Hardware Foundation for Secure Computing

Donato Kava, Alice Lee, Aaron Mills, and Michael Vai
MIT Lincoln Laboratory
244 Wood Street
Lexington MA 02421
{donato.kava, alee, arron.mills, mvai} @ll.mit.edu

***Abstract*—Software security solutions are often considered to be more adaptable than their hardware counterparts. However, software has to work within the limitations of the system hardware platform, of which the selection is often dictated by functionality rather than security. Performance issues of security solutions without proper hardware support are easy to understand. The real challenge, however, is in the dilemma of "what should be done?" vs. "what could be done?" Security software could become ineffective if its "liberal" assumptions, e.g., the availability of a substantial trusted computing base (TCB) on the hardware platform, are violated. To address this dilemma, we have been developing and prototyping a security-by-design hardware foundation platform that enhances mainstream microprocessors with proper hardware security primitives to support and enhance software security solutions. This paper describes our progress in the use of a customized security co-processor to provide security services.**

Keywords— Hardware security; security co-processor; secure architecture; processor customization; processor optimization; verification; trust; assurance; protection; RISC-V

I. INTRODUCTION

Software implementation of security principles are often looked upon favorably by developers, since in theory, software could be designed to operate independently of its hardware foundation (e.g., by using a hypervisor). In practice, this software advantage must be enabled and supported by the underlying hardware platform.

For example, the use of a formally verified separation kernel (e.g., seL4 [1]) to implement the principle of isolation is not a software only decision. First, the kernel must have been ported to and verified for the hardware platform in the system under design. Furthermore, security software often make liberal assumptions using an abstract, superficial model of the underlying hardware. As software cannot defend against attacks targeting hardware itself, we need to ensure that attack surface is not pushed down to the hardware layers. Spectre/Meltdown attacks are infamous examples of exploiting hardware vulnerabilities not represented in the abstract hardware mode [2].

Hardware security support at the processor level has had a slow start, but processors with security features, such as ARM TrustZone [3], ARM MTE [4], Intel SGX [5], etc., are now available commercially. Understandably, most COTS (Commercial Off-The-Shelf) approaches are evolutionary, due to the need of backward compatibility, so their threat models and security effectiveness need to be understood by system designers.

The development of new processors with advanced security architectures has been a popular research topic. Particularly, the DARPA CRASH (Clean-slate design fo Resilient, Adaptive, Secure Hosts) program has inspired a new wave of processor development projects, for example, [6][7]. Nevertheless, for reasons including, but not limited to, backward incompability, different threat models, and immature ecosystems, their commercialization and adaption into mission systems such as airborn platforms have been a slow and uncertain process. Unlike commercial systems such as IoT devices, which could rapidly adapt new processors, mission critical applications, due to lengthy safety certification processes, have much slower update cycles.

Our project aims at the development of a hardware foundation for secure computing. Instead of proposing yet another secure processor, we have turned to an approach for creating a mission-centric hardware foundation that includes capable, mainstream processors, chosen by best practice, and security co-processors, such as a crypto engine, to provide hardware security primitives to software security solutions. A hardware platform consisting of main-stream processors and customized co-processors has the following benefits. First, this hardware foundation resonates with mission stakeholders as it is based on COTS main-stream processors. Second, the entire foundation could potentially be implemented with a COTS System-on-Chip Field Programmable Gate Array (SoC FPGA), such as Xilinx UltraScale [8] and Intel Cyclone [9], which provide FPGA fabric along with main-stream processors (in this case ARMs) so that the entire foundation could be implemented and deployed as a single, integrated device.

This paper begins with the establishment of a reference architecture for secure computing. We will then explain the use of its operating stages to identify hardware foundation security requirements. The architectural detail of the hardware foundation, will be explained with an asymmetric multiprocessing model, in which a mission-specific co-processor is developed as a root-of-trust/recovery. The benefit of this approach is that we have the main-stream processor providing the processing performance needed by the mission, while a co-processor, customized for its purpose, complements and supports security features provided by the processor and software stacks. This asymmetric architecture also enables the use of a trusted foundry to manufacture the co-processor, as it typically does not rely on performance only achievable with advanced semiconductor technologies.

 This material is based upon work supported by the Under Secretary of Defense for Research and Engineering under Air Force Contract No. FA8702-15-D-0001. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Under Secretary of Defense for Research and Engineering.

We then describe the implementation of a configurable embedded hardware platform testbed, which is built on top of the Synopsys ASIP (Application Specific Instruction Set Architecture Processor) design environment [10]. We will explain the current hardware components we have had prototyped using this testbed. Future plan of expanding the testbed capabilities will be described.

We conclude this paper by illustrating the customization of the co-processor, in this case a RISC-V processor, to contain only instructions needed in its operation of providing authentication, confidentiality, and integrity to the software code executed on the main processor. Ongoing and future research and development directions will also be provided.

II. System Architecture

Figure 1 shows a reference secure and resilient embedded system architecture that we have adapted for this development. As memory vulnerability accounts for 70% of attacks [11], we have chosen a secure computing stack that focuses on a tagged architecture for type and memory safety and fine-grained compartmentalization. This represents the state-of-the-art, semantically aware secure computing architecture [7].

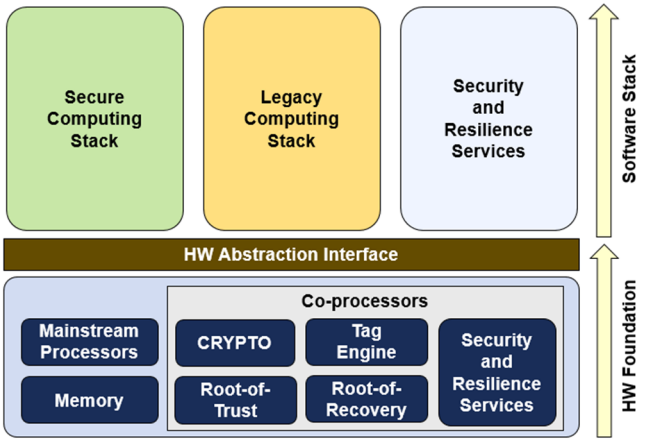

Figure 1. Reference secure and resilient embedded system architecture.

In order to resonate with mission stakeholders, we have also included a legacy computing stack in the reference architecture. The legacy computing stack consists of a separation kernel and a hypervisor to support legacy operating systems and applications. This legacy computing stack, like many software solutions, often inadequately assumes that the entire hardware layers are included in the TCB. Hardware primitives will have to be developed to minimize residual vulnerabilities of such a stack.

For the hardware layers, one or more COTS processors, such as ARM, popular in mission systems, are incorporated for functional computing so that mission stakeholders will readily resonate with the architecture. Processors (e.g., ARM MTE) that support fundamental tagging have begun to be available [4]. Advanced tag and crypto capabilities could be included as co-processors for hardware primitives to support the computing stacks. A hardware abstraction interface is used to simplify the interaction between software and hardware.

III. Security and Resilience Requirements

When the hardware is deficient, it is impossible to build secure systems. Apparently the hardware should provide proper performance to run demanding applications, such as real-time algorithms. The hardware foundation should also be protected against attacks. However, as the system cannot be hardened against all attacks, recoverability, the ability to return to a predetermined, known-good state when protection fails, must be provided. The hardware must thus support the detection, and isolation of cyber induced failures, and enable a recovery to either continue with the mission or switch into a survival mode. In a summary, the hardware should support, enable, or enhance the implementation of the following Cyber Security and Resilience (CSR) properties:

Separation – The division of functionality into isolated modules with rigorously controlled interfaces to limit attack surface and avoid single point-of-failure. This also provides a base from which it is possible to fight through an attack on the system.

Redundancy – The insertion of functionality that is redundant to those already in the system. The goal is to achieve detection of failures or exploits and provide fail-over capability as well as detection of aberrant behavior. Diversified redundancy will be required so that attackers do not have the "break-one, break-all" advantage.

Obfuscation – The concealing of data, code, or logic by transforming it via deterministic or non-deterministic means. Cryptography is a popular way to obfuscate data in a difficult to reverse way.

Authentication – The capability to verify the integrity or provenance of code, data, or hardware.

Monitoring – The capability to check the integrity of the system in execution.

The implementation of these CSR features will be dependent on many mission-specific factors. We have thus adopted a strategy, to be explained below, to specify the technologies to be incorporated. Many mission embedded systems are severely SWaPC (size, weight, power, and cost) constrained, the performance overhead and the cost of obtaining the CSR features at the hardware level have to be minimized. Recognizing that a requirement is not meaningful if it is not verifiable, information for verifying our claims on the hardware foundation must be available. In order to avoid vendor lock-in, the components in the hardware foundation should be available from multiple sources.

IV. Hardware Foundation Architecture

While secure processing technologies have made long leaps in many aspects, no one-size-fits-all solution is available. We have thus begun by establishing a customizable hardware foundation that adapts and synergizes various technologies. This approach is quite effective as often a small change in the architecture can greatly ease our task to build a secure mission system.

We have taken a design-for-mission-assurance approach to analyze the requirements and apply proper technologies and

techniques in the hardware foundation [12]. The overall concept of this approach is to focus on typical embedded system operational sequence steps, such as pre-boot, secure-boot, code loading, and execution. For each of these steps, security requirements, currently available software protections, and their assumptions on hardware are analyzed to derive the development of the hardware platform. This analysis has been summarized in Figure 2.

Operating Stages	Example Critical Information	Key Mitigation Techniques		
		Hardening	Detection	Recovering
Pre-boot	Data-at-Rest: CPI, OS, Boot Code, Apps, ...	Crypto	Crypto	Redundancy
Load Boot Code	Boot Code	Crypto	Crypto	Redundancy
Load SW First Layer	OS, VM	Crypto	Crypto	Redundancy
Load App Code	Apps, CPI	Crypto	Crypto	Redundancy
Execute Apps	Apps	Testing, Formal Verification, Isolation	Monitoring	Checkpoint, Reloadable Golden Copy
Memory R/W	Products	Crypto	Crypto	Redundancy
Access Peripherals	Video, ISR, GPS, ...	Crypto	Crypto	Redundancy
Communicate Off System	Messages, Controls, ...	Crypto	Crypto	Redundancy

Figure 2. Derivation of mitigation techniques for typical embedded systems. (CPI: Critical Program Information)

Various attacks, some of which may be unknown at design time, could cause the same effect. We have thus shifted the emphasis of analysis from specific attacks to effects, described as violations of confidentiality, integrity, and availability on essential functions, thus enabled us to analyze the system more easily. Our strategy in the development of a hardware foundation is thus to view it as a journey going from an assembly of state-of-the-art, matured technologies towards inherently secured clean-slate designs. This vision leads us to create a configurable asymmetric multicore processing architecture, as originally illustrated in Figure 1.

Any adequate mainstream processors could be included. For the reasons we have discussed earlier, such as to support seL4, ARM cores were initially selected as the mainstream processors for compatibility with legacy systems. A number of customizable co-processors were included to extend the capabilities of the processors and provide proper hardware primitives to support software security. Candidate co-processor functions included tag engine, crypto engine, root-of-trust, root-of-recovery, and a programmable controller, labelled as security and resilience services in Figure 1, to provide and coordinate CSR services.

We have adapted SHAMROCK (Self-contained High Assurance MicRO Crypto and Key-management) [13], a hardware crypto and key management accelerator as a co-processor. For the programmable controller, we have adopted the RISC-V ISA (Instruction Set Architecture) to take advantage of its free, open standard for verifiability. The fact that the RISC-V ISA does not dictate its implementation allows us to customize it to target specific system needs. Another benefit is that we could leverage the significant research effort ongoing in the RISC-V community, such as the tagged RISC-V architecture [14] as a baseline for the tag engine. The co-processor of a tag engine is an ongoing development task.

Together, the controller, SHAMROCK, root-of-trust, and root-of-recovery form the TCB of the system. The controller uses cryptography and key management for booting, monitoring, recovering, and any survival essential functions required by the mission system to be resilient.

V. Development and Assessment Testbed

A single simple change to a piece of functioning hardware could trigger a lengthy redesign process. However, the proposed hardware foundation needs to be customized and optimized for specific mission requirements. For this purpose, we have leveraged a design environment to perform its customization and optimization in a design space consisting of SWaPC and assurance. Figure 3 shows the setup of this design environment. The current design environment has been built with Synopsys ASIP (Application-Specific Instruction-set Processor) Designer [10]. Using this environment, a mission-specific instantiation of the hardware architecture, described in a high-level definition language for its architecture and instruction set, is compiled into an emulated prototype with associated software development kit (compiler, library, etc.). The critical functions of mission applications are then mapped onto the emulated prototype for assessment.

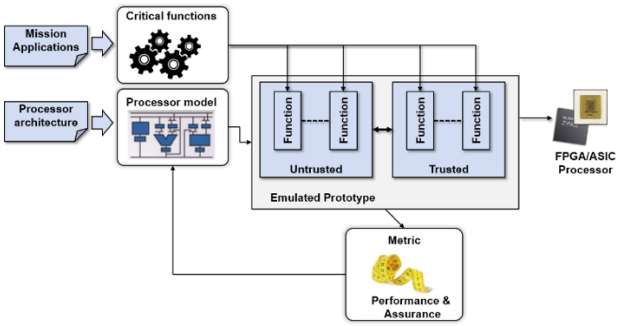

Figure 3. Hardware foundation design environment.

During the development stage, we rely heavily on emulation. When a satisfactory design is identified, the design environment allows the design to be converted into synthesizable register transfer level (RTL) hardware description language (HDL) code compatible with common electronic design automation (EDA) tool chains. The entire hardware foundation can be synthesized into either FPGA or ASIC. Apparently the decision of doing an ASIC implementation, which has a high non-recurring expenses (NRE), has to be justified by its required quantity and other considerations.

VI. Security Co-processor for Software Update

In this section, we describe a simplified design example of applying the hardware foundation and its design environment to protect a legacy system that is used in an infrastructure for essential services, thus a target of cyber-attacks. This system consists of a group of spatially dispersed and dedicated controllers for sensing and controlling physical entities. Unsurprisingly, this legacy system was not built for security and has been relying on regular software patching to mitigate potential attacks. Although it is now a common practice to patch OS and applications regularly, legacy systems are often unsuitable for modern security, as improperly implemented

security measures could create new attack surface for malware infection and denial-of-service attack.

The system to be protected operates continuously and maintains real-time responses. The critical threat has been assessed to be missed, delayed, partial, or malformed control and measurement signals. The most concerning attack surfaces consist of the software update process, as well as bugs and/or malicious code embedded in a legitimate update package.

In order to ensure meaningful interactions between the stakeholder, the functionality team, and the security team, we have described the system threat model by a vignette:

> "The hacker attacks by introducing malware into the system through software patching to compromise its operation."

The foremost CSR requirement is thus to provide uncompromised capability by ensuring the integrity and authenticity of the software patching package delivered to the system.

Our solution is the use of our security co-processor to manage patching by encryption and authentication so that the integrity and authenticity of the patching package are maintained. In order to detect and mitigate bugs and malicious activities in updates, the co-processor will also monitor system functions, detect anomalies, and recover from attacks.

The co-processor core is used with cryptography and key management for booting, monitoring, and recovering essential functions required for the system to be uninterruptible. This constitutes the minimized TCB represented, and could be developed as a trusted component with formal verification and/or trusted manufacturing. The co-processor provides bootstrapping to bring up the functional processor and manages failure detection during operation, by monitoring key signal and control ranges. The application has been ensured that it is reloadable and restartable for recovery. Cryptographic approaches are used to check any software patch package received for its authenticity and integrity, meaning that it has come unaltered from an authorized source.

VII. Co-processor Customization

In this section we discuss the customization and optimization of a security co-processor for the protected system. Multiple approaches can be used to implement security features in the co-processor. Security and survival features, such as cryptography, key management, monitoring, etc., required by the system can be implemented with a mixture of software, firmware, and hardware. Each implementation will have its advantages and drawbacks in performance, security, and verifiability. Our systems analysis on performance, SWaPC, and security has prescribed the use of a co-processor with a secure, minimal code base to act as the root of trust.

The co-processor is based on the RISC-V 32IM ISA. Following the security principle of minimalism, we have reduced the complexity of the co-processor by customizing its instruction set for its role in the system. For example, floating point instructions and related hardware can be removed if they are not required. Minimizing an instruction set facilitates verification and validation.

While conceptually straightforward, this customization approach of modifying the instruction set of a processor would not be practical if its customized product requires a significant amount of additional design and validation effort (e.g., development cost). In practice, minor changes to a design often cause significant design and validation overheads. Additionally, this approach custom tailors each processor depending on the system needs, as the number of removed instructions will vary with the co-processor complexity.

On the other hand, removed instructions may need to be emulated, at the cost of performance overhead, with the remaining ones, as losing a function (e.g., memory access or flow control) could jeopardize the underlying microprocessor operations. Therefore, an analysis to determine the system effects from creating an application-specific instruction set is necessary. Instruction set optimization and minimization need to be performed in a processor design space consisting of security, functionality, performance, and verifiability.

The development and assessment testbed described in Section V was leveraged as a design environment for performing processor instruction set customization and optimization. The fast-turnaround capability of synthesizing the co-processor into an emulated prototype has allowed a designer to abstract the hardware design and quickly verify changes with the generated processor specific simulator and software compiler tools. Changes that could take weeks to implement and functionally validate are often reduced to days.

With a design environment that enables rapid processor development, we can readily adjust the hardware foundation by changing the number of co-processor cores and customizing their functions. Each core has its own capabilities and makes up part of the hardware foundation. Functional verification could then be kept in small batches, as each new addition is minimal in its areas and functionality.

For the security co-processor under design, a systematic analysis has been used to determine instructions not essential for the co-processor as a means to minimize its verification footprint. For example, one of the main functions of the co-processor is to perform AES-256 encryption, we have thus performed an analysis of the instructions used by its compiled binary code.

Figure 4 shows the result of analyzing an implementation of the AES-256 cryptographic function in the co-processor. The result has indicated that the AES-256 code has only used 35 instructions (56.45%) out of the 62 instructions defined for a "RISC-V 64-bit Integer" implementation [14]. Further analysis has shown that the unused instructions belong to four separate categories: control status instructions, memory access instructions, data shift instructions, and logic OR instructions. The consequence of removing these instructions on other "house-keeping" functions of the co-processor will need to be investigated.

We found that, in this example, the effect of instruction removal on non-AES-256 related control operations (i.e., the house-keeping functions) is minimal. Although some functionality will be lost by removing control status instructions, such as software timing data, the processor still retains the ability to load, operate on, and store data. The operations of removed instructions can be emulated with the remaining instructions. For example, one of the data load instructions, LB (Load Byte), unused by AES-256, if removed, can be emulated by two to three remaining instructions, as shown in Figure 5. In this case, the overhead of emulation is negligible and thus acceptable.

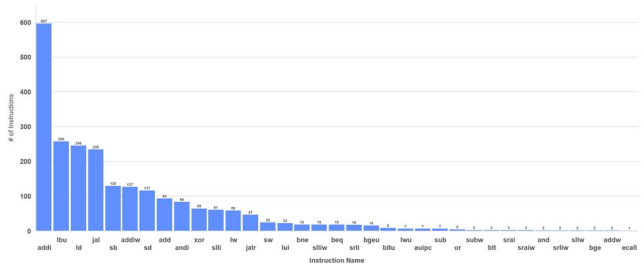

Figure 4. Instruction analysis of a 64-bit RISC-V for an AES 256 binary.

Removed instruction:

LB a0, 0(sp)

Required Emulation of load byte instruction:

LW a1, 0(sp)

ANDI a0, a1, 0x000000FF

Figure 5. Emulation example of removed instruction (LB: Load Byle, sp: stack pointer, LW: Load Word, ANDI: Logic AND with immediate value; a0 and a1: registers).

VIII. Summary and Ongoing Work

In summary, we have been developing a configurable and customizable hardware foundation platform that includes processing cores, and potentially tag engines, either separated from processing cores and/or embedded within them, and crypto engines, etc., to provide proper hardware security primitives.

A systematic approach has been developed and used to identify security requirements by considering mission specific operational sequence steps. A fast turnaround design environment has been created as a testbed. Currently, as a proof of concept, we have prototyped and demonstrated a co-processor which has security features to verify the authenticity and integrity of software patches to an exemplary distributed control system.

A number of interesting activities are ongoing. There are other interesting potential security benefits of a security co-processor with a customized instruction set. By strategically removing instructions, the unused opcode space becomes available to play a defensive role in the security co-processor. The removed opcodes can be implemented as traps and place the processor in a locked state when a malicious process compiled with a "full-sized" instruction set executes a trapping instruction.

With respect to the design environment, we will continue to streamline the process of generating application-specific security co-processors used as TCBs. The goal is to fully support the development of a secure-by-design methodology that strikes a balance between verifiability and performance. To further simplify the designing process, we are looking to develop an automated method to automatically generate a co-processor that meet specific TCB requirements. This will allow the designer to focus on TCB verification and interfacing it to the rest of the system.

Last but not least, we plan to expand the hardware security primitives in a few directions. Candidates are:

- Efficient tag processing to support beyond single-threaded, in-order operations.
- Embedded crypto primitives that allow applications to use crypto services without ever accessing the keys.
- Multiple-stepped bootstrapping and recovery to deliver multiple levels of functionality, from fully mission capable to severely degraded but still operational and others in between.

IX. References

[1] G. Klein et al., "seL4: Formal verification of an os kernel," *Proc. of the ACM SIGOPS 22nd SOSP'09*.

[2] https://www.wired.com/story/meltdown-spectre-bug-collision-intel-chip-flaw-discovery/, accessed 1 June 2020.

[3] https://developer.arm.com/ip-products/security-ip/trustzone, accessed 1 June 2020.

[4] https://community.arm.com/developer/ip-products/processors/b/processors-ip-blog/posts/enhancing-memory-safety, accessed 1 June 2020.

[5] V. Costan and S. Devadas, Intel SGX explained, https://eprint.iacr.org/2016/086.pdf.

[6] https://www.draper.com/explore-solutions/inherently-secure-processor, accessed 1 June 2020.

[7] https://www.cl.cam.ac.uk/research/security/ctsrd/cheri/, accessed 1 June 2020.

[8] https://www.xilinx.com/products/technology/ultrascale.html, accessed 1 June 2020.

[9] https://www.intel.com/content/www/us/en/products/programmable/fpga/cyclone-10.html, accessed 1 June 2020.

[10] https://www.synopsys.com/dw/ipdir.php?ds=asip-designer, assessed 1 June 2020.

[11] https://www.zdnet.com/article/microsoft-70-percent-of-all-security-bugs-are-memory-safety-issues/, accessed 1 June 2020.

[12] D. Whelihan, M. Vai, et al, "Designing Agility and Resilience into Embedded Systems," MILCOM 2017.

[13] D. Whelihan *et al.*, "SHAMROCK: A Synthesizable High Assurance Cryptography and Key management coprocessor," *MILCOM 2016 - 2016 IEEE Military Communications Conference*, Baltimore, MD, 2016, pp. 55-60, doi: 10.1109/MILCOM.2016.7795301.

[14] https://riscv.org/, accessed 1 June 2020.