MicroScope: Enabling Microarchitectural Replay Attacks

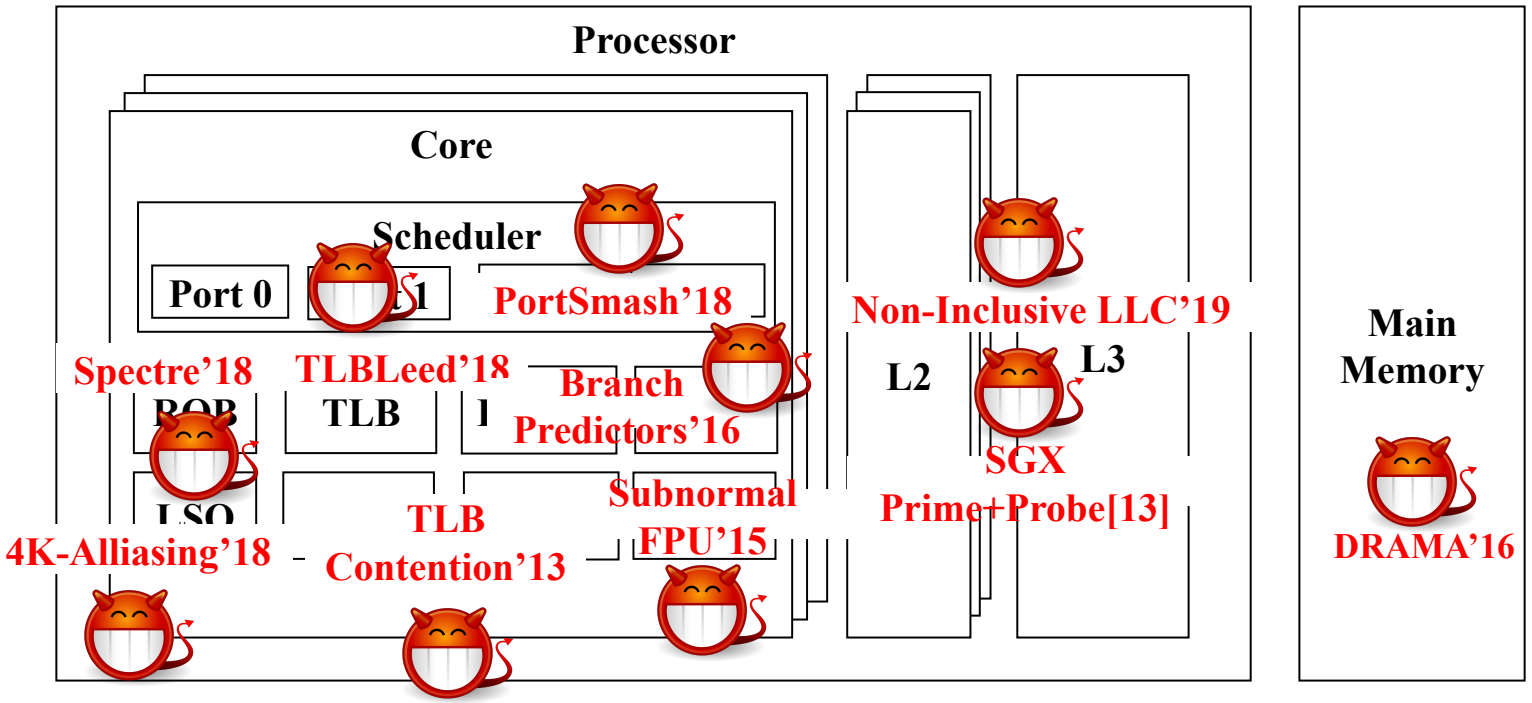
Dimitrios Skarlatos, Mengjia Yan, Bhargava Gopireddy, Read Sprabery, Josep Torrellas, and Christopher W. Fletcher
University of Illinois at Urbana-Champaign

ILLINOIS

Overview

ILLINOIS

The Era of Side-Channels



ILLINOIS

How much can leak over side channels?

Victim:

```
if (secret)
  use resource
else
  don't use resource
```

Attacker:

```
for ..
  t1 = time()
     use resource
  t2 = time()
```

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

ILLINOIS

Contribution: Microarchitectural Replay Attacks

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

} Primitive to denoise arbitrary side channels

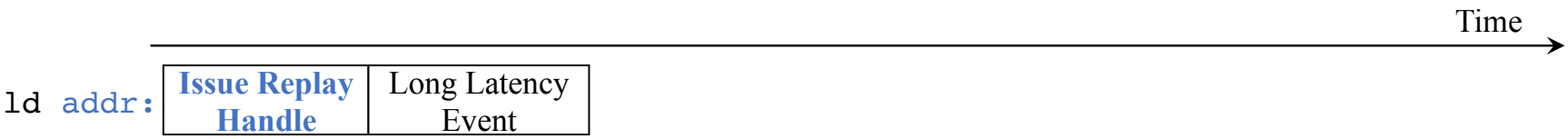
Victim:

```
ld addr // "replay handle"
…
ld secret // secret the attacker tries to leak
```

{ Memory operation that will cause a squash and re-execute }

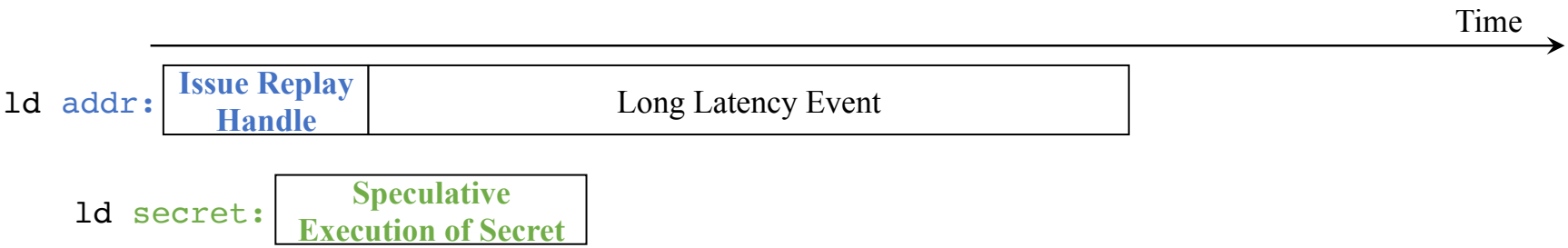
ILLINOIS

Contribution: Microarchitectural Replay Attacks



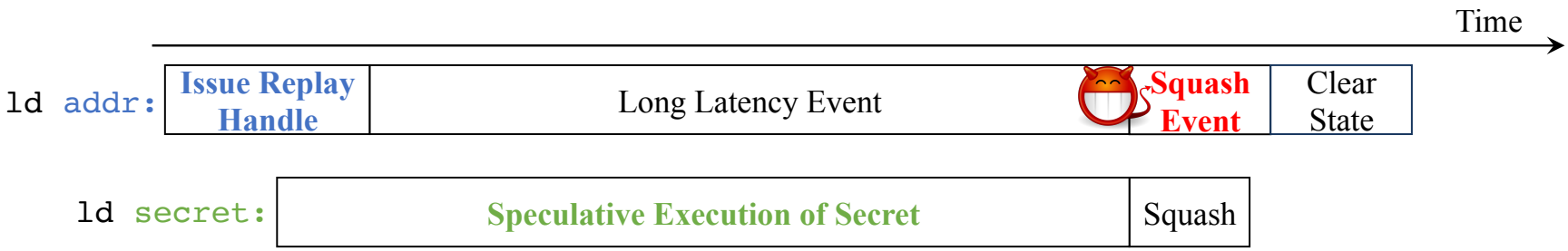
ILLINOIS

Contribution: Microarchitectural Replay Attacks



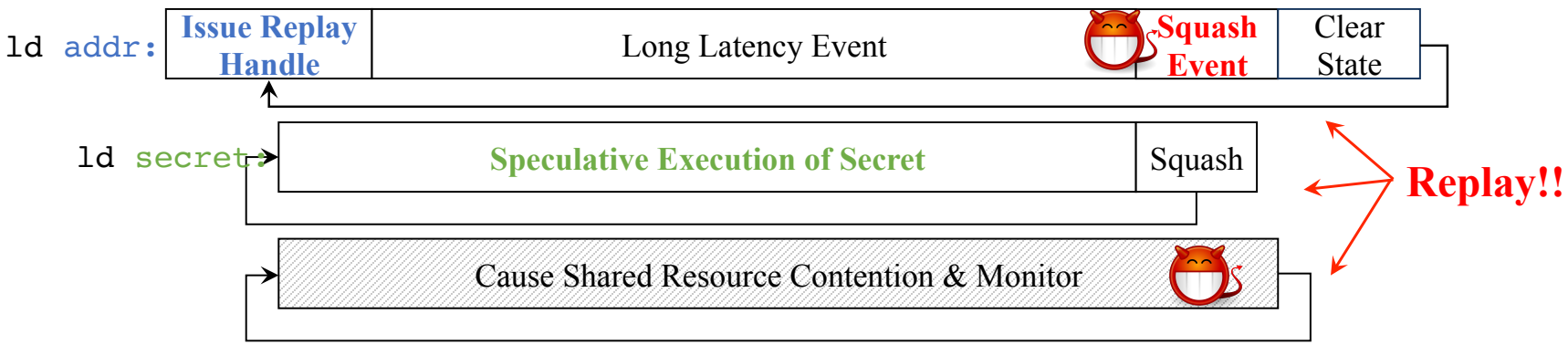
ILLINOIS

Contribution: Microarchitectural Replay Attacks



ILLINOIS

Contribution: Microarchitectural Replay Attacks

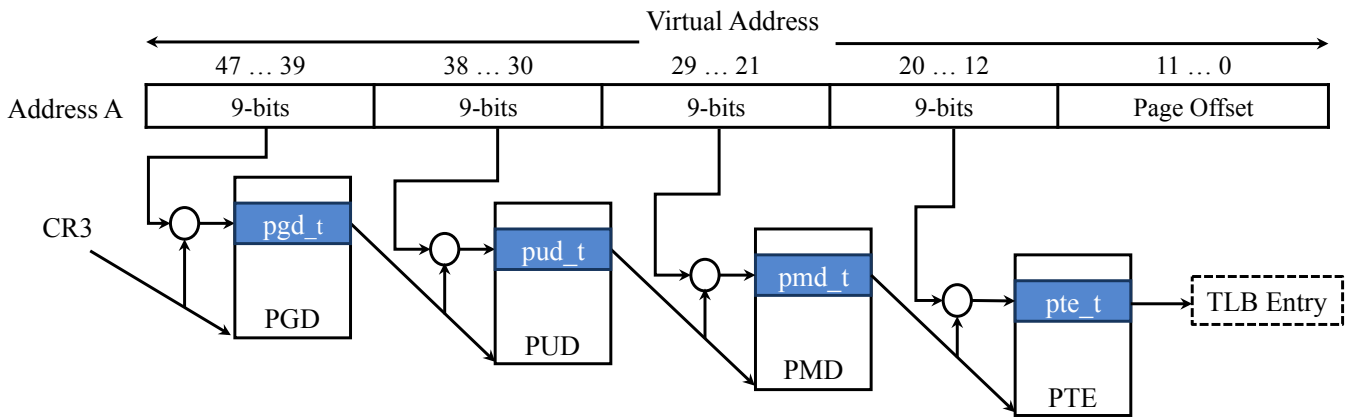


ILLINOIS

Background

ILLINOIS

Page Tables Background



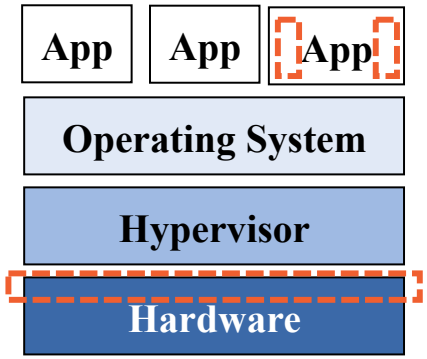
- Page tables stored in memory
- On a TLB Miss → "*page walk*" = memory accesses
  - Each step of page walk = cache hit/miss.
  - Page walk cache (PWC): hardware cache of translations
- If Present bit in pte_t is cleared → Page Fault, invoke OS

ILLINOIS

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



Attack Surface

ILLINOIS

Threat Model

ILLINOIS

Threat Model

In SGX the OS is responsible

- Demand paging

Attacker (OS) can:

- Evict TLB entries
- Evict page walk cache entries
- Monitor side channels

ILLINOIS

MicroScope:

A framework to exploit microarchitectural replay attacks

ILLINOIS

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.   ...
```

ILLINOIS

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

ILLINOIS

Timeline of a MicroScope Attack - Setup

Attacker Victim

ILLINOIS

Timeline of a MicroScope Attack - Setup

Attacker Victim

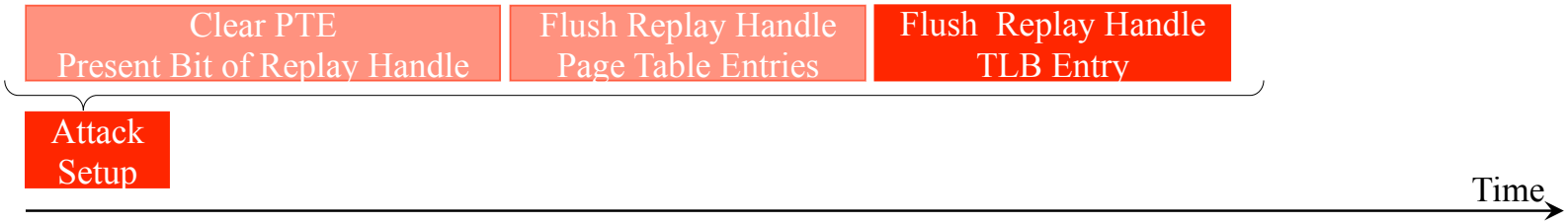
ILLINOIS

Timeline of a MicroScope Attack - Setup

Attacker Victim

ILLINOIS

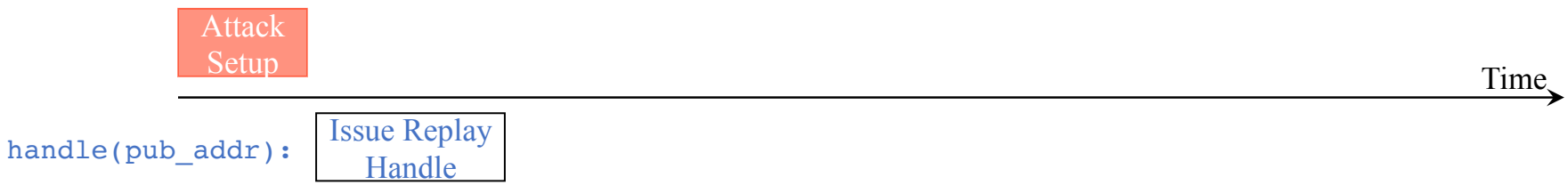
Timeline of a MicroScope Attack - Setup



Attacker Victim

ILLINOIS

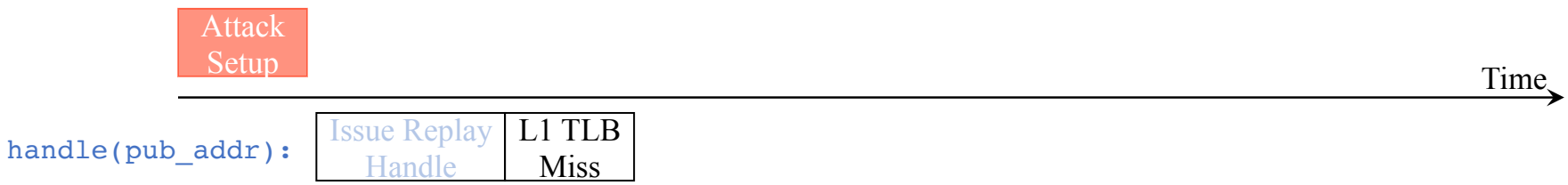
Timeline of a MicroScope Attack



Attacker Victim

ILLINOIS

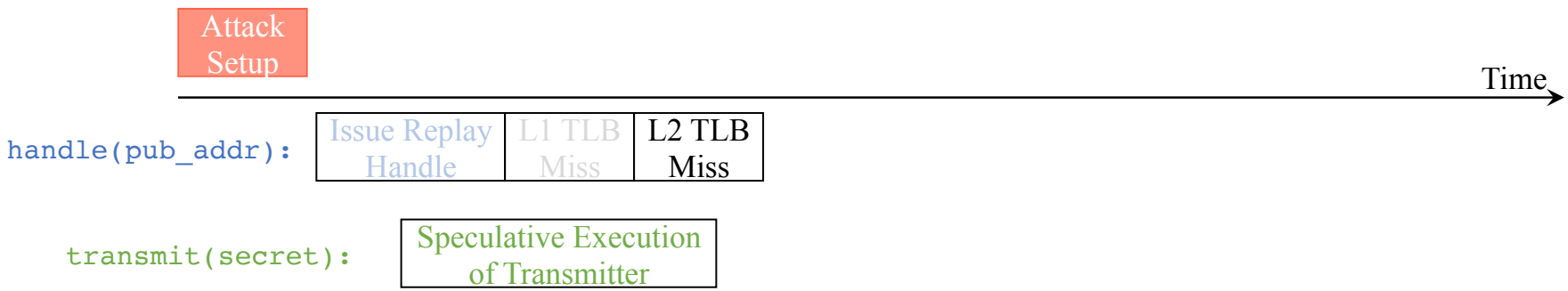
Timeline of a MicroScope Attack



Attacker Victim

ILLINOIS

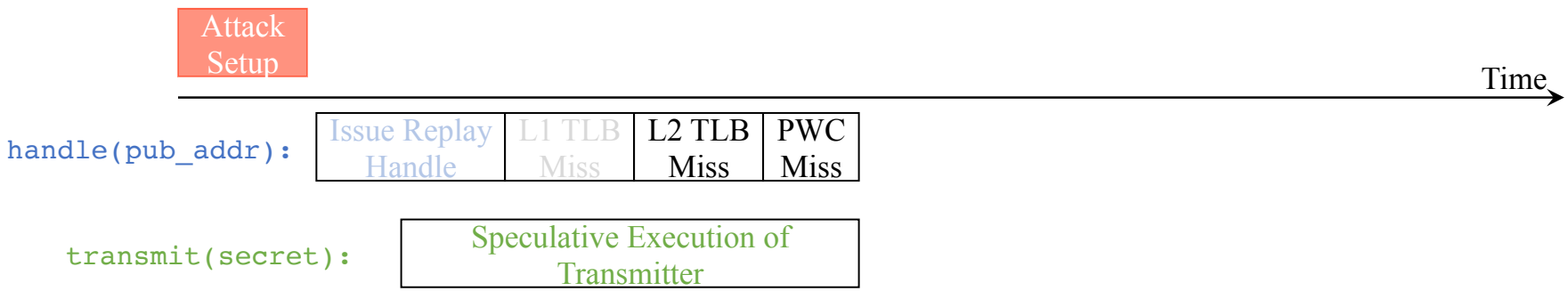
Timeline of a MicroScope Attack



Attacker Victim

ILLINOIS

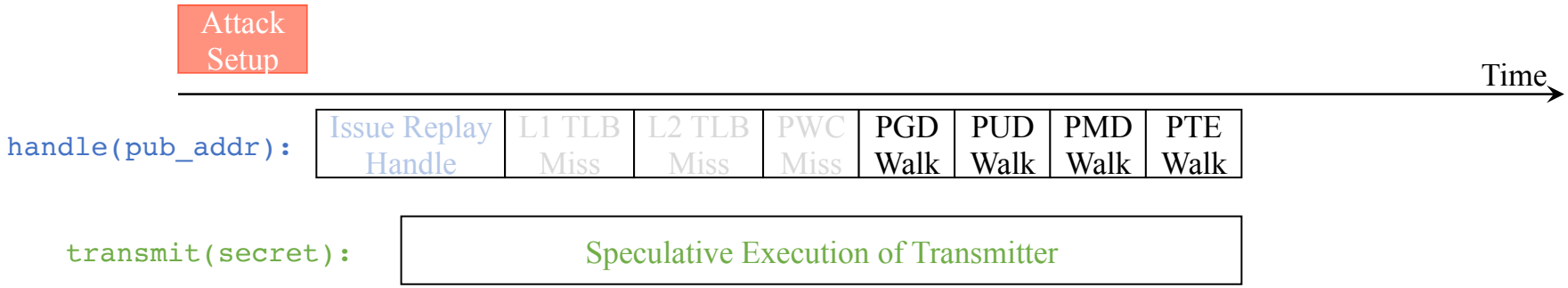
Timeline of a MicroScope Attack



Attacker Victim

ILLINOIS

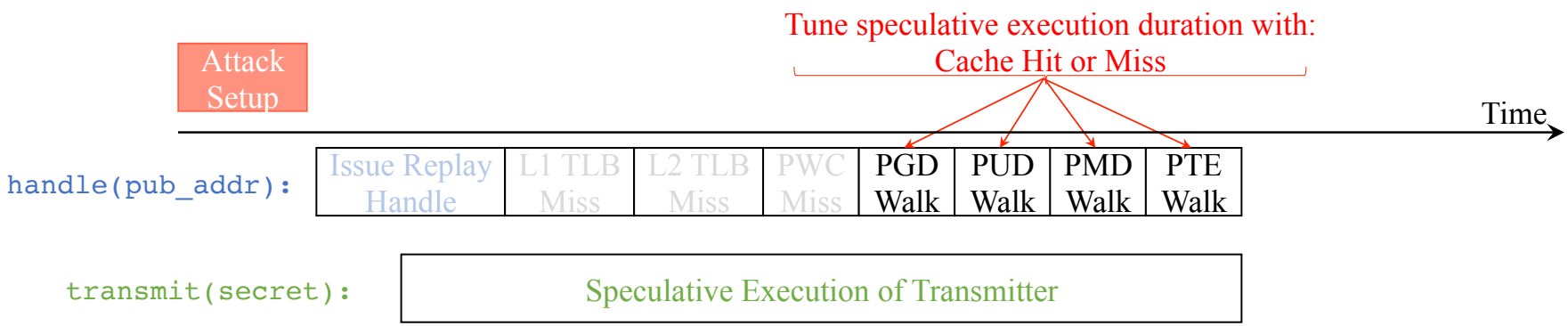
Timeline of a MicroScope Attack



Attacker Victim

ILLINOIS

Timeline of a MicroScope Attack



Attacker Victim

ILLINOIS

Timeline of a MicroScope Attack

Attacker Victim

ILLINOIS

Timeline of a MicroScope Attack

Attacker Victim

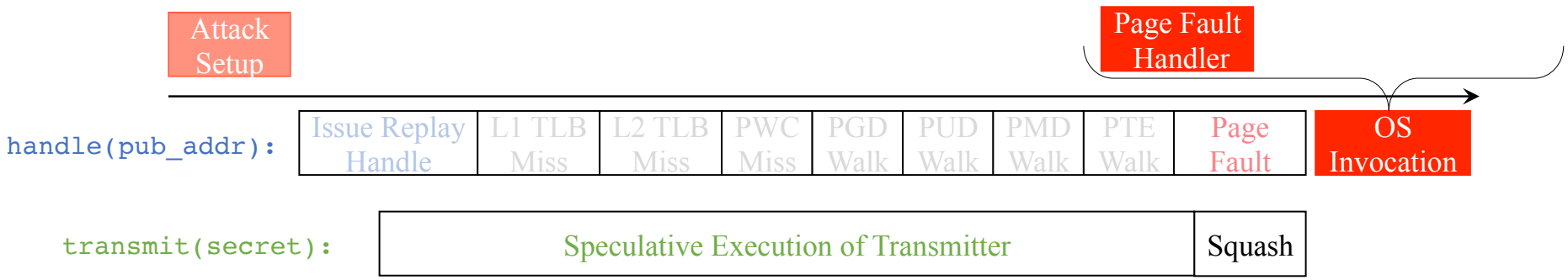
ILLINOIS

Timeline of a MicroScope Attack

Attacker Victim

ILLINOIS

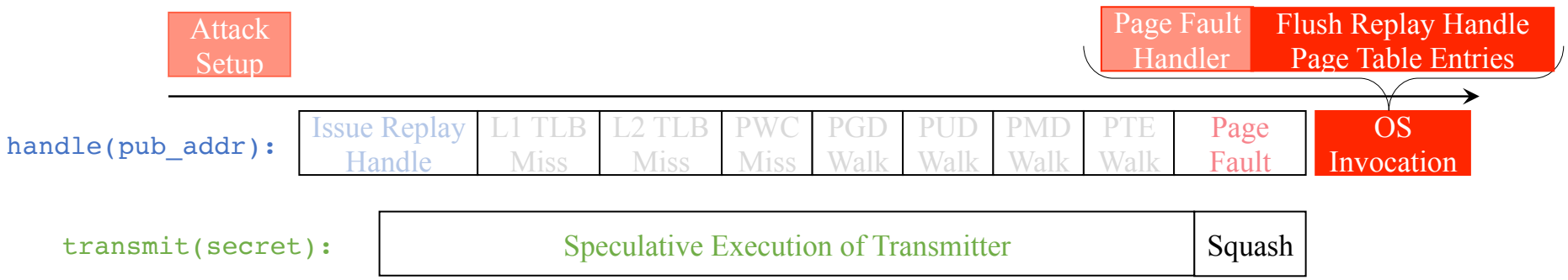
Timeline of a MicroScope Attack



Attacker Victim

ILLINOIS

Timeline of a MicroScope Attack



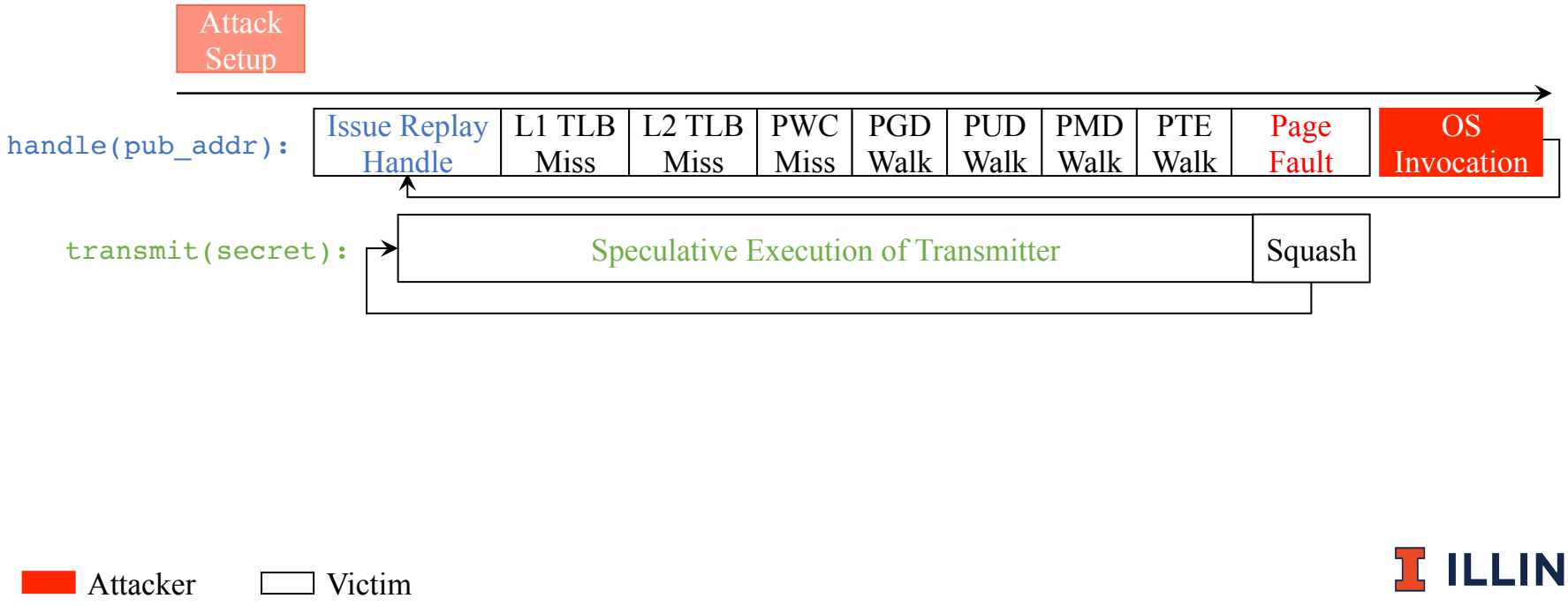
Attacker Victim

ILLINOIS

Timeline of a MicroScope Attack

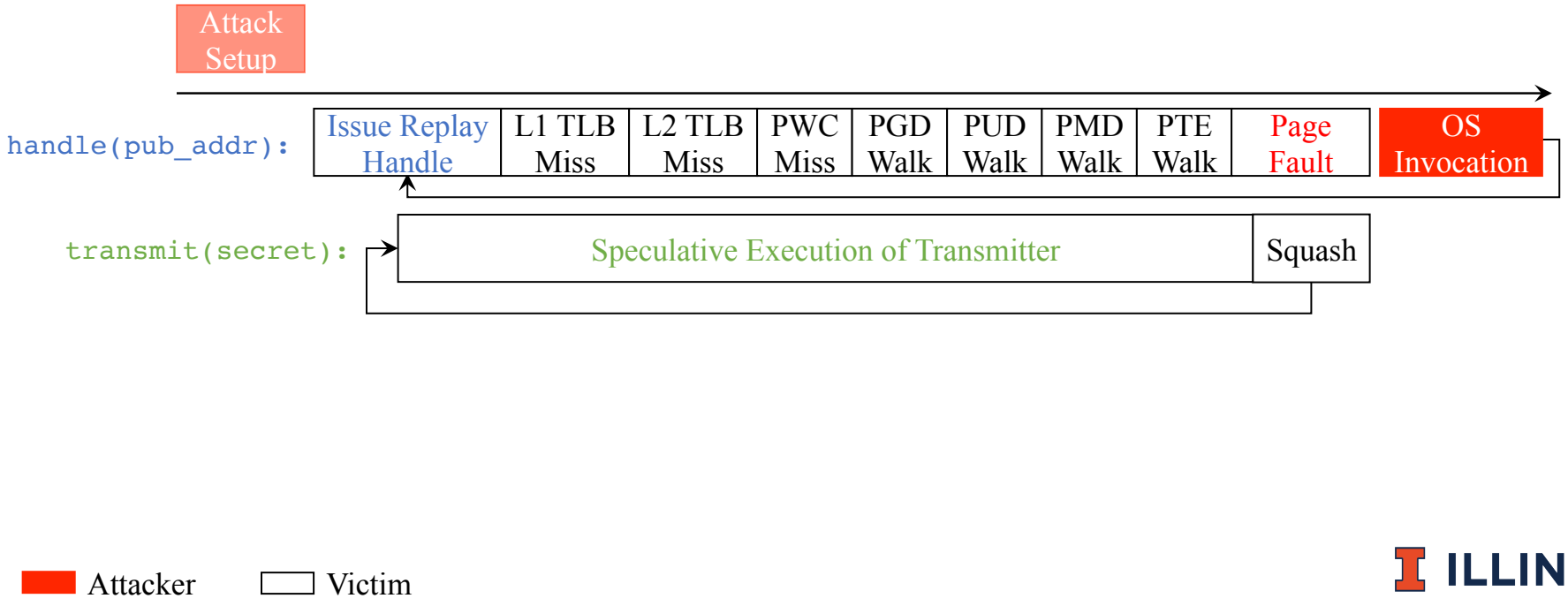
ILLINOIS

Timeline of a MicroScope Attack



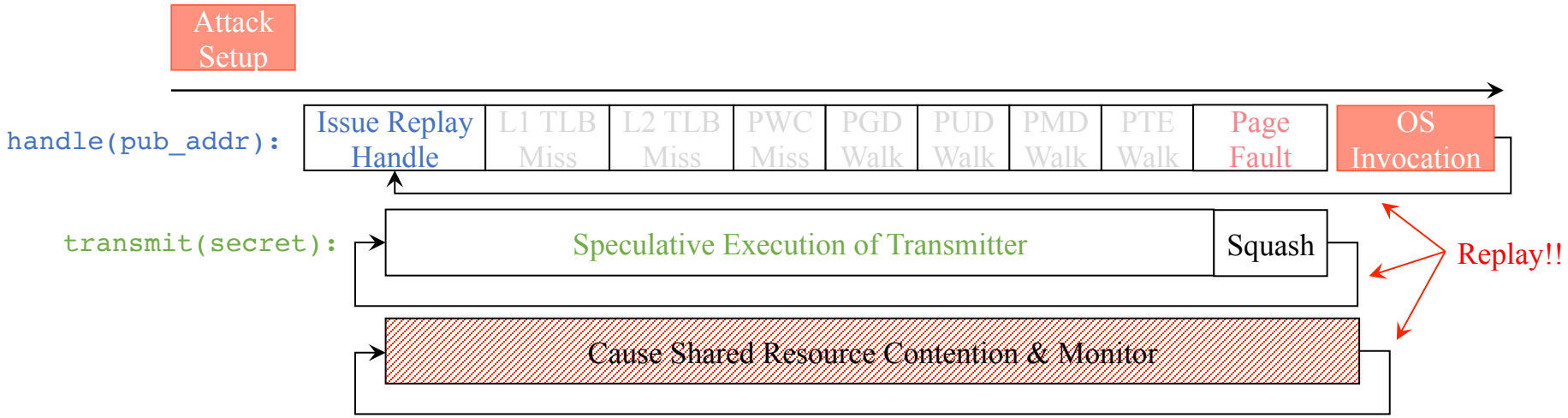
ILLINOIS

Timeline of a MicroScope Attack



ILLINOIS

Timeline of a MicroScope Attack



Attacker Victim Attacker Monitor/Contention thread

ILLINOIS

Loop Example

ILLINOIS

Loop Example

Victim:

```
for i in ...
  handle(pub_addrA);
  ...
  transmit(secret[i]);
  ...
  memOp(pub_addrB);
  ...
```

- **Goal:**
  - Leak secret in every iteration
- **Challenge:**
  - Monitor window of a replay handle < ROB length

ILLINOIS

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

ILLINOIS

In a Loop

Static Code

```
for i in ...
  handle(pub_addrA);
  ...
  transmit(secret[i]);
  ...
  pivot(pub_addrB);
  ...
```

Dynamic code

Iteration i=0

```
handle(pub_addrA);
...
transmit(secret[0]);
...
pivot(pub_addrB);
...
```

Iteration i=1

```
handle(pub_addrA);
...
transmit(secret[1]);
....
pivot(pub_addrB);
```

Monitor window

Replay Handle Transmit Code Pivot Instruction

ILLINOIS

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

ILLINOIS

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

Replay Handle Transmit Code Pivot Instruction

ILLINOIS

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

Replay Handle Transmit Code Pivot Instruction

ILLINOIS

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

New Monitor Window

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

Replay Handle Transmit Code Pivot Instruction

ILLINOIS

Port Contention Attack

ILLINOIS

Port Contention Attack with MicroScope

Victim:

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

No Loop!

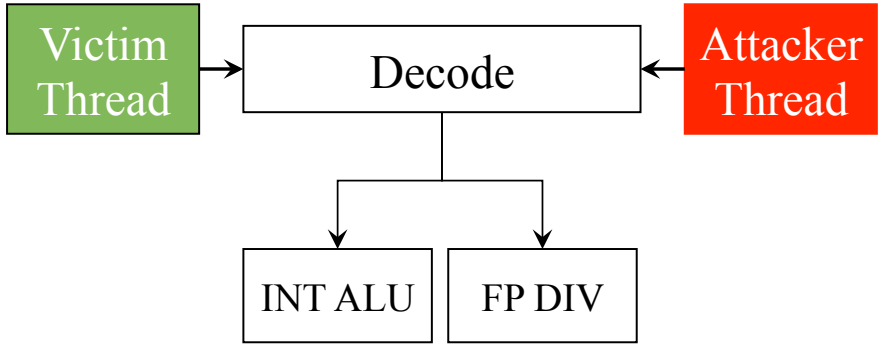
Attacker:

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

Replay Handle Transmit Code Attacker Code

ILLINOIS

Port Contention Attack with MicroScope



Victim:

```
handle(pub_addrA);
if (secret)
    victim_mul()
else
    victim_div()
```

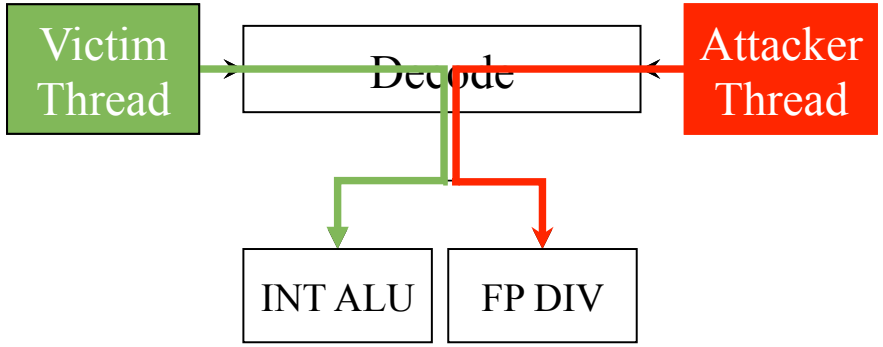
Attacker:

```
Setup replay attack
...
for ...
    t1 = time()
    attacker_div()
    t2 =time()
}
```

Replay Handle Transmit Code Attacker Code

ILLINOIS

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. }

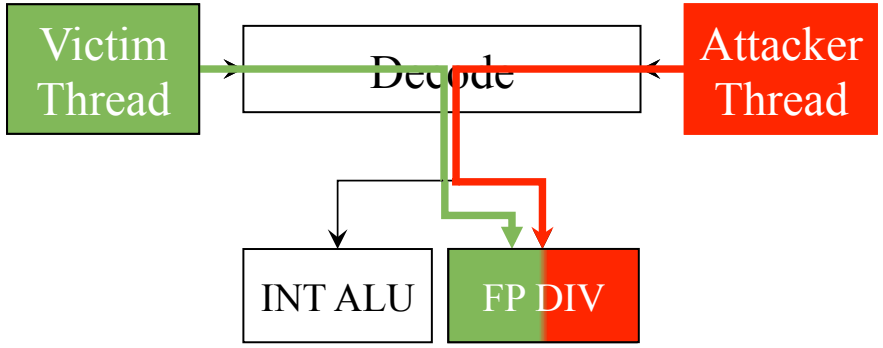
Replay Handle Transmit Code Attacker Code

ILLINOIS

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. }

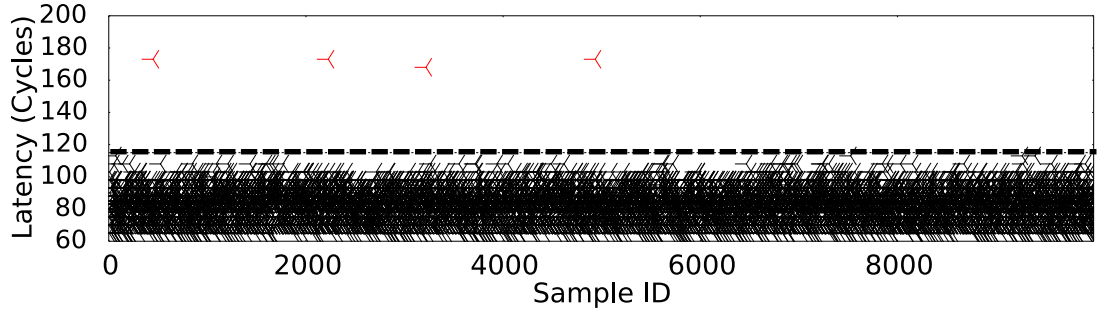
10,000 → 1

Replay Handle Transmit Code Attacker Code

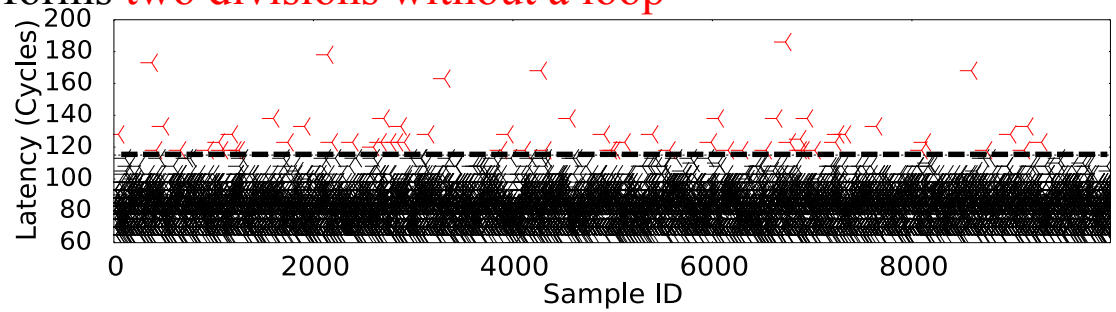
ILLINOIS

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

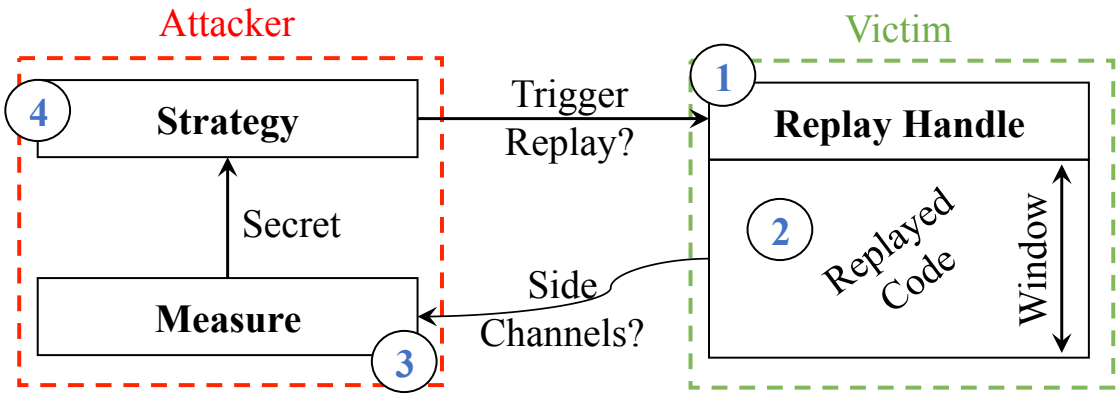


ILLINOIS

Generalizing Microarchitectural Replay Attacks

ILLINOIS

Microarchitectural Replay Attacks



This work:

1. Replay Handle → Page fault-inducing load
2. Replayed Code → Leaky instruction
3. Side Channel → uarch structures
4. Attacker strategy → Page fault until denoise

Changing each can result in different attacks!!

ILLINOIS

More On the Paper

- Other attack examples
- MicroScope framework implementation
- Attacks on AES
- Countermeasures discussion

ILLINOIS

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

ILLINOIS

Takeaways

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!

ILLINOIS