Hardware Malware Detectors

John Demme, Matthew Maycock, Jared Schmitz, Adrian Tang, Adam Waksman, Simha Sethumadhavan, Salvatore Stolfo

Computer Science Department,
Columbia University
Worms

Trojan Horses

Botnets

Adware

Spyware
Malware in the Wild

AVTest unique database entries (av-test.org)
Why we are losing the battle?

Lines of Code

Sources: “An analytical framework for cybersecurity” DARPA’s Peter “Mudge” Zatko 2011 & Ohloh.net
State of Practice

- Software-based antivirus runs above the O/S, hypervisor and hardware
- Operating systems and hypervisors have exploited bugs
- Software antivirus is vulnerable and can’t help it!
Proposed Solution

- Hardware-based antivirus not susceptible O/S bugs
- Hardware tends to have fewer bugs & exploits
Outline

• Detecting malware with performance counters
  • ... using machine learning
• Experimental setup
• Results for Android
• An architecture for hardware antivirus systems
# How do we build hardware A/V?

<table>
<thead>
<tr>
<th>Primitives</th>
<th>How it works</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Software</strong></td>
<td>Files, Downloads, File Systems, Registry Entries, System Calls</td>
</tr>
<tr>
<td><strong>Hardware</strong></td>
<td>Memory, Dynamic Instructions, uArch Events, System Calls</td>
</tr>
<tr>
<td><strong>Scans downloads for static signatures</strong></td>
<td>?</td>
</tr>
</tbody>
</table>
Programs have Unique $\mu$Arch Signatures

Can $\mu$arch footprints uniquely identify malware during execution?

Could we build a database of malicious $\mu$arch signatures?
Outline

• Detecting malware with performance counters
  • ... using machine learning
• Experimental setup
• Results for Android
• An architecture for hardware antivirus systems
Machine Learning: Classifiers

Training (at A/V Vendor)
- Malware
- Not Malware

Production (on consumer device)
- Malware?
- Classifier
  - $p(\text{malware})$

Thursday, June 27, 13
Machine Learning on $\mu$Arch Events

- Microarchitectural event counts yield performance vectors over time
- Feed each vector into classifier, results in $p(\text{malware})$ over time
- Average over time, decide if malware or not with threshold
Machine Learning on $\mu$Arch Events

- Microarchitectural event counts yield performance vectors over time
- Feed each vector into classifier, results in $p(\text{malware})$ over time
- Average over time, decide if malware or not with threshold
Outline

• Detecting malware with performance counters
  • ... using machine learning

• Experimental setup

• Results for Android

• An architecture for hardware antivirus systems
Android Malware Detection

• 17x increase malware detections in 2012 ("Trends for 2013" ESET Latin America’s Lab)

• Android 4.1 mobile O/S

• ARM/TI PandaBoard

• 6 performance counters
Malware Families & Variants

- Family of variants which all do similar things
- Usually packaged with different host software
- Expectation: similar malicious code, different host code
Can we detect new variants after seeing only old ones?

Train classifier on these malware apps

Evaluate our classifier with different variants in the same family
Experimental Setup

- 503 Malware apps
  - 37 families
  - Taken from internet repository [1] and previous work [2]

- 210 Non-malware apps
  - Most popular apps on Google play
  - System applications (ls, bash, com.android.*)

- 3.68e8 data points total

- Tapsnake
- Zitmo
- Loozfon-android
- Android.Steek
- Android.Trojan.Qic somos
- CruseWin
- Jifake
- AnserverBot
- Gone60
- YZHC
- FakePlayer
- LoveTrap
- Bgserv
- KMIN
- DroidDreamLight
- HippoSMS
- Dropdialerab
- Zsone
- Endofday
- AngryBirds-Lena.C
- jSMSHider
- Plankton
- PJAPPS
- Android.Sumzand
- RogueSPPush
- FakeNetflix
- GEINIMI
- SndApps
- GoldDream
- CoinPirate
- BASEBRIDGE
- DougaLeaker.A
- Newzitmo
- BeanBot
- GGTracker
- FakeAngry
- DogWars

Very Realistic, Noisy Data

- Network connectivity **allowed:**
  - Malware could phone home
  - Additional noise introduced
- Input bias **allowed:**
  - Multiple users conducted data collection
- Environmental noise **allowed:**
  - Malware ran with system applications, not isolated

- Contamination between training & testing data **prevented:**
  - Non-volatile storage wiped, eliminating ‘sticky’ malware
  - Training/testing split before data collection

- Makes our task harder, better feasibility study
Outline

• Detecting malware with performance counters
  • ... using machine learning

• Experimental setup

• Results for Android

• An architecture for hardware antivirus systems
Experiments in Paper

• Detection of malicious packages on Android
• Detection of malicious threads on Android
• Linux rootkit detection
• Cache side-channel attack detection
Rootkit Detection

Percentage of Malware Identified vs. False Positive Rate

- KNN
- DecisionTree
- Tensor
- RandomForest
- FANN

Thursday, June 27, 13
Result Summary

• Detection of malicious packages on Android
  • 90% accuracy
• Detection of malicious threads on Android
  • 80% accuracy
• Linux rootkit detection
  • About 60% accuracy
  • Difficult problem; rootkits are tiny slices of execution
• Cache side-channel attack detection
  • 100% accuracy, no false positives
One Way to Improve Results

Android Software Package

Malware writers package with non-malware.

Problem: what’s actually malware?

Our (bad) solution: all of it

- Raises false-positives
Outline

• Detecting malware with performance counters
  • ... using machine learning

• Experimental setup

• Results for Android

• An architecture for hardware antivirus systems
Recommendations for Hardware A/V System Design

1. Provide strong isolation mechanisms to enable anti-virus software to execute without interference.

2. Investigate both on-chip and off-chip solutions for the AV implementations.

3. Allow performance counters to be read without interrupting the executing process.

4. Ensure that the AV engine can access physical memory safely.

5. Investigate domain-specific optimizations for the AV engine.

6. Increase performance counter coverage and the number of counters available.

7. The AV engine should be flexible enough to enforce a wide range of security policies.

8. Create mechanisms to allow the AV engine to run in the highest privilege mode.

9. Provide support in the AV engine for secure updates.
Recommendations for Hardware A/V System Design

1. Provide strong isolation mechanisms to enable anti-virus software to execute without interference.
2. Investigate both on-chip and off-chip solutions for the AV implementations.
3. Allow performance counters to be read without interrupting the executing process.
4. Ensure that the AV engine can access physical memory safely.
5. Investigate domain-specific optimizations for the AV engine.
6. Increase performance counter coverage and the number of counters available.
7. The AV engine should be flexible enough to enforce a wide range of security policies.
8. Create mechanisms to allow the AV engine to run in the highest privilege mode.
9. Provide support in the AV engine for secure updates.
Strong isolation mechanisms enable anti-virus software to execute without interference

- Non-interruptable
- A/V requires data from cores
- Starvation == exploit
- A/V uses off-die memory
- Starvation == exploit
Allow Secure System Updates

- Update protocol:
  - Decrypt
  - Verify signature
  - Check revision number
- Disallows access to classifiers & action programs
Outline

- Detecting malware with performance counters
  - ... using machine learning
- Experimental setup
- Results for Android
- An architecture for hardware antivirus systems
Contributions

(1) First hardware-based antivirus detection

• Promising results, reasons to believe results will improve
• First branch predictors started at 80% accuracy...

(2) Dataset available: http://castl.cs.columbia.edu/colmalset

Much to follow on: 0-day exploit detection, attacks, counterattack malware detection, better machine learning, precise training labels, ML accelerators, prototypes, etc.