State of the Art
Ethereum Smart Contract Fuzzing in 2022

Patrick Ventuzelo (@Pat_Ventuzelo)

- Founder & CEO of **FuzzingLabs** | Senior **Security Researcher**
  - Fuzzing and vulnerability research
  - Development of security tools

- Training/Online courses
  - **Rust** Security Audit & Fuzzing
  - **Go** Security Audit & Fuzzing
  - **WebAssembly** Reversing & Analysis
  - **Ethereum/Solidity** Security (WIP)
  - **Cairo** Security (WIP)

FUZZING LABS

- Blockchain security since 2016
  - EthCC speaker (twice), Devcon speaker
  - Creator of **Octopus**
  - Public research about EVM reversing & analysis
  - Lead developer of **Beaconfuzz**, eth2 differential fuzzer
  - Fuzzing and audits of dozen of L1/L2 implementations

FUZZING LABS

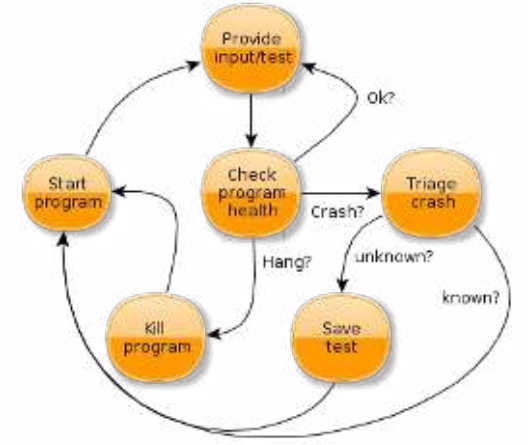
State of the Art

What’s fuzzing?

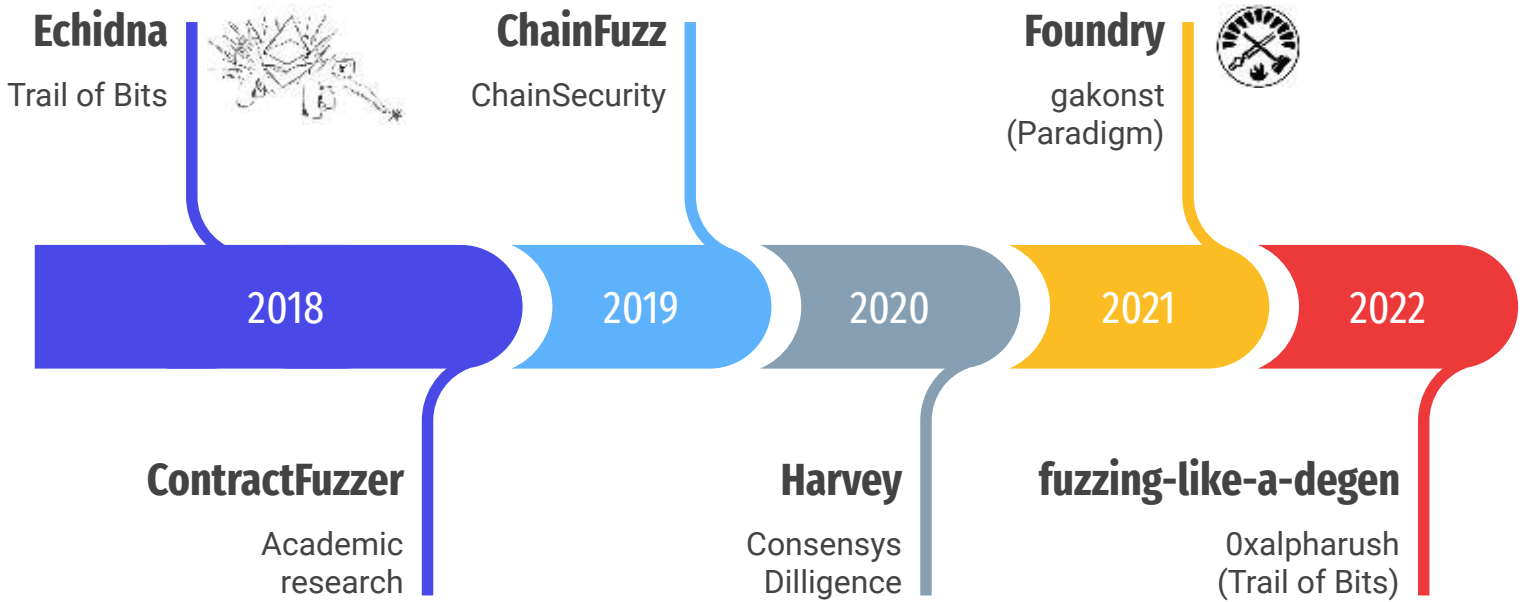
- Fuzzing or fuzz testing is an **automated software testing technique** that involves providing invalid, unexpected, or **random data as inputs** to a computer program. The program is then **monitored for exceptions** such as crashes, failing built-in code assertions, or for finding potential memory leaks and other unexpected behaviors - source

- **The most efficient technique to find bugs!**

- Different fuzzing approaches:
  - **Black box**:
    - You don’t have any real knowledge of the target
    - You don’t have access to the source code
    - You are not able to recompile the target.
  - **Gray box**:
    - You have some knowledge of the target
    - You are not able to recompile the target.
  - **White box**:
    - You have access to the source code
    - You can recompile the target.

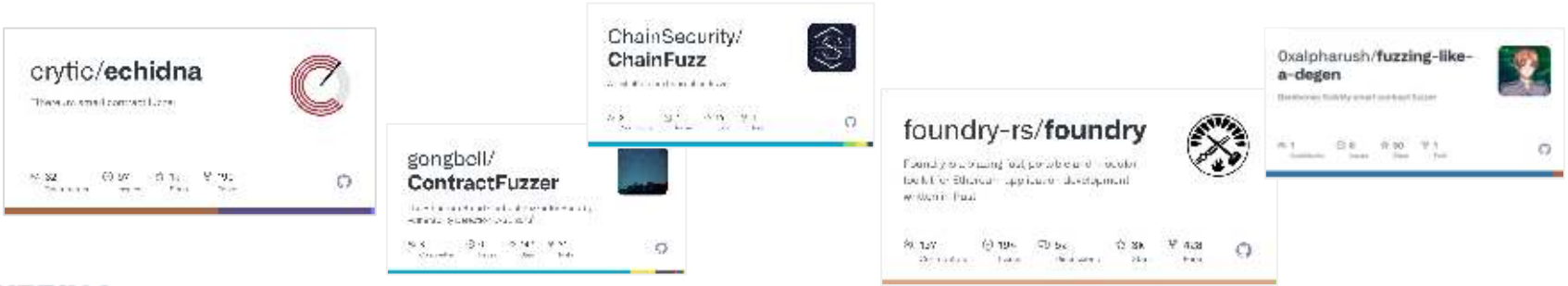


Existing Ethereum smart contract fuzzers



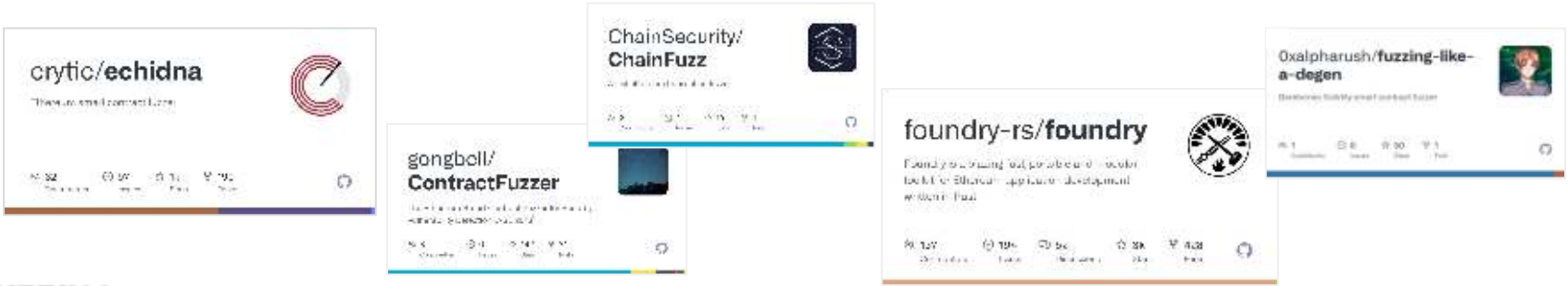
Existing fuzzers - Quick overview

	Echidna	ContractFuzzer	ChainFuzz	Harvey	Foundry	fuzzing-like-a-degen
Open-source	Yes	Yes	Yes	No	Yes	Yes



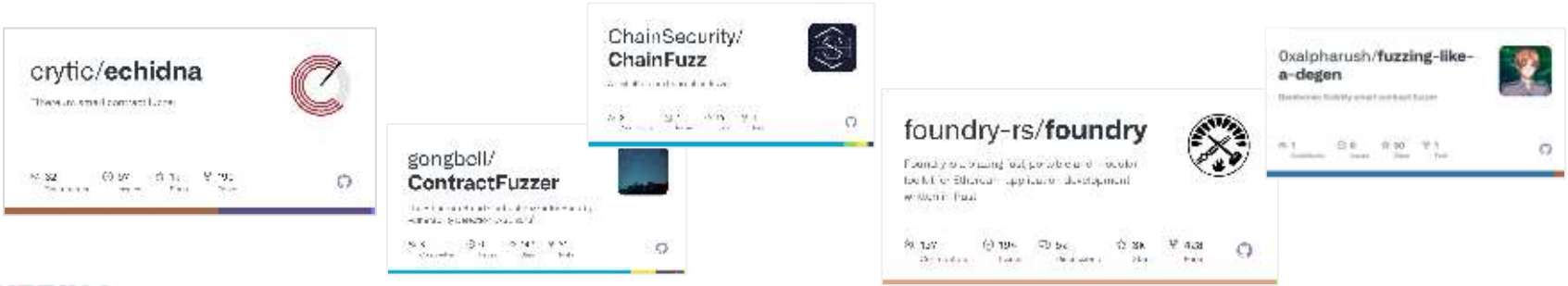
Existing fuzzers - Quick overview

	Echidna	ContractFuzzer	ChainFuzz	Harvey	Foundry	fuzzing-like-a-degen
Open-source	Yes	Yes	Yes	No	Yes	Yes
Language of development	Haskell	Go	Go	Unknown	Rust	Python



Existing fuzzers - Quick overview

	Echidna	ContractFuzzer	ChainFuzz	Harvey	Foundry	fuzzing-like-a-degen
Open-source	Yes	Yes	Yes	No	Yes	Yes
Language of development	Haskell	Go	Go	Unknown	Rust	Python
Documentation	Yes	No	No	Not public	Yes	No



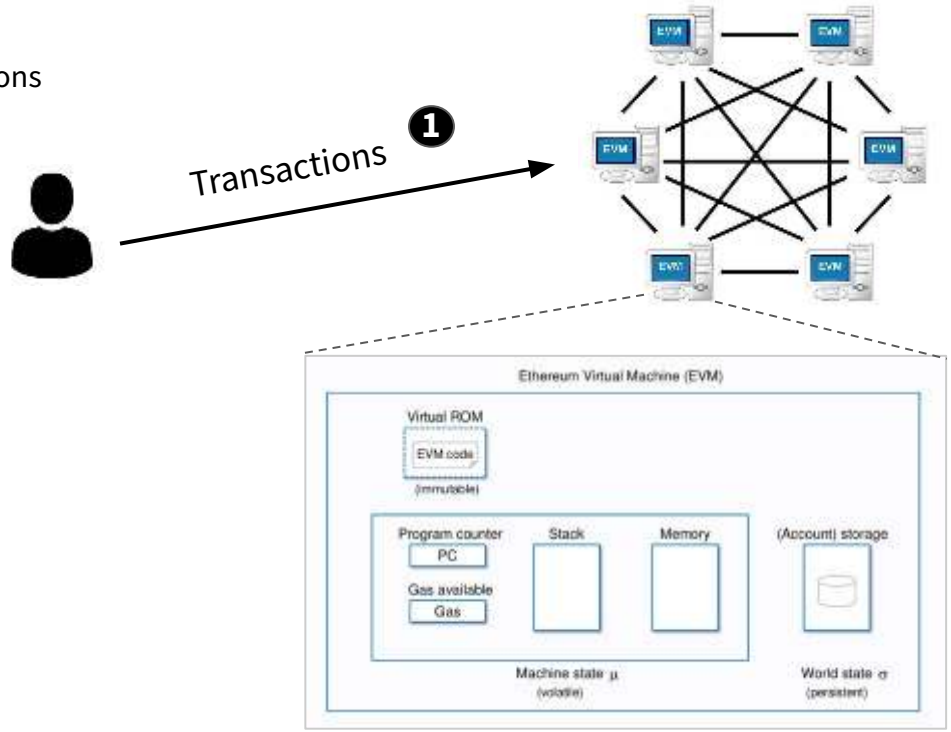
Existing fuzzers - Quick overview

	Echidna	ContractFuzzer	ChainFuzz	Harvey	Foundry	fuzzing-like-a-degen
Open-source	Yes	Yes	Yes	No	Yes	Yes
Language of development	Haskell	Go	Go	Unknown	Rust	Python
Documentation	Yes	No	No	Not public	Yes	No
Maintained	Yes	No	No	Yes	Yes	PoC

Fuzzers internals

Fuzzers internals - Main features

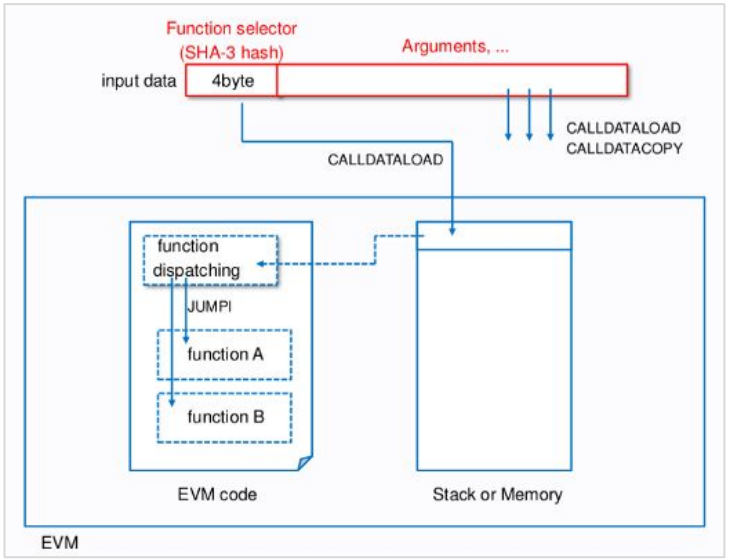
1. **Input generation**
   - Create transactions and sequence of transactions
2. **Execution engine**
   - Run the contract bytecode
3. **Detection mechanisms**
   - Detect when bugs or vulnerabilities occur
4. **Code coverage** (EXTRA)
   - Monitor the bytecode executed during fuzzing
5. **Corpus replay** (EXTRA)
   - Re-execute previous testing inputs
6. **Gas usage** (EXTRA)
   - Detect abnormal gas consumption



Fuzzers internals - Input generation

- Generation of random **input calldata**
  - Leveraging on Solidity source code & Bytecode & ABI
    - to get Method IDs + function parameters types
  - More complicated if you only have the Bytecode
    - Bytecode analysis to find the Method IDs
    - 4-byte signatures DB to find parameters types

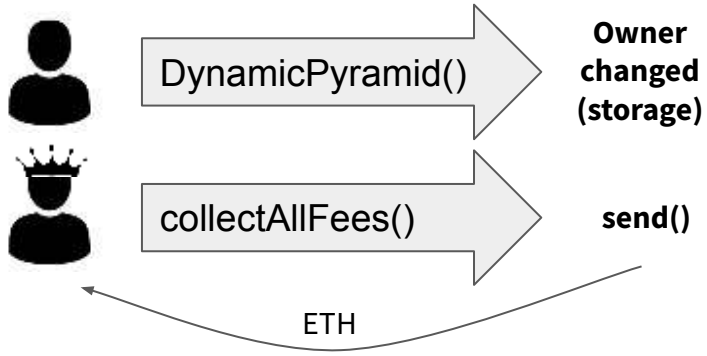
```
contract Foo {
  function Bar(int256 a, int256 b) public view returns (int256) {
    if (a == 42) {
      if (b == 1337) {
        assert(false);
        return 1;
      }
    } else {
        return 0;
    }
  }
}
```



Fuzzers internals - Input generation

- Generation of random **input calldata**
  - Leveraging on Solidity source code & Bytecode & ABI
    - to get Method IDs + function parameters types
  - More complicated if you only have the Bytecode
    - Bytecode analysis to find the Method IDs
    - 4-byte signatures DB to find parameters types

- Generation of **sequences of transactions**
  - Will change the internal contract state

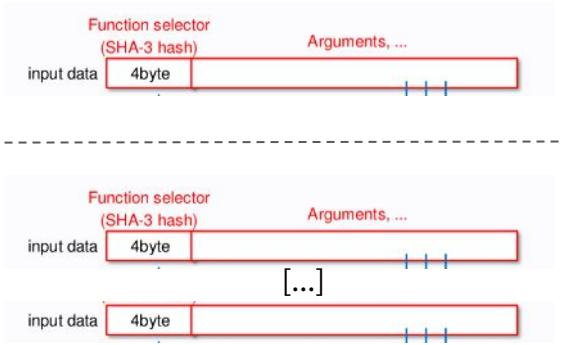



```
contract Rubixi {
  address private owner;
  function DynamicPyramid() { owner = msg.sender; }
  function collectAllFees() { owner.send(collectedFees); }
  ...
```

Fuzzers internals - Input generation

- Generation of random **input calldata**
  - Leveraging on Solidity source code & Bytecode & ABI
    - to get Method IDs + function parameters types
  - More complicated if you only have the Bytecode
    - Bytecode analysis to find the Method IDs
    - 4-byte signatures DB to find parameters types

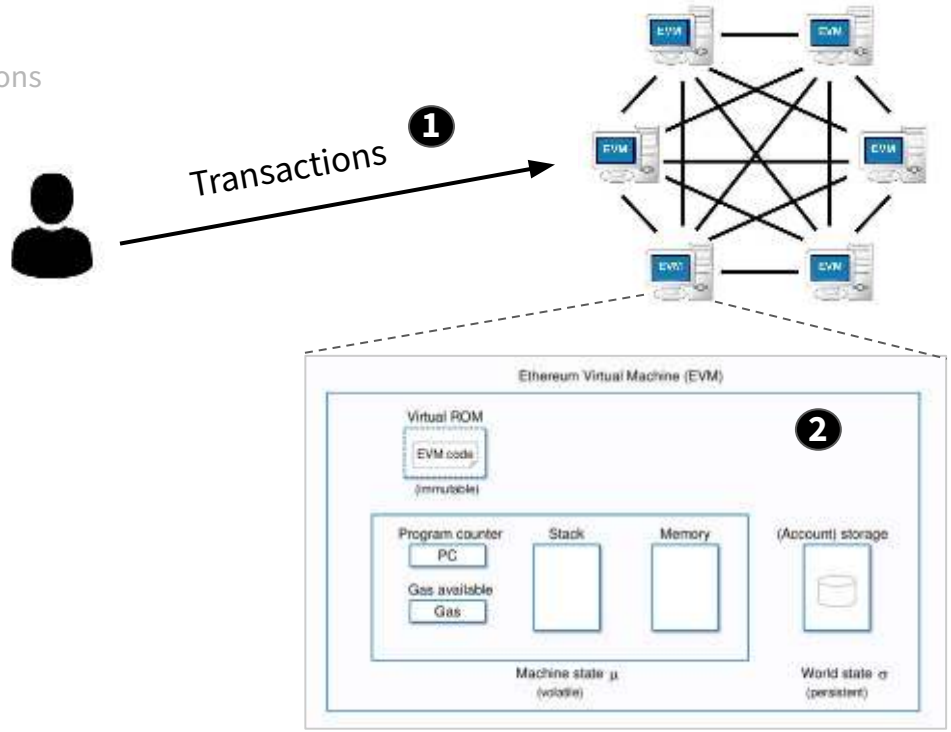
- Generation of **sequences of transactions**
  - Will change the internal contract state



	Echidna	ContractFuzzer	ChainFuzz	Harvey	Foundry	fuzzing-like-a-degen
Smart contract requirement	Source code	Bytecode or Bytecode + ABI	Source code	Source code	Source code	Source code
Input generation	Rng Tx + seq	Rng Tx + seq	Rng Tx + seq	Rng Tx + seq	Rng Tx	Rng Tx

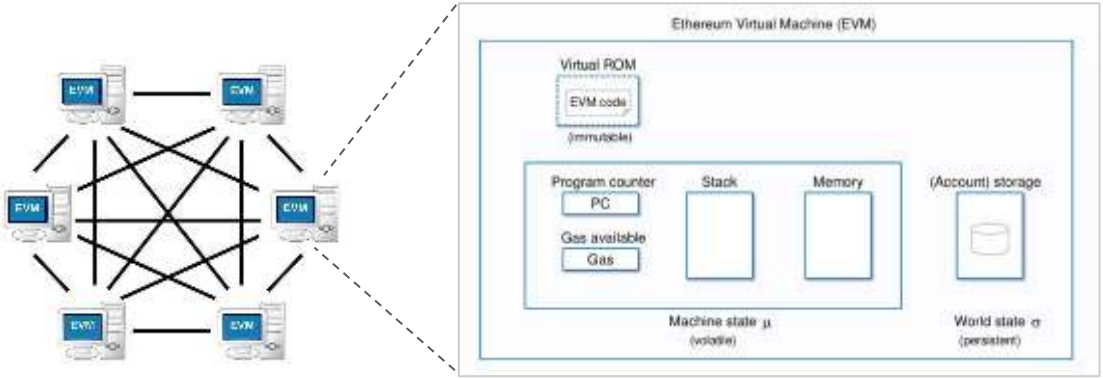
Fuzzers internals - Main features

1. **Input generation**
   - Create transactions and sequence of transactions
2. **Execution engine**
   - Run the contract bytecode
3. **Detection mechanisms**
   - Detect when bugs or vulnerabilities occur
4. **Code coverage** (EXTRA)
   - Monitor the bytecode executed during fuzzing
5. **Corpus replay** (EXTRA)
   - Re-execute previous testing inputs
6. **Gas usage** (EXTRA)
   - Detect abnormal gas consumption



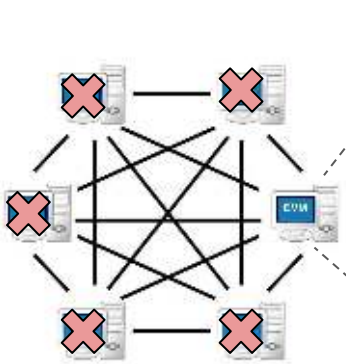
Fuzzers internals - Execution engine

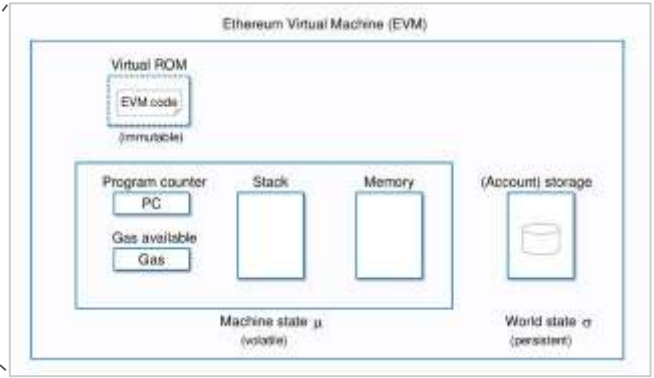
- **Local testnet**
  - Set up and run a local testnet
  - Easy to implement using Ganache



Fuzzers internals - Execution engine

- **Local testnet**
  - Set up and run a local testnet
  - Easy to implement using Ganache
- **EVM emulation**
  - Directly process the Tx to the EVM
  - Execute the contract
  - Generate the new state
  - Faster execution speed



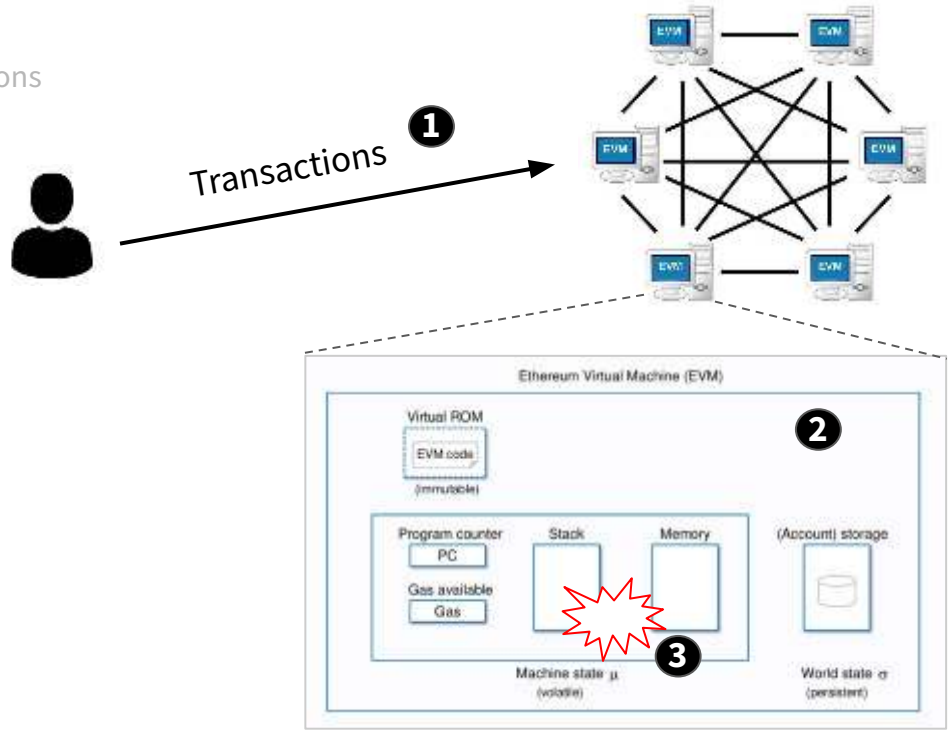


	Echidna	ContractFuzzer	ChainFuzz	Harvey	Foundry	fuzzing-like-a-degen
Execution environment	Local EVM (hevm)	Local testnet	Local testnet (Ganache)	Local testnet (Ganache)	Local EVM (revm)	Local testnet (Ganache)

FUZZING LABS

Fuzzers internals - Main features

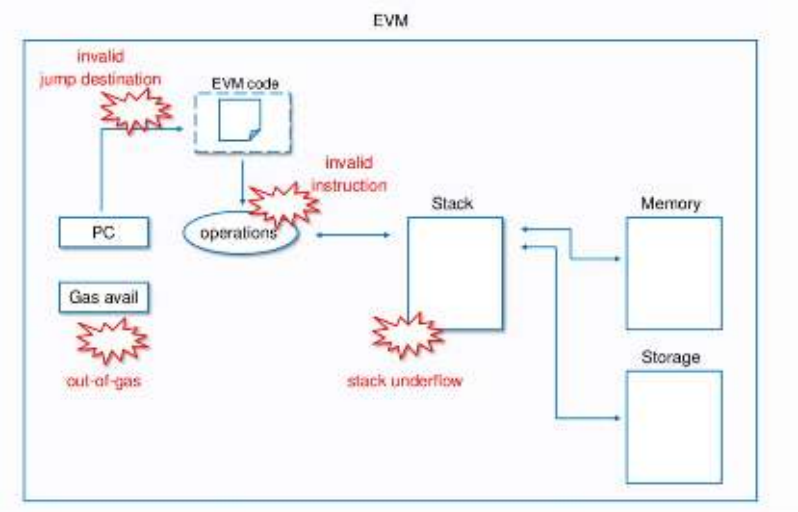
1. **Input generation**
   - Create transactions and sequence of transactions
2. **Execution engine**
   - Run the contract bytecode
3. **Detection mechanisms**
   - Detect when bugs or vulnerabilities occur
4. **Code coverage** (EXTRA)
   - Monitor the bytecode executed during fuzzing
5. **Corpus replay** (EXTRA)
   - Re-execute previous testing inputs
6. **Gas usage** (EXTRA)
   - Detect abnormal gas consumption



Fuzzers internals - Detection mechanisms

- **EVM runtime exceptions**
  - Explicit opcodes: STOP, REVERT, SELFDESTRUCT, INVALID
  - Out of Gas, Illegal Instruction, etc.
  - Zero division, Assertion failure, etc.
  - Arithmetic Overflow/Underflow

```
// List evm execution errors
var (
        ErrOutOfGas                 = errors.New("out of gas")
        ErrCodeStoreOutOfGas        = errors.New("contract creation code storage out of gas")
        ErrDepth                    = errors.New("max call depth exceeded")
        ErrInsufficientBalance      = errors.New("insufficient balance for transfer")
        ErrContractAddressCollision = errors.New("contract address collision")
        ErrExecutionReverted        = errors.New("execution reverted")
        ErrMaxCodeSizeExceeded      = errors.New("max code size exceeded")
        ErrInvalidJump              = errors.New("invalid jump destination")
        ErrWriteProtection          = errors.New("write protection")
        ErrReturnDataOutOfBounds    = errors.New("return data out of bounds")
        ErrGasUintOverflow          = errors.New("gas uint64 overflow")
        ErrInvalidCode              = errors.New("invalid code: must not begin with 0xef")
        ErrNonceUintOverflow        = errors.New("nonce uint64 overflow")
```



Fuzzers internals - Detection mechanisms

- **EVM runtime exceptions**
  - Explicit opcodes: STOP, REVERT, SELFDESTRUCT, INVALID
  - Out of Gas, Illegal Instruction, etc.
  - Zero division, Assertion failure, etc.
  - Arithmetic Overflow/Underflow

- **Property testing**
  - Testing function checking particular assertions
  - Useful to detect logic bugs
  - Need to be written manually

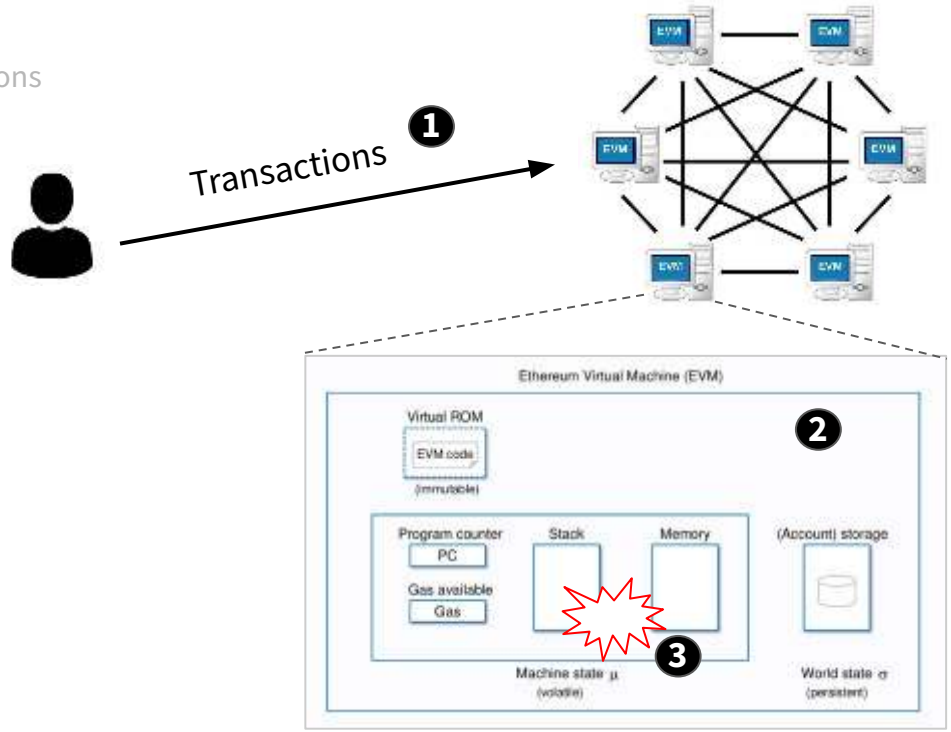
```
function sub(uint x, uint y) internal pure returns (uint z) {
    require((z = x - y) <= x);
}

function checkAnInvariant() public {
    bytes32 senderSlate = votes[msg.sender];
    address option = slates[senderSlate];
    uint256 senderDeposit = deposits[msg.sender];

    assert(approvals[option] >= senderDeposit);
}
```

	Echidna	ContractFuzzer	ChainFuzz	Harvey	Foundry	fuzzing-like-a-degen
Runtime exceptions	Yes	Yes	Yes	Yes	Yes	No
Property testing	Solidity	Not supported	Solidity	Scribble/Solidity	Solidity	Solidity

FUZZING LABS

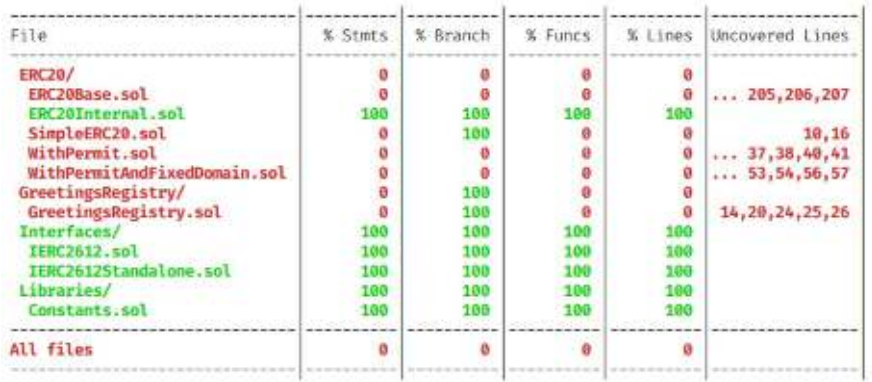
Fuzzers internals - Main features

1. **Input generation**
   - Create transactions and sequence of transactions
2. **Execution engine**
   - Run the contract bytecode
3. **Detection mechanisms**
   - Detect when bugs or vulnerabilities occur
4. **Code coverage** (EXTRA)
   - Monitor the bytecode executed during fuzzing
5. **Corpus replay** (EXTRA)
   - Re-execute previous testing inputs
6. **Gas usage** (EXTRA)
   - Detect abnormal gas consumption



Fuzzers internals - Extra features

- **Code coverage**
  - Monitor the code executed during fuzzing

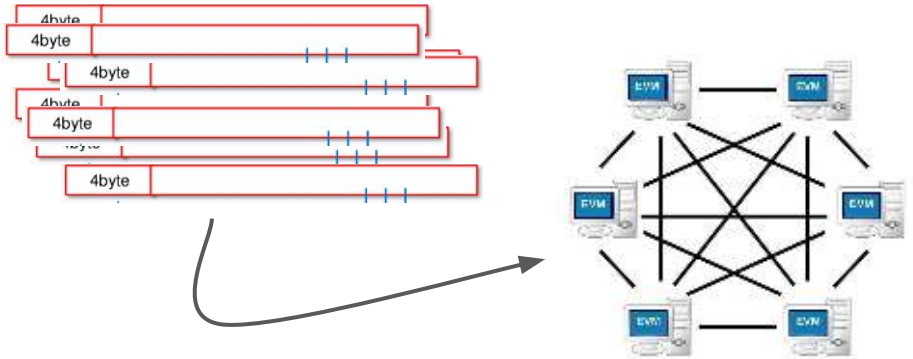


File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
ERC20/	0	0	0	0	
ERC20Base.sol	0	0	0	0	... 205,206,207
ERC20Internal.sol	100	100	100	100	
SimpleERC20.sol	0	100	0	0	10,16
WithPermit.sol	0	0	0	0	... 37,38,40,41
WithPermitAndFixedDomain.sol	0	0	0	0	... 53,54,56,57
GreetingsRegistry/	0	100	0	0	
GreetingsRegistry.sol	0	100	0	0	14,20,24,25,26
Interfaces/	100	100	100	100	
IERC2612.sol	100	100	100	100	
IERC2612Standalone.sol	100	100	100	100	
Libraries/	100	100	100	100	
Constants.sol	100	100	100	100	
All files	0	0	0	0	

	Echidna	ContractFuzzer	ChainFuzz	Harvey	Foundry	fuzzing-like-a-degen
Code coverage	Yes	No	Yes	Yes	Partial	No

Fuzzers internals - Extra features

- **Code coverage**
  - Monitor the code executed during fuzzing
- **Corpus/Transactions replay**
  - Re-execute previous testing inputs
  - Make it easy to integrate into your CI/CD



	Echidna	ContractFuzzer	ChainFuzz	Harvey	Foundry	fuzzing-like-a-degen
Code coverage	Yes	No	Yes	Yes	Partial	No
Corpus replay	Yes	No	No	Yes	No	No

Fuzzers internals - Extra features

- **Code coverage**
  - Monitor the code executed during fuzzing
- **Corpus/Transactions replay**
  - Re-execute previous testing inputs
  - Make it easy to integrate into your CI/CD
- **Gas usage report**
  - Detect abnormal gas consumption

Greeter contract					
Deployment Cost	Deployment Size				
370727	1859				
Function Name	min	avg	median	max	# calls
gm	2431	10109	2871	25025	3
greet	1371	12457	12457	23543	2
greeting	1285	1285	1285	1285	2
transferOwnership	2351	2351	2351	2351	6

	Echidna	ContractFuzzer	ChainFuzz	Harvey	Foundry	fuzzing-like-a-degen
Code coverage	Yes	No	Yes	Yes	Partial	No
Corpus replay	Yes	No	No	Yes	No	No
Gas usage	Yes	No	No	No	Yes	No

Which one to choose?

	Echidna	ContractFuzzer	ChainFuzz	Harvey	Foundry	fuzzing-like-a-degen
Input generation	Rng Tx + seq	Rng Tx + seq	Rng Tx + seq	Rng Tx + seq	Rng Tx	Rng Tx
Smart contract requirement	Source code	Bytecode or Bytecode + ABI	Source code	Source code	Source code	Source code
Execution environment	Local EVM (hevm)	Local testnet	Local testnet (Ganache)	Local testnet (Ganache)	Local EVM (revm)	Local testnet (Ganache)
Runtime exceptions	Yes	Yes	Yes	Yes	Yes	No
Property violations	Solidity	Not supported	Solidity	Scribble/Solidity	Solidity	Solidity
Code coverage	Yes	No	Yes	Yes	Partial	No
Corpus replay	Yes	No	No	Yes	No	No
Gas usage	Yes	No	No	No	Yes	No
Speed	Fast	Unknown	Unknown	Unknown	Fast++	Slow++

	Echidna	ContractFuzzer	ChainFuzz	Harvey	Foundry	fuzzing-like-a-degen
Input generation	Rng Tx + seq	Rng Tx + seq	Rng Tx + seq	Rng Tx + seq	Rng Tx	Rng Tx
Smart contract requirement	Source code	Bytecode or Bytecode + ABI	Source code	Source code	Source code	Source code
Execution environment	Local EVM (hevm)	Local testnet	Local testnet (Ganache)	Local testnet (Ganache)	Local EVM (revm)	Local testnet (Ganache)
Runtime exceptions	Yes	Yes	Yes	Yes	Yes	No
Property violations	Solidity	Not supported	Solidity	Scribble/Solidity	Solidity	Solidity
Code coverage	Yes	No	Yes	Yes	Partial	No
Corpus replay	Yes	No	No	Yes	No	No
Gas usage	Yes	No	No	No	Yes	No
Speed	Fast	Unknown	Unknown	Unknown	Fast++	Slow++

FUZZING LABS

Which one to choose?

1. Echidna
   - One of the first fuzzer available
   - Detailed and in-depth tutorials and documentation
   - **The most complete**

2. Foundry
   - Really fast, extendable, and written in Rust
   - Active community
   - **The most promising**
     - A dedicated fuzz subcommand should be available

3. Harvey
   - Look to have all the features a good fuzzer needs
   - But it's **CLOSED SOURCE!!!!!!!!!!!!!!!**

- **Do we need another fuzzer?**
  - Maybe a BlackBox fuzzer (EVM bytecode only)
  - Detecting bugs at the bytecode level by monitoring contracts state

Thanks for your time! Any questions?

- Twitter: @Pat_Ventuzelo
- Mail: patrick@fuzzinglabs.com

SLIDES