Supporting A Dynamic Program Signature: An Intrusion Detection Framework for Microprocessors

Koji Inoue
Department of Informatics, Kyushu University
Japan Science and Technology Agency
Background

- Trusted Program
- Malicious Program

- Branch Prediction
- Selective Activation
- Pipeline
- Value Prediction
- Super Scalar
- TLP

High Performance

- OOO Exe.
- MLP
- On-chip Cache

Low Power/Energy

- Resizing
- Drowsy Operation
- Clock Gating
- DVS
- Signal Gating

MLP
TLP
ILP
The Goal of This Research

Architectural Support for Security

Run-Time Program Authentication

High Performance
- SuperScalar
- ILP
- OOO Exe.
- Pipelining
- Value Prediction
- On-chip Cache
- MLP

Low Power/Energy
- Selective Activation
- Signal Gating
- Clock Gating
- DVS
- Resizing
- Drowsy Operation
Outline

- Introduction
- Processor Level Intrusion Detection (Conventional)
- Supporting A Dynamic Signature
- Evaluation
  - Security Strength
  - Performance and Cost Overhead
- Conclusions
What Is The Problem of Current Intrusion Detection? (1/2)

- **Search Malicious Programs**
  - Static Virus Scan (Pattern Matching)
  - Dynamic Virus Scan (Rule Matching)

  **Problems**
  - Require Virus Lists
  - Impossible to Detect Unknown Virus

- **Allow to Execute Only Authenticated Programs**
  - Static Certificate (w/ Key)
  - Dynamic

  **Problems**
  - Corruption of key check programs
  - Hijack of the program execution control of authenticated programs
What Is The Problem of Current Intrusion Detection? (2/2)

- Dynamic Program Authentication [Wanger’01]
  - Extract a Rule (or Behavior) From the Program Code
  - Check the Rule at Run Time

- Problems
  - Limits of The Extracted Rule
  - Can NOT Detect Viruses Which Have The Same Rule

---

Outline

- Introduction
- Processor Level Intrusion Detection (Conventional)
- Supporting A Dynamic Signature
- Evaluation
  - Security Strength
  - Performance and Cost Overhead
- Conclusions
Concept of Dynamic Program Signature

- Determine an execution behavior for the program authentication from a secret key
  - E.g., Memory-Access Pattern
- Control the execution behavior at compile time
- Monitor the execution behavior at run time
  - E.g., Hardware Profiler

Program → compile → object code

Secret Key → Behavior Model → CPU
Dynamic Execution Behavior for Program Authentication

Address $X$ is Accessed in Every $N$ Instructions!
How Can We Control The Execution Behavior At Compile Time?

- **Branches**
  - Unify the size of basic blocks

- **Speculative/OOO Execution**
  - Monitor the sequence of committed instructions

A special instruction to generate a dynamic signature
Outline

- Introduction
- Processor Level Intrusion Detection (Conventional)
- Supporting A Dynamic Signature
- Evaluation
  - Security Strength
  - Performance and Cost Overhead
- Conclusions
Security Strength

We can NOT detect malicious attacks if

- The attacker knows the secret key information and algorithm how to determine the pre-defined behavior
- The attack codes are inserted at the compile time
- The attacker can predict the pre-defined behavior
  - Provability: $\frac{1}{2^{\text{ADDRbit-width}}+2^N}$ where N is the candidate of the unified basic-block size
- The size of malicious code is smaller than that of unified basic-block
  - E.g., less than 5 or 30 instructions
Performance/Cost Overhead (1/2)

- Evaluation
  - Code size (Cost)
  - Execution time (Performance)

- Experimental Environment
  - ARMSimplescalar
    - StrongARM model
    - Support the dynamic signature
  - Benchmark programs
    - SPECint95 (129.compress)
  - Dynamic Signature
    - Unified basic-block size: from 5 to 25 instructions
    - #of key-load instructions in each basic block: 1
Performance/Cost Overhead (2/2)

- Code size
  - x5 in maximum, and x1.7 in minimum
- Execution time overhead
  - 50% in max. and 10% in min.
Outline

- Introduction
- Processor Level Intrusion Detection (Conventional)
- Supporting A Dynamic Signature
- Evaluation
  - Security Strength
  - Performance and Cost Overhead
- Conclusions
Conclusions

Summary
- Supporting Dynamic Program Signature
  - Run-time execution authentication
  - Detect the execution of malicious programs
- Overhead Evaluation (unified BB size = 5 inst.)
  - Code size: about x1.7
  - Execution Time: about x1.1

Future Work
- Support secure compilation
- Explore the tradeoff between security and cost, performance, and also energy consumption