SecureCore: A Multicore-based Intrusion Detection Architecture for Real-Time Embedded Systems

Man-Ki Yoon, Sibin Mohan, Jaesik Choi, Jung-Eun Kim, Lui Sha

Dept. of Computer Science, UIUC
Information Trust Institute, UIUC
Lawrence Berkeley National Lab

Apr 9th, 2013
Rethinking Real-Time Embedded System Security

- Increased Capability
- More Networked
- Open, Standard Platform

More Vulnerable to Security Attacks
SecureCore Architecture

**Intrusion Detection**, not prevention
- Most critical component: control application
- System recovery upon detection

**Behavior** monitoring
- Predictable **timing** behaviors of real-time apps
- Profile using statistical learning

**Multicore**-based core-to-core monitoring
- On-chip HW for processor state inspection
- Hypervisor-based protection/isolation
Rest of the Talk

• System and Application Model
• Timing-based Intrusion Detection (Overview)
• SecureCore
  – Architecture Design
  – Timing-based Intrusion Detection (Detail)
• Implementation and Evaluation
• Limitations and Future Work
System and Application Model

- Multicore-based Real-Time Control System

SecureCore: A Multicore-based Intrusion Detection Architecture for Real-Time Embedded Systems
System and Application Model

- Multicore-based Real-Time Control System

SecureCore Architecture

Physical plant

Sensor data

Controller

SecureCore

MonitoredCore

Threat Model: Malicious code execution
- Embedded in the control code
- Activated after system initialization
- Irrelevant how it gained entry
Timing-Based Intrusion Detection

- **Idea:** **Deterministic timing** of real-time applications
  - Any malicious activity consumes finite **time** to execute
  - Deviation from expected timing → **Suspicious**!

`SecureCore`: A Multicore-based Intrusion Detection Architecture for Real-Time Embedded Systems
Timing-Based Intrusion Detection

• **Idea:** *Deterministic timing* of real-time applications
  – Any malicious activity consumes finite *time* to execute
  – *Deviation* from expected timing → *Suspicious!*

---

**SecureCore:** A Multicore-based Intrusion Detection Architecture for Real-Time Embedded Systems
Timing-Based Intrusion Detection

• Idea: **Deterministic timing** of real-time applications
  – Any malicious activity consumes finite time to execute
  – **Deviation** from expected timing → **Suspicious**!

Statistical learning-based profiling/detection

• Profile **probabilistic** execution time model
  • Estimate **Prob(e*)**
  • Capture even **legitimate variations**
Outline

• System and Application Models
• Timing-based Intrusion Detection (Overview)
• SecureCore
  – Architecture Design
  – Timing-based Intrusion Detection (Detail)
• Implementation and Evaluation
• Limitations and Future Work
SecureCore Architecture

Monitored Core

- Complex Controller
  - Scratch Pad Memory
  - Timing Trace Module

Secure Core

- Secure Monitor
  - Decision Module
  - I/O Proxy
  - Inter-Core Communication

OS

Hypervisor

Plant

SecureCore: A Multicore-based Intrusion Detection Architecture for Real-Time Embedded Systems
Timing-Based Intrusion Detection

- **Block-level monitoring**
  - Narrowing estimation domain
    - Less variation, better accuracy
  - Block boundary: check point
    - Detect unexpected flow deviations
How to Get Timing Profiles

SecureCore: A Multicore-based Intrusion Detection Architecture for Real-Time Embedded Systems

Raw Traces → Trace Tree → Profiles

(Addr₁, t₁)
(Addr₂, t₂)
(Addr₃, t₃)
(Addr₄, t₄)
(Addr₅, t₅)
(Addr₆, t₆)
(Addr₇, t₇)
(Addr₈, t₈)
(Addr₉, t₉)
(Addr₁₀, t₁₀)
(Addr₁₁, t₁₁)
(Addr₁₂, t₁₂)
(Addr₁₃, t₁₃)
(Addr₁₄, t₁₄)
...

Statistical Learning
Timing Trace Module

main() {
    INST_REG_PID;
    ...
    INST_ENABLE_TRACE;
    ...
    foo();
    ...
    INST_DISABLE_TRACE;
}

foo() {
    INST_TRACE;
    Do_something();
    INST_TRACE;
    Do_something();
    INST_TRACE;
}

INST_REG_PID: rlwimi 0,0,0,0,1
INST_ENABLE_TRACE: rlwimi 0,0,0,0,2
INST_DISABLE_TRACE: rlwimi 0,0,0,0,3
INST_TRACE: rlwimi 0,0,0,0,4

Trace Instructions

- **PID registration** for preventing traces from being forged
- **BA: Base Address** (= PC of INST_REG_PID)

**SPM Layout**

<table>
<thead>
<tr>
<th>PID</th>
<th>BA</th>
<th>Addr&lt;sub&gt;Head&lt;/sub&gt;</th>
<th>Addr&lt;sub&gt;Tail&lt;/sub&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x000</td>
<td></td>
<td>Timestamp&lt;sub&gt;j+1&lt;/sub&gt;</td>
<td>Addr&lt;sub&gt;j+1&lt;/sub&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>0x010</td>
<td></td>
<td>Timestamp&lt;sub&gt;i&lt;/sub&gt;</td>
<td>Addr&lt;sub&gt;i&lt;/sub&gt;</td>
</tr>
<tr>
<td>0x8a0</td>
<td></td>
<td>Timestamp&lt;sub&gt;i+1&lt;/sub&gt;</td>
<td>Addr&lt;sub&gt;i+1&lt;/sub&gt;</td>
</tr>
<tr>
<td>0x8b0</td>
<td></td>
<td>Timestamp&lt;sub&gt;i+2&lt;/sub&gt;</td>
<td>Addr&lt;sub&gt;i+2&lt;/sub&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>0xFF0</td>
<td></td>
<td>Timestamp&lt;sub&gt;j&lt;/sub&gt;</td>
<td>Addr&lt;sub&gt;j&lt;/sub&gt;</td>
</tr>
</tbody>
</table>

**Timing Trace Module**

**Scratch Pad Memory**
Timing Trace Module

main() {
    INST_REG_PID;
    ...
    INST_ENABLE_TRACE;
    ...
    foo();
    ...
    INST_DISABLE_TRACE;
}

INST_TRACE
Do something();
INST_TRACE;

foo() {
    ...
    INST_TRACE;
    ...
}

INST_TRACE

 INST_REG_PID  rlwimi 0,0,0,0,1
 INST_ENABLE_TRACE  rlwimi 0,0,0,0,2
 INST_DISABLE_TRACE  rlwimi 0,0,0,0,3
 INST_TRACE  rlwimi 0,0,0,0,4

Trace Instructions

- Read **Timestamp** and **Program Counter** from the processor registers
- **Addr_i = BA − PC_i** (i.e., relative address from BA)

<table>
<thead>
<tr>
<th>PID</th>
<th>BA</th>
<th>Addr^{Head}</th>
<th>Addr^{Tail}</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x010</td>
<td></td>
<td>Timestamp j+1</td>
<td>Addr j+1</td>
</tr>
<tr>
<td>0x8a0</td>
<td></td>
<td>Timestamp i</td>
<td>Addr i</td>
</tr>
<tr>
<td>0x8b0</td>
<td></td>
<td>Timestamp i+1</td>
<td>Addr i+1</td>
</tr>
<tr>
<td>0x8c0</td>
<td></td>
<td>Timestamp i+2</td>
<td>Addr i+2</td>
</tr>
<tr>
<td>0xFF0</td>
<td></td>
<td>Timestamp j</td>
<td>Addr j</td>
</tr>
</tbody>
</table>

SPM Layout

- Timing Trace Module
- Scratch Pad Memory

[Image of timing trace module and its components: Timing Trace Module, Scratch Pad Memory, Timing Trace Module diagram with trace instructions and SPM layout table.]

Raw Traces

(Addr_1, t_1)
(Addr_2, t_2)
(Addr_3, t_3)
(Addr_7, t_4)
(Addr_1, t_5)
(Addr_2, t_6)
(Addr_4, t_7)
(Addr_6, t_8)
(Addr_7, t_9)
(Addr_1, t_{10})
(Addr_2, t_{11})
(Addr_4, t_{12})
(Addr_5, t_{13})
(Addr_7, t_{14})
...

SecureCore: A Multicore-based Intrusion Detection Architecture for Real-Time Embedded Systems
Trace Tree

(Addr₁, t₁)
(Addr₂, t₂)
(Addr₃, t₃)
(Addr₇, t₄)
(Addr₁, t₅)
(Addr₂, t₆)
(Addr₄, t₇)
(Addr₆, t₈)
(Addr₇, t₉)
(Addr₁, t₁₀)
(Addr₂, t₁₁)
(Addr₄, t₁₂)
(Addr₅, t₁₃)
(Addr₇, t₁₄)
...

Addr₁
   Block 1
     t₂-t₁
     t₆-t₅
     t₁₁-t₁₀

Addr₂
   Block 2
     t₃-t₂
     ...

Addr₃
   Block 2
     t₄-t₃
     ...

Addr₄
   Block 3
     t₇-t₆
     t₁₂-t₁₁

Addr₅
   Block 4
     t₁₃-t₁₂
     ...

Addr₆
   Block 5
     t₈-t₇
     ...

Addr₇
   Block 6
     t₁₄-t₁₃
     ...

i l l i n o i s . e d u

SecureCore: A Multicore-based Intrusion Detection Architecture for Real-Time Embedded Systems
From a trace tree, we can get
- Execution time samples (each node)
- Legitimate execution flows

Same execution block, but on different paths.
Each has its own timing profile

(Addr₁, t₁)
(Addr₂, t₂)
(Addr₃, t₃)
(Addr₄, t₄)
(Addr₅, t₅)
(Addr₆, t₆)
(Addr₇, t₇)
(Addr₈, t₈)
(Addr₉, t₉)
(Addr₁₀, t₁₀)

• Executions samples (each node)
• Legitimate execution flows

...
Timing Profile

• What is a good estimation of execution times?
  – Min & max, mean, ...
    • Not representative
    • Cannot capture variations well

  – Probabilistic timing model
    • Estimate the likelihoods of execution times!
      – Probability distribution
    • Parametric vs. Non-parametric distribution
      – Unknown shape
Execution Time Profile Using Kernel Density Estimation (KDE)

- Non-parametric Probability Density Function Estimation

1. Given samples of execution times
2. Draw scaled distribution at each sample point
3. Sum them up

\[
\hat{f}_h(e|e^{(1)}, \ldots, e^{(m)}) = \frac{1}{m} \sum_{i=1}^{m} K_h(e - e^{(i)})
\]

- Kernel & bandwidth affect **shape** and **smoothness**
- Gaussian kernel
Intrusion Detection Using Timing Profiles

PDF of the Execution Time of an example block

How much deviation should we consider malicious?

Threshold test

\[ \text{Prob}(e^{\uparrow*}) < \theta \rightarrow \text{Malicious} \]
\[ \text{Prob}(e^{\uparrow*}) \geq \theta \rightarrow \text{Legitimate} \]

• E.g., \( \theta = 0.01 \) or 0.05
• At least \( \theta \) of measurements were close to \( e^{\uparrow*} \)

Multiple peaks: different inputs or system effects

SecureCore: A Multicore-based Intrusion Detection Architecture for Real-Time Embedded Systems
Summary of Timing-Based Intrusion Detection

Monitored Core

Complex Controller

Secure Core

Timing Trace Module

Scratch Pad Memory

[Run-time Execution]

[Profile]

Trace

(Addr₁, tᵢ)
(Addr₂, tᵢ₊₁)
(Addr₄, tᵢ₊₂)
(Addr₆, tᵢ₊₃)
(Addr₇, tᵢ₊₄)

Traverse and check

SecureCore: A Multicore-based Intrusion Detection Architecture for Real-Time Embedded Systems
Outline

• System and Application Models
• Timing-based Intrusion Detection (Overview)
• SecureCore
  – Architecture Design
  – Timing-based Intrusion Detection (Detail)

• Implementation and Evaluation
• Limitations and Future Work
Implementation

**Freescale P4080 on Simics**
- Only two cores (Core 0 and 1)
- Cache (L1 and L2) and bus models for system effects
- ISA modification for trace instruction
Implementation

Inverted Pendulum Control
- **Controller** and **dynamics** (cart position, rod’s angle)
- Generated from Simulink IP model
Application Model

- **IP Control + FFT (EEMBC)**

  - **FFT Init**
  - **FFT Phase #1**
    - PathID = 1, 2
  - **FFT Phase #2**
    - 1 run if PathID = 0, 1
    - 2 runs if PathID = 2
  - **FFT Phase #3**
  - **IP Control**

  - **Malicious code**
    - Injected at the end of **FFT Phase #3**
    - **Simple loop** (some array copy)
      - 440, 720, 1000 cycles for 1,3,5 loops
      - (FFT Phase#3: ~260,000 cycles)
    - Activated when the cart passes +0.7 m
    - Execute randomly thereafter
      - Loop execution
      - Sends old actuation cmd

  - **Timing Profile**
    - ~10,000 runs (no malicious code activation)
    - ‘ksdensity’ (Matlab) for Gaussian KDE

  - **Total exec time**: 850,000 ~ 1,200,000 cycles (~1ms)
  - **Control period**: 10 ms

*securecore.cs.illinois.edu*
Early Detection

\[ \theta = 0.01 \text{ (1\%)} \]

Loop count: 3 (~ 720 cycles)

- Simplex only
- Attack activated
- No attack
- Our method
- No protection
Intrusion Detection Accuracy

• **Criteria**: False prediction rates
  – **False positive**: predict “malicious” when not
  – **False negative**: fail to detect a real attack

<table>
<thead>
<tr>
<th></th>
<th>False positive rates</th>
<th>False negative rates</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1/1024 (0.10%)</td>
<td>827/1022 (81%)</td>
</tr>
<tr>
<td></td>
<td>7/1015 (0.69%)</td>
<td>574/1046 (55%)</td>
</tr>
</tbody>
</table>

**Trade off**: Low $\theta$? High $\theta$?

- Detect well
- More false alarms

- Miss often
- Fewer false alarms

**SecureCore**: A Multicore-based Intrusion Detection Architecture for Real-Time Embedded Systems
Limitations and Future Work

• **Limitations**
  - Low detection accuracy for short malicious code
    → More deterministic execution
  - Still high false positive
    → Long-term monitoring

• **Other future work**
  - Monitoring multiple applications on multiple cores
  - Monitoring of other behavioral aspects (e.g., Memory, I/O)
  - Multi-dimensional monitoring
Thank you