GhostRider: A Hardware-Software System for Memory Trace Oblivious Computation

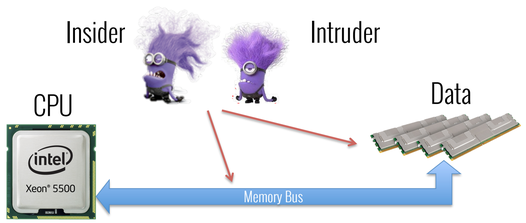
Chang Liu, Austin Harris, Martin Maas, Michael Hicks, Mohit Tiwari, Elaine Shi

Highly Sensitive Data in the Cloud

- Organizations (including the U.S. government [1]) are offloading workloads into the cloud.
- Many operate on highly sensitive data, whether for privacy, regulatory or competitive reasons.
- Need strong confidentiality for this data.

The Security Challenge

- Cloud operators have physical access to machines.
- Can eavesdrop on data traveling between CPU and main memory (e.g. using malicious NVDIMMs [2]).



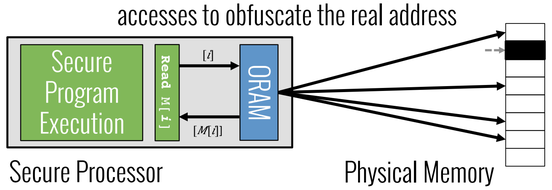
This threat is real. Attacks can originate from:

- Malicious employees (e.g. infiltrating cloud providers or bribing existing employees)
- Intruders that break into a data center to wiretap machines (it has been done: [3])

Secure Processors

- Secure processors (e.g., [4,5]) protect against physical access through a tamper-proof processor that encrypts all data in memory and provides remote attestation.
- Still leaks the memory addresses that being accessed; Oblivious RAM (ORAM) can be used to hide them.

ORAM requests turn into multiple DRAM accesses to obfuscate the real address

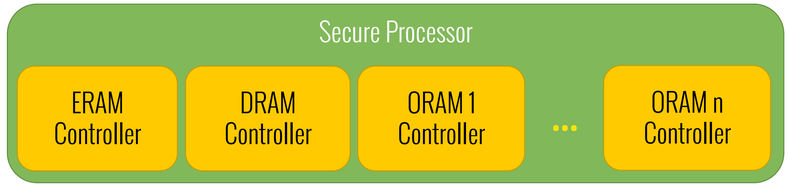


Challenges in current Oblivious Processors

- ORAM is slow relative to a non-secure baseline: every single memory access incurs polylog(N) additional accesses for an ORAM of size N.
- Overheads can be 100-250x of data movement per access. Hardware optimizations exist (e.g., treetop caching, PLB), but most overhead is inherent to the ORAM scheme.

Hybrid Memory Model

- Our approach: Achieve large speed-up with Hybrid Memory Model that allows to use DRAM, encrypted DRAM (ERAM) and smaller ORAMs when security is not sacrificed.



Memory Trace Obliviousness (MTO)

- What does "security is not sacrificed" mean? ORAM fully hides the addresses that are being accessed, but when accessing different memories/ORAM banks/etc., an attacker can see which memory is being accessed; this leaks information as well.
- Formal Property: Memory Trace Obliviousness (MTO), proposed in [5].

> MTO: **Memory trace** does not depend on **secret input**

Example 1: The next instruction fetched and the next write leak the value of secret input s.

Program	Input	Trace
if(s) x=1 else y=1	x, y are in ERAM	s=0	**read(s)** **instruction fetch(x=1)** **write(x)**

Not MTO

Example 2: By putting code and x, y into ORAMs, the trace does not depend on s anymore.

Program	Input	Trace
if(s) x=1 else y=1	x, y are in ORAM o Code is in ORAM i	s=1	**read(s)** **instruction fetch(i)** **write(o)**

MTO

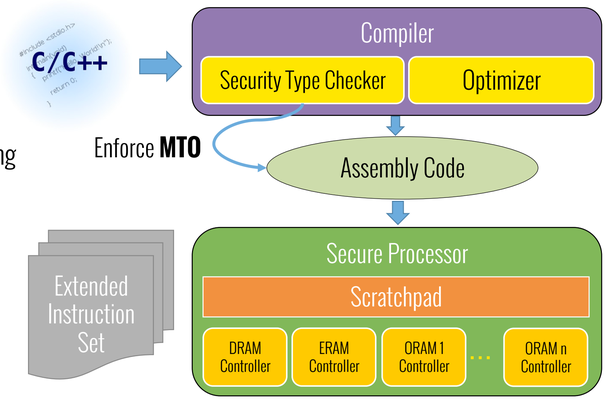
How to generate code that satisfies MTO? Let the compiler help you!

GhostRider: Co-Designing Hardware and Compiler

GhostRider: A Hardware-software co-designed system comprised of:

1. A compiler to produce provably MTO-compliant code
2. A secure processor for executing this code (prototyped on an FPGA-platform)

The compiler produces code for an extended instruction set that supports an explicitly managed scratchpad and allows moving data between it and memories.

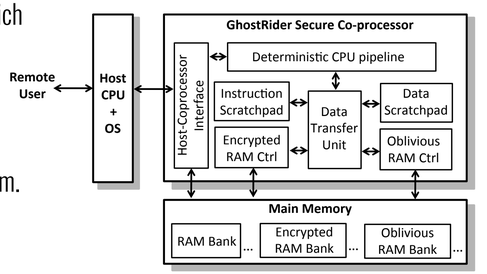


The GhostRider Compiler

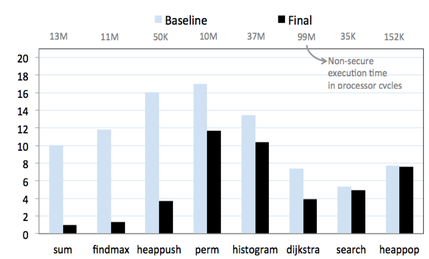
- Implements a Type System for the assembly-style target code, keeps track of security labels and symbolic values for registers and blocks and computes trace patterns. (more details in the paper)

GhostRider Architecture & FPGA Prototype

- Based on Phantom Secure Processor [3], which is based on the RISC-V Rocket CPU.
- Supports extension of the RISC-V ISA [6] for accessing different DRAM/ERAM/ORAM banks and performing DMAs between them.
- Implemented on Convey HC-2ex FPGA platform.



Evaluation of the FPGA Prorotype



Evaluated GhostRider on FPGA and in simulation:

- See paper for more details and simulations.
- Achieve substantial speed-up over non-secure baseline for some workloads, no slow-downs.
- Speed-up depends on the program and its memory access patterns.

References:

[1] "Microsoft reveals plans for a government cloud platform," ZDNet, 10/8/13; [2] "Case Study: Physical Penetration, Hosting Provider," SecureState, www.securestate.com/Services/Profiling/Pages/Physical-Attack-and-Penetration.aspx; [3] Maas, et al. "Phantom: Practical oblivious computation in a secure processor." In Proc. of CCS 2013. [4] Ren, et al. "Design Space Exploration and Optimization of Path Oblivious RAM in Secure Processors" in Proc. of ISCA 2013; [5] Liu, et al. "Memory Trace Oblivious Program Execution", In Proc. of CSF 2013; [5] The RISC-V Instruction Set Architecture, www.riscv.org