Protecting Processors from Software Exploitation Cyberattacks: RISC-V Sets the Stage for a Streamlined CoreGuard® Integration

Steven Milburn
CTO and Co-Founder
Dover Microsystems, Inc.
Waltham, MA USA
steve@dovermicrosystems.com

***Abstract*—Dover Microsystems' CoreGuard® is the only solution for embedded systems that prevents the exploitation of software vulnerabilities and immunizes processors against entire classes of network-based attacks. It does this by hardwiring security directly into the silicon, providing a physical layer of protection that prevents an attacker's ability to take over the processor. CoreGuard technology is delivered as IP blocks that give processor and SoC designers a variety of options for using, modifying, or extending a RISC-V processor. The "right" option is based on many factors, such as PPA (Power, Performance, and Area) constraints and how much flexibility the designer has over the processor implementation. This paper describes options for integrating CoreGuard IP with a RISC-V host processor. While CoreGuard technology is architecture agnostic, the openness and privilege model of the RISC-V architecture sets a perfect stage for the most adaptable and streamlined methods of integration.**

Keywords—cybersecurity, cyberattacks, embedded systems, silicon IP, software vulnerabilities, exploits, RISC-V processor

I. Introduction

Today's processors blindly execute instructions, and do not have the knowledge to distinguish between good and bad behavior. Compounding this issue is the fact that most cyberattackers exploit bugs in software, and there are on average 15-50 bugs per thousand lines of delivered code [1]. With an unprecedented approach to cybersecurity, CoreGuard technology addresses both facets of this problem.

II. How CoreGuard Technology Works

CoreGuard is silicon IP that integrates with RISC processors to protect embedded systems from cyberattacks by enforcing security, safety, and privacy rules—called **micropolicies**—that precisely define allowed versus disallowed behavior at the per-instruction level. CoreGuard maintains micropolicy-relevant **metadata** about every word in memory, and then uses this metadata to crosscheck each instruction processed against the installed set of micropolicies. If an instruction violates any micropolicy, CoreGuard **Policy Enforcer** hardware quarantines the instruction's output and sends a micropolicy violation to the host processor.

First International Workshop on Secure RISC-V Architecture Design Exploration (SECRISC-V'20). It is held in conjunction with the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS) - August 23rd, 2020 in Boston, Massachusetts, USA.

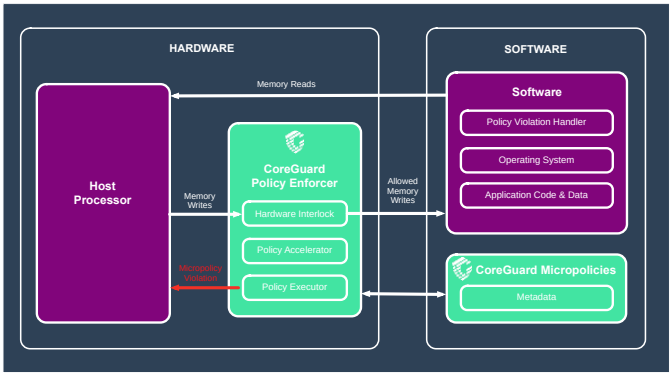

Fig. 1. CoreGuard High-Level Architecture

Fig. 1 is a notional representation of the CoreGuard architecture. Depending on the requirements and constraints of a design, CoreGuard technology can be instantiated entirely external to the host processor core, or it can be integrated within the host processor core logic. Before describing the various integration options, let's take a closer look at CoreGuard's two main components: micropolicies and the Policy Enforcer.

A. Micropolicies

With an estimated 111 billion lines of new software code produced each year [2], there is an ever-expanding universe of vulnerabilities for cyberattackers to exploit. Fortunately, there are a lot of publicly available and frequently updated data about specific vulnerabilities *and* categories of software weaknesses.

Dover uses two broadly-accepted databases maintained by The MITRE Corporation and sponsored by the U.S. Department of Homeland Security Cybersecurity and Infrastructure Security Agency: CVE (Common Vulnerabilities and Exposures) [3] and CWE (Common Weakness Enumeration) [4].

Informed by these databases, Dover writes targeted CoreGuard micropolicies to protect against classes of attack that exploit common categories of software weaknesses [5]. For example, CoreGuard Heap and Stack micropolicies are designed to block attacks that exploit buffer overflow vulnerabilities. It

doesn't matter how many new bugs are discovered within a particular category of weaknesses; micropolicies block the entire class of attack, whether it is associated with two vulnerabilities or two million.

Dover writes CoreGuard micropolicies in a domain-specific language, the Dover Policy Language (DPL). Because DPL is designed solely for the purpose of defining micropolicies, it includes specialized features and constructs that result in the most efficient code possible. A smaller codebase significantly decreases the likelihood for software bugs. Additionally, micropolicies and metadata are isolated from the rest of the system in a separate, protected section of memory that cannot be accessed by the system's operating system or its applications; only CoreGuard hardware can see and run CoreGuard micropolicies. CoreGuard micropolicies are also encrypted and code-signed, and installed using a secure boot process to guarantee the integrity of micropolicy code.

B. *Policy Enforcer*

The Policy Enforcer is comprised of multiple IP blocks, and the composition and design of those blocks will vary depending on the method of integration. We will look at the variances later, but it is helpful to first understand the main CoreGuard Policy Enforcer blocks shown in Fig. 1: Hardware Interlock, Policy Accelerator, and Policy Executor.

The **CoreGuard Hardware Interlock** serves as the gate keeper between the host processor and the memory system, including memory-mapped peripherals. In some integrations, this block is either not needed or is replaced by a different block that manages memory writes.

The **CoreGuard Policy Accelerator** is needed in any type of integration. It optimizes the performance of CoreGuard by maintaining multiple caches, including a rule cache that stores rule processing data so that future requests for data can be served faster. The Policy Accelerator is responsible for performing rule cache lookups for each instruction.

The **CoreGuard Policy Executor** executes the micropolicy code to generate rules for the Policy Accelerator. Policy Executor functionality is needed with any integration, but can be designed to use the host processor core, a dedicated processor core, or a portion of an existing support processor (such as a security core).

III. Types of CoreGuard Integrations

We divide CoreGuard integration options into two main categories: **trace-based** and **direct pipeline**. In a trace-based integration, the CoreGuard Policy Enforcer needs a few signals from the host processor but does not require any changes to the processor pipeline. With a direct pipeline integration, CoreGuard IP blocks are incorporated directly into the processor pipeline.

With both trace-based and direct pipeline integrations to a RISC-V processor, the CoreGuard technology takes advantage of key aspects of the RISC-V ISA. For example, the Physical Memory Protection (PMP) feature prevents destructive-read attacks/abuses by ensuring that a peripheral with destructive read registers is readable only by the appropriate thread.

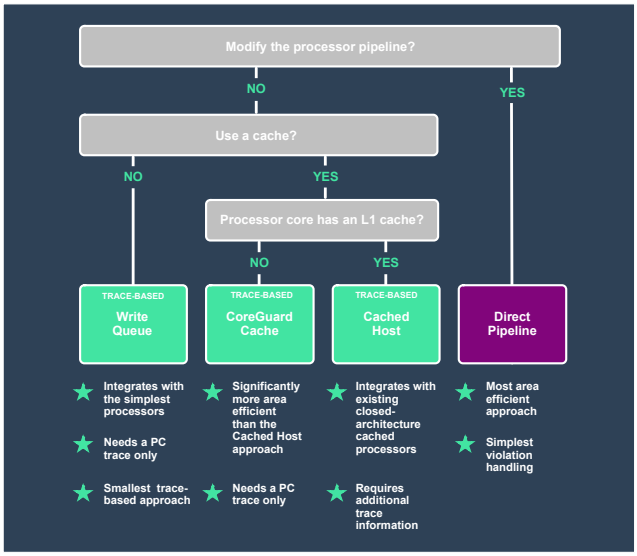

Fig. 2. Choosing a CoreGuard Integration Approach

Fig. 2 summarizes some decision points for choosing a CoreGuard integration approach. Next, we'll take a closer look at each of these approaches.

IV. Trace-Based Integrations

With a trace-based integration, CoreGuard Policy Enforcer IP blocks are wired up next to an existing host processor. This makes it a suitable approach for SoC designers working with a processor they cannot modify. Trace-based approaches—with various caching and trace buffer requirements—let a designer achieve different levels of power, performance, and area efficiency. There are three trace-based integration approaches: Cached Host, CoreGuard Cache, and Write Queue.

A. *Cached Host Approach*

With a Cached Host integration, CoreGuard Policy Enforcer IP blocks integrate with a host core that includes its own L1 caches. This approach uses a trace buffer to create an elastic interface between the host processor and the CoreGuard Accelerator.

The trace buffer captures the host processor's instruction and data trace for validation of the retired instruction by the CoreGuard Policy Accelerator. In addition, the Policy Accelerator has its own L1 metadata cache to match the caching performance of the cached host. Ideally, the geometry, capacity, and eviction strategy (replacement policy) of the metadata cache will match that of the host processor's cache, but the Policy Accelerator's trace buffers are able to cushion any tag cache misses from affecting overall system performance.

The Policy Accelerator is highly configurable, with settings to change the geometry and capacity of the metadata caches. The eviction strategy is not yet configurable, but will be in a future CoreGuard version that can integrate with different host processors that employ a variety of eviction strategies.

In a Cached Host configuration, CoreGuard does not protect writes to the host processor's embedded L1 cache, but employs additional measures to ensure that a violation affects only the current execution thread.

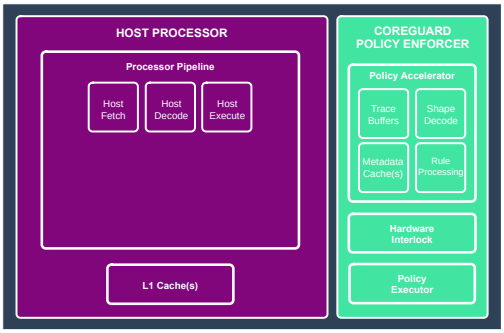

Fig. 3. Cached Host Trace-Based Integration

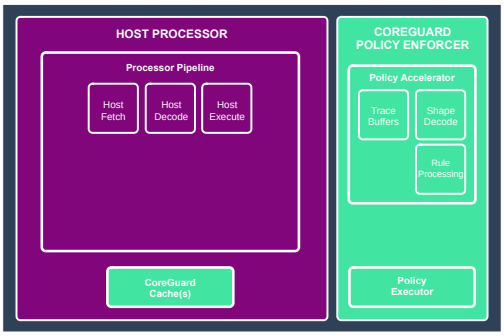

Fig. 4. CoreGuard Cache Trace-Based Integration

While a Cached Host approach enables CoreGuard technology to integrate to existing high-performance cores, it is the least area efficient integration option due to the additional caches and replication of pipeline and decode logic.

B. CoreGuard Cache Approach

With this approach, an uncached host processor combines with a CoreGuard Cache to create a cached host processor. The CoreGuard Cache caches data and metadata together, using a coherent write queue to protect stores to the cached data until associated instructions are validated. Violation handling code is significantly less complex than with the Cached Host approach, but still provides only an asynchronous exception that results in killing an entire current execution thread.

The CoreGuard Cache works with the CoreGuard Accelerator to provide instruction words and data addresses for metadata processing, enabling integration with processors containing a minimal/filtered trace of only the retired PC stream. The CoreGuard Accelerator therefore doesn't require its own L1 metadata caches, significantly reducing overall area and bus fabric complexity. As shown in Fig. 4, this approach needs only a small trace buffer because the CoreGuard Cache maintains metadata with instruction/data lines. This minimizes memory traffic and cache-miss penalties by fetching data and metadata together.

C. Write Queue Approach

With the Write Queue approach, CoreGuard IP can integrate to the simplest of processor cores, requiring nothing more than a PC trace. CoreGuard does not require a cache, but instead uses a CoreGuard Write Queue to prevent writes to the memory system while the trace-based CoreGuard Policy Accelerator verifies retired instructions.

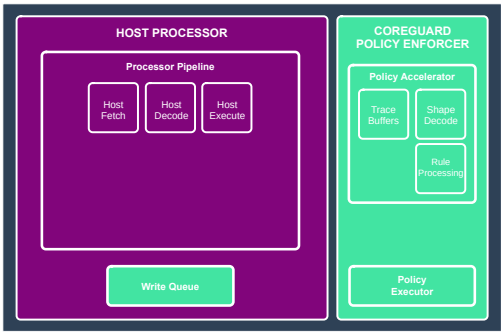

Fig. 5. Write Queue Trace-Based Integration

CoreGuard sniffs instruction reads from the processor and uses the Write Queue to capture a history of the instruction addresses and instruction words. When the PC trace is presented from the output of the processor, CoreGuard matches up instruction words based on the instruction address, discarding buffered instruction words until it finds a match to the next retired PC. This enables CoreGuard to gain access to the instruction word without adding memory bandwidth, and without modification to the processor to present the instruction word at the trace output.

Data access of the processor goes through the CoreGuard Write Queue, which sends a copy of store addresses to the CoreGuard Accelerator for use in vetting store instructions against the installed set of micropolicies. Once a store instruction is vetted, the store is forwarded to the memory system. The CoreGuard Write Queue is pipelined to match the pipeline depth of the CoreGuard Accelerator and prevent any reduction in memory bandwidth. In addition, the CoreGuard Write Queue maintains memory order and coherency between read and write interfaces.

A Write Queue integration results in the smallest type of trace-based CoreGuard implementation. With no caching effects, it has the lowest power consumption. Additionally, this approach benefits from the same savings in violation handling complexity as a CoreGuard Cache integration—primarily because it doesn't have to deal with figuring out which dirty words from the cache to save to memory and which to discard.

V. Direct Pipeline Integrations

The openness of the RISC-V architecture presents designers with the best opportunity for a tightly-coupled CoreGuard integration. In a Direct Pipeline integration, CoreGuard Policy Enforcer IP blocks are embedded directly into the processor pipeline using simple interfaces that hook into common processor stages to handle metadata processing in lock-step with instruction execution of the processor. Minimally, these interfaces will pass in 1) a stalling mechanism for when the CoreGuard logic requires more cycles than the native processor logic, and (2) a synchronous exception mechanism to invoke an exception handler rather than retire an instruction during the final Write-Back stage of the pipeline. Each block can be pipelined to additional cycles to match up with a processor's pipeline and clock period requirement. A Direct Pipeline integration can be done as a cache-less design or using the CoreGuard Cache block.

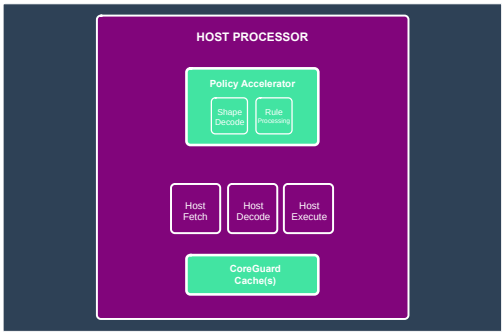

Fig. 6. Direct Pipeline Integration

As Fig. 6 illustrates, the Shape Decode and Rule Processing stages are smaller than in a trace-based integration because they rely on the host processor control pipeline, rather than re-implement their own.

A Direct Pipeline integration enables CoreGuard to present policy violations as synchronous exception events in the pipeline. Rather than killing an entire execution thread after there is a violation, exceptions are handled immediately in the context of the specific instruction that triggered the violation.

Fig. 6 also shows that there is no separate Policy Executor in a Direct Pipeline integration. The job of executing micropolicy code is handled by the host processor in M-mode (machine mode). M-mode is ideal for running security processes because it is RISC-V's highest privilege mode and not interruptible by lower modes.

Running in M-mode also gives the Policy Enforcer access to RISC-V's PMP. As mentioned earlier, all CoreGuard integrations use PMP to prevent destructive-read attacks/abuses, but with a Direct Pipeline integration, PMP can be leveraged to correct a destructive-read violation without having to resort to killing an entire thread.

All these factors combine to make a CoreGuard Direct Pipeline implementation the simplest and most area efficient design of a secure processor.

VI. PPA and Memory Requirements

Integrating CoreGuard directly with the processor pipeline provides the greatest opportunities for area efficiency, but both Direct Pipeline and trace-based approaches enable SoC and processor designers to balance the costs and benefits of embedding security into their systems. Exact requirements for power, performance, area (PPA) and memory will depend on SoC specifications and the selected set of micropolicies, but Table 1 describes typical requirements based on a trace-based integration with a microcontroller host processor target synthesized with a Global Foundries 28nm HPP process. Note that the requirements with an LPP process would be significantly different.

With a Direct Pipeline integration, a simple five-stage, in order pipeline RISC-V RV32I core grows by approximately 30%. The overhead on a more complex RISC-V core would be even lower since the CoreGuard area stays fixed while the overall processor's area grows.

TABLE I. CoreGuard PPA and Memory Requirements

Power	CoreGuard's total power consumption is 21.9 milliwatts with a 500 MHz clock frequency. (21.9 mW @ 500 MHz, 33.0 mW @ 750 MHz, 44.1 mW @ 1000 MHz)
Performance	There is little to no impact on performance when running with CoreGuard's base set of micropolicies (RWX, Heap, and Stack).
Area	CoreGuard's total synthesized area is 0.121 square millimeters with a 500 MHz clock frequency. (0.121 mm2 @ 500 MHz, 0.123 mm2 @ 750 MHz, 0.130 mm2 @ 1000 MHz)
Memory	Size of micropolicy metadata is a function of which micropolicies are installed on a system and the system's memory map. Roughly, metadata will consume 20-25% of the memory footprint.

VII. Conclusion

The CoreGuard solution addresses our cybersecurity problem at the root cause: the attackers' ability to exploit software vulnerabilities in order to hijack processors and get them to do their bidding. Because it is hardwired directly into the silicon, CoreGuard is unassailable and cannot be subverted over the network. Adding any hardware IP comes at a price, however, and SoC and processor designers strive to balance the costs and benefits of embedding security into their systems. CoreGuard is architected to help designers with this critical balancing act.

For the SoC designer implementing either a soft- or hard-core RISC-V processor, there are a variety of trace-based integration approaches that do not require any modifications to the processor pipeline. This flexibility allows designers to avoid vendor lock in and standardize on CoreGuard across a multitude of RISC-V processor offerings.

For the processor designer creating a new RISC-V processor—with full freedom to modify the processor implementation—there is a direct pipeline approach that results in the most efficient design and empowers designers to create a major competitive advantage by having the most robust cybersecurity solution on the market today.

Whatever the integration approach, CoreGuard technology and the RISC-V architecture are a perfect match for designing a processor core or SoC that optimizes power, performance, area, **and security**.

References

[1] S. McConnell, Code Complete, Second ed., Microsoft Press, 2004.

[2] Cybersecurity Ventures, "Application Security Report," Cybersecurity Ventures, Menlo Park, 2017.

[3] The MITRE Corporation, "Common Vulnerabilities and Exposures (CVE)," 05 February 2019. [Online]. Available: https://cve.mitre.org/index.html.

[4] The MITRE Corporation, "Common Weakness Enumeration (CWE)," The MITRE Corporation, 20 February 2020. [Online]. Available: https://cwe.mitre.org/.

[5] Dover Microsystems, "CoreGuard Cybersecurity Scorecard," February 2020. [Online]. Available: https://app.hubspot.com/documents/3382333/view/64518455?accessId=ca5f29.