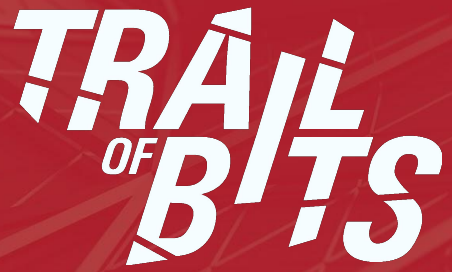


Spearbit: Echidna Workshop

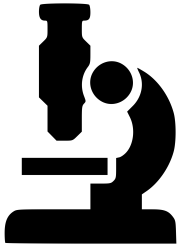
Gustavo Grieco - Trail of Bits

Trail of Bits

Cyber security research company - High-end security research with a real-world attacker mentality to reduce risk and fortify code.

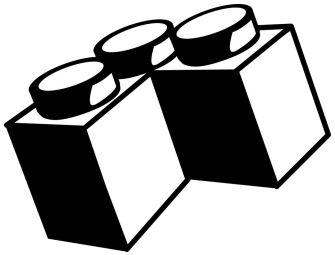
Security Research

- Create and release open source research tools as a leading cybersecurity research provider to companies and governments



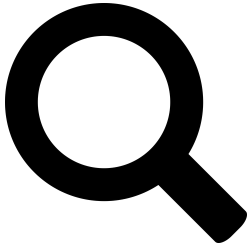
Security Engineering

- Custom engineering for every stage of software creation, from initial planning to enhancing the security of completed works



Security Assessments

- Security auditing for code and systems requiring extreme robustness and niche system expertise



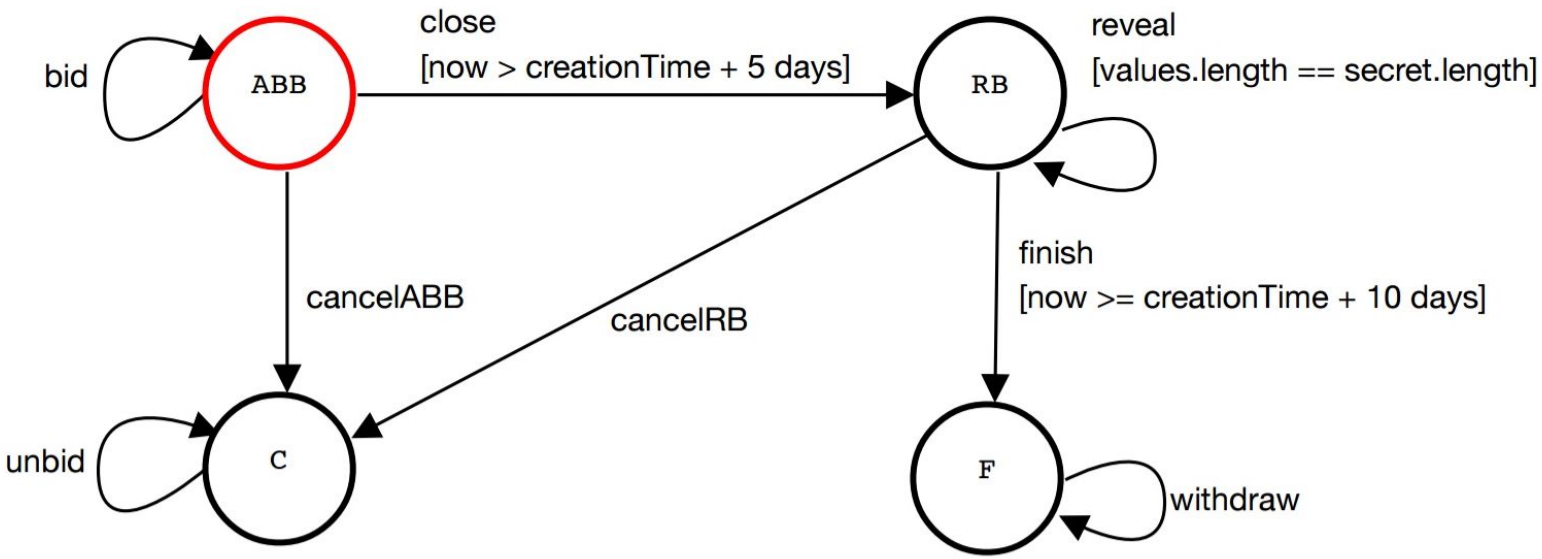
Objectives

- Understand what is fuzzing and why it is important
- Learn how to use Echidna
- Participate in an interactive session of Echidna
- Run some fuzzing campaigns and iterate over the invariants

Introduction

TRAIL OF BITS

Smart contract as state machines



"Designing secure Ethereum smart contracts: a finite state machine based approach" by Mavridou & Laszka, FC'18

Smart contract security in a nutshell

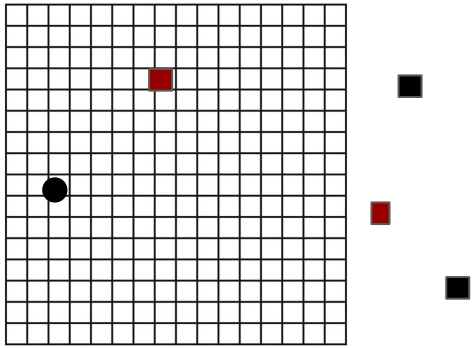
Two important questions:

1. When is a state “bad”?
2. What inputs cause “bad” behavior?

To illustrate, let’s dive into the industry’s solutions

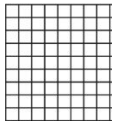
Security (simplified)

Unrestricted Input



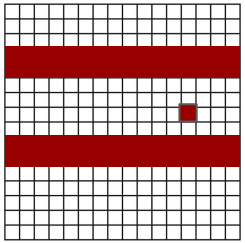
- Anything can happen
- Minimal understanding of what code “should” do

Restrict input



- Typed language
- Delete code
- Privilege separation

Test input



- Unit tests
- Fuzzing
- Property tests

Phase 0: Try really hard

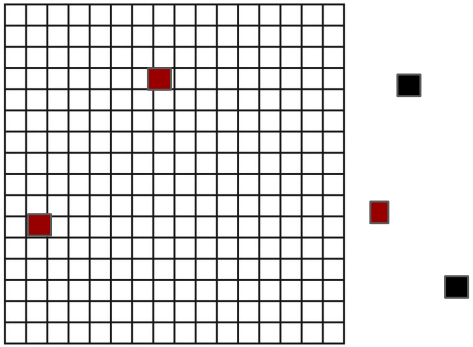
"I would simply think really hard and not introduce issues in my smart contract"

- This absolutely does not work
- How are you managing your team
- Honestly wtf

Phase 1: Try a few inputs

"I would simply list all the things I forgot, then make unit tests"

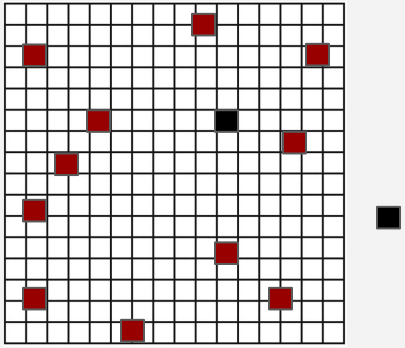
- *Considerably* better than phase 0
- Still doesn't really work
- Most things aren't unit tested
- Programmers won't know all their unknowns



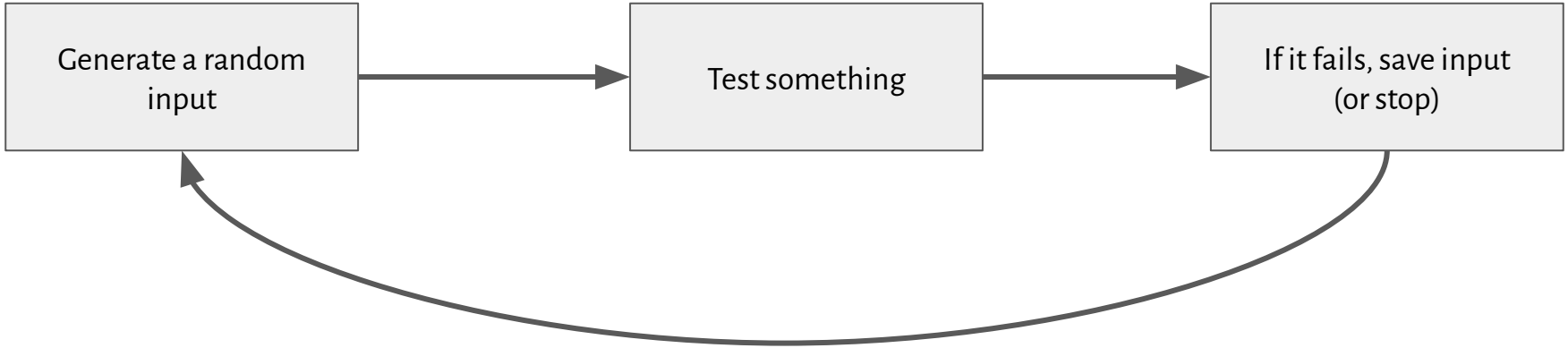
Currently, approximately industry state of the art

Phase 2: Try lots of random inputs

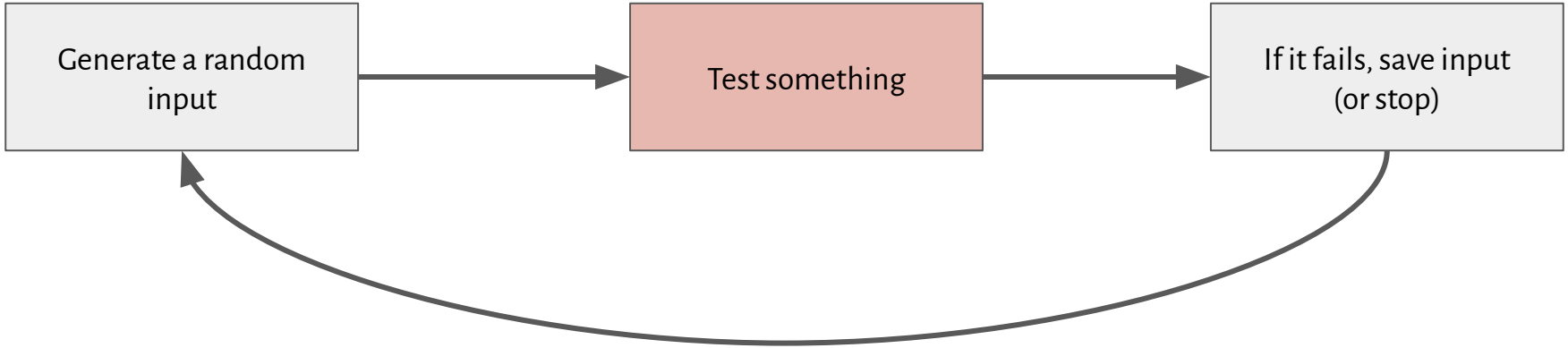
- **Fuzzers, property-based testing**
- **Hot new research area!**
  - Tons of fuzzer papers
  - Tons of property-based testing talks/libraries
- **Fuzzing is starting to gain industry acceptance**



What is fuzzing?



What is fuzzing?



Property based testing

- More general than unit test.
- It has some inputs and a procedure to check if a property failed or not.

Unit test:	Property test:
`isEven(2) == true`	`propEven(n): isEven(2*n) == true`

Echidna: a Fuzzer for Ethereum Smart contracts

- **Probably one of the first open-source smart contract fuzzers available (2018).**
- **Implemented in Haskell using HEVM.**
- **Input generation based on the contract ABI.**
- **Focuses on:**
  - Being ready to use (just download and run it!)
  - Obtaining good results in a short fuzzing campaigns.
  - Well maintained and easy to integrate into your CI pipeline.

A simple example

```
contract Example {

  bool private s0 = true;
  bool private s1 = true;

  function set0(int val) public {
    if (val % 100 == 0) { s0 = false; }
  }
  function set1(int val) public {
    if (val % 10 == 0 && !s0) { s1 = false; }
  }

  function invariant() public {
    assert(s0 || s1);
  }
}
```

Running a fuzzing campaign

```
$ echidna-test simple.sol --test-mode
assertion
...
invariant(): failed!💥
  Call sequence, shrinking (3994/5000):
    set0(0)
    set1(0)
    invariant()
```

When is a state “bad”? (I)

- **Testing is done from Solidity, defining a “target” contract**
- **Several testing mode are supported: boolean properties, assertions, ..**
- **There are no "generic" bug detectors on Echidna:**
  - Integer overflow
  - Reentrancy
  - …

When is a state “bad”? (II)

- In assertion mode, Echidna will randomly run all the functions from the target contract
- It will report when a Solidity assertion failed:

```
function checkInvariant(..) public {
    // Any name and number of arguments is supported

    // The following statements can trigger a failure using `assert`
    assert(..);
    publicFunction(..);
    internalFunction(..);
    ..
} // side effects are preserved
```

The fuzzing loop

1. Write invariants in Solidity
2. Run a fuzzing campaign
3. Check the results and return to (1)

The fuzzing loop

1. Write invariants in Solidity
2. Run a fuzzing campaign
3. Check the results and return to (1)
   - Inspect why invariants fail: review pre- and post-conditions
   - Monitor coverage to make sure all lines are eventually explored

Peeking inside the machine

```
contract Example {
  bool private s0 = true;
  bool private s1 = true;
  event Value(string, bool);

  function set0(int val) public {
    if (val % 100 == 0) { s0 = false; }
  }
  function set1(int val) public {
    if (val % 10 == 0 && !s0) { s1 = false; }
  }
  function invariant() public {
    emit Value(“s0”, s0);
    emit Value(“s1”, s1);
    assert(s0 || s1);
  }
}
```

Running a fuzzing campaign

```
$ echidna-test simple.sol --test-mode
assertion
...

Event sequence: Panic(1),
Value("s0", false)
Value("s1", false)
```

Visualizing coverage

```
$ echidna-test simple.sol --test-mode assertion --corpus-dir
corpus
…
$ ls corpus/ -R
corpus/:
coverage  covered.1652873219.txt

corpus/coverage:
-6154293090267651312.txt  -7375315023076800812.txt
-7413198817945434618.txt
```

Visualizing coverage

```
$ echidna-test simple.sol --test-mode assertion --corpus-dir
corpus
...
$ cat corpus/covered.1652873219.txt
...
*   |   function set0(int val) public {
*   |     if (val % 100 == 0) { s0 = false; }
    |   }
*   |   function set1(int val) public {
*   |     if (val % 10 == 0 && !s0) { s1 = false; }
    |   }
    |
*r  |   function invariant() public {
*r  |     assert(s0 || s1);
    |   }
```

Let's get cracking!

TRAIL OF BITS

The Tool

Install Echidna 2.0.2:

- Install/upgrade slither:
  `pip3 install slither-analyzer --upgrade`

- Recommended option: precompiled binaries:
  Linux (x86) and MacOS (x86 and M1).

- Alternative option: use docker (x86 only)

The Target

```
contract SomeDefi is ERC20 {
  ERC20 public token;

  function mintShares(uint256 tokens) public {
    …
  }

  function withdrawShares(uint256 shares) public {
    …
  }

  function sharesOf(address user) public returns (uint256) {
    …
  }
}
```

The Test

```
$ echidna-test
--test-mode assertion
--contract TestSomeDefi
--config SomeDefi.yaml
SomeDeFi.sol
```

The recommended Solidity version for the fuzzing campaign is 0.8.1, however, more recent releases can be used as well.

Echidna (interactive) demo

Some pointers to continue the testing

1. Start reading the documentation/specification
2. Think of basic properties for every operation
3. Consider when an operation should or it should *not* revert
4. Optimize properties, perhaps merging some of them.

https://github.com/crytic/echidna-spearbit-demo

Trail of Bits is hiring

- jobs.lever.co/trailofbits
- **Security Consultant**
  - Work with leading industry teams to review their code
  - Contribute to our tools & push the state of the art
  - All chains, from Ethereum to Cosmos, going through Solana & Cairo
- **Apprenticeship**
  - 3 months program to train security consultant
  - Audits shadowing & mentor personal feedback