



Efficient Verification of Fault Injection Attack Protections at Binary Level

Yanis Sellami

Noémie Bénard

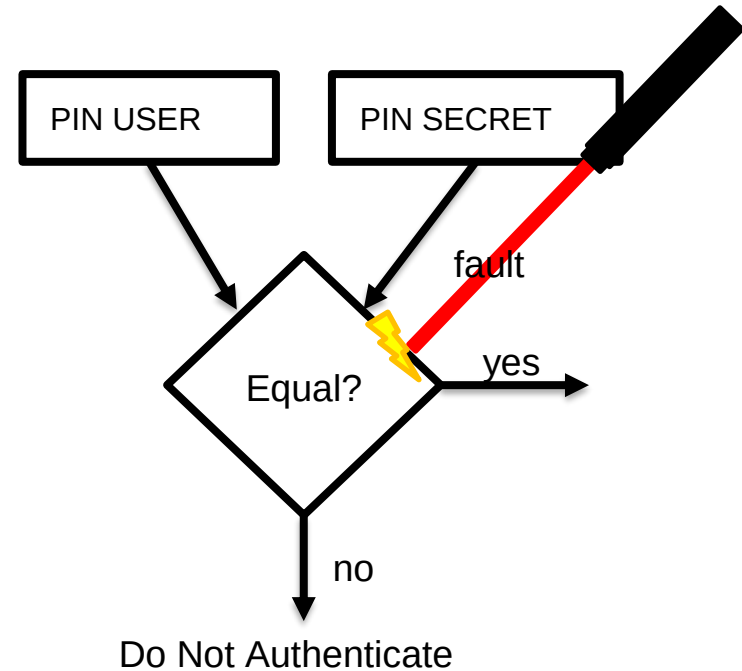
Frédéric Recoules

Sébastien Bardin

Université Paris-Saclay, CEA List Palaiseau, France

Fault Injection Attacks

- Create a physical perturbation on the hardware (power, clock, laser, em, ...)
- Impacts the execution of the software
- Introduce a vulnerability or break the security





Software Countermeasures Against Fault Injection

- It is critical to have protection against fault injection attacks
- There are multiple ways to protect against fault injection attacks
- Hardware protection: prevent the attacker's faults from having an impact
Optimal efficiency but requires specialized hardware

In This Talk

- Software protection: protect the executed code directly (source protection, secure compilation, ...)
Can be applied on any hardware but can be costly
- Hardware/software protection: mitigate the software cost with a limited hardware protection



Verifying Countermeasures

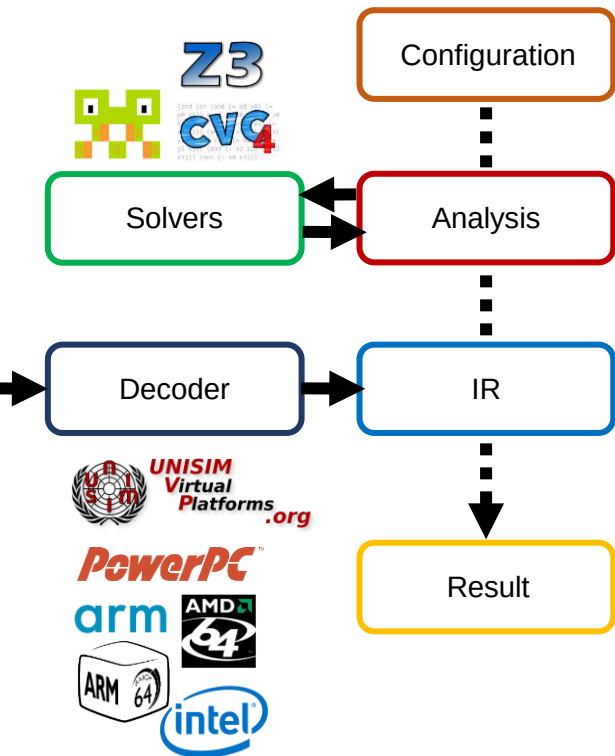
- General Idea: Verify that an countermeasure is correctly applied and generates secure code
- Countermeasures applied on the software → We need to analyze the software
- Secure Compilation → We need to analyze the software **after** compilation
- A Countermeasure protects against a **fault model** → We need to represent these fault models



Fault Models at Binary Level

- Abstraction to model the effect of a fault in the execution of a program (ISA level)
- Example of fault models:
Value injection, Bit flip, Bit set, Bit reset, Test inversion, Instruction skip
- Transient faults (once in the execution trace) or permanent faults (lasting recurrent effect)
- Where we are allowed to fault (registers, memory, code, ...)
- Single fault or multi fault

BINSEC for Fault Injection Analysis



Since 2012

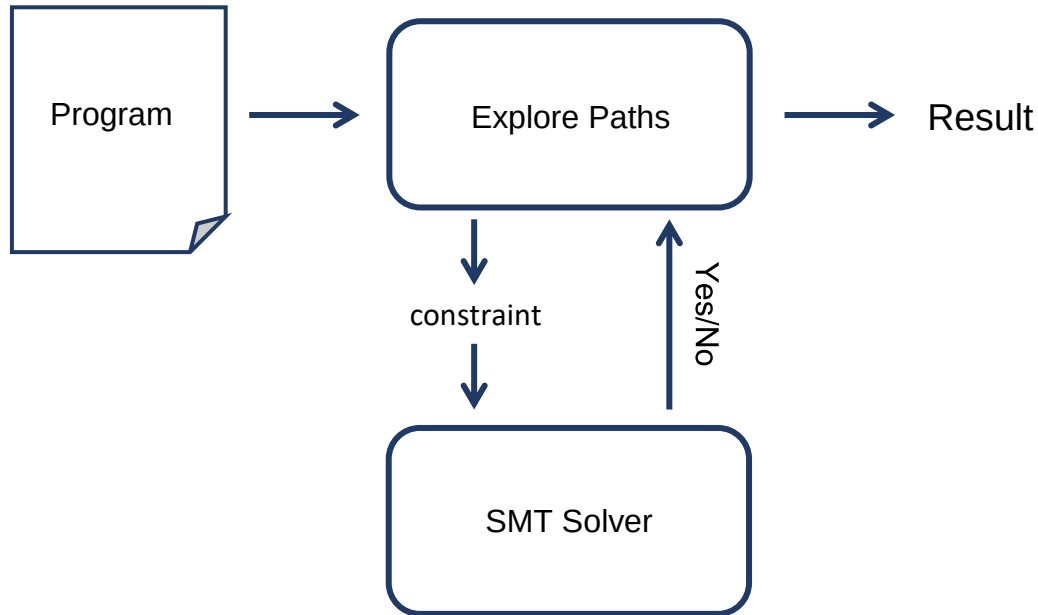
 **BINSEC**

A Unified Plugin for
Fault Injection Analysis



Preview Demo

Symbolic Execution

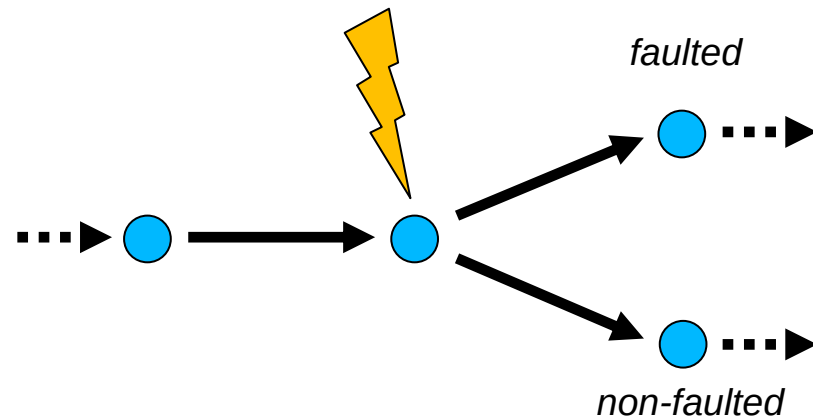


Summary :

- **Build logic constraints for each program path**
- **Solve constraints to find program inputs**
- **Under-approximation: correct but incomplete (maybe k-complete)**

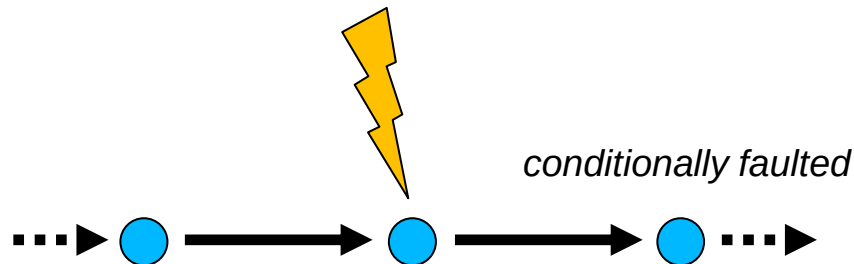
Modeling Faults in Symbolic Execution

- Working on the intermediate representation
- For each possible fault effect
- We must consider the faulted case and the non faulted case
- Exponential increase in the number of paths to consider
- We need to do better (especially for multi fault)

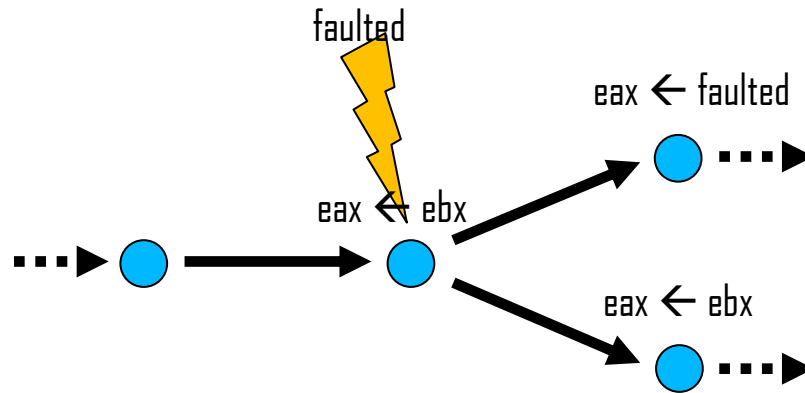


Adversarial Reachability [Ducousso et al. 2023]

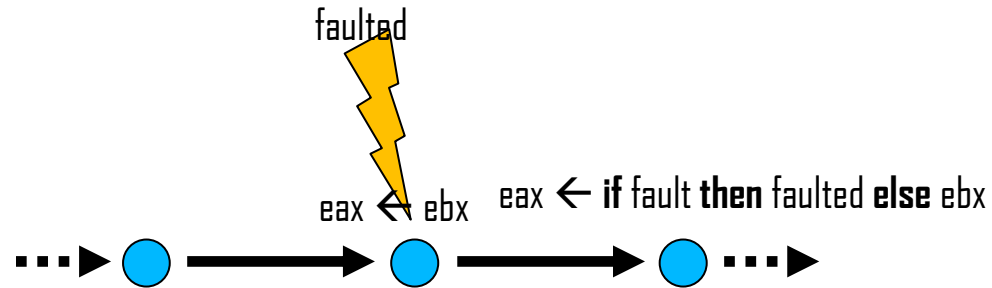
- For each possible fault effect
- Do not fork the path
- Represent the fault effect with a conditional expression instead
- Trade-off: more complex constraints but no additional path



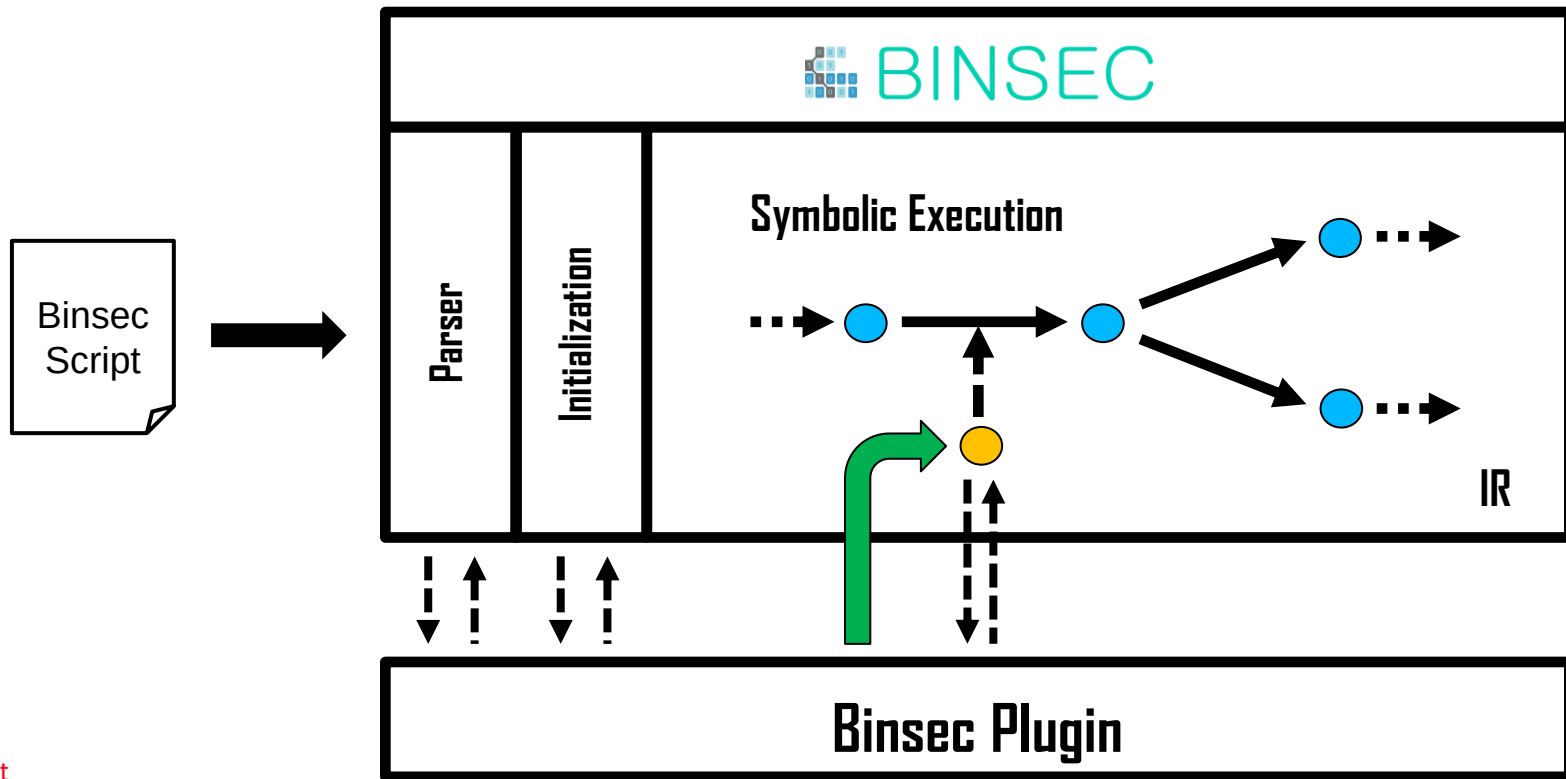
Example: Forking Fault Injection



Example: Forkless Fault Injection



BINSEC Plugins



IR Faults: Bit Flips

- Flip one bit of a given value
- In practice: more general model of mask flip
- Construct the faulted value by letting the solver chose how much to flip the mask

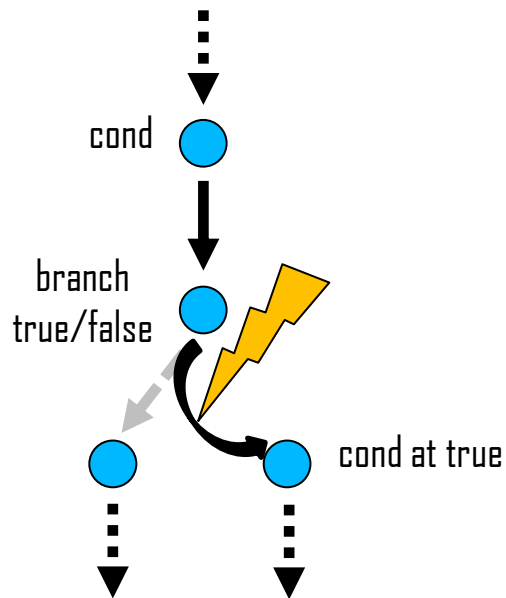
$$\text{faulted} \leftarrow \text{mask} \ll \text{faulted_index}$$
$$\text{mask} = 0b1$$



IR Faults: Transient Faults, Persistent Faults

- Transient faults: change the value when it is read, but the following reads are unaffected
- Persistent faults: change the value stored when it is read → the following reads also return the faulted values

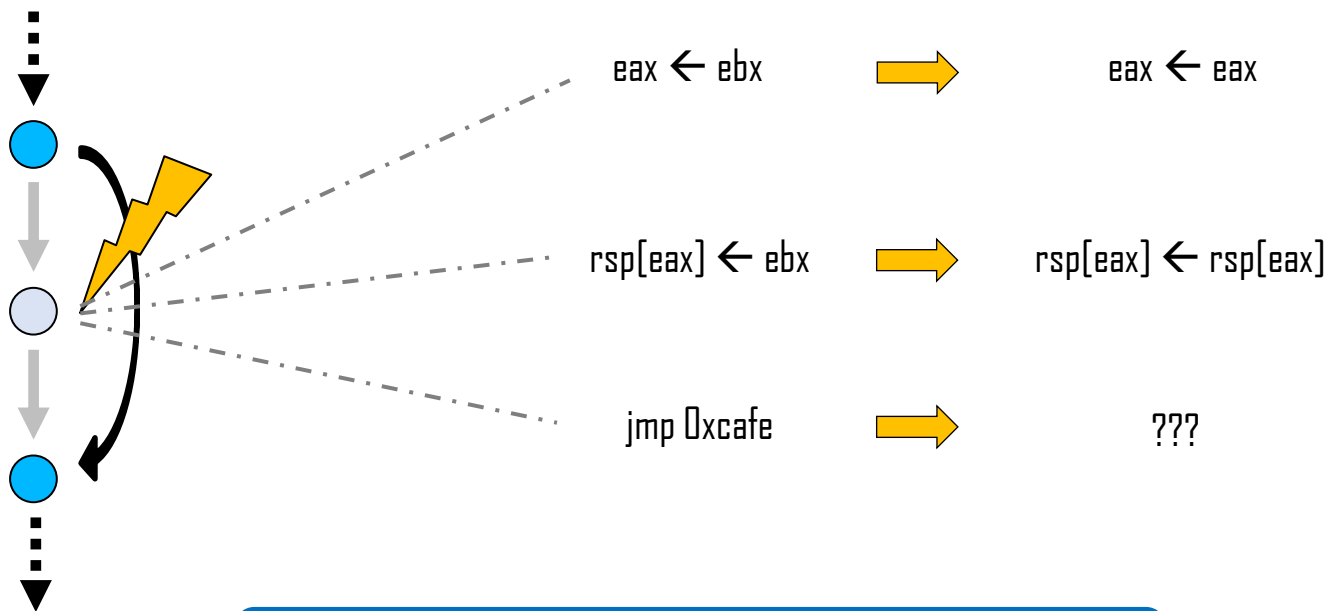
IR Faults Problems: Test Inversion



- Inverting the test with a value fault can be done by changing the value of the condition
- But if we reuse this value later, we model more than the expected effect of the fault

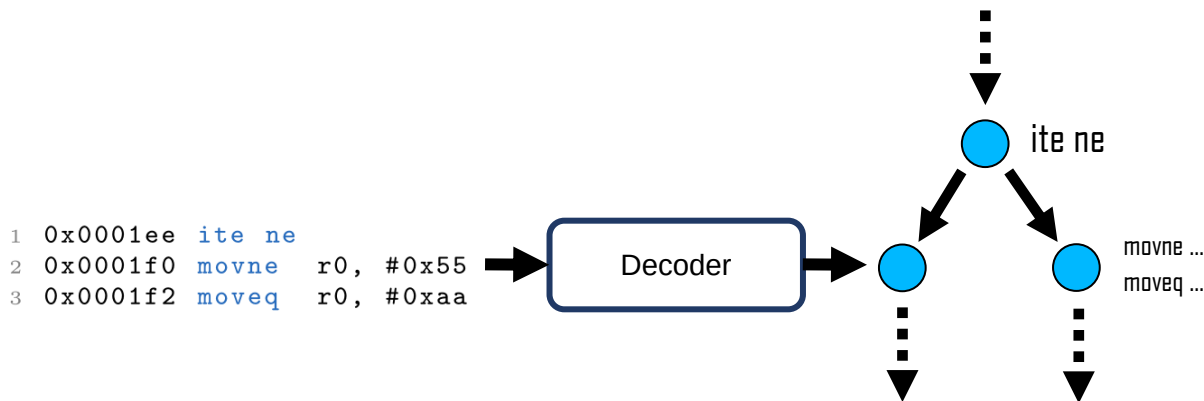
Modelling test inversion with value faults has side effects

IR Faults Problems: Instruction Skip



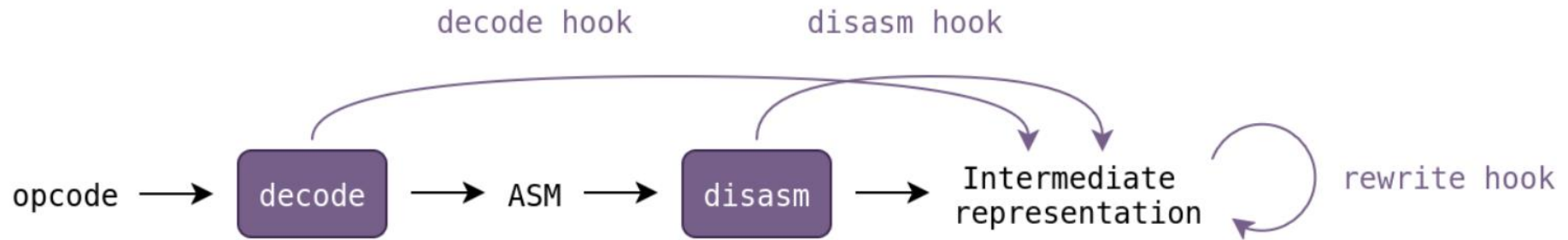
Not all instructions can be skipped with a value fault

IR Faults Problems: Efficiency vs Representation



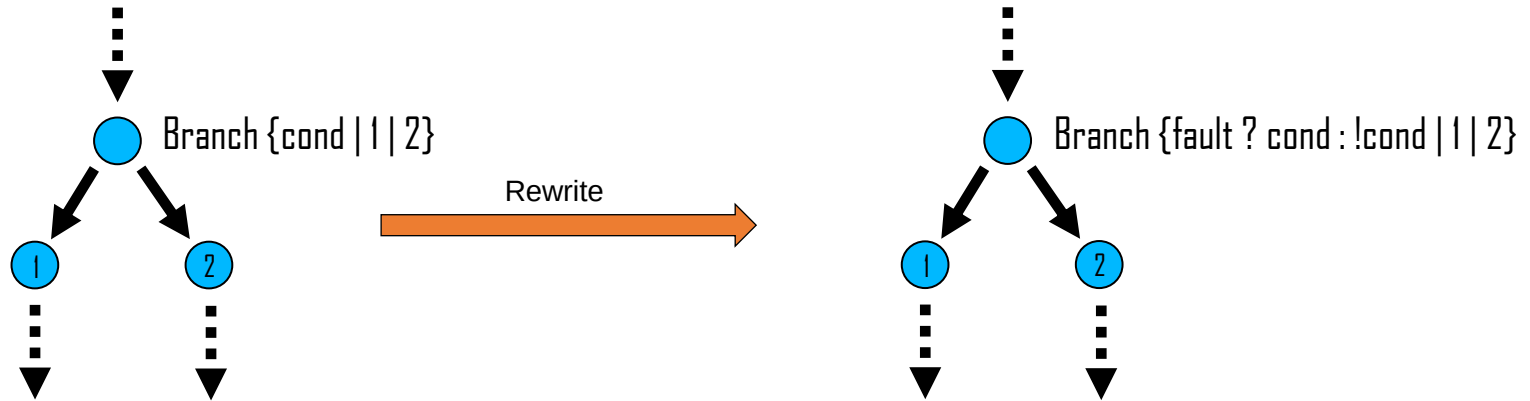
Efficiency decision at the IR level can create difficulties to model instruction-level effects

Decode, Disasm and Rewrite Hooks



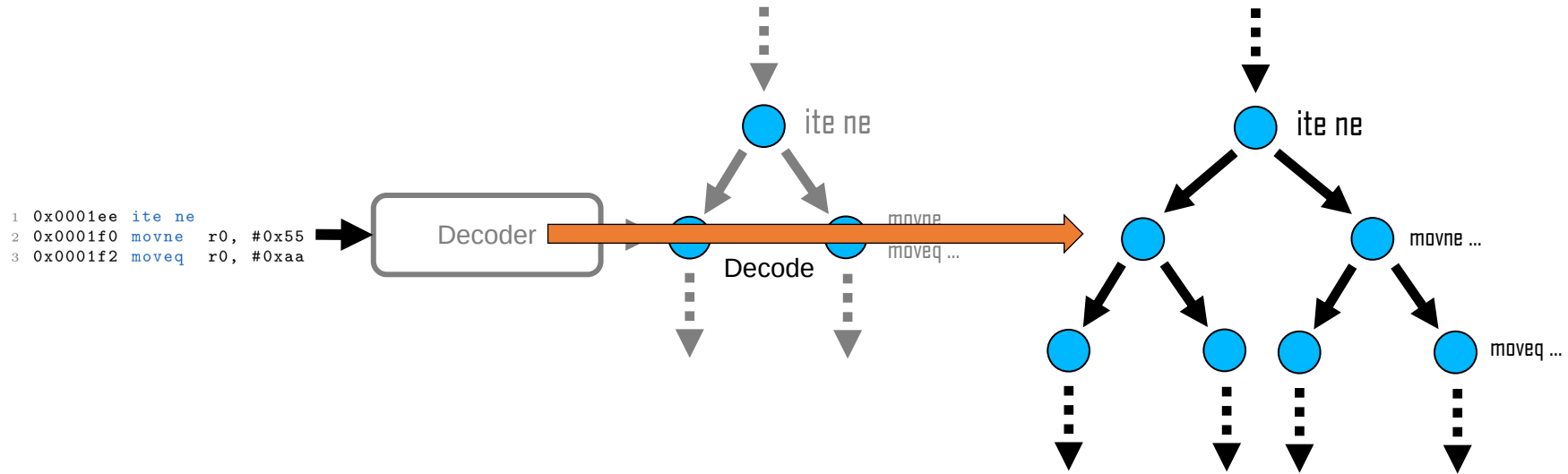
New BINSEC plugin hooks to solve these issues

Example: Rewriting Tests to Model Faults



No more side effects

Example: Rewriting ARM IT blocks



Produce the IR in which we can correctly model our faults

Validating the BINSEC Plugin

Program	Faults	Time old version (s)	Time new version (s)	Status
verifyPIN_0	0	0.045	0.040	✓
verifyPIN_0	1	0.044	0.043	✓
verifyPIN_0	2	0.045	0.041	✓
verifyPIN_1	0	0.043	0.041	✓
verifyPIN_1	1	0.045	0.040	✓
verifyPIN_1	2	0.048	0.043	✓
WooKey_original	0	4.591	3.084	✓
WooKey_original	1	15.613	12.527	✓
WooKey_original	2	19.086	17.925	✓
WooKey_Lacombe	0	2.807	3.062	✓
WooKey_Lacombe	1	16.505	19.609	✓
WooKey_Lacombe	2	30.398	34.500	✓

Table 4: Results on the test inversion benchmark

- Comparison against non-plugin historical version
- Comparison against code mutants analysis

Same results, Varying time difference

Program	Nb of faulted instructions	Time Mutants (s)	Time ASE (s)	Status
fissc-vp.0.00	79	4.62	46.147	✓
fissc-vp.0.01	43	2.551	0.669	✓
fissc-vp.0.02	29	1.48	0.515	✓
fissc-vp.0.03	45	2.213	0.249	✓
fissc-vp.0.0s	38	1.97	0.783	✓
fissc-vp.5.00	94	5.829	88.914	✓
fissc-vp.5.01	62	3.17	1.211	✓
fissc-vp.5.02	51	2.522	0.911	✓
fissc-vp.5.03	45	2.254	0.31	✓
fissc-vp.5.0s	47	2.194	0.992	✓
fissc-vp.8.00	479	2238.373	10000.03	✗
fissc-vp.8.01	150	10.618	42.306	✓
fissc-vp.8.02	169	11.852	82.318	✓
fissc-vp.8.03	61	3.667	1.836	✓
fissc-vp.8.0s	107	8.042	24.127	✓
fissc-vp.9.00	1168	855.621	10000.043	✗
fissc-vp.9.01	132	8.984	24.751	✓
fissc-vp.9.02	61	3.785	2.294	✓
fissc-vp.9.03	61	3.719	2.326	✓
fissc-vp.9.0s	62	3.783	4.618	✓

Table 5: Results on a sample of the ARM mutants benchmark



Leveraging the BINSEC Plugin

- Evaluating compiler protection against fault attacks
- Fault models: Test inversion, Instruction skip, Call skip (1 fault and 2 faults)
- Evaluated compilers:
 - CompCert** [Pesin et. al]
 - CompaSec** [Geier et al.]



Verifying Software Hardening Strategies

Program	No CM		CompCert		CompaSec	
	1F	2F	1F	2F	1F	2F
aes	-	-	-	-	-	-
ark	X	X	✓/✓	X/X	X/X	X/X
fu	X	X	✓/✓	X/X	X/X	X/X
memcmps	X	X	✓/✓	X/X	X/X	X/X

Table 7: Results on the compilers benchmark for test inversion

Legend: Experimental Result / Expected Result

Program	No CM		CompCert		CompCert (call skips only)		CompaSec	
	1F	2F	1F	2F	1F	2F	1F	2F
aes	-	-	-	-	-	-	-	-
ark	X	X	X/X	X/X	✓/✓*	✓/X*	X/✓	X/✓
fu	X	X	X/X	X/X	✓/✓	✓/X	X/✓	X/✓
memcmps	X	X	X/X	X/X	✓/✓	✓/X	X/✓	X/✓

Table 8: Results on the compilers benchmark for instruction skip

*no function call in the range of addresses skipped

Interesting Finding: The CompCert countermeasure can offer more protection than claimed
 Interesting Finding: The CompaSec countermeasure has limitations



A Unified Fault Injection Plugin for BINSEC

- All fault models in a single plugin (Value faults, Masks, Test inversion, Instruction skips)
- Multiarchitecture by default
- Fault budgets, transient and permanent faults, fault target control
- Permits the verification of software protection at binary level
- Preview Demo





Fault Injection in BINSEC: Next Steps

- Robustification and scalability improvements
- Hardware/Software protection modeling
- Contracts for fault injection attacks

Questions

Yanis Sellami

Noémie Bénard

Frédéric Recoules

Sébastien Bardin

Université Paris-Saclay, CEA List Palaiseau, France