Unsupervised Anomaly-based Malware Detection Using Hardware Features

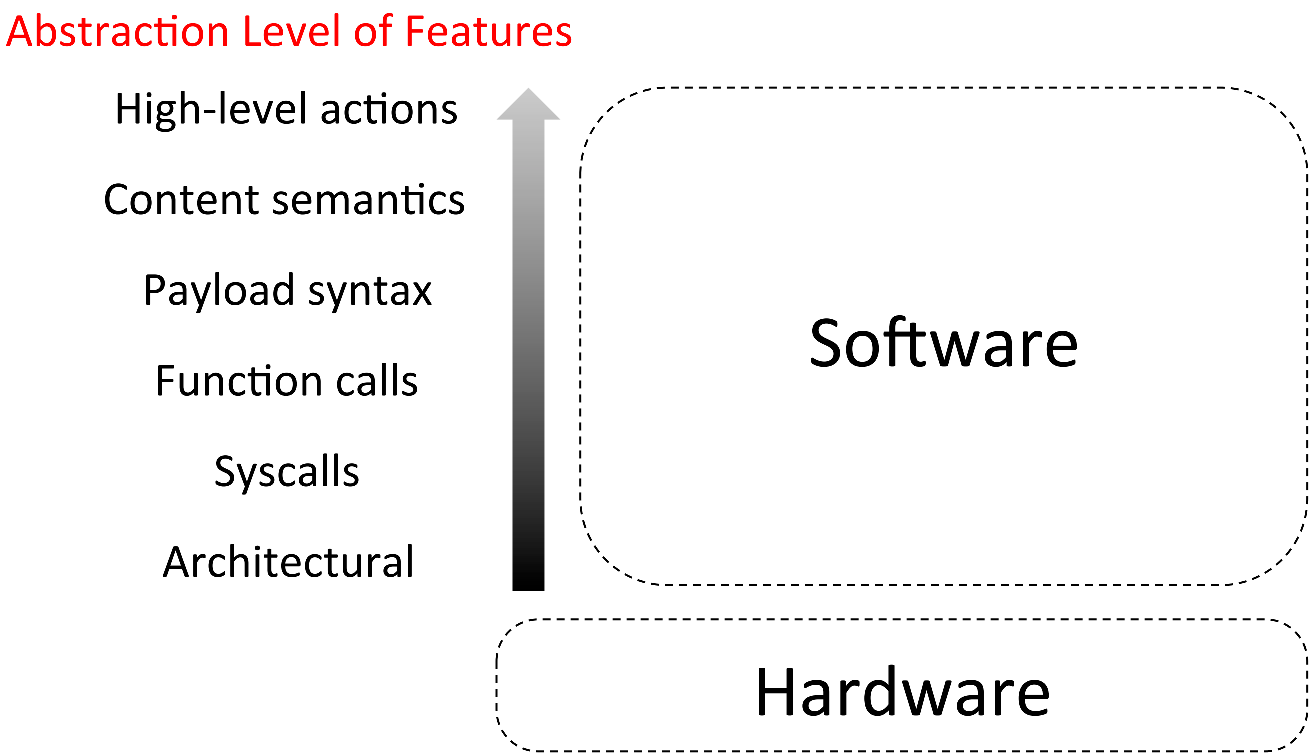
Adrian Tang
Simha Sethumadhavan
Salvatore Stolfo

RAID, September 2014, Gothenburg, SE

COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK

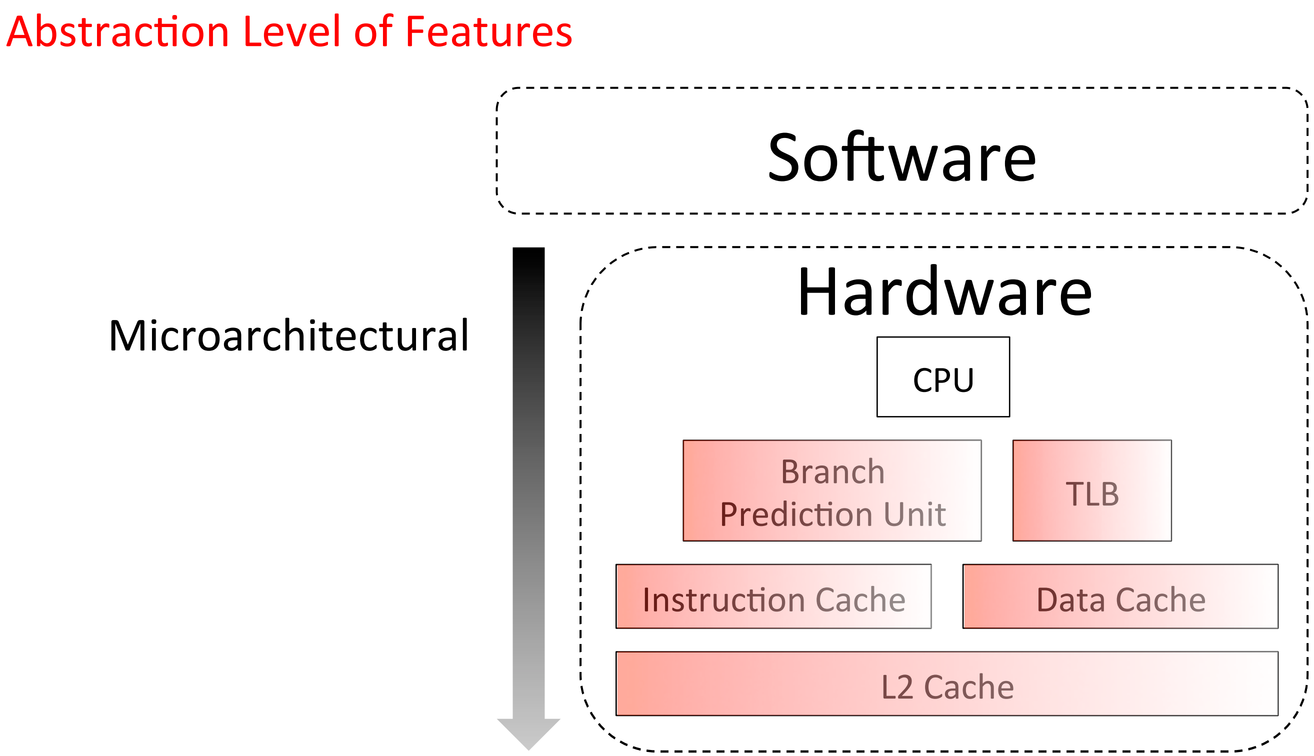
Detecting Malware

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



Detecting Malware

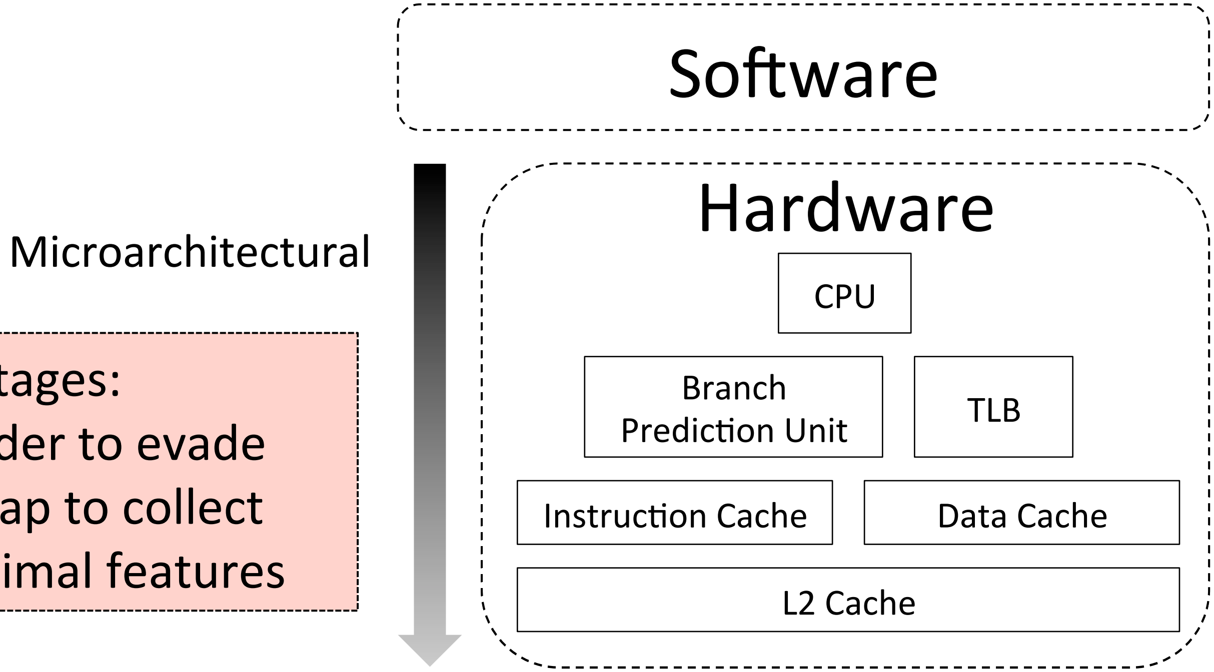
- Malware is a significant problem
- This work - Use lower-level features to detect exploits



Detecting Malware

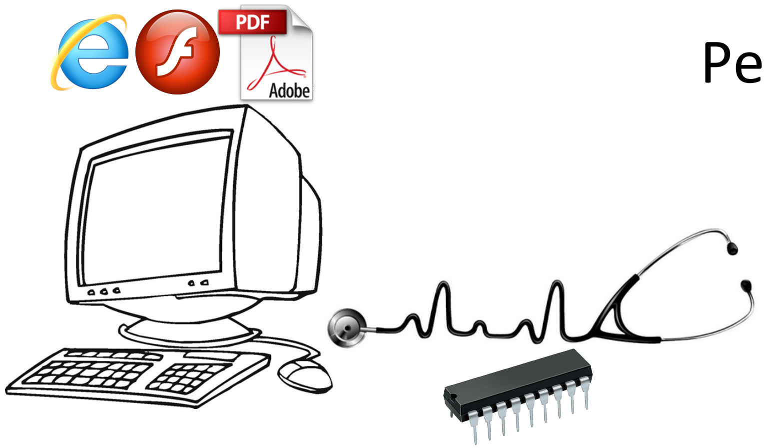
- Malware is a significant problem
- This work - Use <u>lower-level</u> features to detect exploits

Abstraction Level of Features



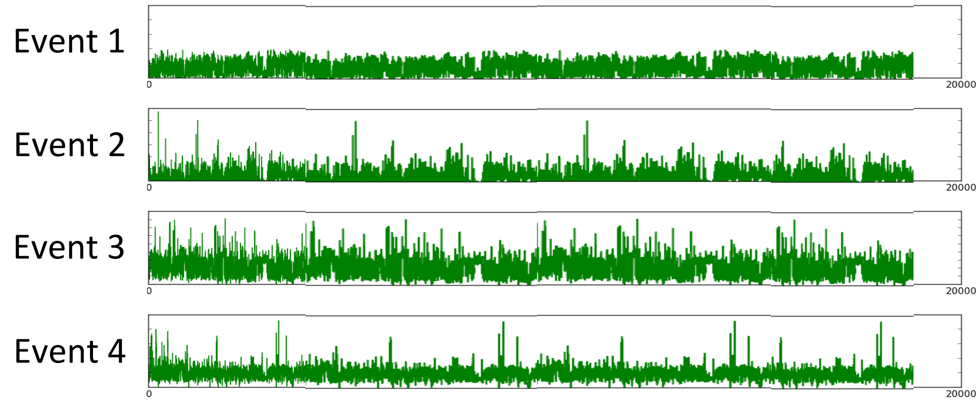
Advantages:
- Harder to evade
- Cheap to collect
- Minimal features

Key Idea

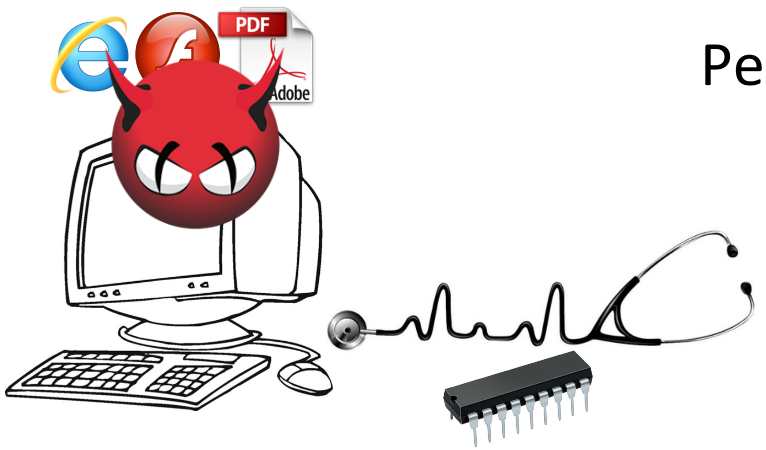


Hardware Performance Counters (HPCs)

Per-process HPC measurements over time



Key Idea



Hardware Performance Counters (HPCs)

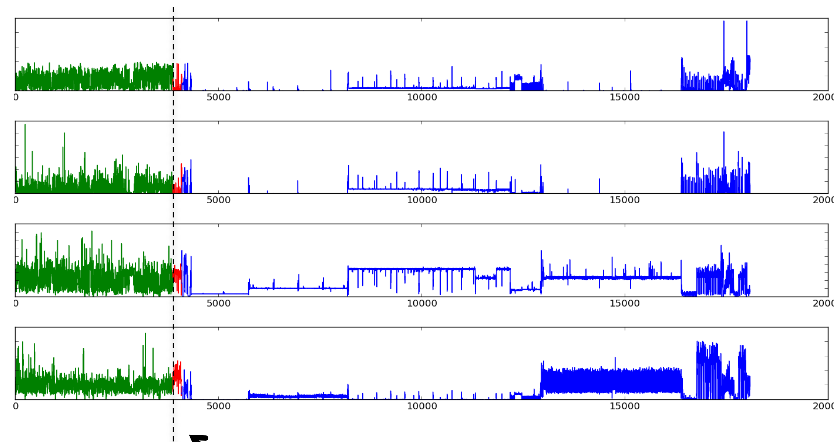
Per-process HPC measurements over time

Event 1

Event 2

Event 3

Event 4



Malware exploit begins to execute

Exploits perturb μArch behavior of attacked programs

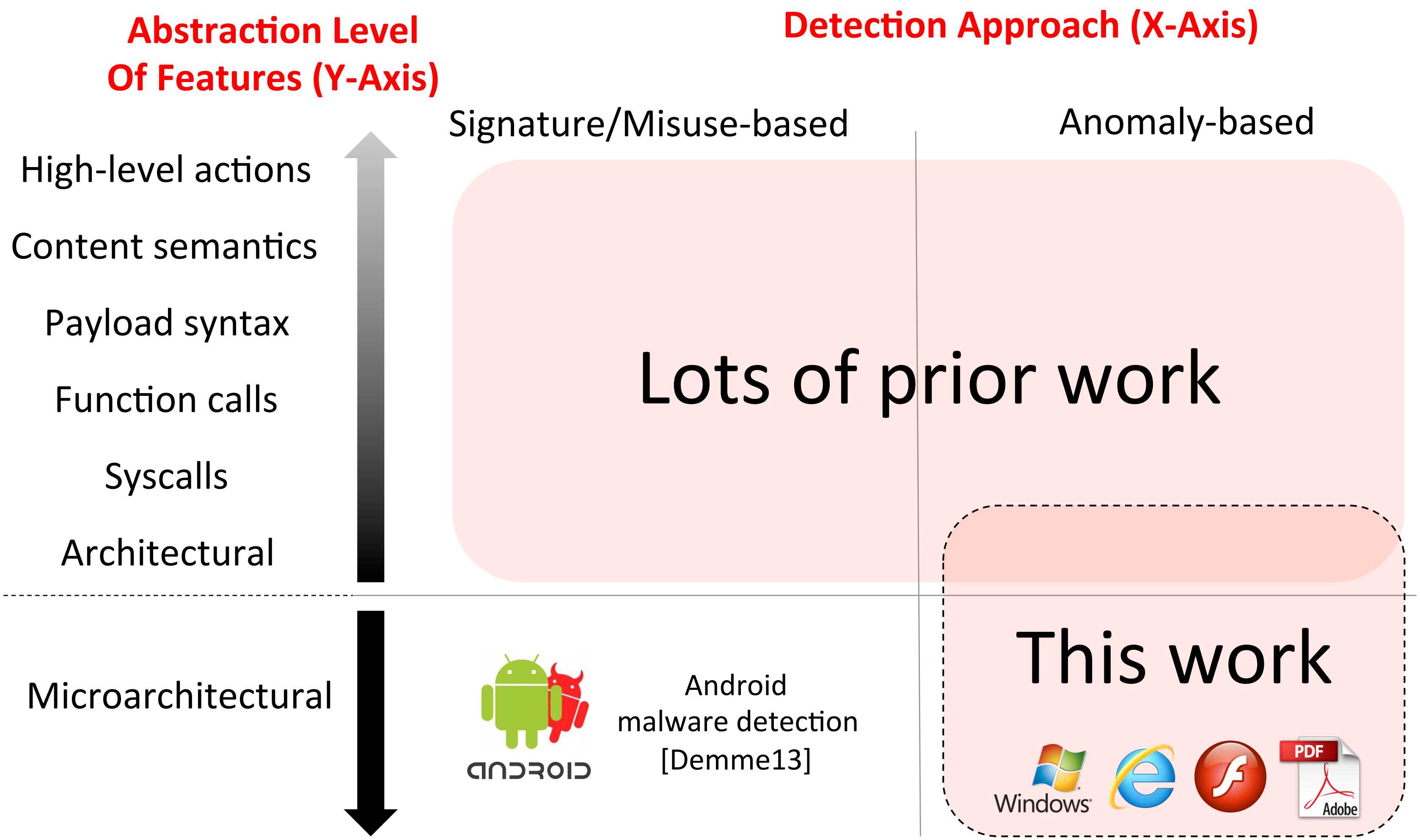
Can we detect anomalies caused by malware exploits on attacked programs …

at a lower level, stripped of semantic info?

Outline

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

Related Work



Outline

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

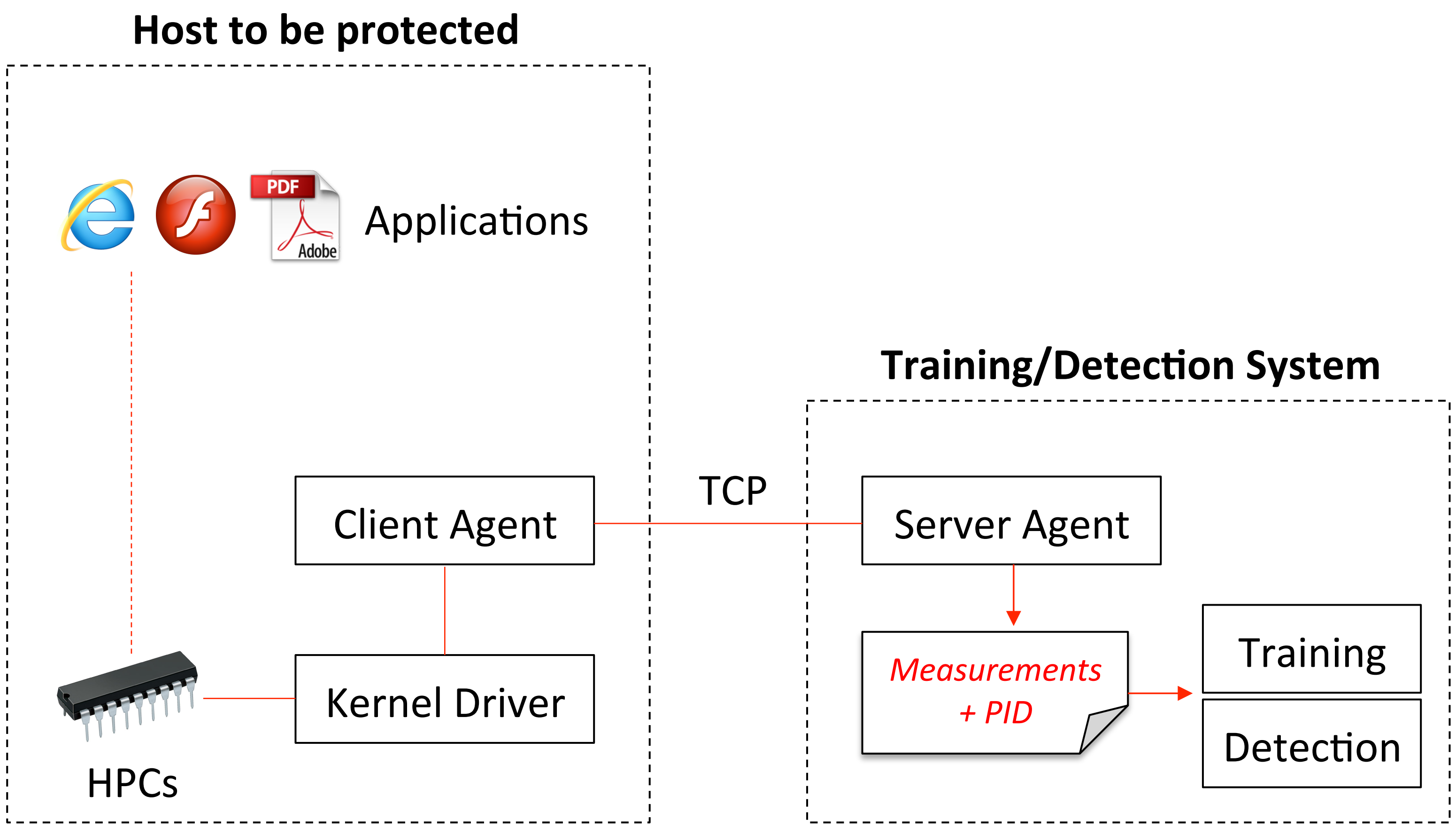
Methodology Overview

Collection
Feature selection
Feature extraction
Model building
Detection
Remediation

Collection infrastructure

Labeling data from malware

Collection Infrastructure



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

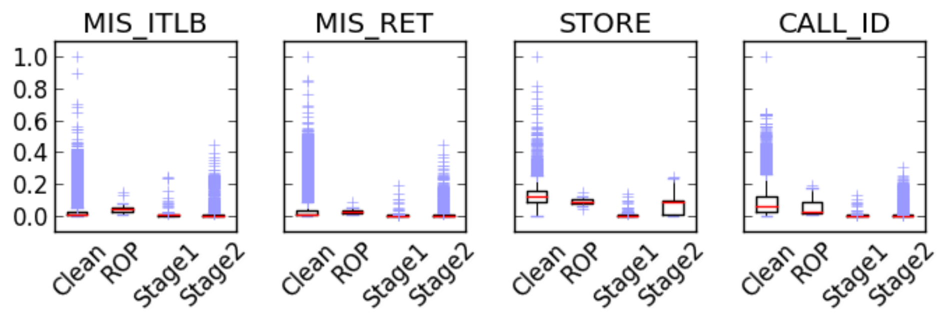
Remediation

Power transform

Fisher score

Feature Selection

Challenge #1: Perturbations caused by malware are small

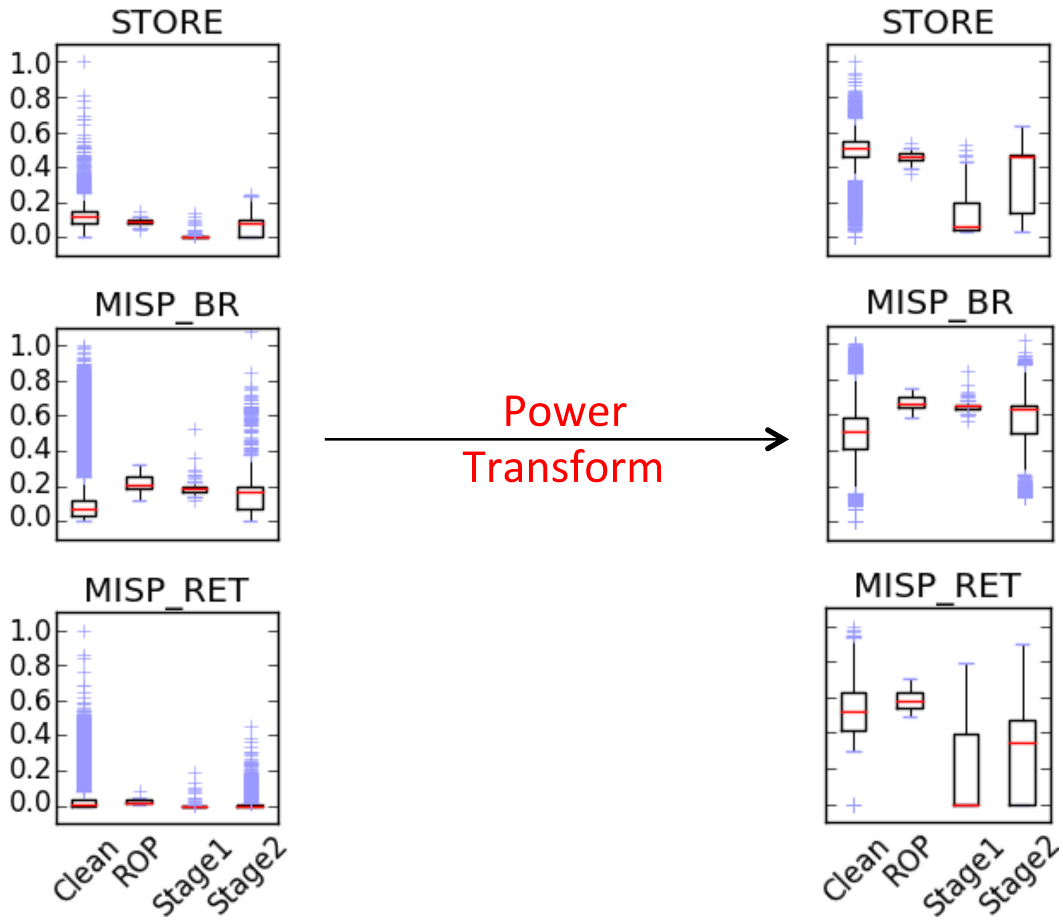


- Our approach:
  - Use rank-preserving Power Transform
  - For each event *i*, find the appropriate power parameter λ^i s.t. the normalized median for clean data is within tolerance ε of 0.5

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

Feature Selection

Challenge #1: Perturbations caused by malware are small



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

Architectural Events	
Name	**Event Description**
LOAD	Load instructions (ins.)
STORE	Store ins.
ARITH	Arithmetic ins.
BR	Branch (br.) ins.
CALL	All near call ins.
CALL_D	Direct near call ins.
CALL_ID	Indirect near call ins.
RET	Near return ins.

Microarchitectural Events	
Name	**Event Description**
LLC	Last level cache references
MIS_LLC	Last level cache misses
MISP_BR	Mispredicted br. ins.
MISP_RET	Mispred. near return ins.
MISP_CALL	Mispred. near call ins.
MISP_BR_C	Mispred. conditional br.
MIS_ICACHE	iCache misses
MIS_ITLB	iTLB misses

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

Higher F-Scores

Architectural-only

ROP

Set 'A-0'

```
Ret
Call_D
Store
Arith
```

Microarchitectural-only

Stage2

Set 'M-2'

```
Misp_Ret
STLB_Hit
Mis_ICache
Mis_ITLB
```

Both combined

Stage1

Set 'AM-1'

```
Store
Load
Misp_Ret
Call_ID
```

Methodology Overview

Collection

Feature selection

Feature extraction

Model building

Detection

Remediation

Temporal modeling

One-class Support Vector Machines

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

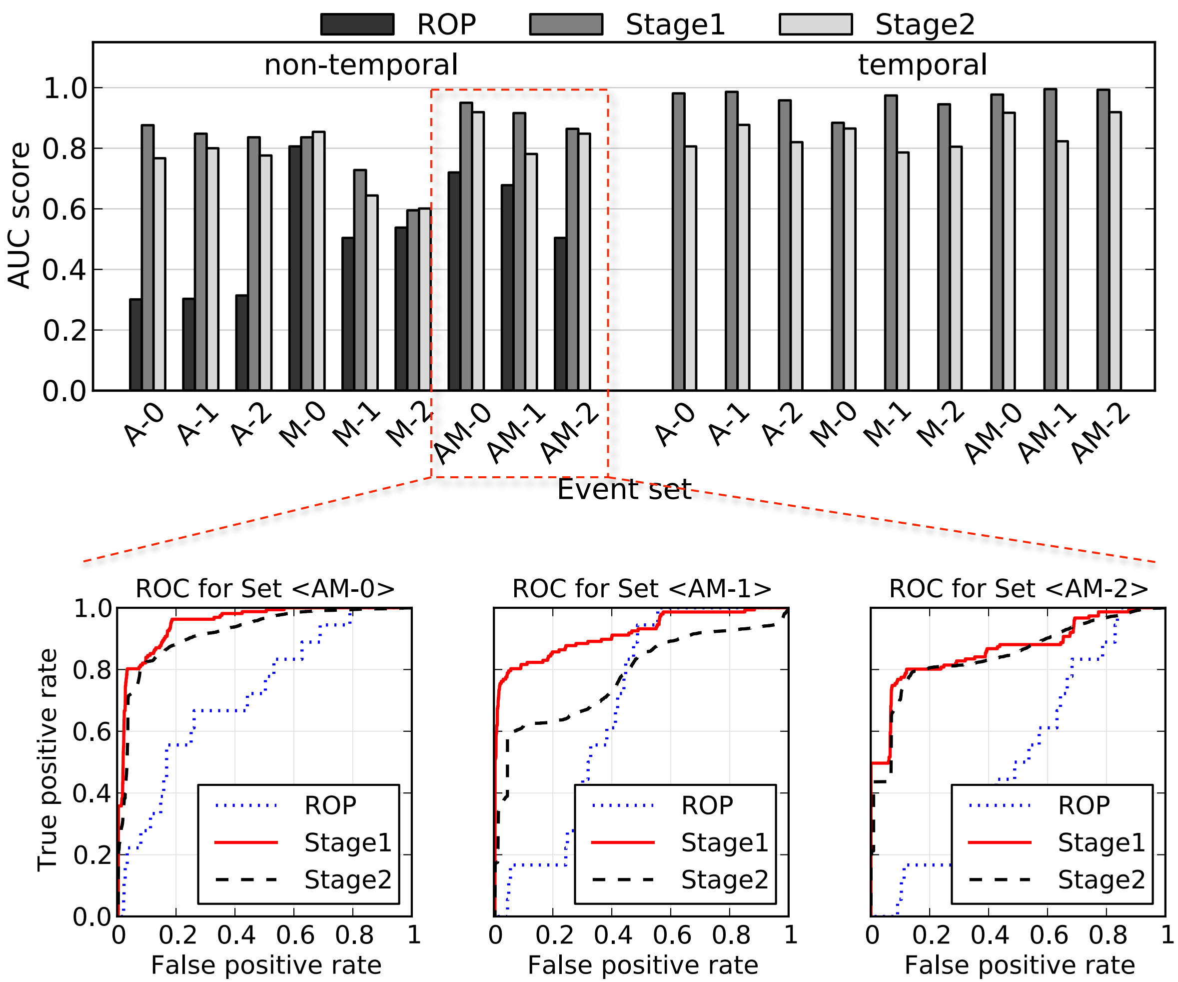
Remediation

Use models to detect exploits

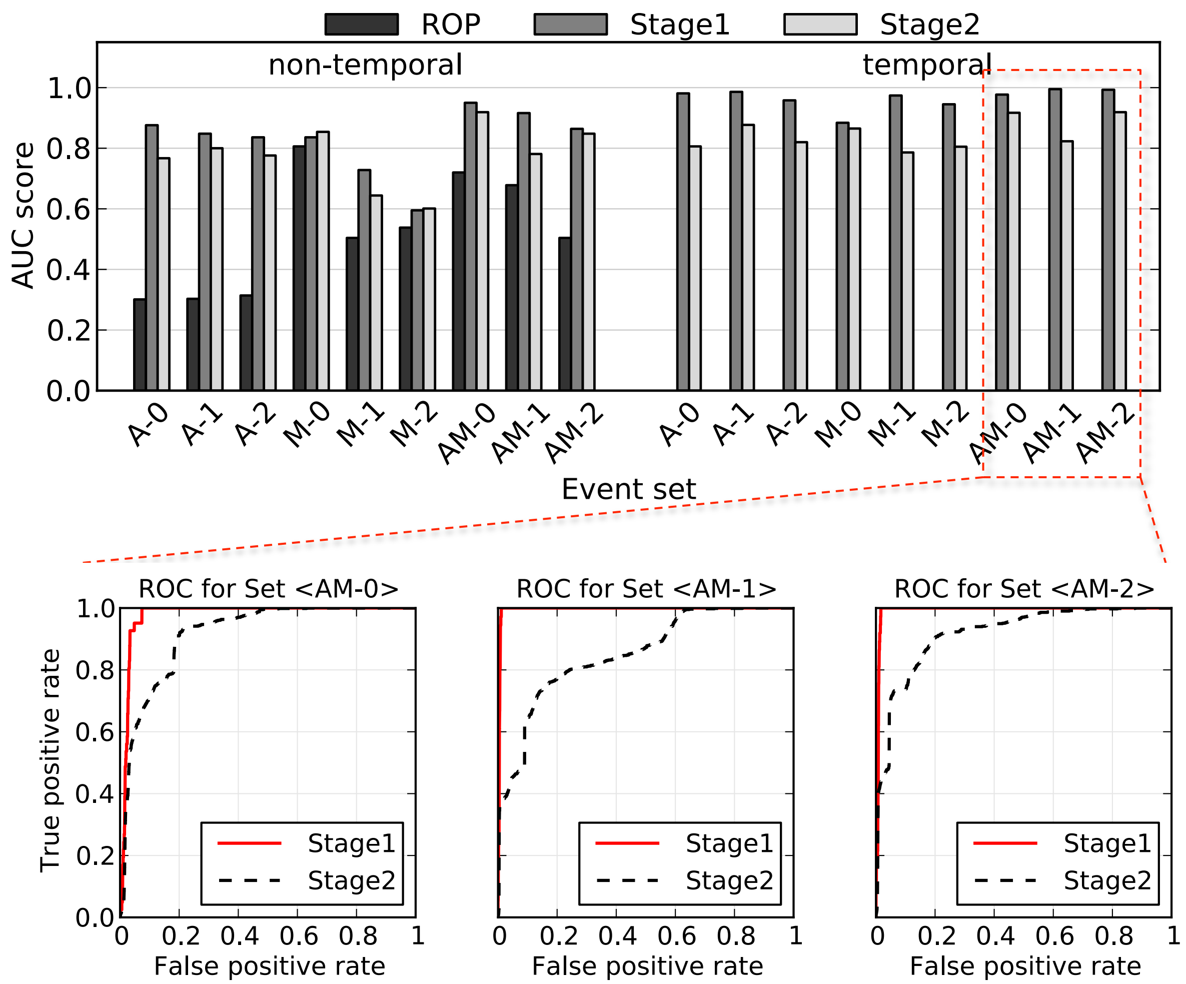
Outline

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

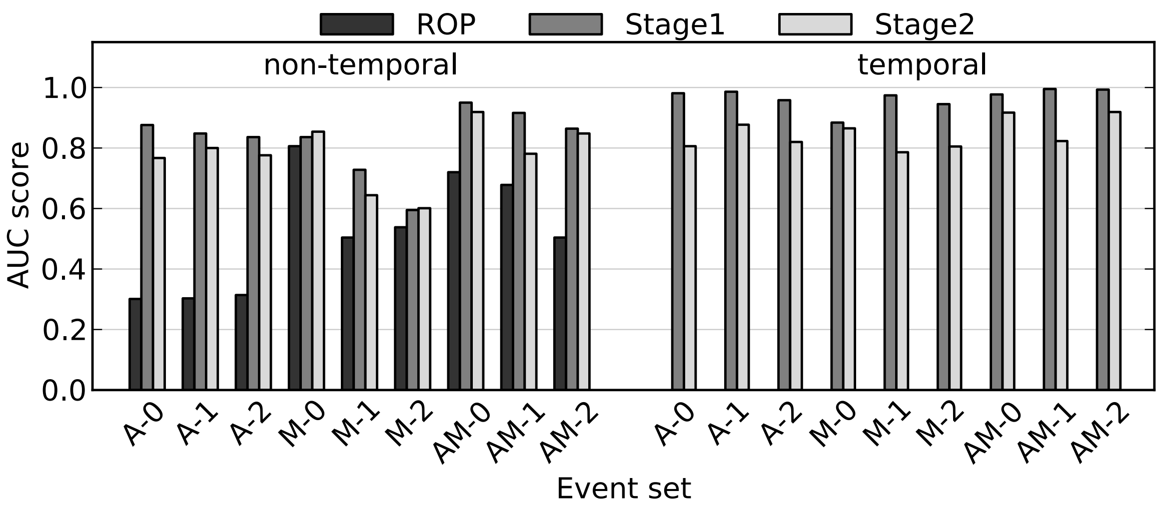
Results – Different Malware Stages



Results – Different Malware Stages



Results – Different Malware Stages

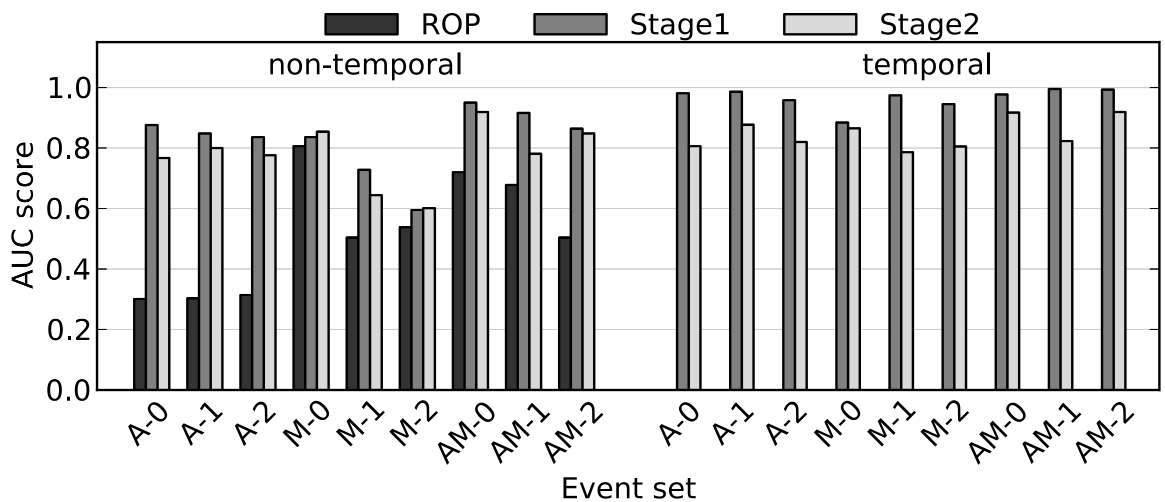


Models perform best in detection of *Stage1* shellcode

Better detection with the temporal modeling approach

Mediocre detection performance for *ROP* shellcode

Results – Arch vs μArch Events

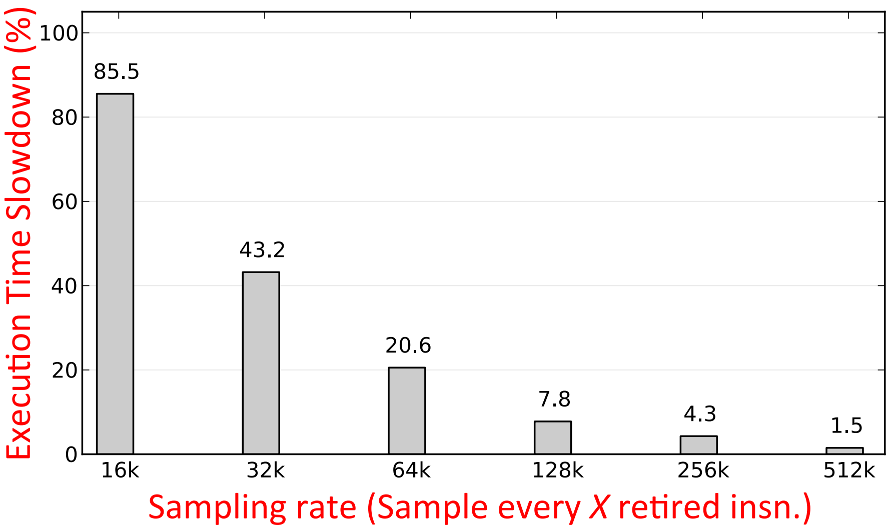


Arch-only (A-*) models perform better than μArch-only (M-*) models

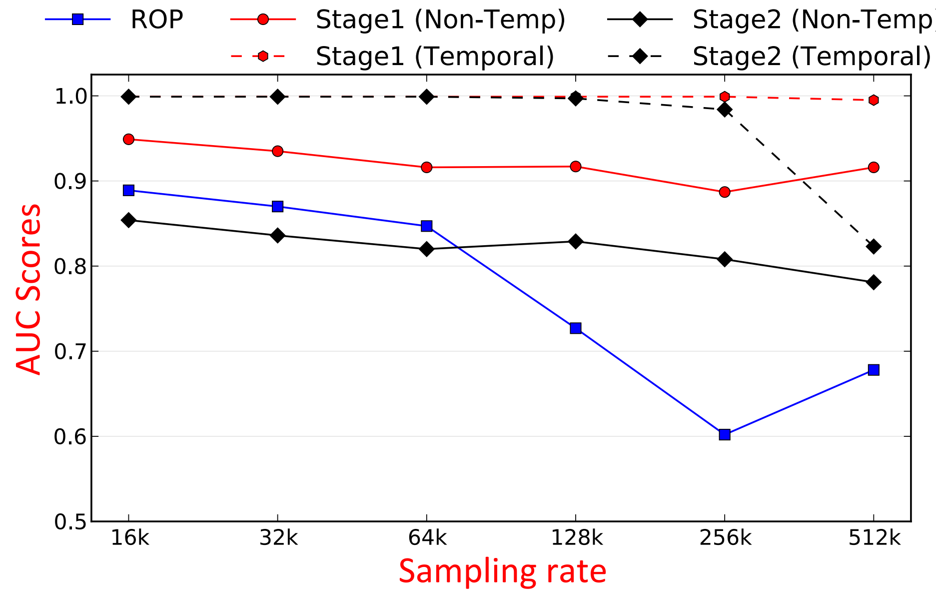
Combining the use of both Arch and μArch events in (AM-*) models achieves better detection performance

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!