when our target proc is loaded:
3 0.00007683 \Device\HarddiskVolume1\Documents and Settings\Administrator\Desktop...
4 0.00008130
5 6.51882076 targeted process got loaded - our callback was invoked
6 6.51882505 add function hooks to target process
7 6.53178644 rerouting target function 00409870 -> F8C147F0
8 6.53179884 rerouting target function 004098B0 -> F8C14852
9 6.64146662
10 6.64147091 [user -> kernel] hooked_ZwLoadDriver() gateway
11 6.64147425
12 6.64147997 0x409870 targeted function exec. Reroute to our hooked code
13 6.64148426 hooked_foo2(BEEF)
14 6.64148760 restore context
15 6.64149094 let go
16 6.64161825
17 6.64162159 [user -> kernel] hooked_ZwLoadDriver() gateway
18 6.64162493
19 6.64162970 0x409870 targeted function exec. Reroute to our hooked code
20 6.64163399 hooked_foo2(BEEF)
21 6.64163733 restore context
22 6.64164066 let go
23 6.64175606
24 6.64175987 [user -> kernel] hooked_ZwLoadDriver() gateway
25 6.64176273
26 6.64176750 0x409870 targeted function exec. Reroute to our hooked code
27 6.64177132 hooked_foo2(BEEF)
28 6.64177465 restore context
29 6.64177799 let go
30 6.64201307
31 6.64201641 [user -> kernel] hooked_ZwLoadDriver() gateway
32 6.64201975
33 6.64202452 0x409870 targeted function exec. Reroute to our hooked code
34 6.64202833 hooked_foo2(DEAD)
35 6.64203167 restore context
36 6.64203501 let go
37 6.64215040
38 6.64215422 [user -> kernel] hooked_ZwLoadDriver() gateway
39 6.64215708
40 6.64216280 0x4098B0 targeted function exec. Reroute to our hooked code
41 6.64216614 hooked_foo4(1, 2)
42 6.64216995 restore context
43 6.64217329 let go
44 9.80533981 ---------- Driver Unloaded

How it works


(4) IP → OEP
Target Process
Foo2()
push ebp
mov ebp, esp
mov eax, [ebp]
push eax
5 bytes
Foo2()
jmp Trampoline
push eax
IP (5)
Trampoline
(6)
(8)
pushad // registers
pushfd // flags
push 0x4098B0 // ID
jmp dword ptr MyHandler
MyHandler:
mov eax, 0x61 // ZwLoadDriver ID
mov edx, esp
_emit 0x0F // SYSENTER
_emit 0x34
User
Kernel
(2)
(3)
(1)
Hades
(7)
Kernel
LoadDriver()
1. Hook LoadDriver()
2. Copy Trampoline Code
3. Inject function reroute
IP
push ebp
mov ebp, esp
mov eax, [ebp]
jmp ID+6
hooked_LoadDriver()
print func, params
restore_context()

Poor man's tutorial

DriverEntry()




```
    hook_syscalls();

    debug("register our callback for when our target proc is loaded:\n %ws\n\n",
            target_file_loc);

#if BREAK_POINT
    // register a callback func that is invoked when our target proc is loaded
    ret = PsSetLoadImageNotifyRoutine(add_one_time_bp);
#endif

#if DATA_MINING
    ret = PsSetLoadImageNotifyRoutine(add_hooks_for_data_mining);
#endif
```

Hook system call

Register callback

```
//------------------------------------------------------------------------
// Hook the system calls to allow us to pass control from user to kernel...
// LoadDriver system call hook is our gateway
//------------------------------------------------------------------------
VOID hook_syscalls()
{
    debug("\t add hook to ZwLoadDriver to reroute to our " \
          "hooked_ZwLoadDriver() \n");

    orig_ZwLoadDriver =
        (void *)InterlockedExchange(
        (unsigned int *) &syscall_tbl[SYSCALL_INDEX(ZwLoadDriver)],
        (unsigned int) hooked_ZwLoadDriver);
}
```

Poor man's tutorial

```
5    6.51882076 targeted process got loaded - our callback was invoked
6    6.51882505   add function hooks to target process
7    6.53178644 rerouting target function 00409870 -> F8C147F0
8    6.53179884 rerouting target function 004098B0 -> F8C14852
```

add_hooks_for_data_mining()

```
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//
// ADD HOOKED CODE HERE
//
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
target_foo2 = (int (__cdecl *)(int))0x409870;
reroute_function(target_foo2, hooked_foo2);

target_foo4 = (int (__cdecl *)(int, int))0x4098B0;
reroute_function(target_foo4, hooked_foo4);

// copy shared memory function to shared user space memory
CLEAR_WP_FLAG;
RtlCopyMemory((PVOID)shared_kern_mem, shared_mem_data_mining,
              SIZE_OF_SHARED_MEM);
RESTORE_CR0;
```

reroute_function()

```
// Dest - CurrentAddr - SizeJump
//
// NOTE: Why offset?  Look at function shared_mem_data_mining()...
// There are 0xC bytes offset for each hooked function
// that the user process needs to jump to in shared memory space.  Just
// make sure that the order of the hooked functions is important
if (offset != 0)
{
    jmp_shared = (shared_user_mem + offset) -
                 (unsigned int)orig_func - SIZE_OF_JMP;
}
else
{
    jmp_shared = shared_user_mem - (unsigned int)orig_func - SIZE_OF_JMP;
}

offset += TRAMPOLINE_OFFSET;

jmp_op[0] = 0xE9;
memcpy(jmp_op+1, &jmp_shared, SIZE_OF_JMP);

// inject jmp into user space (reroute instruction pointer)
CLEAR_WP_FLAG;
RtlCopyMemory(orig_func, jmp_op, SIZE_OF_JMP);
RESTORE_CR0;

// save off the hooked function addresses
if (idx < MAX_ARRAY_HOOKED_CALLS)
{
    array_hooked_calls[0][idx] = (unsigned int)orig_func;
    array_hooked_calls[1][idx] = (unsigned int)hooked_func;
    idx++;
}
```

Poor man's tutorial

```
.0   6.64147091 [ user -> kernel ] hooked_ZwLoadDriver() gateway
.1   6.64147425
.2   6.64147997 0x409870 targeted function exec.  Reroute to our hooked code
```

Trampoline

```
    pushad              // registers
    pushfd              // flags
    push 0x4098B0       // ID
    jmp dword ptr MyHandler
MyHandler:
    mov eax, 0x61       // ZwLoadDriver ID
    mov edx, esp
    _emit 0x0F          // SYSENTER
    _emit 0x34
```

hooked_ZwLoadDriver()

```
// look for our identifier - our BP was pushed on the stack from shared_mem
_asm
{
    push eax
    mov eax, edx        // EDX == ESP
    mov gORIG_ESP, eax  // Save off the ESP to be restored from the driver.
                        // however, the alignment is off by 2 DWORDs...
    sub eax, 8
    mov eax, [eax]
    mov gBP, eax
    pop eax
}

debug("\n[ user -> kernel ] hooked_ZwLoadDriver() gateway\n\n");

DATA_MINING
{
    handle_hooked_calls();
```

handle_hooked_calls()

```
//                STACK  (Low)
//               -------
//               | ID  | <- Identifier (User Hooked Function Addr)
//               | EFG | <- EFLAGS
//  gORIG_ESP -> | EDI |
//               | ESI |
//               | EBP |
//               | ESP |
//               | EBX |
//               | EDX |
//               | ECX |
//               | EAX | (High)
//               | RET | <- Return address of the caller (Orig. stack frame befor
//               -------
//--------------------------------------------------------------------------------
void handle_hooked_calls()
{
    unsigned int hooked_call = 0;

    save_context();

    debug("0x%X targeted function exec.  Reroute to our hooked code\n", gID);
    hooked_call = get_hooked_call_addr();
```

Poor man's tutorial

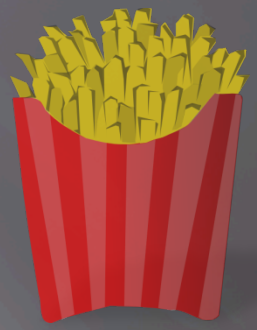
```
6.64148426 hooked_foo2(BEEF)
6.64148760 restore context
6.64149094 let go
6.64161825
6.64162159 [ user -> kernel ] hooked_ZwLoadDriver() gateway
```

```
void __cdecl hooked_foo2(int a)
{
    DbgPrint("hooked_foo2(%X)\n", a);
    debug("restore context\n");
    debug("let go\n");

    restore_context_switch_dm();

    _asm
    {
        // Execute stolen bytes
        _emit 0x55                  //push ebp
        _emit 0x8B                  //mov ebp, esp
        _emit 0xEC
        _emit 0x8B                  //mov eax, dword ptr [ebp+8]
        _emit 0x45
        _emit 0x08

        // Jump to user process
        add gID, 6

        // Restore the eflags
        push gDM_EFLAGS
        popfd

        jmp gID
    }
}
```

```
mov eax, gDM_EAX
mov ebx, gDM_EBX
mov ecx, gDM_ECX
mov edx, gDM_EDX
mov esi, gDM_ESI
mov edi, gDM_EDI
mov ebp, gDM_EBP

mov esp, gDM_ESP
```

It slices, it dices, …

- …it even debugs!
- If we can reroute function calls, why not instructions?
  - Limited to one breakpoint right now
  - One-time use breakpoint
  - Modify away and share ☺



Debugger fun

Make these modifications before adding breakpoint

hades.h

```
// defines
#define DATA_MINING             0
#define BREAK_POINT             1
```

debugger.h

```
#ifndef DEBUGGER_H
#define DEBUGGER_H

#define BP1 0x4098B0
```

Short view

```
---------------- Driver Loaded

   !!! BREAKPOINT HIT @ 4098B0!!!
   EAX = 0x0000000C, EBX = 0x7C80AC51, ECX = 0x004011A3, EDX = 0x00411B70
   ESI = 0x00000002, EDI = 0x00000A28, ESP = 0x0012FEEC, EBP = 0x0012FF70

---------------- Driver Unloaded
```

Verbose view

```
---------------- Driver Loaded
  add hook to ZwLoadDriver to reroute to our hooked_ZwLoadDriver()
register our callback for when our target proc is loaded:
 \Device\HarddiskVolume1\Documents and Settings\Administrator\Desktop\Hel.

targeted process got loaded - our callback was invoked
  add a one time bp to target process
  before memory bp = 55 8B EC 51 8B
    adding bp to va 0x4098B0
    generated bp jump ins to shared mem: E9 4B 6F BD 7F
  stolen bytes = 55 8B EC 51 8B
bp successfully added to user land at 0x4098B0

let go


[ user -> kernel ] hooked_ZwLoadDriver() gateway

  now handle the bp

   !!! BREAKPOINT HIT @ 4098B0!!!
   EAX = 0x0000000C, EBX = 0x7C80AC51, ECX = 0x004011A3, EDX = 0x00411B70
   ESI = 0x00000002, EDI = 0x00000A28, ESP = 0x0012FEEC, EBP = 0x0012FF70

  replace stolen bytes
  before: E9 4B 6F BD 7F
  after: 55 8B EC 51 8B
  return control back to user space at loc 0x4098B0
---------------- Driver Unloaded
```

Debugger fun

Inject the runtime breakpoint

```
//---------------------------------------------------------------------------
// Inject breakpoint into user space
//---------------------------------------------------------------------------
int add_bp(void)
{
    // dest - currentAddr - sizeJump
    unsigned int jmp_mine = (unsigned int)&shared_mem - BP1 - SIZE_OF_JMP;
    unsigned int jmp_shared = shared_user_mem - BP1 - SIZE_OF_JMP;

    // steal memory to be patched later...this will be were the breakpoint
    // will be added - stolen_code is were stored
    RtlCopyMemory(stolen_code, breakpoint, SIZE_OF_JMP);

    if (jmp_mine > 0)
    {
        debug("\t\t\t adding bp to va 0x%X\n", BP1);

        jmp_op[0] = 0xE9;
        memcpy(jmp_op + 1, &jmp_shared, SIZE_OF_JMP);

        debug("\t\t\t generated bp jump ins to shared mem: ");
        print_memory(jmp_op, 5);

        // inject jmp into user space (reroute instruction pointer)
        CLEAR_WP_FLAG;
        RtlCopyMemory(breakpoint, jmp_op, SIZE_OF_JMP);
        RESTORE_CR0;
    }

    // copy shared memory function to shared user space memory
    CLEAR_WP_FLAG;
    RtlCopyMemory((PVOID)shared_kern_mem, shared_mem, SIZE_OF_SHARED_MEM);
    RESTORE_CR0;
```

Breakpoint has been hit: handle_bp()

```
debug("\t now handle the bp\n");

save_context_dbg();

// replace stolen bytes
if (breakpoint)
{
    debug("\t replace stolen bytes\n");
    debug("\t before: ");
    print_memory(breakpoint, 5);

    CLEAR_WP_FLAG;
    RtlCopyMemory(breakpoint, stolen_code, SIZE_OF_JMP);
    RESTORE_CR0;

    debug("\t after: ");
    print_memory(breakpoint, 5);

    // how to change registers
    //modify_register(my_EAX, 1);

    // now jump back right from the kernel to the user sp
    // NOTE:
    // if you would like to change the EIP then instead o
    // EIP to the BP you can change it to were ever you w
    // instruction pointer to go
    debug("\t return control back to user space at loc 0x
    gEIP = gBP;
    return_to_user_app();
}
```

Where to get it

- Available from github
  - https://github.com/jnraber/Hades
- POC: Jason Raber
  - jraber@riversideresearch.org
  - Work: 937-427-7080
  - Cell: 937-848-1143

Questions?