Rethinking System Audit Architectures for High Event Coverage and Synchronous Log Availability

Varun Gandhi, Sarbartha Banerjee, Aniket Agarwal, Adil Ahmed, Sangho Lee, and Marcus Peinado
System auditing is critical for forensic analysis
System auditing is critical for forensic analysis

Enterprise Machine

Audit System
System auditing is critical for forensic analysis

Trace of security-related events
E.g., system calls
System auditing is critical for forensic analysis

Audit System

Trace of security-related events
E.g., system calls

Enterprise Machine

Forensic Analyst
(Monet Version)
What makes a log effective for forensic analysis?

Enterprise Machine

Audit System

Trace of security-related events
E.g., system calls

Forensic Analyst
(Monet Version)
What makes a log effective for forensic analysis?

Enterprise Machine

Audit System

LOG

Trace of security-related events
E.g., system calls

Should not be tampered

Forensic Analyst (Monet Version)
What makes a log effective for forensic analysis?

- Trace of security-related events
  E.g., system calls

- Should not be tampered

- Keep detailed event trace

Audit System

Enterprise Machine

Forensic Analyst
(Monet Version)
What makes a log effective for forensic analysis?

- Should not be tampered
- Keep detailed event trace
- Trace of security-related events
  E.g., system calls

Yay! I can analyze the attack

Audit System

Enterprise Machine

Forensic Analyst (Monet Version)
What makes a log effective for forensic analysis?

Present day audit systems do not achieve this:
- Should not be tampered
- Keep detailed event trace

Yay! I can analyze the attack

Trace of security-related events
E.g., system calls

Audit System

Enterprise Machine

Forensic Analyst (Monet Version)
Kernel exploits can tamper audit logs
Kernel exploits can tamper audit logs
Current systems don’t protect all logs up to the compromise.
Current systems don’t protect all logs up to the compromise

High I/O Latency!

Protected Storage
Current systems don’t protect all logs up to the compromise

Enterprise Machine

OS

Audit System
Log Generation

Kernel Memory

Protected Storage

High I/O Latency!
Current systems don’t protect all logs up to the compromise.
Current systems don’t protect all logs up to the compromise
Audit systems also don’t keep a detailed trace of events
Audit systems also don’t keep a detailed trace of events

- Rule sets determine what system call events get logged
Audit systems also don’t keep a detailed trace of events

- Rulesets determine what system call events get logged

- A set of rulesets exist as standards across government, industry, and academia
Audit systems also don’t keep a detailed trace of events

• Rulesets determine what system call events get logged

• A set of rulesets exist as standards across government, industry, and academia

• Analyzed 164 PoC kernel exploits on a subset of standard rule sets
Audit systems also don’t keep a detailed trace of events

• Rule sets determine what system call events get logged

• A set of rulesets exist as standards across government, industry, and academia

• Analyzed 164 PoC kernel exploits on a subset of standard rule sets

<table>
<thead>
<tr>
<th></th>
<th>MITRE</th>
<th>NISPOM</th>
<th>STIG</th>
<th>MSD</th>
<th>Academic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>14.7%</td>
<td>12.4%</td>
<td>9.8%</td>
<td>10.2%</td>
<td>15.4%</td>
</tr>
<tr>
<td>Coverage</td>
<td>60/164</td>
<td>77/164</td>
<td>77/164</td>
<td>80/164</td>
<td>11/164</td>
</tr>
</tbody>
</table>
Audit systems also don’t keep a detailed trace of events

- Rule sets determine what system call events get logged
- A set of rulesets exist as standards across government, industry, and academia
- Analyzed 164 PoC kernel exploits on a subset of standard rule sets

<table>
<thead>
<tr>
<th></th>
<th>MITRE</th>
<th>NISPOM</th>
<th>STIG</th>
<th>MSD</th>
<th>Academic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>14.7%</td>
<td>12.4%</td>
<td>9.8%</td>
<td>10.2%</td>
<td>15.4%</td>
</tr>
<tr>
<td>Coverage</td>
<td>60/164</td>
<td>77/164</td>
<td>77/164</td>
<td>80/164</td>
<td>11/164</td>
</tr>
</tbody>
</table>
Audit systems also don’t keep a detailed trace of events

- Rule sets determine what system call events get logged
- A set of rulesets exist as standards across government, industry, and academia
- Analyzed 164 PoC kernel exploits on a subset of standard rule sets

<table>
<thead>
<tr>
<th></th>
<th>MITRE</th>
<th>NISPOM</th>
<th>STIG</th>
<th>MSD</th>
<th>Academic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>14.7%</td>
<td>12.4%</td>
<td>9.8%</td>
<td>10.2%</td>
<td>15.4%</td>
</tr>
<tr>
<td>Coverage</td>
<td>60/164</td>
<td>77/164</td>
<td>77/164</td>
<td>80/164</td>
<td>11/164</td>
</tr>
</tbody>
</table>
How do we build an audit system with these guarantees?

Enterprise Machine

Audit System

Should not be tampered

Keep detailed event trace

Forensic Analyst (Monet Version)
Synchronous logging prevents tampering

Enterprise Machine

User Process

Operating System

Audit System

Log Generation

Protected Storage
Synchronous logging prevents tampering

Enterprise Machine

User Process

Operating System

1. Syscall Invoked

Audit System

Log Generation

Protected Storage
Synchronous logging prevents tampering

Enterprise Machine

User Process

Operating System

1. Syscall Invoked
   Log Generation

2. Protect Log Record

Audit System

Protected Storage
Synchronous logging prevents tampering

Enterprise Machine

User Process

Operating System

1. Syscall Invoked

Audit System

2. Protect Log Record

Log Generation

3. Syscall Executed

Protected Storage
Synchronous logging prevents tampering

Enterprise Machine

User Process

Operating System

1. Syscall Invoked
   Audit System

2. Protect Log Record

3. Syscall Executed

Log Generation

Protected Storage

Too Slow!
Logging all system calls ensures detailed event tracing
Logging all system calls ensures detailed event tracing

- Kernel exploits leverage a diverse set of system calls
Logging all system calls ensures detailed event tracing

- Kernel exploits leverage a diverse set of system calls
- Logging all system calls guarantees event coverage
We measured Auditd overhead when logging all system calls on real-world workloads.

- Kernel exploits leverage a diverse set of system calls.
- Logging all system calls guarantees event coverage.
- We measured Auditd overhead when logging all system calls on real-world workloads.

Logging all system calls ensures detailed event tracing.
Logging all system calls ensures detailed event tracing

- Kernel exploits leverage a diverse set of system calls
- Logging all system calls guarantees event coverage
- We measured Auditd overhead when logging all system calls on real-world workloads
- Even for asynchronous logging, the slowdown is prohibitive

![Auditd Overhead Graph]

Auditd RuleSet = All Syscalls
OMNIlog addresses these efficiency challenges

Challenge 1: High I/O latency for synchronous logging

Challenge 2: Inefficient logging pipeline
OMNIlog addresses these efficiency challenges

Challenge 1: High I/O latency for synchronous logging

- Isolating logs in memory within a protected environment and eventually persist

Challenge 2: Inefficient logging pipeline
OMNIlog addresses these efficiency challenges

Challenge 1: High I/O latency for synchronous logging

- Isolating logs in memory within a protected environment and eventually persist

Challenge 2: Inefficient logging pipeline

- Optimizing the end-to-end pipeline from log generation to persistence
How does OMNILog build a protected environment?

Enterprise Machine

User Process

Audit System

Log Generation

Protected Buffer

Protected Disk

Untrusted OS

TrustZone/VMX

Hardware
How does OMNIlog build a protected environment?

Enterprise Machine

User Process

Audit System

Log Generation

Protected Buffer

Untrusted OS

TrustZone/VMX

Hardware

Protected Disk
How does OMNIlog build a protected environment?

Enterprise Machine

- User Process

Audit System

- Log Generation

- Protected Buffer

Untrusted OS

TrustZone/VMX

Hardware

Protected Disk
How does OMNIlog build a protected environment?

Enterprise Machine

User Process

Audit System

Log Generation

Protected Buffer

Protected Disk

Untrusted OS

TrustZone/VMX

Hardware
How does OMNIlog build a protected environment?

1. Syscall Invoked

- Log Generation
- Audit System

Protected Buffer

Enterprise Machine

User Process

Untrusted OS

TrustZone/VMX

Hardware

Protected Disk
How does OMNILog build a protected environment?

Enterprise Machine

User Process

Audit System

1. Syscall Invoked

2. Sync Protect Log Record

Log Generation

Protected Buffer

Protected Disk

Untrusted OS

TrustZone/VMX

Hardware
How does OMNIlog build a protected environment?

1. Syscall Invoked
   - Log Generation
2. Sync Protect Log Record
   - Protected Buffer
3. Syscall Executed
   - Audit System
   - Protected Disk

Enterprise Machine

User Process

Untrusted OS

Trusted Zone/VMX

Hardware
How does OMNILog build a protected environment?

1. Syscall Invoked
   - Log Generation

2. Sync Protect Log Record
   - Protected Buffer

3. Syscall Executed

4. Async Persist Logs
   - Protected Disk

Enterprise Machine

User Process

Untrusted OS

TrustZone/VMX

Hardware
How does OMNIILog optimize the logging pipeline?

Native
Auditd

OMNIILog
How does OMNILog optimize the logging pipeline?

Native Auditd

Log Generation

Human-Readable
~12k cycles | 1KB
How does OMNILOG optimize the logging pipeline?

Native Auditd -> Log Generation

Human-Readable
~12k cycles | 1KB

Global Buffer
How does OMNILog optimize the logging pipeline?

Log Generation

Human-Readable
~12k cycles | 1KB

Global Buffer

wait when the buffer is full

Disk

Native Auditd

OMNILog
How does OMNIlog optimize the logging pipeline?

Native Auditd

Log Generation → Global Buffer

Human-Readable
~12k cycles | 1KB

wait when the buffer is full

Disk

OMNIlog

Fast
Log Generation

Raw and compressed
~3k cycles | 64B
How does OMNILog optimize the logging pipeline?

Native Auditd

Log Generation → Global Buffer → Disk

Human-Readable

~12k cycles | 1KB

wait when the buffer is full

OMNILog

Fast Log Generation → Isolated Per-core Memory Region

Raw and compressed

~3k cycles | 64B

Eliminate inter-core contention
How does OMNILog optimize the logging pipeline?

Native Auditd

Log Generation → Global Buffer → Disk

Human-Readable

~12k cycles | 1KB

wait when the buffer is full

OMNILog

Fast
Log Generation

Isolated Per-core Memory Region

Raw and compressed

~3k cycles | 64B

Eliminate inter-core contention

Dual-buffers and background writes

Protected Disk
OMNILog incurs low overhead over native execution

OMNILog incurs low overhead over native execution.
Conclusion

• Current audit systems architectures:
  • Can’t prevent tampering of all logs under kernel exploits
  • Can’t keep a detailed trace of security-related events

• OmniLog redesigns audit architecture to:
  • Prevent all log tampering for all events
  • Keep a full trace of all syscalls executed during kernel exploits

• OMNILog’s overhead compared to native execution is \( \sim 3.5\% \) (geomean)

Thanks!
vgandhi@g.harvard.edu