Adversarial Reachability for Program-level Security Analysis

(published at ESOP 2023)

Soline Ducousso[1], Sébastien Bardin[1], Marie-Laure Potet[2]

[1] Univ. Paris-Saclay, CEA, List, Saclay, France
[2] Univ. Grenoble Alpes, VERIMAG, Grenoble, France
soline.ducousso@cea.fr, sebastien.bardin@cea.fr, marie-laure.potet@univ-grenoble-alpes.fr

Context - Program Security Evaluation

- Code review
- Pen testing

Manual, not exhaustive, time consuming

Context - Program Security Evaluation

- ❑ Code review
- ❑ Pen testing

Manual, not exhaustive, time consuming

Automatic **program analysis** techniques

Context - Formal Program Analysis

- Formal methods → all possible behaviors are studied
- Verification specifications, bug finding or absence of bugs
- **Industrial success for *safety***

What About Security ?

Reuse standard safety analyzers:

- ❏ Useful (e.g., buffer overflows) and worst case
- ❏ **Weak attacker model** ➔ can only craft smart inputs

```
$ ./exploit
```

What About Security ?

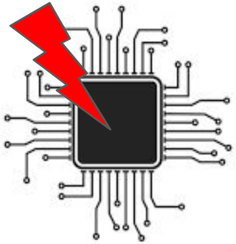
Reuse standard safety analyzers:

- ❏ Useful (e.g., buffer overflows) and worst case
- ❏ Weak attacker model ➔ can only craft smart inputs

```
$ ./exploit
```

Real-world attackers are more powerful

- ❏ **Various attack vectors** ➔ fault injection
- ❏ Side channels
- ❏ **Multiple actions** in one attack



Our Goal

Our goal is to devise a technique to automatically and efficiently reason about the impact of an advanced attacker onto a program security properties.*

Challenges:

C1: Formal framework

Impact of advanced attacker

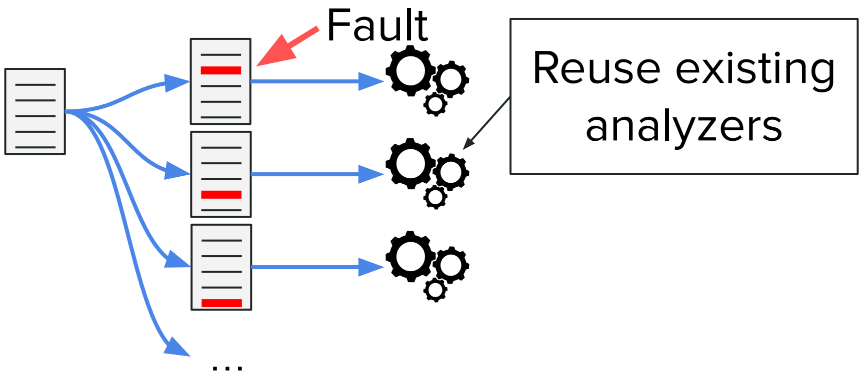
C2: Efficient and generic algorithm

Multi-fault without path explosion

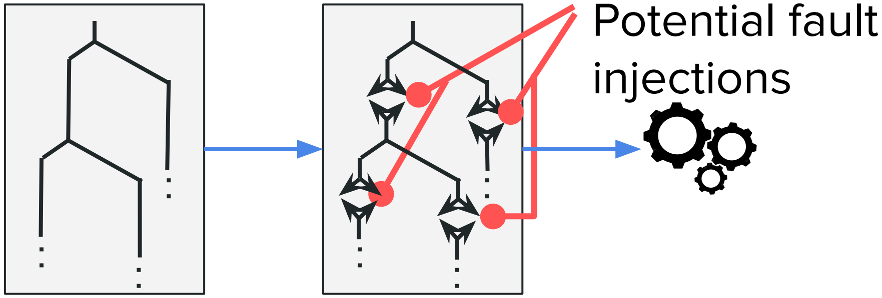
*attacker able to perform multi-fault injections

State-of-the-Art: software-implemented fault injection

Mutant Generation



Forking technique



Scalability issues

Few predefined fault models - no multi-fault - source level analysis

Contributions

- ❏ Formalize of the **Adversarial Reachability** problem
- ❏ **Adversarial Symbolic Execution** to answer adversarial reachability
  - ❏ a novel **forkless fault encodings** preventing path explosion
  - ❏ 2 **optimizations** reducing query complexity
- ❏ **Implementation** and **evaluation** of our technique
- ❏ **Security scenarios** and security analysis of the **WooKey bootloader**

Introduction

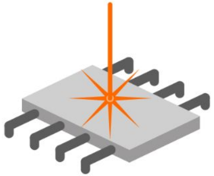
Adversarial Reachability Formalization

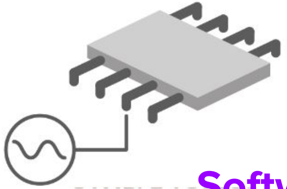
Forkless Adversarial Symbolic Execution

Experimental Evaluation

Conclusion

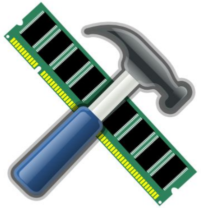
Fault Injection Attacks Everywhere



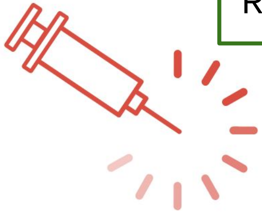


Hardware attacks

Software-implemented hardware attacks



Electromagnetic pulses	Power glitch	Faultline	Rowhammer	
Clock glitch	Laser beam	Spectre	Binary rewriting	DVFS
Race condition	Load Value Injection	Halt and modify execution		



Micro-architectural attacks

Man-At-The-End attacks

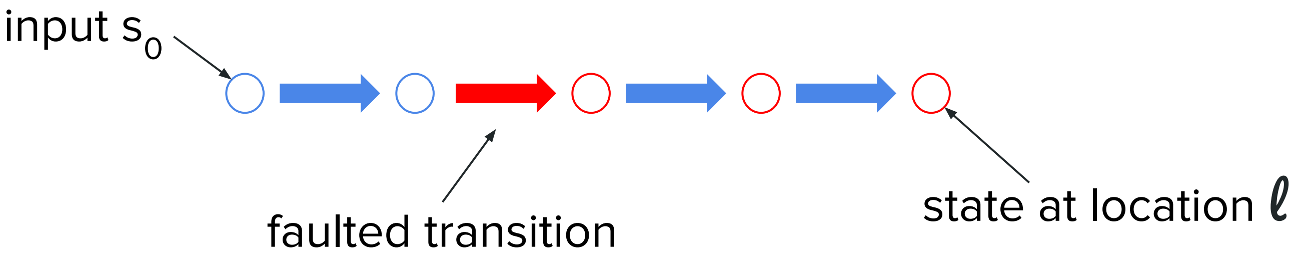
Link with data-only attacks

Model of an advanced attacker

1) A set of attacker actions (equivalent to fault models)
2) A maximum number of actions
3) A goal expressed as a reachability query

Adversarial reachability

Adversarial reachability: A location ℓ is adversarially reachable in a program P for an attacker model A if $S_0 \mapsto^* \ell$,
where $\mapsto^*$ is a succession of **normal transitions** interleaved with **faulty transitions**



Definition of ***correctness*** and ***completeness*** of an analysis w.r.t an attacker model

Introduction
Adversarial Reachability Formalization
Forkless Adversarial Symbolic Execution
Experimental Evaluation
Conclusion

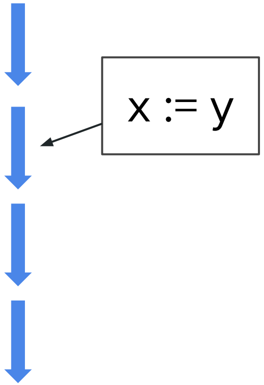
Forkless Adversarial Symbolic Execution (FASE)

Design guideline	**Technical solution**
Correct and k-complete for adversarial reachability	Based on Symbolic Execution
Prevent path explosion	Forkless fault encoding
Reduce complexity of created formulas	Avoid introducing extra faults with 2 optimizations

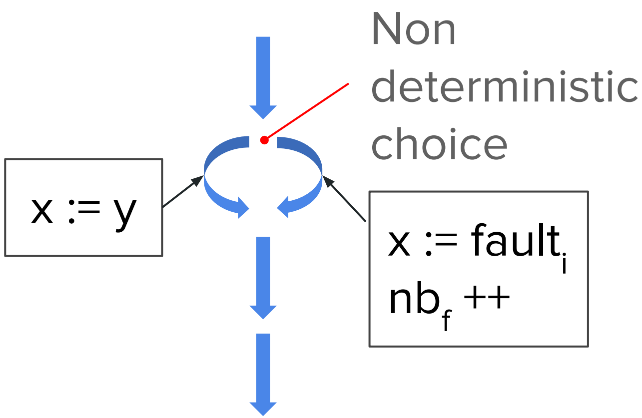
Faults on data

Faults on control-flow

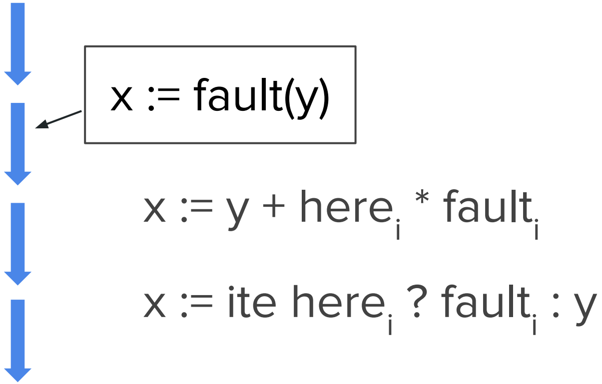
Symbolic execution



Forking technique



FASE



+ Covers all adversarial behaviors

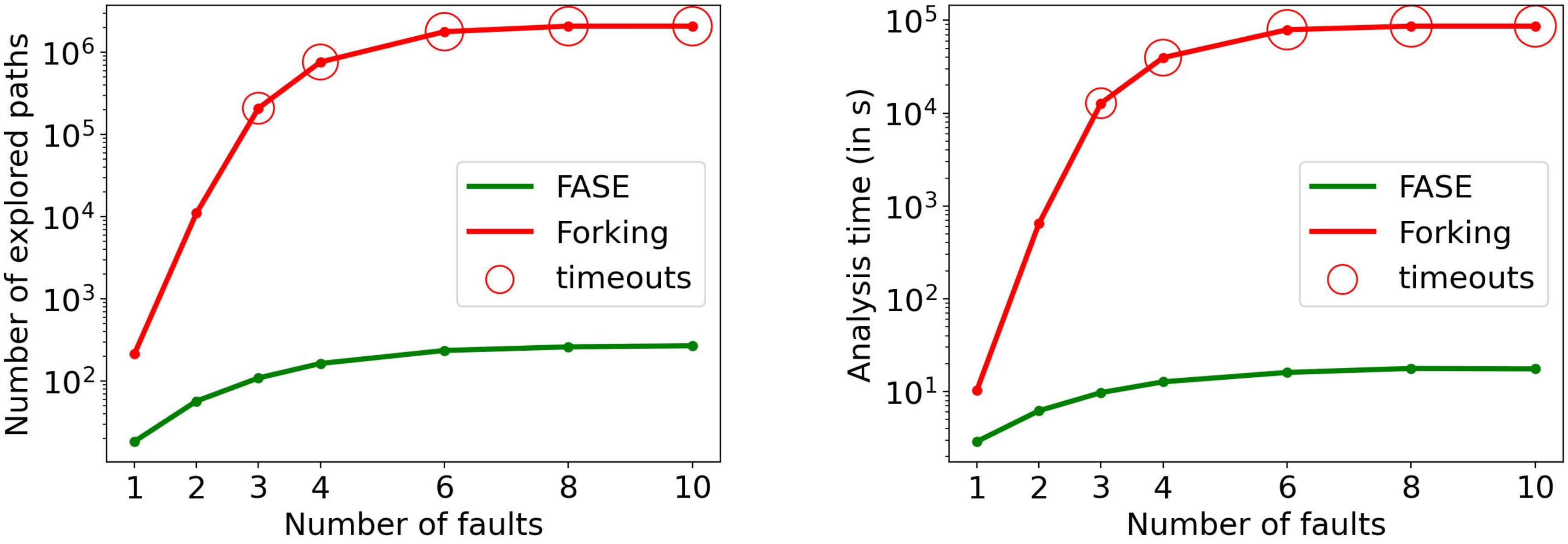
- **#path exponential with #fault injection points**

+ Covers all adversarial behaviors

+ **No extra path**

- **More complex formulas**

Experimental Evaluation - Path explosion

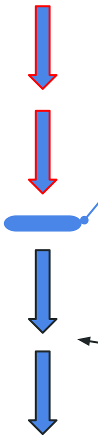


- ➔ Forking explodes in explored paths while FASE doesn't
- ➔ Translates to improved analysis time overall

Optimizations

- Reduce #injection points to simplify formulas
- Remain correct and k-complete

Early Detection of Fault Saturation (EDS)
Stop injection as soon as possible

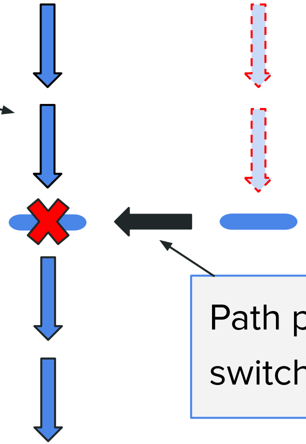


- SAT with a fault margin ($nb_f < max_f$)
- SAT with exactly the fault budget ($nb_f == max_f$)
- infeasible

No more fault injection

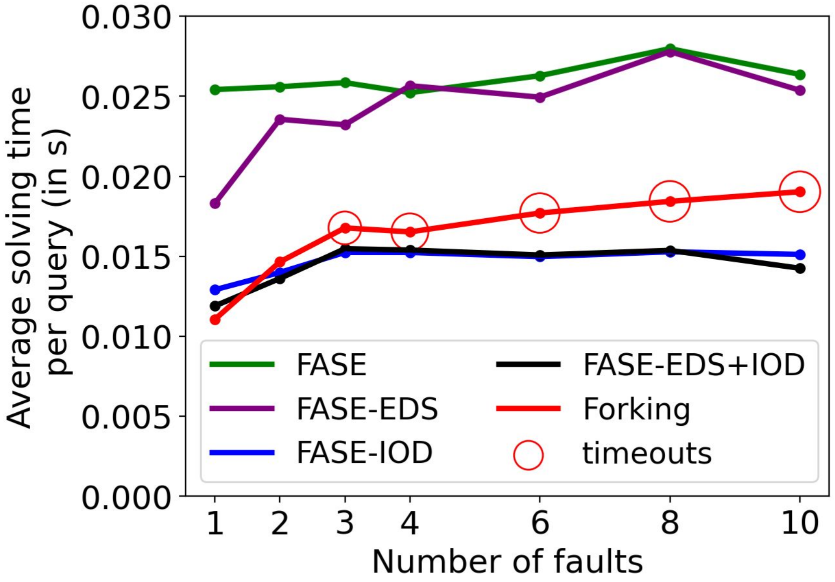
Injection On Demand (IOD)
Add faults only when necessary

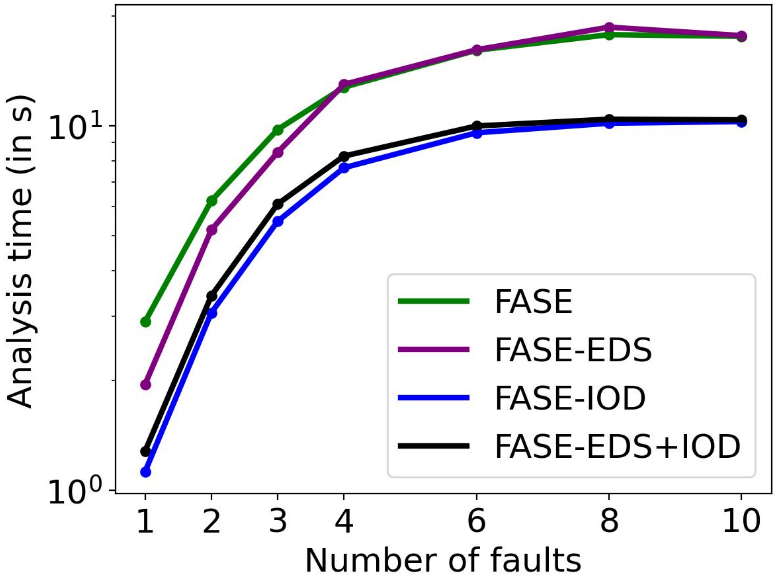
Start without any faults



Path predicate switch

Experimental Evaluation - Optimizations’ Impact





- ➔ EDS has a moderate impact
- ➔ IOD halves solving time per query (5745 →3050 avg ite /query) + most efficient
- ➔ IOD+EDS is slightly more expensive

Other Forkless Fault Models

Fault model	original instruction	Forkless encoding
Arbitrary data	$x := expr$	$x := ite\ fault\_here\ ?\ fault\_value\ :\ expr$
Variable reset	$x := expr$	$x := ite\ fault\_here\ ?\ 0x00000000\ :\ expr$
Variable set	$x := expr$	$x := ite\ fault\_here\ ?\ 0\text{xffffffff}\ :\ expr$
Bit-flip	$x := expr$	$x := ite\ fault\_here\ ?$ $(expr\ xor\ 1 << fault\_value)\ :\ expr$
Test inversion	$if\ cdt$ $then\ goto\ addr_1$ $else\ goto\ addr_2$	$if\ (ite\ fault\_here\ ?\ \neg cdt\ :\ cdt)$ $then\ goto\ addr_1$ $else\ goto\ addr_2$
Instruction skip	$x := expr$	$x := ite\ fault\_here\ ?\ x\ :\ expr$
	$jump\ addr$	$if\ fault\_here\ then\ jump\ next$ $else\ jump\ addr$

NEW

Introduction

Adversarial Reachability Formalization

Forkless Adversarial Symbolic Execution

Experimental Evaluation

Conclusion

Evaluation

Implementation inside BINSEC for x86-32 and ARM architectures with SMT solver Bitwuzla

Benchmarks (RQ1 to 3) from [1, 2]

- ❑ **RQ1:** is our tool correct and k-complete? In particular, can we find attacks on vulnerable programs and prove secure resistant programs?
- ❑ **RQ2:** can we scale in number of faults?
- ❑ **RQ3:** what is the impact of our optimizations?
- ❑ **Different security scenarios** using different fault models
- ❑ **Larger case study** of the WooKey bootloader [ANSSI security challenge]

[1] Dureuil et al. *FISSC: A fault injection and simulation secure collection*. 2016.
[2] Le et al. *Resilience evaluation via symbolic fault injection on intermediate code*. 2018

BellCoRe attack on CRT-RSA

Goal: reproduce the evaluation of different CRT-RSA protections [1]

Attacker model: 1 reset fault

Version	Ground truth	Result
CRT-RSA basic	Insecure	Insecure ✓
CRT-RSA Shamir	Insecure	Time-out without finding attacks ✗
CRT-RSA Aumuller	Secure	Time-out without finding attacks ✓

[1] Puys et al. *High-level simulation for multiple fault injection evaluation*. 2014

Secret keeping machine [1]

Goal: evaluate the impact of implementation on program vulnerability

Attacker model: 1 bit-flip in memory

Version	Attacker model	Ground truth	Result
Linked-list	1 bit-flip in memory	Insecure	Insecure ✓
Array	1 bit-flip in memory	Secure	Secure ✓
Array	1 bit-flip anywhere	Insecure	Insecure ✓

[1] Dullien *Weird machines, exploitability, and provable unexploitability*. 2017

SecSwift [1] protection on VerifyPIN

Goal: evaluate the impact of the protection

Detail: partial implementation [2] only preventing the execution from deviating from the CFG.

Attacker model: 1 arbitrary data fault or 1 test inversion

Version	Ground truth	Result
VerifyPIN_0 with SecSwift	Insecure	Insecure ✓

[1] de Ferrière *Software countermeasures in the llvm risc-v compiler.* 2021
[2] Lacombe et al. *Combining static analysis and dynamic symbolic execution in a toolchain to detect fault injection vulnerabilities*. 2021

Neural Network

NEW

Goal: evaluate the robustness of a neural network (based on [1]) to fault injection

Attacker model: 1 bit-flip

Version	Ground truth	Result
Neural Network	Insecure	Insecure ✓

[1] Mathieu Dumont et al. *Evaluation of parameter-based attacks against embedded neural networks with laser injection*. arXiv preprint, 2023

Case study

WooKey bootloader [security challenge]: secure data storage by ANSSI, 3.2k loc

Attacker model: 1 arbitrary data — or test inversion with equivalent effect

1. **Find known attacks** (from source-level analysis)
   a. Boot on the old firmware instead for the newest one [1]
   b. A buffer overflow triggered by fault injection [1]
   c. An incorrectly implemented countermeasure protecting against one test inversion [2]
2. **Evaluate recent countermeasures** [1]
   a. Evaluate original code ➔ We **found an attack not mentioned before***
   b. Evaluate existing protection scheme [1]
   c. Propose and evaluate our own protection scheme

*After discussion with the authors [1], it turns out that they actually found this path but did not report it in the article, as they did not consider it as a real attack w.r.t. the Wookey challenge.

[1] Lacombe et al. *Combining static analysis and dynamic symbolic execution in a toolchain to detect fault injection vulnerabilities*. 2021
[2] Martin et al. *Verifying redundant-check based countermeasures: a case study*. 2022

Introduction

Adversarial Reachability Formalization

Forkless Adversarial Symbolic Execution

Experimental Evaluation

Conclusion

Conclusion

- New formalization: Adversarial Reachability
- Efficient algorithm: FASE + forkless encoding + optimizations
- Implementation inside BINSEC
- Evaluation: path explosion mitigated + increased efficiency + broad usability.

Limitations:

- no support for general instruction modifications
- no efficient algorithm for faults on addresses

Future perspectives

- Extend attacker model support and efficient algorithms
- Design a hybrid forking/forkless injection technique and heuristics
- Algorithm to find the minimal attacker for a program and a security property

C1: Formal Framework → Adversarial Reachability

C2: Efficient Algorithm → Forkless Adversarial Symbolic Execution (+ 2 optimizations)

Implementation

Evaluation

Use Case: WooKey Bootloader

Security Scenarios

BINSEC