SHARK: Architectural Support for Autonomic Protection Against Stealth by Rootkit Exploits

Vikas R. Vasisht
Hsien-Hsin S. Lee

School of Electrical and Computer Engineering
Georgia Tech
Rootkit Definition

A set of programs that allows a permanent or consistent, undetectable presence on a computer

- Not an exploit to gain elevated access
- Conceal all evidences and malware activities

Rootkit’s functions:

Hide processes, files, network connections and conceal malware activities
Example - Hidden Keylogger

Adversary

Task Manager looks clean

www.anybank.com
...login...
...password...

OS compromised & Rootkit installed

SHARK: Vasisht & Lee
Rootkit Technique (I)

System Administrator (E.g., “ps”, “top”)

User Program

API Function

Library

Import Address Table

Choose Interrupt Handler from IDT

Choose Syscall from SSDT

Syscall Function

Interrupt Descriptor Table

System Service Descriptor Table

SHARK: Vasisht & Lee
Rootkit Technique (I)

System Administrator (E.g., “ps”, “top”)

USER SPACE

User Program

API Function

Library

USER SPACE

Import Address Table

KERNEL SPACE

Choose Interrupt Handler from IDT

Choose Syscall from SSDT

KERNEL SPACE

Interrupt Descriptor Table

System Service Descriptor Table

Return

Modify OS execution flow to hide traces of malware

SHARK: Vasisht & Lee
Rootkit Technique (II)

"ps"

Safe machine
>>ps
P-1
P-2
P-3
Malware P-5

Compromised Machine
>>ps
P-1
P-2
P-3
P-5

Direct Kernel Object Modification

Manipulate Kernel Data to remove malware information

SHARK: Vasisht & Lee
Rootkit Detection Techniques

- **Software based techniques:**
  - Signature/Behavioral detection
    - Works for only known rootkits
  - Cross-View based detection
    - Complex rootkits compromise low level OS view
  - Integrity based detection
    - Rootkits fake memory contents – Shadow Walker rootkit

- **Hardware based techniques:**
  - **CoPilot** *(N. Petroni et al. [USENIX’04])*
    - Integrity of host memory checked in a remote admin station
    - Send a faked memory snapshot to the remote machine.
Sophisticated Rootkits

Sub-Virt\textsuperscript{1}

Host OS

Hardware

Infection

Malicious OS

Virtual Machine Monitor

Host OS

Hardware

Bluepill\textsuperscript{2}

Native OS

VMRUN

Native OS continues execution inside VM

Hypervisor installed on-the-fly

Host OS downgraded to VM

Hypervisor below the host OS

1. King et al. [Symposium on Security and Privacy’06]
2. Joanna Rutkowska [Black Hat’06]

SHARK: Vasisht & Lee
Challenge

We cannot detect hidden processes, VMs and VMMs using software techniques

Getting direct feedback from HW

No

Seeing a clean system

Malware enjoying hardware resources

SHARK: Vasisht & Lee
Motivation – Process Context Aware Architecture

OS completely manages processes and HW can be fooled.
Address space isolation achieved by page table encryption
SHARK – Secure Hardware Against RootKits

- Hardware assisted PID Generation
  - Software PIDs vulnerable

- Page Table Encryption/Decryption
  - Page table update: Hardware support for every update
  - TLB miss: Page table decryption

- Process Authentication
  - On a context switch, PID → HPID Register
  - TLB miss: HPID used for decryption
Hardware Assisted PID Generation

OS

New Process → Page Table

Encrypted PTE

SHARK

PID++

1st PTE

Counter mode AES Encryption

64-bit Counter

PID

128-bit Secret Hardware Key

PID returned to the OS only after initial encryption

SHARK: Vasisht & Lee
Page Table Encryption (x86)

Faulted VPN

Level 1  Level 2  Level 3  Byte within page

V  PTE

128 V-Bits

32-bits PTE

3rd Level - PT

4 PTEs

2nd Level - PMD

V  PDE

128 V-Bits

1st Level - PGD

CR3

SHARK: Vasisht & Lee
Page Table Encryption (x86)

SHARK: Vasisht & Lee

Counter(PID) not a secret; HW key is secret
TLB Update (x86) – Handled by SSM

Memory Access

TLB miss: Two V-bit array decryptions + one PTE decryption

Hardware Page Table Walk

Page Table walk

Counter Mode Decryption (AES-128)

Counter Mode Decryption (AES-128)

TLB Update

VPN

PPN

3rd Level - PT

CR3

HPID Register

SHARK: Vasisht & Lee
Instructions supported in SHARK

- **GENPID-** Generate a new PID
  - Used when a new process is created

- **MODPT-** Update the page table of a process
  - Used when page tables have to be modified

- **DECPT-** Decrypt a process' page table entry
  - Used to know the physical pages of processes
MODPT: Physical Page Tracking

- MODPT used to Invalidate a page table entry:

- MODPT used to Validate a page table entry:

Tracks the association of memory page and owning process
Stealth Checker

- Implemented in Firmware
- Encrypts and sends PIDs to a remote system admin machine
- Hardware and software lists compared in the remote machine

SHARK: Vasisht & Lee
Experimental Analysis

• Functionality Evaluation
  – BOCHS emulator + modified Linux 2.6.16.33
  – Rootkits installed: Adore 0.42, Knark 2.4.3, Phide, Enyelkm.en.v1.1, and Mood-nt-2.3
  – SHARK was able to detect all rootkits

• Performance Evaluation
  – VirtuTech SIMICS
  – Performance overhead due to encryption/decryption

SHARK: Vasisht & Lee
Performance Evaluation

- SPEC 2006 benchmark suite
- Emulated first 2B instructions
  - More page faults and TLB updates
- SHARK Overhead in recompiled Linux kernel 2.6.16.33
  - MODPT instruction: 6 * AES + SHA-256
  - TLB Refill: 3 * AES
  - DECPT instruction: 3 * AES
- Sensitivity study for different TLB configurations
  - 4 KB and 2 MB pages supported (x86)
  - Varied number of TLB entries
- TLB flushed upon every context switch as in x86 machines
SPEC2006

Performance impact with different TLB organizations

- More context switches and more TLB misses
- Sensitive to the number of entries for 2MB pages in TLB
- Average CPI overhead is 1.3%

SHARK: Vasisht & Lee
**SPEC2006 (6 System Configurations)**

<table>
<thead>
<tr>
<th>TLB Config 1</th>
<th>4 KB Page, 128 Entries, 4 Way</th>
</tr>
</thead>
<tbody>
<tr>
<td>TLB Config 2</td>
<td>4 KB Page, 128 Entries, 4 Way</td>
</tr>
<tr>
<td>TLB Config 3</td>
<td>4 KB Page, 256 Entries, 4 Way</td>
</tr>
<tr>
<td>TLB Config 4</td>
<td>4 KB Page, 256 Entries, 4 Way</td>
</tr>
<tr>
<td>TLB Config 5</td>
<td>4 KB Page, 512 Entries, 4 Way</td>
</tr>
<tr>
<td>TLB Config 6</td>
<td>4 KB Page, 512 Entries, 4 Way</td>
</tr>
</tbody>
</table>

- Larger AES latency increases the overhead
- Larger L2 cache (longer L2 latency) lowers the overhead
- Average overhead:
  
  **Range : 0.45% - 4.7%**

SHARK: Vasisht & Lee
Conclusions

- SHARK is the first synergistic micro-architecture and OS technique to address the Rootkit exploits.

- Concealed activity at User, Kernel and VMM levels will be revealed.

- Low performance overhead makes it practical.
Thank you