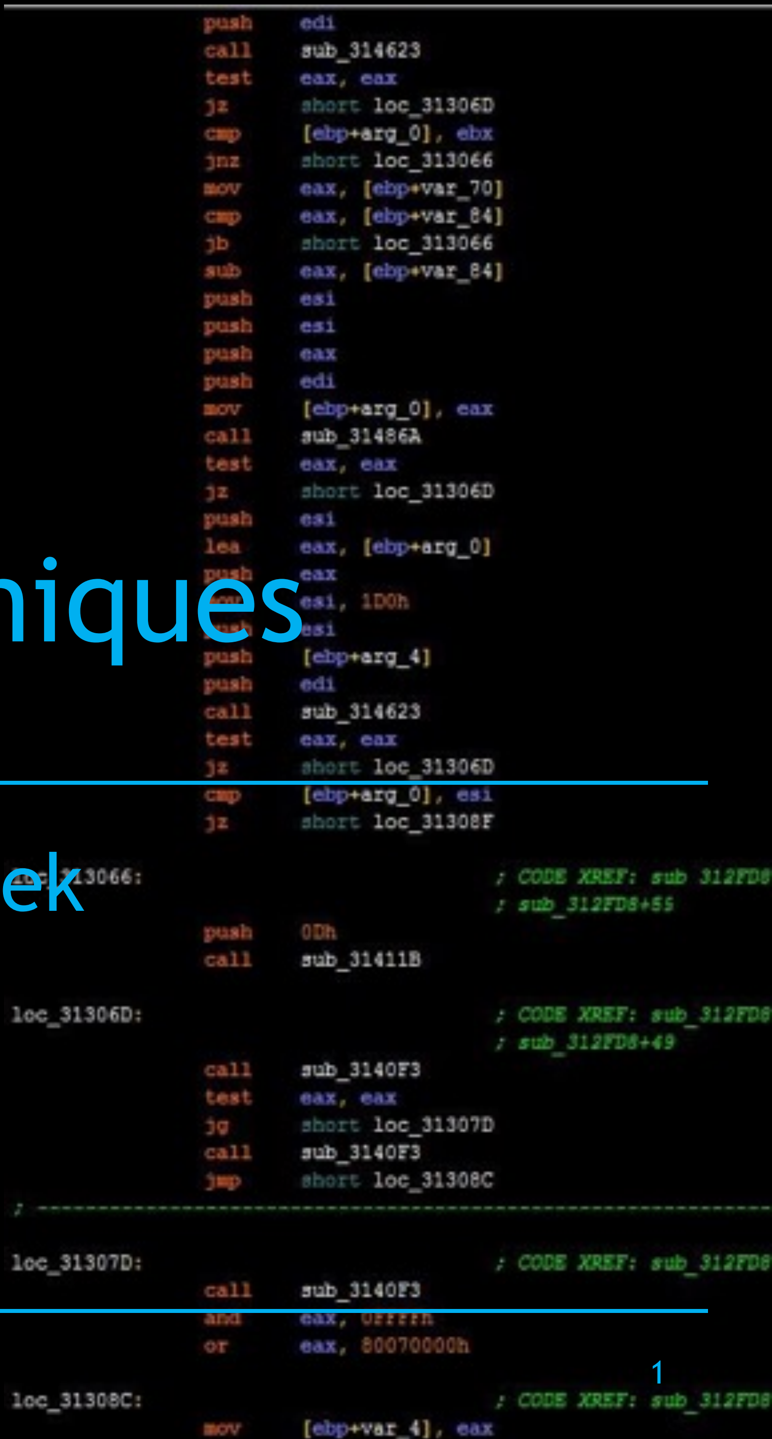
Rootkit Techniques

Aaron Sedlacek



```
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], ebx
                jnz     short loc_313066
                mov     eax, [ebp+var_70]
                cmp     eax, [ebp+var_84]
                jb      short loc_313066
                sub     eax, [ebp+var_84]
                push    esi
                push    esi
                push    eax
                push    edi
                mov     [ebp+arg_0], eax
                call    sub_31486A
                test    eax, eax
                jz      short loc_31306D
                push    esi
                lea     eax, [ebp+arg_0]
                push    eax
                mov     esi, 1D0h
                push    esi
                push    [ebp+arg_4]
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], esi
                jz      short loc_31308F

loc_313066:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+55
                push    0Dh
                call    sub_31411B

loc_31306D:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+49
                call    sub_3140F3
                test    eax, eax
                jg      short loc_31307D
                call    sub_3140F3
                jmp     short loc_31308C
; ---------------------------------------------------------

loc_31307D:                             ; CODE XREF: sub_312FD8
                call    sub_3140F3
                and     eax, 0FFFFh
                or      eax, 80070000h

loc_31308C:                             ; CODE XREF: sub_312FD8
                mov     [ebp+var_4], eax
```

Agenda

- Hooking
- Memory Patching
- Direct Kernel Object Manipulation

```
                      push    edi
                      call    sub_314623
                      test    eax, eax
                      jz      short loc_31306D
                      cmp     [ebp+arg_0], ebx
                      jnz     short loc_313066
                      mov     eax, [ebp+var_70]
                      cmp     eax, [ebp+var_84]
                      jb      short loc_313066
                      sub     eax, [ebp+var_84]
                      push    esi
                      push    esi
                      push    eax
                      push    edi
                      mov     [ebp+arg_0], eax
                      call    sub_31486A
                      test    eax, eax
                      jz      short loc_31306D
                      push    esi
                      lea     eax, [ebp+arg_0]
                      push    eax
                      mov     esi, 1D0h
                      push    esi
                      push    [ebp+arg_4]
                      push    edi
                      call    sub_314623
                      test    eax, eax
                      jz      short loc_31306D
                      cmp     [ebp+arg_0], esi
                      jz      short loc_31308F

loc_313066:                                   ; CODE XREF: sub_312FD8
                                              ; sub_312FD8+55
                      push    0Dh
                      call    sub_31411B

loc_31306D:                                   ; CODE XREF: sub_312FD8
                                              ; sub_312FD8+49
                      call    sub_3140F3
                      test    eax, eax
                      jg      short loc_31307D
                      call    sub_3140F3
                      jmp     short loc_31308C
; ---------------------------------------------------------------------------

loc_31307D:                                   ; CODE XREF: sub_312FD8
                      call    sub_3140F3
                      and     eax, 0FFFFh
                      or      eax, 80070000h

loc_31308C:                                   ; CODE XREF: sub_312FD8
                      mov     [ebp+var_4], eax
```

Kernel Mode Rootkits

- Kernel Mode Rootkits are installed as drivers
- Most rootkits target 32-bit Windows OS's
  - 64-bit Windows architectures require drivers to be signed by Microsoft before they can be installed
    - To subvert this, attackers will:
      - Install a valid, signed driver with a known exploit
      - Use stolen signing certificates
      - Exploit the kernel itself, lol
    - http://www.sekoia.fr/blog/windows-driver-signing-bypass-by-derusbi/

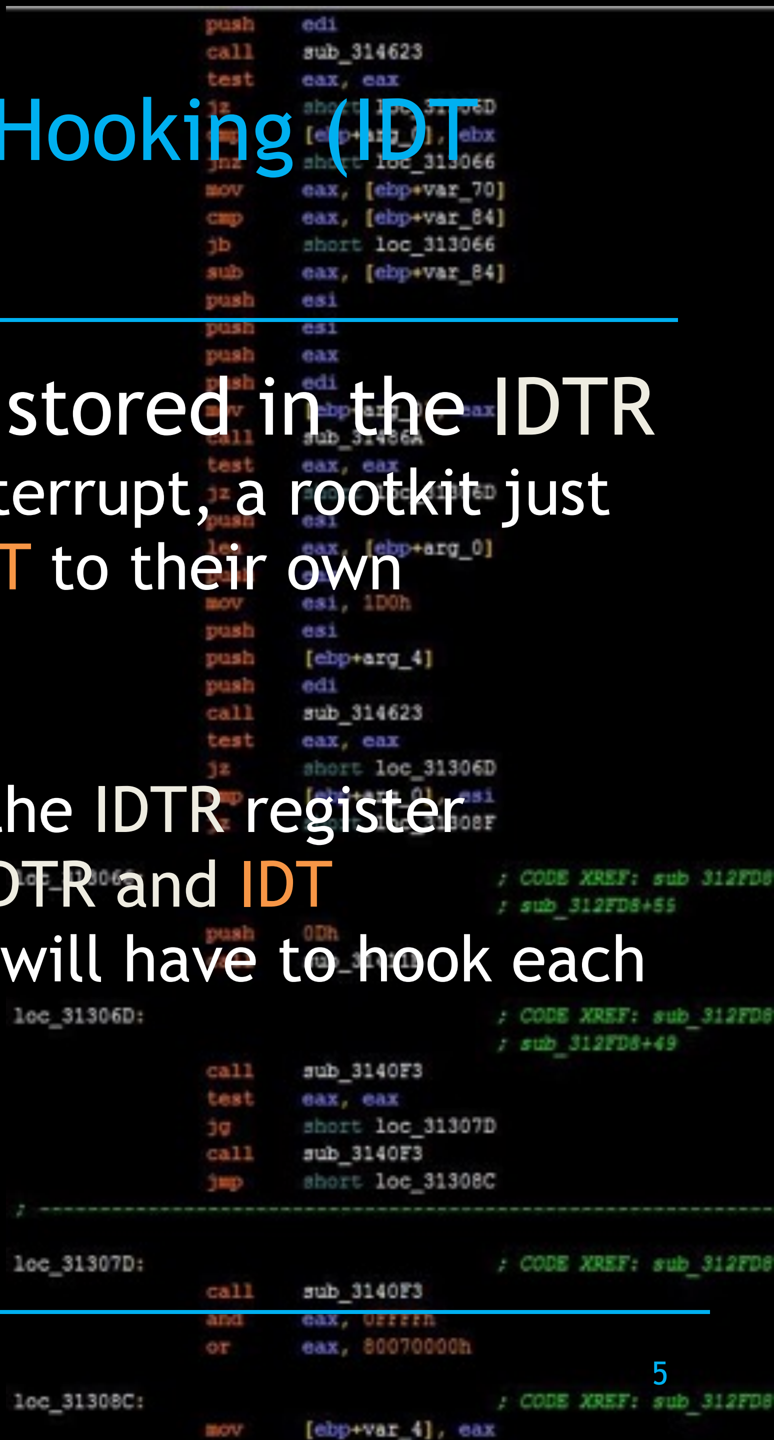
Hooking

- The most classic of all kernel rootkit techniques
- Simple to implement, simple to detect
- Still widely used
  - Often in conjunction with techniques discussed later in this lecture!

```
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], ebx
                jnz     short loc_313066
                mov     eax, [ebp+var_70]
                cmp     eax, [ebp+var_84]
                jb      short loc_313066
                sub     eax, [ebp+var_84]
                push    esi
                push    esi
                push    eax
                push    edi
                [illegible] [illegible]rg_0], eax
                call    sub_31486A
                test    eax, eax
                jz      short loc_31306D
                push    esi
                lea     eax, [ebp+arg_0]
                push    [illegible]
                [illegible]
                push    esi
                push    [ebp+arg_4]
                push    edi
                call    sub_314623
                test    eax, eax
                [illegible] [ebp+arg_0], eax
                jz      short loc_31308F

loc_313066:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+55
                push    0Dh
                call    sub_31411B

loc_31306D:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+49
                call    sub_3140F3
                test    eax, eax
                jg      short loc_31307D
                call    sub_3140F3
                jmp     short loc_31308C
; ---------------------------------------------------------------

loc_31307D:                             ; CODE XREF: sub_312FD8
                call    sub_3140F3
                and     eax, 0FFFFh
                or      eax, 80070000h

loc_31308C:                             ; CODE XREF: sub_312FD8
                mov     [ebp+var_4], eax
```

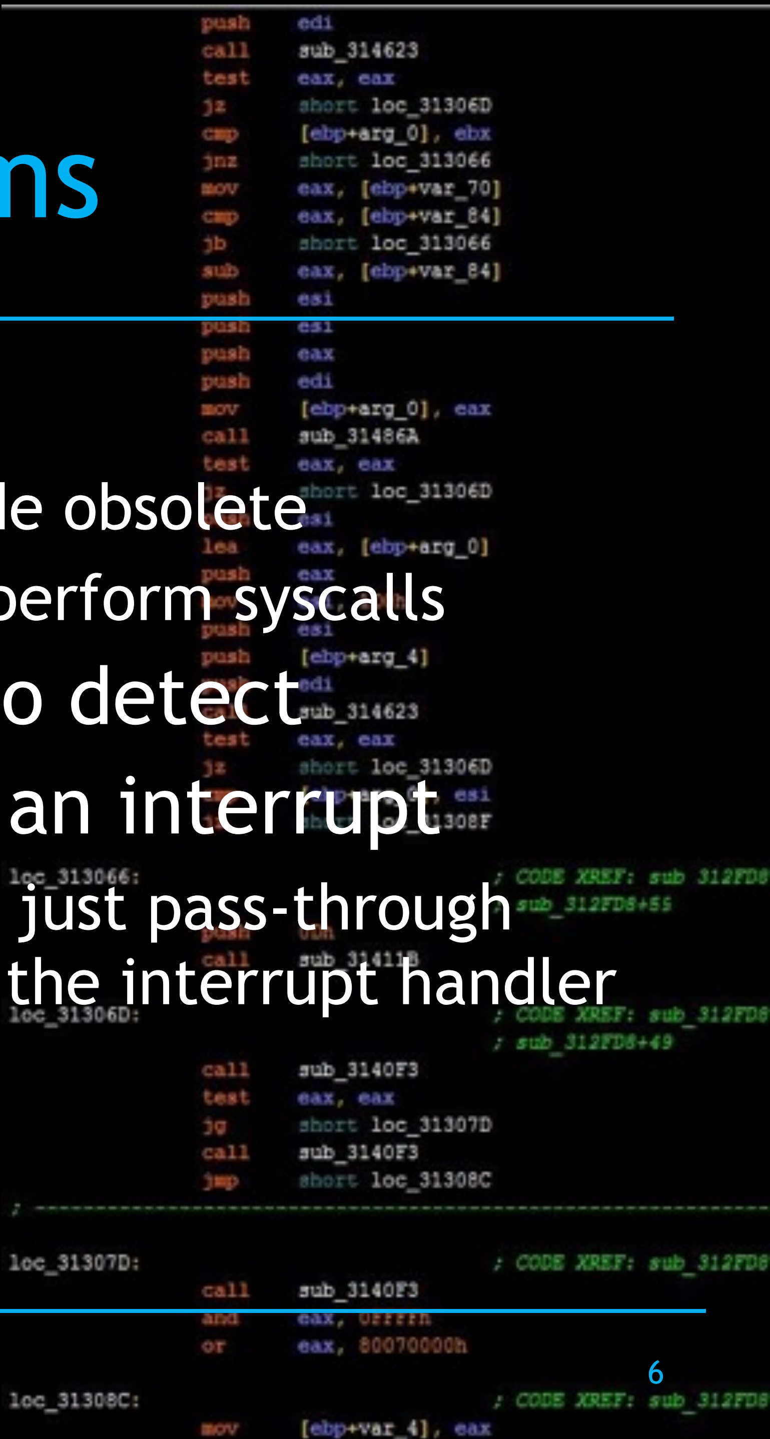
Interrupt Descriptor Table Hooking (IDT Hooking)

- Base address of the IDT is stored in the IDTR
  - In order to hook a specific Interrupt, a rootkit just changes the pointer in the IDT to their own malicious function
- SIDT and LIDT instructions
  - Used to read/write to/from the IDTR register
  - Each processor has it’s own IDTR and IDT
    - This means that a rootkit will have to hook each IDT



IDT Hooking Problems

- This technique is old
  - As of 2009, INT 0x2E was made obsolete
    - SYSENTER is now used to perform syscalls
- Interrupt hooking is easy to detect
- No way to filter results of an interrupt
  - The rootkit’s hook function is just pass-through code that is executed before the interrupt handler



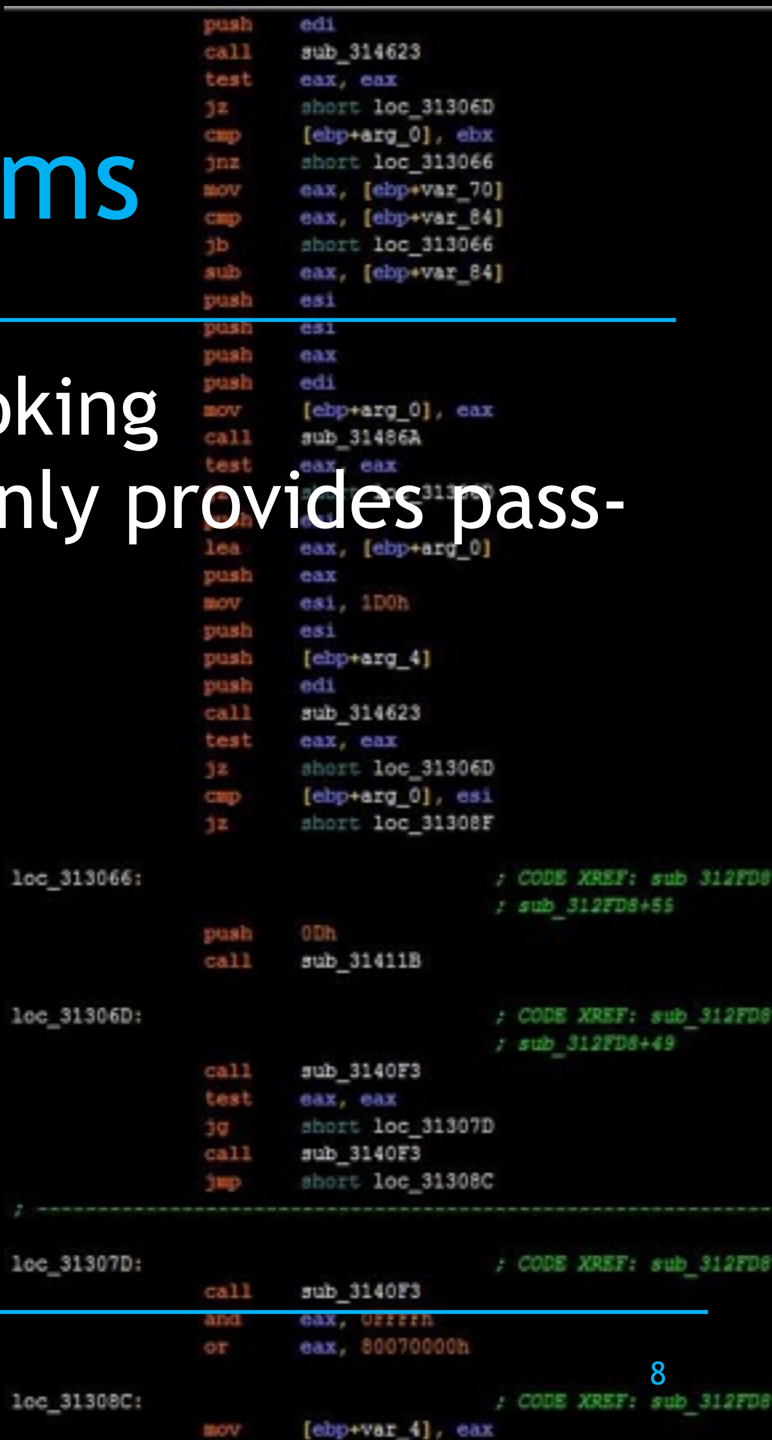
```
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], ebx
                jnz     short loc_313066
                mov     eax, [ebp+var_70]
                cmp     eax, [ebp+var_84]
                jb      short loc_313066
                sub     eax, [ebp+var_84]
                push    esi
                push    esi
                push    eax
                push    edi
                mov     [ebp+arg_0], eax
                call    sub_31486A
                test    eax, eax
                jz      short loc_31306D
                [illegible] esi
                lea     eax, [ebp+arg_0]
                push    eax
                [illegible]
                push    esi
                push    [ebp+arg_4]
                [illegible] edi
                [illegible] sub_314623
                test    eax, eax
                jz      short loc_31306D
                [illegible] esi
                [illegible] 31308F

loc_313066:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+55
                [illegible]
                call    sub_31411B
loc_31306D:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+49
                call    sub_3140F3
                test    eax, eax
                jg      short loc_31307D
                call    sub_3140F3
                jmp     short loc_31308C
; ---------------------------------------------------------------------------

loc_31307D:                             ; CODE XREF: sub_312FD8
                call    sub_3140F3
                and     eax, 0FFFFh
                or      eax, 80070000h

loc_31308C:                             ; CODE XREF: sub_312FD8
                mov     [ebp+var_4], eax
```

Machine Specific Register Hooking (MSR Hooking)

- This is how we hook SYSENTER
  - SYSENTER switches to kernel-mode using three MSR's
    - IA32_SYSENTER_CS → 0x174, 16-bit selector of ring 0 code segment
    - IA32_SYSENTER_EIP → 0x176, 32-bit offset into ring 0 code segment
    - IA32_SYSENTER_ESP → 0x175, 32-bit stack pointer for ring 0 stack
- Just like the IDTR, there are instructions for accessing the MSR's
  - RDMSR and WRMSR - read/write MSR
  - MSR's are processor specific just like IDT's

MSR Hooking Problems

- More modern than IDT hooking
- Still easy to detect, and only provides pass-through functions :-(

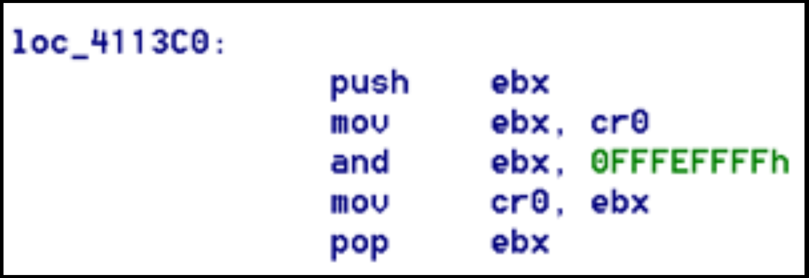


```
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], ebx
                jnz     short loc_313066
                mov     eax, [ebp+var_70]
                cmp     eax, [ebp+var_84]
                jb      short loc_313066
                sub     eax, [ebp+var_84]
                push    esi
                push    esi
                push    eax
                push    edi
                mov     [ebp+arg_0], eax
                call    sub_31486A
                test    eax, eax
                [illegible] short loc_313[illegible]
                push    edi
                lea     eax, [ebp+arg_0]
                push    eax
                mov     esi, 1D0h
                push    esi
                push    [ebp+arg_4]
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], esi
                jz      short loc_31308F

loc_313066:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+55
                push    0Dh
                call    sub_31411B

loc_31306D:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+49
                call    sub_3140F3
                test    eax, eax
                jg      short loc_31307D
                call    sub_3140F3
                jmp     short loc_31308C
; ---------------------------------------------------------------

loc_31307D:                             ; CODE XREF: sub_312FD8
                call    sub_3140F3
                and     eax, 0FFFFh
                or      eax, 80070000h

loc_31308C:                             ; CODE XREF: sub_312FD8
                mov     [ebp+var_4], eax
```

System Descriptor Table Hooking (SDT, SSDT)

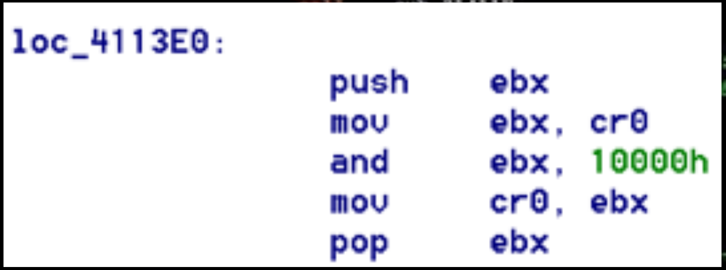
- SSDT resides in read-only memory
    - Rootkits have to disable and then re-enable the Write Protection (WP) bit in the CR0 register
        - Rootkit authors could also map an MDL over the SSDT

Disable WP



```
loc_4113C0:
                push    ebx
                mov     ebx, cr0
                and     ebx, 0FFFEFFFFh
                mov     cr0, ebx
                pop     ebx
```

Enable WP



```
loc_4113E0:
                push    ebx
                mov     ebx, cr0
                and     ebx, 10000h
                mov     cr0, ebx
                pop     ebx
```

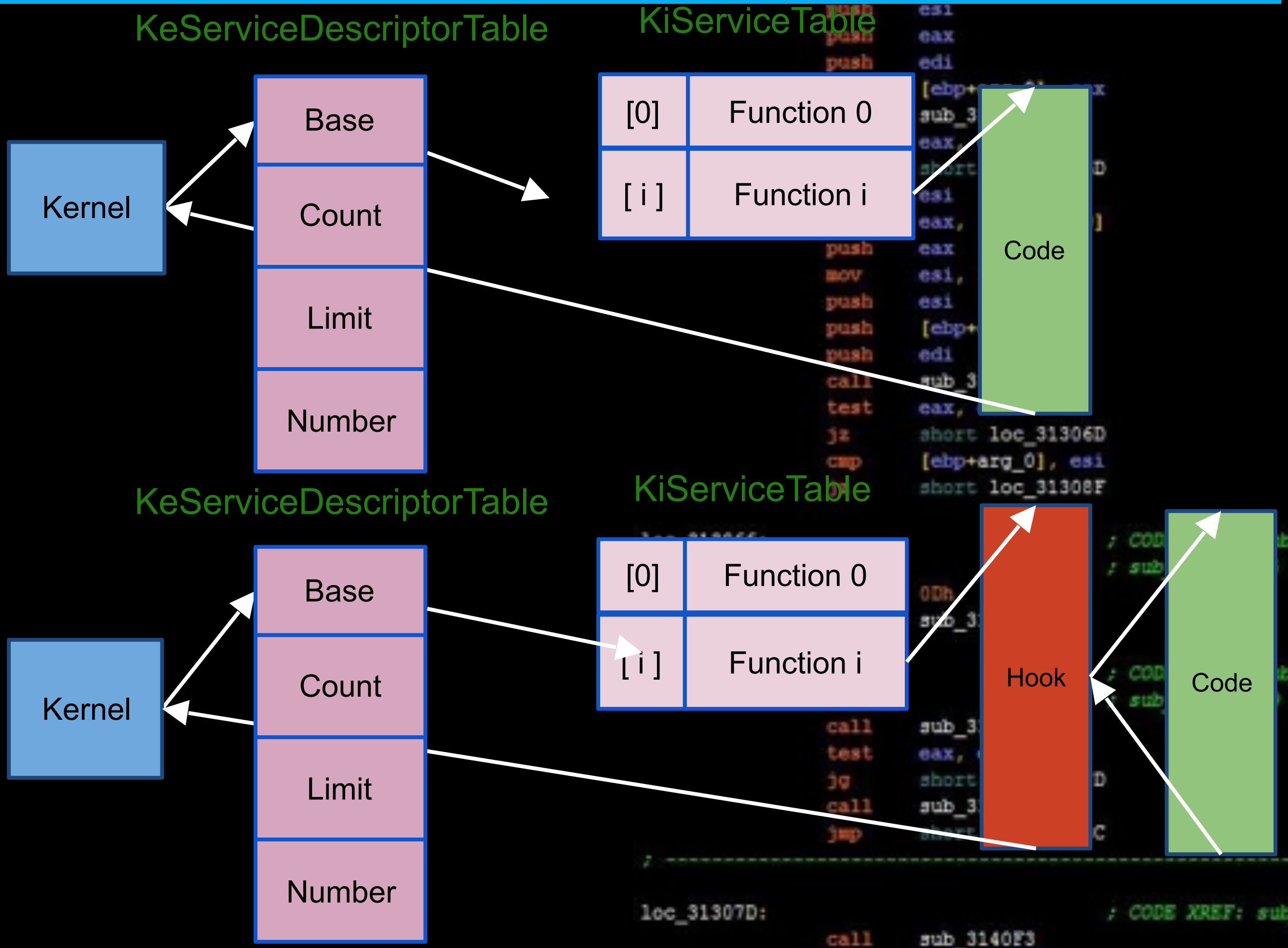
System Descriptor Table Hooking (SDT, SSDT)

- With WP off, the attacker swaps a new address into the target address
  - ○ Declare the original syscall prototype (e.g., ZwSetValueKey())
  - ○ Declare a corresponding function ptr (e.g., ZwSetValueKeyPtr)
  - ○ Define a function ptr (e.g., oldZwSetValueKey)
  - ○ Implement a hook routine (e.g., newZwSetValueKey())
  - ○ InterlockedExchange() to swap in a ptr to new function
    1. The new function can execute the old syscall, and filter the results
       - Hook ZwQueryDirectoryFile() to hide directories
       - Hook ZwQuerySystemInformation() to hide processes

System Descriptor Table Hooking (SDT, SSDT)

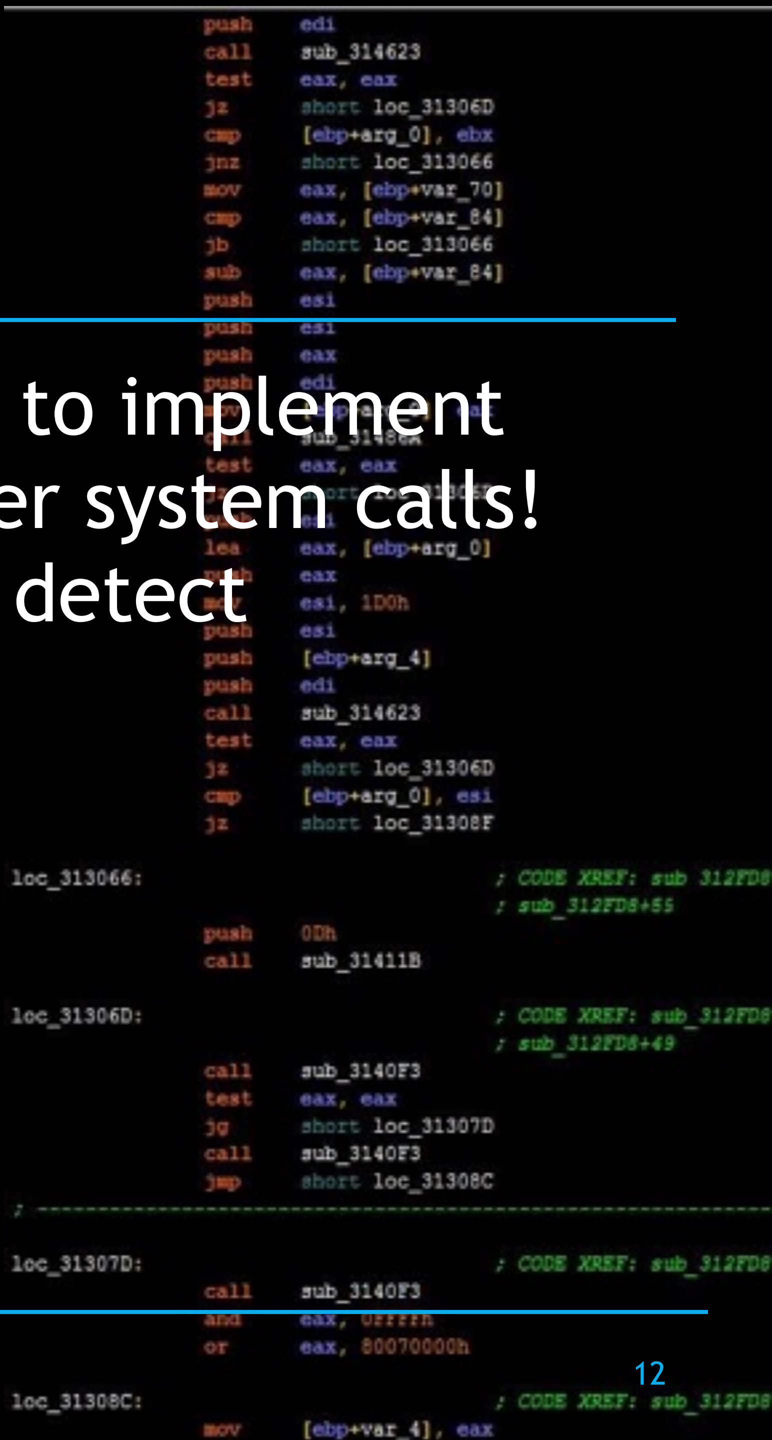
Before Hook:

After Hook:



SSDT Hooking Problems

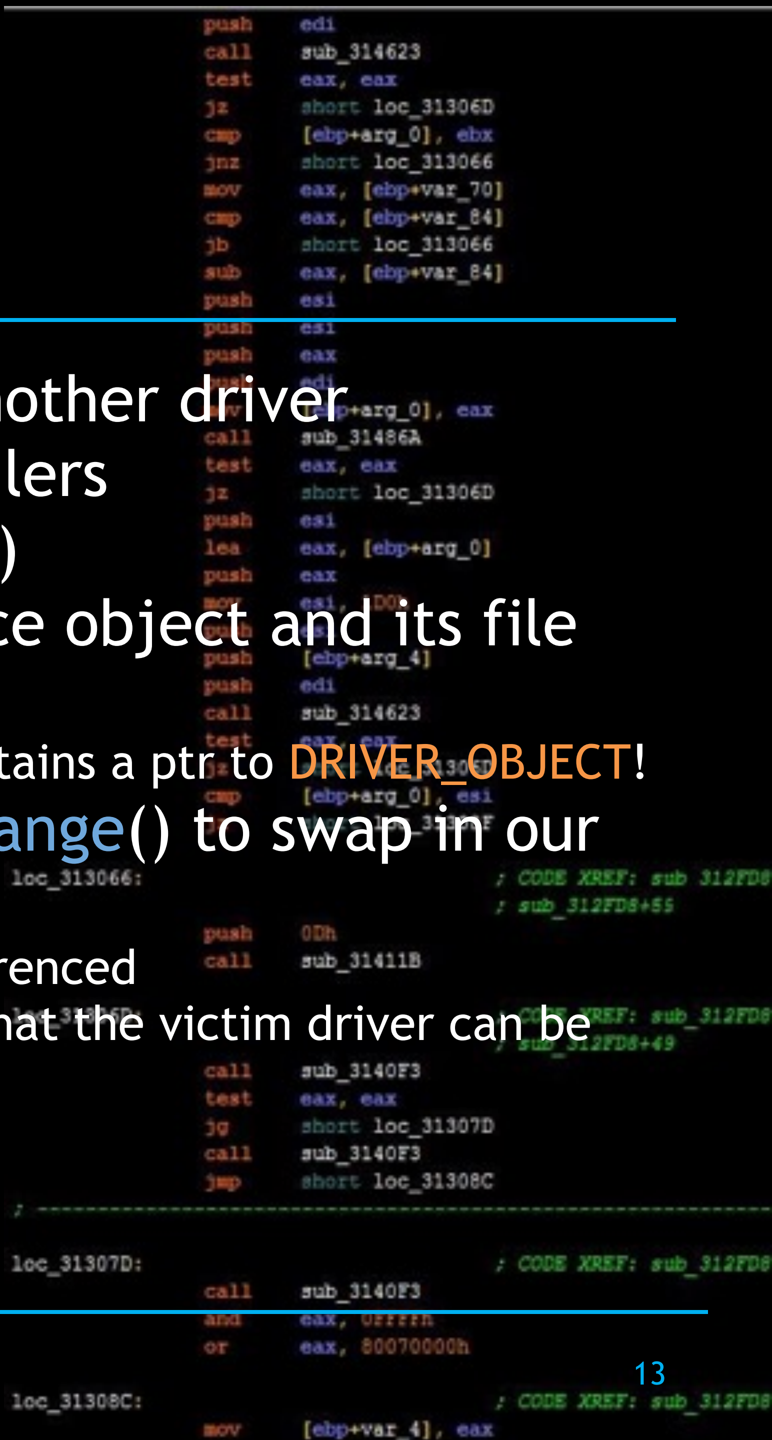
- Relatively straightforward to implement
- Provides the ability to filter system calls!
- On it’s own, still trivial to detect



```
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], ebx
                jnz     short loc_313066
                mov     eax, [ebp+var_70]
                cmp     eax, [ebp+var_84]
                jb      short loc_313066
                sub     eax, [ebp+var_84]
                push    esi
                push    esi
                push    eax
                push    edi
                mov     [ebp+arg_0], eax
                call    sub_31486A
                test    eax, eax
                jz      short loc_31306D
                push    esi
                lea     eax, [ebp+arg_0]
                push    eax
                mov     esi, 1D0h
                push    esi
                push    [ebp+arg_4]
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], esi
                jz      short loc_31308F

loc_313066:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+55
                push    0Dh
                call    sub_31411B

loc_31306D:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+49
                call    sub_3140F3
                test    eax, eax
                jg      short loc_31307D
                call    sub_3140F3
                jmp     short loc_31308C
; ---------------------------------------------------------------

loc_31307D:                             ; CODE XREF: sub_312FD8
                call    sub_3140F3
                and     eax, 0FFFFh
                or      eax, 80070000h

loc_31308C:                             ; CODE XREF: sub_312FD8
                mov     [ebp+var_4], eax
```

Hooking IRP Handlers

- Access the DRIVER_OBJECT of another driver
    - Hook the MajorFunction handlers
    - IoGetDeviceByObjectPointer()
        - Returns a ptr to to a device object and its file object.
            - DEVICE_OBJECT structure contains a ptr to DRIVER_OBJECT!
        - Then use InterlockedExchange() to swap in our hook function
            - Device object must be dereferenced (ObDereferenceObject()) So that the victim driver can be unloaded in the future



```
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], ebx
                jnz     short loc_313066
                mov     eax, [ebp+var_70]
                cmp     eax, [ebp+var_84]
                jb      short loc_313066
                sub     eax, [ebp+var_84]
                push    esi
                push    esi
                push    eax
                push    edi
                mov     [ebp+arg_0], eax
                call    sub_31486A
                test    eax, eax
                jz      short loc_31306D
                push    esi
                lea     eax, [ebp+arg_0]
                push    eax
                mov     esi, 1D0h
                push    esi
                push    [ebp+arg_4]
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], esi
                jnz     short loc_31307F

loc_313066:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+55
                push    0Dh
                call    sub_31411B

loc_31306D:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+49
                call    sub_3140F3
                test    eax, eax
                jg      short loc_31307D
                call    sub_3140F3
                jmp     short loc_31308C
; ---------------------------------------------------------------------------

loc_31307D:                             ; CODE XREF: sub_312FD8
                call    sub_3140F3
                and     eax, 0FFFFh
                or      eax, 80070000h

loc_31308C:                             ; CODE XREF: sub_312FD8
                mov     [ebp+var_4], eax
```

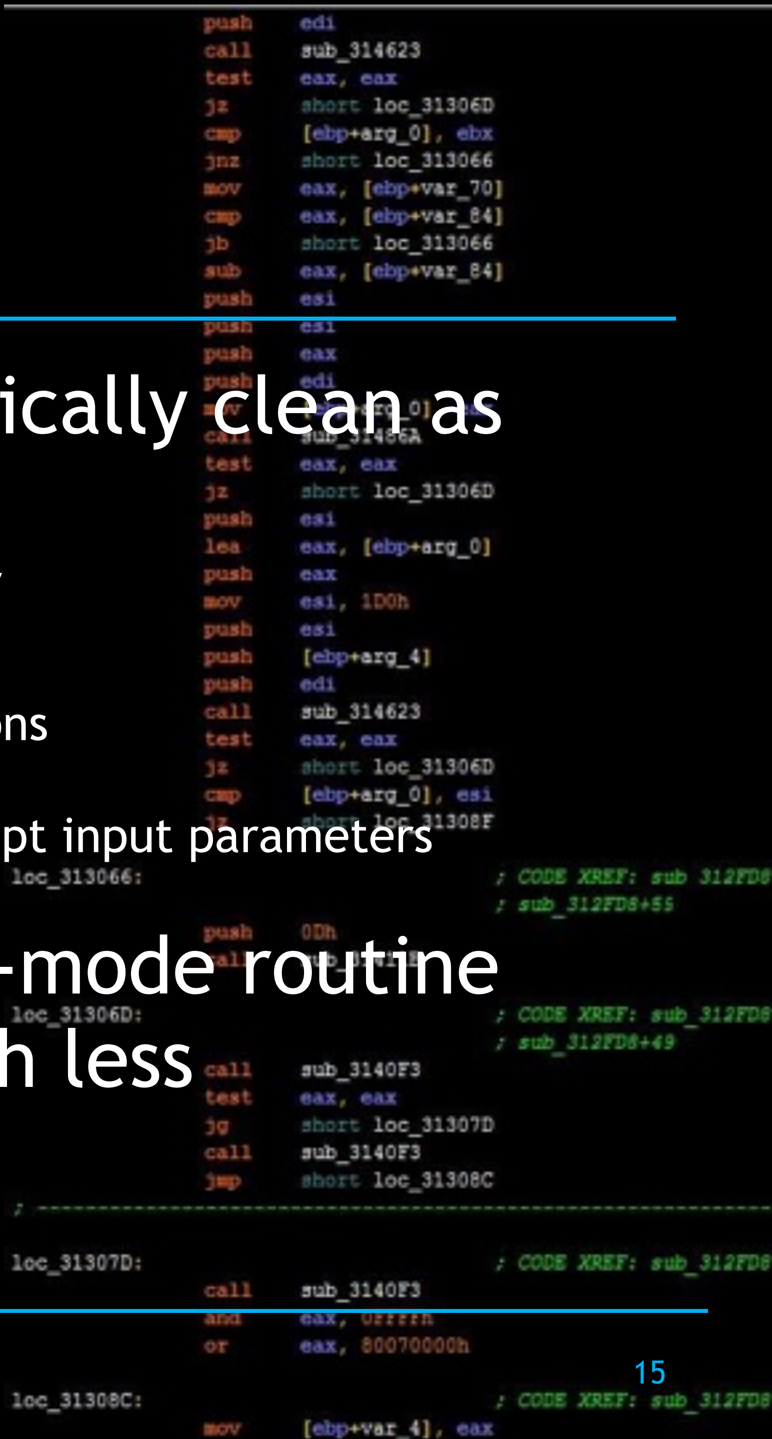
Agenda

- Hooking
- Memory Patching
- Direct Kernel Object Manipulation

```
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], ebx
                jnz     short loc_313066
                mov     eax, [ebp+var_70]
                cmp     eax, [ebp+var_84]
                jb      short loc_313066
                sub     eax, [ebp+var_84]
                push    esi
                push    esi
                push    eax
                push    edi
                mov     [ebp+arg_0], eax
                call    sub_31486A
                test    eax, eax
                jz      short loc_31306D
                push    esi
                lea     eax, [ebp+arg_0]
                push    eax
                mov     esi, 1D0h
                push    esi
                push    [ebp+arg_4]
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], esi
                jz      short loc_31308F

loc_313066:                             ; CODE XREF: sub 312FD8
                                        ; sub_312FD8+55
                push    0Dh
                call    sub_31411B

loc_31306D:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+49
                call    sub_3140F3
                test    eax, eax
                jg      short loc_31307D
                call    sub_3140F3
                jmp     short loc_31308C
; ---------------------------------------------------------------

loc_31307D:                             ; CODE XREF: sub_312FD8
                call    sub_3140F3
                and     eax, 0FFFFh
                or      eax, 80070000h

loc_31308C:                             ; CODE XREF: sub_312FD8
                mov     [ebp+var_4], eax
```

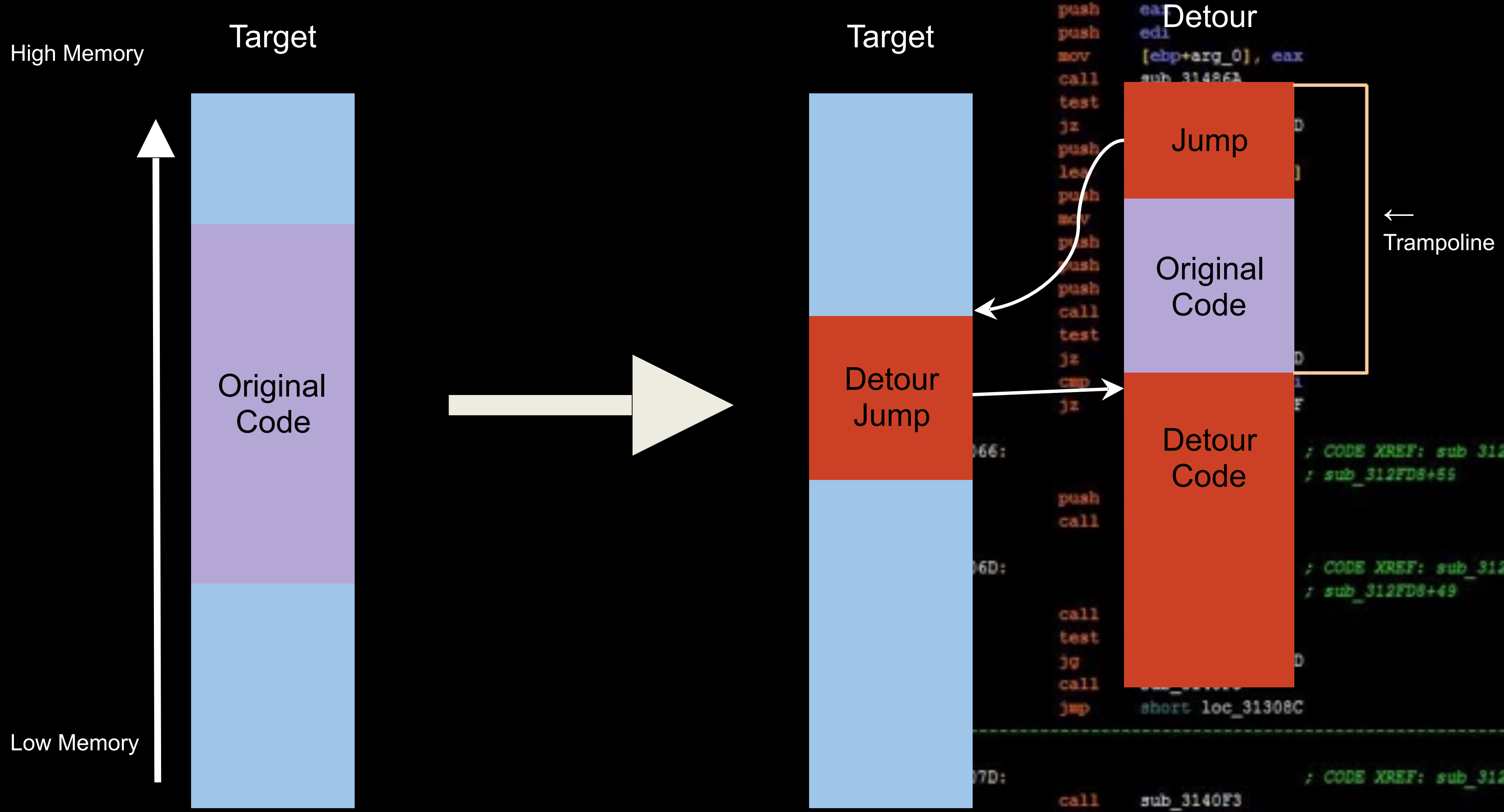
Detour Patching

- Not nearly as programmatically clean as hooking
  - However, the payoff is higher
  - We can:
    - Block calls made by applications
    - Replace entire routines
    - Trace system calls and intercept input parameters
    - Filter output parameters
- We can modify any kernel-mode routine
- Detecting patching is much less straightforward

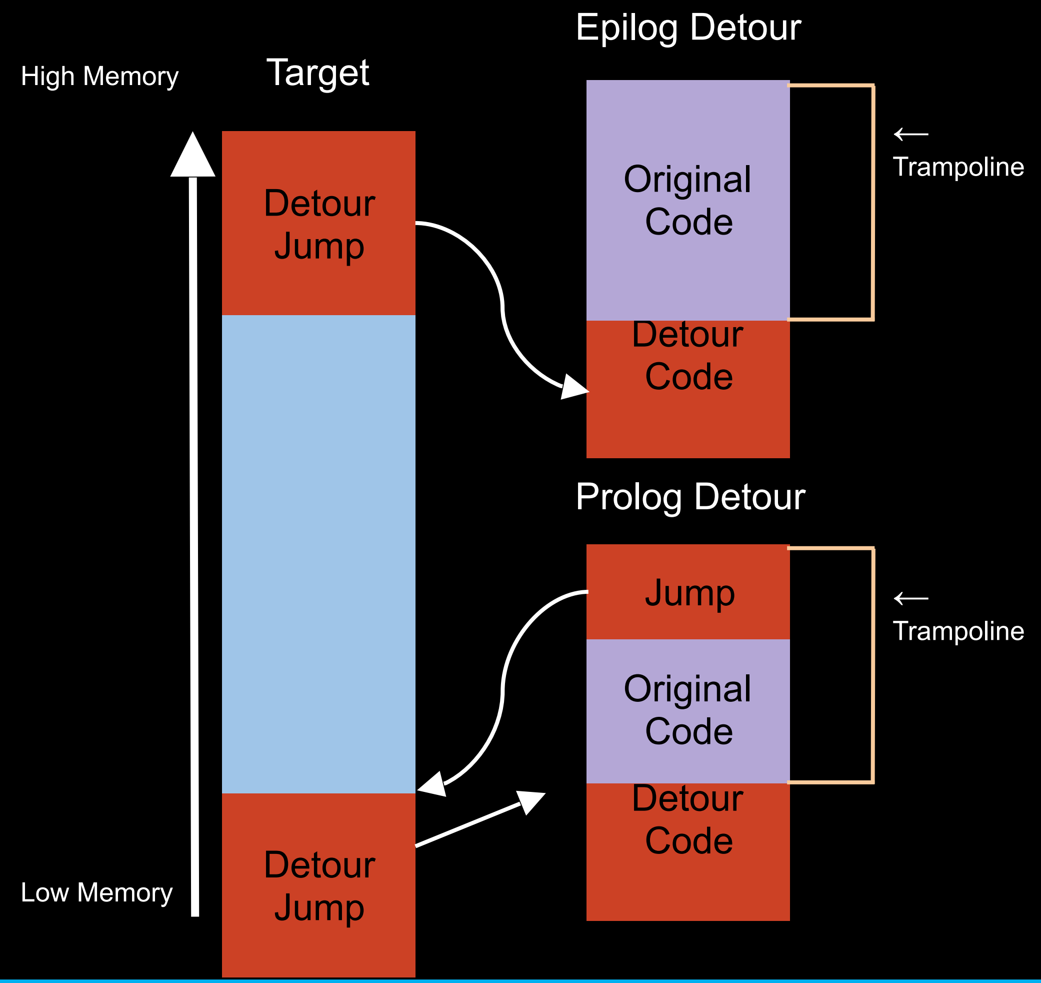


```
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], ebx
                jnz     short loc_313066
                mov     eax, [ebp+var_70]
                cmp     eax, [ebp+var_84]
                jb      short loc_313066
                sub     eax, [ebp+var_84]
                push    esi
                push    esi
                push    eax
                push    edi
                mov     [ebp+arg_0], eax
                call    sub_31466A
                test    eax, eax
                jz      short loc_31306D
                push    esi
                lea     eax, [ebp+arg_0]
                push    eax
                mov     esi, 1D0h
                push    esi
                push    [ebp+arg_4]
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], esi
                jz      short loc_31308F

loc_313066:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+55
                push    0Dh
                call    [illegible]

loc_31306D:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+49
                call    sub_3140F3
                test    eax, eax
                jg      short loc_31307D
                call    sub_3140F3
                jmp     short loc_31308C
; ---------------------------------------------------------------

loc_31307D:                             ; CODE XREF: sub_312FD8
                call    sub_3140F3
                and     eax, 0FFFFh
                or      eax, 80070000h

loc_31308C:                             ; CODE XREF: sub_312FD8
                mov     [ebp+var_4], eax
```

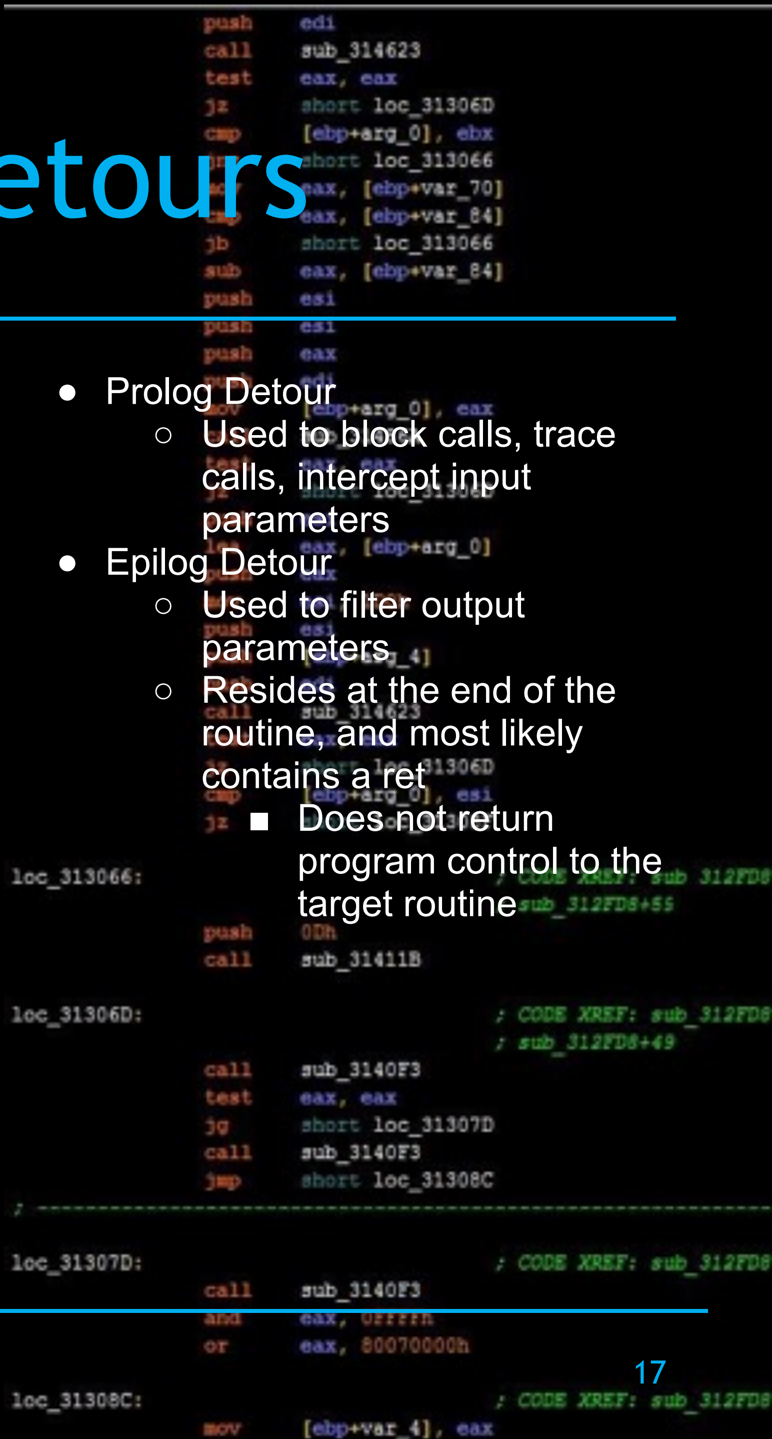
Detour Patching



Epilog and Prolog Detours



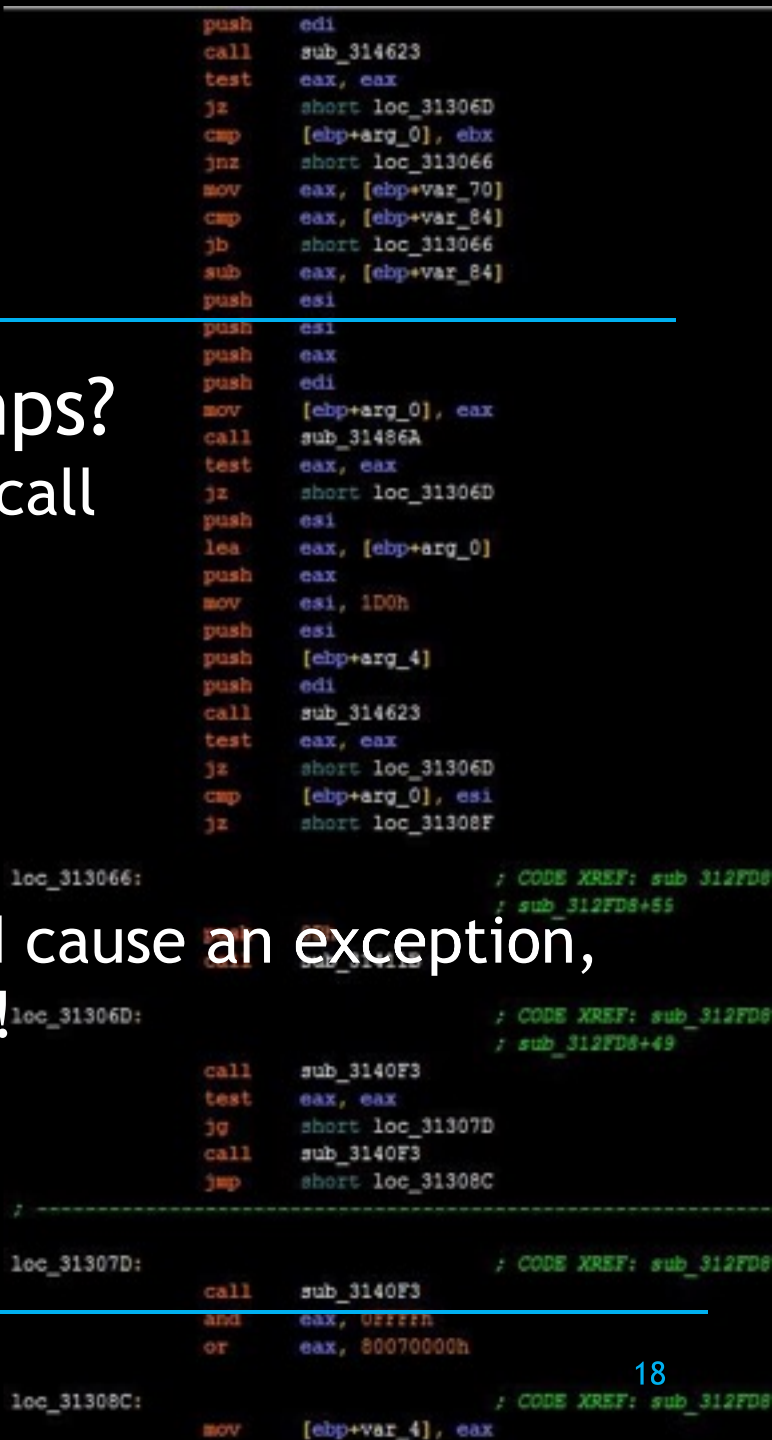
- Prolog Detour
  - Used to block calls, trace calls, intercept input parameters
- Epilog Detour
  - Used to filter output parameters
  - Resides at the end of the routine, and most likely contains a ret
    - Does not return program control to the target routine



```
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], ebx
                [illegible] short loc_313066
                mov     eax, [ebp+var_70]
                cmp     eax, [ebp+var_84]
                jb      short loc_313066
                sub     eax, [ebp+var_84]
                push    esi
                push    esi
                push    eax
                [illegible]
                mov     [ebp+arg_0], eax
                [illegible]
                [illegible] short loc_31306D
                [illegible] eax, [ebp+arg_0]
                [illegible]
                call    sub_314623
                [illegible] short loc_31306D
                cmp     [ebp+arg_0], esi
                jz      [illegible]

loc_313066:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+55
                push    0Dh
                call    sub_31411B

loc_31306D:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+49
                call    sub_3140F3
                test    eax, eax
                jg      short loc_31307D
                call    sub_3140F3
                jmp     short loc_31308C
; ---------------------------------------------------------------------------

loc_31307D:                             ; CODE XREF: sub_312FD8
                call    sub_3140F3
                and     eax, 0FFFFh
                or      eax, 80070000h

loc_31308C:                             ; CODE XREF: sub_312FD8
                mov     [ebp+var_4], eax
```

Detour Jumps

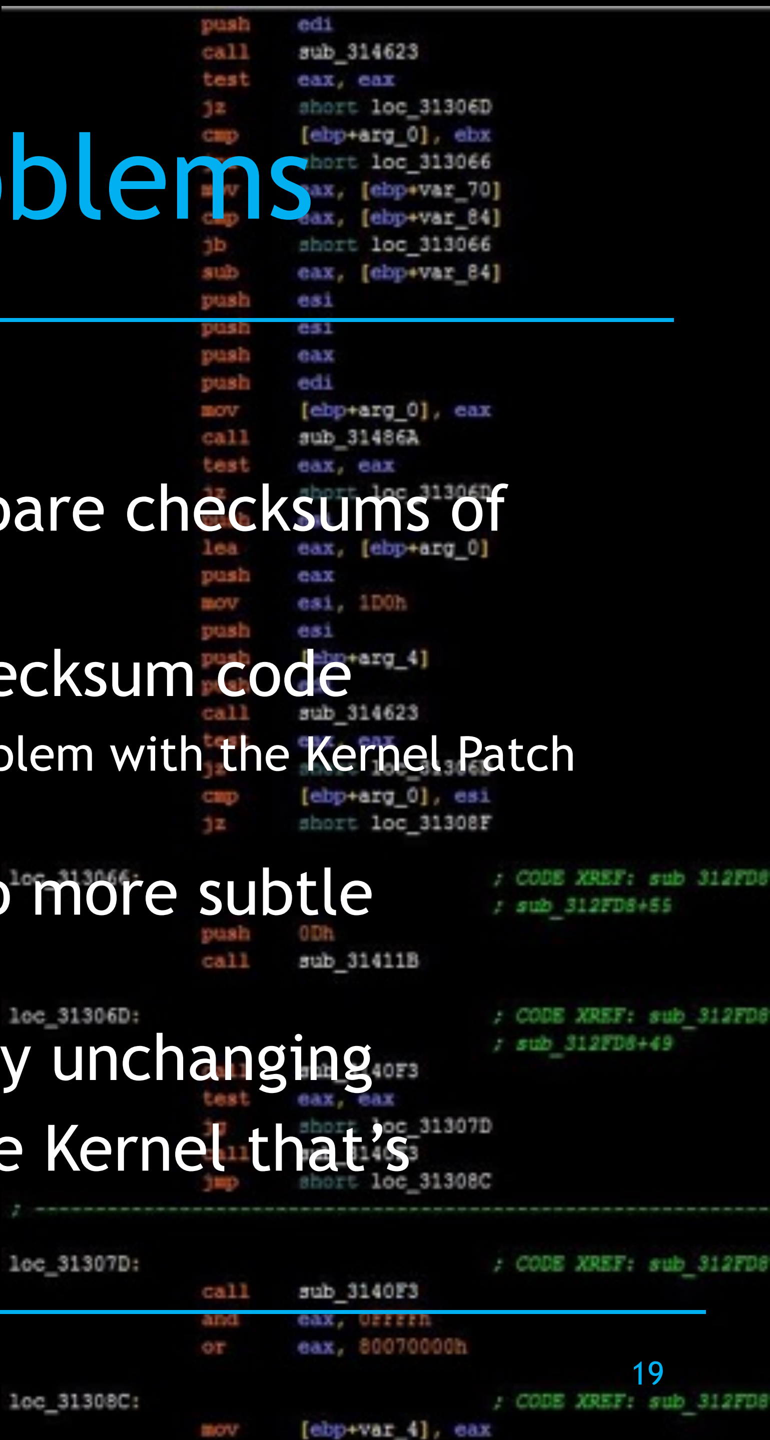
- How do rootkits place jumps?
  - More obvious - near Jump or call
    - mov ebx, 0xCAFEBABE
    - jmp [ebx] or call [ebx]
  - Middle ground - push and ret
    - push 0xCAFEBABE
    - ret
  - Less obvious - modify IDT and cause an exception, just like our anti-analysis lab!



```
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], ebx
                jnz     short loc_313066
                mov     eax, [ebp+var_70]
                cmp     eax, [ebp+var_84]
                jb      short loc_313066
                sub     eax, [ebp+var_84]
                push    esi
                push    esi
                push    eax
                push    edi
                mov     [ebp+arg_0], eax
                call    sub_31486A
                test    eax, eax
                jz      short loc_31306D
                push    esi
                lea     eax, [ebp+arg_0]
                push    eax
                mov     esi, 1D0h
                push    esi
                push    [ebp+arg_4]
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], esi
                jz      short loc_31308F

loc_313066:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+55
                [illegible]

loc_31306D:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+49
                call    sub_3140F3
                test    eax, eax
                jg      short loc_31307D
                call    sub_3140F3
                jmp     short loc_31308C
; ---------------------------------------------------------------

loc_31307D:                             ; CODE XREF: sub_312FD8
                call    sub_3140F3
                and     eax, 0FFFFh
                or      eax, 80070000h

loc_31308C:                             ; CODE XREF: sub_312FD8
                mov     [ebp+var_4], eax
```

Detour Patching Problems

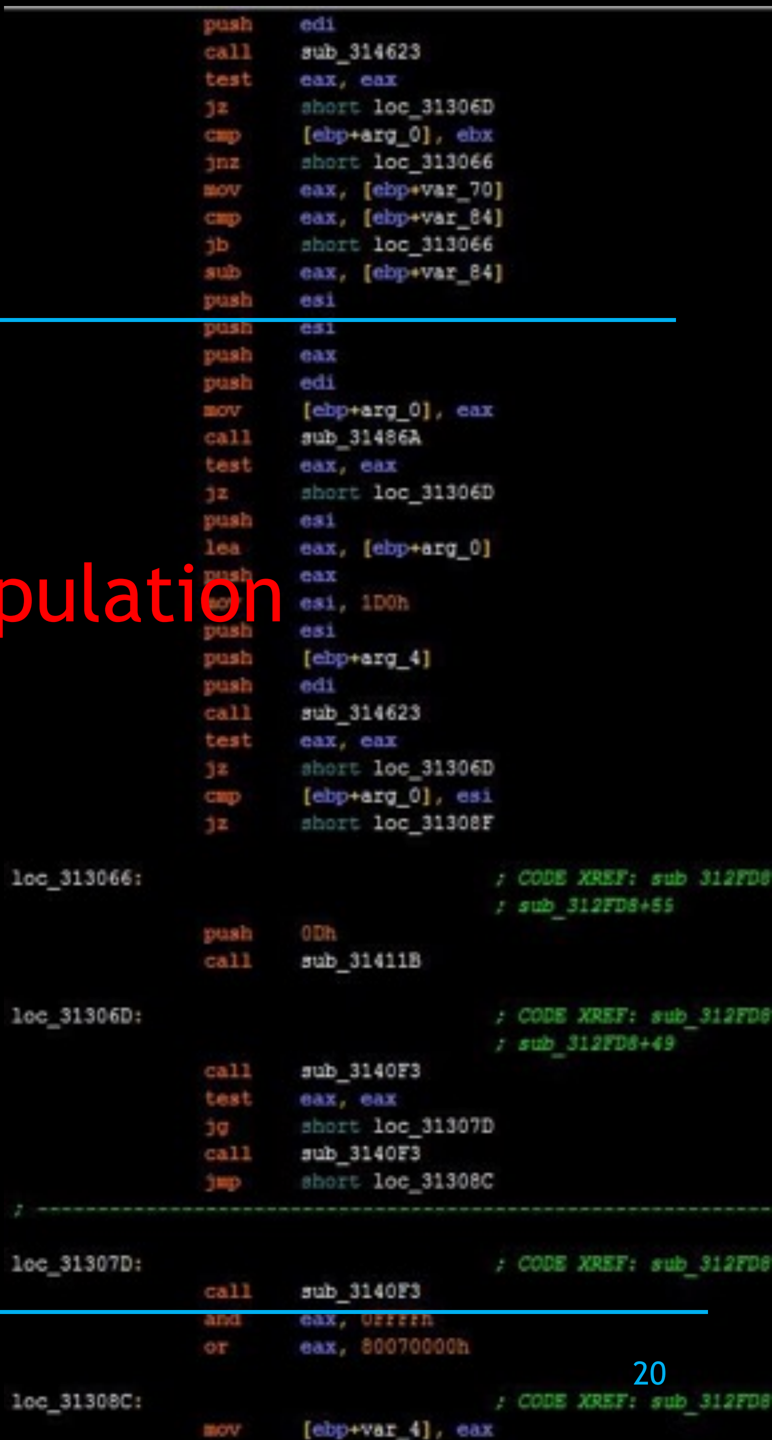
- Detour Patching detection
  - Analysts can create and compare checksums of functions
    - Rootkits can patch the checksum code
      - This is Microsoft’s current problem with the Kernel Patch Protection feature
  - Most rootkit authors prefer to more subtle techniques
    - Code is static and normally unchanging
    - Instead, alter a part of the Kernel that’s dynamic!



```
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], ebx
                        short loc_313066
                        eax, [ebp+var_70]
                        eax, [ebp+var_84]
                jb      short loc_313066
                sub     eax, [ebp+var_84]
                push    esi
                push    esi
                push    eax
                push    edi
                mov     [ebp+arg_0], eax
                call    sub_31486A
                test    eax, eax
                                 loc_31306D
                lea     eax, [ebp+arg_0]
                push    eax
                mov     esi, 1D0h
                push    esi
                                 +arg_4]
                call    sub_314623
                cmp     [ebp+arg_0], esi
                jz      short loc_31308F

loc_313066:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+55
                push    0Dh
                call    sub_31411B

loc_31306D:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+49
                            40F3
                test    eax, eax
                        short loc_31307D
                jmp     short loc_31308C
; ---------------------------------------------------------------

loc_31307D:                             ; CODE XREF: sub_312FD8
                call    sub_3140F3
                and     eax, 0FFFFh
                or      eax, 80070000h

loc_31308C:                             ; CODE XREF: sub_312FD8
                mov     [ebp+var_4], eax
```

Agenda

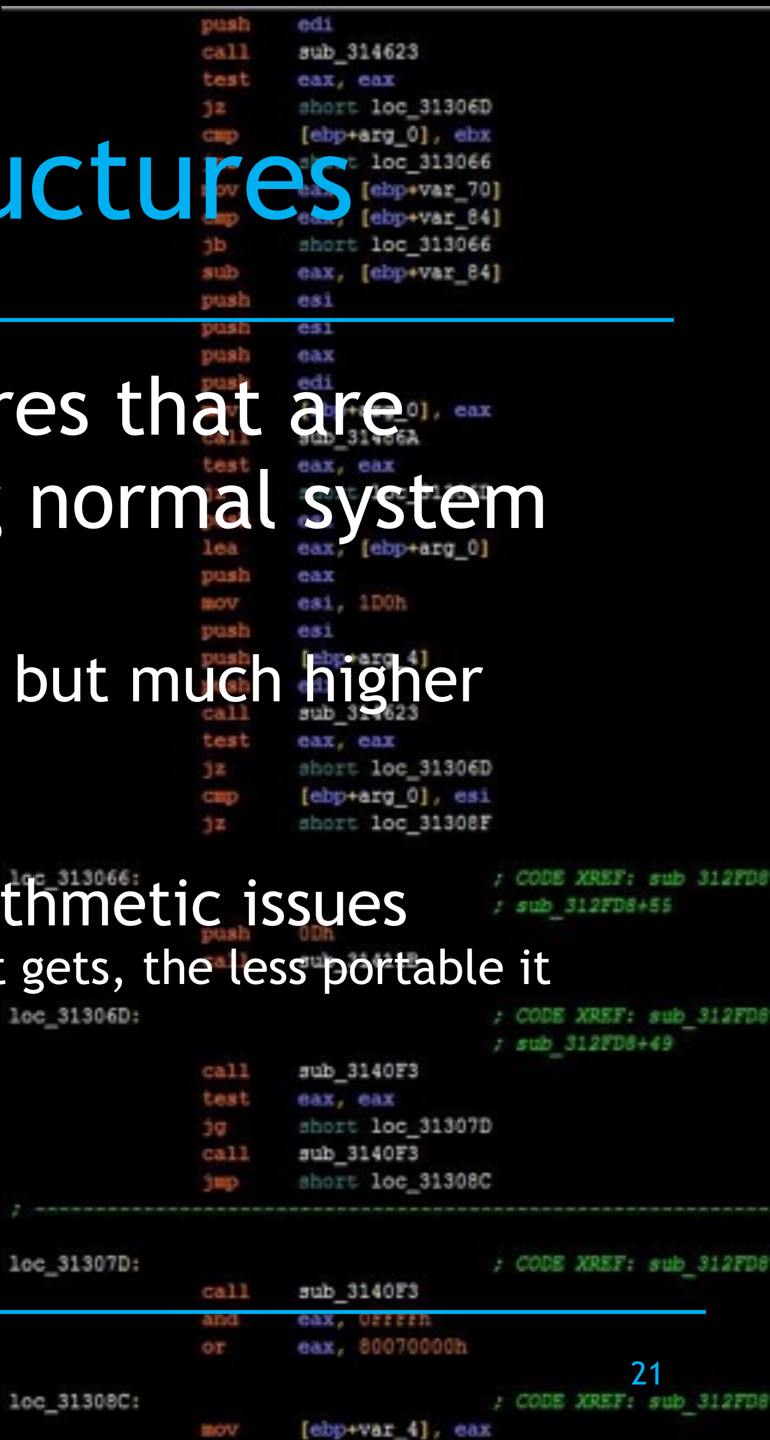
- Hooking
- Memory Patching
- Direct Kernel Object Manipulation



```
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], ebx
                jnz     short loc_313066
                mov     eax, [ebp+var_70]
                cmp     eax, [ebp+var_84]
                jb      short loc_313066
                sub     eax, [ebp+var_84]
                push    esi
                push    esi
                push    eax
                push    edi
                mov     [ebp+arg_0], eax
                call    sub_31486A
                test    eax, eax
                jz      short loc_31306D
                push    esi
                lea     eax, [ebp+arg_0]
                push    eax
                mov     esi, 1D0h
                push    esi
                push    [ebp+arg_4]
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], esi
                jz      short loc_31308F

loc_313066:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+55
                push    0Dh
                call    sub_31411B

loc_31306D:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+49
                call    sub_3140F3
                test    eax, eax
                jg      short loc_31307D
                call    sub_3140F3
                jmp     short loc_31308C
; -------------------------------------------------------------------

loc_31307D:                             ; CODE XREF: sub_312FD8
                call    sub_3140F3
                and     eax, 0FFFFh
                or      eax, 80070000h

loc_31308C:                             ; CODE XREF: sub_312FD8
                mov     [ebp+var_4], eax
```

Dynamic Kernel Structures

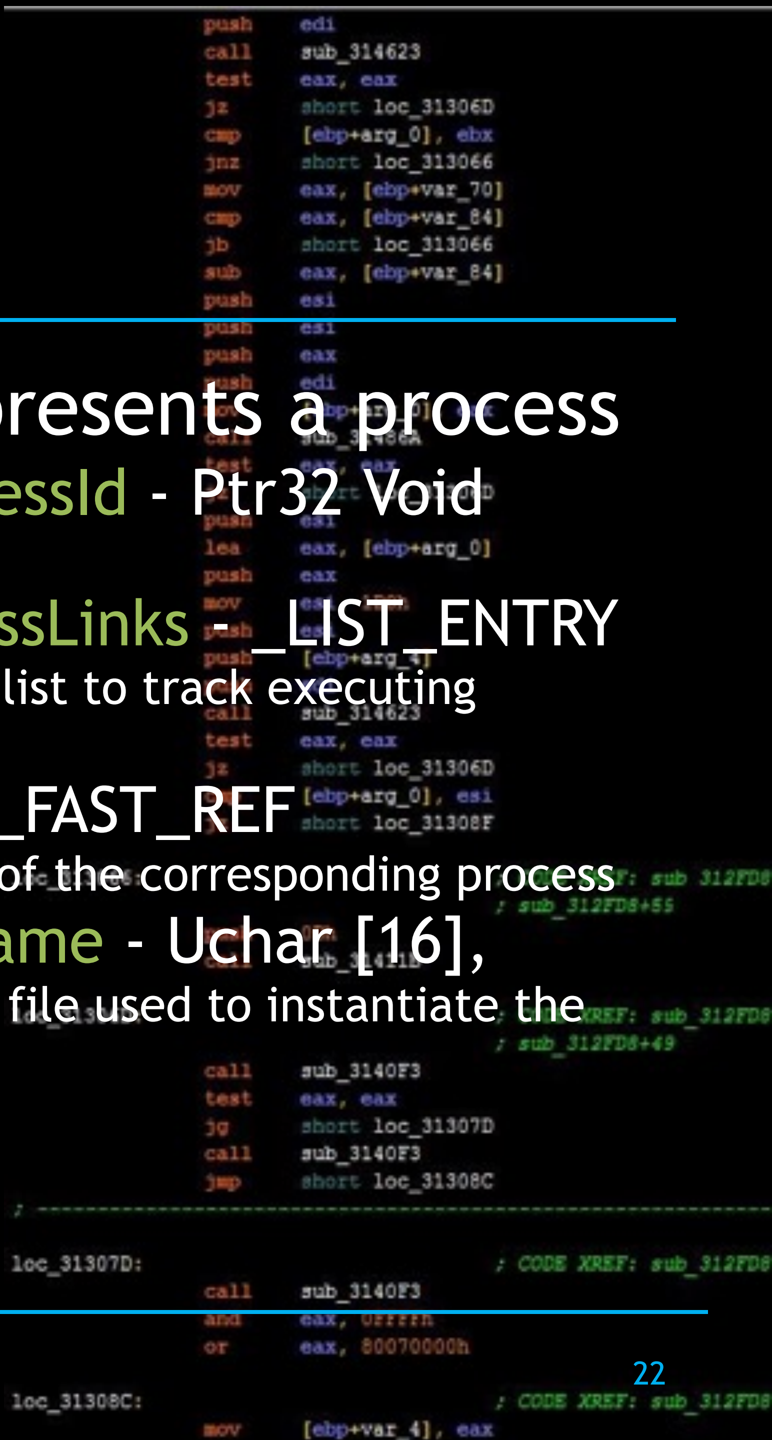
- Manipulate kernel structures that are frequently updated during normal system operation
  - Even higher levels of stealth, but much higher complexity
    - Concurrency issues
    - Portability and pointer arithmetic issues
      - The more specialized a rootkit gets, the less portable it becomes



```
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], ebx
                        short loc_313066
                        [ebp+var_70]
                        [ebp+var_84]
                jb      short loc_313066
                sub     eax, [ebp+var_84]
                push    esi
                push    esi
                push    eax
                        edi
                                  0], eax
                test    eax, eax
                lea     eax, [ebp+arg_0]
                push    eax
                mov     esi, 1D0h
                push    esi
                call    sub_3
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], esi
                jz      short loc_31308F

loc_313066:                             ; CODE XREF: sub_312FD
                                        ; sub_312FD8+55
                        0Dh

loc_31306D:                             ; CODE XREF: sub_312FD
                                        ; sub_312FD8+49
                call    sub_3140F3
                test    eax, eax
                jg      short loc_31307D
                call    sub_3140F3
                jmp     short loc_31308C
; ---------------------------------------------------------------

loc_31307D:                             ; CODE XREF: sub_312FD
                call    sub_3140F3
                and     eax,
                or      eax, 80070000h

loc_31308C:                             ; CODE XREF: sub_312FD
                mov     [ebp+var_4], eax
```

EPROCESS Object

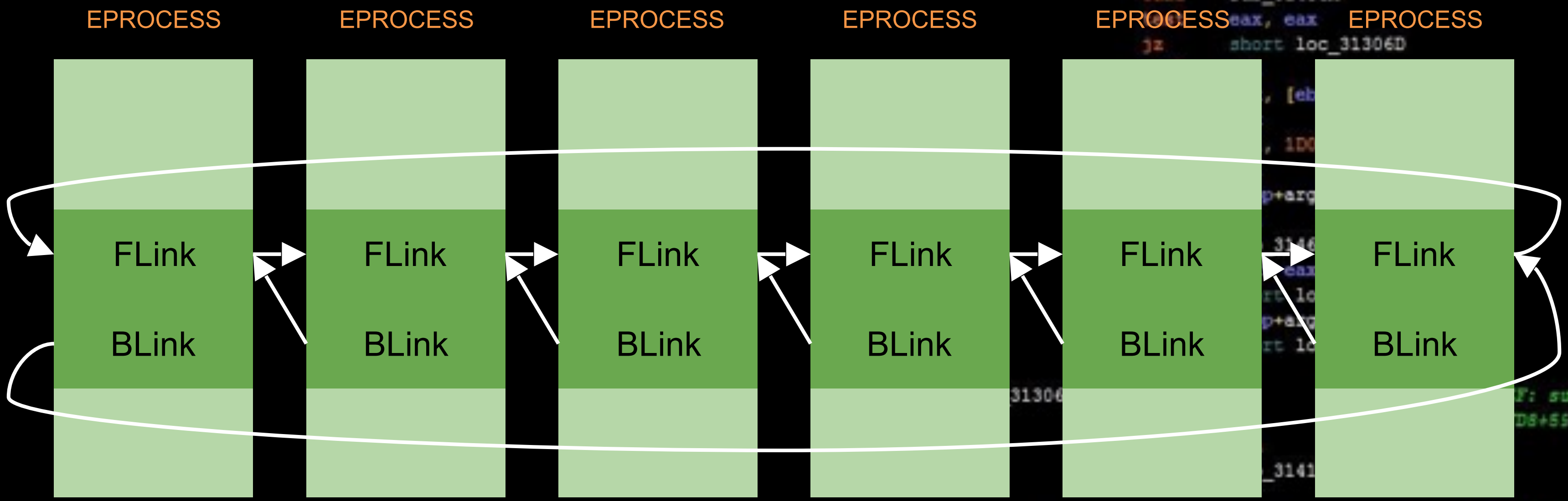
- Opaque structure that represents a process
  - Offset 0x09C: UniqueProcessId - Ptr32 Void
    - Pointer to a 32-bit process ID
  - Offset 0x0a0: ActiveProcessLinks - _LIST_ENTRY
    - Windows uses a doubly linked list to track executing processes
  - Offset 0x0E0: Token - _EX_FAST_REF
    - Address of the security token of the corresponding process
  - Offset 0x14C: ImageFileName - Uchar [16],
    - Stores the name of the binary file used to instantiate the process



```
push    edi
call    sub_314623
test    eax, eax
jz      short loc_31306D
cmp     [ebp+arg_0], ebx
jnz     short loc_313066
mov     eax, [ebp+var_70]
cmp     eax, [ebp+var_84]
jb      short loc_313066
sub     eax, [ebp+var_84]
push    esi
push    esi
push    eax
push    edi
```

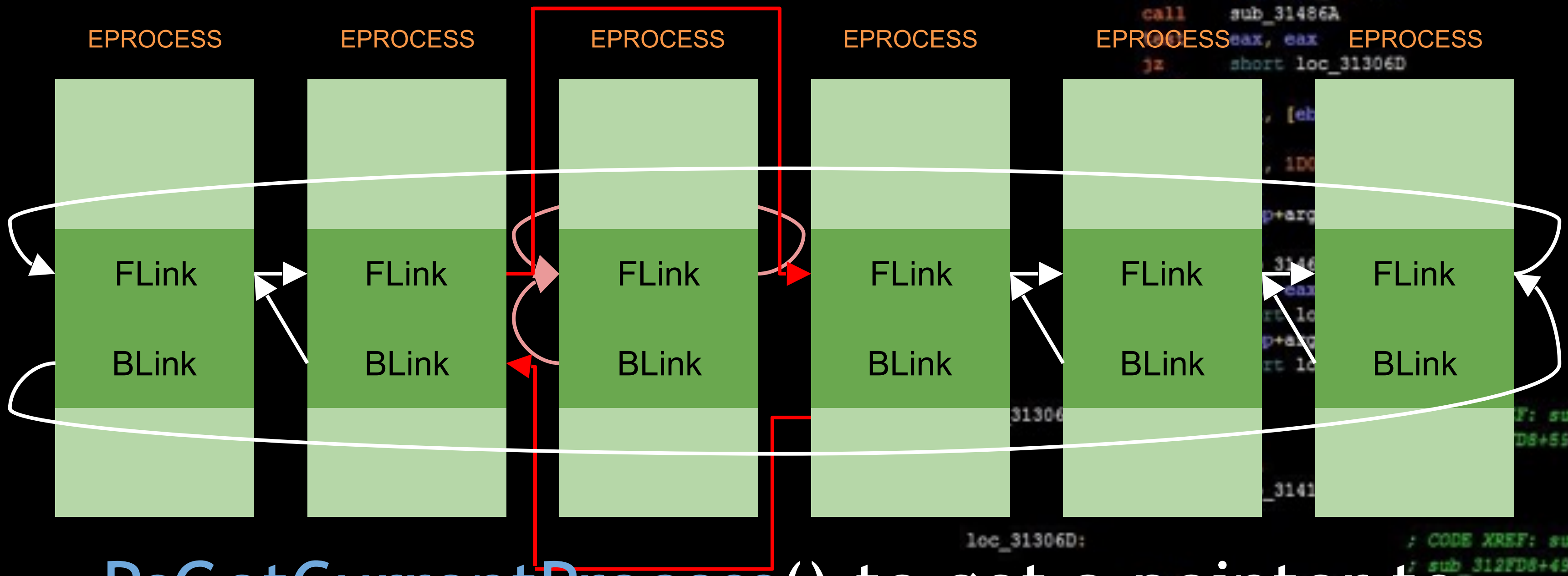
```
call    sub_3140F3
test    eax, eax
jg      short loc_31307D
call    sub_3140F3
jmp     short loc_31308C
; ---------------------------------------------------------------

loc_31307D:                             ; CODE XREF: sub_312FD8
call    sub_3140F3
and     eax, 0FFFFh
or      eax, 80070000h

loc_31308C:                             ; CODE XREF: sub_312FD8
mov     [ebp+var_4], eax
```

EPROCESS Manipulation



- Doubly linked list can be modified to hide a process

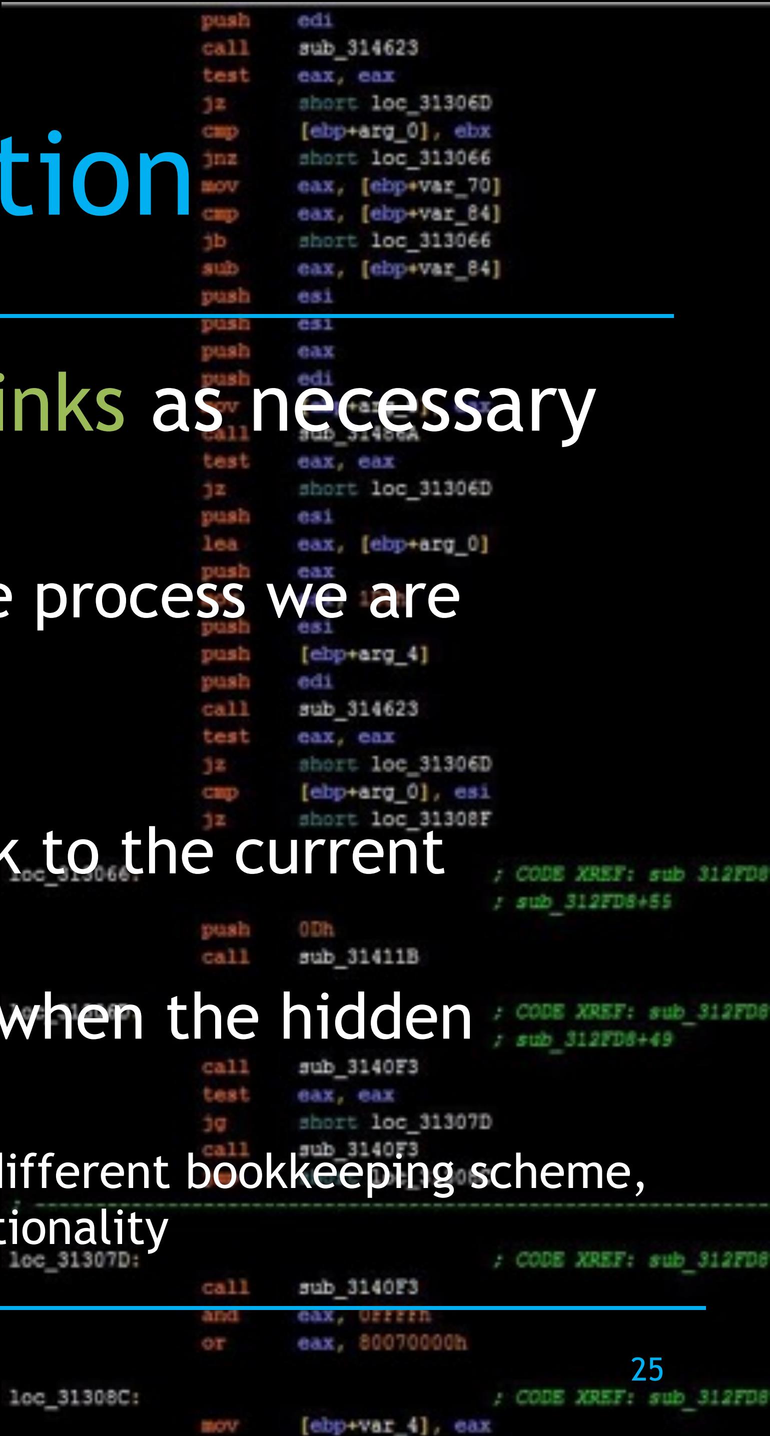
EPROCESS Manipulation



- PsGetCurrentProcess() to get a pointer to the current EPROCESS, then traverse the list

EPROCESS Manipulation

- Modify the ActiveProcessLinks as necessary
  - Neighboring processes
    - FLink and BLink ignore the process we are hiding
  - Process being hidden
    - FLink and BLink point back to the current process
    - This is to prevent a BSOD when the hidden process is terminated
      - The kernel dispatcher uses a different bookkeeping scheme, there is no loss of kernel functionality

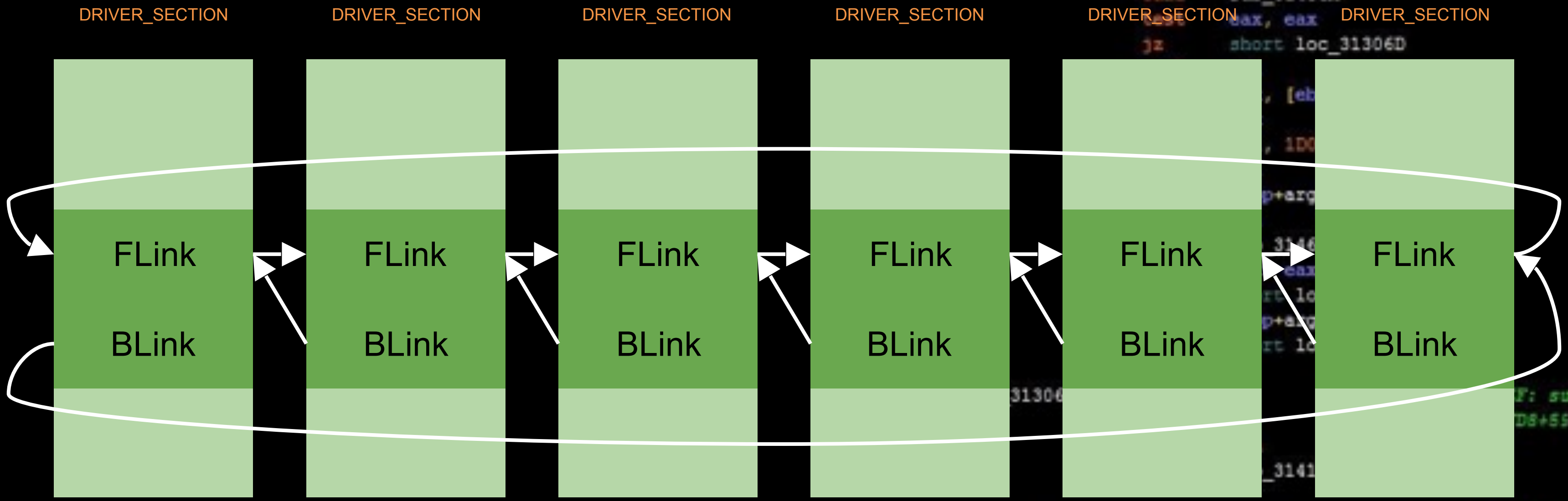


```
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], ebx
                jnz     short loc_313066
                mov     eax, [ebp+var_70]
                cmp     eax, [ebp+var_84]
                jb      short loc_313066
                sub     eax, [ebp+var_84]
                push    esi
                push    esi
                push    eax
                push    edi
                mov     [illegible]
                call    sub_31456A
                test    eax, eax
                jz      short loc_31306D
                push    esi
                lea     eax, [ebp+arg_0]
                push    eax
                [illegible]
                push    esi
                push    [ebp+arg_4]
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], esi
                jz      short loc_31308F

loc_313066:                             ; CODE XREF: sub 312FD8
                                        ; sub_312FD8+55
                push    0Dh
                call    sub_31411B

[illegible]                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+49
                call    sub_3140F3
                test    eax, eax
                jg      short loc_31307D
                call    sub_3140F3
                [illegible]
; ---------------------------------------------------------------

loc_31307D:                             ; CODE XREF: sub_312FD8
                call    sub_3140F3
                and     eax, 0FFFFh
                or      eax, 80070000h

loc_31308C:                             ; CODE XREF: sub_312FD8
                mov     [ebp+var_4], eax
```

DRIVER_SECTION Object

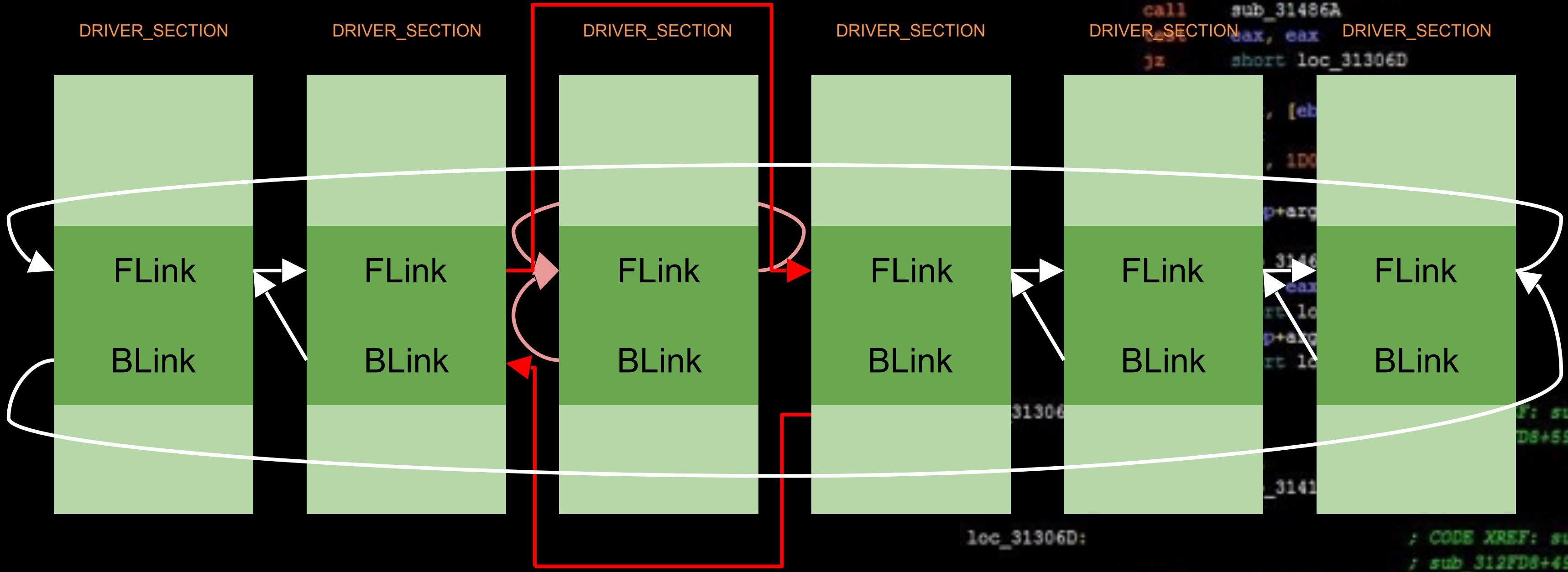
- Another very frequently manipulated structure
  - Used to help the system track loaded drivers
  - VOID ptr in the DRIVER_OBJECT points to it
    - Contains fields like filePath and fileName
  - The first entry in a DRIVER_SECTION is a _LIST_ENTRY
    - This list entry has a FLink and a BLink
    - Drivers can be hidden the exact same was a processes!

DRIVER_SECTION Manipulation



- Doubly linked list can be modified to hide a process

DRIVER_SECTION Manipulation

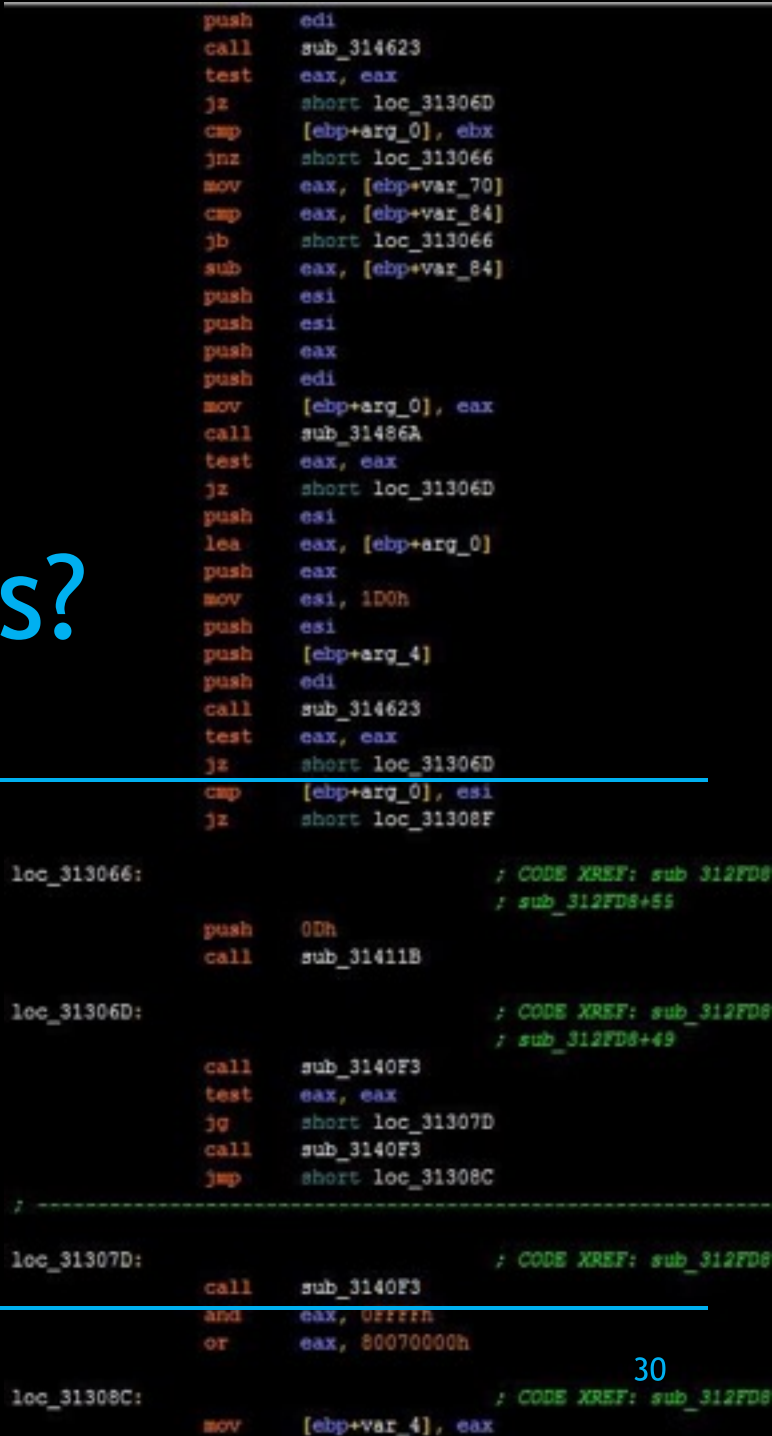


```
push    edi
call    sub_314623
test    eax, eax
jz      short loc_31306D
cmp     [ebp+arg_0], ebx
jb      short loc_313066
sub     eax, [ebp+var_84]
push    esi
push    esi
push    eax
push    edi
mov     [ebp+arg_0], eax
call    sub_31486A
jz      short loc_31306D

loc_31306D:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+49
call    sub_3140F3
test    eax, eax
jg      short loc_31307D
call    sub_3140F3
jmp     short loc_31308C
; ---------------------------------------------------------------

loc_31307D:                             ; CODE XREF: sub_312FD8
call    sub_3140F3
and     eax, 0FFFFh
or      eax, 80070000h

loc_31308C:                             ; CODE XREF: sub_312FD8
mov     [ebp+var_4], eax
```

Access Tokens

- Each process gets an access token
  - Specifies the user, security groups, and privileges associated with the process
  - All of these fields can be edited by a rootkit!
    - You can change the user running a process, its privileges, etc.
  - Each EPROCESS holds a pointer to its TOKEN object

Questions?



```
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], ebx
                jnz     short loc_313066
                mov     eax, [ebp+var_70]
                cmp     eax, [ebp+var_84]
                jb      short loc_313066
                sub     eax, [ebp+var_84]
                push    esi
                push    esi
                push    eax
                push    edi
                mov     [ebp+arg_0], eax
                call    sub_31486A
                test    eax, eax
                jz      short loc_31306D
                push    esi
                lea     eax, [ebp+arg_0]
                push    eax
                mov     esi, 1D0h
                push    esi
                push    [ebp+arg_4]
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], esi
                jz      short loc_31308F

loc_313066:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+55
                push    0Dh
                call    sub_31411B

loc_31306D:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+49
                call    sub_3140F3
                test    eax, eax
                jg      short loc_31307D
                call    sub_3140F3
                jmp     short loc_31308C
; ---------------------------------------------------------------

loc_31307D:                             ; CODE XREF: sub_312FD8
                call    sub_3140F3
                and     eax, 0FFFFh
                or      eax, 80070000h

loc_31308C:                             ; CODE XREF: sub_312FD8
                mov     [ebp+var_4], eax
```

References

1. Dang, Bruce, and Alexandre Gazet. *Practical Reverse Engineering: X86, X64, ARM, Windows Kernel, Reversing Tools, and Obfuscation*. Print.

1. Blunden, Bill. *The Rootkit Arsenal Escape and Evasion in the Dark Corners of the System, Second Edition*. 2nd ed. Burlington, Mass.: Jones & Bartlett Learning, 2013. Print.