Hardware-Assisted Runtime Defenses Against Malware Attacks

Adrian Tang

PhD Candidacy Exam
Apr 9, 2015
Attacks are getting complex

Multi-layer
Multi-stage
Evolving

Has Equation Group hacked your hard drives? You won’t be able to tell.

Infection can survive formatting and reinstalling the operating system

 RELATED
Reformatting won’t rename invisible and persistent infections hard...

REGIN EXPLOIT STAGES

ROP → JIT-ROP → Blind-ROP → ...

http://www.extremetech.com/computing/200898-windows-pcs-vulnerable-to-stuxnet-attack-five-years-after-patches
http://www.networkworld.com/article/2885814/security0/has-equation-group-hacked-your-hard-drives-you-won-t-be-able-to-tell.html

... and more ...
In a Nutshell

Key runtime defenses that harness hardware-oriented assistive technologies in commodity processors to detect malware attacks

I. Detection Technique Taxonomy
   The dawn of attack detection
   3 dimensions of the taxonomy

II. On Virtual Machine Introspection (VMI)
   Properties of VMM leveraged for introspection
   Design patterns

III. Defenses Across The Stack
   Hardware-assisted detection
   Benefits of capitalizing hardware assistance
I. **DETECTION TECHNIQUE TAXONOMY**
   - The dawn of attack detection
   - 3 dimensions of the taxonomy

II. **ON VIRTUAL MACHINE INTROSPECTION (VMI)**
   - Properties of VMM leveraged for introspection
   - Design patterns

III. **DEFENSES ACROSS THE STACK**
   - Hardware-assisted detection
   - Benefits of capitalizing hardware assistance
The dawn of Intrusion/Attack Detection

Need to review audit trails automatically to facilitate monitoring

[Anderson, 1980]

Intrusion detection expert system (IDES) – Amongst the first IDS built to model normal behavior of users as collective “profiles”, using deviations to detect unauthorized use

[Denning, 1987]
**Taxonomy // 3 Dimensions**

**Abstraction Layer**
- Where to get the features

**Modeling Approach**
- What to model
  - Known Goods → Anomaly-based [Forrest et al., 1994]
  - Known Bads → Signature-based [Anderson et al., 1995]

**Analysis Type**
- How to get the features
  - Static → Syntactic
  - Dynamic / Runtime behavior → Semantic
    - VMI [Garfinkel et al., 2003]
    - Emulation [Polychronakis et al., 2006]
## Taxonomy // Across the stack

<table>
<thead>
<tr>
<th><strong>NETWORK</strong></th>
<th><strong>APPLICATION</strong></th>
<th><strong>OS</strong></th>
<th><strong>SUB-OS</strong></th>
<th><strong>μARCH</strong></th>
<th><strong>PHYSICAL</strong></th>
<th><strong>HARDWARE ASSISTANCE</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>[McCune et al., 2010] [Wang et al., 2010] [Azab et al., 2014]</td>
<td>HW Perf Ctrs</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>[Demme et al., 2013] [Tang et al., 2014] [Wicherski, 2013]</td>
<td>LBR (Last Branch)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>[Kim et al., 2008] [Hoffmann et al., 2013] [Clark et al., 2013]</td>
<td>Intel VT-x</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>[Pappas et al., 2013] [Xia et al., 2012]</td>
<td>AMD SVM</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>[Pfoh et al., 2011] [Payne et al., 2008] [Wang and Karri, 2014] [Liu et al., 2014]</td>
<td>Intel TSX</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>[Vasiliadis et al., 2008] [Snow et al., 2011]</td>
<td>TPM</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ARM Trustzone</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Power sensors</td>
</tr>
</tbody>
</table>
I. DETECTION TECHNIQUE TAXONOMY
   The dawn of attack detection
   3 dimensions of the taxonomy

II. ON VIRTUAL MACHINE INTROSPECTION (VMI)
   Properties of VMM leveraged for introspection
   Design patterns

III. DEFENSES ACROSS THE STACK
   Hardware-assisted detection
   Benefits of capitalizing hardware assistance
VMI // The good and the bad

Monitoring from “outside” at the VMM level has its perks

- Isolation ➔ Attack resistance
- Inspection ➔ Can “see” everything
- Interposition ➔ Can intercept target code

[Garfinkel and Rosenblum, 2003]

These benefits come with caveats

- Semantic gap ➔ How to interpret states
  ➔ Deriving context-specific meaning to collections of states

3 design patterns for interpreting state information

- In-band, Out-of-band, Derivative

VMI // Hardware-rooted derivative

Derivative view-generation is most robust to evasion

- Using state information derived at the lowest level possible
- Information stipulated by architecture specifications

Example of hardware-binding technique

- Deriving the Syscall table from IDT Register of CPU

Where are we

I. Detection Technique Taxonomy
   The dawn of attack detection
   3 dimensions of the taxonomy

II. On Virtual Machine Introspection (VMI)
    Properties of VMM leveraged for introspection
    Design patterns

III. Defenses Across the Stack
    Hardware-assisted detection
    Benefits of capitalizing hardware assistance
Network // Pattern-matching and Emulation

Boost performance with general purpose accelerators

- Pattern-matching is computationally intensive (75% time)
- Offload pattern-matching to GPUs
- ~2X speedup over traditional Snort

Substantial performance boost & higher-fidelity emulation

- Software-based CPU emulation has limitations
- Harness hardware virtualization (Intel VT-x) to directly execute on native CPU within guest VM
- Throughput ~10x (350Mbps), over Nemu

[Vasiliadis et al., 2008]

[Snow et al., 2011]
Achieve transparent and efficient runtime branch tracing

I. Branch Trace Store (BTS) – memory buffer
   • Store all indirect branch targets collected during training
   • Investigate any branch targets not seen in training
   • ~6.1% runtime overhead

II. Last Branch Recording (LBR) – 16 pairs of registers
   • Before selected syscalls, identify CFI violations
     • RETs that are not preceded by CALLs
     • ROP gadget: < 21 contiguous instructions ending in RETs
     • Sequence of > 8 ROP gadgets
   • ~1% runtime overhead

[Xia et al., 2012]

[Pappas et al., 2013]
# Syscall tracing via VMI

## Transparent, isolated, efficient runtime syscalls monitoring

<table>
<thead>
<tr>
<th>System</th>
<th>Generality</th>
<th>Transparency</th>
<th>Guest Portability</th>
<th>Isolation</th>
<th>Overhead</th>
<th>Technique</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software (auditd)</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td>~75 - 150%</td>
<td>ptrace, systrace&lt;br&gt;Kernel patch to record audit logs.</td>
</tr>
<tr>
<td>Lares [Payne et al., 2008]</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>~63%</td>
<td>Implant hooks protected by hypervisor.</td>
</tr>
<tr>
<td>Ether</td>
<td>✓ ✓ ✓ ✓ ✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>~34.7%</td>
<td>Induce #PF on syscalls by directing key MSRs and IDT entries to NP pg. x64 not supported.</td>
</tr>
<tr>
<td>Nitro [Pfoh et al., 2011]</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>~14.1%</td>
<td>Induce #GP by virtualizing IDT with reduced bounds, and nulling SYSENTER_CS_MSR. Induce #UD for x64 syscall using EFER.SCE.</td>
</tr>
</tbody>
</table>


OS // Monitoring for rootkit actions

Efficiently detect “additional” hooking code added by rootkits

- Collect architectural events using HPC at the VMM layer
- Enable/disable HPC counting using VMI syscall tracing
- INS counts from selected infected syscalls > 10x the baseline

[Wang and Karri, 2014]

Efficiently detect writes to critical kernel structures

- Harness hardware transactional memory (Intel TSX)
- Monitor critical kernel data at cacheline granularity by adding the addresses into the TSX read sets
- Writes to these addresses trigger transactional abort & VMI checking

[Liu et al., 2014]

I. Intel System Management Mode (SMM)
   • Reside attestation code in SMM layer (below hypervisor) for isolation
   • Rely on PCI NIC device (driver in SMM) to read physical memory
   • Monitor control registers and static code section of hypervisor

II. ARM TrustZone
   • Strips the target OS of its capability to do selected sensitive tasks
   • These capabilities are mediated by monitor code in Secure World
   • Without these capabilities, rootkits cannot modify kernel code in memory
   • Practical \(\rightarrow\) Deployed on Samsung phones

[Wang et al., 2010]

Bootstrap trust from TPM and efficient attestation

- **Root of trust:** Bootstrap from TPM
- **Efficient TPM ops:** Scalable delegation of trust to software-based μTPM
- Allows developers to achieve code-security, data integrity and secrecy for selected portions of an application

μArch // Monitor low-level features

High-level actions
Content semantics
Payload syntax
Function calls
Syscalls
Architectural

Microarchitectural

Software

Hardware

CPU
Branch Prediction Unit
TLB
Instruction Cache
Data Cache
L2 Cache
μArch // Monitor low-level features

Transparent runtime monitoring of μArch side-effects of malicious actions using HPC

I. Programs (including malware) show distinct μArch profiles
   • Signature-based detection of Android malicious apps
   • Detection ~90% TP rate for < 5% FP rate
   • Set the stage for hardware AV

II. Shellcode perturbs the normal μArch profiles
   • Anomaly-based detection of Windows shellcode
   • Detect in-flight shellcode that executes within target program
   • Prelim results on adversarial difficulty of evading such defenses

[Demme et al., 2013]

[Tang et al., 2014]

μArch // Detecting ROP

Use μArch events as “trigger” for CFI violation checks

- Key insight: ROP results in mispredicted RETs
- Configure HPC to interrupt after 8 RET mispredictions
- Interrupt handler checks for illegal non call-preceded RETs
- ~3.6% - ~4.4% runtime overhead (kernel compilation)

Physical // Power fingerprinting

Monitor for anomalous execution via side-channel power consumption patterns

I. Windows Mobile on HP iPAQ Phone
   - Monitor power consumption using external Hall-Effect probe
   - Collection $\rightarrow$ Noise-filtering $\rightarrow$ Compression $\rightarrow$ Chi-square dist.
   - Detect battery-depletion attacks and synthetic mobile worms
     [Kim et al., 2008]

II. Embedded devices (eg. Medical devices, PLCs)
   - Non-intrusively monitor power, using sense register at AC neutral
   - Transparent to devices
     [Clark et al., 2013]

Monitor for anomalous execution via side-channel power consumption patterns

III. Futility of malware detection using power patterns
   - Use PowerTutor software – battery state of discharge power models
   - Test more representative malicious behavior (e.g., premium sms)
   - Challenging to detect mobile malware
     - On full-featured phones
     - With lots of sensors like accelerometers and GPS
     - Error rates of current measurement techniques too high to detect additional energy consumption

[Hoffmann et al., 2013]
Conclusions

• Survey the breadth of hardware-assisted runtime malware detection works

• Virtualization technology is a key enabler for many of these works

• Hardware assistive technologies when applied right can be
  • Complementary to software approaches
  • Disruptive by opening new possibilities for monitoring
  • Telling in informing hardware design
What’s next ...

Adversarial cost in Defense-in-depth using both μArch and Arch features

Context-sensitive and efficient μArch profiling

Intel recent/upcoming HW features for malware detection
Thank you!

Image credits: Illustrated guide to PhD. Matt Might.
Some prescient quotes

Where the threat ... is considered high, it would be possible to augment the internal auditing mechanisms of the individual computer with external measurements of busy or idle states of the CPU, memory, secondary storage ... and detect “pure” phantom use ...

[Anderson, 1980]

... it may be possible for a person to escape detection through gradual modifications of behavior or through subtle forms of intrusion that use low-level features of the target system that are not monitored (because they would produce too much data) ...

[Dennings, 1987]


