PTCC Forward – 31/03/2026

Introduction to SAMVA

Antoine Gicquel - Sébastien Michelland

RÉPUBLIQUE FRANÇAISE
Liberté Égalité Fraternité

Inria

UMR IRISA

Université de Rennes

Outline

1. Context
2. Introduction to SAMVA
3. Use case: CHAPATI

4. How it works - Algorithms
5. New features

Fault injection attacks evolution

Multi-fault : multiple fault injections during the same attack.

- Allows the disturbance to **persist over time**
  - Laser : multiple spots ; 300 instructions in a single fault [DRP+19]
  - Clock glitch : sequence of 20 instructions [CPH+21]
- Improved **spatial** and **temporal** precision

[DRP+19] Dutertre, Riom, Potin, Rigaud « Experimental analysis of the laser-induced instruction skip fault model » *Secure IT Systems: 24th Nordic Conference, NordSec 2019*

[CPH+21] Claudepierre, Péneau, Hardy, Rohou « TRAITOR: a low-cost evaluation platform for multifault injection » *Proceedings of the 2021 International Symposium on Advanced Security on Software and Systems*

Fault injection attacks evolution

Multi-fault : multiple fault injections during the same attack.

- Allows the disturbance to **persist over time**
  - Laser : multiple spots ; 300 instructions in a single fault [DRP+19]
  - Clock glitch : sequence of 20 instructions [CPH+21]
- Improved **spatial** and **temporal** precision

Software countermeasures can be **compromised.** 

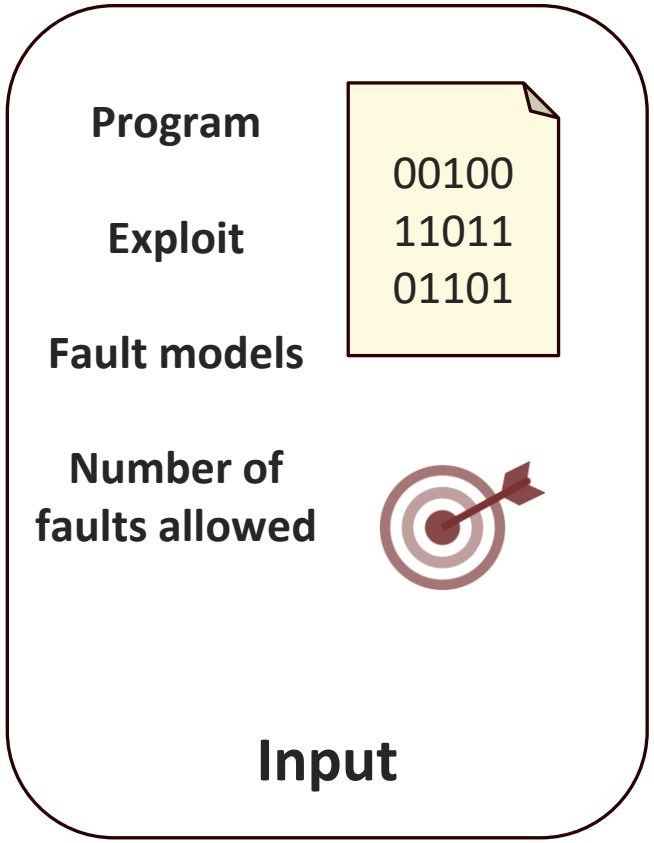
- Code and data redundancy
- Control flow integrity
- Specific data encoding

→ **Attacks require more code to be faulted** 

[DRP+19] Dutertre, Riom, Potin, Rigaud « Experimental analysis of the laser-induced instruction skip fault model » *Secure IT Systems: 24th Nordic Conference, NordSec 2019*

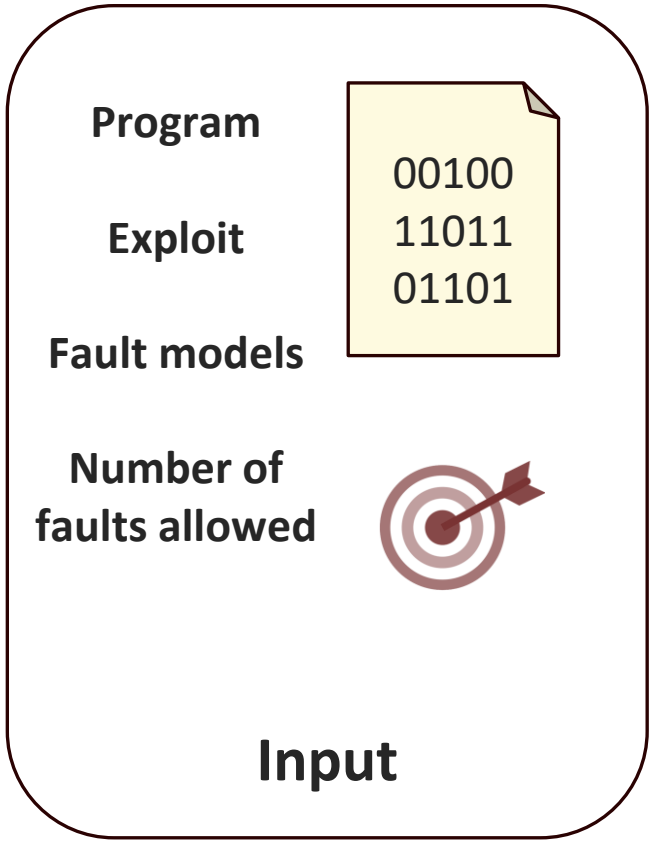
[CPH+21] Claudepierre, Péneau, Hardy, Rohou « TRAITOR: a low-cost evaluation platform for multifault injection » *Proceedings of the 2021 International Symposium on Advanced Security on Software and Systems*

Program analysis



Assess security without conducting **fault injection campaign**.

Program analysis

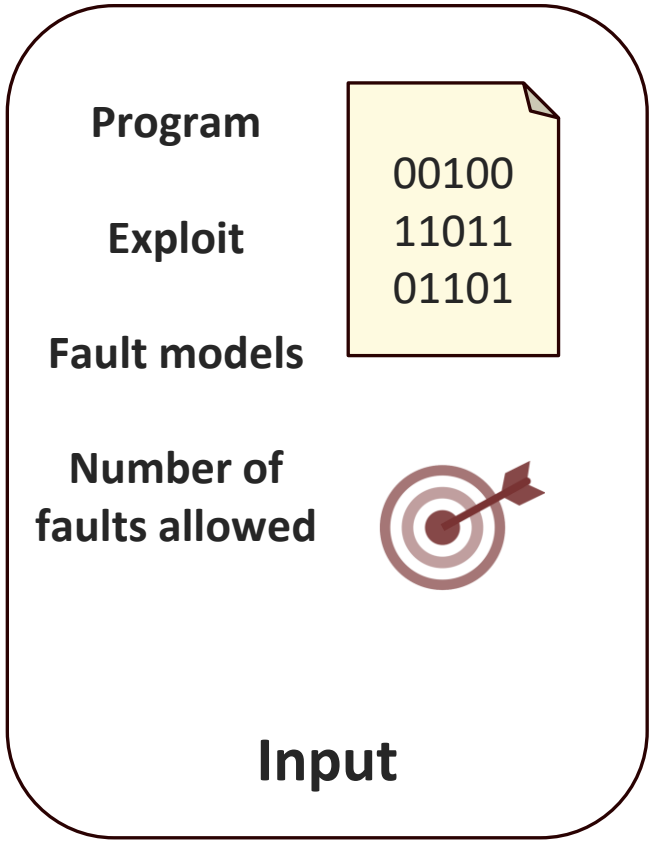


Assess security without conducting **fault injection campaign**.

Considering

- Fault models retrieved from characterization → **abstraction**
- Exploit depends on the program targeted
- Number of faults depends on the injection techniques

Program analysis



Assess security without conducting **fault injection campaign**.

Considering

- Fault models retrieved from characterization → **abstraction**
- Exploit depends on the program targeted
- Number of faults depends on the injection techniques

Also helpful for **countermeasures** designers.

Program analysis

Program

Exploit

Fault models

00100
11011
01101

Number of faults allowed

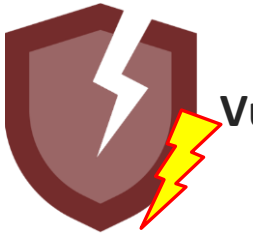
Input

Analysis

Simulation
Concrete values + Observation

Symbolic execution
Symbolic values + solver

Model checking
Formal model + Verification



Vulnerabilities

Formal proofs

Output

Difficulties related to multi-fault

Simulation: Impossible to be exhaustive

Symbolic execution: Forking / Solver

Model verification: Solver

Difficulties related to multi-fault

Simulation: Impossible to be exhaustive

Symbolic execution: Forking / Solver

Model verification: Solver

Scaling is a challenge

- Number of faults / code size
- Width of faults

Difficulties related to multi-fault

Simulation: Impossible to be exhaustive

Symbolic execution: Forking / Solver

Model verification: Solver

Scaling is a challenge

- Number of faults / code size
- Width of faults

Here it comes... SAMVA

- **Antoine Gicquel PhD** ; **Sébastien Michelland Post-doc**
- Determines attack paths using **static analysis**
- Focus on **scalability** – numerous faults
- Supports **Arm** and **RISC-V**
- Written in Python, now open source!

Attacker threat considered

Accessibility exploit: reach several code locations while avoiding others,
e.g. granting authentication without getting detected.

Currently, **2 fault models** considered by SAMVA.

Attacker threat considered

Accessibility exploit: reach several code locations while avoiding others,
e.g. granting authentication without getting detected.

Currently, **2 fault models** considered by SAMVA.

```
c.addi sp, sp, -32
c.sw ra, 28(sp)
c.sw s0, 24(sp)
c.addi s0, sp, 32
lui a5, 0x20001
```

Instruction skip

Attacker threat considered

Accessibility exploit: reach several code locations while avoiding others,
e.g. granting authentication without getting detected.

Currently, **2 fault models** considered by SAMVA.

1 fault
width = 3

```
c.addi sp, sp, -32
c.sw ra, 28(sp)
c.sw s0, 24(sp)
c.addi s0, sp, 32
lui a5, 0x20001
```

Instruction skip

Attacker threat considered

Accessibility exploit: reach several code locations while avoiding others,
e.g. granting authentication without getting detected.

Currently, **2 fault models** considered by SAMVA.

1 fault
width = 3

```
c.addi sp, sp, -32
c.sw ra, 28(sp)
c.sw s0, 24(sp)
c.addi s0, sp, 32
lui a5, 0x20001
```

Instruction skip

```
c.addi sp, sp, -32
c.sw ra, 28(sp)
c.sw s0, 24(sp)
c.addi s0, sp, 32
lui a5, 0x20001
```

Word Replay

Attacker threat considered

Accessibility exploit: reach several code locations while avoiding others,
e.g. granting authentication without getting detected.

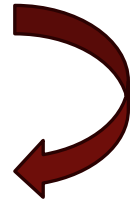
Currently, **2 fault models** considered by SAMVA.

1 fault
width = 3

```
c.addi sp, sp, -32
c.sw ra, 28(sp)
c.sw s0, 24(sp)
c.addi s0, sp, 32
lui a5, 0x20001
```

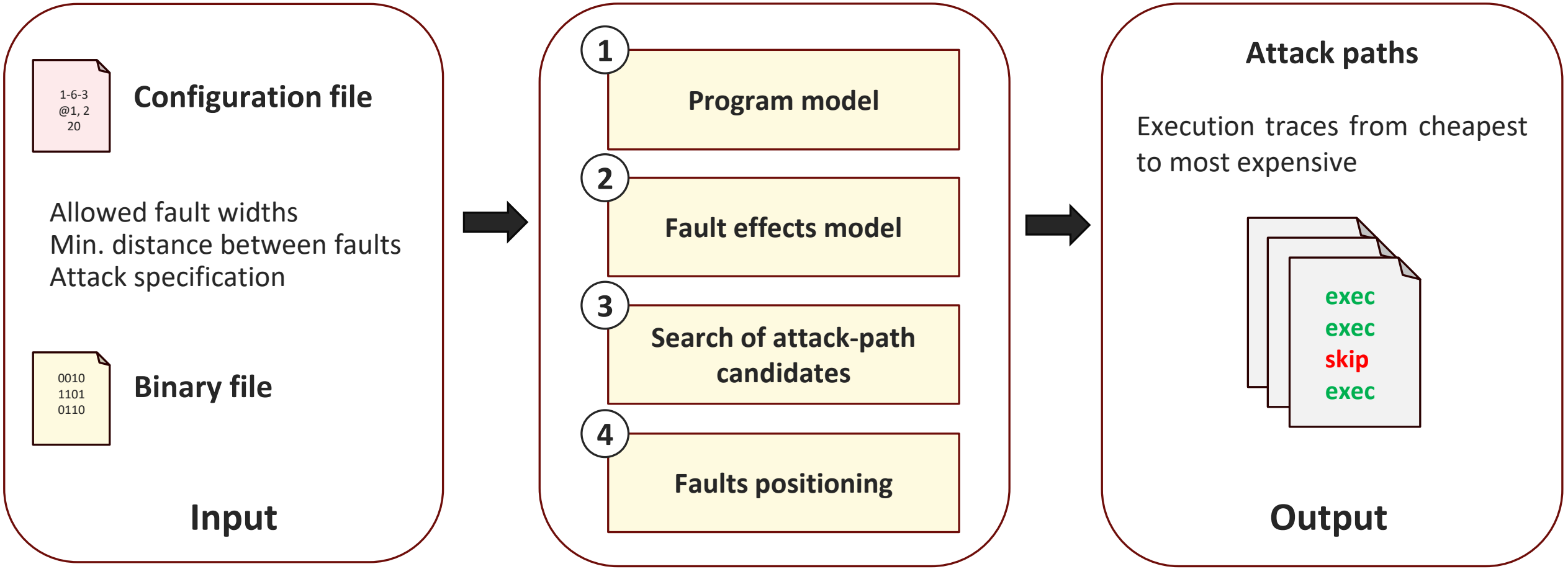
Instruction skip

```
c.addi sp, sp, -32
c.sw ra, 28(sp)
c.sw s0, 24(sp)
c.sw ra, 28(sp)
c.sw s0, 24(sp)
```



Word Replay

SAMVA Overview



1 Program model

Raw binary cannot be directly used by analysis → Require a **front-end**

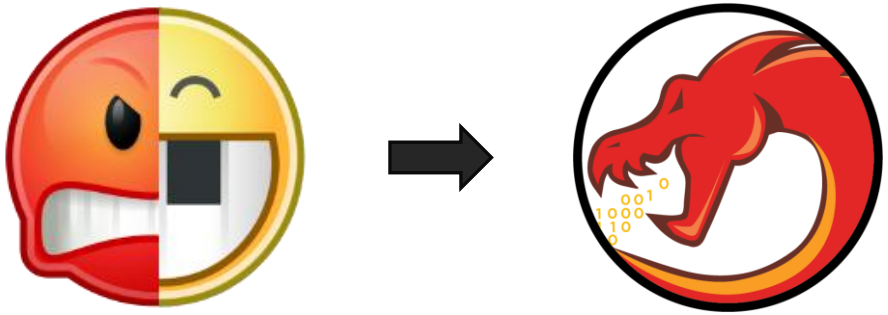
- Disassemble the binary into instructions
- Analysis or reverse engineer some program properties
- Build intermediate representation (IR), in order to be architecture independent

1 Program model

Raw binary cannot be directly used by analysis → Require a **front-end**

- Disassemble the binary into instructions
- Analysis or reverse engineer some program properties
- Build intermediate representation (IR), in order to be architecture independent

In the context of FORWARD, SAMVA switched from **angr** to **Ghidra**.



Supports **more architecture**, including RISC-V
Better **stability** and **support**

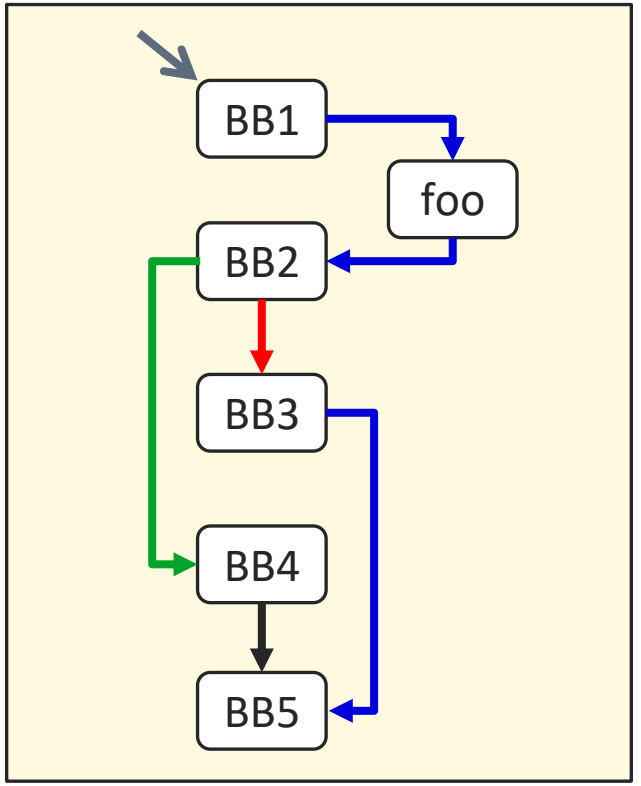
2 Fault effects model

- Build **Python objects** from PyGhidra and retrieve **instruction IR.**

2 Fault effects model

- Build **Python objects** from PyGhidra and retrieve **instruction IR.**

- Generate an **initial control flow graph (CFG).**
  - Node = basic block, i.e., straight-line sequence of code with only one entry point and only one exit
  - Edge = control flow between basic blocks, i.e., un·conditional branches



Example of initial CFG

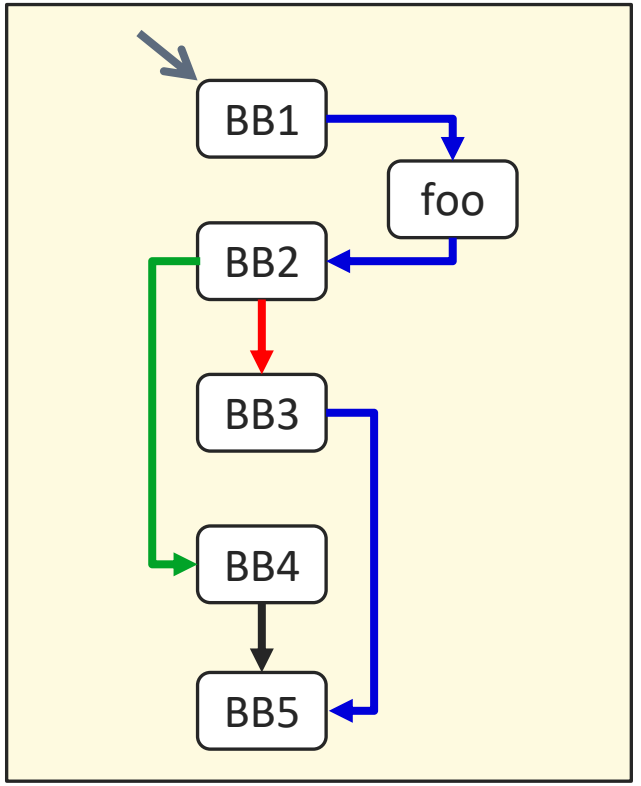
2 Fault effects model

- Build **Python objects** from PyGhidra and retrieve **instruction IR.**

- Generate an **initial control flow graph (CFG).**
  - Node = basic block, i.e., straight-line sequence of code with only one entry point and only one exit
  - Edge = control flow between basic blocks, i.e., un·conditional branches

Afterward, **SAMVA runs passes** (in similar way as a compiler) by

- Adding or removing **edges** of the CFG
- Adding **attributes** on the edges of the CFG
- Adding **information** on instructions, words, basic blocks... All Python objects

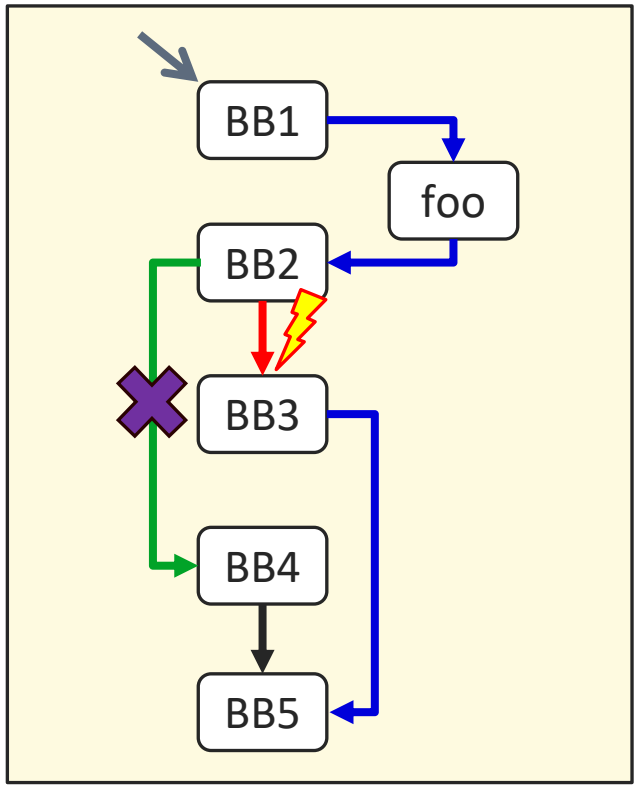


Example of initial CFG

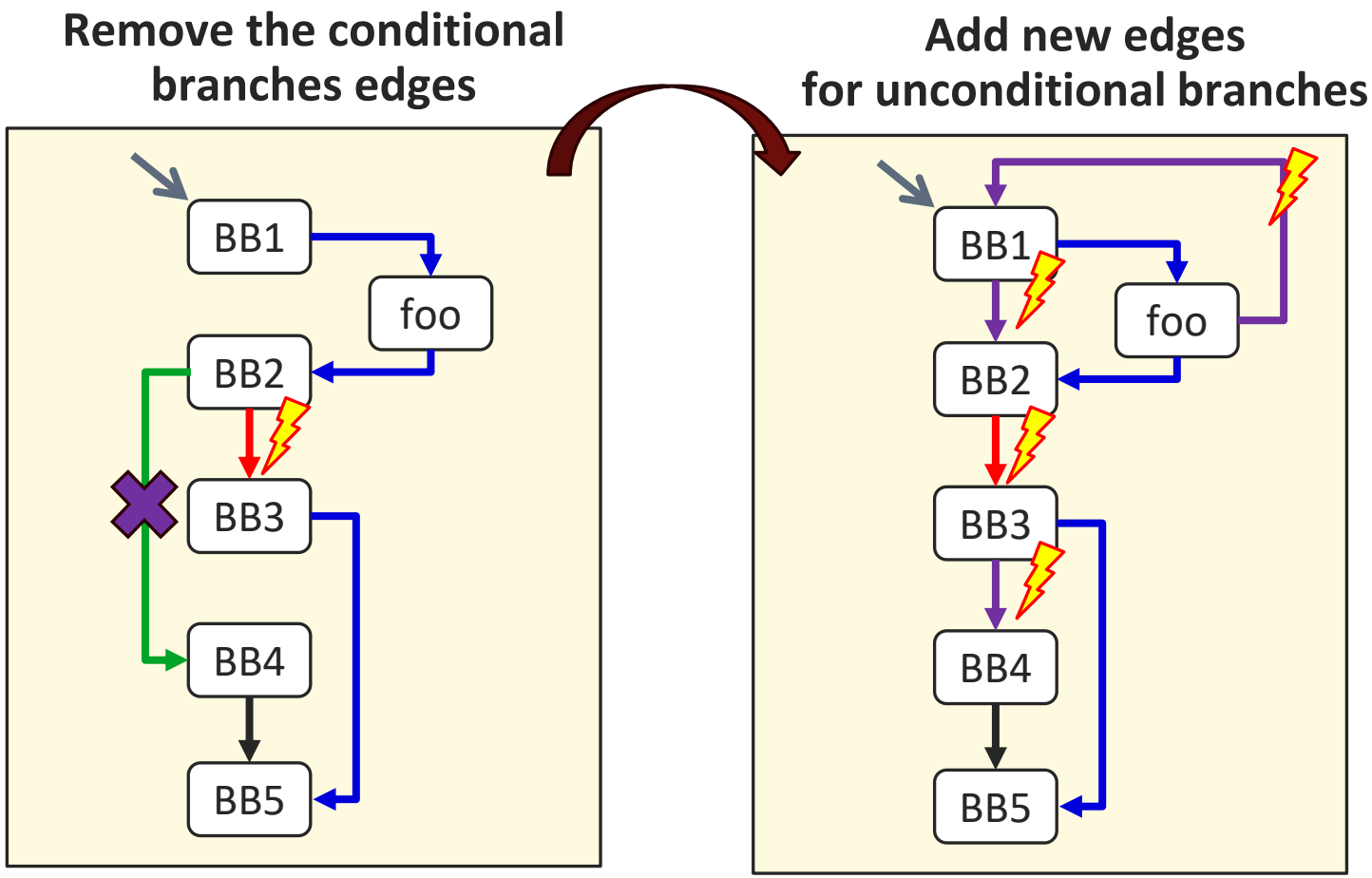
Example of the instruction skip fault model

Example of the instruction skip fault model

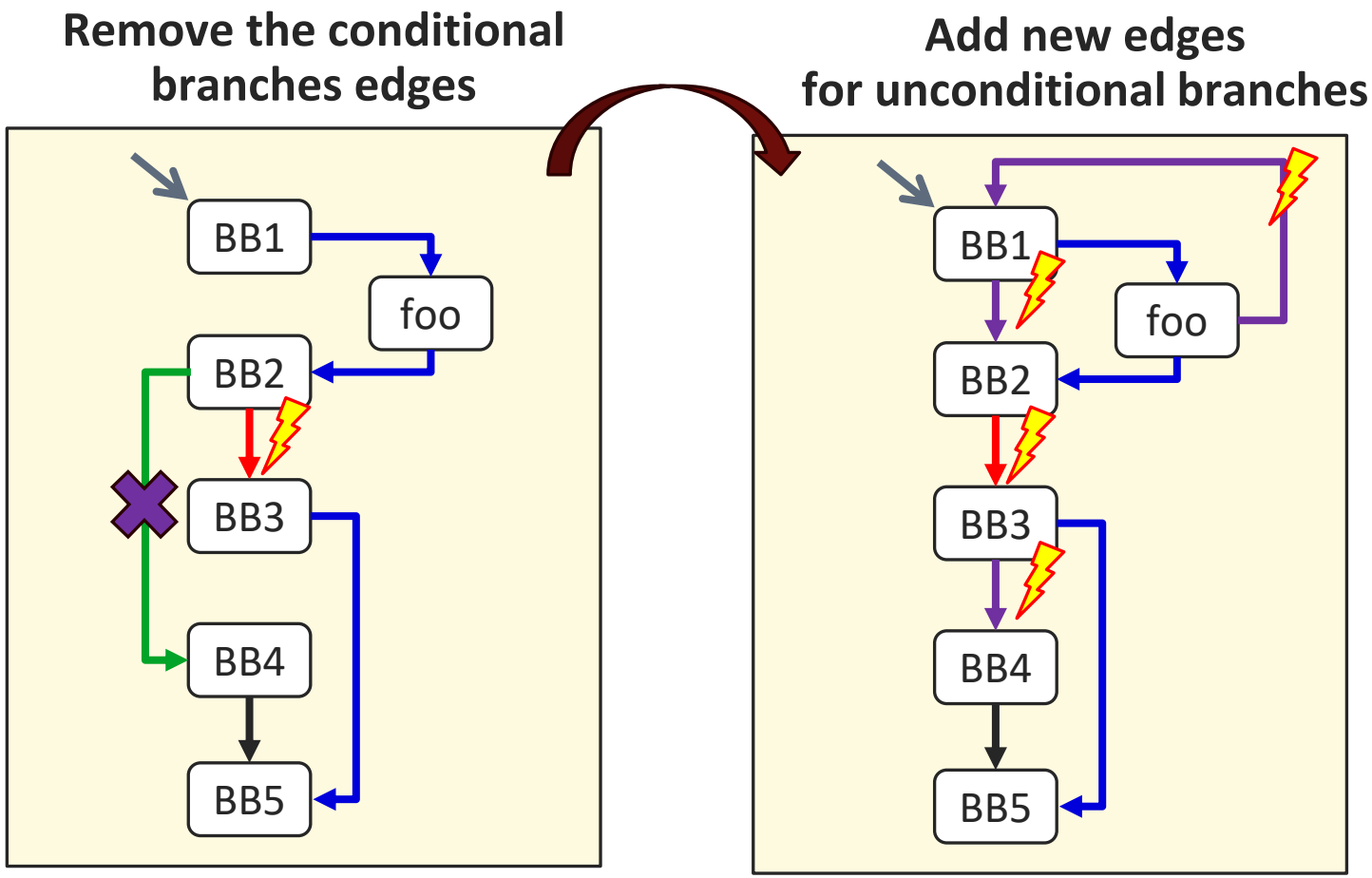
Remove the conditional branches edges



Example of the instruction skip fault model



Example of the instruction skip fault model

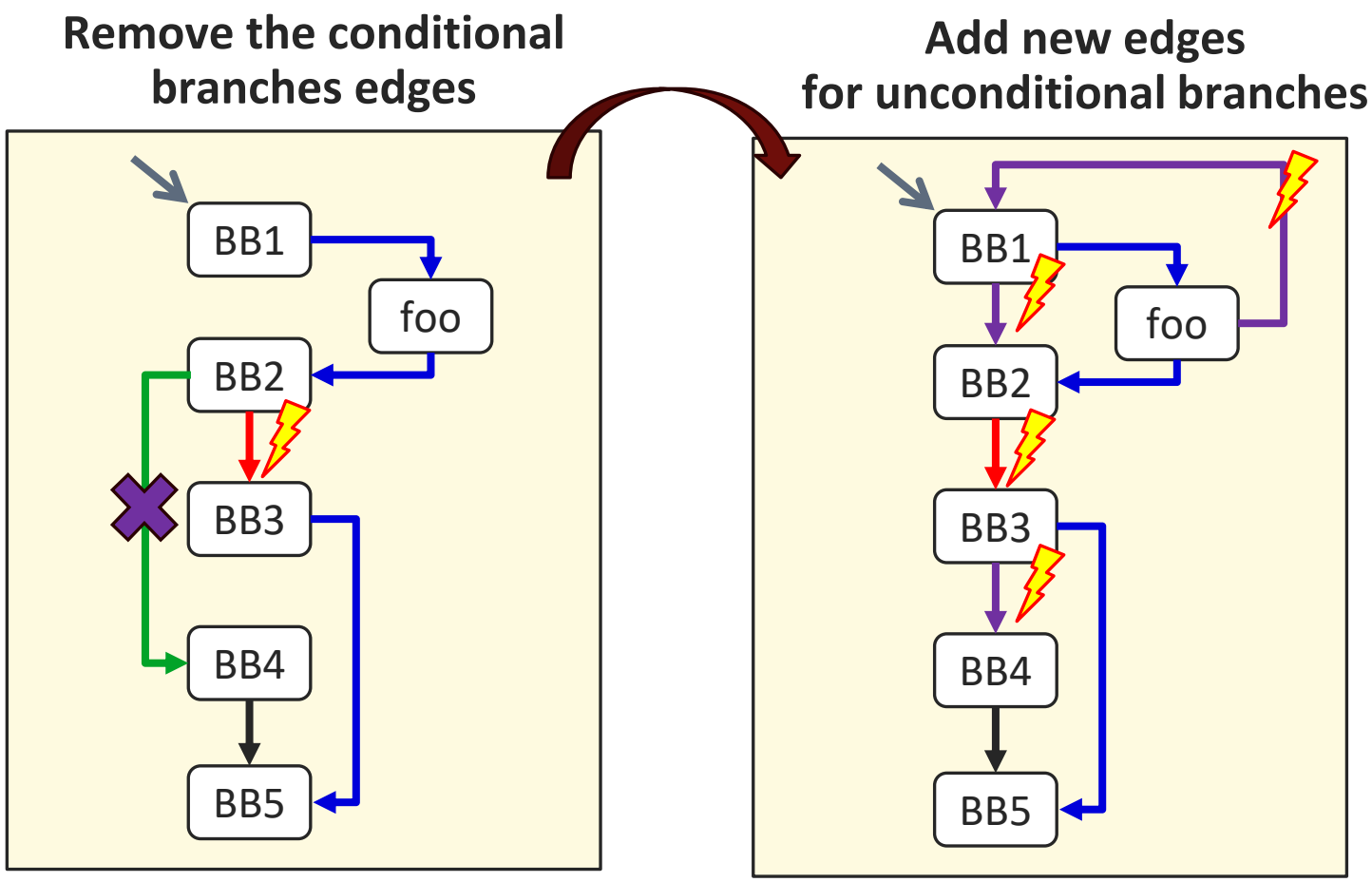


Add attributes on CFG edges.

Type per instruction

- **execute**
- **skip**
- **neutral**

Example of the instruction skip fault model



Add attributes on CFG edges.

Type per instruction

- **execute**
- **skip**
- **neutral**

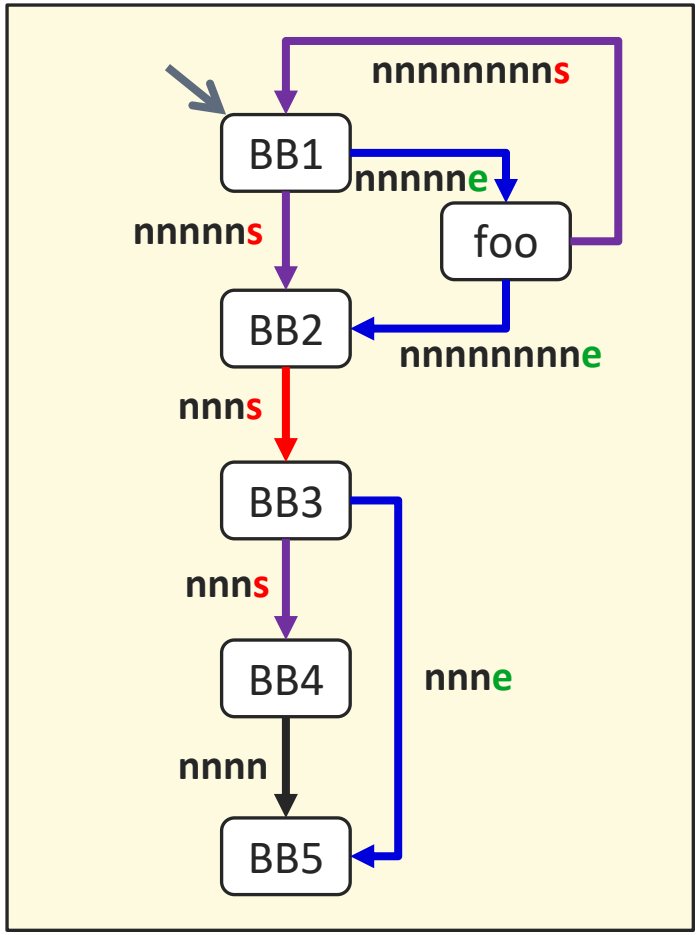
neutral is useful later to position the faults.

Additional typing rules can be executed (conditional passes).

3 Search of attack-path candidates (updated!)

Graph exploration algorithm

- Cheapest paths first
- Cost evaluation: fault positioning algorithm (Sébastien)

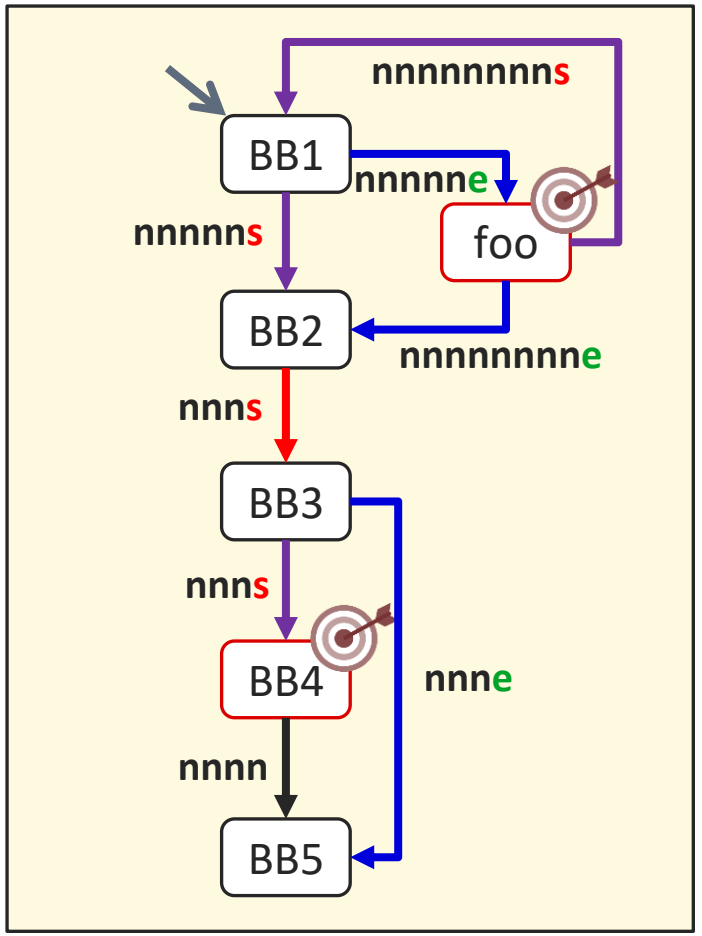


3 Search of attack-path candidates (updated!)

Graph exploration algorithm

- Cheapest paths first
- Cost evaluation: fault positioning algorithm (Sébastien)

Depends on the exploit, e.g. targets = [foo; B4]



3 Search of attack-path candidates (updated!)

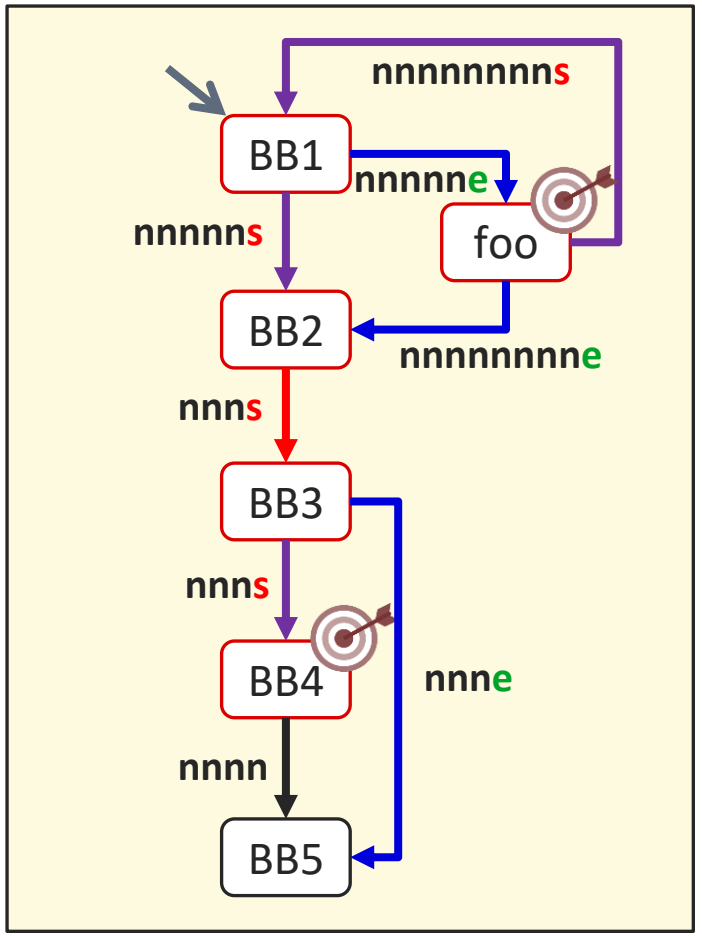
Graph exploration algorithm

- Cheapest paths first
- Cost evaluation: fault positioning algorithm (Sébastien)

Depends on the exploit, e.g. targets = [foo; B4]

Trace generation <address, instruction type>

- B1-foo + foo-B2 + B2-B3 + B3-B4
- **nnnnne + nnnnnnnne + nnnns + nnns**



3 Search of attack-path candidates (updated!)

Graph exploration algorithm

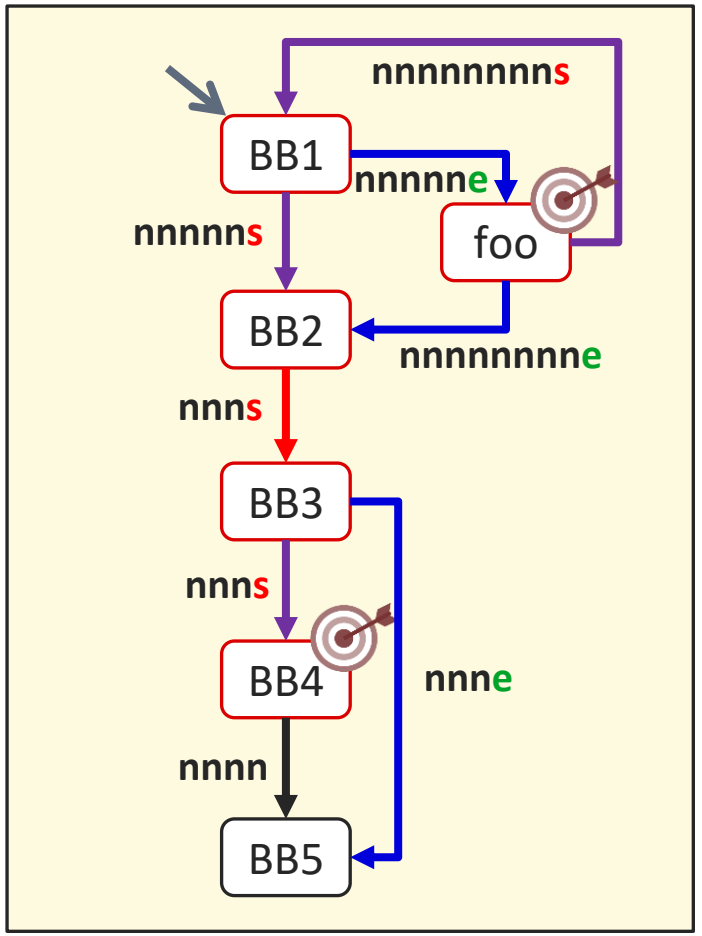
- Cheapest paths first
- Cost evaluation: fault positioning algorithm (Sébastien)

Depends on the exploit, e.g. targets = [foo; B4]

Trace generation <address, instruction type>

- B1-foo + foo-B2 + B2-B3 + B3-B4
- **nnnnne + nnnnnnnne + nnnns + nnns**

→ Possible according to attacker capacities?



4 Fault positioning (updated!)

```
0x104F8: execute
0x104FC: neutral
0x10500: neutral
0x10504: skip
0x10508: skip
0x1050C: neutral
0x10510: execute
0x10514: neutral
0x10518: neutral
0x1051C: skip
0x10520: skip
0x10524: neutral
0x10528: neutral
0x1052C: skip
0x10530: neutral
0x10534: neutral
0x10538: neutral
```

Conditions for fault positioning

- All instructions typed "**skip**" are covered by a fault
- No instructions typed "**execute**" are covered by a fault
- Fault width and distance between faults

Algorithm returns a list of fault sequences for a single candidate.
e.g. of a valid sequence: [(pos = 3, width = 3), (pos = 10, w = 5)]

4 Fault positioning (updated!)

```
0x104F8: execute
0x104FC: neutral
0x10500: neutral
0x10504: skip
0x10508: skip
0x1050C: neutral
0x10510: execute
0x10514: neutral
0x10518: neutral
0x1051C: skip
0x10520: skip
0x10524: neutral
0x10528: neutral
0x1052C: skip
0x10530: neutral
0x10534: neutral
0x10538: neutral
```

Conditions for fault positioning

- All instructions typed "**skip**" are covered by a fault
- No instructions typed "**execute**" are covered by a fault
- Fault width and distance between faults

Algorithm returns a list of fault sequences for a single candidate.
e.g. of a valid sequence: [(pos = 3, width = 3), (pos = 10, w = 5)]

Can be validated using **simulation** with **Unicorn engine**.
Faults are simulated by PC manipulation.

Use case: CHAPATI

Difficulties to conduct physical fault injection attacks

- Lots of parameters to test: delay, duration, intensity, position of the fault
- **Multi-fault** attack: impossible exhaustiveness + stochastic effects

Use case: CHAPATI

Difficulties to conduct physical fault injection attacks

- Lots of parameters to test: delay, duration, intensity, position of the fault
- **Multi-fault** attack: impossible exhaustiveness + stochastic effects

Program analysis → Enable an evaluator to find the instructions to fault, but...

Use case: CHAPATI

Difficulties to conduct physical fault injection attacks

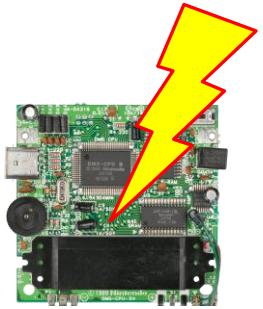
- Lots of parameters to test: delay, duration, intensity, position of the fault
- **Multi-fault** attack: impossible exhaustiveness + stochastic effects

Program analysis → Enable an evaluator to find the instructions to fault, but...

How to conduct an attack from an attack path?

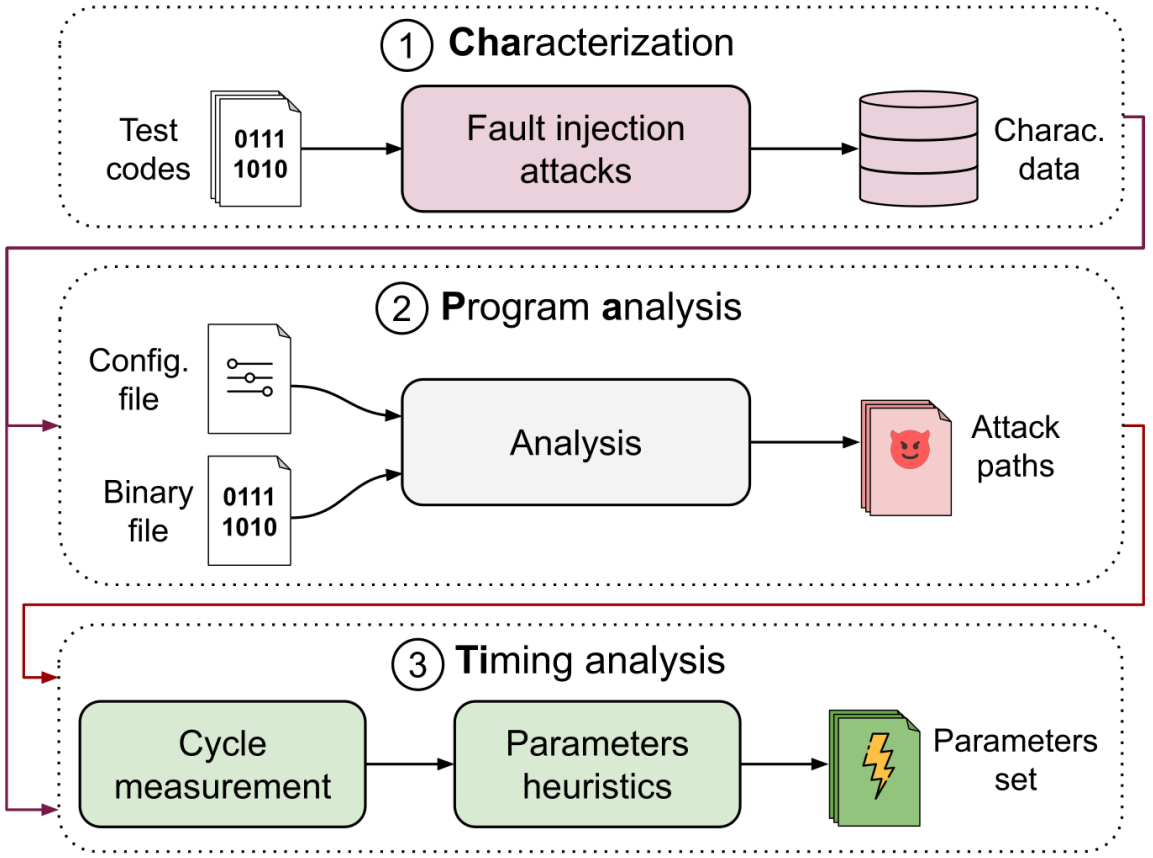
```
skip instruction x
skip instruction y
```


```
delay = 58 cycles
duration = 4 cycles
intensity = 0.7V
…
```



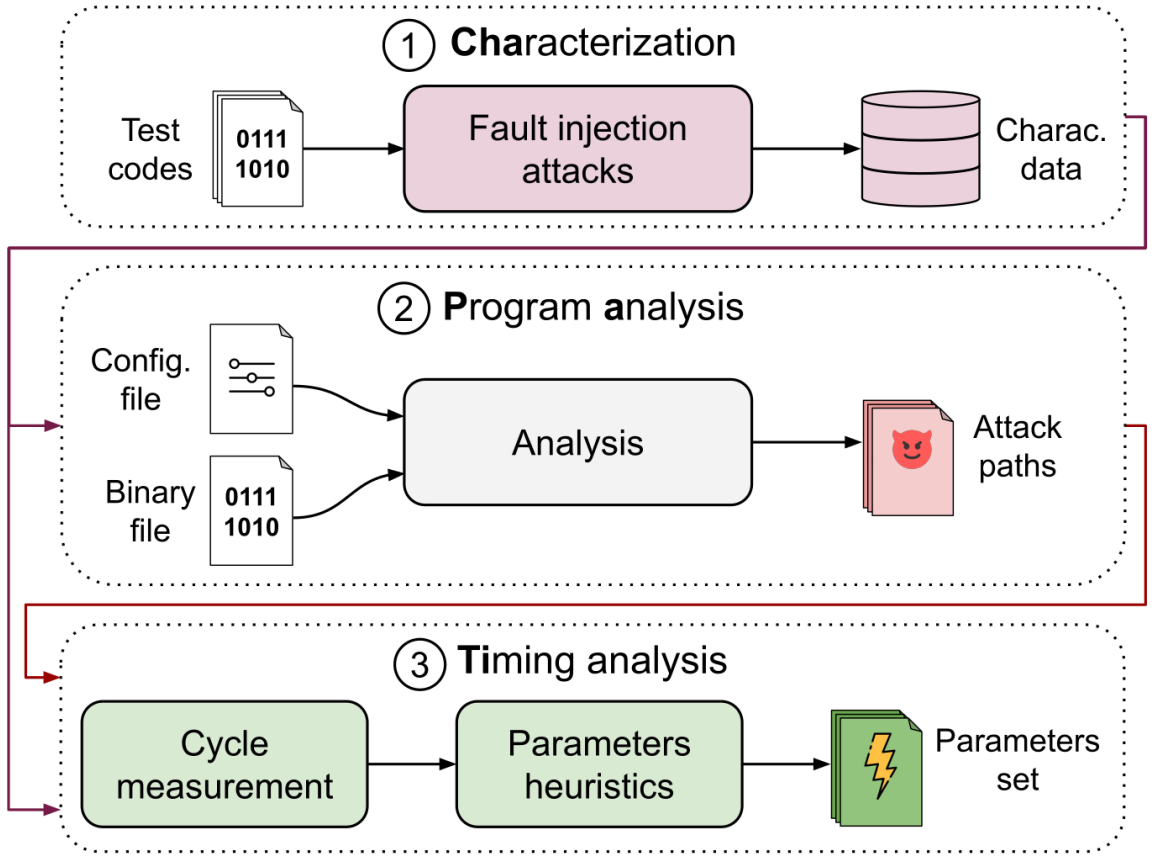
It is necessary to convert attack paths into physical parameters

Use case: CHAPATI



Methodology for facilitating multi-fault injection campaigns

Use case: CHAPATI

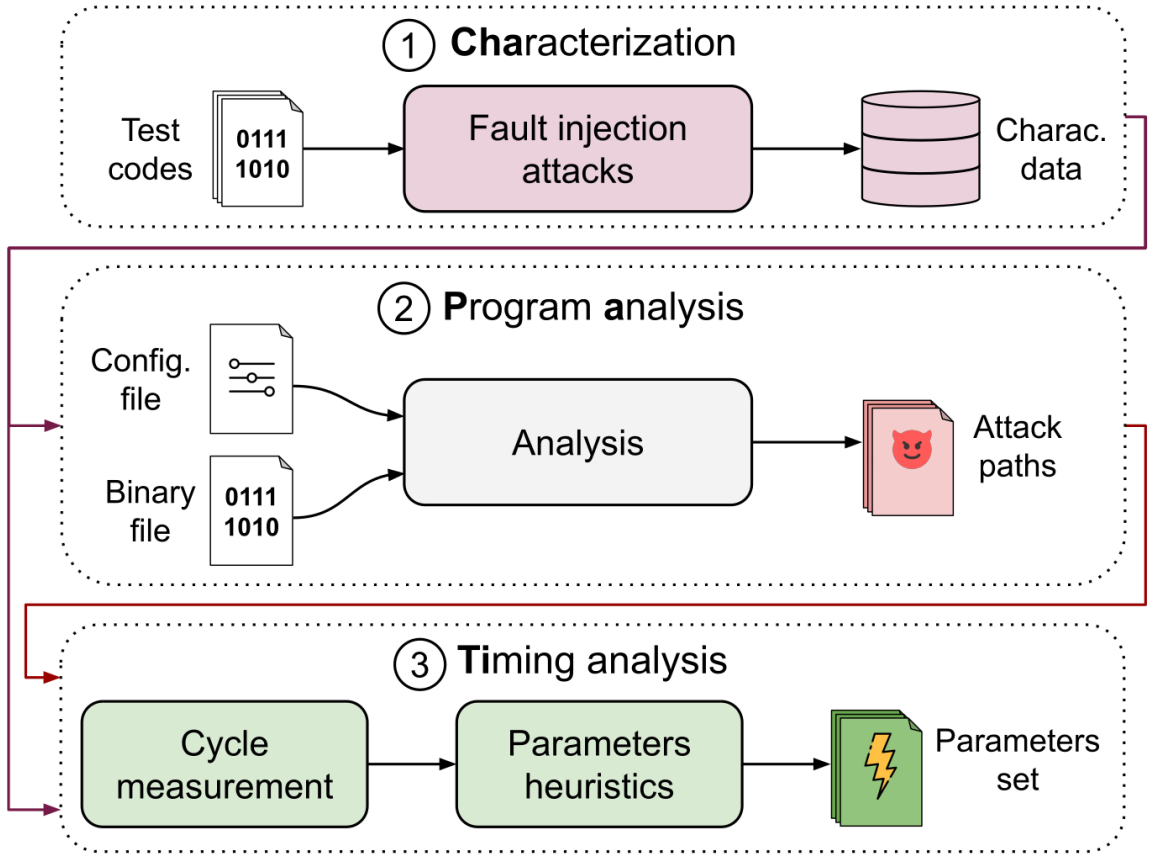


Methodology for facilitating multi-fault injection campaigns

Clock glitch experiments: Managed up to **8 faults** affecting more than **80 instructions**

Will be published at **HOST 2026** this year, let's meet there!

Use case: CHAPATI



Methodology for facilitating multi-fault injection campaigns

Clock glitch experiments: Managed up to **8 faults** affecting more than **80 instructions**

Will be published at **HOST 2026** this year, let's meet there!