Survivor: A Fine-Grained Intrusion Response and Recovery Approach for Commodity Operating Systems

Ronny Chevalier¹,²  David Plaquin¹  Chris Dalton¹  Guillaume Hiet²
ACSAC, December 13th, 2019

¹ HP Labs, United Kingdom (ronny.chevalier@hp.com, david.plaquin@hp.com, cid@hp.com)
² CIDRE Team, CentraleSupélec/Inria/CNRS/IRISA, France (ronny.chevalier@centralesupelec.fr, guillaume.hiet@centralesupelec.fr)
Agenda

Problem Statement

Approach and Prototype

Evaluation

Conclusion
Preventive Security is not Sufficient

Examples of preventive security mechanisms

• Access control
• Cryptography
• Firewalls
Preventive Security is not Sufficient

Examples of preventive security mechanisms

- Access control
- Cryptography
- Firewalls

Attackers will eventually bypass our security policy

- (Unknown) vulnerability
- System not updated
- Misconfiguration
Preventive Security is not Sufficient

Examples of preventive security mechanisms

- Access control
- Cryptography
- Firewalls

Operating systems should not only prevent but detect and survive intrusions

- System not updated
- Misconfiguration
Intrusion Detection Systems\(^1\) exist in commodity OSs
e.g., Antivirus software share many aspects of host-based IDSs\(^2\)

---


\(^2\) Morin and Mé, “Intrusion detection and virology: an analysis of differences, similarities and complementariness”. 
Commodity Operating Systems Can Detect but Cannot Survive Intrusions

Intrusion Detection Systems exist in commodity OSs e.g., Antivirus software share many aspects of host-based IDSs.

What can we do after a system has been compromised? Eventually we want to patch the system.

---

1 Anderson, Computer Security Threat Monitoring and Surveillance; Denning, “An Intrusion-Detection Model”.
2 Morin and Mé, “Intrusion detection and virology: an analysis of differences, similarities and complementariness”.
Commodity Operating Systems Can Detect but Cannot Survive Intrusions

Intrusion Detection Systems\(^1\) exist in commodity OSs
e.g., Antivirus software share many aspects of host-based IDSs\(^2\)

What can we do after a system has been compromised?
Eventually we want to patch the system

What should we do while waiting for the patches?
Deliver service despite the attacker’s presence

---


\(^2\) Morin and Mé, “Intrusion detection and virology: an analysis of differences, similarities and complementariness”.
Related Work: Survivability, Recovery, and Response

Intrusion Survivability$^3$

- Trade-off between the availability and the security risk
- Limitations: lack of focus on commodity OSs

---

Related Work: Survivability, Recovery, and Response

**Intrusion Survivability**
- Trade-off between the availability and the security risk
- Limitations: lack of focus on commodity OSs

**Intrusion Recovery**
- Restore the system in a safe state when an intrusion is detected
- Limitations: the system is still vulnerable and can be reinfected

---


4 Goel et al., "The Taser Intrusion Recovery System"; Xiong, Jia, and Liu, "SHELF: Preserving Business Continuity and Availability in an Intrusion Recovery System".
Related Work: Survivability, Recovery, and Response

**Intrusion Survivability**

- Trade-off between the availability and the security risk
- Limitations: lack of focus on commodity OSs

**Intrusion Recovery**

- Restore the system in a safe state when an intrusion is detected
- Limitations: the system is still vulnerable and can be reinfected

**Intrusion Response**

- Limit the impact of an intrusion on the system
- Limitations: coarse-grained responses and few host-based solutions

---


Related Work: Survivability, Recovery, and Response

Intrusion Survivability

- Trade-off between the availability and the security risk
- Limitations: lack of focus on commodity OSs

Intrusion Recovery

- Restore the system in a safe state when an intrusion is detected
- Limitations: the system is still vulnerable and can be reinfected

Intrusion Response

- Limit the impact of an intrusion on the system
- Limitations: coarse-grained responses and few host-based solutions

Existing approaches do not allow commodity OSs to survive intrusions while maintaining the availability of the services.

---


How to design an OS so that it can **survive** ongoing intrusions by making a **trade-off** between **availability** and **security risk**?
Agenda

Problem Statement

Approach and Prototype

Evaluation

Conclusion
Running Example

Service: Gitea, a Git Self-Hosting Server
Open source clone of Github (git repositories, bug tracking,...)

Intrusion: Ransomware
It compromises data availability
Approach Overview

Illustrative Example

Running Example
Gitea infected with some ransomware

When Detected
• Recovery: We restore the service and the encrypted files to a previous state
• Apply restrictions: We remove the ability to write on the file system

Positive Impact
If the ransomware reinfects the service → cannot compromise the files

Degraded Mode
Users can no longer push to repositories → trade-off between availability and security risk
Approach Overview

During the normal operation of the system

Intrusion Detection Monitor Operating System

Service n Apache Gitea

Devices Network Filesystem
During the normal operation of the system

Approach Overview

Intrusion Detection

Monitor

Operating System

Service

Apache

Gitea

Devices

Network

Filesystem

Checkpoint & Log
Approach Overview

During the normal operation of the system

1. Periodic checkpointing
Approach Overview

During the normal operation of the system

1. Periodic checkpointing
2. Log file write accesses
Approach Overview

How our approach allows the system to survive intrusions after their detection?
How our approach allows the system to survive intrusions after their detection?
Approach Overview

How our approach allows the system to survive intrusions after their detection?

1. Restore infected objects

- Restore service
- Restore files

Use Logs / States

Operating System

- Service n
- Apache
- Gitea

Intrusion Detection

Monitor

Alert

Recovery & Response
Approach Overview

How our approach allows the system to survive intrusions after their detection?

Intrusion Detection → Monitor → Operating System

- Service
- Apache
- Gitea

Devices
- Network
- Filesystem

Recovery & Response
1. Restore infected objects
2. Withstand reinfection

Alert

- Restore service
- Apply restrictions
- Restore files

Use Logs / States

Remove privileges and decrease resource quotas

Per-service responses to prevent attackers to achieve their goals
Approach Overview

How our approach allows the system to survive intrusions after their detection?

1. Restore infected objects
2. Withstand reinfection
3. Maintain core functions

Potential Degraded Mode

The degraded mode maintains core functions while waiting for patches
Approach Overview

How our approach allows the system to survive intrusions after their detection?

- **Intrusion Detection**
  - Monitor
  - Alert
  - Policies

**Recovery & Response**
- 1. Restore infected objects
- 2. Withstand reinfection
- 3. Maintain core functions

- **Operating System**
  - Service
  - Apache
  - Gitea
  - Devices
  - Network
  - Filesystem

- **Logs / States**
  - Use

- **Restore service**
  - Apply restrictions
  - Restore files
We select responses that **minimize** the availability impact on the service while **maximizing** the security
Cost-Sensitive Response Selection

understand the intrusion - find possible responses - select a response
Cost-Sensitive Response Selection

Malicious behaviors
- Availability violation
- Consume system resources
- Crack passwords
- Mine for cryptocurrency
- Compromise data availability
- Compromise access to information assets
- Command and Control
- Determine C2 server
- Generate C2 domain name(s)
- Receive data from C2 server
- Control malware via remote command
- Update configuration
- ...

Example of malicious behaviors

Costs
very low, low, moderate, high, very high, critical
Cost-Sensitive Response Selection

Costs
very low, low, moderate, high, very high, critical
Cost-Sensitive Response Selection

Costs
very low, low, moderate, high, very high, critical

Example of a non-exhaustive malicious behavior hierarchy (Source: MAEC of the STIX project)
Cost-Sensitive Response Selection

Example of a non-exhaustive malicious behavior hierarchy (Source: MAEC of the STIX project)

Costs

very low, low, **moderate**, high, very high, critical
Cost-Sensitive Response Selection

- Threat Intelligence
- Additional Information
- Intrusion Detection
- Initial Alert
- Responses
- Initial Alert
- Malicious Behaviors
- Malicious Behaviors Costs
- Response Costs

- read-only FS, disable syscall, ...
- ransomware
- Example

Defined by the administrator/developer

Diagram:
- Initial Alert
- Malicious Behaviors
- Malicious Behaviors Costs
- Response Costs
- read-only FS, disable syscall, ...
- ransomware
- Example

Text:
- Defined by the administrator/developer
- Example
Cost-Sensitive Response Selection

Example of a non-exhaustive per-service response hierarchy

Responses may be provided via the exchange format STIX (e.g., the course of action field)
Cost-Sensitive Response Selection

- Initial Alert
- Additional Information
- Threat Intelligence
- Intrusion Detection
- Responses: read-only FS, disable syscall, ...
- Malicious Behaviors Costs
- Response Costs
- Response Efficiency
- Malicious Behaviors
- ransomware

Costs
- Efficiency
- Likelihood
- Very likely

Defined by the administrator/developer
Defined by threat intelligence
Text Example
Cost-Sensitive Response Selection

**Risk Matrix**

<table>
<thead>
<tr>
<th>Malicious Behavior Cost</th>
<th>Very low (0–0.2)</th>
<th>Low (0.2–0.4)</th>
<th>Moderate (0.4–0.6)</th>
<th>High (0.6–0.8)</th>
<th>Very high (0.8–1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Very likely</td>
<td>L</td>
<td>M</td>
<td>H</td>
<td>H</td>
<td>H</td>
</tr>
<tr>
<td>Likely</td>
<td>L</td>
<td>M</td>
<td>M</td>
<td>H</td>
<td>H</td>
</tr>
<tr>
<td>Probable</td>
<td>L</td>
<td>L</td>
<td>M</td>
<td>M</td>
<td>H</td>
</tr>
<tr>
<td>Unlikely</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>Very unlikely</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>L</td>
</tr>
</tbody>
</table>

**Risk Matrix**
- Defined by the administrator/developer
- Defined by threat intelligence
- Defined by the organization

**Example**
- read-only FS, disable syscall,...
- ransomware

**Threat Intelligence**
- Additional Information

**Intrusion Detection**
- Initial Alert

**Malicious Behaviors Costs**
- Response Costs
- Response Efficiency

**Confidence (Likelihood)**
- Very likely (0.8–1)
- Likely (0.6–0.8)
- Probable (0.4–0.6)
- Unlikely (0.2–0.4)
- Very unlikely (0–0.2)
Cost-Sensitive Response Selection

Cost vs Efficiency

It prioritizes efficiency if the risk is high, and cost if the risk is low

$max(Risk \times Efficiency + (1 - Risk) \times (1 - Cost))$
Cost-Sensitive Response Selection

We rely on:
- Possible responses
- Malicious behaviors
- Likelihood

We assign:
- Response costs
- Malicious behavior costs
- Risk matrix

We select responses based on:
- Response cost
- Risk
- Response efficiency

\[
\max (\text{Risk} \times \text{Efficiency} + (1 - \text{Risk}) \times (1 - \text{Cost}))
\]
## Prototype Implementation for Linux-Based Systems

### Projects Used or Modified

<table>
<thead>
<tr>
<th>Project</th>
<th>What does it do? What is it?</th>
<th>Why do we use/modify it?</th>
<th>Lines of C code added</th>
</tr>
</thead>
<tbody>
<tr>
<td>systemd</td>
<td>system and service manager</td>
<td>Orchestration</td>
<td>2639</td>
</tr>
<tr>
<td>CRIU</td>
<td>checkpoint &amp; restore processes</td>
<td>Restoration</td>
<td>383</td>
</tr>
<tr>
<td>snapper</td>
<td>manage snapshots of file systems</td>
<td>Restoration</td>
<td>0</td>
</tr>
<tr>
<td>Linux kernel</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cgroups</td>
<td>set of processes bound to a set of limits</td>
<td>Logging &amp; Responses</td>
<td>460</td>
</tr>
<tr>
<td>seccomp</td>
<td>filter system calls</td>
<td></td>
<td></td>
</tr>
<tr>
<td>namespaces</td>
<td>partition kernel resources</td>
<td></td>
<td></td>
</tr>
<tr>
<td>audit</td>
<td>record security relevant events</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[...]</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Agenda

Problem Statement

Approach and Prototype

Evaluation

Conclusion
Evaluation Setup

What Do We Evaluate?

- Responses effectiveness
- Cost-sensitive response selection
- Availability cost and performance impact
- Stability of degraded services
Evaluation Setup

What Do We Evaluate?

- Responses effectiveness
- Cost-sensitive response selection
- Availability cost and performance impact
- Stability of degraded services

Malware and Attacks

- Different types of malicious behaviors (botnet, ransomware, cryptominer,...)

Performance Evaluation Setup

- Various types of services (Apache, nginx, mariadb, beanstalkd, mosquitto, gitea)
- Both synthetic and real-world benchmarks using Phoronix test suite
## Security Evaluation

### Restoration and Responses Effectiveness

<table>
<thead>
<tr>
<th>Attack Scenario</th>
<th>Malicious Behavior</th>
<th>Per-service Response Policy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux.BitCoinMiner</td>
<td>Mine for cryptocurrency</td>
<td>Ban mining pool IPs</td>
</tr>
<tr>
<td>Linux.BitCoinMiner</td>
<td>Mine for cryptocurrency</td>
<td>Reduce CPU quota</td>
</tr>
<tr>
<td>Linux.Rex.1</td>
<td>Join P2P botnet</td>
<td>Ban bootstrapping IPs</td>
</tr>
<tr>
<td>Hakai</td>
<td>Communicate with C&amp;C</td>
<td>Ban C&amp;C servers’ IPs</td>
</tr>
<tr>
<td>Linux.Encoder.1</td>
<td>Encrypt data</td>
<td>Read-only filesystem</td>
</tr>
<tr>
<td>GoAhead exploit</td>
<td>Open reverse shell</td>
<td>Forbid connect syscall</td>
</tr>
<tr>
<td>GoAhead exploit</td>
<td>Data theft</td>
<td>Render paths inaccessible</td>
</tr>
</tbody>
</table>

### Results

- The service is restored
- The service can withstand the reinfection
Security Evaluation

**Cost-Sensitive Response Selection**

**Goal**
Evaluate the impact of the IDS accuracy when selecting responses

→ accurate likelihood (1), inaccurate likelihood (2), false positive (3)

**Scenario**
Survive ransomware that compromised Gitea

**Results**

- High risk: read-only filesystem (1, 3)
  - Ransomware failed to reinfect
  - Gitea still usable (can access all repositories, clone them, log in)

- Low risk: read-only paths of important git repositories (2)
  - Ransomware could not encrypt important repositories
  - Gitea still usable (can access important repositories, clone them)
Performance Evaluation

Availability Cost

- less than 300 ms to checkpoint
- less than 325 ms to restore
Performance Evaluation

Availability Cost

- less than 300 ms to checkpoint
- less than 325 ms to restore

Monitoring Cost

- Overhead present only on applications that write to the file system

(a) MB/s score with the Compilebench benchmark (more is better)
Performance Evaluation

Availability Cost

- less than 300 ms to checkpoint
- less than 325 ms to restore

Monitoring Cost

- Overhead present only on applications that write to the file system
- Small overhead in general (0.6 % - 4.5 %)

(b) Time (in seconds) to build the Linux kernel (less is better)
Performance Evaluation

Availability Cost

- less than 300 ms to checkpoint
- less than 325 ms to restore

Monitoring Cost

- Overhead present only on applications that write to the file system
- Small overhead in general (0.6 % - 4.5 %)
- Worst case (28.7 % overhead): writing small files asynchronously in burst

(c) Time (in seconds) to extract the archive (.tar.gz) of the Linux kernel source code (less is better)
Performance Evaluation

Availability Cost

• less than 300 ms to checkpoint
• less than 325 ms to restore

Monitoring Cost

• Overhead present only on applications that write to the file system
• Small overhead in general (0.6 % - 4.5 %)
• Worst case (28.7 % overhead): writing small files asynchronously in burst
• e.g., SHELF\textsuperscript{6} has 8 % and 67 % overhead

\textsuperscript{6}Xiong, Jia, and Liu, "SHELF: Preserving Business Continuity and Availability in an Intrusion Recovery System".
Agenda

Problem Statement

Approach and Prototype

Evaluation

Conclusion
Scientific Contributions and Future Work

Operating systems should not only prevent but detect and survive intrusions

What were the challenges?
- Survive while waiting for the patches
- Maintain availability while maximizing security
- Realistic use cases

Future work
- Checkpointing limitations
- Models input

ACSAC’19
Thanks for your attention!
Operating systems should not only **prevent** but **detect** and **survive** intrusions

**What were the challenges?**

- Survive while waiting for the patches
- Maintain availability while maximizing security
- Realistic use cases

---

**ACSAC’19**


