HERQULES: Securing Programs via Hardware-Enforced Message Queues

Extended Abstract

Daming D. Chen
Phillip B. Gibbons
Wen Shih Lim
James C. Hoe
Carnegie Mellon University

1 Motivation

Computer programs written in unsafe languages directly manipulate memory using unbounded pointers, which may introduce memory safety bugs [26]. In response, past work has developed various runtime defenses, including memory safety checks [24] as well as mitigations like stack canaries [7], no-execute memory [27], and control-flow integrity [4] (CFI), which validates runtime control-flow transitions against an expected control-flow graph (CFG). However, these runtime defenses may need to update runtime metadata to maximize accuracy, which is difficult to do precisely, efficiently, and securely.

In this paper, we present an efficient solution by adding a fast hardware-based append-only inter-process communication (IPC) primitive, named AppendWrite, which leverages existing inter-process memory protections. Our approach resembles that of fine-grained instruction monitoring [2, 5, 8, 10], which modifies the processor to send a log of execution events elsewhere for analysis, except that we avoid significant hardware change, reduce performance overhead, and maximize flexibility with software-defined events.

2 Past Work

Existing proposals for fine-grained instruction monitoring have significant drawbacks, as shown in Table 1. All require significant microarchitectural change to generate, filter, and process events in hardware, which indicate, e.g., retired instructions, function calls, or memory accesses. For example, Guardian Council [2] adds up to 24 dedicated microcontroller-sized cores (μCores) to process events, whereas FlexCore [8] adds an on-chip FPGA. These designs generate fixed hardware-defined events, which may not be used by all software policies, yet nevertheless incur both energy and logic costs. For example, under FADE [10], hardware must ultimately filter and discard 84%–99% of all events as irrelevant.

Although Processor Trace [1] (PT) is included by many Intel processors, it is designed for performance monitoring, and not as a security mechanism. Event packets can be lost or overwritten due to, e.g., interrupt skid, which defeats security. PT incurs tremendous overhead and has limited support for software-defined events via the PTWRITE instruction. Past PT-based CFI approaches [9, 11, 13, 20] have measured over 500x overhead [20] for tracing/decoding hardware-defined events on the SPEC benchmarks; as a result, they limit CFI checks to 7-10 system calls (e.g., execve, mmap, etc.), which are rarely called by compute-heavy benchmarks like SPEC.

3 Key Insights

We introduce a simple AppendWrite IPC primitive that provides both authentication and integrity security properties for messages transmitted from a monitored program to a verifier process. Using our AppendWrite primitive, we build HERQULES, a framework for efficiently enforcing program integrity, as shown in Figure 1. Our approach uses compiler instrumentation to insert runtime AppendWrite calls into the instrumented program (1a), which transmit software-defined policy events to the verifier (2a, 2b, 3a). Since the instrumented program begins execution in a benign state, and its code is read-only, it must send a message containing evidence of a policy violation before it occurs. Even if the program is later compromised, AppendWrite ensures that this evidence cannot be retracted. We maximize performance by executing the verifier concurrently with the instrumented program, and synchronize only at the program’s system calls (3b). To prevent externally-visible side effects from a corrupted program, we employ bounded validation; i.e., we use a kernel module to pause execution of the instrumented program until the verifier confirms that no policy checks have failed (4a, 4b).

Using HERQULES, we develop various security policies, including HQ-CFI, a state-of-the-art pointer integrity [15, 21] CFI design, as well as others for, e.g., memory safety. Our HQ-CFI design supplements pointer integrity with invalidation to detect use-after-free bugs (UAF), which is not supported by past work and infeasible under approaches such as cryptographic MACs [21], and we evaluate its correctness, effectiveness, and performance against related work.
4 Main Artifacts

We develop two designs for AppendWrite: one in an FPGA-based PCIe programmable accelerator [17, 22], named **AppendWrite-FPGA**, and another in the microarchitecture itself, named **AppendWrite-μarch**. Our accelerator-based design is compatible with existing systems, synthesizes a simple append-only message queue from programmable logic, and accepts messages from memory-mapped I/O writes. Our microarchitectural design extends each processor core to support an appendable memory region by introducing two new privileged registers, one new instruction to the ISA, and some additional logic to TLB lookups. In comparison to our FPGA-based design, it reduces overhead by decreasing round-trip time and leveraging existing hardware mechanisms for caching and out-of-order execution.

We implement **HerQules**, our framework for integrity-based security policies, which we have made available online as open-source software\(^1\). It is composed of four different components: our AppendWrite primitive, compiler instrumentation passes, a runtime policy verifier, and a kernel module. We evaluate our pointer integrity design, HQ-CFI, on a suite of benchmark programs, including RIPE [23, 28], SPEC CPU2006 [14], SPEC CPU2017 [3], and NGINX [25].

5 Key Results and Contributions

As a micro-benchmark, we compare the security and performance of various IPC primitives to AppendWrite in Table 2, and demonstrate that only AppendWrite ensures append-only messages with high performance. Software-based primitives either lack performance (not asynchronous) or message integrity (not append-only). System calls execute synchronously on the calling thread, and incur a privilege transition that flushes hardware caches, especially after recent kernel page-table isolation [12] mitigations for microarchitectural side channels [18]. Traditional workarounds, like client-side buffering, would violate message integrity by allowing alteration or erasure of in-flight messages.

We evaluate two variants of our HQ-CFI design, both of which achieve superior precision, benchmark correctness, and runtime performance when compared to past work. On the RIPE testsuite of buffer overflow exploits, **HQ-CFI-RETPtr** checks all stack return pointers, rendering it invulnerable to all exploits, whereas **HQ-CFI-SFESTR** uses a *safe stack* [15] that relies on information hiding, trading-off precision for performance. Both detect use-after-frees, which allowed us to identify and fix undetected memory safety bugs in the SPEC benchmarks.

Our designs do not affect benchmark correctness, as shown in Table 3, whereas past works cause crashes/hangs (errors), emit false positives (false Ps.), and generate incorrect output (invalid), which we attempted to manually fix in their implementations. In Table 4, we quantify performance by computing a baseline-relative geometric mean across our SPEC CPU2006, SPEC CPU2017, and NGINX benchmarks, and use a software-only model for AppendWrite-μarch as a lower-bound estimate. Our design is compatible with existing libraries, and does not require masking of all pointers.

---

\(^1\)https://github.com/secure-foundations/herqules
References


