HerQules: Securing Programs via Hardware-Enforced Message Queues

Extended Abstract

Daming D. Chen Wen Shih Lim Mohammad Bakhshalipour
Phillip B. Gibbons James C. Hoe Bryan Parno
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-grain 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.

Design	**Events**	**Recip.**	**Paradigm**	**HW Δ**
FADE [10]	HW/SW	Core	Filter-Update	Big
FlexCore [8]	HW/SW	FPGA	Reconfigure	Big
Guardian Council [2]	HW/SW	μCores	Filter-Map-Red.	Big
LBA [5]	HW/SW	Core	Filter-Update	Big
Processor Trace [1]	HW/SW	Mem.	Filter-Update	–
HerQules	SW	Core	Message Passing	Small

Table 1. Comparison of fine-grained instruction monitoring.

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.

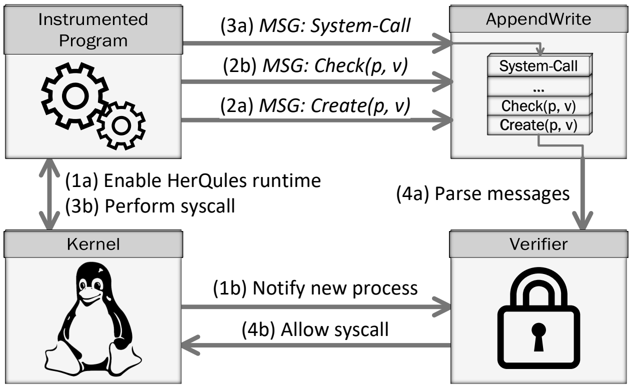

Figure 1. Runtime overview of HerQules, showing interactions between the instrumented program, AppendWrite primitive, kernel module, and verifier for the HQ-CFI policy.

IPC Primitive	Append Only	Async.	Cost	Time (ns)
Message Queue	✓	×	System Call	146
Named Pipe	✓	×	System Call	316
Socket	✓	×	System Call	346
Shared Memory	×	✓	Mem. Write	12
Light-Weight Contexts	✓	×	System Call	2010 [19]
AppendWrite-FPGA	✓	✓	Mem. Write	102
AppendWrite-μarch	✓	✓	Mem. Write	2

Table 2. Comparison of measured IPC primitives, by type (*top*: software, *center*: hardware, *bottom*: proposed).

Design	Errors	False Ps.	Invalid	Ok
Baseline	0	0	0	48
Clang/LLVM CFI [6]	0	15	0	33
CCFI [21]	12	29	9	19
CPI [15, 16]	14	0	14	34
HQ-CFI	0	0	0	48

Table 3. Correctness of evaluated control-flow integrity designs, by possibly non-exclusive category.

Design	Mechanism	Prec.	UAF	Compat.	Perf.
Clang/LLVM CFI [6]	Language-level Types	●	×	●●	94%
CCFI [21]	Cryptographic MACs	●●●	×	●	49%
CPI [16]	Software Fault Isolation	●●	×	●	96%
HQ-CFI-SfeStk-Model	AppendWrite	●●	✓	●●●	87%
HQ-CFI-RetPtr-Model	AppendWrite	●●●	✓	●●●	55%

Table 4. Comparison of evaluated control-flow integrity designs, by precision (*top*: low, *center/bottom*: high). More ● is better.

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-SfeStk** 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

[1] 2020. Intel® 64 and IA-32 Architectures Software Developer's Manual. https://software.intel.com/content/www/us/en/develop/articles/intel-sdm.html

[2] Sam Ainsworth and Timothy M. Jones. 2020. The Guardian Council: Parallel Programmable Hardware Security. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '20)*. Association for Computing Machinery, Lausanne, Switzerland, 1277–1293. https://doi.org/10.1145/3373376.3378463

[3] James Bucek, Klaus-Dieter Lange, and Jóakim v. Kistowski. 2018. SPEC CPU2017: Next-Generation Compute Benchmark. In *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering (ICPE '18)*. Association for Computing Machinery, Berlin, Germany, 41–42. https://doi.org/10.1145/3185768.3185771

[4] Miguel Castro, Manuel Costa, and Tim Harris. 2006. Securing Software by Enforcing Data-Flow Integrity. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI '06)*. USENIX Association, Berkeley, CA, USA, 147–160. https://doi.org/10.5555/1298455.1298470

[5] Shimin Chen, Michael Kozuch, Theodoros Strigkos, Babak Falsafi, Phillip B. Gibbons, Todd C. Mowry, Vijaya Ramachandran, Olatunji Ruwase, Michael Ryan, and Evangelos Vlachos. 2008. Flexible Hardware Acceleration for Instruction-Grain Program Monitoring. In *2008 International Symposium on Computer Architecture*. 377–388. https://doi.org/10.1109/ISCA.2008.20

[6] Peter Collingbourne. 2015. Control Flow Integrity Design Documentation. https://clang.llvm.org/docs/ControlFlowIntegrityDesign.html

[7] Crispin Cowan, Calton Pu, Dave Maier, Heather Hintony, Jonathan Walpole, Peat Bakke, Steve Beattie, Aaron Grier, Perry Wagle, and Qian Zhang. 1998. StackGuard: Automatic Adaptive Detection and Prevention of Buffer-Overflow Attacks. In *Proceedings of the 7th Conference on USENIX Security Symposium - Volume 7 (SSYM'98)*. USENIX Association, San Antonio, Texas, 5.

[8] Daniel Y. Deng, Daniel Lo, Greg Malysa, Skyler Schneider, and G. Edward Suh. 2010. Flexible and Efficient Instruction-Grained Run-Time Monitoring Using On-Chip Reconfigurable Fabric. In *2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*. 137–148. https://doi.org/10.1109/MICRO.2010.17

[9] Ren Ding, Chenxiong Qian, Chengyu Song, William Harris, Taesoo Kim, and Wenke Lee. 2017. Efficient Protection of Path-Sensitive Control Security. In *Proceedings of the 26th USENIX Conference on Security Symposium (SEC'17)*. USENIX Association, Vancouver, BC, Canada, 131–148.

[10] Sotiria Fytraki, Evangelos Vlachos, Onur Kocberber, Babak Falsafi, and Boris Grot. 2014. FADE: A Programmable Filtering Accelerator for Instruction-Grain Monitoring. In *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*. 108–119. https://doi.org/10.1109/HPCA.2014.6835922

[11] Xinyang Ge, Weidong Cui, and Trent Jaeger. 2017. GRIFFIN: Guarding Control Flows Using Intel Processor Trace. *ACM SIGARCH Computer Architecture News* 45, 1 (April 2017), 585–598. https://doi.org/10.1145/3093337.3037716

[12] Daniel Gruss, Moritz Lipp, Michael Schwarz, Richard Fellner, Clémentine Maurice, and Stefan Mangard. 2017. KASLR Is Dead: Long Live KASLR. In *Engineering Secure Software and Systems (Lecture Notes in Computer Science)*, Eric Bodden, Mathias Payer, and Elias Athanasopoulos (Eds.). Springer International Publishing, Cham, 161–176. https://doi.org/10.1007/978-3-319-62105-0_11

[13] Yufei Gu, Qingchuan Zhao, Yinqian Zhang, and Zhiqiang Lin. 2017. PT-CFI: Transparent Backward-Edge Control Flow Violation Detection Using Intel Processor Trace. In *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy (CODASPY '17)*. Association for Computing Machinery, Scottsdale, Arizona, USA, 173–184. https://doi.org/10.1145/3029806.3029830

[14] John L. Henning. 2006. SPEC CPU2006 Benchmark Descriptions. *ACM SIGARCH Computer Architecture News* 34, 4 (Sept. 2006), 1–17. https://doi.org/10.1145/1186736.1186737

[15] Volodymyr Kuznetsov, László Szekeres, Mathias Payer, George Candea, R. Sekar, and Dawn Song. 2014. Code-Pointer Integrity. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation (OSDI'14)*. USENIX Association, 147–163. https://doi.org/10.5555/2685048.2685061

[16] Volodymyr Kuznetsov, Laszlo Szekeres, Mathias Payer, George Candea, and Dawn Song. 2015. Poster: Getting The Point(Er): On the Feasibility of Attacks on Code-Pointer Integrity. In *2015 IEEE Symposium on Security and Privacy*. IEEE, 2.

[17] Liu Ling, Neal Oliver, Chitlur Bhushan, Wang Qigang, Alvin Chen, Shen Wenbo, Yu Zhihong, Arthur Sheiman, Ian McCallum, Joseph Grecco, Henry Mitchel, Liu Dong, and Prabhat Gupta. 2009. High-Performance, Energy-Efficient Platforms Using in-Socket FPGA Accelerators. In *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA '09)*. Association for Computing Machinery, Monterey, California, USA, 261–264. https://doi.org/10.1145/1508128.1508172

[18] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. 2018. Meltdown: Reading Kernel Memory from User Space. In *Proceedings of the 27th USENIX Conference on Security Symposium (SEC'18)*. USENIX Association, Baltimore, MD, USA, 973–990.

[19] James Litton, Anjo Vahldiek-Oberwagner, Eslam Elnikety, Deepak Garg, Bobby Bhattacharjee, and Peter Druschel. 2016. Light-Weight Contexts: An OS Abstraction for Safety and Performance. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16)*. USENIX Association, Berkeley, Calif.

[20] Yutao Liu, Peitao Shi, Xinran Wang, Haibo Chen, Binyu Zang, and Haibing Guan. 2017. Transparent and Efficient CFI Enforcement with Intel Processor Trace. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 529–540. https://doi.org/10.1109/HPCA.2017.18

[21] Ali Jose Mashtizadeh, Andrea Bittau, Dan Boneh, and David Mazières. 2015. CCFI: Cryptographically Enforced Control Flow Integrity. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS '15)*. ACM, New York, NY, USA, 941–951. https://doi.org/10.1145/2810103.2813676

[22] Neal Oliver, Rahul R. Sharma, Stephen Chang, Bhushan Chitlur, Elkin Garcia, Joseph Grecco, Aaron Grier, Nelson Ijih, Yaping Liu, Pratik Marolia, Henry Mitchel, Suchit Subhaschandra, Arthur Sheiman, Tim Whisonant, and Prabhat Gupta. 2011. A Reconfigurable Computing System Based on a Cache-Coherent Fabric. In *2011 International Conference on Reconfigurable Computing and FPGAs*. 80–85. https://doi.org/10.1109/ReConFig.2011.4

[23] Hubert Rosier. 2019. RIPE64. National University of Singapore. https://github.com/hrosier/ripe64

[24] Dokyung Song, Julian Lettner, Prabhu Rajasekaran, Yeoul Na, Stijn Volckaert, Per Larsen, and Michael Franz. 2019. SoK: Sanitizing for Security. In *2019 IEEE Symposium on Security and Privacy (SP)*. 1275–1295. https://doi.org/10.1109/SP.2019.00010

[25] Igor Sysoev. 2020. NGINX. Nginx, Inc.. https://www.nginx.com/

[26] László Szekeres, Mathias Payer, Tao Wei, and Dawn Song. 2013. SoK: Eternal War in Memory. In *2013 IEEE Symposium on Security and Privacy*. 48–62. https://doi.org/10.1109/SP.2013.13

[27] The PaX Team. 2003. Address Space Layout Randomization. https://pax.grsecurity.net/docs/aslr.txt

[28] John Wilander, Nick Nikiforakis, Yves Younan, Mariam Kamkar, and Wouter Joosen. 2011. RIPE: Runtime Intrusion Prevention Evaluator.

In *Proceedings of the 27th Annual Computer Security Applications Conference (ACSAC '11)*. Association for Computing Machinery, Orlando, Florida, USA, 41–50. https://doi.org/10.1145/2076732.2076739