The SPARCHS Project

Hardware Support for Software Security

Simha Sethumadhavan, Salvatore J. Stolfo,
Angelos Keromytis, Junfeng Yang
Department of Computer Science
Columbia University
New York, NY
(simha,sal,angelos,junfeng)@cs.columbia.edu

David August
Department of Computer Science
Princeton University
Princeton, NJ
august@cs.princeton.edu

I. Problem Statement

Current security research is largely oriented to top-down design, where the most exposed layers --- the network/ application layers --- are first studied assuming the lower layers are secure, even when they are not. The lower layers are studied when new threats appear at those layers. Security, thus, has become an arms race to the bottom. For every software mitigation strategy today, vulnerabilities in the software layer below it can be used to attack and weaken the mitigation strategy. There are many examples of such attacks in the literature including those attacks that target anti-virus, libraries, operating systems, hypervisors, and BIOS routines.

A solution to the above problem is to push the security mechanisms down to hardware, which is typically immutable. Growing on-chip transistor budgets provide the opportunity to explore this possibility. In addition to offering immutable security, there are two further advantages to implementing security mechanisms in hardware. First, hardware supported security mechanisms can be much more energy-efficient compared to software only mechanisms. Given that energy- and power-efficiency significantly influence computing today, hardware support could very well be necessary for security mechanisms to gain traction in many real world settings. Second, implementing security mechanisms in hardware can provide unmatched visibility into execution. This provides an opportunity for new security techniques.

The SPARCHS project is considering a new computer systems design methodology that considers security as a first-order design requirement at all levels, starting from hardware, in addition to the usual design requirements such as programmability, usability, speed, and power/energy-efficiency. The rest of the paper describes the proposed hardware security mechanisms and the current status of the project.

II. Research Directions

Directly implementing security mechanisms in hardware poses a significant problem. First, since hardware is finite not all known security mechanisms can be implemented in hardware. Second, hardware is less flexible than software, so it cannot be easily updated when new attacks are discovered[1]. Thus, ideally, hardware mechanisms should also be able to cover attacks that are not yet discovered. This begs the question what hardware mechanisms can cover a wide variety of known and unknown attacks?

Instead of trying to discover unknown attacks, which is hard, and develop specific defenses, which is also hard, our strategy is to mimic the defenses from the biological world where a fantastic number of defenses have evolved over many thousands of years to survive constantly attacking predators. Our goal is to find counterparts to successful biological protection mechanisms and implement them in hardware.

The benefits of bio-inspired approaches to security have been mentioned as a promising direction in several reports[1,2]. The idea of applying bio inspired security principles to the hardware level is the key novel contribution of the SPARCHS project. Next we touch upon some biological protection mechanisms and then describe their hardware/software formulations that can mimic these mechanisms.

A. Biological defenses

Defensive strategy is pervasive at all levels in the animal and plant kingdom where existence is constantly threatened due to predators and environmental vagaries. At the molecular level, our genetic code is suspected to contain a

[1] FPGAs offer an opportunity to update software in the field but currently have limited utility in general purpose computing.

high-level of redundancy, at the cellular level lymphocytes offer innate protection against viruses and microbes, at the organ level, redundancy (e.g., two kidneys) and regeneration (e.g., skin cuts, lizards dropping tails under attacks) allow continuous function and recovery under attack, and organisms have amazing ability to learn from past attacks (e.g., vaccination.) In many cases, multiple organisms co-operate (e.g., microbiomes) from symbiotic relationships to provide immunity over and above innate and adaptive immunity. Innate immunity mechanisms is typically a first, generic response to attacks from foreign organisms. Typical functions of innate immunity include capturing cellular debris, foreign particles and invading microorganisms. The adaptive immune response provides facilities to recognize and remember specific attack vectors, and provide stronger protection as more attacks are encountered in future.

In the biological world, the attackers have also evolved many sophisticated techniques to thwart existing defenses. The most notorious of the attackers attack the immune system itself (e.g., HIV) and is difficult to destroy because it constantly changes its tertiary structure (polymorphism), which guarantees the virus a safe harbor in the host. To provide these amazing security features organisms spend nearly 30% of their energy in defense. Given the success of flora and fauna, the defensive strategies used in biological systems are certainly worth emulating.

To summarize the biological techniques, we aim to provide hardware support to mimic the following biological primitives: (1) Innate Immunity for detection and isolation, (2) Diversity and polymorphism for prevention, (3) Symbiotic Immunity for implementing protection and detection techniques, (4) Adaptive Immunity for prevention, (5) Optimized redundant execution for continued execution, (6) Autotomy to contain damage when all else fails.

B. Hardware Analogues

Innate Immunity One basic function of innate immunity is to identify and contain foreign particles. Translated to the computer systems, this translates to ensuring untrusted data does not reach confidential code, and trusted data is not sent to untrusted code. Information flow tracking (IFT) essentially provides the above functionality. While IFT is no means a new technique, it has been difficult to implement correctly without hardware support. Verifying the data flow of program is insufficient to verify that no illegal information flows occur in the program. Current solution is to allow implicit flows by converting all data dependences to control dependences. In the SPARCHS project we are considering how the implicit flows, or the flow of information through control dependences, must be determined with static analysis as information can flow through segments of code that do not execute. This information will be conveyed to the hardware to track these flows.

Diversity and Polymorphism The key idea in digital defensive polymorphism is to change the execution of a program dynamically to thwart attackers. One of the simplest ways to change execution is to change the hardware each time a program executes. We call this type of shape-shifting hardware as polymorphic hardware. Polymorphic hardware succeeds against attackers by purposely injecting randomness into program execution. This method could have three very useful impacts for security at different levels of execution:

1) At the hardware level this architecture would make side-channel attacks very difficult, because it is always harder to attack a moving or unpredictable target. Conceptually each execution of a program happens on a different hardware, and with this type of uncertainty the attacker cannot reliably interpret the side-channel data. This resilience to side-channel attack is leveraged by symbiotes to avoid detection.

2) Polymorphism can provide resilience against semantic attacks: Consider a code-injection attack. The attacker takes advantage of knowledge of the programs application binary interface (ABI) and the instruction's semantics in the program to carry out the attack. With polymorphism at the instruction-set level — Instruction-Set Randomization (ISR) — this attack can be thwarted because the attacker can no longer know the semantics of each instruction.

3) Polymorphism can provide resilience against program logic bugs. While most security attacks to date exploit bugs in serial programs, more parallel programs are being produced because of adoption of multicore programs. It is well known that reliable parallel programming is harder than sequential programming, and it is likely that attackers will take advantage of concurrency bugs in the near future.

The polymorphic architecture can decrease the chance of security attacks on emerging parallel programs by reducing the chance of race conditions because of the diversified, random execution substrate. Additionally, polymorphism may have a side benefit of improving program performance by reducing unintentional contention on shared resources, and also enable better testing of programs through automatic fuzzing of program execution. The SPARCHS project is investigating how these shape-shifting features can be implemented in the simplest way into existing processors without undue performance impact.

Symbiotic Immunity The idea of how symbiotes can be adapted to computer systems was first proposed in the

Minestrone project at Columbia. At a high-level, the symbiote is a small program that is embedded in a host program. The symbiote can reside within any arbitrary body of software, regardless of its place within the system stack. While symbiotes share some commonalities to reference monitors in terms of benefits they offer, a key difference is that symbiote cannot survive without the host program and the host program cannot survive without the symbiote. This interdependency is not required for reference monitors. The SPARCHS project aims to provide hardware support that will allow symbiotes to have this property.

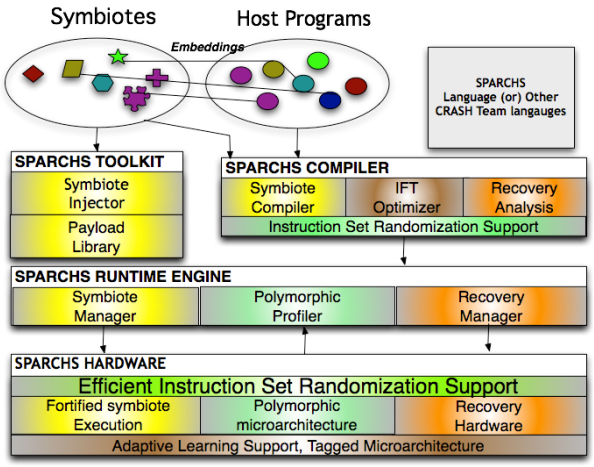
Symbiotes can be supported in hardware through three distinct ways that have different ease-of-implementation vs. benefits trade-offs. First, one or few cores in a multicore processor may be hidden from all system software by modifying the BIOS, and having the symbiotes run on a hidden core. Since system software cannot see the disabled core, the symbiotes functions cannot be monitored. This solution assumes that the hidden core has access to all of the on-chip memory, which can be easily architected. The second solution is not to disable the cores (thus not reduce throughput) but use the shape-shifting polymorphic architecture such that no side-channels are possible. Finally, the most efficient option is to build a special hardware unit that guarantees physical and execution isolation for the symbiotes.

Adaptive Immunity Adaptive immunity requires methods to learn about normal and abnormal program behaviors. We are developing better hardware support to identify anomalous execution points by using fine grained measurements from on-chip performance counters. We are also working on newer performance counter architectures as on-chip performance monitors today are tuned for collecting information on the common case; in security (and software engineering) we are interested more in the uncommon case. Other methods for program characterization include learning about control and data flow execution graphs. The SPARCHS project is investigating hardware primitives that can be securely used to learn about program behavior.

Optimized Redundant Execution Mimicking biological redundancy in digital systems is fairly obvious. N-versioning is already a very common approach but it is also terribly impractical approach for many modern execution environments such as mobile and server environments. We are planning to make N-versioning better with compiler and hardware optimizations. We plan to use values produced from one redundant copy in another to improve the speed of the diversified replica or use special purpose microarchitectures to quickly communicate values between the N-versions.

Autotomy/Apotasis SPARCHS will bring autotomy to computing systems by detecting attacks and faults in a software subcomponent and responding to it by removing the component from the critical system. The goal is to heal the programs, and make them available as much as possible. SPARCHS Autotomy will employ Rescue Points which are locations in the application code in which error handling is performed with respect to a given set of programmer foreseen error conditions. Rescue points basically create a mapping between the set of errors that could occur during a program's execution and the limited set of errors that have been explicitly handled in the program code. Thus, a failure that would cause the program to crash is translated into a return with an error.

C. Integrated System



While the project's main focus is on discovering efficient hardware primitives for security another important goal is to demonstrate how the proposed primitives can be used in software. Towards this goal we are developing hardware and software for the SPARCHS system. Figure 1 illustrates the different facets of the SPARCHS system and how it interfaces with existing and proposed research techniques. At the hardware level, SPARCHS includes support for microarchitecture-level polymorphism, support for memory versioning and checkpointing to support roll back and recovery, and specially fortified hardware to support symbiote execution. These techniques are orthogonal to and can be integrated with information flow tracking and strong instruction set randomization. SPARCHS guarantees that an outside attacker cannot simply turn off the protection mechanism. SPARCHS includes a simple management layer that provides safe storage of keys and feeds program profile information to the SPARCHS compiler. The management layer can also provide simple recovery services.

SPARCHS is full-system effort and includes many software aspects. The SPARCHS compilation suite serves three main purposes: first, it combines the application, the symbiote, and the symbiote policy into a single binary, and applies instruction-set randomization to the binary; second, it provides static analysis techniques for managing recovery and repair. Finally, it provides analysis to enforce correct dynamic information flow in hardware. The SPARCHS environment includes toolkits for mining static, dynamic and program information to help programmers specify policies, symbiote payload libraries, and also standalone injection of symbiotes in binaries if necessary.

III. CURRENT STATUS

We have been working on this project for three quarters now. We have made significant progress on several fronts. First, we have created software symbiote infrastructures to understand how they should be protected in hardware[3]. We are planning to create symbiotes in x86 and ARM to demonstrate feasibility in a wide variety of architectures. The software port is likely to be completed in the next quarter and detailed hardware analysis is going to begin in the following years. We have made significant progress on hardware support for learning/adaptivity[4]. Existing methods for accessing performance counters on x86 machines seem terribly out-of-date. In fact, popular tools like Vtune, PAPI and Oprofile use heavyweight kernel calls which perturb hardware measurements. We have created new tools that will allow to precisely read the performance counters and have about 70x lower overhead compared to PAPI. The tool is available for download from: http://castl.cs.columbia.edu/limit. This tool is currently being used to learn normative execution characteristics of programs. As a stepping stone to ISR, we have developed a full-system ISR mechanism[5], not just covering single program program binaries but including support for DLLs, shared libraries, key management etc. Hardware modifications and full system prototypes are underway. To test our systems we are working on creating new attacks (concurrency based[6]) and also demonstration of defense mechanisms against such attacks. There is much exciting work to be done in this area. One major open question is what further primitives can be added to hardware. Our experience with SPARCHS could help answer this question.

[1] A. Somayaji, S.Hofmeyr, S. Forrest, "Principles of a Computer Immune System," NSPW '97 Proceedings of the 1997 workshop on New security Paradigms, 1997.

[2] S. E. Goodman, H. S. Lin, "Toward a Safer and More Secure CyberSpace," Committee on Improving Cybersecurity Research in the United States, The National Academies Press, Washington D.C., 2007

[3] A. Cui, S.J. Stolfo, "Defending Legacy Embedded Systems with Software Symbiotes," Proceedings of the 14th International Symposium on Recent Advances in Intrusion Detection. RAID 2011.

[4] J. Demme, S. Sethumadhavan, "Rapid Characterization of Architectural Bottlenecks via Precise Event Counting," Proceedings of the 39th ACM/IEEE International Symposium on Computer Architecture, June 2011.

[5] G. Portokalidis, A.D. Keromytis: Fast and practical instruction-set randomization for commodity systems. ACSAC 2010: 41-48

[6] J. Yang, A. Cui, J. Gallagher, S. Stolfo, S.Sethumadhavan, "Concurrency Attacks,", Department of Computer Science Technical Report, CUCS-028-11, 2011.