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]).
- 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.
- 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.

HYBRID MEMORY MODEL

- 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].

MEMORY TRACE OBLIVIOUSNESS (MTO)

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

```
Program: if(x>1) x, y are in ERAM
          else y=0
```

Input: s=0

Trace:
```
read(s) instruction fetch(x=1) write(y)
```

Net MTO: Memory trace does not depend on secret input s.

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

```
Program: if(x>1) x, y are in ORAM
          else y=0
          Code is in DRAM
```

Input: s=1

Trace:
```
read(s) instruction fetch(x=1) write(y)
```

MTO: Memory trace does not depend on secret input s.

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 DIMMs between them.
- Implemented on Canvay HC-2x FPGA platform.

EVALUATION OF THE FPGA PROTOTYPE

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.