Anti-Virus in Silicon

Adrian Tang, John Demme, Simha Sethumadhavan, Salvatore Stolfo
Columbia University
{atang, jdd, simha, sal}@cs.columbia.edu

Anti-virus (AV) software is fundamentally broken. AV systems today rely on correct functioning of not only the AV software but also the underlying OS and VMM. Thus proper functioning of software AV requires millions of lines of complex code – which houses thousands of bugs – to work correctly. Needless to say, and as evidenced in numerous software AV attacks, effective software AV systems have been difficult to build. At the same time, malware incidents are increasing and there is strong demand for good anti-virus solutions; the software anti-virus market is estimated at close to 8B dollars annually.

In this work we present a new class of robust AV systems called Silicon anti-virus systems. Unlike software AV systems, these systems are lean and mostly implemented in hardware to avoid reliance on complex software, but, like software AV systems, are updatable in the field when new malware is encountered. We describe the first generation of silicon AV that uses simple machine learning techniques with existing performance counter infrastructure. Our published and unpublished work shows that common malware such as viruses and adware, and even zero day exploits can be detected accurately [1, 2]. These systems form a very effective first-line, energy-efficient defense against malware.

1 Malware Detection Techniques

The idea behind the hardware malware detector is simple. Our research has shown that programs, be they malicious or benign, exhibit regular, reproducible behavior at the microarchitectural level. While these execution signatures vary to some degree in identical or very similar programs, they tend to differ radically across different types of programs. Based on this observation, we have empirically demonstrated the feasibility of detecting both known malware and zero-day (unseen) malware.

Known Malware with Supervised Learning Nearly 98% of malware infections are from known malware. We collected performance counter measurements from nearly 700 Android applications, both Android malware and typical Android goodware, and train models that describe what constitutes malicious and benign behavior using a series of *supervised* machine learning techniques (such as Decision Trees, Artificial Neural Networks). Using measurements collected from the execution of another different testing set of malware and goodware for the evaluation of the trained models, we observed that we correctly detect up to 90% of malicious malware packages with a less than 5% false positive rate, as presented in Figure 1 *(Left)*.

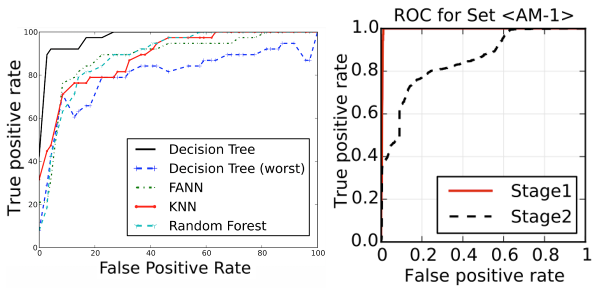

Figure 1: Accuracy of ***(Left)***: Signature-based classifiers in determining if families of malware and normal Android program packages are malware and ***(Right)***: Anomaly-based classifiers in determining if a given time-epoch sample for IE is malicious or benign, for 1 event set.

Detecting Novel Malware with Unsupervised Learning Our malware detector can also catch zero-day or unseen malware. We make the key observation that during the exploitation stage, in the act of infecting a system, malware alters the original execution flow of the vulnerable program to execute peculiar non-native code (termed *shellcode*). Through empirical experiments, we find that these malicious code execution cause observable perturbations to the original dynamic microarchitectural characteristics of the benign programs. With careful extraction of the features combined with *unsupervised* machine learning (Power Transform followed by One-Class Support Vector Machines with RBF), we can build baseline models of benign program execution and use these profiles to detect deviations that occur as a result of malware exploitation. Specifically, we characterize the normal behavior of Internet Explorer (IE) 8 and Adobe Reader 9 on a WinTel platform and evaluate the trained baseline models with malware exploits. Since the HPC can only monitor up to four events simultaneously, we experiment with using different combinations of events. For one event set, we are identified up to 100% of the time-epoch samples attributed to malware exploit execution with 1.1% false positives, as presented in Figure 1 *(Right)*.

2 Design

Figure 2 describes our malware detection system. This system offers very low data collection overheads, fast hardware classification and excellent isolation from attackers. We describe each of these features in more detail.

Data Collection Existing processors contain performance monitoring counters (PMCs) which monitor pro-

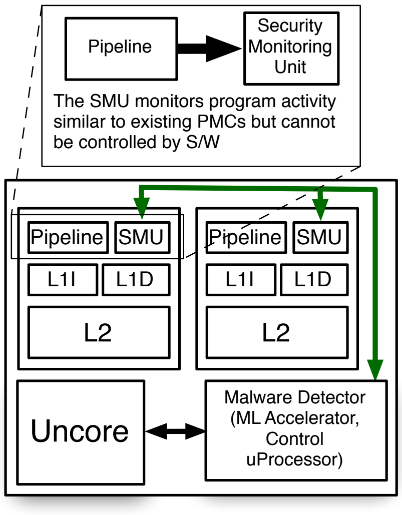
Pipeline
Security Monitoring Unit
The SMU monitors program activity similar to existing PMCs but cannot be controlled by S/W
Pipeline
SMU
L1I
L1D
L2
Pipeline
SMU
L1I
L1D
L2
Uncore
Malware Detector (ML Accelerator, Control uProcessor)

Figure 2: Our hardware antivirus is composed of modified performance counters called security monitoring units (SMUs) in each core plus an IP block which analyzes SMU data and reacts to malware detection events.

gram execution. Our hardware antivirus uses a similar monitoring structure – called the security monitoring unit (SMU) – but does not allow software to view, modify, or control the SMUs. Instead, the SMUs are controlled by the silicon anti-virus and collected data is transmitted only to the AV via a dedicated interconnect. Thus versus software the SMUs both eliminate data collection overhead and ensure that attackers cannot avoid being monitored.

Data Analysis The SMUs sample performance data at relatively high frequency (once every 25,000 cycles or less) which must be classified with a high throughput so as keep up. As such, one major functionality of the AV hardware is to serve as a machine learning (ML) accelerator. Our best performing classifier for AV thus far is a simple decision tree algorithm. While easy to compute, the decision trees must be stored in virtually isolated memory and their traversal can be memory latency bound. To maintain high throughput and reduce memory bandwidth requirements, the ML accelerator must store the upper portions of the decision tree in a scratchpad and batch requests for lower portions of the tree to amortize these accesses over many samples. While this strategy increases the latency in classifying each sample, many must be classified before an entire thread or process can be classified, so throughput is critical and only some fairness necessary.

FPGA Prototype We are using Xilinx's Zynq SoC to prototype our hardware AV. The Zynq contains two ARM Cortex A9 cores, a set of standard peripherals, and an FPGA fabric. To emulate SMUs, we are using the ARM cores' existing performance monitoring counters and their data samples are being transmitted over a shared interconnect to our hardware AV unit on the FPGA fabric. As a result, our prototype will have similar characteristics to production design in terms of functionality and performance impact. Once the prototype is hardened into a set of IP for SoC inclusion, it will have the security benefits of isolation from software.

Secure Updating To deal with emerging threats, like software AV, our hardware engine can be updated with new malware signatures as they become available. To create signatures, AV vendors will analyze new malware using performance counters and characterize their microarchitectural behaviors into signatures. Each update will contain these signatures, a configuration file that determines which performance features are to used with what classifiers, and possibly one or more classifiers (as new classification techniques are discovered). This data will be cryptographically signed and delivered to the signature-based detection component securely similar to how microcode update patches are delivered and applied today. Conversely, the zero-day detection component in our system does not need malware signature updates. It continuously trains its models and self-updates using the performance data from the SMUs.

3 Conclusions

Our silicon AV system is useful in a variety of applications which we plan to target. Mobile SoC vendors can include our IP and offer robust, low power AV as a value-added feature. Anti-virus vendors could also find our data collection and ML accelerators useful in their analyses of malware. Finally, our AV is also useful in the cloud settings; our AV should be able to detect malware running within customers' virtual machines, allowing providers to isolate the VMs from other customers.

Traditionally, the problem of dealing with malware has been relegated to the software community. Despite many advances in this area, the limitations of pure software approach in practical settings remain significant. It is our strong belief that moving AV (and other security capabilities, for that matter) into hardware is *essential* to significantly raise the bar against malicious entities. Our realization of this hardware AV implementation is an advance towards our vision to take the fight against malware to a new arena, one with fewer limitations and better security guarantees.

References

[1] J. Demme, M. Maycock, J. Schmitz, A. Tang, A. Waksman, S. Sethumadhavan, and S. Stolfo. On the feasibility of online malware detection with performance counters. In *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ISCA '13, pages 559–570, New York, NY, USA, 2013. ACM.

[2] A. Tang, S. Sethumadhavan, and S. J. Stolfo. Unsupervised anomaly-based malware detection using hardware features. *CoRR*, abs/1403.1631, 2014.