Tradeoffs in Secure Accelerator Designs

Masab Ahmad, Omer Khan
University of Connecticut, Storrs, CT, USA
{masab.ahmad, khan}@uconn.edu

***Abstract*—Hardware accelerators have emerged as an effective mechanism to design ultra-efficient architectures. However, as accelerators get more and more intelligent, virtual and physical address environments become susceptible to security vulnerabilities. Control flow and address dependent data accesses could result in an accelerator overwriting undesired address space. Such anomalies should be considered as buffer-overflow style attacks. Conventional approaches such as capability systems do base and bound checks on data structures, while maintaining spatio-temporal security. Further optimizations for security selectively check metadata for susceptible data structures to improve performance. We show how allocating more area (to exploit parallelism) within the accelerator design can negate performance overheads associated with security checks.**

I. Introduction

Stringent performance and energy requirements are forcing the rise of dedicated hardware accelerators for crucial applications [7]. These accelerators read data from I/Os into shared memory, and then perform computations that can be easily unfolded to ensure computational parallelism [5]. Well known examples include sensor processing and mapping of graphic processing style applications onto a smaller form factor [8]. Compaction of design generally inhibits separation of memory space, and all these accelerators end up using a global memory space. This gives rise to accelerators reading and writing from the same global address space, which presents potential security vulnerabilities.

Data used by accelerators is generally read from the I/O ports, where nefarious entities can push malicious data [3] and potentially cause buffer overflows and unwanted accesses in private memory. Figure 1 shows how an address/data dependency can cause a potential overflow to private memory space. An accelerator reads data from an I/O port, and the address where this data is written is also determined by some function of the input data computed through some dataflow within the accelerator. An attacker can therefore insert a value through the I/O port that can cause the accelerator to write into a private memory space being used by other trusted applications, e.g. a kernel running an RSA or AES algorithm.

Bound checking is a well known scheme used to detect and thwart memory violations [6]. It associates base and bounds metadata to chunks of memory space. All read/write data accesses are first checked whether they conform to the security metadata for the associated memory chunk, and then they are committed to memory. Bound checking does not suffer from high memory overheads and false positive rates. However, it does have performance overheads associated with fetching bounds metadata and comparing it with the current data access

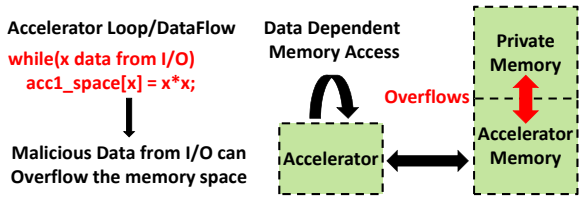

Fig. 1. Potential memory violation vulnerabilities in hardware accelerators.

procedure. We therefore propose to use and characterize bound checking for secure accelerator design space exploration.

A. Accelerator Threat Model

There are a number of ways an accelerator can potentially corrupt a shared memory space. We assume a memory space being shared by a slave accelerator and another master device, possibly a processor CPU. Our trusted computing base only encompasses hardware, primarily targeting embedded systems style processors running without an OS or any memory management unit. Such hardware can only be secured at design time, and thus architectural design and pre-synthesis analysis is key for conformation. We assume that an attacker is ideal, and can send any quantity and type of malicious data through the I/O channel.

II. Bounds Checking in Accelerator Designs

Accelerator designs for applications require a large amount of bound checks for each data structure read/write access to ensure sustainable security. This requires both checks on all reads and writes to provide full memory safety. Bounds metadata is read into an accelerator's registers, and then a comparative check is performed that determines whether the data access or address de-reference is within the bound or not. Passing this check allows the accelerator to continue with its execution, while failing the check is expected to raise an exception. Security heuristics only check bounds metadata for perceived vulnerable data structures, such as those being directly or indirectly accessed by an I/O channel. This strategy improves performance by removing redundant or unnecessary checks.

III. Methods

In this section we discuss how we add security checks within an accelerator design space exploration framework. We then explain the applications and analyze security versus efficiency tradeoffs.

TABLE I
ALADDIN ARCHITECTURAL PARAMETERS FOR EVALUATION.

Architectural Parameters		
Operation	Cycles	Area (um2)
ADD	1	280
MULT	1	4595
MEM-OP	6	7.98
CMP	1	200
Aladdin Para.	Type	Partit./Unroll.
Array Partit.	Cyclic	4 - 32768
Loop Unroll.	Non-Flat	4 - 32768
Hardware Pipelining = Off		

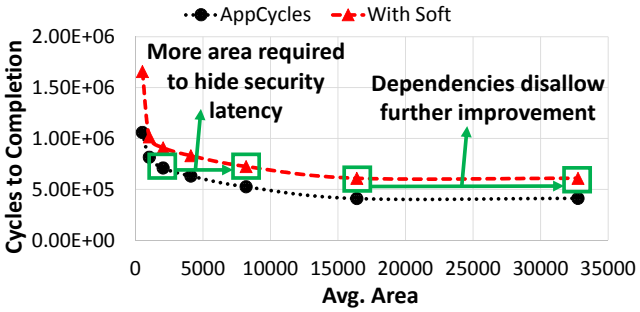

Fig. 2. Latency Improvements for **SSSP** by accelerating Security Checks.

A. Secure Accelerator Modeling

We modify the Aladdin simulator to generate accelerator designs [7]. To incorporate security checks in accelerator designs, we use the SoftBound-CETS tool [6] to identify vulnerabilities in an application, and then use it to add wrapper functions to secure the vulnerabilities. Application data is first compiled with SoftBound-CETS, after which an application trace is generated to be evaluated by Aladdin. In the case of security checks, a single cycle is assumed for each comparative check (CMP). We assume that the metadata is already present in the accelerator's registers once loaded from memory, and can be immediately compared with the address/data the accelerator intends to use. Table I shows the architectural parameters for Aladdin. Parameters include array partitioning between parallel datapaths within an accelerator, as well as loop unrolling to divide loop iterations between these parallel datapaths.

We optimize several parameters in the Aladdin simulator to minimize security overheads. To improve performance, we assume that all data loads that are accessed after a softbound check, are prefetched during the security check. This hides all latencies associated with softbound loads and metadata compares. In the case of store latencies, only the types that are expected to have a known outgoing load operation, such as in loops and function calls, can be hidden. Stores showing true dependencies, and operations whose addresses cannot be resolved earlier, their latencies are not hidden.

We use **SSSP**, a graph workload that finds shortest paths from the CRONO suite [1]. For each benchmark, we measure timing (in cycles) and power components for the baseline application, as well as for bounds checking separately.

IV. Secure Accelerator Design Tradeoffs

In this section we initially discuss performance overheads due to additional security checks for the proposed secure accelerator design. We then discuss how adding extra area/power can allow security checks to observe latency benefits from effective parallelization. All completion time results are for accelerator designs that have been partitioned and unrolled by a factor of 16. However, the area analysis results show designs partitioned and unrolled in the range 4 to 32,768. We choose a factor of 16 as it enables a parallelization versus area/power tradeoff for the dataflow within the accelerator design.

Now we show how a secure accelerator design can tradeoff latency overheads due to security checks. As shown in Figure 2, the case of **SSSP** is somewhat different. This application has a lot of read after write dependencies, which cannot be effectively parallelized within an accelerator design. Such applications also have more temporal checks for security, and future values are dependent on prior results, which are only available at runtime. Hence, adding area via parallelization does not improve cycles to completion time after a certain threshold. However, we also observed design points for **SSSP** that could benefit from additional area/power overheads to reduce the latency overheads. Similar tradeoffs can be observed in other works as well [4] [2].

V. Conclusion and Future Work

In this paper we propose how malicious entities can threaten accelerator centric architectures, where accelerators with conventional security vulnerabilities work in conjunction with a processor. We show how adding area/power by parallelizing security checks can overcome the latency overheads. This technique can be used to design secure accelerators that approach completion times of a baseline accelerator with no security mechanism.

References

[1] M. Ahmad and et. al., "Crono : A benchmark suite for multithreaded graph algorithms executing on futuristic multicores," in *Proceedings of the IEEE International Symposium on Workload Characterization*, ser. IISWC '15. IEEE, 2015.

[2] C. Fletchery and et. al., "Suppressing the oblivious ram timing channel while making information leakage and program efficiency trade-offs," in *High Performance Computer Architecture (HPCA), 2014 IEEE 20th International Symposium on*, Feb 2014, pp. 213–224.

[3] H. Hu and et. al., "Automatic generation of data-oriented exploits," in *24th USENIX Security Symposium (USENIX Security 15)*. Washington, D.C.: USENIX Association, Aug. 2015, pp. 177–192.

[4] O. Khan and et. al., "Exploring the performance implications of memory safety primitives in many-core processors executing multi-threaded workloads," HASP 2015.

[5] S. Kumar and et. al., "Dasx: Hardware accelerator for software data structures," in *Proceedings of the 29th ACM on International Conference on Supercomputing*, ser. ICS '15. ACM, 2015, pp. 361–372.

[6] S. Nagarakatte and et. al., "Softbound: Highly compatible and complete spatial memory safety for c," in *Proceedings of the 30th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI '09. New York, NY, USA: ACM, 2009, pp. 245–258.

[7] Y. S. Shao and et. al., "Aladdin: A pre-rtl, power-performance accelerator simulator enabling large design space exploration of customized architectures," in *Proceeding of the 41st Annual International Symposium on Computer Architecuture*, ser. ISCA '14. Piscataway, NJ, USA: IEEE, 2014, pp. 97–108.

[8] L. Wu and et. al., "Navigating big data with high-throughput, energy-efficient data partitioning," in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ser. ISCA. ACM, 2013, pp. 249–260.