Virtual Ghost:
Protecting Applications from Hostile Operating Systems

John Criswell, Nathan Dautenhahn, and Vikram Adve
New Job
New Job
Do You Trust Your Operating System?
Do You Trust Your Operating System?
Do You Trust Your Operating System?
Do You Trust Your Operating System?
Do You Trust Your Operating System?

Online Shopping!

Voting Machines!

Filing Taxes!

Medical Data!
Do You Trust Your Operating System?

Online Shopping!
Filing Taxes!
Voting Machines!
National Security!
Medical Data!
Commodity Operating Systems Are Vulnerable!

<table>
<thead>
<tr>
<th>Vulnerability</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buffer Overflows</td>
<td>BugTraq ID 12911, 13589, 13207, 13225, 12295</td>
</tr>
<tr>
<td>Integer Overflows</td>
<td>BugTraq ID 10179, 63707</td>
</tr>
<tr>
<td>Information Leaks</td>
<td>BugTraq ID 8831, 64677, 64746, 64742, 62405</td>
</tr>
<tr>
<td>Kernel-level Malware</td>
<td>Adore rootkit</td>
</tr>
</tbody>
</table>
If the operating system kernel is exploited, all security guarantees are \textit{null} and \textit{void}.
Virtual Ghost Contributions

- *Protects* application data *confidentiality* and *integrity*
- Uses *compiler techniques* thanks to *LLVM*
- *Same privilege level* as kernel
- *Faster* than hypervisor-based approaches
Outline

• Motivation

• Design

• Results

• Future Work
Goal: Application That *Protects* Itself from OS

Required Features

1. Private data and code
2. Incorruptible control flow
3. Reliable encryption key delivery
Challenges
Challenges

1. Processor lets privileged software access all memory
Challenges

1. Processor lets privileged software access all memory

2. Operating System *must* manipulate application state
   - Process and thread creation
   - Executing new programs (exec() family of system calls)
   - Signal handler dispatch
Virtual Ghost

• OS compiled to virtual instruction set
  • Designed to be easy to analyze and instrument
  • Low-level instructions (SVA-OS) replace assembly code
• Translate ahead-of-time, boot-time, or run-time
Virtual Ghost

- OS compiled to virtual instruction set
  - Designed to be easy to analyze and instrument
  - Low-level instructions (SVA-OS) replace assembly code
- Translate ahead-of-time, boot-time, or run-time
Virtual Instruction Set

- **SVA-Core: Compiler Instrumentation**
  - Based on LLVM IR: Typed, Explicit SSA form
  - Sophisticated compiler analysis and instrumentation

- **SVA-OS: Virtual Ghost Runtime**
  - OS-neutral instructions to support a commodity OS
  - Encapsulates & controls hardware and state manipulation
  - Implemented as a run-time library linked into kernel
Kernel cannot access User-Space Memory

Kernel cannot access Ghost Memory

Kernel cannot access Virtual Ghost VM Memory

Kernel cannot access Kernel Memory

Private Data and Code
Ghost Memory Instrumentation

- Software Fault Isolation
  - Protects Ghost and VM Memory
  - Avoids TLB flush
- Control-Flow Integrity
  - Prevents instrumentation bypass
  - Provides kernel protection
Software Fault Isolation Instrumentation

Ghost Memory

0xffffffff0000000000 – 0xffffffff8000000000

mask = ((((p >> 32) == 0xffffffff00 ? 0x8000000000 : 0);)
p |= mask;
store v, *p;
Control-Flow Integrity Instrumentation

- Insert NOP labels at target addresses
  - Function entry
  - Call sites
- Instrument all computed jumps
  - Bitmask to force pointer into kernel code
  - Check label at target of computed jump

1. Zeng, Tan, and Morrisett, *Combining Control-flow Integrity and Static Analysis for Efficient and Validated Data Sandboxing*, CCS 2011
Secure Application Control Flow

- Program state in VM Memory
  - OS cannot modify directly
- SVA-OS vets/performs changes
  - Signal handler dispatch
  - Thread creation
  - Exec() system calls
Secure Application Encryption Keys

Executable

- Application Code
- Application Key Pair

Process

- Code Segment
- Memory

Virtual Ghost
Kernel Injects Wrong Key

**Executable**
- Application Code
- Kernel-Inserted Key Pair

**Process**
- Code Segment
- Ghost Memory

Virtual Ghost
Kernel Replaces Code

Executable

- Kernel-Inserted Code
- Application Key Pair

Process

- Code Segment
- Ghost Memory

Virtual Ghost
Secure Application Encryption Keys

**Executable**

- $E_{VG}(\text{Application Code})$
- $E_{VG}(\text{Application Key Pair})$
- $E_{VG}(\text{Hash of Executable})$

**Process**

- Code Segment
- Ghost Memory

**Virtual Ghost**
Secure Application Encryption Keys

Executable

$E_{VG}(\text{Application Code})$

$E_{VG}(\text{Application Key Pair})$

$E_{VG}(\text{Hash of Executable})$

Process

Code Segment

Ghost Memory

Application Key Pair

Virtual Ghost
Outline

• Introduction

• Design

• Results

• Future Work
Implementation

• Developed a x86_64 64-bit implementation of Virtual Ghost

• Ported FreeBSD 9.0 to Virtual Ghost
  • FreeBSD compiles with LLVM out of the box

• Modified OpenSSH applications to use ghosting
  • ssh client
  • ssh-agent key-chain server
  • ssh-add utility
Kernel Malware Attack

**Trick Application into Putting Data into the Clear**

- Install signal handler to malicious code in application
- Malicious code copies data to traditional memory

```
memcpy(g, t);
write(fd, t, 10);
```
Kernel Malware Attack

**Trick Application into Putting Data into the Clear**

- Install signal handler to malicious code in application
- Malicious code copies data to traditional memory

```
memcpy(g, t);
write(fd, t, 10);
```

**Malicious Data Flow**

- Kernel
- Malware Driver
- ssh-agent
- Traditional Memory
- ssh-agent
- Ghost Memory

23
LMBench Execution Time Normalized to Native

![Graph showing execution time normalized to native for various benchmarks with InkTag and Virtual Ghost.]  

InkTag[1]  
Virtual Ghost  

Imbench Benchmark

Web Server Performance for thttpd

- ApacheBench: 100 clients, 100,000 requests
- Performance overhead negligible
Unmodified SSH Server Performance

- 23% reduction of bandwidth on average
- 45% reduction in worst case
Ghosting SSH Client Performance

- 5% reduction in worst case

<table>
<thead>
<tr>
<th>File Size (KB)</th>
<th>Original ssh</th>
<th>Ghosting ssh</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>16</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>64</td>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td>256</td>
<td>256</td>
<td>256</td>
</tr>
<tr>
<td>1024</td>
<td>1024</td>
<td>1024</td>
</tr>
<tr>
<td>4096</td>
<td>4096</td>
<td>4096</td>
</tr>
<tr>
<td>16384</td>
<td>16384</td>
<td>16384</td>
</tr>
<tr>
<td>65536</td>
<td>65536</td>
<td>65536</td>
</tr>
<tr>
<td>262144</td>
<td>262144</td>
<td>262144</td>
</tr>
<tr>
<td>1048576</td>
<td>1048576</td>
<td>1048576</td>
</tr>
</tbody>
</table>
Outline

• Introduction

• Design

• Results

• Future Work
Future Work

• Improved performance
  • Advanced optimization (e.g., type safe check optimization)

• Cryptographic protocols for preventing OS attacks
  • Prevent replay attacks

• Compiler transforms to use Virtual Ghost features
Started Open-Source Release

- LLVM Compiler Extensions
- Virtual Ghost Run-time Library
Summary

• Virtual Ghost allows applications to protect themselves from an OS

• Uses compiler instrumentation
  • Keeps higher processor privilege levels free

• Faster than hypervisor-based approaches

See what we do at http://sva.cs.illinois.edu!