MicroScope: Enabling Microarchitectural Replay Attacks

Dimitrios Skarlatos, Mengjia Yan, Bhargava Gopireddy, Read Sprabery, Josep Torrellas, and Christopher W. Fletcher
University of Illinois at Urbana-Champaign
Overview
The Era of Side-Channels
How much can leak over side channels?

<table>
<thead>
<tr>
<th>Victim:</th>
<th>Attacker:</th>
</tr>
</thead>
<tbody>
<tr>
<td>if (secret)</td>
<td>for ..</td>
</tr>
<tr>
<td>use resource</td>
<td>t1 = time()</td>
</tr>
<tr>
<td>else</td>
<td>use resource</td>
</tr>
<tr>
<td>don’t use resource</td>
<td>t2 = time()</td>
</tr>
</tbody>
</table>

- Need repeated measurements to be confident → Denoise
- However, many applications run only once → Attacker gets 1 measurement
- Can attackers really extract secrets?
Contribution: Microarchitectural Replay Attacks

- Attacker leverages speculative execution
  - To repeatedly replay a snippet of victim code
  - That runs only once

Victim:

```
l d addr // "replay handle"
...  
l d secret // secret the attacker tries to leak
```
Contribution: Microarchitectural Replay Attacks
Contribution: Microarchitectural Replay Attacks

1d addr: Issue Replay Handle
Long Latency Event

1d secret: Speculative Execution of Secret

SPECTRE

ILLINOIS
Contribution: Microarchitectural Replay Attacks

ld addr: Issue Replay Handle | Long Latency Event | Squash Event | Clear State

ld secret: Speculative Execution of Secret | Squash
Contribution: Microarchitectural Replay Attacks

ld addr:

Issue Replay Handle

Long Latency Event

Squash

Event

Clear State

ld secret:

Speculative Execution of Secret

Squash

Cause Shared Resource Contention & Monitor

Replay!!
Background
Page Tables Background

- Page tables stored in memory
- On a TLB Miss $\rightarrow$ “page walk” = memory accesses
  - Each step of page walk = cache hit/miss.
  - Page walk cache (PWC): hardware cache of translations
- If Present bit in pte_t is cleared $\rightarrow$ Page Fault, invoke OS
Trusted Computing with SGX

- Designed to run sensitive applications in the cloud
- Do not trust OS/Hypervisor
- OS/Hypervisor cannot introspect/tamper enclave
- Unfortunately, OS/Hypervisor still manages demand paging

```
Attack Surface With Enclaves

App  App  App
Operating System
Hypervisor
Hardware
```

Illinois
Threat Model
Threat Model

In SGX the OS is responsible
- Demand paging

Attacker (OS) can:
- Evict TLB entries
- Evict page walk cache entries
- Monitor side channels
MicroScope:
A framework to exploit microarchitectural replay attacks
Attack Examples

**Victim Code**

1. //public address
2. handle(pub_addr);
3. ...
4. transmit(secret);
5. ...

**Loop Victim Code:**

1. for i in ...
2. handle(pub_addrA);
3. ...
4. transmit(secret[i]);
5. ...
6. memOp(pub_addrB);
7. ...
Terminology

Victim Code
1. //public address
2. handle(pub_addr);
3. ...
4. transmit(secret);
5. ...

Replay handle:
- Load to a public address (known to OS)

Transmitter:
- Any instruction(s) whose execution reveals secret through some side channel
- Occurs <$ ROB length from Replay Handle
Timeline of a MicroScope Attack - Setup

- **Attack Setup**
- **Time**

- **Attacker**
- **Victim**
Timeline of a MicroScope Attack - Setup

- Clear PTE
- Present Bit of Replay Handle
- Attack Setup

Attacker  Victim
Timeline of a MicroScope Attack - Setup

- Attack Setup
- Clear PTE Present Bit of Replay Handle
- Flush Replay Handle Page Table Entries
Timeline of a MicroScope Attack - Setup
Timeline of a MicroScope Attack

```
handle(pub_addr):

  Attack Setup

  Issue Replay

  Handle
```

Attacker ✋ Victim
Timeline of a MicroScope Attack

handle(pub_addr): [Issue Replay, Handle, L1 TLB Miss]
Timeline of a MicroScope Attack

- **Attack Setup**
  - `handle(pub_addr)`: Issue Replay Handle, L1 TLB Miss, L2 TLB Miss
  - `transmit(secret)`: Speculative Execution of Transmitter
Timeline of a MicroScope Attack

handle(pub_addr):
  Issue Replay Handle
  L1 TLB Miss
  L2 TLB Miss
  PWC Miss

transmit(secret):
  Speculative Execution of Transmitter
Timeline of a MicroScope Attack

Timeline:

1. **Attack Setup**
2. `handle(pub_addr):`
   - Issue Replay Handle
   - L1 TLB Miss
   - L2 TLB Miss
   - PWC Miss
   - PGD Walk
   - PUD Walk
   - PMD Walk
   - PTE Walk

3. `transmit(secret):`
   - Speculative Execution of Transmitter

Roles:
- **Attacker**
- **Victim**
Timeline of a MicroScope Attack

<table>
<thead>
<tr>
<th>Attack Setup</th>
<th>Tune speculative execution duration with: Cache Hit or Miss</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle(pub_addr):</td>
<td>Issue Replay Handle</td>
</tr>
<tr>
<td>transmit(secret):</td>
<td>Speculative Execution of Transmitter</td>
</tr>
</tbody>
</table>
Timeline of a MicroScope Attack

handle(pub_addr):

transmit(secret):

Speculative Execution of Transmitter

Attack Setup

Issue Replay Handle

L1 TLB Miss

L2 TLB Miss

PWC Miss

PGD Walk

PUD Walk

PMD Walk

PTE Walk

Page Fault

Time

Attacker

Victim
Timeline of a MicroScope Attack

```
handle(pub_addr):  
    | Issue Replay | L1 TLB | L2 TLB | PWC  | PGD  | PUD  | PMD  | PTE | Page  |
    | Handle       | Miss   | Miss   | Miss | Walk | Walk | Walk | Walk | Fault |
transmit(secret):  
    | Speculative Execution of Transmitter | Squash |
```

Attacker  
Victim
Timeline of a MicroScope Attack

handle(pub_addr):

transmit(secret):

Speculative Execution of Transmitter

Squash
Timeline of a MicroScope Attack

handle(pub_addr):
Issue Replay Handle L1 TLB Miss L2 TLB Miss PWC Miss PGD Walk PUD Walk PMD Walk PTE Walk Page Fault

transmit(secret):
Speculative Execution of Transmitter Squash

Attacker
Victim
Timeline of a MicroScope Attack

handle(pub_addr):

transmit(secret):

Speculative Execution of Transmitter

Squash

Attacker  Victim
Timeline of a MicroScope Attack

handle(pub_addr):

transmit(secret):

Speculative Execution of Transmitter

Squash

Issue Replay
Handle

L1 TLB Miss
L2 TLB Miss
PWC Miss
PGD Walk
PUD Walk
PMD Walk
PTE Walk
Page Fault

OS Invocation

Replay!!
Timeline of a MicroScope Attack

handle(pub_addr): Issue Replay Handle
   L1 TLB Miss | L2 TLB Miss | PWC Miss | PGD Walk | PUD Walk | PMD Walk | PTE Walk | Page Fault | OS Invocation
transmit(secret): Speculative Execution of Transmitter
   Squash
Timeline of a MicroScope Attack

`handle(pub_addr):` Issue Replay Handle, L1 TLB Miss, L2 TLB Miss, PWC Miss, PGD Walk, PUD Walk, PMD Walk, PTE Walk, Page Fault, OS Invocation

`transmit(secret):` Speculative Execution of Transmitter, Squash
Timeline of a MicroScope Attack

handle(pub_addr):

transmit(secret):

Speculative Execution of Transmitter

Squash

Cause Shared Resource Contention & Monitor

OS Invocation

Replay!!

ILLINOIS

Attack Setup

Attacker
Victim
Attacker Monitor/Contention thread
Loop Example
Loop Example

Victim:

1. for i in ...
2. handle(pub_addrA);
3. ...
4. transmit(secret[i]);
5. ...
6. memOp(pub_addrB);
7. ...

• Goal:
  • Leak secret in every iteration

• Challenge:
  • Monitor window of a replay handle < ROB length
Loop Example

Victim:

1. for i in ...
2. handle(pub_addrA);
3. ...
4. transmit(secret[i]);
5. ...
6. pivot(pub_addrB);
7. ...

Instruction used to pivot around transmit instructions

- Replay Handle
- Transmit Code
- Pivot Instruction
In a Loop

Static Code
1. for i in ...
2. handle(pub_addrA);
3. ...
4. transmit(secret[i]);
5. ...
6. pivot(pub_addrB);
7. ...

Dynamic code
1. handle(pub_addrA);
2. ...
3. transmit(secret[0]);
4. ...
5. pivot(pub_addrB);
6. ...
7. handle(pub_addrA);
8. ...
9. transmit(secret[1]);
10....
11. pivot(pub_addrB);

Monitor window
- Replay Handle
- Transmit Code
- Pivot Instruction
In a Loop

Static Code

1. for i in ...
2. handle(pub_addrA);
3. ...
4. transmit(secret[i]);
5. ...
6. pivot(pub_addrB);
7. ...

Dynamic code

1. handle(pub_addrA);
2. ...
3. transmit(secret[0]);
4. ...
5. pivot(pub_addrB);
6. ...
7. handle(pub_addrA);
8. ...
9. transmit(secret[1]);
10. ...
11. pivot(pub_addrB);

Pivot page fault

Replay Handle  Transmit Code  Pivot Instruction
In a Loop

**Static Code**

1. for i in ...
2. handle(pub_addrA);
3. ...
4. transmit(secret[i]);
5. ...
6. pivot(pub_addrB);
7. ...

**Dynamic code**

1. handle(pub_addrA);
2. ...
3. transmit(secret[0]);
4. ...
5. pivot(pub_addrB);
6. ...
7. handle(pub_addrA);
8. ...
9. transmit(secret[1]);
10. ...
11. pivot(pub_addrB);

Retired instructions

Cause page fault on pivot
**In a Loop**

### Static Code
1. for i in ...
2.   handle(pub_addrA);
3.   ...
4.   transmit(secret[i]);
5.   ...
6.   pivot(pub_addrB);
7.   ...

### Dynamic Code
1. handle(pub_addrA);
2.   ...
3.   transmit(secret[0]);
4.   ...
5.   pivot(pub_addrB);
6.   ...
7.   handle(pub_addrA);
8.   ...
9.   transmit(secret[1]);
10.  ...
11.  pivot(pub_addrB);

Retired instructions

Cause page fault in new replay handle
In a Loop

Static Code

1. for i in ...
2.   handle(pub_addrA);
3.   ...
4.   transmit(secret[i]);
5.   ...
6.   pivot(pub_addrB);
7.   ...

Use this idea to denoise side channels on AES (see paper)

Dynamic code

1. handle(pub_addrA);
2. ...
3. transmit(secret[0]);
4. ...
5. pivot(pub_addrB);
6. ...
7. handle(pub_addrA);
8. ...
9. transmit(secret[1]);
10. ...
11. pivot(pub_addrB);
Port Contention Attack
Port Contention Attack with MicroScope

Victim:
1. handle(pub_addrA);
2. if (secret)
3.   victim_mul()
4. else
5.   victim_div() \{ \text{No Loop!} \} 

Attacker:
1. Setup replay attack 
2. ... 
3. for ...
4.   t1 = time()
5.   attacker_div()
6.   t2 = time()
7. }
Port Contention Attack with MicroScope

Victim:
1. handle(pub_addrA);
2. if (secret)
3.   victim_mul()
4. else
5.   victim_div()

Attacker:
1. Setup replay attack
2. ...
3. for ...
4.   t1 = time()
5.   attacker_div()
6.   t2 =time()
7. }

Replay Handle  Transmit Code  Attacker Code
Port Contention Attack with MicroScope

Victim:
1. `handle(pub_addrA);`
2. `if (secret)`
3. `victim_mul()`
4. `else`
5. `victim_div()`

Attacker:
1. Setup replay attack
2. ...
3. for ...
4. `t1 = time()`
5. `attacker_div()`
6. `t2 = time()`
7. `}`
Port Contention Attack with MicroScope

Victim:

1. handle(pub_addrA);
2. if (secret)
3.   victim_mul()
4. else
5.   victim_div()

Attacker:

1. Setup replay attack
2. ...
3. for ...
4.   t1 = time()
5.   attacker_div()
6.   t2 = time()
7. }

10,000 → 1
Port Contention Results

- Victim and attacker run on adjacent SMT contexts of same core
- Attacker performs divisions in a loop and measures time
- Victim performs two multiplications without a loop

- Victim performs two divisions without a loop
Generalizing Microarchitectural Replay Attacks
Microarchitectural Replay Attacks

This work:
1. Replay Handle $\rightarrow$ Page fault-inducing load
2. Replayed Code $\rightarrow$ Leaky instruction
3. Side Channel $\rightarrow$ uarch structures
4. Attacker strategy $\rightarrow$ Page fault until denoise

Changing each can result in different attacks!!
More On the Paper

• Other attack examples
• MicroScope framework implementation
• Attacks on AES
• Countermeasures discussion
Open source!

- MicroScope Framework
  - Denoise nearly arbitrary microarchitectural side channels using page faults
  - Only a *single* run of the victim
  - Demonstrate attacks on notoriously noisy side channels
    - Detect the presence or absence of two divide instructions
    - Single-step and denoise cache-based attacks on AES

https://github.com/dskarlatos/MicroScope
Microarchitectural Replay Attacks

⇒ New class of attacks

• Opens large new attack surface (noisy side channels)
• Implications for integrity, TSX, physical side channels

• Exploits core microarchitectural feature
  • Dynamic instructions can be replayed through controlled squashes!

• We need new security properties to mitigate these attacks!