A Genetic Algorithm for a Spectre Attack Agnostic to Branch Predictors

Dorian Bourgeoisat
dorian.bourgeoisat@telecom-paris.fr
Le June 17, 2023

Plan

Plan

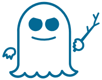
A familly of attacks

Spectre:

- familly of attacks
- speculative execution to leak secret data through a covert channel (cache, contentions, etc.).

Main steps:

- Train the branch predictor
- Speculatively execute some code
- Exploit the covert channel



TELECOM Paris

IP PARIS

Spectre v1

Most simple variant of Spectre, targets the PHT branch predictor. Victim code with a gadget of the form with x an attacker controlled value:

```
if (x < array1_size)
  y = array2[array1[x]*CACHE_LINE_SIZE];
```

Spectre v1

Gadget(x):

```
if (x < array1_size)
  y = array2[array1[x]*CACHE_LINE_SIZE];
```

Attack:

```
for (int i = 0; i < N_TRAIN; i++)
  gadget(0);
```

State:

Address	array2[0]	array2[...]	...
Cache?	N	N	...

Spectre v1

Gadget(x):

```
    if (x < array1_size) // train BP to take the branch
      y = array2[array1[x]*CACHE_LINE_SIZE];
```

Attack:

```
for (int i = 0; i < N_TRAIN; i++)
  gadget(0);
```

State:

Address	**array2[0]**	**array2[...]**	**...**
Cache?	N	N	**...**

Spectre v1

Gadget(x):

```
    if (x < array1_size) // BP takes the branch speculatively
      y = array2[array1[x]*CACHE_LINE_SIZE];
```

Attack:

```
for (int i = 0; i < N_TRAIN; i++)
  gadget(0);
gadget(&secret - array1);
```

State:

Address	**array2[0]**	**array2[...]**	**...**
Cache?	N	N	**...**

Spectre v1

Gadget(x):

```
    if (x < array1_size)
      y = array2[array1[x]*CACHE_LINE_SIZE]; // spec access
```

Attack:

```
for (int i = 0; i < N_TRAIN; i++)
  gadget(0);
gadget(&secret - array1);
```

State:

Address	**array2[0]**	**array2[...]**	**array2[secret]**	**...**
Cache?	N	N	**Y**	**...**

TELECOM Paris

IP PARIS

Spectre v1

Gadget(x):

```
if (x < array1_size)
  y = array2[array1[x]*CACHE_LINE_SIZE];
```

Attack:

```
for (int i = 0; i < N_TRAIN; i++)
  gadget(0);
gadget(&secret - array1);
// Infer cache state from access time
```

State:

Address	array2[0]	array2[...]	array2[secret]	...
Cache?	N	N	Y	...

A small change can break the attack

BOOM: an open-source Out-of-Order processor core using the RISC-V ISA.

- Spectre attacks were replicated on BOOMv2.

A small change can break the attack

BOOM: an open-source Out-of-Order processor core using the RISC-V ISA.

- Spectre attacks were replicated on BOOMv2.
- *Spectre v1 failed, as it was written, on BOOMv3.*

A small change can break the attack

BOOM: an open-source Out-of-Order processor core using the RISC-V ISA.

- Spectre attacks were replicated on BOOMv2.
- *Spectre v1 failed, as it was written, on BOOMv3.*
- What changed ?

A small change can break the attack

BOOM: an open-source Out-of-Order processor core using the RISC-V ISA.

- Spectre attacks were replicated on BOOMv2.
- *Spectre v1 failed, as it was written, on BOOMv3.*
- What changed ?
- Gshare $\rightarrow$TAGE-**L**

Why did it break ?

Gadget(x):

```
if (x < array1_size)
  y = array2[array1[x]*CACHE_LINE_SIZE];
```

Attack:

```
for (int i = 0; i < N_TRAIN; i++)
  gadget(0);
gadget(&secret - array1);
//[...]
```

Why did it break ?

Gadget(x):

```
if (x < array1_size)
  y = array2[array1[x]*CACHE_LINE_SIZE];
```

Attack:

```
for (int i = 0; i < N_TRAIN; i++)
  gadget(0);
gadget(&secret - array1);
//[...]
```

- Fixed amount of training iterations before attack.

Why did it break ?

Gadget(x):

```
        if (x < array1_size) // taken N_TRAIN times before attack
            y = array2[array1[x]*CACHE_LINE_SIZE];
```

Attack:

```
for (int i = 0; i < N_TRAIN; i++)
  gadget(0);
gadget(&secret - array1);
//[...]
```

- Fixed amount of training iterations before attack.
- After a few iterations of train + attack →loop predictor kicks in.

Why did it break ?

Gadget(x):

```
        if (x < array1_size) // will not be taken
          y = array2[array1[x]*CACHE_LINE_SIZE];
```

Attack:

```
    for (int i = 0; i < N_TRAIN; i++)
      gadget(0);
    gadget(&secret - array1);
    //[...]
```

- Fixed amount of training iterations before attack.
- After a few iterations of train + attack →loop predictor kicks in.
- No mispredict in the attack phase →no data leakage.

TELECOM Paris
IP PARIS

The underlying issue

- Opennness of RISC-V

The underlying issue

- Opennness of RISC-V
- Many different implementations + slight variations in branch predictor designs.

landscape.riscv.org

The underlying issue

- Opennness of RISC-V
- Many different implementations + slight variations in branch predictor designs.

landscape.riscv.org

- Hard to write a single attacking binary that works reliably on all of them.

What would an attacker want ?

An attack that works:

- On many microarchitectures
- With minimal code
- Stealthily

What would an attacker want ?

An attack that works:

- On many microarchitectures
- With minimal code
- Stealthily

Self-adapting attacks:

- No need to know the target's precise microarchitecture beforehand
- No need to manually write the attack for each target

What would an attacker want ?

An attack that works:

- On many microarchitectures
- With minimal code
- Stealthily

Self-adapting attacks:

- No need to know the target's precise microarchitecture beforehand
- No need to manually write the attack for each target

Many possibliities: Neural networks, Q learning, Genetic algorithms, Fuzzing, etc ...

What is a Spectre v1 attack ?

Gadget(x):

```
if (x < array1_size)
  y = array2[array1[x]*CACHE_LINE_SIZE];
```

Attack:

```
for (int i = 0; i < N_TRAIN; i++)
  gadget(0);
gadget(&secret - array1);
// Infer cache state from access time
```

What is a Spectre v1 attack ?

Gadget(x):

```
if (x < array1_size)
  y = array2[array1[x]*CACHE_LINE_SIZE];
```

Attack:

```
for (int i = 0; i < N_TRAIN; i++)
  gadget(0);
gadget(&secret - array1);
// Infer cache state from access time
```

What is a Spectre v1 attack ?

Attack:

```
for (int i = 0; i < N_TRAIN; i++)
  gadget(0);
gadget(&secret - array1);
```

What is a Spectre v1 attack ?

Attack:

```
for (int i = 0; i < N_TRAIN + 1; i++)
  gadget(x[i]);
```

What is a Spectre v1 attack ?

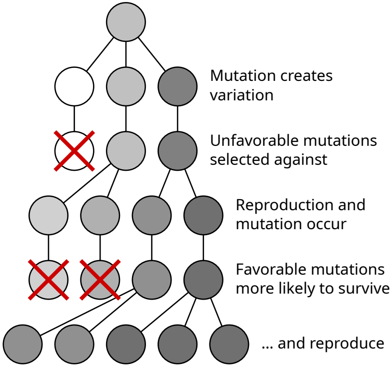
Attack:

```
for (int i = 0; i < N_TRAIN + 1; i++)
    gadget(x[i]);
```

Train + attack sequence: series of calls to the gadget with different arguments

Plan

Genetic Algorithms



- Principles of natural selection to find an optimal solution to a problem
- Fitness function to evaluate the quality of a solution
- Selection, crossover and mutation to generate new solutions

Genetic Algorithms

1. Initialize the population
2. Evaluate the fitness
3. Generate new candidates from the best solutions
4. Repeat steps 2 to 4

- Principles of natural selection to find an optimal solution to a problem
- Fitness function to evaluate the quality of a solution
- Selection, crossover and mutation to generate new solutions

The Genome

Genome

A genome is a set of parameters that will be mutated and selected by the algorithm. They entirely caracterise a solution that expresses a *phenotype*.

Genome of the attack

Each gene corresponds to the attacker controlled value `x` in a call to the gadget.

Parameters	**x[0]**	**x[1]**	**x[2]**	**x[3]**	**x[4]**	**...**
Values	0	4	2	1	ATTACK	...

The Fitness Function

Fitness Function

The fitness function is a function that takes a genome as input and returns a score. The higher the score, the better the genome.

$$F = R_f * C_f + R_s * C_s + T + B$$

Our fitness function takes into account:

- How much the secret was leaked
- The number of calls to the gadget

Our Genetic Algorithm

1. Initialize the population: null genome
2. Evaluate the fitness
3. Keep the best solutions
4. Generate new candidates:
   4.1 Randomly swap the parameters of the 2 best solutions (*crossing over*)
   4.2 Mutating and randomly swapping two consecutive parameters
5. Repeat steps 2 to 5

Plan

TELECOM Paris

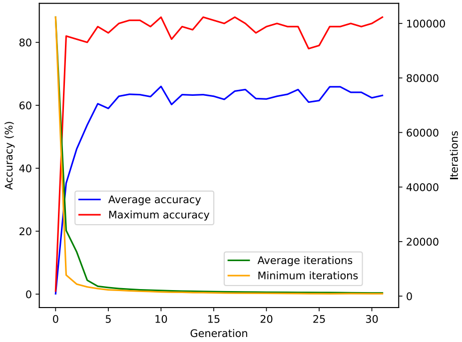
IP PARIS

Setup

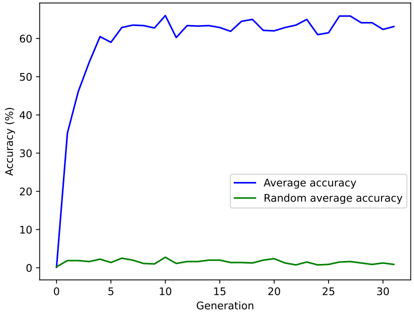
- 1 BOOMv3 core
- Synthesized on a Nexys Video board @100MHz
- Alpha21264 and TAGE-L branch predictors

Results

Accuracy: percentage of secret leaked



TAGE-L



Comparison with random on TAGE-L

Plan

Conclusion

- Able to find train + attack sequences
- 2 different branch predictors with a single binary
- works on TAGE-L without specific code

Future works

- Test on more branch predictors
- Try other genetic algorithms or other self-adapting approaches
- Full adaptation to the target (cache, etc.)

Thank you for your attention!

Any questions?