Resilience of Cyber-Physical Systems and Technologies

Zbigniew Kalbarczyk and Ravishankar K. Iyer

University of Illinois at Urbana-Champaign
Email: {kalbarcz, rkiyer}@illinois.edu
Depend Research Group
Building a system is hard...
But maintaining *Reliability & Security* is even... HARDER
Failures and Attacks are Inevitable

Source: An Executive’s Guide to 2013 Data Breach Trends, by Risk Based Security
Design for Resiliency

- A resilient system is expected to maintain an acceptable level of service in presence of internal and external disturbances.

- Design for resiliency is a multi-disciplinary task that brings together experts in security, fault tolerance, human factors, and others.

- Achieving resiliency requires mechanisms for efficient monitoring, detection, and recovery from failures due to malicious attacks and accidental faults with minimum negative impact on the delivered service.
While is this hard?

• **Design and assessment**
  
  – systems become untrustworthy due to a combination of: human failures, hardware faults, software bugs, network problems, and inadequate balance between the cyber and the physical systems e.g. the network and control infrastructures

• **Delivery of critical services**
  
  – cyber-physical systems (e.g., energy delivery, transportation, communications, Health Care) are expected to provide uninterruptable services

• **Interdependencies among systems**
  
  – resiliency of one system may be conditioned on availability of another system, e.g.,
    
    • resiliency of the transportation system may heavily depend on the robust operation of energy delivery infrastructure,
    
    • human-in-the-decision-loop – role of human intelligence in system remediation, service restoration and recovery
Our Approach: Continuous Monitoring

• **Coverage vs. Cost tradeoffs**
  – Detectability/Latency/Root of trust
  – Human/Resources

• **Methods**
  – Active vs. passive monitoring
  – Monitoring coordination
  – Automated reasoning
  – Domain aware techniques
Agenda

• **Leveraging power grid semantic**
  – Integrate power system analysis into network monitoring

• **Virtual machine monitoring**
  – Active vs. Passive

• **Probabilistic inference on security logs**
  – Monitor coordination
  – Automated reasoning
LEVERAGING POWER GRID SEMANTIC
Cyber Threats in Power Systems

- **SCADA (Supervisory Control And Data Acquisition) system**
  - Monitor and control geographically distributed assets in industrial control environment, e.g., power grid, gas pipeline, etc.
  - Modern SCADA systems integrate commercial computer systems and network
    - Compromise in control center, e.g., stolen credentials and software vulnerability
    - Compromise in substation, e.g., vulnerability in intelligent devices
Example Scenario of Control-related Attack

**Option 1:** attackers learn network topology, estimate system states, and determine attack strategies, e.g., which transmission lines to open.

**Option 2:** open lines at random when systems operate under high generation or load demands.

**Attack Entry Points**
- Access Control Center
- Data Historian
- State Estimation & Contingency Analysis
- Access Field Devices
- Installed Malware in Substations

**Attack Preparation Stage (offline)**

1. Generate legitimate but malicious network packets (a sample DNP3 packet to open 4 breakers simultaneously):
   - IP + TCP Headers
   - DNP3 Headers
   - Output Control Codes
   - Four Control Relay Objects

2. To hide system changes, intercept and/or alter the network packets sent to the control center in response to the commands.
Why Is Detection of Control-related Attacks a Challenge?

- Hard to detect based solely on power systems’ electrical states
  - Traditional contingency analysis considers low-order incidents, i.e., the “N-1” contingency
  - Traditional state estimation is performed periodically, detecting attacks after physical damage
  - Measurements may be compromised

- Hard to detect based solely on the network intrusion detection systems
  - Commands can be encoded in correct syntax
  - Not detectable by traditional network intrusion detection systems (IDS)
Detection Mechanism

• Combine system knowledge on both cyber and physical infrastructure in the power grid
  – Integrate network monitoring with look-ahead power flow analysis

• Detect malicious commands at their *first appearances*, instead of identifying power system’s physical damage after the fact
Example Approach: Adapting IDS for SCADA

Cyber Infrastructure

- Adapt specification-based IDS (e.g., Bro) for SCADA systems
  - Detect unexpected network activities based on deviation from security specifications, e.g., protocol definition

- Develop SCADA protocol (e.g., DNP3) analyzer and integrate with IDS system
  - Intercept SCADA commands at runtime
Example Approach: Bring Semantic Analysis

Physical Infrastructure

- Identify control commands from the network
- Invoke look-ahead power flow analysis
- Adapt power flow analysis algorithm for quick (low latency) detection
Evaluation: Detection Accuracy and Latency

- Very high detection accuracy: low false positive and false negative rate << 1%

- Low detection latency

Running on a PC with Intel i3 (3.07 GHz) quad-core and 8 GB memory and Ubuntu 12.04
Summary

**Attack Model**
- Control-related attack in the context of power grid

**Detection**
- Intercept commands
  - Use network analyzer for SCADA protocols (DNP3) and integrate it with the IDS
  - *Proactively* estimate commands’ execution consequences
    - Invoke rapid adaptive power flow analysis

**Response**
- Intrusion response:
  - use *reclosing logic* in modern relays
  - use software-defined networking technology (SDN) to allow flexible responses to attacks

**Evaluation**
- Simulation of power systems with different scales
- Detection performance, i.e., latency and accuracy
- Integrated simulation of SDN network and power system
VIRTUAL MACHINE MONITORING
x86 servers were virtualized in 2012

Source: 451 Research’s ThelInfoPro service reports

VM Monitoring Overview

Virtual Machine

Apps
OS

Hypervisor

- STRONG ISOLATION
- SEMATIC GAP
Continuous VM Monitoring

- Hypervisor
  - Hypertap
  - Hprobe

- App
- OS

- ✓ Root of trust: HW invariants
- ✓ Tamper-proofed
- ✓ Low runtime overhead

- ✓ Dynamic
- ✓ Supports both VM applications and OS
- ✓ Simple interface
- ✓ Flexible usage
Traditional VM Monitoring

Out-of-VM monitor is manipulated by in-VM attacker!

😭 Places trust on guest Operating System Invariants

😢 Polling monitoring - cannot capture VM’s operations
Hardware Invariant Approach

Virtual Machine

Process list

1

2

3

(Rootkit)

Context switch to process 1

Hardware*

load cr3, p1

Force VM Exit

Hypervisor

Notify VM Exit

Monitor

Inspect running processes

Inspect process 1

✓ Places trust on Hardware Architectural Invariants
✓ Event-driven monitoring

* x86 with Hardware Assisted Virtualization (HAV) enabled. CR3 holds the Page Directory Base Pointer (PDBP) of running process.
Hardware Invariant Approach

1. Places trust on **Hardware Architectural Invariants**
2. Event-driven monitoring

* x86 with Hardware Assisted Virtualization (HAV) enabled. CR3 holds the Page Directory Base Pointer (PDBP) of running processes.
## VM Monitoring via HW Invariants

<table>
<thead>
<tr>
<th>Event</th>
<th>Hardware* Invariants (x86)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Context switch</td>
<td>MMU, CR3 access</td>
</tr>
<tr>
<td>Thread/task switch</td>
<td>Page protection, TSS</td>
</tr>
<tr>
<td>System call</td>
<td>MSR, Exception</td>
</tr>
<tr>
<td>IO access</td>
<td>IO instructions, Interrupts</td>
</tr>
<tr>
<td>Memory access</td>
<td>Page protection, Exception</td>
</tr>
</tbody>
</table>

*Basis to support a wide range of failure & attack detections*

* x86 with Hardware Assisted Virtualization (HAV) enabled.
HyperTap Framework

- Prototyped in KVM
  - Small modification to KVM

- Auditors
  - Implement monitoring policies
  - Run as user processes on host user space
  - Grouped in a container (LXC) per VM

Audit containers in host user space

Container 1
- Auditor 1
- Auditor 2
- Auditor 3

Container 2

Virtual Machine 1
Event Forwarder
KVM Hypervisor

Virtual Machine 2

Event Multiplexer
Linux Kernel

Remote Health Checker

VM Exit event
Evaluation of HRKD (Hidden Rootkit Detection)

- Evaluated against real world rootkits on Windows and Linux
- All rootkits successfully detected

<table>
<thead>
<tr>
<th>Rootkit</th>
<th>Target OS</th>
<th>Hiding techniques</th>
</tr>
</thead>
<tbody>
<tr>
<td>FU</td>
<td>Windows XP, Vista</td>
<td>DKOM</td>
</tr>
<tr>
<td>HideProc</td>
<td>Windows XP, Vista</td>
<td>Hijack system calls</td>
</tr>
<tr>
<td>AFX</td>
<td>Windows XP</td>
<td>Hijack system calls</td>
</tr>
<tr>
<td>HideToolz</td>
<td>Windows Vista, 7</td>
<td>Hijack system calls</td>
</tr>
<tr>
<td>HE4Hook</td>
<td>Windows XP</td>
<td>Hijack system calls</td>
</tr>
<tr>
<td>BH</td>
<td>Windows XP</td>
<td>Hijack system calls</td>
</tr>
<tr>
<td>Enyelkm 1.2</td>
<td>Linux kernel 2.6</td>
<td></td>
</tr>
<tr>
<td>SucKIT</td>
<td>Linux kernel 2.6</td>
<td>Kmem, dkom</td>
</tr>
<tr>
<td>PhalanX</td>
<td>Linux kernel 2.6</td>
<td>DKOM</td>
</tr>
</tbody>
</table>

- Detection capability not affected by implementation or hiding techniques of the rootkits
- HRKD can detect future hidden rootkits regardless of their newly invented hiding mechanism
Evaluation of Privilege Escalation Detection (PED)

- Privilege Escalation Attack
- Detection
## Privilege Escalation Detection (PED)

<table>
<thead>
<tr>
<th>Ninja</th>
<th>Location</th>
<th>Description</th>
<th>Monitoring</th>
</tr>
</thead>
<tbody>
<tr>
<td>O-Ninja</td>
<td>In-VM</td>
<td>Original Ninja</td>
<td>Polling</td>
</tr>
<tr>
<td>H-Ninja</td>
<td>Out-of-VM</td>
<td>Uses OS invariants</td>
<td>Polling</td>
</tr>
<tr>
<td>HT-Ninja</td>
<td>Out-of-VM</td>
<td>Uses HW invariants (HyperTap)</td>
<td>Event-driven</td>
</tr>
</tbody>
</table>

**HT-Ninja** checks a process at context switches and IO system calls
• Transient attacks against polling monitoring

Laughing Face Icon: O-Ninja and H-Ninja are highly vulnerable to transient attacks
Smiling Face Icon: HT-Ninja uses event-driven monitoring and is not vulnerable to transient attacks
Performance Overhead

- Combined overhead < sum of individual overheads
- <2% overhead for CPU workloads
- <5% overhead for IO workloads
- Micro-benchmark:
  - Highest performance loss for NOOP system call (~19%)
# VM Monitoring Overview

<table>
<thead>
<tr>
<th>HyperTap (DNS'14)</th>
<th>LiveWire (CCS'03)</th>
<th>LibVM1 (ACSAC'07)</th>
<th>SIM (CCS'09)</th>
<th>Lycosid (ACM'08)</th>
<th>Antfarm (ACM'08)</th>
<th>Nitro (ATC'11)</th>
<th>Ether (CCS'08)</th>
<th>Osck (ASPLOS'11)</th>
<th>Virtuoso (SP'11)</th>
<th>VMST (SP'12)</th>
<th>TxIntro (HPCA'14)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Root-of-trust (invariant)</td>
<td>HW</td>
<td>OS</td>
<td>OS</td>
<td>OS</td>
<td>OS</td>
<td>OS</td>
<td>HW</td>
<td>HW</td>
<td>OS</td>
<td>OS</td>
<td>OS</td>
</tr>
<tr>
<td>Continuous/Poling Mon.</td>
<td>C</td>
<td>C</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>C</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>P</td>
</tr>
<tr>
<td>Changes to VM</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Custom Auditors</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Online Detection</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Auto-generate Invariants</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Userspace Monitoring</td>
<td>X</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
PROBABILISTIC INFERENCE ON SECURITY LOGS
Need for continuous and comprehensive monitoring
- Heterogeneous host and network-level logs

Use probabilistic graphical models as an inference framework
- Detection of progressing attacks
Integrating Heterogeneous Monitoring Data Using Probabilistic Graphical Models

USER STATES

benign suspicious malicious

EVENTS

LOGIN_REMOTELY OS_FINGERPRINT DOWNLOAD_SENSITIVE COMPILE RESTART_SYS_SERVICE

RAW LOGS

sshd: Accepted <user>
$ uname -a; w
$ wget bad: domain.com/vm.c
$ gcc vm.c -o a; ./a
sshd: Received SIGHUP; restarting.

Factor function

time
**Construct factor functions based on past incidents**

**Extract events corresponding to an incident**

**Construct per-user factor graph**

**Infer the user states**

*AttackTagger Workflow*

- Construction → Factor functions
- User events
- Events → User state
- Exact inference or Gibbs sampling:
  - benign
  - suspicious
  - malicious
Metrics: Detection timeliness & Preemption timeliness

- Detection timeliness
  - $t_0$ (The first event)
  - Detected by AttackTagger

- Preemption timeliness
  - $t_m$

- Detected by security analysts
  - $t_s$

- The last event
  - $t_n$
46 of 62 malicious users were detected in tested incidents (74%)

41 of 46 identified malicious users were identified before the system misuse

Detection timeliness & Preemption Timeliness

Percentage of events observed until attack detection
Conclusions

• **Design for resiliency** needs multi-disciplinary experts in security, fault tolerance, human factors

• **Achieving resiliency needs**
  
  – Application driven continuous monitoring and response to intrusions
  
  – Combination of knowledge on **cyber** and **physical** aspects of the system to devise protection while preserving system performance
  
  – Scientifically sound inference methods (and tools) to determine system/application state based on runtime observations on the system behavior