Subverting System Authentication With Context-Aware, Reactive Virtual Machine Introspection

Yangchun Fu, Zhiqiang Lin, Kevin Hamlen

Department of Computer Science
The University of Texas at Dallas

December 12th, 2013

Malicious Virtual Machine Monitor (VMM)

Goal

- Subverting authentication (e.g., `login`) with Context-Aware, Reactive Virtual Machine Introspection(VMI)
- Attackers can gain both fun and profit, by accessing sensitive data in a computer

Malicious Virtual Machine Monitor (VMM)

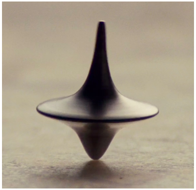
Goal

- Subverting authentication (e.g., `login`) with Context-Aware, Reactive Virtual Machine Introspection(VMI)
- Attackers can gain both fun and profit, by accessing sensitive data in a computer

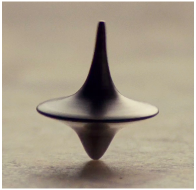
Assumptions

- Assume physical access (lost of laptop, VMs running in a cloud)
- Possible attackers/users
  - Malicious cloud providers (cloud being compromised)
  - Law enforcement (accessing criminal's computer)

Running a machine inside a malicious VMM



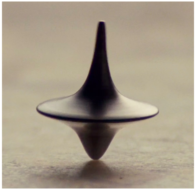
Running a machine inside a malicious VMM



Inception Attack

- Changing your idea using a dream
- Dream can be inside a dream

Running a machine inside a malicious VMM



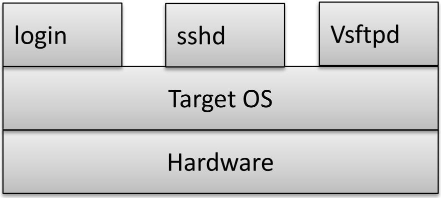
Inception Attack

- Changing your idea using a dream
- Dream can be inside a dream

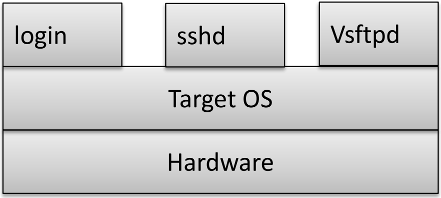
Malicious Virtualization Monitor

- Running a machine inside a **virtual machine**
- We change the guest OS state from the **malicious virtual machine** without the awareness from any inside programs

Traditional computer system structure



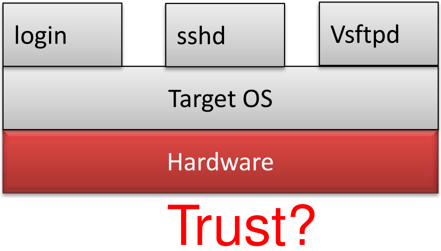
Traditional computer system structure



Authentication protection Mechanism

- Anti-debugging Logic
- Cryptographic Security
- Code Obfuscation
- Self-Checking

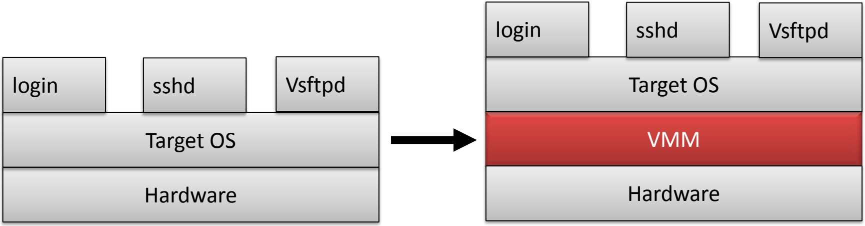
Traditional computer system structure



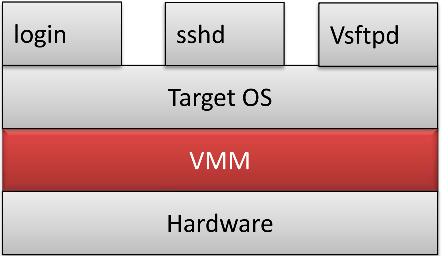
Authentication protection Mechanism

- anti-debugging logic
- cryptographic security
- code obfuscation
- self-checking

Virtualization



Key Idea



Adding a virtualization layer

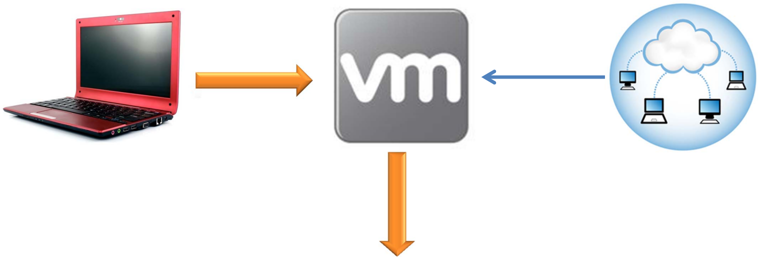
- VMM runs at higher privilege than guest OS
- Great isolation, more stealthy
- A full control of guest OS
- A grand view of the entire state of guest OS.

How it works

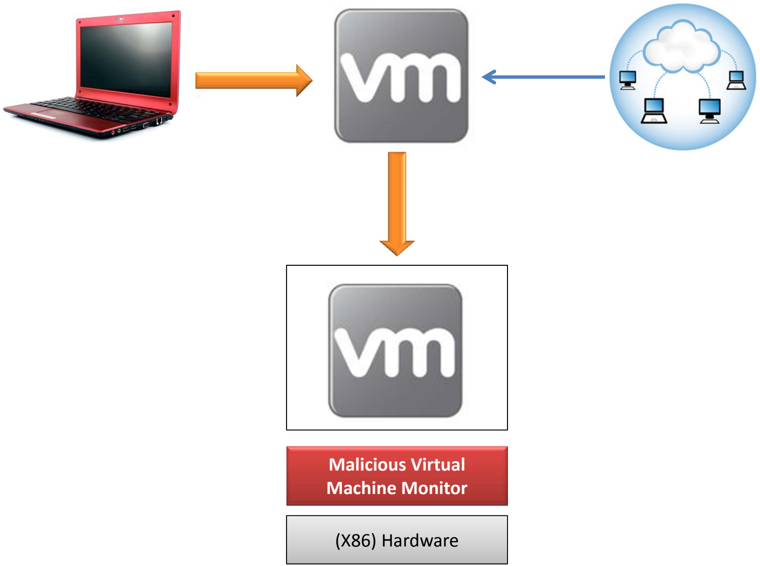
How it works

How it works

How it works

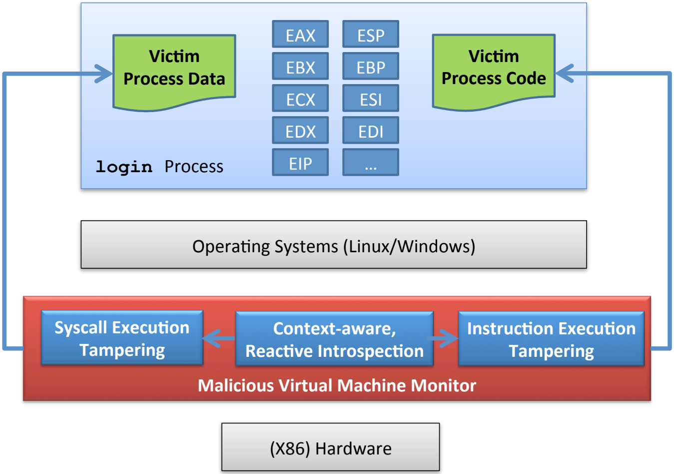


How it works

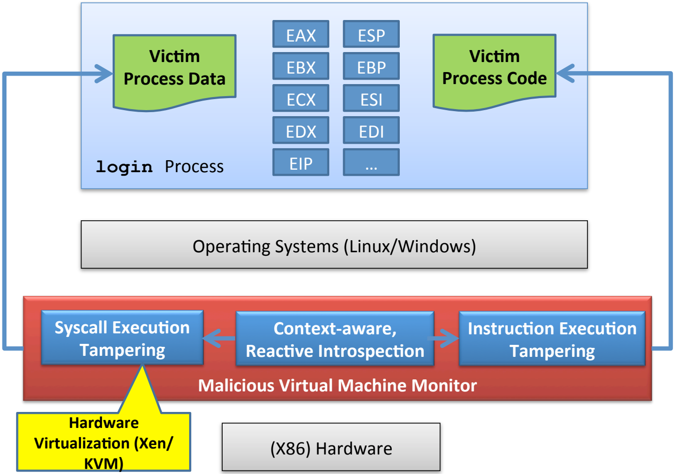


How it works

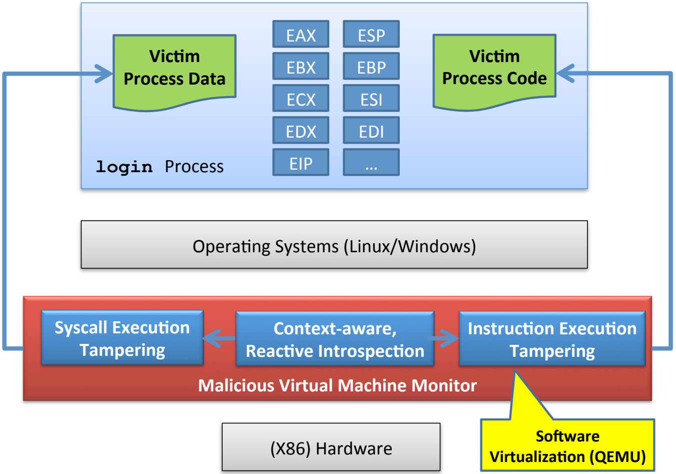
Overview


Victim
Process Data
EAX
ESP
EBX
EBP
ECX
ESI
EDX
EDI
EIP
...
Victim
Process Code
login Process
Operating Systems (Linux/Windows)
Syscall Execution
Tampering
Context-aware,
Reactive Introspection
Instruction Execution
Tampering
Malicious Virtual Machine Monitor
(X86) Hardware

Using Hardware Virtualization



Using Software Virtualization



Working Example: from instructions perspective

```
if (pw_auth (user_passwd, username, reason, (char *) 0) == 0) {
 804a868:     a1 0c 62 05 08            mov    0x805620c,%eax
 804a86d:     c7 44 24 0c 00 00 00      movl   $0x0,0xc(%esp)
 804a874:     00
 804a875:     89 3c 24                  mov    %edi,(%esp)
 804a878:     89 44 24 08               mov    %eax,0x8(%esp)
 804a87c:     a1 48 65 05 08            mov    0x8056548,%eax
 804a881:     89 44 24 04               mov    %eax,0x4(%esp)
 804a885:     e8 86 87 00 00            call   8053010<pw_auth>
 804a88a:     85 c0                     test   %eax,%eax
 804a88c:     0f 84 6d fd ff ff         je     804a5ff<main+0x64f>
            goto auth_ok;
}
```

Figure : Binary Code Snippet of the `login` Program.

Insight-I

Instruction Execution Tampering

- Tampering with Instruction Opcode
  - `804a88c:0f 84 (je)` → `0f 85 (jne)`
- Tampering with Instruction Operand
  - `804a88a:test %eax,%eax` → Tampering w/ `eax/EFLAGS`
- Tampering with both Opcode and Operand
  - `804a885:call 8053010<pw_auth>` → `mov $0,%eax`

Working Example: from system call perspective

```
  1 execve("/bin/login", ["login"], [/* 16 vars */]) = 0
  2 uname({sys="Linux", node="ubuntu", ...}) = 0
...
409 open("/etc/passwd", O_RDONLY)                  = 4
410 fcntl64(4, F_GETFD)                             = 0
411 fcntl64(4, F_SETFD, FD_CLOEXEC)                 = 0
412 _llseek(4, 0, [0], SEEK_CUR)                    = 0
413 fstat64(4, {st_mode=S_IFREG|0644, st_size=952, ...}) = 0
414 mmap2(NULL, 952, PROT_READ, MAP_SHARED, 4, 0) = 0x4021a000
415 _llseek(4, 952, [952], SEEK_SET)                = 0
416 munmap(0x4021a000, 952)                         = 0
417 close(4)                                        = 0
418 open("/etc/shadow", O_RDONLY)                  = 4
419 fcntl64(4, F_GETFD)                             = 0
420 fcntl64(4, F_SETFD, FD_CLOEXEC)                 = 0
421 _llseek(4, 0, [0], SEEK_CUR)                    = 0
422 fstat64(4, {st_mode=S_IFREG|0640, st_size=657, ...}) = 0
423 mmap2(NULL, 657, PROT_READ, MAP_SHARED, 4, 0) = 0x4021a000
424 _llseek(4, 657, [657], SEEK_SET)                = 0
425 munmap(0x4021a000, 657)                         = 0
426 close(4)                                        = 0
...
```

Figure : System Call Trace Snippet of the `login` Program.

Insight-II

System Call Execution Tampering

- Tampering with Disk-IO Syscall
  - Replacing `/etc/shadow` file when it loads to the memory. Essentially a man-in-the-middle Attack. We can hijack the file `open` syscall and provide an attacker controlled password file
- Tampering with Memory-Map Syscall
  - Tampering with `mmap2` syscall by replacing the memory contents mapped by this syscall (immediately after it finishes) with the password hash values we control.

Insight-II

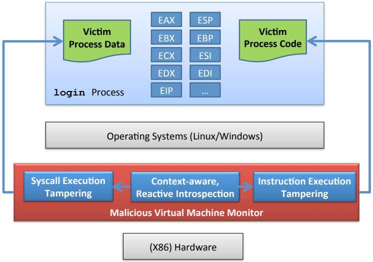
System Call Execution Tampering

- Tampering with Disk-IO Syscall
  - Replacing `/etc/shadow` file when it loads to the memory. Essentially a man-in-the-middle Attack. We can hijack the file `open` syscall and provide an attacker controlled password file
- Tampering with Memory-Map Syscall
  - Tampering with `mmap2` syscall by replacing the memory contents mapped by this syscall (immediately after it finishes) with the password hash values we control.

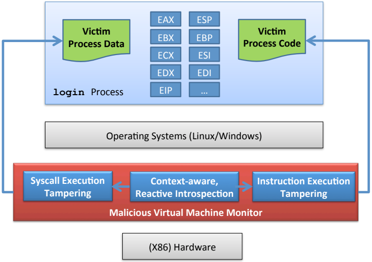
Advantages

- Transparent, can work for many other `login` types of programs
- No binary code reverse engineering

Challenges


Victim
Process Data
EAX
ESP
EBX
EBP
ECX
ESI
EDX
EDI
EIP
...
Victim
Process Code
login Process
Operating Systems (Linux/Windows)
Syscall Execution
Tampering
Context-aware,
Reactive Introspection
Instruction Execution
Tampering
Malicious Virtual Machine Monitor
(X86) Hardware

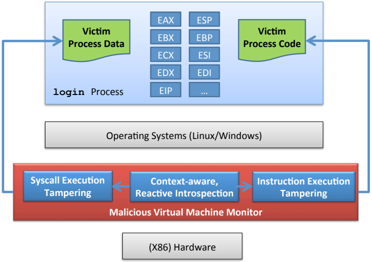
Challenges



Identifying the "dreaming" context at the VMM layer

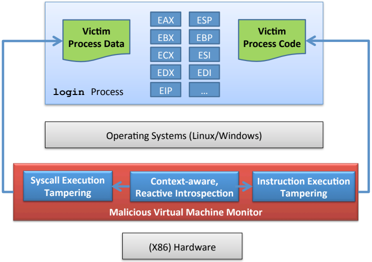
- (**C1**) a particular process execution;
- (**C2**) a particular syscall in **C1**;
- (**C3**) a particular instruction in **C1**;
- (**C4**) a particular instruction in **C1** under a particular call stack.

Solutions



Victim
Process Data
EAX
EBX
ECX
EDX
EIP
ESP
EBP
ESI
EDI
...
Victim
Process Code
login Process
Operating Systems (Linux/Windows)
Syscall Execution
Tampering
Context-aware,
Reactive Introspection
Instruction Execution
Tampering
Malicious Virtual Machine Monitor
(X86) Hardware

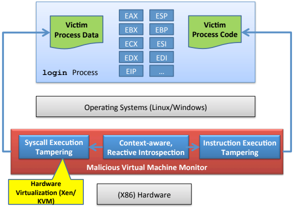
Solutions



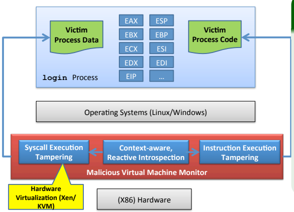
Context-Aware, reactive introspection

- **Introspection**: a variant of Virtual Machine Introspection [Garfinkel et al, NDSS'03]
- **Reactive**: not a passive, read-only introspection, it is reactive
- **Context-Aware**: context ranges from **C1** to **C4**

Solutions: Designing with Xen/KVM (SYSVMI)



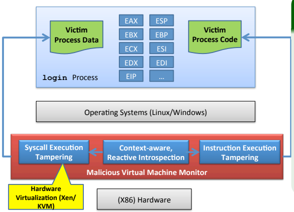
Solutions: Designing with Xen/KVM (SYSVMI)



Execution Context Identification

- (**C1**) – process context: CR3 and code hash of `login`
- (**C2**) – syscall in **C1**: `sysenter/sysret,int 0x80/iret`

Solutions: Designing with Xen/KVM (SYSVMI)



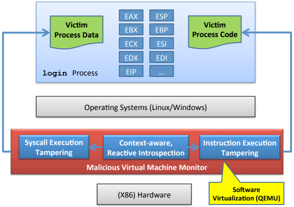
Execution Context Identification

- (**C1**) – process context: CR3 and code hash of `login`
- (**C2**) – syscall in **C1**: `sysenter/sysret,int 0x80/iret`

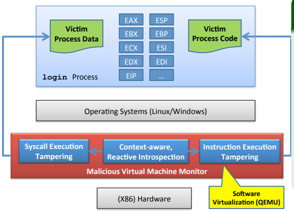
Attack Strategies

- **A1**: Tampering with Instruction Code.
- **A2**: Tampering with Syscall Arguments and Return Values
- **A3**: Tampering with Syscall Produced Data
- **A4**: Using IO Virtualization

Solutions: Designing with QEMU (INSTVMI)



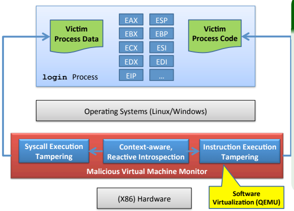
Solutions: Designing with QEMU (INSTVMI)



Execution Context Identification

- (**C3**) – instruction execution: Program Counter (PC)
- (**C4**) – call stack: instrumenting `call/ret`

Solutions: Designing with QEMU (INSTVMI)



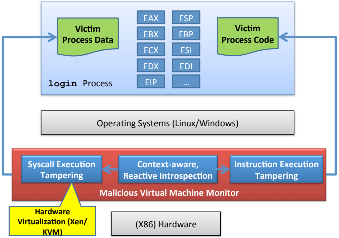
Execution Context Identification

- (**C3**) – instruction execution: Program Counter (PC)
- (**C4**) – call stack: instrumenting `call/ret`

Attack Strategies

- **A5**: Tampering with Instruction Code at PC Level
- **A6**: Tampering with Instruction Operand
- **A7**: Tampering with Function Call Arguments and Return Values

Implementation

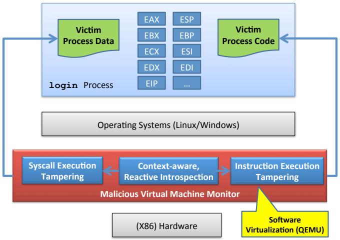


SYSVMI: Using Xen-4.12

Malicious-VMM w/ Xen-4.12	C1∼C2	A1	A2	A3	A4	Total
	1,748	17	10	75	45	1,895

- Implementing **A1** to **A4** with only 1,895 LOC in total (a very low cost for attacker).

Implementation



INSTVMI: Using QEMU-1.01

Malicious-VMM w/	C1 $\sim$ C4	A5	A6	A7	Total
QEMU-1.01	3,513	35	34	25	3,607

- INSTVMI$_a$ ported the SYSVMI implementation (**C1** and **C2**, and **A1** – **A4**) to a most recent QEMU-1.01
- INSTVMI$_b$ implemented the new attacks unique to the software virtualization (**A5** – **A7**) with fine-grained execution context identification (**C3** and **C4**)

Overall Result

Target	SYSVMI			$INSTVMI_a$			$INSTVMI_b$		
	A1	A2,A3	A4	A1	A2,A3	A4	A5	A6	A7
`login`	✓	✓	✓	✓	✓	✓	✓	✓	✓
`sshd`	✓	✓	✓	✓	✓	✓	✓	✓	✓
`vsftpd`	✓	✓	✓	✓	✓	✓	✓	✓	✓
`telnetd`	✓	✓	✓	✓	✓	✓	✓	✓	✓

Table : Effectiveness of our virtual machine inception attack against the authentication program. Each ✓symbols denotes a successful way of incepting the victim software.

Performance Overhead

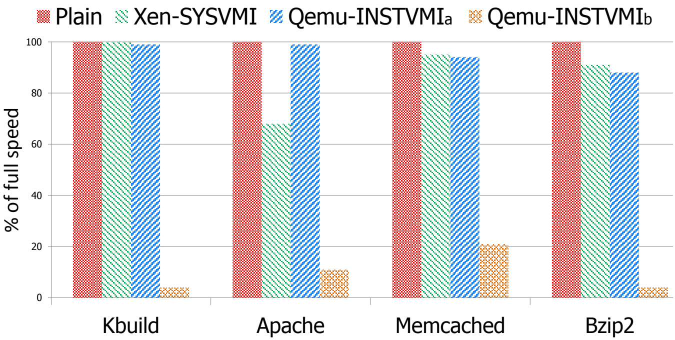


Figure : Macro-benchmark Evaluation of the Performance Overhead of Our VMI

Performance Overhead

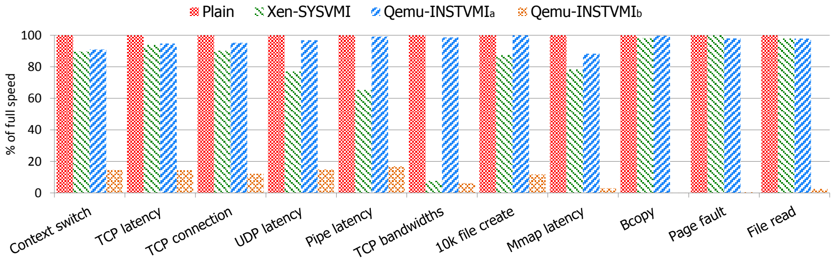
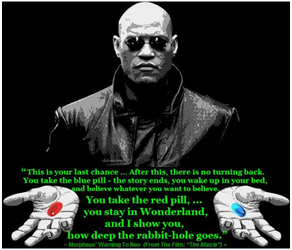


Figure : Micro-benchmark Evaluation of the Performance Overhead of Our VMI

Hardware Virtualization Rootkits

Blue Pill

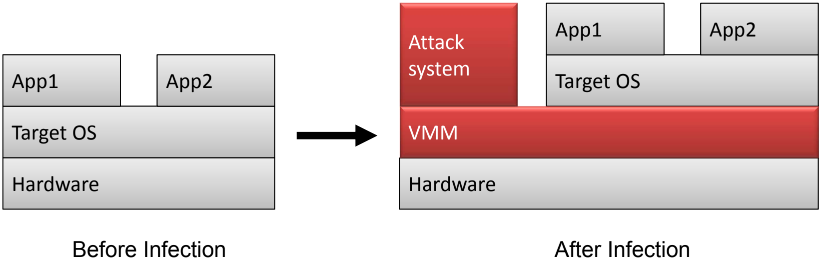
- The codename for a rootkit based on x86 virtualization. [J. Rutkowska, Blackhat'06]
- Trapping a running instance of the OS by starting a thin hypervisor
- Vitriol [D. Zov, Blackhat'06].



Key Differences

- Thin vs. Thick Hypervisor
- Our attack explores the relative freedom that a malicious virtualization owner has to easily launch stealthy attacks against the virtualized software

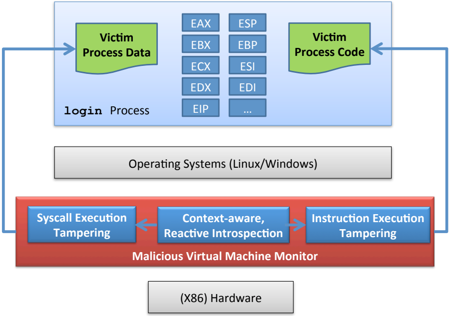
Subvert, SubXen



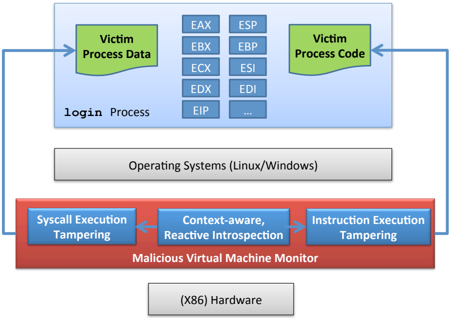
Key Differences

- Subvert aims to infect other's virtualization (to be thin to avoid large footprints)
- Subvert does not bridge the semantic gap, whereas our attack involves a context-aware approach to infer the guest OS semantics.

Summary

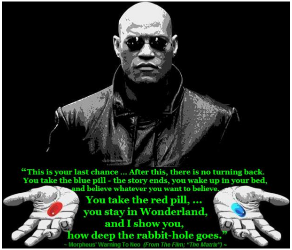

Victim
Process Data
EAX
ESP
EBX
EBP
ECX
ESI
EDX
EDI
EIP
...
Victim
Process Code
login Process
Operating Systems (Linux/Windows)
Syscall Execution Tampering
Context-aware, Reactive Introspection
Instruction Execution Tampering
Malicious Virtual Machine Monitor
(X86) Hardware

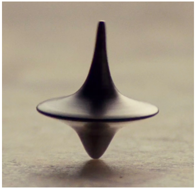
Summary



- We design and implement a context-aware, reactive virtual machine to break authentication mechanism.
- Our result indicates that the approach is practical against real-world authentication programs.
- It is useful for both malicious attack and forensics analysis of virtualized systems and software.

Take Away





This attack is so easy

- Small lines of code
- Can be done by anyone, given physical access

Rethink the `login` design

- `login` can be executed in a “dreaming” context now
- We have to redesign `login` to counter against, or at least raise the bar for this attack

Thank you

Questions?

To contact us: {yangchun.fu,zhiqiang.lin,hamlen}@utdallas.edu