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

Ronny Chevalier [1,2] David Plaquin [1] Chris Dalton [1] Guillaume Hiet [2]

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

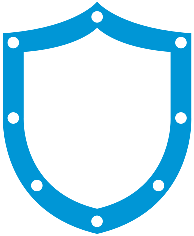
Evaluation

Conclusion

Preventive Security is not Sufficient

Examples of preventive security mechanisms

- Access control
- Cryptography
- Firewalls



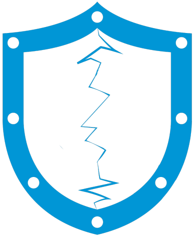
Preventive Security is not Sufficient

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

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]

[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

[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

---

[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".

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**

[3] Knight and Strunk, "Achieving Critical System Survivability Through Software Architectures"; Ellison et al., *Survivable Network Systems: An emerging discipline*.

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**

Intrusion Recovery[4]

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

---

[3] Knight and Strunk, "Achieving Critical System Survivability Through Software Architectures"; Ellison et al., *Survivable Network Systems: An emerging discipline*.

[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[3]

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

Intrusion Recovery[4]

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

Intrusion Response[5]

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

---

[3] Knight and Strunk, "Achieving Critical System Survivability Through Software Architectures"; Ellison et al., *Survivable Network Systems: An emerging discipline*.

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

[5] Balepin et al., "Using Specification-Based Intrusion Detection for Automated Response"; Shameli-Sendi, Cheriet, and Hamou-Lhadj, "Taxonomy of Intrusion Risk Assessment and Response System".

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**

Intrusion Recovery[4]

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

Intrusion Response[5]

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

---

[3] Knight and Strunk, “Achieving Critical System Survivability Through Software Architectures”; Ellison et al., *Survivable Network Systems: An emerging discipline*.

[4] Goel et al., “The Taser Intrusion Recovery System”; Xiong, Jia, and Liu, “SHELF: Preserving Business Continuity and Availability in an Intrusion Recovery System”.

[5] Balepin et al., “Using Specification-Based Intrusion Detection for Automated Response”; Shameli-Sendi, Cheriet, and Hamou-Lhadj, “Taxonomy of Intrusion Risk Assessment and Response System”.

Problem Addressed

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

Prevent

Detect

Survive

Agenda

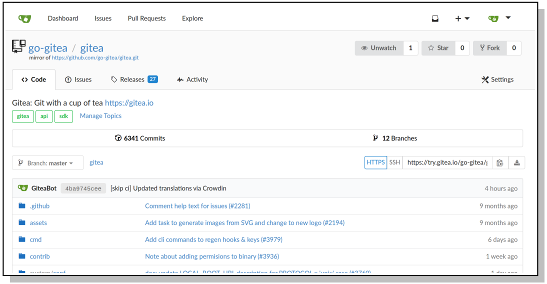
Problem Statement

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

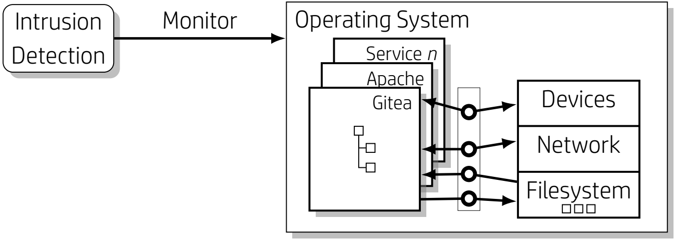
If the ransomware reinfects the service $\rightarrow$ cannot compromise the files

Degraded Mode

Users can no longer push to repositories $\rightarrow$ trade-off between availability and security risk

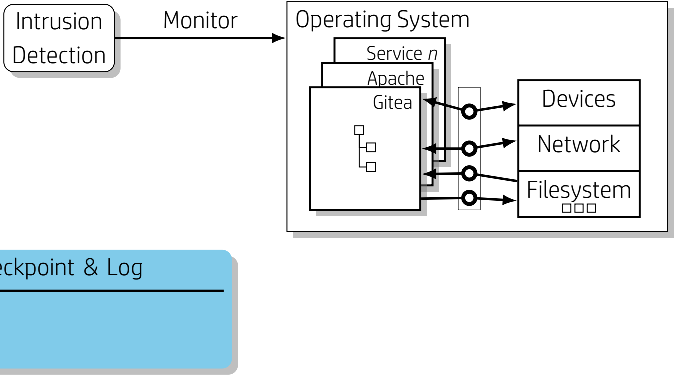
Approach Overview

During the normal operation of the system



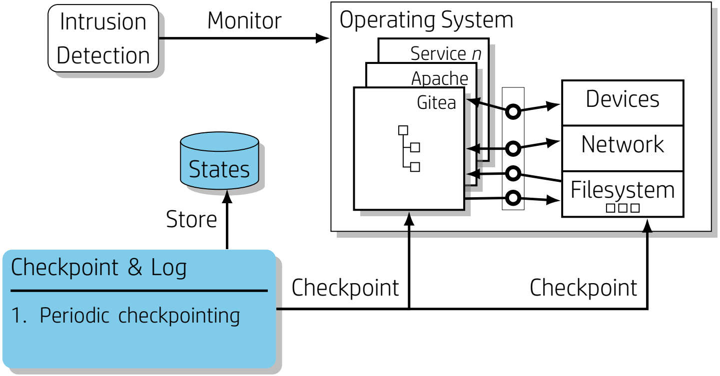
Approach Overview

During the normal operation of the system



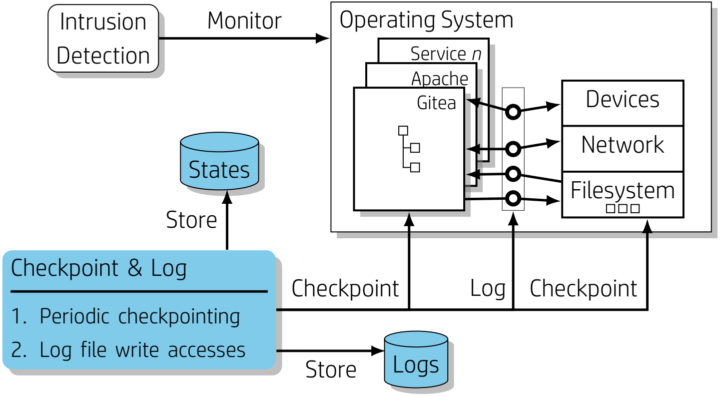
Approach Overview

During the normal operation of the system



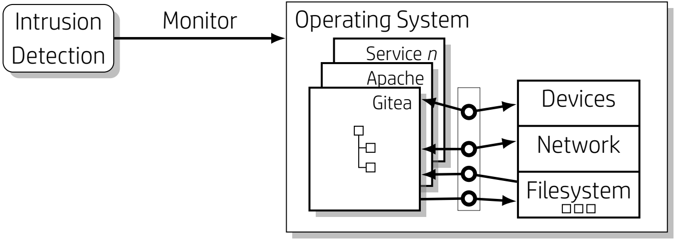
Approach Overview

During the normal operation of the system



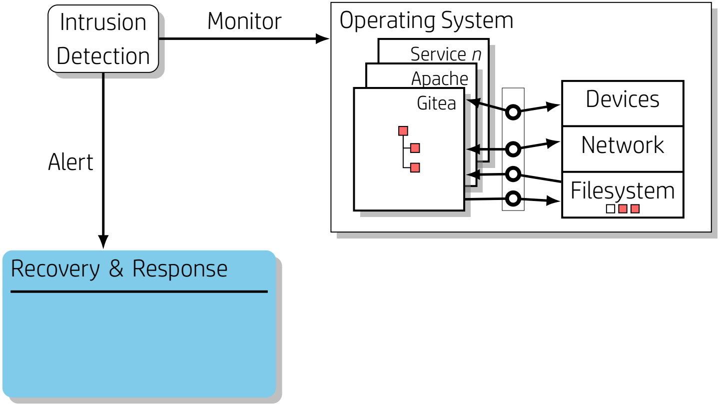
Approach Overview

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



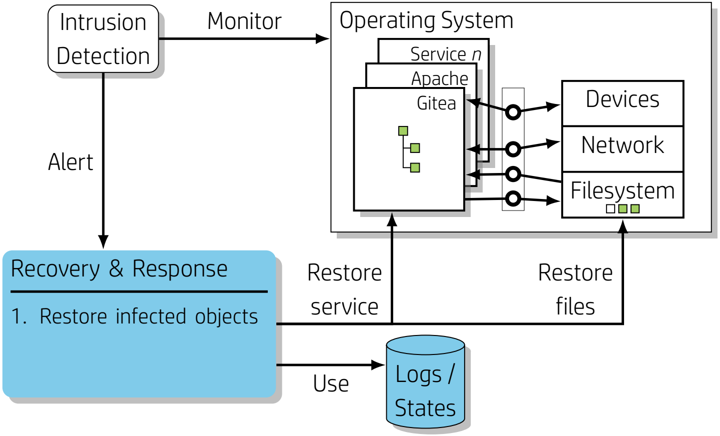
Approach Overview

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



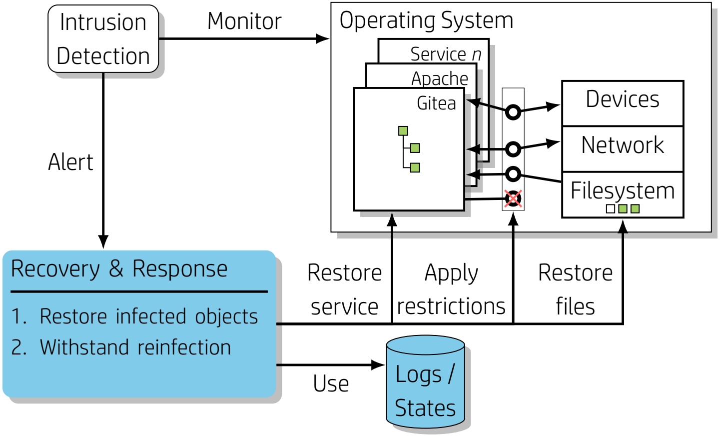
Approach Overview

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?

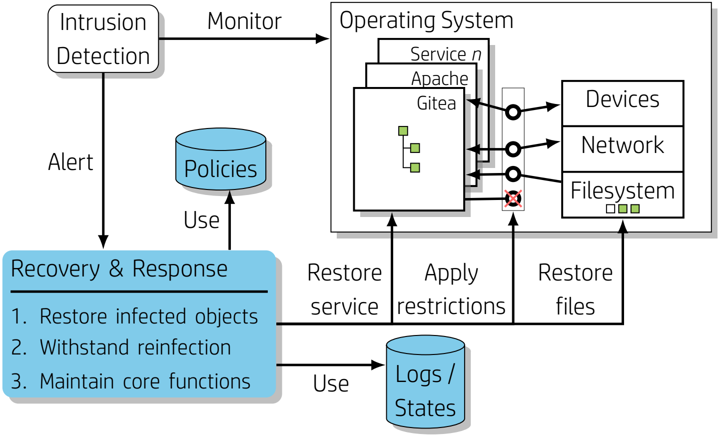


Remove privileges and decrease resource quotas

Per-service responses to prevent attackers to achieve their goals

Approach Overview

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

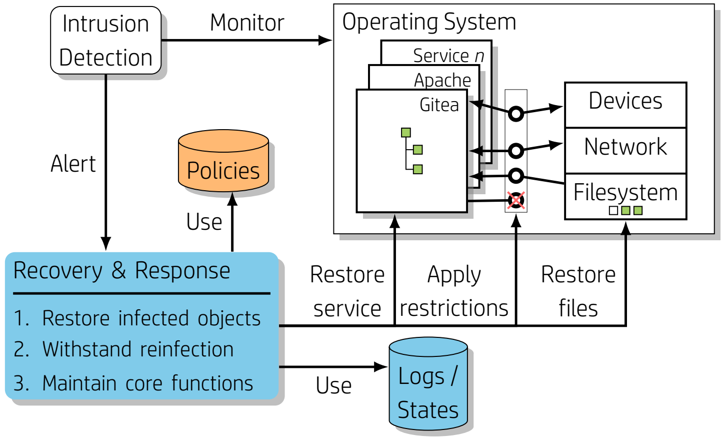


Potential Degraded Mode

The degraded mode maintains core functions **while waiting for patches**

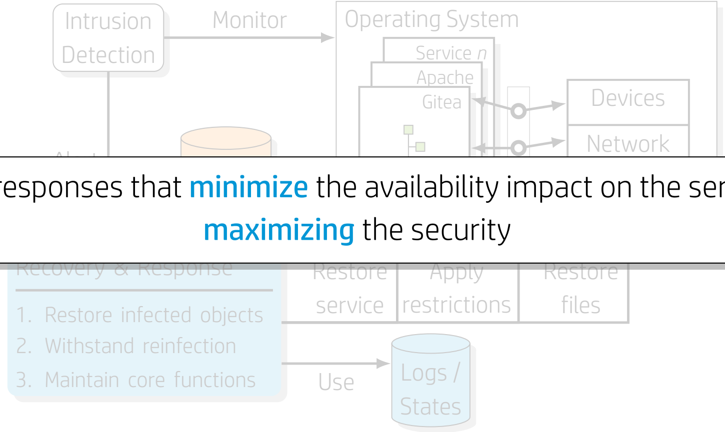
Approach Overview

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



Approach Overview

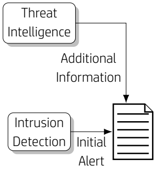
How our approach allows the system to survive intrusions after their detection?



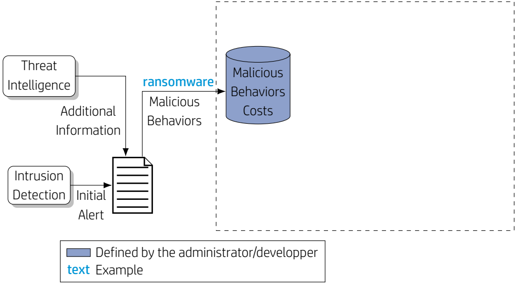
We select responses that **minimize** the availability impact on the service while **maximizing** the security

Cost-Sensitive Response Selection

understand the intrusion - find possible responses - select a response



Cost-Sensitive Response Selection



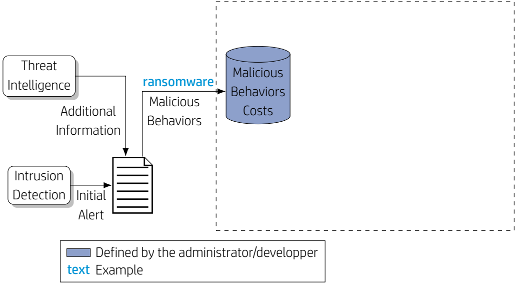
Malicious behaviors
- Availability violation
- Consume system resources
- Crack passwords
- Mine for cryptocurrency
- Compromise data availability
- Compromise access to information assets
- Command and Control
- Determine C2 server
- Generate C2 domain name(s)
- Receive data from C2 server
- Control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

Example of malicious behaviors

Costs

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

Cost-Sensitive Response Selection



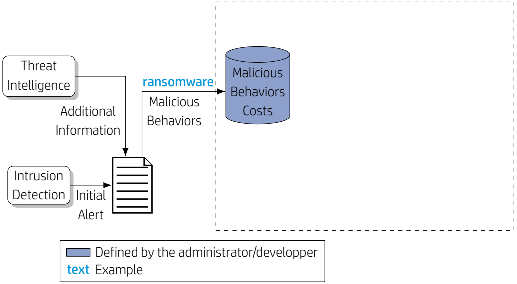
- Malicious behaviors
  - Availability violation
    - Consume system resources
      - Crack passwords
      - Mine for cryptocurrency
    - Compromise data availability
      - Compromise access to information assets
    - ...
  - Command and Control
    - Determine C2 server
      - Generate C2 domain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

Example of 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



Malicious behaviors

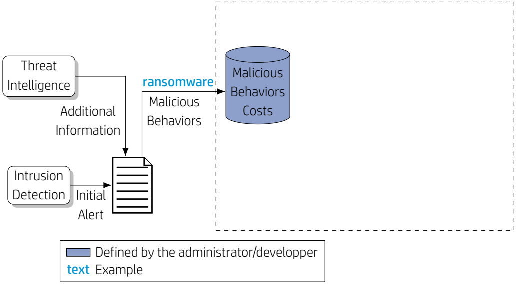
- Availability violation=**moderate**
  - Consume system resources
    - Crack passwords
    - Mine for cryptocurrency
  - Compromise data availability
    - Compromise access to information assets
  - …
- Command and Control
  - Determine C2 server
    - Generate C2 domain name(s)
  - Receive data from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

Example of 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



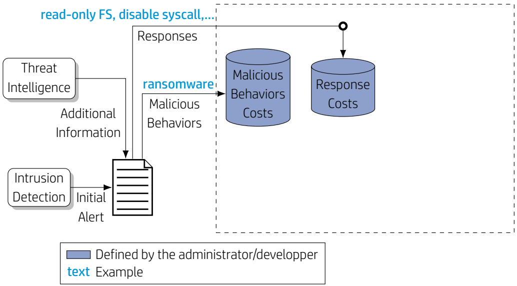
Malicious behaviors
- Availability violation=**moderate**
  - Consume system resources=**moderate**
    - Crack passwords=**moderate**
    - Mine for cryptocurrency=**moderate**
  - Compromise data availability=**moderate**
    - Compromise access to information assets=**moderate**
  - ...
- Command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

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

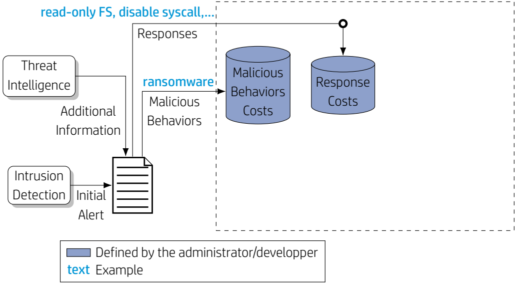
Costs

very low, low, **moderate**, high, very high, critical

Cost-Sensitive Response Selection



Cost-Sensitive Response Selection

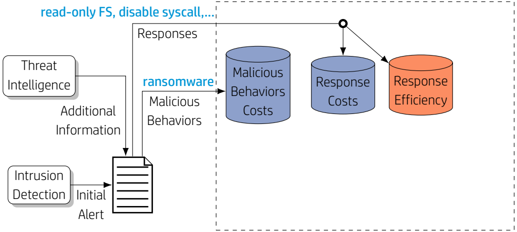


- Per-service responses
  - File system
    - Read-only file system
      - Read-only path
    - Inaccessible path
  - System calls
    - Blacklist any system call
      - Blacklist a list or a category of system calls
  - Network
    - Disable network
      - Blacklist IP addresses
      - Blacklist ports
      - ...
  - Resources
    - CPU quota
    - ...
  - ...

Example of a non-exhaustive per-service response hierarchy

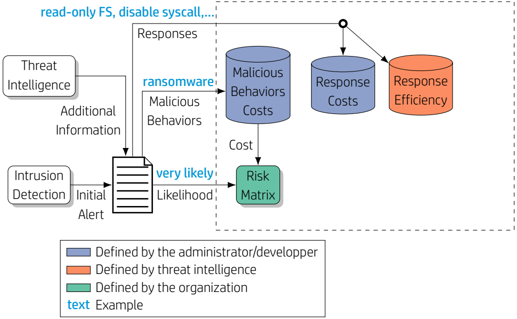
Responses may be provided via the exchange format STIX (e.g., the course of action field)

Cost-Sensitive Response Selection



Defined by the administrator/developper
Defined by threat intelligence
text Example

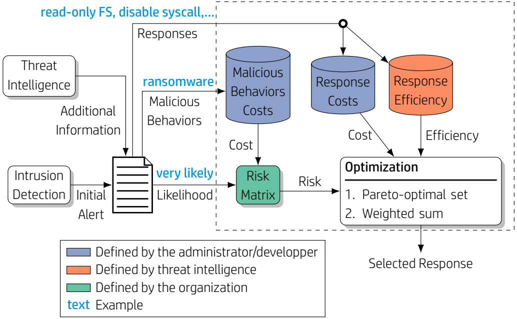
Cost-Sensitive Response Selection



Risk Matrix

Confidence (Likelihood)	Malicious Behavior Cost: Very low 0 – 0.2	Low 0.2 – 0.4	Moderate 0.4 – 0.6	High 0.6 – 0.8	Very high 0.8 – 1
Very likely 0.8 – 1	L	M	H	H	H
Likely 0.6 – 0.8	L	M	M	H	H
Probable 0.4 – 0.6	L	L	M	M	H
Unlikely 0.2 – 0.4	L	L	L	M	M
Very unlikely 0 – 0.2	L	L	L	L	L

Cost-Sensitive Response Selection



Cost vs Efficiency

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

$$\max\left(Risk \times Efficiency + (1 - Risk) \times (1 - Cost)\right)$$

Cost-Sensitive Response Selection

read-only FS, disable syscall,...

Responses

Cost vs Efficiency

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

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

$$\max\left(Risk \times Efficiency + (1 - Risk) \times (1 - Cost)\right)$$

Prototype Implementation for Linux-Based Systems

Projects Used or Modified

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

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,…)
- Linux.BitCoinMiner, Linux.Rex.1, Hakai, Linux.Encoder.1, GoAhead Exploit

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

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

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

$\rightarrow$ 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

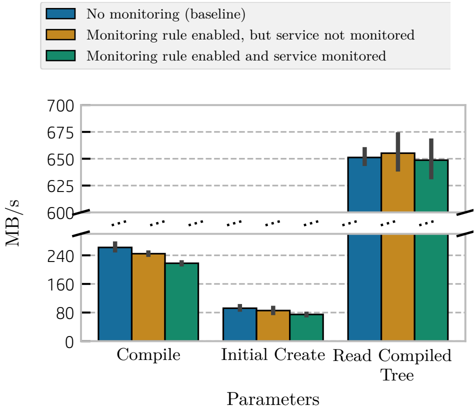
Performance Evaluation

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)

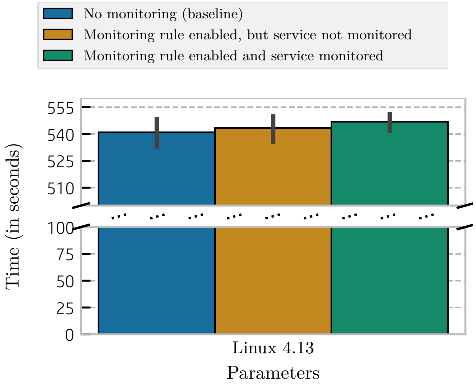
Performance Evaluation

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)

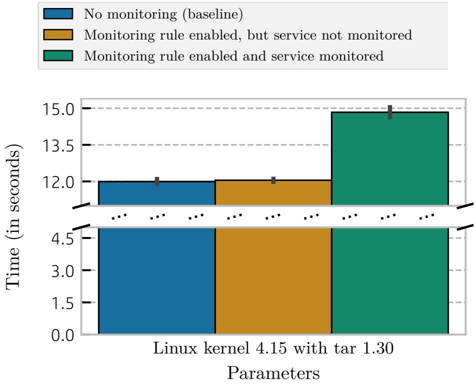
Performance Evaluation

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[6] has 8 % and 67 % overhead

[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

Ronny Chevalier, David Plaquin, Chris Dalton, and Guillaume Hiet. "Survivor: A Fine-Grained Intrusion Response and Recovery Approach for Commodity Operating Systems". Dec. 2019

Thanks for your attention!

Questions?

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

Ronny Chevalier, David Plaquin, Chris Dalton, and Guillaume Hiet. "Survivor: A Fine-Grained Intrusion Response and Recovery Approach for Commodity Operating Systems". Dec. 2019

References i

- Anderson, James P. *Computer Security Threat Monitoring and Surveillance*. Tech. rep. James P. Anderson Co., Fort Washington, PA. Apr. 1980. URL: http://seclab.cs.ucdavis.edu/projects/history/papers/ande80.pdf.
- Balepin, Ivan et al. "Using Specification-Based Intrusion Detection for Automated Response". In: *Recent Advances in Intrusion Detection*. 2003, pp. 136–154. DOI: 10.1007/978-3-540-45248-5_8.
- Denning, Dorothy E. "An Intrusion-Detection Model". In: *Proceedings of the 1986 IEEE Symposium on Security and Privacy* (Oakland, CA, USA). IEEE Computer Society, Apr. 1986, pp. 118–131. DOI: 10.1109/SP.1986.10010.
- Ellison, Robert J. et al. *Survivable Network Systems: An emerging discipline*. Tech. rep. Software Engineering Institute, Carnegie Mellon University, Nov. 1997. URL: https://apps.dtic.mil/dtic/tr/fulltext/u2/a341963.pdf.

References ii

- Goel, Ashvin et al. “The Taser Intrusion Recovery System”. In: *Proceedings of the 20th ACM Symposium on Operating Systems Principles* (Brighton, United Kingdom). SOSP ’05. 2005, pp. 163–176. DOI: `10.1145/1095810.1095826`.
- Knight, John C. and Elisabeth A. Strunk. “Achieving Critical System Survivability Through Software Architectures”. In: *Architecting Dependable Systems II*. Ed. by Rogério de Lemos, Cristina Gacek, and Alexander Romanovsky. 2004, pp. 51–78. ISBN: 978-3-540-25939-8.
- Morin, Benjamin and Ludovic Mé. “Intrusion detection and virology: an analysis of differences, similarities and complementariness”. In: *Journal in Computer Virology* 3.1 (Apr. 1, 2007), pp. 39–49. DOI: `10.1007/s11416-007-0036-2`.
- Shameli-Sendi, Alireza, Mohamed Cheriet, and Abdelwahab Hamou-Lhadj. “Taxonomy of Intrusion Risk Assessment and Response System”. In: *Computers & Security* 45 (Sept. 2014), pp. 1–16. DOI: `10.1016/j.cose.2014.04.009`.

Xiong, Xi, Xiaoqi Jia, and Peng Liu. "SHELF: Preserving Business Continuity and Availability in an Intrusion Recovery System". In: *Proceedings of the 25th Annual Computer Security Applications Conference*. ACSAC '09. IEEE Computer Society, 2009, pp. 484–493. DOI: `10.1109/ACSAC.2009.52`.