Unsupervised Anomaly-based Malware Detection Using Hardware Features

Adrian Tang
Simha Sethumadhavan
Salvatore Stolfo

RAID, September 2014, Gothenburg, SE
Detecteding Malware

- Malware is a significant problem
- Prior works - Focus on higher-level features

Abstraction Level of Features
- High-level actions
- Content semantics
- Payload syntax
- Function calls
- Syscalls
- Architectural

Software

Hardware
Detecting Malware

- Malware is a significant problem
- This work - Use **lower-level** features to detect exploits

**Abstraction Level of Features**

**Hardware**
- CPU
- Branch Prediction Unit
- TLB
- Instruction Cache
- Data Cache
- L2 Cache

**Software**

**Microarchitectural**
Detecting Malware

- Malware is a significant problem
- This work - Use lower-level features to detect exploits

Abstraction Level of Features

Advantages:
- Harder to evade
- Cheap to collect
- Minimal features
Key Idea

Per-process HPC measurements over time

- Event 1
- Event 2
- Event 3
- Event 4

Hardware Performance Counters (HPCs)
Unsupervised Anomaly-based Malware Detection Using Hardware Features
A. Tang, S. Sethumadhavan, S. Stolfo. RAID 2014

Key Idea

Hardware Performance Counters (HPCs)

Can we detect anomalies caused by malware exploits on attacked programs ... at a lower level, stripped of semantic info?

Exploits perturb μArch behavior of attacked programs

Per-process HPC measurements over time

Malware exploit begins to execute
Outline

- Related Work
- System & Methodology
- Results
- Future Work
Related Work

Abstraction Level Of Features (Y-Axis)

- High-level actions
- Content semantics
- Payload syntax
- Function calls
- Syscalls
- Architectural
- Microarchitectural

Detection Approach (X-Axis)

- Signature/Misuse-based
- Anomaly-based

Lots of prior work

This work

- Android malware detection [Demme13]
- Windows
- e:
- f:
- PDF
Outline

• Related Work

• System & Methodology

• Results

• Future Work
Methodology Overview

Collection

Feature selection

Feature extraction

Model building

Detection

Remediation

Collection infrastructure

Labeling data from malware
Collection Infrastructure

Host to be protected

Applications

HPCs

Client Agent

Kernel Driver

TCP

Training/Detection System

Server Agent

Measurements + PID

Training

Detection
Labeled Data attributed to Malware

• Labeled measurements when malware executes
  - To inform feature selection
  - For testing and evaluation

• Typical malware exploits infect in stages
  1) Code Reuse Shellcode (ROP)
  2) Stage1 Shellcode (Stage1)
  3) Stage2 Payload (Stage2)

• Use Metasploit to generate exploit samples
  - For labels, instrument the boundaries of the stages using $0xcc$
  - Introduce variations for each stage across the samples
Methodology Overview

Collection → Feature selection → Feature extraction → Model building → Detection → Remediation

Power transform → Fisher score
Feature Selection

Challenge #1: Perturbations caused by malware are small

Our approach:
- Use rank-preserving Power Transform
- For each event $i$, find the appropriate power parameter $\lambda_i$ s.t. the normalized median for clean data is within tolerance $\epsilon$ of 0.5

- Positively-scaled measurements magnify any minute perturbations caused by malware
Feature Selection

Challenge #1: Perturbations caused by malware are small

Unsupervised Anomaly-based Malware Detection Using Hardware Features

A. Tang, S. Sethumadhavan, S. Stolfo. RAID 2014
Feature Selection

Challenge #2: Limited to monitoring up to 4 events at a time

• We want:
  - Shortlist sets of 4 events each that can best distinguish different malware stages from the normal code runs

• Our approach:
  1) Shortlist 19 events based on past work
  2) Pick events with higher discriminative power
## Feature Selection

**Challenge #2:** Limited to monitoring up to 4 events at a time

1) Shortlist 19 events based on past work and informed understanding of malware behavior

<table>
<thead>
<tr>
<th>Architectural Events</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
<td><strong>Event Description</strong></td>
</tr>
<tr>
<td>LOAD</td>
<td>Load instructions (ins.)</td>
</tr>
<tr>
<td>STORE</td>
<td>Store ins.</td>
</tr>
<tr>
<td>ARITH</td>
<td>Arithmetic ins.</td>
</tr>
<tr>
<td>BR</td>
<td>Branch (br.) ins.</td>
</tr>
<tr>
<td>CALL</td>
<td>All near call ins.</td>
</tr>
<tr>
<td>CALL_D</td>
<td>Direct near call ins.</td>
</tr>
<tr>
<td>CALL_ID</td>
<td>Indirect near call ins.</td>
</tr>
<tr>
<td>RET</td>
<td>Near return ins.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Microarchitectural Events</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
<td><strong>Event Description</strong></td>
</tr>
<tr>
<td>LLC</td>
<td>Last level cache references</td>
</tr>
<tr>
<td>MIS_LLCC</td>
<td>Last level cache misses</td>
</tr>
<tr>
<td>MISP_BR</td>
<td>Mispredicted br. ins.</td>
</tr>
<tr>
<td>MISP_RET</td>
<td>Mispred. near return ins.</td>
</tr>
<tr>
<td>MISP_CALL</td>
<td>Mispred. near call ins.</td>
</tr>
<tr>
<td>MISP_BR_C</td>
<td>Mispred. conditional br.</td>
</tr>
<tr>
<td>MIS_ICACHE</td>
<td>iCache misses</td>
</tr>
<tr>
<td>MIS_ITLB</td>
<td>iTLB misses</td>
</tr>
<tr>
<td>MISP_BR_D</td>
<td>D-TLB load misses</td>
</tr>
</tbody>
</table>
Feature Selection

Challenge #2: Limited to monitoring up to 4 events at a time

2) Pick events with high Fisher Score (F-Score)
   - Collect measurements from clean and exploit runs
   - Compute 3 F-Scores for each event
   - Rank the event F-Scores for each malware stage
   - Shortlist 9 most discriminative event sets
Methodology Overview

Collection → Feature selection → Feature extraction → Model building → Detection → Remediation

Temporary modeling
One-class Support Vector Machines

Unsupervised Anomaly-based Malware Detection Using Hardware Features
A. Tang, S. Sethumadhavan, S. Stolfo. RAID 2014
Building Unsupervised Models

• Feature extraction
  1) Non-temporal: A sample spans over 1 time epoch ($X$ executed insn.)
  2) Temporal: A sample spans over $N$ time epochs

• One-Class Support Vector Machine (oc-SVM)
  - Unsupervised: Train using data only from clean runs
  - Non-linear: Radial Basis Function (RBF) kernel
  - Tunable: Modify libSVM to produce a numerical decision function output instead of classification

• Evaluate using hold-out measurements from clean runs and exploit runs
  - Receiving Operating Characteristics (ROC) curves
  - Area Under Curve (AUC) scores
Methodology Overview

Collection → Feature selection → Feature extraction → Model building → Detection → Remediation

Use models to detect exploits
Outline

• Related Work

• System & Methodology

• Results

• Future Work
Results – Different Malware Stages

![Graph showing AUC scores for different malware stages](image)

- Event set

ROC curves for different sets:
- ROC for Set <AM-0>
- ROC for Set <AM-1>
- ROC for Set <AM-2>
Results – Different Malware Stages

![Graph showing AUC scores for different malware stages and event sets.](image)

- **AUC score** for each malware stage is plotted.
- **Event set** categories include non-temporal and temporal.
- **ROC curves** for different malware sets (AM-0, AM-1, AM-2) are shown.

---

Unsupervised Anomaly-based Malware Detection Using Hardware Features
A. Tang, S. Sethumadhavan, S. Stolfo. RAID 2014
Results – Different Malware Stages

Models perform best in detection of Stage1 shellcode

Better detection with the temporal modeling approach

Mediocre detection performance for ROP shellcode
Results – Arch vs \( \mu \)Arch Events

Arch-only (A-*) models perform better than \( \mu \)Arch-only (M-*) models

Combining the use of both Arch and \( \mu \)Arch events in (AM-*) models achieves better detection performance
Results – Detection vs Sampling Overhead

Sampling Overhead

Detection Performance

Coarser-grained sampling rate → Lower sampling overhead
→ Lower detection performance

Gains from lower sampling overhead far outstrips the deterioration of detection performance
Outline

• Related Work

• System & Methodology

• Results

• Future Work
Future Work

• Defense-in-depth
  - Investigate and quantify the multiplicative defensive effects of combining different sensors using higher-level and lower-level features

• Out-of-VM deployment in a Virtual Machine Introspection (VMI)-based setting for cloud environments
  - Minimal guest data structures → Less need to bridge semantic gap

• Further hardware support
  - Additional security counters
  - Separate and dedicated core or co-processor for online detector
Concluding Remarks

• First anomaly-based malware detector using lower-level μArch features from HPCs to detect malware exploits

• Adding μArch features to Arch ones improves detection of anomalies exhibited by exploit shellcode execution

• (More in paper...) Analyze the impact and difficulty of evasion attacks
  - In order to evade detection, exploit crafting becomes a delicate and precise ‘balancing’ act

Thank you!