ExSpectre

Hiding Malware in Speculative Execution

Jack Wampler, Ian Martiny, Eric Wustrow

University of Colorado **Boulder**

Overview

Previous work used speculative execution to leak information

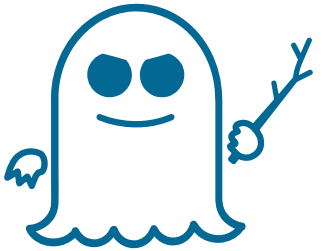
This work:

Use **speculative execution** to **hide** arbitrary computation

- Useful for malware or white-box applications

Outline

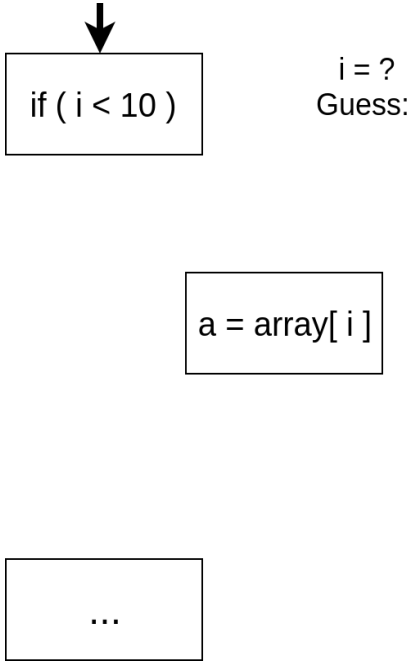
- Background
  - Speculative Execution
  - Spectre / Meltdown
- Threat Model & Architecture
- Fundamental limits of speculative execution
  - How much work can be done?
  - What kinds of work can gadgets do?
- My processor can do what speculatively?!
  - Techniques for obfuscating program behavior
- System Implementation



Background: Speculative Execution & Spectre

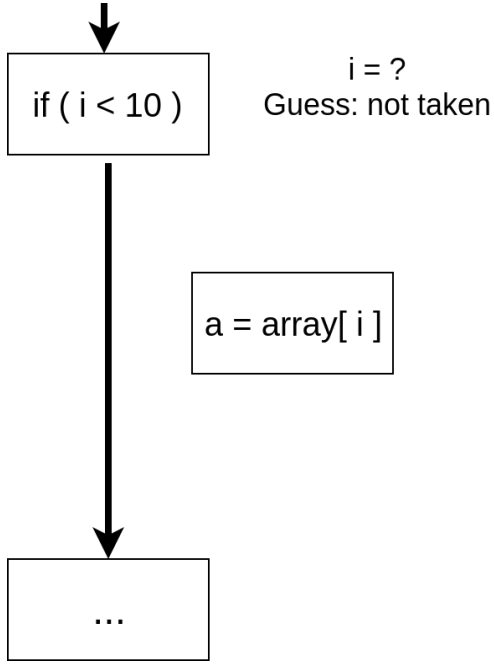
Speculative Execution

```
if (i < 10) {
    a = array[i];
}
```



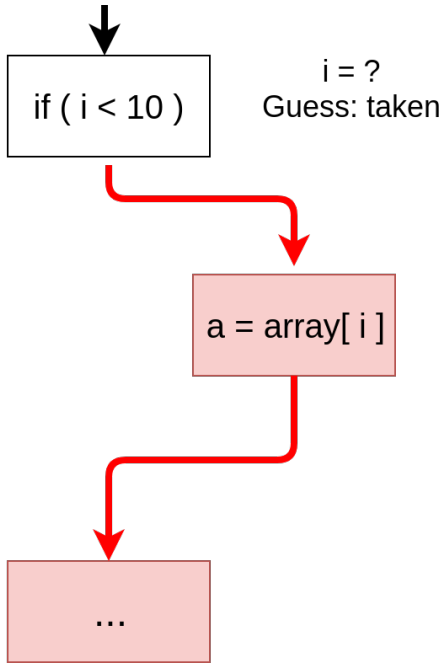
Speculative Execution

```
if (i < 10) {
    a = array[i];
}
```



Speculative Execution

```
if (i < 10) {
    a = array[i];
}
```



Executes instructions inside branch **without** conditions applied.

Results are discarded once speculation resolves… right?

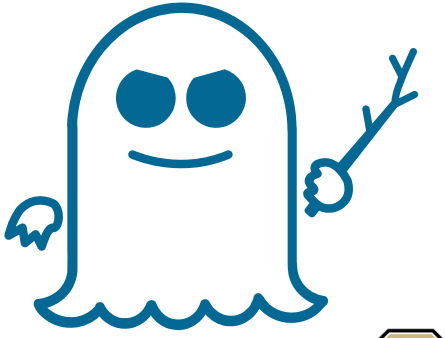
Spectre 1

Direct Jumps

1) Attacker **trains** branch predictor.

```
array[10] = "9876543210"
secret[] = "P4ssw0rd1234";
if (i < 10) {
    a = array[i];
    b = map[a];
}
```

Victim Process



P. Kocher et. al. “Spectre attacks: Exploiting speculative execution,” 2019

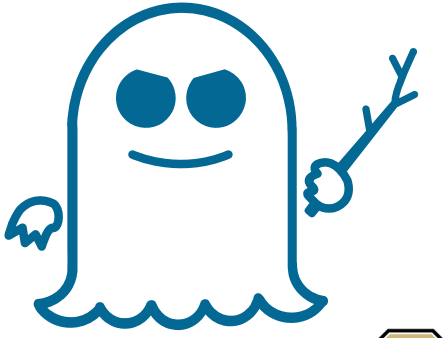
Spectre 1

Direct Jumps

```
array[10] = "9876543210"
secret[] = "P4ssw0rd1234";
if (i < 10) {
    a = array[i];
    b = map[a];
}
```

Victim Process

1) Attacker **trains** branch predictor.
2) Attacker provides out of bounds index to the array.



P. Kocher et. al. “Spectre attacks: Exploiting speculative execution,” 2019

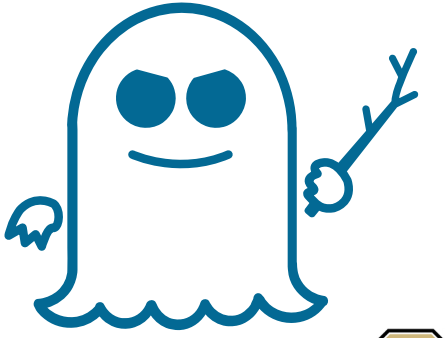
Spectre 1

Direct Jumps

```
array[10] = "9876543210"
secret[] = "P4ssw0rd1234";
if (i < 10) {
    a = array[i];
    b = map[a];
}
```

Victim Process

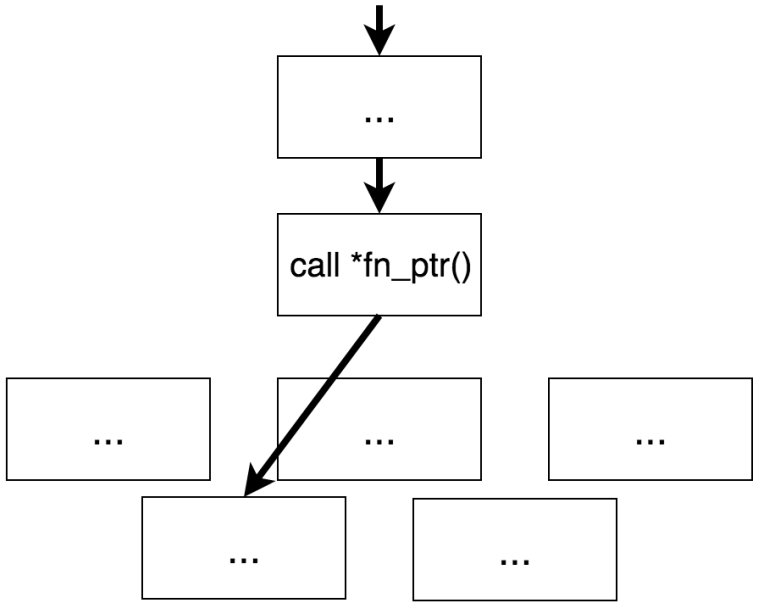
1) Attacker **trains** branch predictor.
2) Attacker provides out of bounds index to the array.
3) Attacker exfiltrates sensitive info via **side-channel**

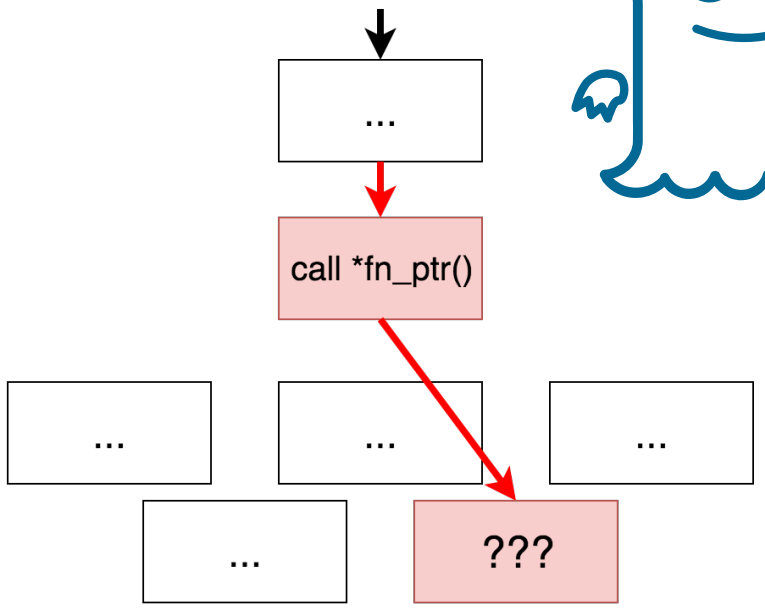


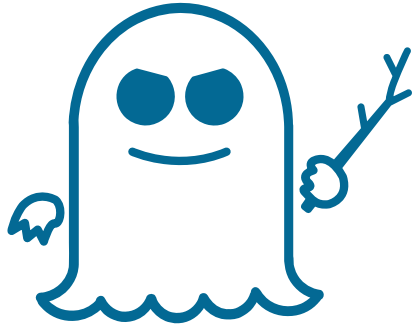
P. Kocher et. al. “Spectre attacks: Exploiting speculative execution,” 2019

Spectre 2

Indirect Jumps







The branch predictor guesses **where** control flow will be redirected.

P. Kocher et. al. “Spectre attacks: Exploiting speculative execution,” 2019

Overview

Spectre / Meltdown use speculative execution to **Leak** information

This work will use speculative execution to **Hide** arbitrary malicious computation

Hiding Computation

Current Malware

- Packers
  - Dynamic analysis can undo packing
- Triggers / Red Pill
  - Static analysis can identify conditions and triggers

- Our Work: **ExSpectre**
  - Require analyst to **precisely model** speculative execution

ExSpectre Threat Model

Attacker Capabilities

✔ Install binary on target machine

✔ Influence trigger program

- Possibly remotely

Reverse Engineer Capabilities

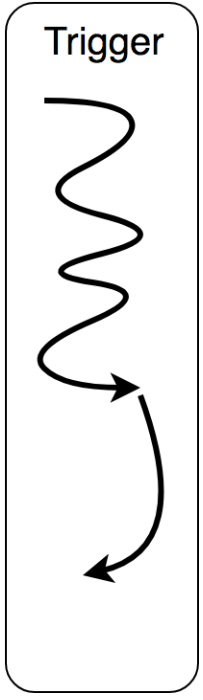
✔ Can use static and dynamic analysis.

✗ Can’t introspect processor’s speculative state.

✗ Can’t run trigger program

ExSpectre Architecture

Trigger – Trains branch predictor
pattern → **target_fn**

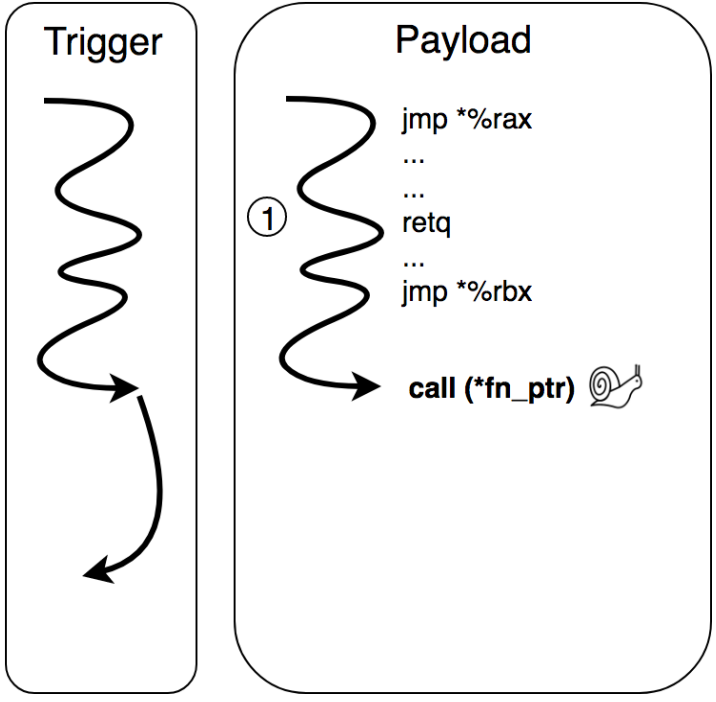


ExSpectre Architecture

Trigger – Trains branch predictor
pattern → **target_fn**

Payload

- Executes same jump **pattern**

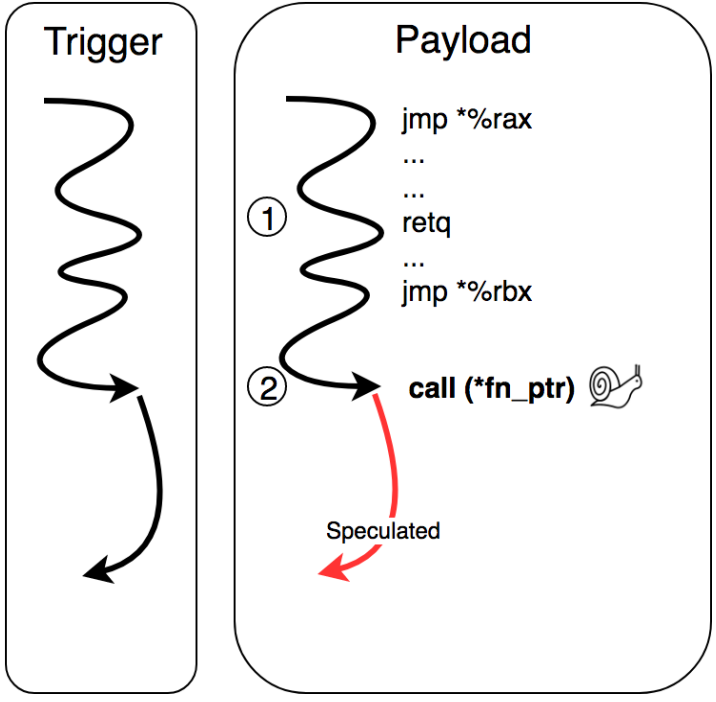


ExSpectre Architecture

Trigger – Trains branch predictor
pattern → **target_fn**

Payload

- Executes same jump **pattern**
- CPU mis-speculates to **target_fn**

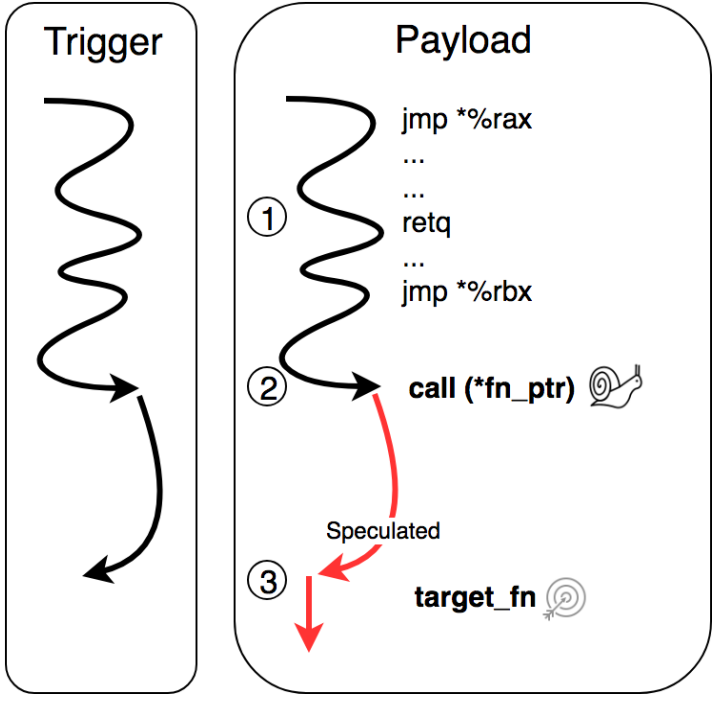


ExSpectre Architecture

Trigger – Trains branch predictor
pattern → **target_fn**

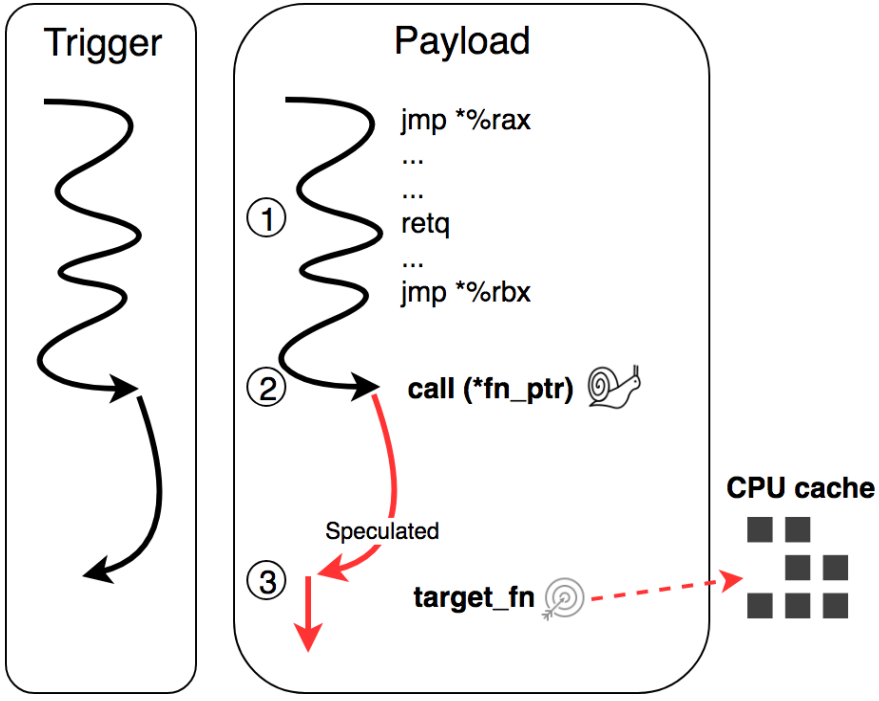
Payload

- Executes same jump **pattern**
- CPU mis-speculates to **target_fn**
- Executes a short **gadget** speculatively



ExSpectre Architecture

Trigger – Trains branch predictor
pattern → **target_fn**

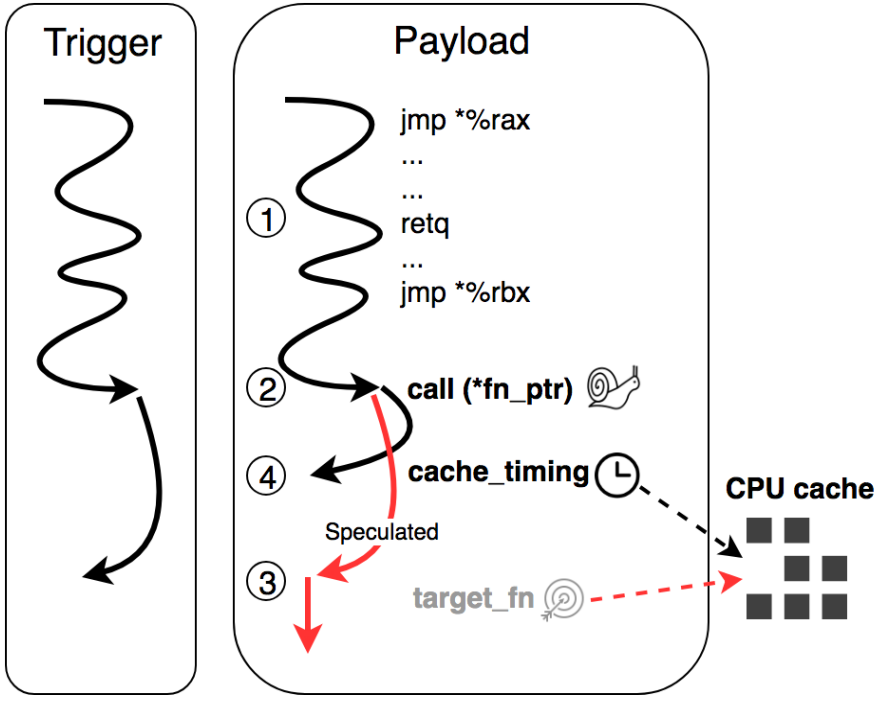


Payload

- Executes same jump **pattern**
- CPU mis-speculates to **target_fn**
- Executes a short **gadget** speculatively
- Results sent to r**eal world** via side channel

ExSpectre Architecture

Trigger – Trains branch predictor
pattern → **target_fn**

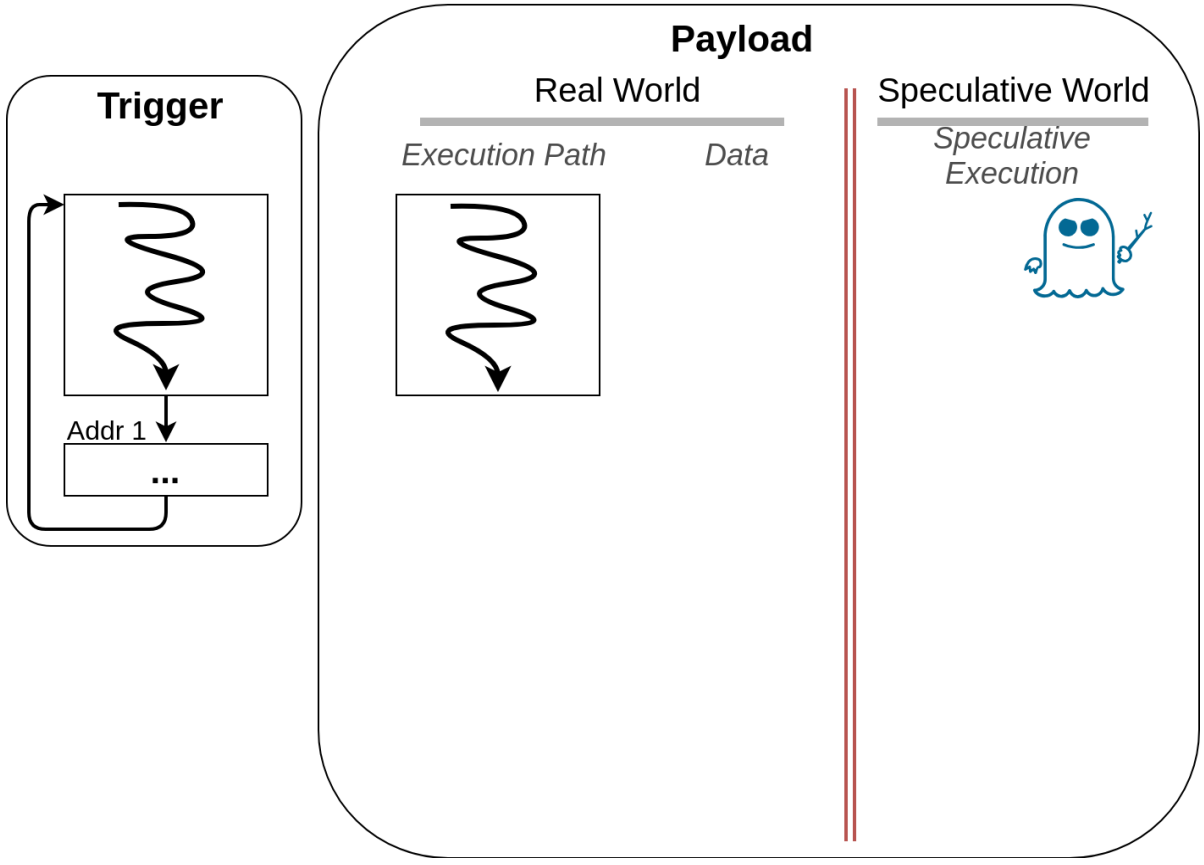


Payload

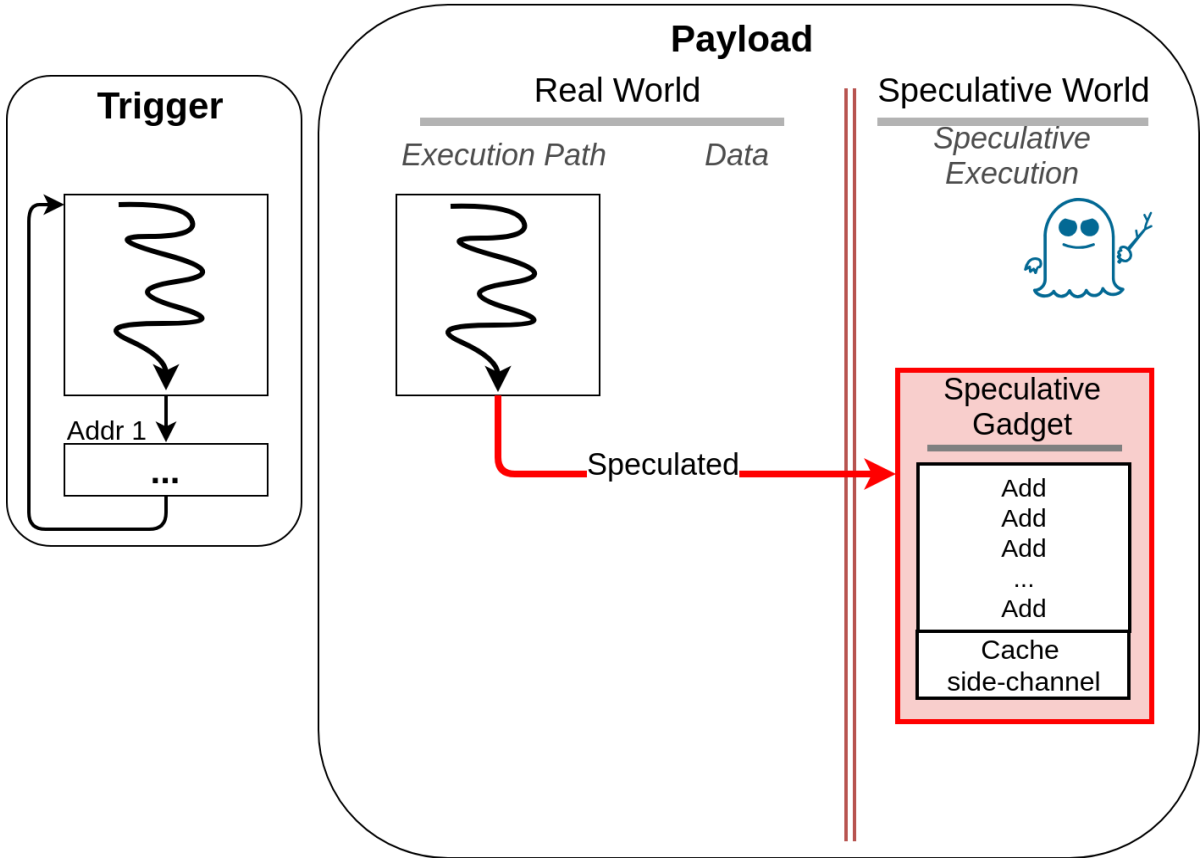
- Executes same jump **pattern**
- CPU mis-speculates to **target_fn**
- Executes a short **gadget** speculatively
- Results sent to r**eal world** via side channel

How much work can be done **speculatively**?

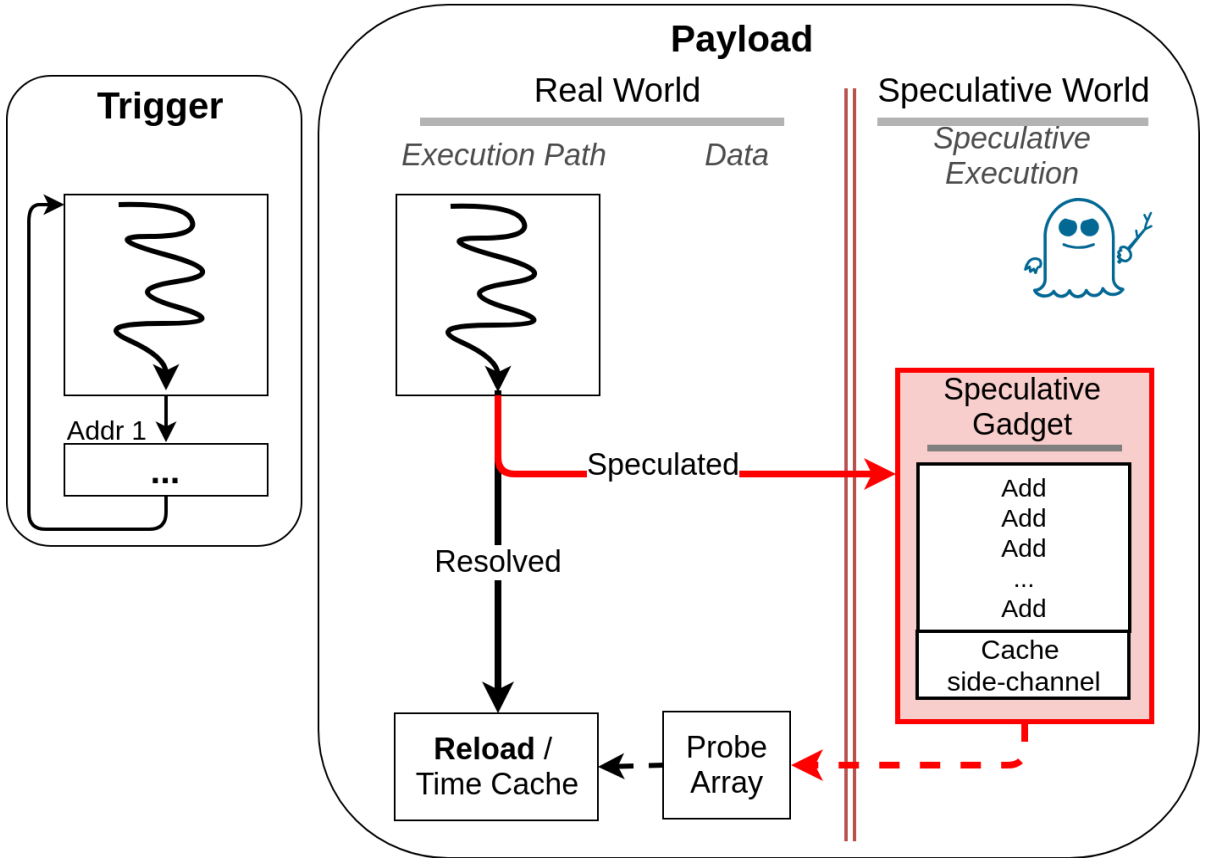
Speculative Window



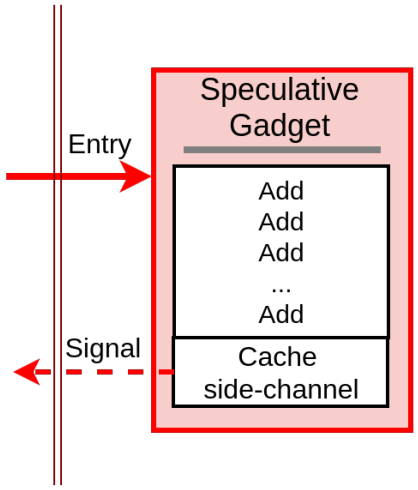
Speculative Window

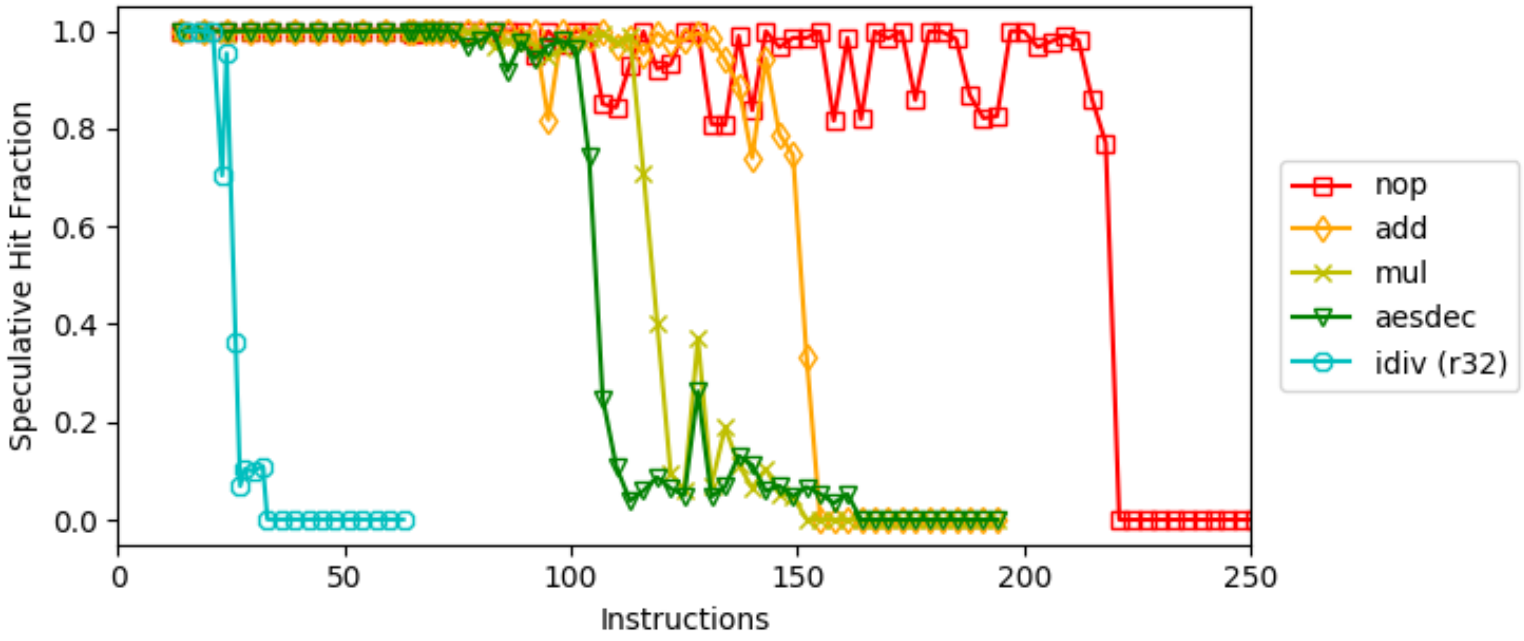


Speculative Window



Speculative Window

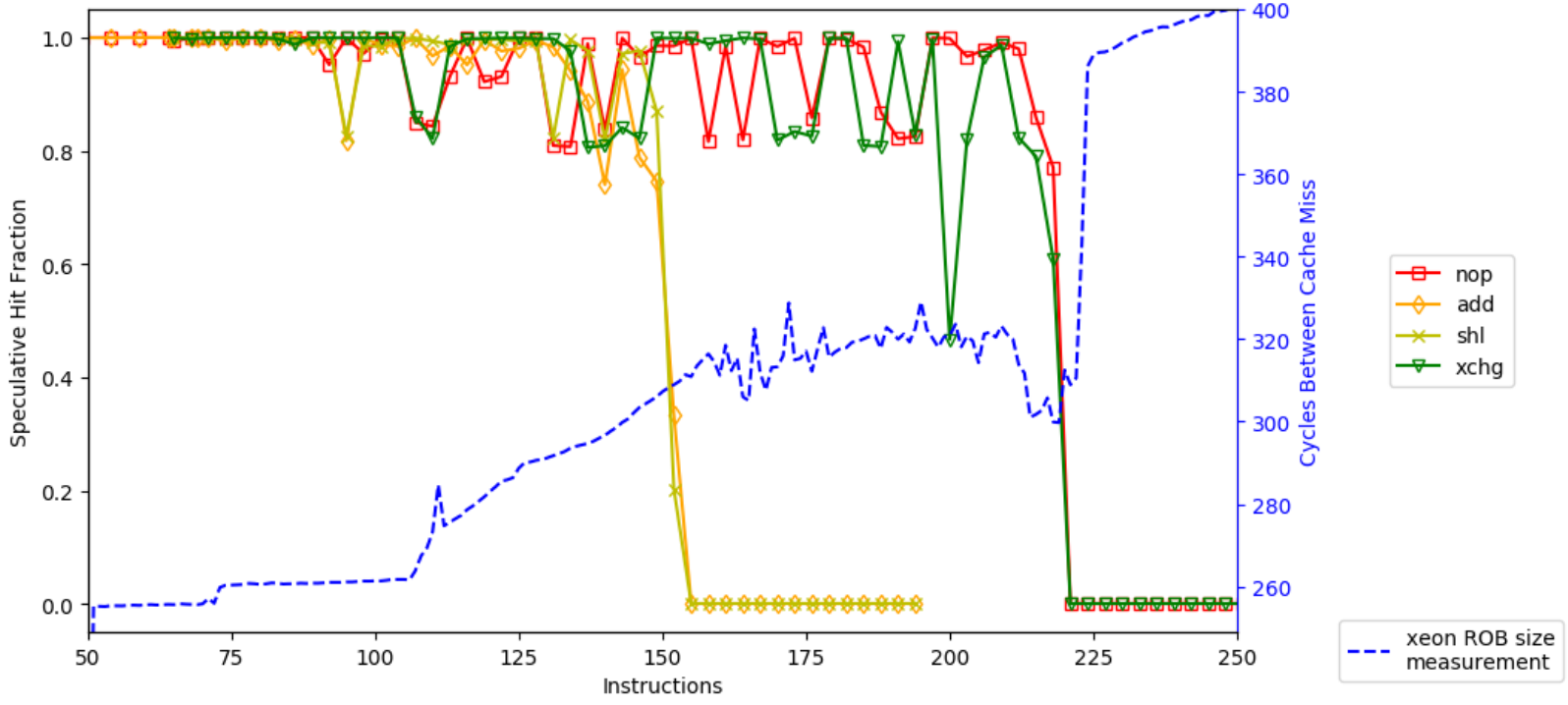




1) Different instructions have different limitations.

2) **Simpler** vs. more **complex** instructions.

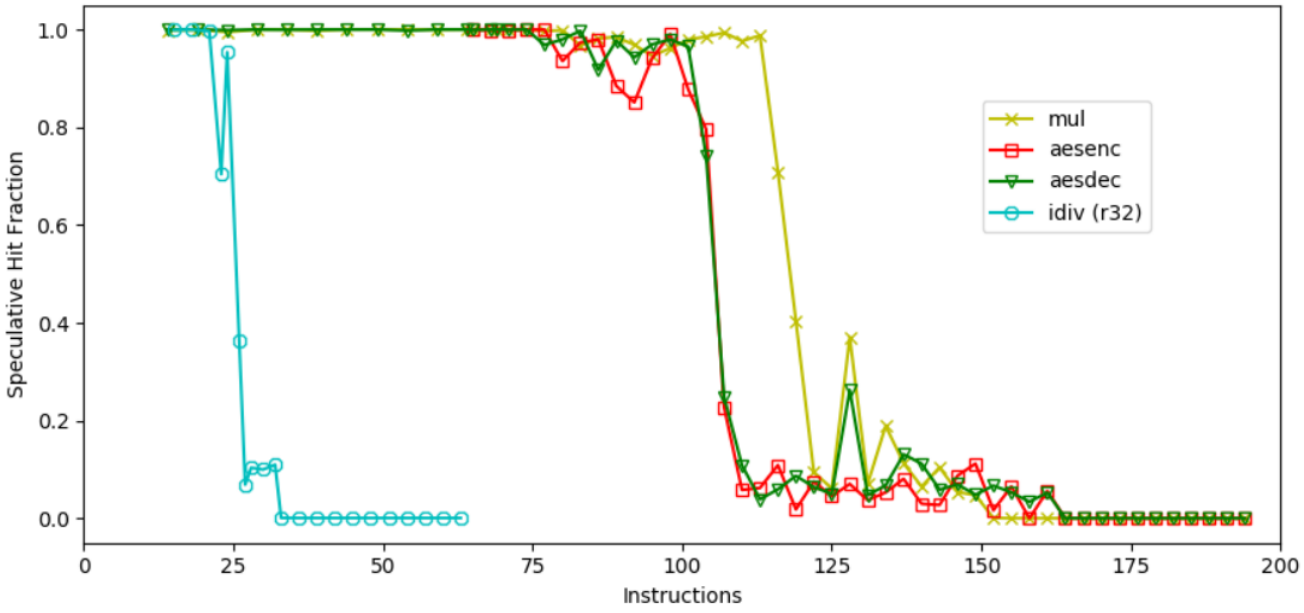
Speculative Window



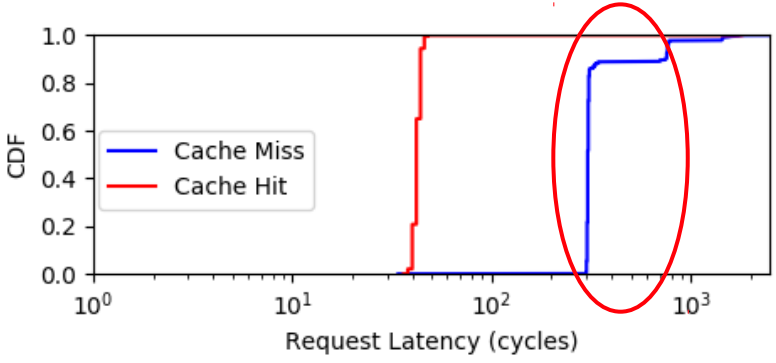
The maximum number of instructions aligns with **ROB size** for the simplest instructions.

H. Wong, “Measuring reorder buffer capacity,” May 2013

Speculative Window



Complex instructions



Cache miss resolves in ~300-750 cycles

Speculative Window

- **Cache Limit** – Cache miss duration
  - Signal still detectable after 80% resolution (300 cycles)
    - No signal after 95% resolution (750 cycles)
  - Using instructions with high CPI hit this limit first.
- **ROB Limit** – ROB size: 220 µops (Skylake)

We can execute ~100-150 instruction speculatively

What kinds of work can gadgets do?

What kinds of work can gadgets do?

- ✔ **Control flow, logical, & arithmetic instructions**

What kinds of work can gadgets do?

- ✔ Control flow, logical, & arithmetic instructions
- ✔ **AES-NI instructions**

What kinds of work can gadgets do?

- ✔ Control flow, logical, & arithmetic instructions
- ✔ AES-NI instructions
- ✔ **Load (in cache)**

What kinds of work can gadgets do?

- ✔ Control flow, logical, & arithmetic instructions
- ✔ AES-NI instructions
- ✔ Load (in cache)
- **Load (out of cache)**

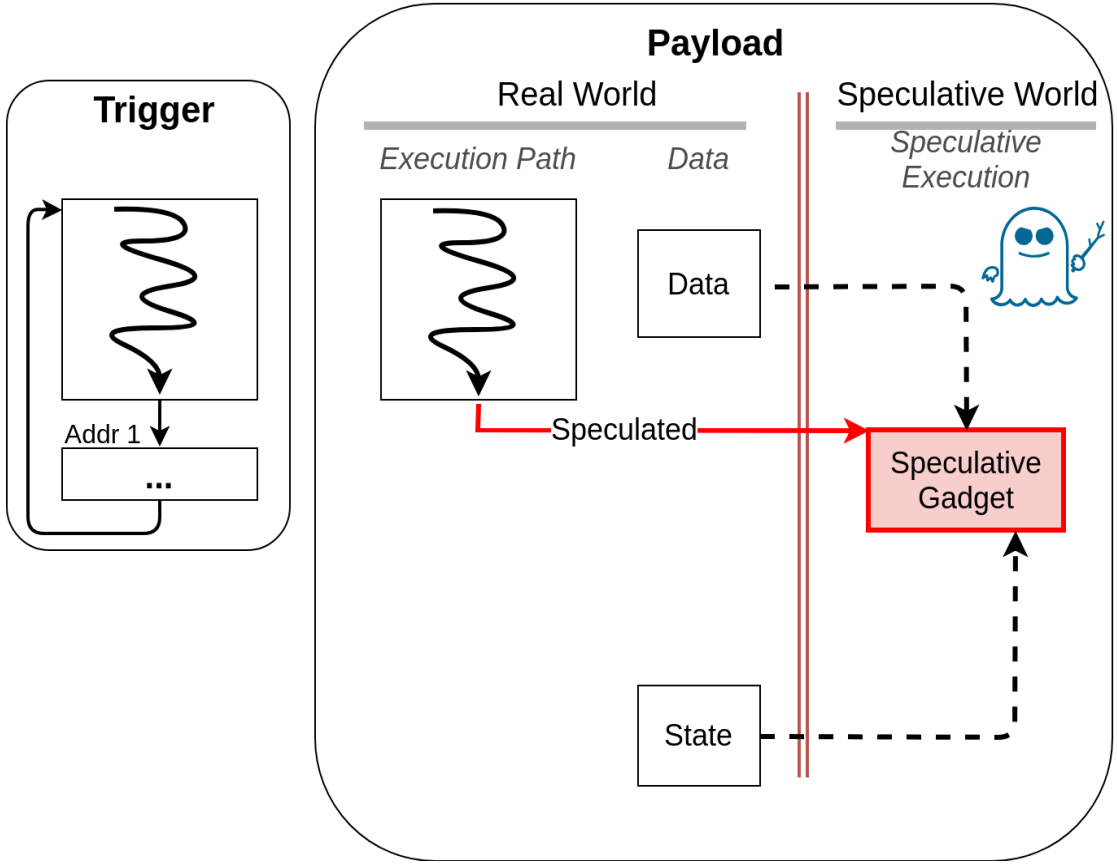
What kinds of work can gadgets do?

- ✔ Control flow, logical, & arithmetic instructions
- ✔ AES-NI instructions
- ✔ Load (in cache)
- ● Load (out of cache)
- ✗ **Store**

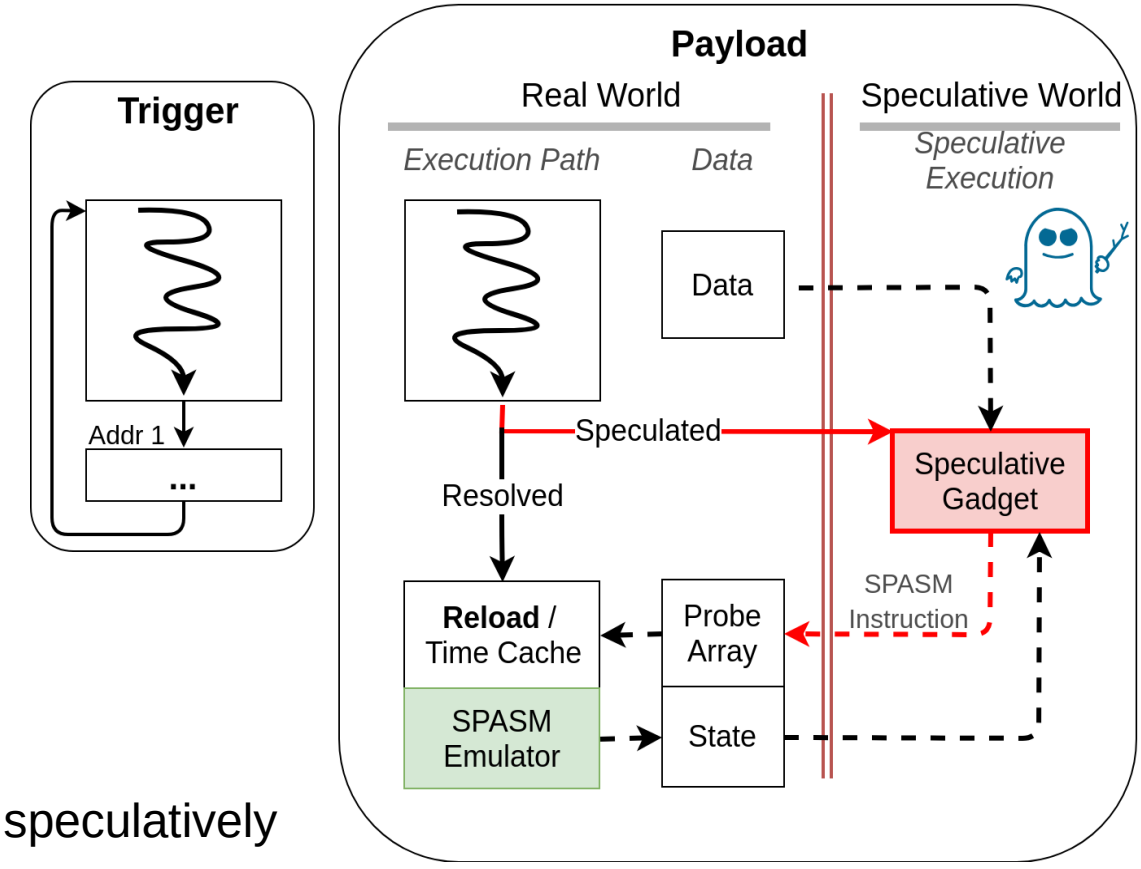
What kinds of work can gadgets do?

- ✔ Control flow, logical, & arithmetic instructions
- ✔ AES-NI instructions
- ✔ Load (in cache)
- ● Load (out of cache)
- ✗ Store
- ✗ **Syscalls**

What kinds of work can gadgets do?



Emulator



Emulator

- ✓ **Store**
- ✓ **Syscall**
- Maintains state
- State accessible speculatively

How to train your Branch Predictor

Triggers – Spectre Variants

Spectre 2 Indirect Branches	Entry point determined by training in **trigger process**.
Returns	Entry point determined by training in **trigger process**.
Spectre 1.1 Direct Branches	Entry point determined by attacker controlled data.

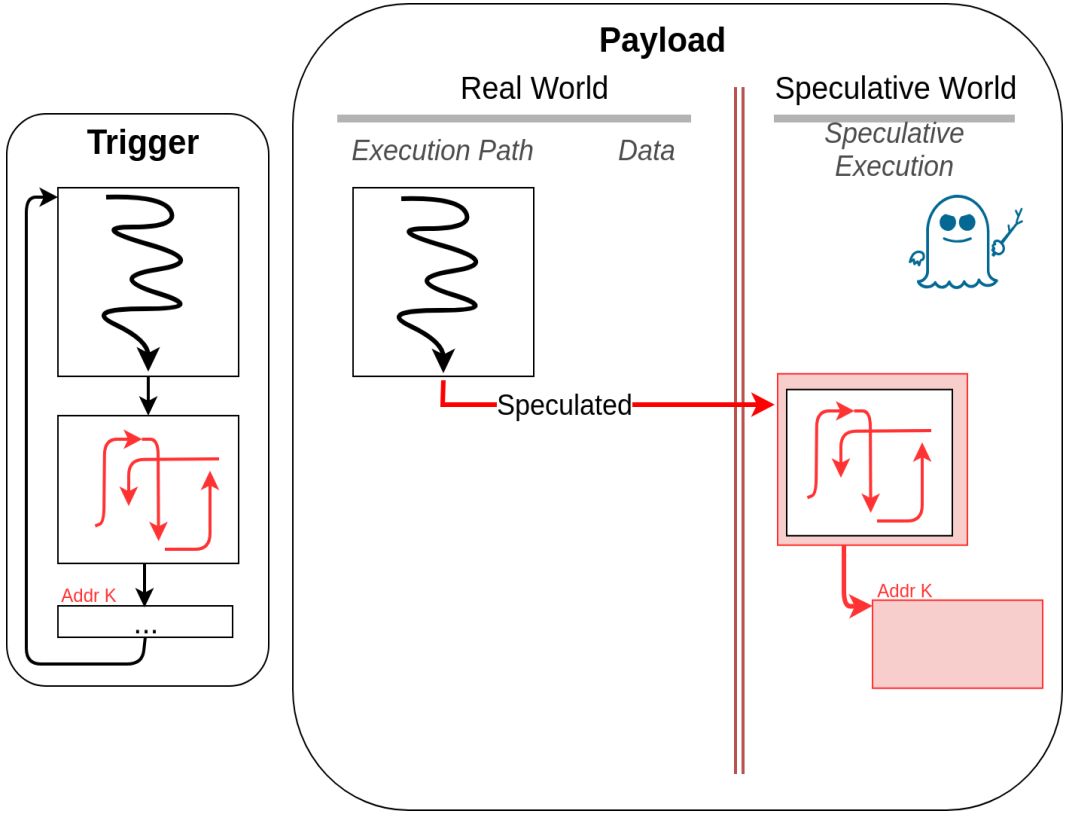
Benign Triggers

The trigger program doesn't have to be complicit (e.g. – openssl)

V. Kiriansky and C. Waldspurger, "Speculative buffer overflows: Attacks and defenses," 2018

Triggers – Nested Speculation

Processors continue speculate branches even while executing speculatively.

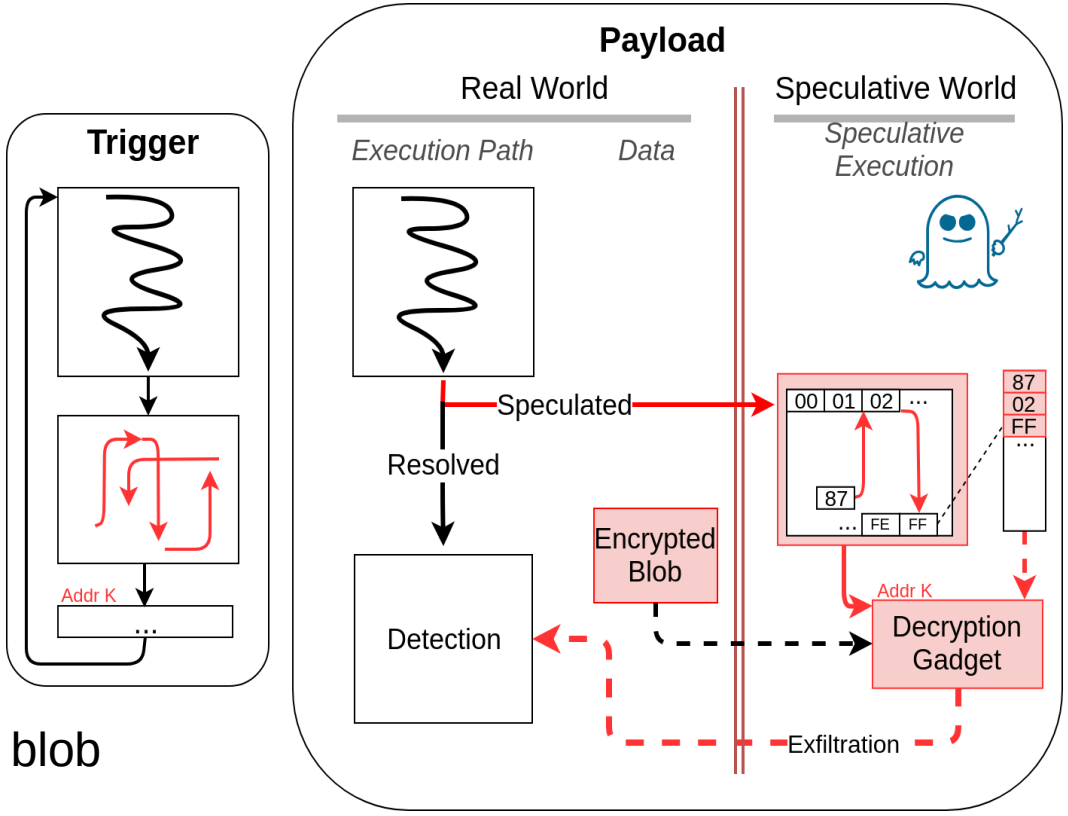


Tested successfully on **Intel Core i5-7200U** and **Intel Xeon CPU E3-1270 v6**

Payloads - Decryption Gadgets

AES-NI instructions

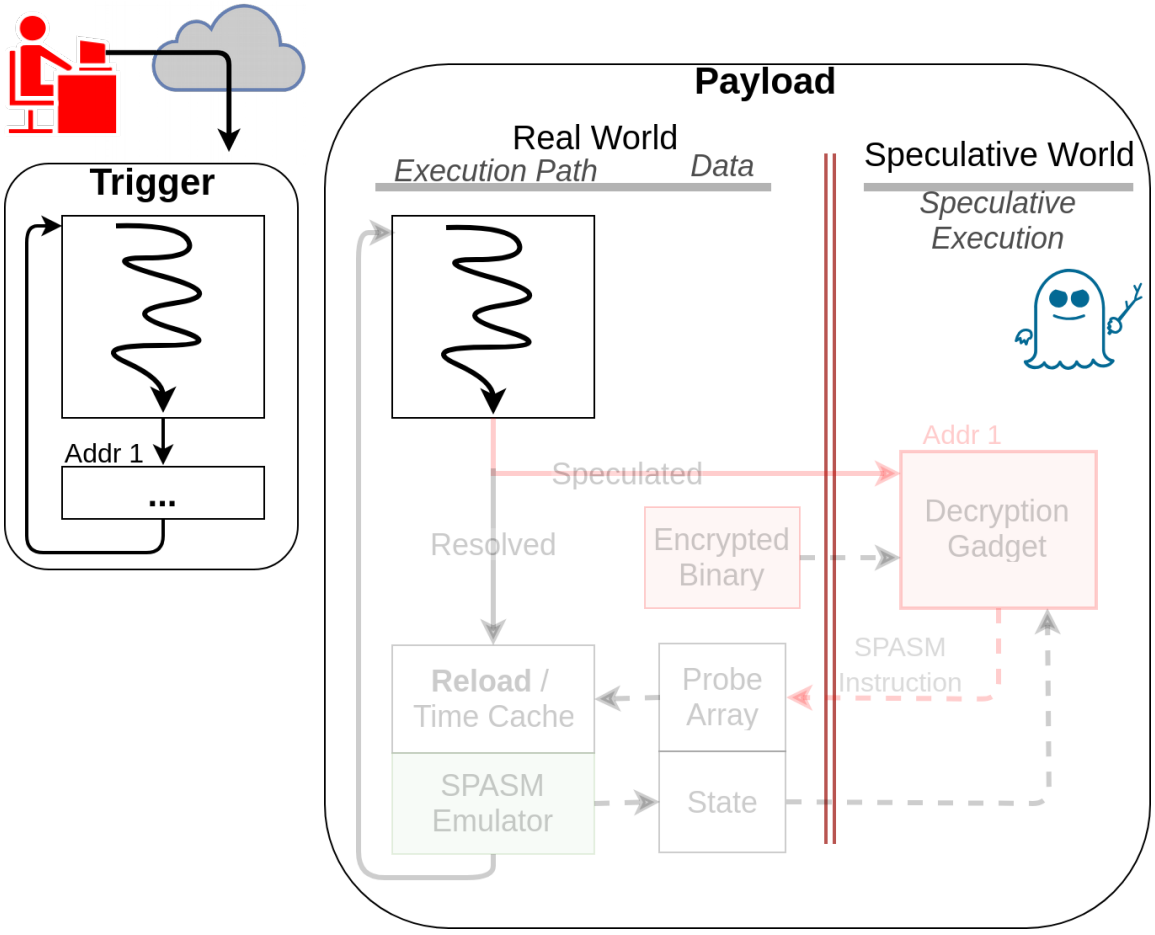
- Hardware supported AES Block Encryption and Decryption
- Abbreviated Key derivation (or partial key expansion)



Incrementally decrypt a data blob **speculatively**

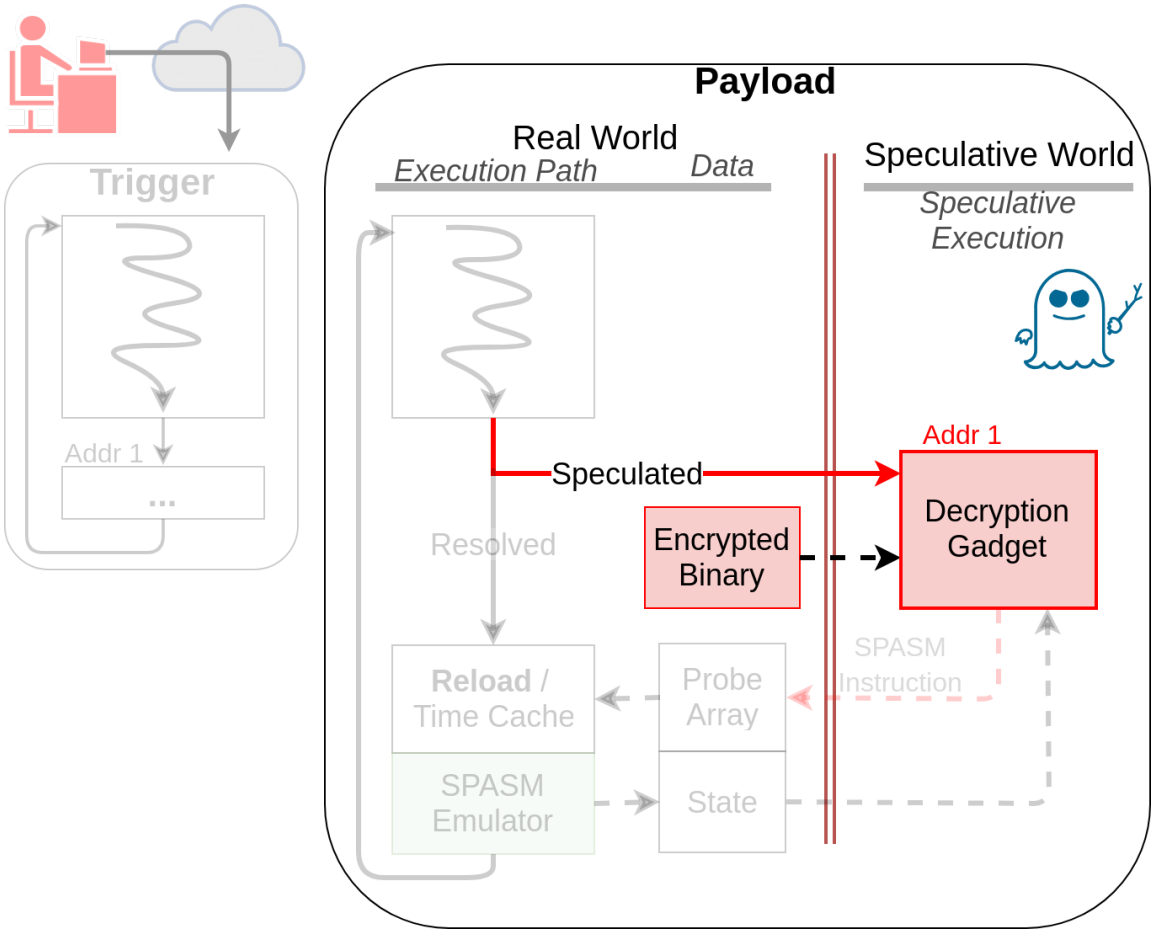
Implementation: Putting it all together

System Implementation



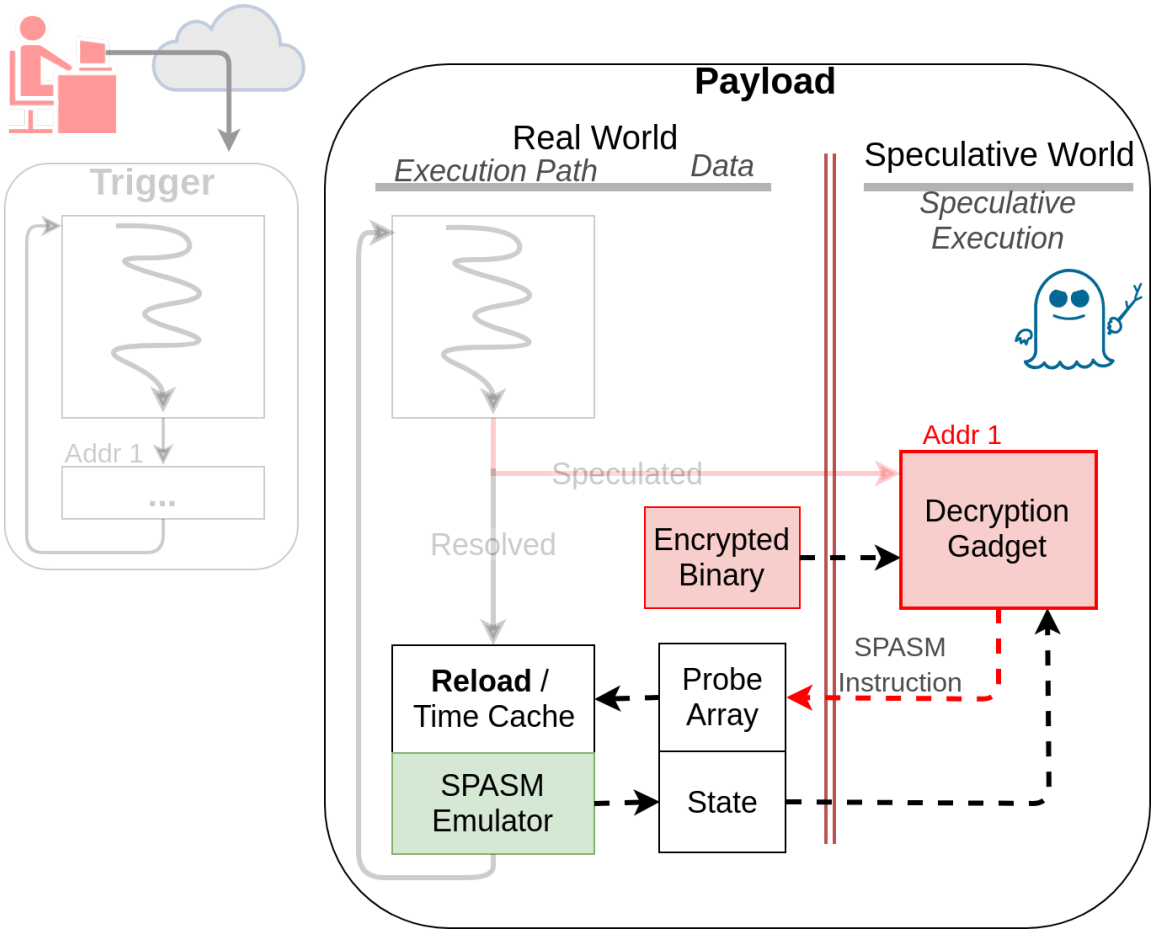
- **Benign Remote Trigger**

System Implementation



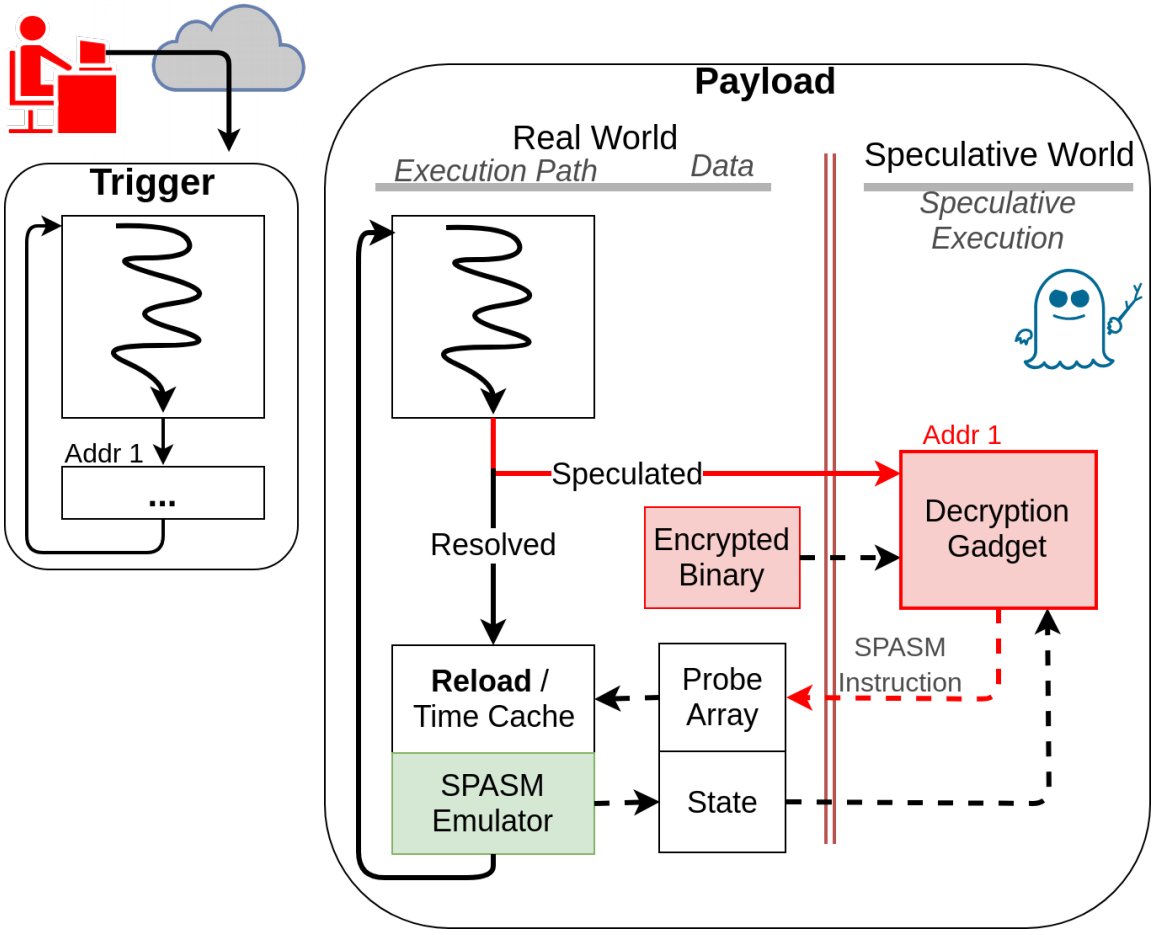
- Benign Remote Trigger
- **Decryption Gadget**

System Implementation



- Benign Remote Trigger
- Decryption Gadget
- **Custom Emulator**

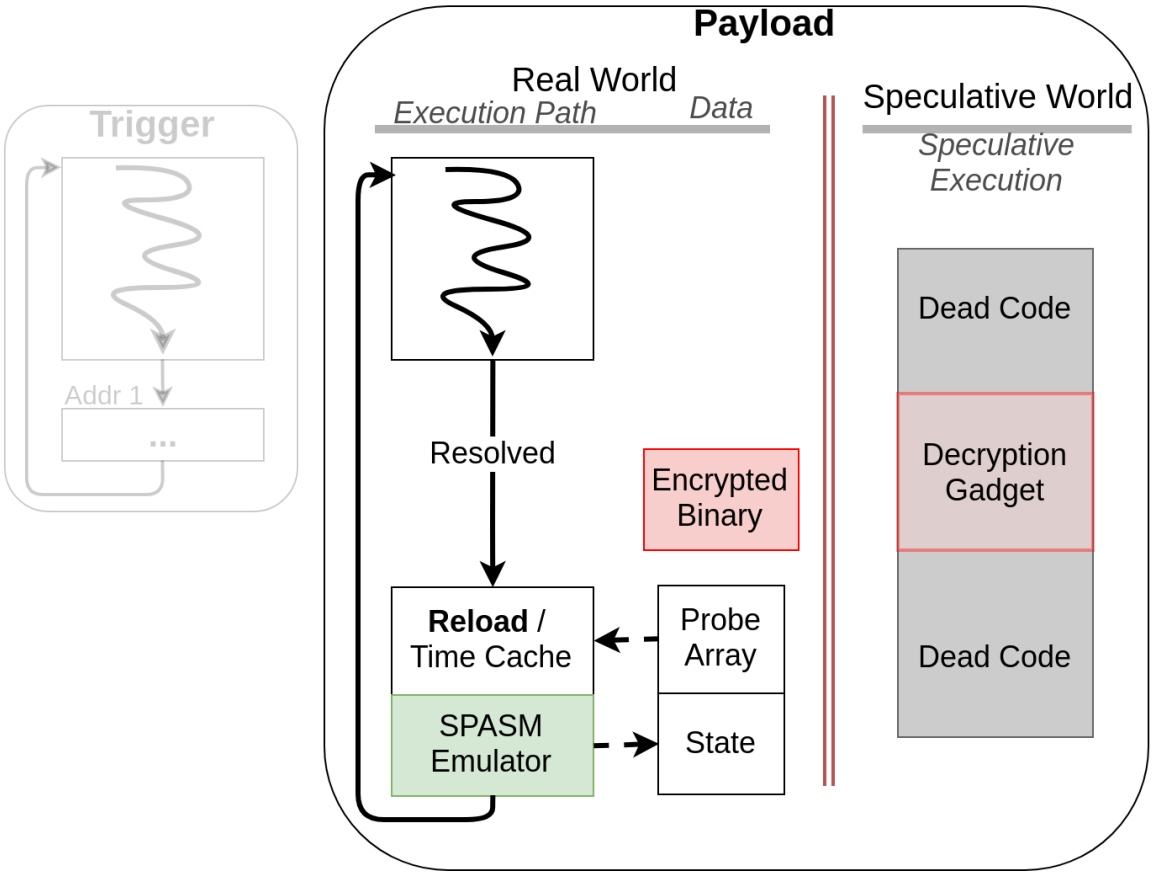
System Implementation



- Benign Remote Trigger
- Decryption Gadget
- Custom Emulator

Quickly launch **reverse shell** once trigger becomes present

Analysis without Trigger



- ✔ Emulator Exists
- ✔ Encrypted Binary
- ✔ Cache Probe

- ✘ Gadgets
- ✘ Entry Point

Defenses & Analysis

Insufficient Defenses

Spectre Defenses:

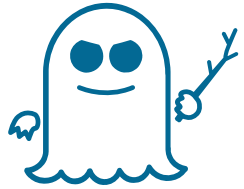
IBPB - Predictor state optionally cleared on context switch

IBRS - Predictor cleared on kernel enter/exit

STIBP - Different predictor per hyperthread

Retpoline - Software patch for Spectre II (opt-in)

Cache Coloring - but still other side channels



...but most are ***opt-in!***

Attacker can choose no defenses

Summary

We use speculative execution to:

- **Hide core malware functionality**
  - Difficult for static/dynamic analysis to reverse engineer
  - Implemented **reverse shell** with support of a small emulator
- **Triggered by**
  - Other potentially benign / remote programs
  - Input data

Questions?

github.com/ewust/speculake