Paper Review Form (1000 words maximum)
cs940: GhostRider: A Hardware-Software System for Memory Trace Oblivious Computation [1]

February 25, 2018

Paper Summary

In this paper, the authors present both a theoretical compilation model and its prototype hardware-software implementation to mitigate the severe performance penalties in using Oblivious RAM (ORAM) to protect sensitive remote computations from attackers with physical access to the infrastructure. The GhostRider solution exploits the lack of obliviousness requirements for memory events generating regular patterns, and holds their corresponding data structures in the faster encrypted RAM (ERAM) instead. Additional measures for runtime determinism were implemented to further prevent information leak. With the aid of an annotated C-style language, the authors demonstrated that their compiler could achieve semantically-provable security against physical attacks. GhostRider's performance improvement from pure-ORAM solutions were demonstrated through simulated and prototyped hardware evaluations.

Pros and Cons

I believe that the paper has the following positive features:

- In the introduction, the authors presented a very clear objective of mitigating ORAM's performance penalties while further reducing information leak, without going into excessive details in explaining past efforts.
- The addition of in-language sensitivity annotations and the specialised deterministic instruction set allowed a formal proof of security to be achieved without significantly compromising the generalisability of the solution.
- In performance evaluations, the authors covered all possible levels of specialisation (with/without presences of ERAM and scratchpad) to present an in-depth analysis of variations in performance.

I believe that the paper has the following negative features:

- The problem the authors were trying to solve was very narrow: most of organisations with sufficient resources to rewrite their codebase for secure remote execution would have sufficient resources to run computations in-house.
- The uses of specialised instruction set and ORAM integration depart radically from conventional computing hardware, making a high performance silicon implementation of the authors' solution unlikely.
- The solution's incomplete hardware prototype and significantly constrained programming requirements left behind significant future work to apply the architecture on real high-performance computing tasks.

The Problem/Motivation

The concept of "cloud computation" requires users of remote computing infrastructures to implicitly place trust on the physical security of the infrastructures, as attackers with physical access to the system can conduct direct and side-channel attacks. Prior solutions to physical vulnerabilities mostly placed computations entirely within large ORAM banks, which imposed severe performance penalties to achieve oblivious memory events, and could still leak some side-channel information. Previous point solutions such as Blanton et al. [2] optimised input data to achieve faster ORAM execution, and did not consider the program context. At the cost of some code generality (such as annotating variables and imposing a more constrained semantics), the program context can be exploited to produce a faster ERAM-ORAM hybrid solution without compromising security.

The Solution/Approach

GhostRider seeks to keep computing operations outside a secure processor confidential from an attacker with physical access, which was achieved over a multitude of components. The specialised hardware contains three types of memory banks: the fastest and insecure RAM, the encrypted but side-channel-vulnerable ERAM, and the slowest, high-security ORAM. The choice of data placement between these memory banks is determined by a specialised compiler drawing information from sensitivity annotations of variables in a C-style language, albeit with additional control-flow constraints. A specialised compiler translates remote computing code written in this language into a RISC-style instruction set cooperating with deterministic-length instructions and cache-replacing scratchpads to prevent side-channel information leak. The compiler also applies type-checking from proved-secure semantics to ensure that low equivalence is maintained over memory trace patterns resulting from all instructions, branching and sub-typing, thus indistinguishable to a physical attacker. Based on existing theoretical frameworks and implementations around ORAM, this solution extends to achieve full data confidentiality.

Evaluation

Owing to the lack of specialised hardware fabrication, this solution was only prototyped on an FPGA-based ARM-like CPU and an external ORAM controller, with very limited computing capabilities. Therefore the solution was additionally tested on conventional computing devices with a simulator. In both set of evaluations, the authors compared performance of insecure solutions with the baseline of prior research (pure-ORAM), as well as with their ERAM/ORAM hybrid solution with or without cache-replacing hardware scratchpads. Speed-ups between 1.05X and 9.03X were achieved in comparison with the pure-ORAM solution, although both were significantly slower than the insecure RAM solution as expected. Discrepancies between solutions and test cases, as well as between the prototype and the simulation were explored and reasoned by the authors. Some performance deficiencies were suggested as future work.

Your Opinion

Data placement segmentation based on differences in possible side-channel information leak was a very novel idea by the authors, and could be applied in a wide range of other performance-security trade-off scenarios. I believe that constructing a working prototype to this effect was the most valuable contribution of this paper.

On the other hand, the solution itself was likely very impractical for further development, due to several drawbacks fundamental to the solution: rewriting existing code is always a hard-to-ignore business cost, even if just for the batch data processing modules – not to mention companies and government organisations who are willing to bear this cost for security are likely to have substantial in-house computing power regardless. The industry is also clearly more interested in building highly-parallel CPU and GPU-based computing systems with conventional hardware rather than paying substantial cost to design and fabricate specialised silicons that cannot be easily repurposed. Without a scalable prototype, the authors made several assumptions in their solution's scalability (especially related to ORAM), which may not be realisable in practice.

Questions for the Authors

- Do you see a realistic case for commercial and governmental organisations to invest in implementing your solution on purposely-fabricated silicon due to its security benefits, or would in-house infrastructures be significantly more preferable for them to use?
- With the emergence of Rowhammer-style attacks on memory modules [3], do you foresee any vulnerabilities or immunities in your solution against possible future attacks of similar types?

References

[1] C. Liu, A. Harris, M. Maas, M. Hicks, M. Tiwari, and E. Shi, "Ghostrider: A hardware-software system for memory trace oblivious computation," *ACM SIGPLAN Notices*, vol. 50, no. 4, pp. 87–101, 2015.

[2] M. Blanton, A. Steele, and M. Alisagari, “Data-oblivious graph algorithms for secure computation and outsourcing,” in *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security.* ACM, 2013, pp. 207–218.

[3] K. Razavi, B. Gras, E. Bosman, B. Preneel, C. Giuffrida, and H. Bos, “Flip feng shui: Hammering a needle in the software stack.” in *USENIX Security Symposium*, 2016, pp. 1–18.