Analysis of Control Flow Events for Timing-based Runtime Anomaly Detection

Sixing Lu and Roman Lysecky
sixinglu@email.arizona.edu
rlysecky@ece.arizona.edu
“2010 prediction of 31 billion internet-connected devices by 2020 now seems an underestimate”

--- McAfee Labs Threats Report: August 2015
### Motivation

<table>
<thead>
<tr>
<th>Rank</th>
<th>Sector</th>
<th>Number of Incidents</th>
<th>Percentage of Incidents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Healthcare</td>
<td>116</td>
<td>37%</td>
</tr>
<tr>
<td>2</td>
<td>Retail</td>
<td>34</td>
<td>11%</td>
</tr>
<tr>
<td>3</td>
<td>Education</td>
<td>31</td>
<td>10%</td>
</tr>
<tr>
<td>4</td>
<td>Gov. &amp; Public Sector</td>
<td>26</td>
<td>8%</td>
</tr>
<tr>
<td>5</td>
<td>Financial</td>
<td>19</td>
<td>6%</td>
</tr>
<tr>
<td>6</td>
<td>Computer Software</td>
<td>13</td>
<td>4%</td>
</tr>
<tr>
<td>7</td>
<td>Hospitality</td>
<td>12</td>
<td>4%</td>
</tr>
<tr>
<td>8</td>
<td>Insurance</td>
<td>11</td>
<td>4%</td>
</tr>
<tr>
<td>9</td>
<td>Transportation</td>
<td>9</td>
<td>3%</td>
</tr>
<tr>
<td>10</td>
<td>Arts and Media</td>
<td>6</td>
<td>2%</td>
</tr>
</tbody>
</table>

**Top 10 Sectors Breached by Number of Incidents**

*ISTR20 Internet Security Threat Report, April 2015*
Timing based anomaly detection

- control flow sequence
- dataflow
- memory
- power
- timing

- Timing requirement is typically already an essential task within the design of embedded systems

- Significantly increases difficulty of mimicry attack

- Simple but robust model

- Creating and/or collecting timing is straightforward
**SHIELD**

- Monitors timing of individual basic blocks within software
- Effective against code injection attacks
- Incurs 6% performance overhead, and requires modifications to processor

---


SecureCore

- Monitors timing distribution of each basic block
- Inserts instructions to trace execution of basic blocks
- Classifies execution times with low probability as possible malware at design-time
- Achieves good detection rates, but suffer from a high false positive rate
- High memory overhead to store timing distribution, with 1000s of bins for each block

Software-based Detection:

- Inserting checkpoints in tasks change original system behavior
- Additional processor core is needed for most approaches
- Goal is typically on maximizing detection rate, not minimizing overhead

Hardware-based Detection:

- Hardware-based sequence detection methods exist, with no performance impact
- Approaches are directly integrated within processor, requiring custom processor designs
- Do not consider the timing behavior in detection method


• Directly interfaces with the trace port and monitors PC to detect events

• Verifies execution sequence and timing with normal specification

• No performance overhead

Model captures normal execution behavior (sequence and timing)

Select monitoring events to meet hardware constraints

Model with reduced monitoring events, will be configured into RAD Hardware
**Definition**

**Pseudo Code**

1. `exists_file`  
2. `mutex_lock`  
3. `file_open`  
4. `file_lseek`  
5. `gener_http_header`  
6. `tcp_write`  
7. `file_read`  
8. `tcp_write`  
9. `file_close`  
10. `mutex_unlock`  
11. `pbuf_free`

**Events**

- **Event**: a single instruction address or a pair of instruction addresses
- **Mimicry Attacks**: mimics the correct system execution
- **Nullified Event**: an function call / system call, substituting null arguments
- **Nullification Rate**:

\[
\gamma = \frac{\text{nullified events}}{\text{total events in mimicked sequence}}
\]
Runtime Security Model (RSM)

(a) RSM event

(b) Expected Sequence Matrix

\[
\begin{bmatrix}
e_{11} & e_8 & e_7 & e_5 & e_0 \\
e_0 & 0 & 0 & 0 & 1 & 0 \\
e_5 & 1 & 0 & 1 & 0 & 0 \\
e_7 & 1 & 1 & 0 & 0 & 1 \\
e_8 & 1 & 0 & 1 & 0 & 1 \\
e_{11} & 0 & 1 & 1 & 0 & 1 \\
\end{bmatrix}
\]
Runtime Security Model (RSM)

Expected Timing Matrix

Periodic/repeating timing requirement
Control Flow Event Selection

Pseudo Code

- exists_file
- mutex_lock
- file_open
- file_lseek
- gener_http_header
- tcp_write
- file_read
- tcp_write
- file_close
- mutex_unlock
- pbuf_free

Timing Distribution

- Probe Timing Distribution
- Minimum range selection
- Minimum variance selection
- Maximum peak probability selection
Experimental Setup

- **Benchmark**
  - A network-connected pacemaker
  - Prototyped on FPGA platform

- **Malware Attacks**
  - Mimicry attack that manipulates logs of cardiac activity
  - Different mimicry nullification rates

- **Experiments**
  - Apply three metrics to select events
  - Executed system 100 times to measure malware detection rate

---

Experimental Results

Nullification rate $\gamma = 0.4$

- 100% for all metrics
- Same for all metrics
- 88% for min range/variance metric
- Less than 40% for max peak probability metric
Nullification rate $\gamma = 0.6$

100% for min range/variance metric

Nullification rate $\gamma = 0.7$

88% for all metrics

Number of monitored events
Experimental Results

These events may not be good candidates for runtime detection.
Conclusions

• Present a formal runtime security model that defines the normal system behavior
• Evaluate three methods based on timing distribution to select monitoring events, given RAD hardware
• Minimum range and variance metrics achieve 100% detection rate when monitor 7 events in any nullification rate malware

Future Work

• Investigate methods to analyze changes in time distributions
• Develop analytical/statistical methods to identify events that are empirically harder to nullify
• Determine to what extent even small malicious changes can be detected by observing changes in time distribution
Thank you  dank u  Danke  Merci

Question?