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 12\textsuperscript{th}, 2013
1. Background
2. Detailed Design
3. Implementation
4. Evaluation
5. Related Work
6. Summary
Traditional computer system structure

- **Hardware**
- **Target OS**
- **login**
- **sshd**
- **Vsftpd**

**Mechanisms**:
- Authentication protection
- Anti-debugging logic
- Cryptographic security
- Code obfuscation
- Self-checking

**Trust?**
## Traditional computer system structure

<table>
<thead>
<tr>
<th></th>
<th>login</th>
<th>sshd</th>
<th>Vsftpd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target OS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hardware</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Authentication protection Mechanism
- Anti-debugging Logic
- Cryptographic Security
- Code Obfuscation
- Self-Checking
Traditional computer system structure

- **Trust?**

- **Authentication protection Mechanism**
  - anti-debugging logic
  - cryptographic security
  - code obfuscation
  - self-checking
Virtualization

```
Hardware
Target OS
login
VMM
VMM
Target OS
login
sshd
Vsftpd
VMM
Target OS
login
sshd
Vsftpd
```

- **Hardware**
- **Target OS**
- **VMM**
- **login**
- **sshd**
- **Vsftpd**
Motivations

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.
Malicious VMM

Goal

- Subverting authentication (e.g., login) with Context-Aware, Reactive Virtual Machine Introspection (VMI)
- Attackers can gain fun and profit: Accessing sensitive data in a computer (e.g., a laptop, or a VM)
Malicious VMM

Goal

- Subverting authentication (e.g., login) with Context-Aware, Reactive Virtual Machine Introspection (VMI)
- Attackers can gain fun and profit: Accessing sensitive data in a computer (e.g., a laptop, or a VM)

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, note that a physical machine can be virtualized)
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 insider programs
How it works
How it works
How it works
How it works
How it works

(X86) Hardware

Malicious Virtual Machine Monitor

Virtual Machine Monitor

Hardware
How it works

Malicious Virtual Machine Monitor

(X86) Hardware
Overview

Victim Process Code

Victim Process Data

Operating Systems (Linux/Windows)

Malicious Virtual Machine Monitor

Syscall Execution Tampering

Context-aware, Reactive Introspection

Instruction Execution Tampering

(X86) Hardware

login Process

EAX
EBX
ECX
EDX
ESP
EBP
ESI
EDI
EIP
...
Using Hardware Virtualization

Operating Systems (Linux/Windows)

Malicious Virtual Machine Monitor

Hardware Virtualization (Xen/KVM)

(X86) Hardware

Syscall Execution Tampering

Context-aware, Reactive Introspection

Instruction Execution Tampering

Victim Process Data

Victim Process Code

login Process

EAX
EBX
ECX
EDX
ESP
EBP
ESI
EDI
EIP
...
Using Software Virtualization

Malicious Virtual Machine Monitor

Operating Systems (Linux/Windows)

Syscall Execution Tampering
Context-aware, Reactive Introspection
Instruction Execution Tampering

Victim Process Data
Victim Process Code

login Process

Software Virtualization (QEMU)

(X86) Hardware

EAX
EBX
ECX
EDX
ESP
EBP
ESI
EDI
EIP
...
if (pw_auth (user_passwd, username, reason, (char *) 0) == 0) {
  mov 0x805620c,%eax
  movl $0x0,0xc(%esp)
  mov %edi,(%esp)
  mov %eax,0x8(%esp)
  mov 0x8056548,%eax
  movl 0x8056548,%eax
  call 8053010<pw_auth>
  test %eax,%eax
  je 804a5ff<main+0x64f>
  goto auth_ok;
}

Figure : Binary Code Snippet of the login Program.
Insight-I

Instruction Execution Tampering

- Tampering with Instruction Opcode
  - \texttt{0x804a88c:0f 84 (je) \rightarrow 0f 85 (jne)}

- Tampering with Instruction Operand
  - \texttt{0x804a88a:test \%eax,\%eax \rightarrow Tampering w/ eax/EFLAGS}

- Tampering with both Opcode and Operand
  - \texttt{0x804a885:call 8053010 \rightarrow mov \$0,\%eax}
Working Example: from system call perspective

1. execve("/bin/login", ["login"], [/* 16 vars */]) = 0
2. uname({sys="Linux", node="ubuntu", ...}) = 0

... 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.
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

Identifying the "dreaming" context at the VMM layer:

1. A particular process execution (C1).
2. A particular syscall in C1 (C2).
3. A particular instruction in C1 (C3).
4. A particular instruction in C1 under a particular call stack (C4).

Operating Systems (Linux/Windows)

Malicious Virtual Machine Monitor

(X86) Hardware

Syscall Execution Tampering → Context-aware, Reactive Introspection → Instruction Execution Tampering
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

Context-Aware, reactive 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 \( C_1 \) to \( C_4 \)
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 \( C_1 \) to \( C_4 \)
Solutions: Designing with Xen/KVM (SYSVMI)

- Execution Context Identification
  - Context C1: process context
    - CR3 and code hash of login
  - Context 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

- Diagram:
  - Victim Process Data
  - Victim Process Code
  - Operating Systems (Linux/Windows)
  - Syscall Execution Tampering
  - Context-aware, Reactive Introspection
  - Instruction Execution Tampering
  - Malicious Virtual Machine Monitor
  - Hardware Virtualization (Xen/KVM)
  - (X86) Hardware
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`
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 \texttt{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
**SYSVMI: Using Xen-4.12**

<table>
<thead>
<tr>
<th>Malicious-VMM w/ Xen-4.12</th>
<th>C1</th>
<th>C2</th>
<th>A1</th>
<th>A2</th>
<th>A3</th>
<th>A4</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1,748</td>
<td>17</td>
<td>10</td>
<td>75</td>
<td>45</td>
<td></td>
<td>1,895</td>
</tr>
</tbody>
</table>

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

---

**INSTVMI: Using QEMU-1.01**

<table>
<thead>
<tr>
<th>Malicious-VMM w/ QEMU-1.01</th>
<th>C1  ∼ C4</th>
<th>A5</th>
<th>A6</th>
<th>A7</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3,513</td>
<td>35</td>
<td>34</td>
<td>25</td>
<td>3,607</td>
</tr>
</tbody>
</table>

- 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

<table>
<thead>
<tr>
<th>Target</th>
<th>SYSVMI</th>
<th>INSTVMI\textsubscript{a}</th>
<th>INSTVMI\textsubscript{b}</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A1</td>
<td>A1</td>
<td>A5</td>
</tr>
<tr>
<td></td>
<td>A2,A3</td>
<td>A2,A3</td>
<td>A6</td>
</tr>
<tr>
<td></td>
<td>A4</td>
<td>A4</td>
<td>A7</td>
</tr>
<tr>
<td>login</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>sshd</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>vsftpd</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>telnetd</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

**Table**: Effectiveness of our virtual machine inception attack against the authentication program. Each ✓ symbols denotes a successful way of incepting the victim software.
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 and virtualizing the rest of the machine under it.
- Vitriol [D. Zov, Blackhat’06] is also a hardware virtualization rootkit

Key Differences

- Thin vs. Thick Hypervisor
- Bluepill aims to compromise other’s virtualization
- Our attack owns the virtualization and has rich features
Subvert, SubXen

Before Infection

App1
Target OS
Hardware

App2

After Infection

VMM
Target OS
App1
App2

Key Differences

- Subvert [King et al., Oakland'06], a virtualization rootkit
- Thin vs. Thick Hypervisor
- Subvert also aims to infect other’s virtualization (to be thin to avoid large footprints)
- Our attack owns the virtualization and has rich features
Outline

1. Background
2. Detailed Design
3. Implementation
4. Evaluation
5. Related Work
6. 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.
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.
Questions?

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