DAGguise
Mitigating Memory Controller Side Channels

Peter W. Deutsch*, Yuheng Yang*, Thomas Bourgeat,
Jules Drean, Joel Emer, and Mengjia Yan

ASPLOS 2022 (Session 3A)
Microarchitectural Side-Channels

Key Defense Tradeoff: Security vs. Performance
DAGguise Key Idea

Directed Acyclic Request Graph (rDAG)

Victim

CPU 0

Memory Shaper

Memory Controller

CPU 1

Attacker

DAGguise achieves:
✓ Formally-Verified Security
✓ Good Performance
Outline

• Memory Controller + Scheduler-based Side Channels
• Existing Approaches
  • Static Partitioning
  • Traffic Shaping
• DAGguise
  • Directed Acyclic Request Graphs (rDAGs)
• Security + Performance Evaluation
• Generalizability
Memory Controller Side Channels

This is a class of “scheduler-based” side channels!
Scheduler-Based Side Channels

Lord of the Rings: Side Channel Attacks on the CPU On-Chip Ring Interconnect Are Practical

Riccardo Pacchietti, Li-Eung Loo, Christopher W. Fletcher
University of Illinois at Urbana-Champaign

We introduce the little attack that David can install. There are two challenges that the attacker needs to overcome: (i) the attacker needs to identify a suitable victim and (ii) the attacker needs to establish a channel to communicate with the victim. We demonstrate our attack on the ARM920T processor and show that the attack is practical.

Bandwidth Utilization Side-Channel on ML Inference Accelerators

Sarabjot Banerjee, Shijia Wei, Prakash Ramachandran, Mohit Tiwari
The University of Texas at Austin
sarabjot@cs.utexas.edu, shijia.wei@utexas.edu, prakashr@cs.utexas.edu, mohitt@cs.utexas.edu

We present a novel side-channel attack on machine learning accelerators. Our attack leverages the fact that machine learning models are vulnerable to side-channel attacks due to their highly parallel and data-dependent nature. We show that by monitoring the clock frequency and power consumption of the accelerator, we can infer the execution of specific instructions or data.

Port Contention for Fun and Profit

Abinav Gokul, Shunyi Fan, Yitao Guo, Wei Hu, Jinyu Chen, Xiangyu Zhou
The University of Texas at Austin
abinavgokul@utexas.edu, shunyifan@utexas.edu, yitao.guo@utexas.edu, weihu@utexas.edu, jinyuchen@utexas.edu, xiangyu@utexas.edu

We investigate the performance of machine learning accelerators on a real-world dataset. We find that machine learning models are vulnerable to side-channel attacks due to their highly parallel and data-dependent nature. Our attack leverages the fact that machine learning models are vulnerable to side-channel attacks due to their highly parallel and data-dependent nature. We show that by monitoring the clock frequency and power consumption of the accelerator, we can infer the execution of specific instructions or data.

Rivalry: Exploiting DRAM Addressing for Cross-CPU Attacks

Peter Pudjianto, Daniel Gruss, Clementine Maurice, Michael Schwarz, Stefan Mangard
CISPA, University of Technology, Aachen

We demonstrate our attack on the ARM920T processor and show that the attack is practical.
Timing Attack Example

The attacker uses its own latencies to leak information!
Static Partitioning in Time

Use a Round Robin, No-Skip Arbitration Policy

Avoiding Information Leakage in the Memory Controller with Fixed Service Policies (Shafiee et al., Utah, ISCA 2015)
Traffic Shaping

**Shaping Strategy:** Delay victim’s existing requests and add fake requests

How do we do this for real applications without significant costs?
Camouflage’s Traffic Shaping Strategy

Shape memory requests to a secret-independent timing distribution

- **Good Performance**: Dynamic sharing of the memory controller
  - Ordering or bank information can reveal the secret

- **Expensive Profiling**: Ideal shaping distribution depends on co-running applications

- **Insecure**: Re-profiling required
  - Increased background bandwidth usage

---

DAGguaise’s Traffic Shaping Strategy

Shape memory requests to a secret-independent

Directed Acyclic Request Graph (rDAG)

✓ Secure
✓ Good Performance
✓ Profile Victim Alone
Directed Acyclic Request Graphs

**Vertices**
Memory requests with *variable* latency

**Edges**
Dependencies between memory requests with *fixed* latency

- $V_0$
- $V_1$
- $V_2$
- $V_3$
- $V_4$

- $W_{01}$
- $W_{02}$
- $W_{03}$
- $W_{34}$

*Memory Controller*
Why shape requests to an rDAG?

✓ Security
  • Shaping to a secret-independent defense rDAG makes victim request patterns *indistinguishable*
  • Defense rDAGs are public and are the only thing an attacker can recover

✓ Performance
  • Allows for *dynamic* sharing of memory resources in the memory controller

✓ Profiling Cost
  • Does not require knowledge of co-located applications
The shaper output is always the same, **no matter the secret!**
The attacker’s observations should be *independent* from victim’s request pattern
Indistinguishability Property

- Attacker’s observation is independent from victim’s request pattern
  - Given an attacker’s request pattern, the attacker has an identical observation when contending with ANY victim’s request pattern
  - This holds for ANY attacker’s request pattern

<table>
<thead>
<tr>
<th>Victim Request Patterns</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>... ...</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Attacker Request Patterns</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>Attacker’s Response Pattern X</td>
</tr>
</tbody>
</table>
Formalization & Verification

• Formalize the indistinguishability property using state transitions

\[ P(S_0, n) := \forall \text{ Req}_{T_x}, \text{ Req}'_{T_x}, \forall \text{ Req}_{R_x} \]
\[ \text{if } S_0 \xrightarrow{\text{Resp}_{T_x}, \text{Resp}_{R_x}} S'_{n} \text{ and } S_0 \xrightarrow{\text{Resp}'_{T_x}, \text{Resp}'_{R_x}} S'_n \]
\[ \text{then } \text{Resp}_{R_x} = \text{Resp}'_{R_x} \]

• Verification with Rosette:
  • First k cycles: symbolic execution
  • Arbitrary cycles: k-induction
rDAG Adaptivity

(a) Victim’s Request Patterns

(b) Unprotected Program’s Request Patterns

(c) Contention between Victim and Unprotected Program on Memory Controller

rDAG’s adaptivity allows for better bandwidth utilization!
Offline Profiling Step

• Not for security, *any* secret-independent rDAG ensures security
• Low profiling cost
  • Victim is profiled *alone*
  • Reduce search space by finding parameters for an rDAG *template*
Experimental Setup

• **Simulator:** gem5 and DRAMSim2

• **Architectural Specifications:**
  • 2 and 8 out-of-order CPU cores
  • 32KB L1i/d, 256kB L2, 1MB/core L3

• **Evaluated Configurations:**
  • DAGguise
  • Fixed Service (Bank Triple Alternation)
  • Baseline

• **Evaluated Applications:**
  • Unprotected SPEC benchmark(s) co-running alongside DAGguise protected application(s)
Experimental Results

DAGguise’s improves performance for both protected and unprotected applications!

DAGguise achieves a 12% performance improvement over Fixed Service in an 8-CPU system
DAGguise Generalization

SMT Contention

Victim Thread → Decode Pipeline → Attacker Thread

μ-ops

Scheduler

Port 0 Port 1 Port 5 Port 7

Resource Contention

Network on Chip Contention

Attacker Accesses → Router → Victim Accesses

Resource Contention
More in the Paper

• Implementation details of DAGguise shaper
• Formal security verification using symbolic execution and k-induction
• Detailed rDAG offline profiling process
• More performance and area overhead evaluation
• Generalizations to other scheduler-based side channels (e.g. port contention)
Conclusion

• DAGguise
  • A memory traffic shaper which:
    • Completely eliminates data leakage
    • Allows for dynamic contention
    • Requires only simple profiling

• rDAGs
  • A general and adaptive request representation

• A formal model of correctness using Rosette

• A generalized scheduler-based attack mitigation framework
DAGguise
Mitigating Memory Controller Side Channels

Peter W. Deutsch
pwd@mit.edu

Yuheng Yang
yuhengy@mit.edu

Thomas Bourgeat
bthom@mit.edu

Jules Drean
dream@mit.edu

Joel S. Emer
jsemer@mit.edu

Mengjia Yan
mengjiay@mit.edu