Creating Secure VM

Comparison between Intel and AMD, and one more thing…

Tsukasa Ooi <li@livegrid.org>
Livegrid Incorporated, Lead Analyst

Related Topics

- Virtualization
- Behavior based Detection
- Reverse Engineering

What is "Secure VM"?

- Virtual Machine having…
  - Invisible Breakpoint
  - Direct Memory Lookup/Modification
  - System I/O Interception
  - Instruction/Memory Tracing
- They are useful for security!
  - Direct Memory Lookup to detect rootkits
  - Hooking disk I/Os to find malwares

How to "secure" Virtual Machines

- There are two major ways
  (if CPU-assisted virtualization is used)
- Using Debug Registers (DR0~DR7)
  - DR7.GD bit : Makes guest DRx registers inaccessible
  - VM exit when DR access : Can hook DRx read/write
- Using Page Table Modification
  - PTE.R/W bit : Can hook page write
  - PTE.P bit : Can hook page access (read/write/exec)
  - PTE.NX bit : Can hook page execution

Livegrid™

How to set Breakpoint (Debug Registers)

- Set VM exit when:
  - DRx access
  - #DB (Debug) exception
- Set guest DR7.GD bit
- Set guest breakpoint using guest DRx
  - Useful point : Kernel Functions, System Calls…
- If breakpoint is hit…
  - Do extra validation

Livegrid™

How to set Breakpoint (Page Table)

- Set VM exit when:
  - #PF (Page Fault) exception
- Reset PTE.P bit or PTE.NX bit of page
  - Do not change guest page table, but shadow.
- If Page Fault is occurred
  - Check if breakpoint is hit (can be false-positive)
  - If breakpoint is hit, do extra validation

Comparison (Debug Registers / Page Table)

- Using Debug Registers is simple
  - Easy to implement
- Using Page Table can handle 5 or more breakpoint
  - Breakpoints of Debug Registers are limited to 4
  - If guest OS also use Debug Registers,
    need to emulate using Page Table Modification
- Using Debug Registers is faster
  - No false-positive, low overhead
- Using Page Table is flexible
  - Can intercept “nearly everything”

Livegrid™

Considerations of CPU-assisted virt.

- Cannot intercept important instructions
  - Such as sysenter, sysexit
- Limited interception
  compared than Binary Translation

Comparison between AMD-V/Intel VT-x

- Only AMD-V can intercept IRET instruction
  - Guest OS can workaround host-set breakpoint by set EFLAGS.RF bit (Resume Flag: Suppress breakpoints for single instruction) and execute IRETD instruction
- AMD-V is better if you use Debug Registers
  - No difference if you use Page Table Modification
  - Enough for hooking system calls (not mostly called through IRET instruction)

Second Level Address Translation (SLAT)

- Newer features for virtualization
- Two page tables to translate addresses
  - Guest Logical Addr → Guest Physical Addr
  - Guest Physical Addr → Host Physical Addr
- Intel and AMD both have SLAT extension
  - Intel : Extended Page Tables (EPT)
  - AMD : Nested Paging / Rapid virtualization index (RVI)
- But they got a problem…

Livegrid™

SLAT considerations

- Limited ways for interception
  - REAL (guest) Page Tables are visible
  - Managing two page tables are bad for security-use
  - In some situation, SLAT cannot be enabled
- Debug Registers can be still used

What about difference?

- Intel EPT can make page “execute-only”
  - If Supported by CPU
    - Currently, all CPUs supporting Intel EPT also supports Execute-Only page
  - Stealthy is not enough, but can make invisible chunk
  - Please consider that guest software also can execute invisible chunk with jump instructions.
- Mostly same but Intel EPT is better

Livegrid™

Lg™

THANK YOU?

Livegrid™

One More Thing:

FULL VIRTUALIZATION OF X86 ARCHITECTURE ON X86_64

Livegrid™

Lg™

x86 emulation on x86_64 architecture

- Using Binary Translation…
- They are possible and (nearly) practical!

Why “x86 on x86_64”?

- Architecture is very similar
  - No need to execute normal instruction by 3 instructions (just 1 or 2)
- Have extra memory
  - x86 emulation on x86_64 requires 44GB of virtual memory range, but it is easy to allocate in 64-bit mode.
- Have extra registers!
  - R8~R15 registers can be used for extra emulation context including trace log pointer…

Livegrid™

How to “emulate” x86? (1)

```
PUSH  EBP

MOV   EBP, ESP
MOV   EAX, [EBP+8]
MOV   EDX, [EBP+12]
MOV   EAX, [EAX*4+EDX]

POP   EBP

RET
```

Livegrid™

How to “emulate” x86? (1)

```
PUSH  EBP                             MOV   [RCX+RBP-4], EBX
                                      LEA   EBP, [EBP-4]
MOV   EBP, ESP                        MOV   EBX, EBP
MOV   EAX, [EBP+8]                    MOV   EAX, [RCX+RBX+8]
MOV   EDX, [EBP+12]                   MOV   EDX, [RCX+RBX+12]
MOV   EAX, [EAX*4+EDX]                LEA   R14D, [EAX*4+EDX]
                                      MOV   EAX, [RCX+R14]
POP   EBP                             MOV   EBX, [RCX+RBP]
                                      LEA   EBP, [EBP+4]
RET                                   (Return Intrinsics)
```

How to “emulate” x86? (2)

- Register Remapping
  - ESP → EBP/RBP (Stack in x86_64 is 8-bytes)
  - EBP → EBX/RBX (To reduce cost of stack reference)
- RCX : Base address of 32-bit virtual memory
  - All virtual memory range!
- R14/R14D : Temporary Register
- R15 : Widely Used by Emulation System

Livegrid™

Using Binary Translation…

- Full instruction/memory tracing of x86!
  - Including execution path, memory read/write…
- Very high overhead but it is worth doing.

Livegrid™

What For?

- Detecting some kind of interferences
  - Exploits
  - Hooks
  - (Table/Memory) Modification
- Reverse Engineering
  - Can Trace Everything!
    - Anti-reverse engineering techniques can be detected
  - Protocol Reversing (File, Network…)
  - Algorithm reversing and finding
    - Finding algorithms are useful for anti-DRM.

Livegrid™

How to “emulate” x86 with tracing? (1)

```
PUSH    EBP                         MOV     [RCX+RBP-4], EBX
                                    LEA     EBP, [EBP-4]
MOV     EBP, ESP                    MOV     EBX, EBP
MOV     EAX, [EBP+8]                MOV     EAX, [RCX+RBX+8]
                                    MOV     [R13], EAX
                                    LEA     R13, [R13+4]
MOV     EDX, [EBP+12]               MOV     EDX, [RCX+RBX+12]
                                    MOV     [R13], EDX
                                    LEA     R13, [R13+4]
MOV     EAX, [EAX*4+EDX]            LEA     R14D, [EAX*4+EDX]
                                    MOV     EAX, [RCX+R14]
                                    MOV     [R13], EAX
                                    LEA     R13, [R13+4]
POP     EBP                         MOV     EBX, [RCX+RBP]
                                    MOV     [R13], EBX
                                    LEA     R13, [R13+4]
                                    LEA     EBP, [EBP+4]
RET                                 (Return Intrinsics)
```

Livegrid™

How to “emulate” x86 with tracing? (2)

- R13 : Trace Log Pointer
  - Result of Memory Read
  - Address of Branch
- 1~5 instructions for emulating normal instructions
  - If translation program is “optimized”,
    required number of instructions can be reduced.

Is it practical?

- Nearly.
  - Current Performance Index w/o Disk-write stalls:

	Pentium 4 1.5GHz
≧	Core i7 3.0GHz in emulation
>	Pentium III 1.0GHz

- Not enough for full-system emulation but enough for user-mode-emulation!
- Faster than any tracing methods before (Like debuggers such as OllyDbg, IDA Pro…)

Livegrid™

Limitations / Considerations

- x86 segmentation and addressing
  - x86_64 is constrained “flat memory model”
  - High overhead to emulate segment limit
- VERY FAST Disks are required
  - Fortunately, writing trace log is almost “sequential-write” so RAID using HDD is useful
- Consumes much of resources
  - Requires much of memory

Livegrid™

When will be available?

- Currently, only experimental programs are working…
  - Not “practical” programs…
- I hope… Q2, 2010

Livegrid™

Have any questions?

THANK YOU.