EthPloit: From Fuzzing to Efficient Exploit Generation against Smart Contracts

Qingzhao Zhang[1,2], **Yizhuo Wang**[1], Juanru Li[1], Siqi Ma[3]

[1]Shanghai Jiao Tong University, China
[2]University of Michigan, America
[3]Data 61, CSIRO, Australia

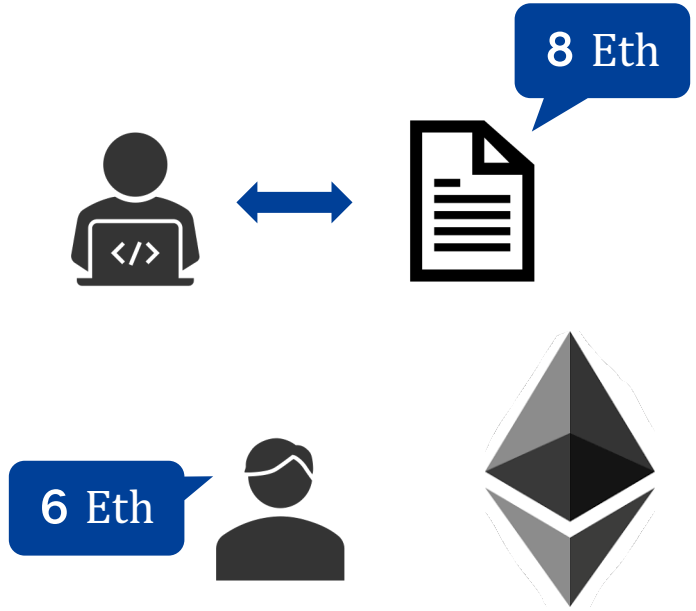
SANER'20, London ON. Canada, Febrary 21, 2020

Contents

Contents

Overview of Ethereum

Ethereum is the second-largest blockchain system



In General

- A **programmable** blockchain
- A platform for **decentralized** applications.

In Detail

- A transaction-based **state machine**
- The heart is **Ethereum Virtual Machine** (EVM)
- Based on Turing-complete programming language (Solidity)

Overview of Ethereum

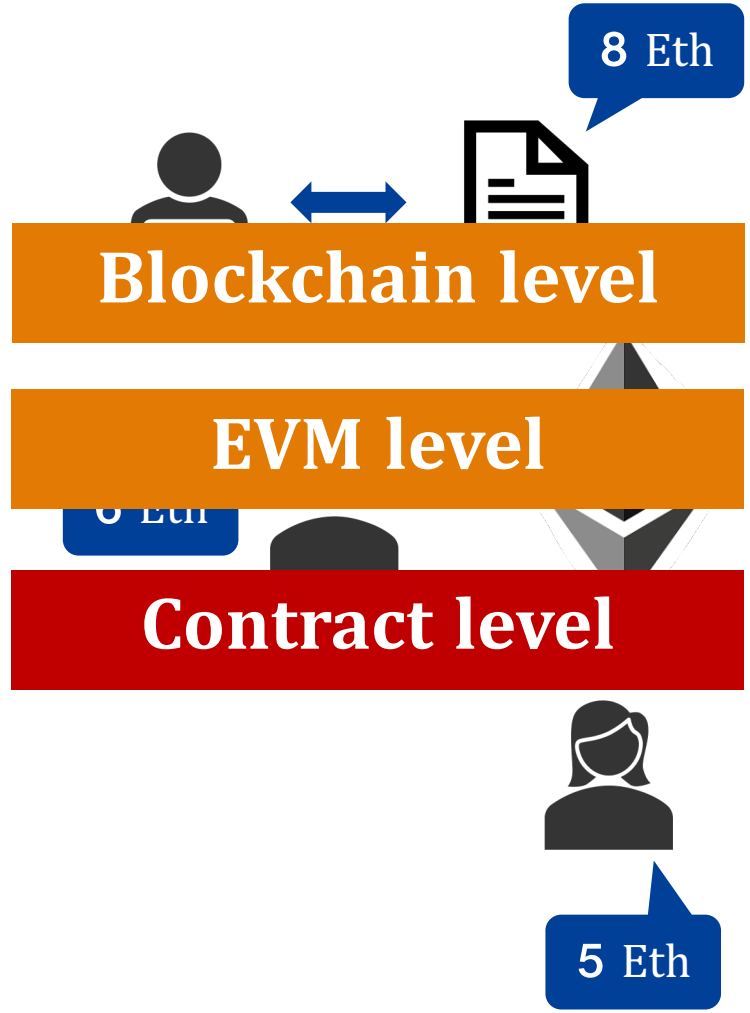
Ethereum is the second-largest blockchain system

In General

- A **programmable** blockchain
- A platform for **decentralized** applications.

In Detail

- A transaction-based **state machine**
- The heart is **Ethereum Virtual Machine** (EVM)
- Based on Turing-complete programming language (Solidity)



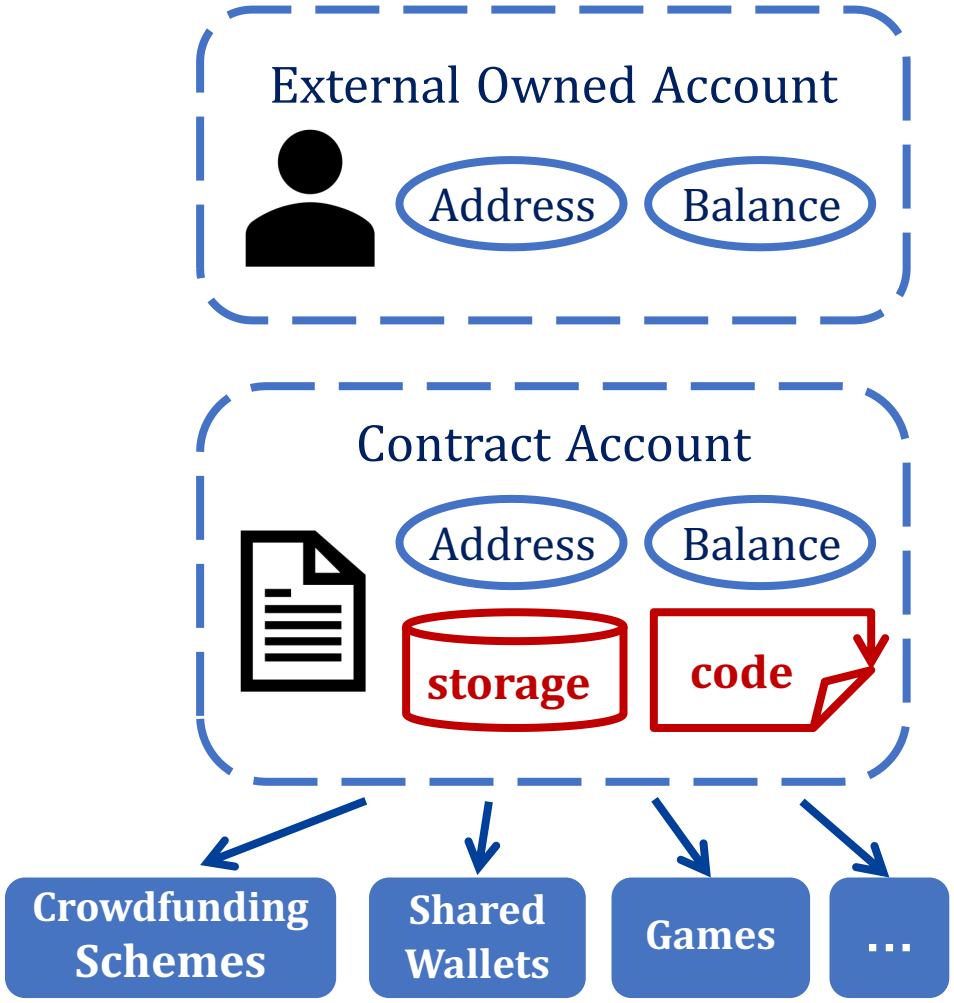
Smart Contract

Contract Code

- Source code written in **Solidity**
- Compiled by ***Solc*** to get bytecode
- Bytecode run on **EVM**

Contract Action

- Created by External Owned Account
- Executed on incoming transactions

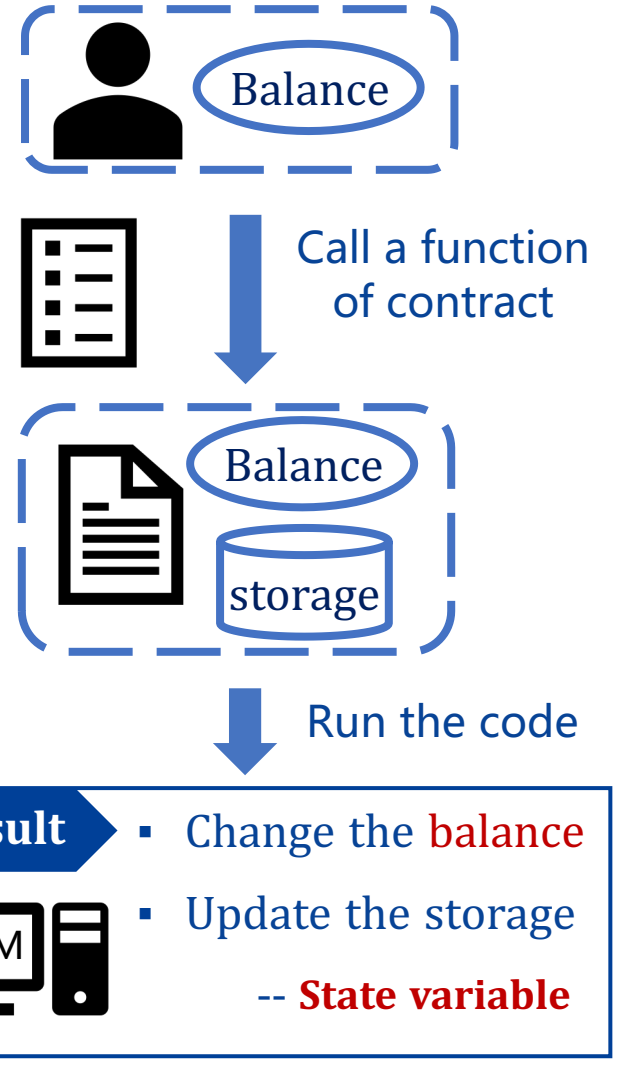


Transaction

Basic Fields

- **From:** Sender's Address
- **To:** Receiver's Address
- **Value:** Amount of Currency
- **Data:** Various situations
  - Empty (just transfer currency)
  - Init code of contract
  - **Called function with arguments**

Simulate a scene



Exploitation of Smart Contract

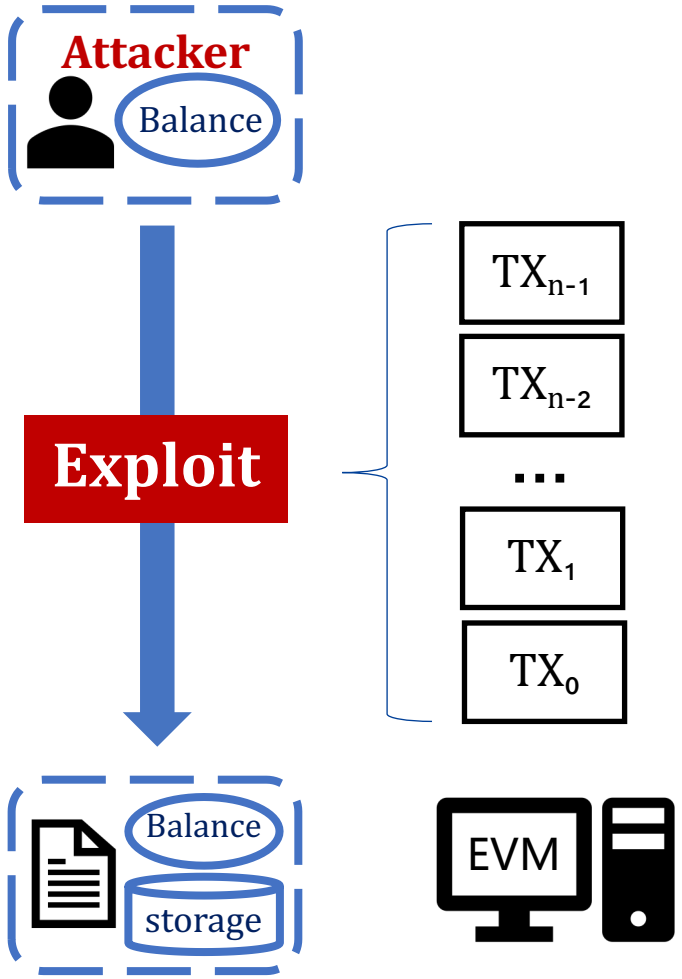
What is the exploitation

- From attacker to target contract
- A sequence of transactions

Categories of exploitation

According to the cause of damages:

- Balance Increment
- Self-destruction
- Code Injection



Exploitable Vulnerabilities

Unchecked Transfer Value

- Misuse of `this.balance`
- Unlimited profit

Vulnerable Access Control

- Missing & misuse of check
  - **before sensitive operation**

Exposed Secret

- **Newly identified vulnerability**
- **Previous tools cannot exploit**

```
function withdraw() notOnPause public {
    if (block.timestamp >= x.c(msg.sender
        ) + 10 minutes) {
        uint _payout = (x.d(msg.sender).
            mul(x.getInterest(msg.sender))
            .div(10000)).mul(
            block.timestamp.sub(x.c(msg.
            sender))).div(1 days);
        x.updateCheckpoint(msg.sender);
    }
    if (_payout > 0)
        msg.sender.transfer(_payout);
}
```

Exploitable Vulnerabilities

Unchecked Transfer Value

- Misuse of `this.balance`
- Unlimited profit

Vulnerable Access Control

- Missing & misuse of check
  - **before sensitive operation**

Exposed Secret

- **Newly identified vulnerability**
- **Previous tools cannot exploit**

```
1  contract Game {
2      address questionSender;
3      string public question;
4      bytes32 responseHash;
5      function Try(string _response) external
           payable{
6          require(msg.sender == tx.origin);
7          if(responseHash == keccak256(_response)
               && msg.value>1 ether){
8              msg.sender.transfer(this.balance);
9          }
10     }
11     function StartGame(string _question,string
           _response) public payable {
12         if(responseHash==0x0){
13             responseHash = keccak256(_response);
14             question = _question;
15             questionSender = msg.sender;
16         }
17     }
18 }
```

Secret checker

Secret setter

Exploitable Vulnerabilities

Unchecked Transfer Value

- Misuse of `this.balance`
- Unlimited profit

Vulnerable Access Control

- Missing & misuse of check
  - **before sensitive operation**

Exposed Secret

- **Newly identified vulnerability**
- **Previous tools cannot exploit**

```
1  contract Game {
2      address questionSender;
3      string public question;
4      bytes32 responseHash;
8                  msg.sender.transfer(this.balance);
13                 responseHash = keccak256(_response);
14                 question = _question;
15                 questionSender = msg.sender;
16             }
17         }
18 }
```

Secret checker

Attackers inspect the secret from the data of previous transactions

Attackers break the secret checking to gain profit

Secret setter

Contents

Goal of the Work

- Unchecked Transfer Value
- Vulnerable Access Control
- Exposed Secret

→ **Vulnerabilities Detected**

↓

Vulnerabilities Exploited

{ Fuzzing → Efficient Exploit Generation }

Challenges of Exploit Generation

Challenge-1: Unsolvable Constraint

< Situation in smart contract >

Condition restricting sensitive operations

- Involve complicated operation like **hash**

```
function Try(string _response) external
    payable{
    require(msg.sender == tx.origin);
    if(responseHash == keccak256(_response)
        && msg.value>1 ether){
        msg.sender.transfer(this.balance);
    }
}
```

< Previous solution>

Previous tools (e.g., `Teether`, `Mythril`) rely on **SMT solver**

- **Cannot solve cryptographic constraint**
- **Ignore the runtime value**
  - **not stored in contract state**

Challenges of Exploit Generation

Challenge-2: Blockchain Effects

< Situation in smart contract >

Blockchain effects of blockchain system affect the execution of smart contracts

- E.g., blockchain properties

```
function withdraw() notOnPause public {
    if ( block.timestamp >= x.c(msg.sender) + 10 minutes) {
        uint _payout = (x.d(msg.sender).mul(x.getInterest(
            msg.sender)).div(10000)).mul( block.timestamp .
            sub(x.c(msg.sender))).div(1 days);
        x.updateCheckpoint(msg.sender);
    }
    if (_payout > 0)
        msg.sender.transfer(_payout);
```

< Previous solution>

Previous tools have difficulties on manipulating blockchain effect:

- **Lack of considering the syntax of blockchain properties**

  e.g., invalid timestamp

- **Ignore the possibility of call reverting, thus lose coverage**

  e.g., Teether, ContractFuzzer

Our Solution

Fuzzing — **EthPloit: a smart contract specific fuzzer**

Feedback of runtime value

Record the runtime values of arguments and variables

- Create a blank seed set
- Update the seed set
- Use for the next generation

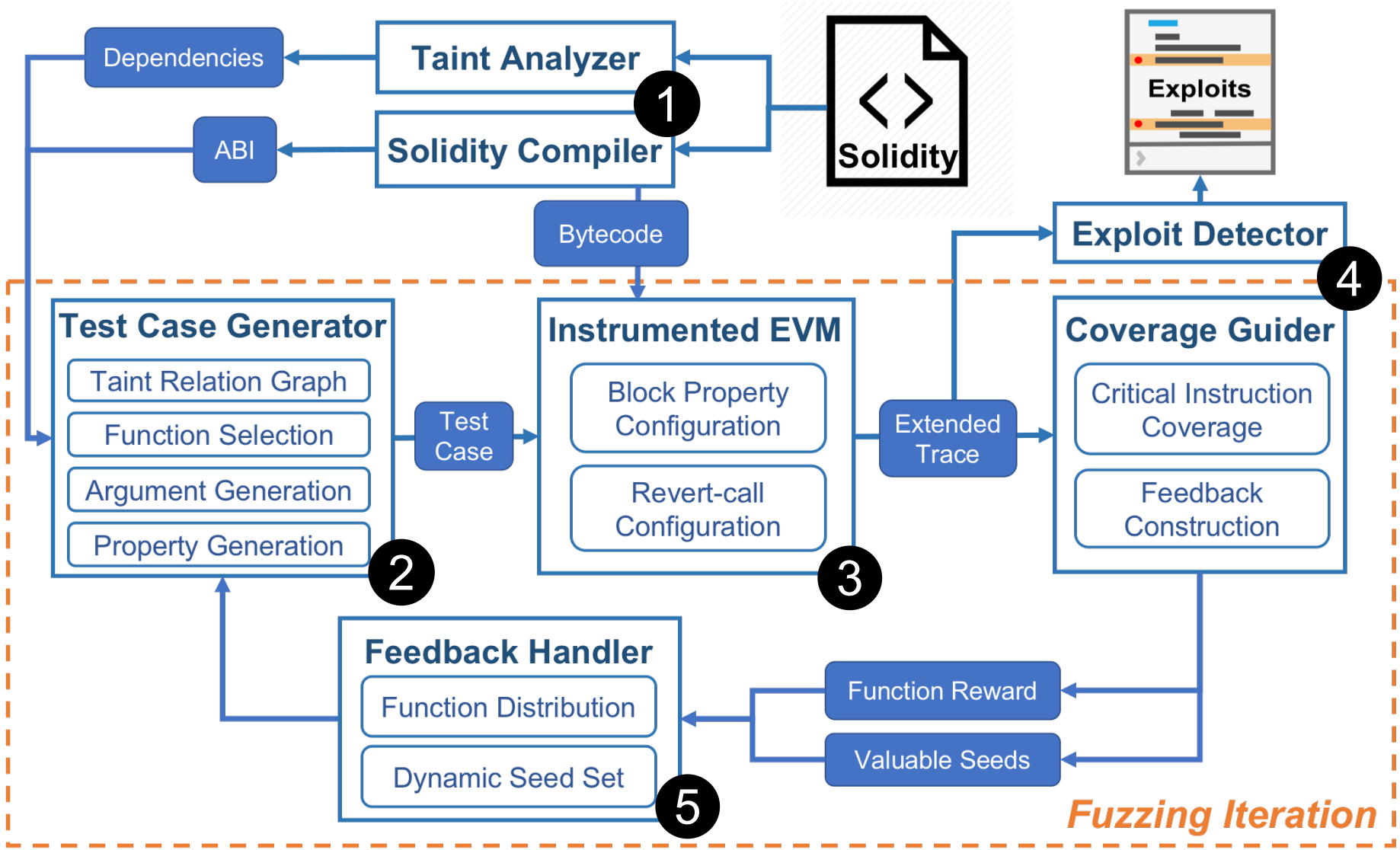
Indicated information:

- **Execution history**
  - **e.g., the hash image**
- **State of the contract**
  - **i.e., the state variable**

Manipulation of blockchain execution — **By instrumenting the execution environment**

Contents

Workflow of EthPloit



Workflow

1 Taint Analyzer

Knowledge of dependencies of modifying contract state improves **fuzzing efficiency**

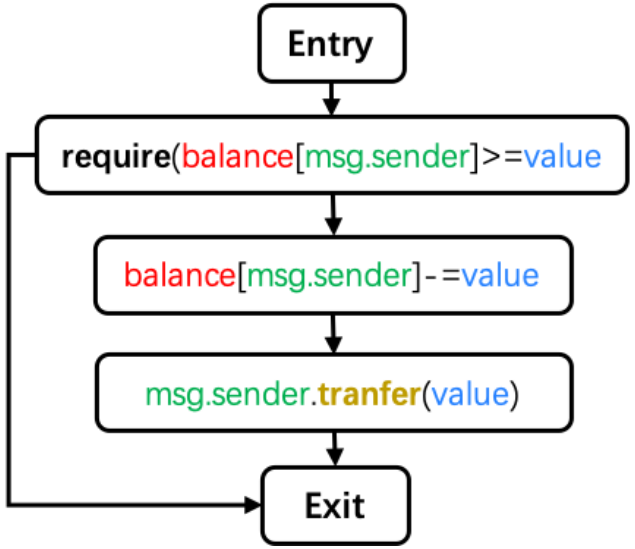
EthPloit applies **static taint analysis** to discover dependencies of modifying contract states:

- Generate control flow graph
- Label taint sources and sinks
- Perform taint propagation

Extract variable-level dependencies

- **Variable-Data** Dependency
- **Variable-Control** Dependency

```
function test(uint value)public
{
    require(balance[msg.sender]
        >=value);
    balance[msg.sender]-=value;
    msg.sender.transfer(value);
}
```



1 Taint Analyzer

Workflow

Knowledge of dependencies of modifying contract state improves **fuzzing efficiency**

EthPloit applies **static taint analysis** to discover dependencies of modifying contract states:

- Generate control flow graph
- Label taint sources and sinks
- Perform taint propagation

Extract variable-level dependencies

- **Variable-Data** Dependency
- **Variable-Control** Dependency

Variable-Data

```
msg.sender taint balance
value      taint balance
```

```
msg.sender taint transfer
value      taint transfer
```

Variable-Control

```
msg.sender crtl transfer
value      ctrl transfer
balance    ctrl transfer
```

❷ Test Case Generator

Optimize the test case by analyzing how inputs affect the execution of exploits

Taint Relation Graph

Extend in-function dependencies to **dependencies among functions**

Function Selection

- Add suitable functions into a set of candidates
- Select function from candidates based on **probability distribution**

Arguments Generation

- From pseudo-random generator
- From **dynamic seed set**

Blockchain Properties Generation

Based on **Instrumented EVM Environment**

3 Instrumented EVM Environment

EthPloit environment

- Based on `remix-debugger`
- Deploy contract
- Execute transaction
- **Extract full execution trace**

Compared to private Ethereum chain

- More light-weight
- More flexible for configure

Three instrumentations

Configure accounts

- For each test case

Configure block properties

- For each execution of transaction

Force external calls to revert

- For each external call
- Revert the 2nd execution of call

4 Trace Analyzers

Coverage Guider

Measure the progress of exploit-oriented fuzzing

Construct feedback as rewards

Critical instruction coverage

Feedback construction

- Seed feedback
- Function distribution feedback

$$P(f) = c_0 + \frac{N_c}{N_t}(c_1 - c_0)$$

Exploit Detector

Balance Increment oracle

- If attackers' balance is **increased**

Self-Destruction oracle

- If the opcode **SELFDESTRUCTION** is found

Code Injection oracle

- If opcodes **CALLCODE, DELEGATECALL** are found
- If destination is controlled by attackers

5 Feedback Handler

Dynamic Seed Strategy

Aim to **guide** the test case generator to produce proper function arguments

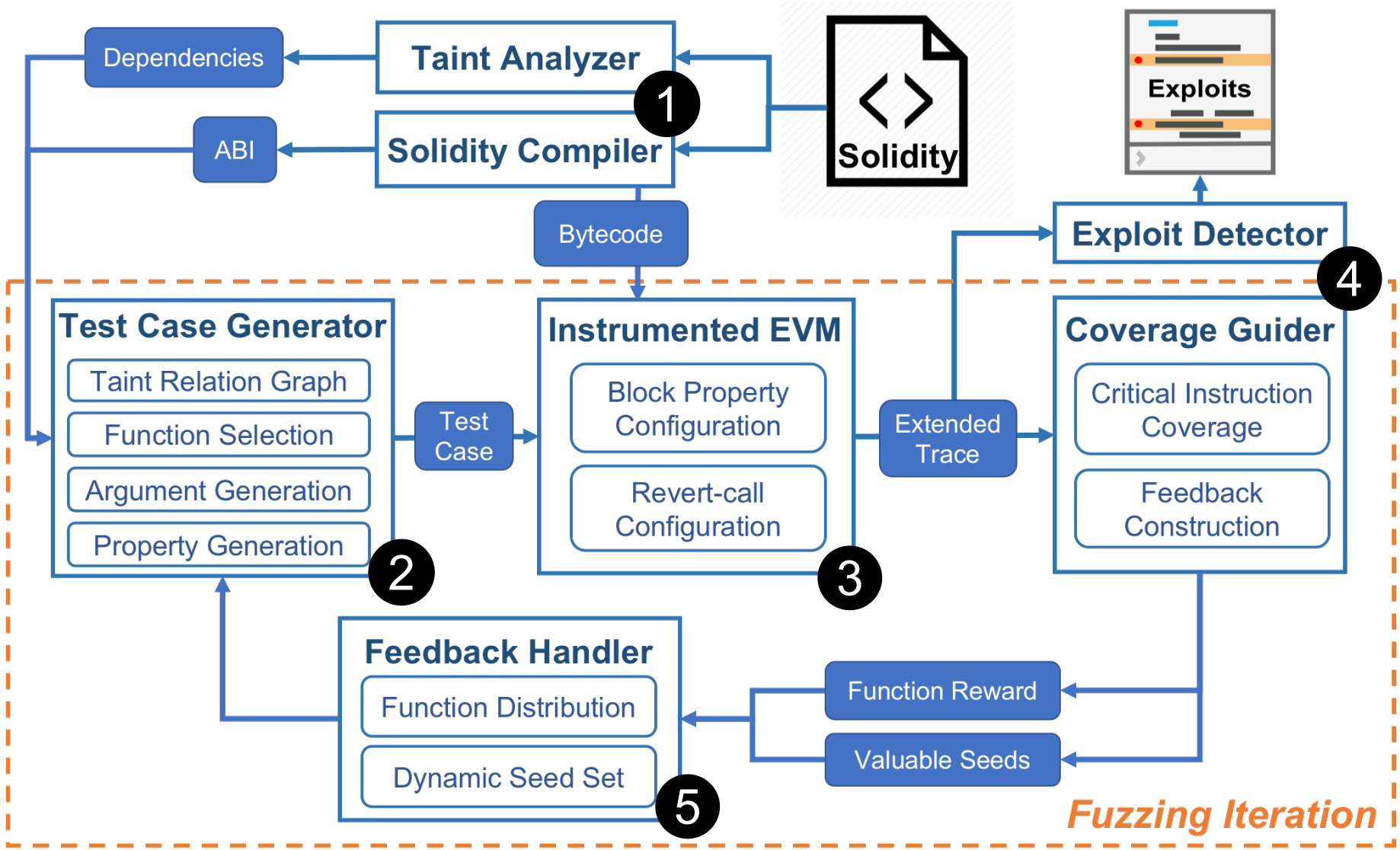
For the whole process of fuzzing

- Perform more mutation based on interesting cases
- Select **global seeds** which have a lifetime during fuzzing one contract
- **All arguments of interesting cases causing coverage increment**

For each test case

- Make use of connections among transactions
- Select **local seeds** after each execution of transaction:
  - Previous arguments
  - State variables
  - **I/O of complicated calls**
  - Constant values

Workflow of EthPloit



Contents

Environment

Dataset — **Totally 45,308 contracts**

Environment — Two 3.60GHz Xeon CPUs with 128GB RAM

Fuzzing Configuration

- Maximum test cases as **1,000**
- Maximum length as **3** for each case

Comparison — **Teether[1] and MAIAN[2]** with a timeout of 5 minutes

[1] Krupp, Johannes, and Christian Rossow. "teether: Gnawing at ethereum to automatically exploit smart contracts." 27th {USENIX} Security Symposium ({USENIX} Security 18). 2018.

[2] Nikolić, Ivica, et al. "Finding the greedy, prodigal, and suicidal contracts at scale." Proceedings of the 34th Annual Computer Security Applications Conference. 2018.

Evaluation of Contract Exploit

EthPloit

- Totally generated **644** exploits
- **No false positive,** verified using real-world EVM
- **600** Balance Increment, **59** Self-destruction, **4** Code Injection

Teether / MAIAN

- unable to analyze **5,123** contracts and **102** contracts
- Teether generated **14** false positive
- MAIAN cannot exploit lots of vulnerable contracts

Evaluation of Contract Exploit

Summary of exploits generated based on triggered vulnerabilities

Tools	Exposed Secret			Unchecked Transfer Value				Bad Access Control	Others	Total
	Cryptographic Checks	Others	Total	**Unlimited Profit**	Misused this.balance	Others	Total			
ETHPLOIT	**104**	8	112	**144**	181	26	351	142	39	644
teether	**0**	0	0	**30**	25	6	61	13	3	77
MAIAN	**0**	4	4	**31**	143	16	190	99	3	296

EthPloit

- For Exposed Secret, **104** out of **112** exploits have **cryptographic checks** in the execution path
- For Unchecked Transfer Value, **144** out of **351** exploits are caused by **Unlimited Profit**

Comparison

- EthPloit has huge advantage over teether and MAIAN
  - Especially in exploiting **Exposed Secret** and **Unchecked Transfer Value**

Evaluation of Contract Exploit

Summary of exploits generated based on two typical vulnerabilities

Tools	Cryptographic Checks	Unlimited Profit
EthPloit	104	144
Teether	0	30
MAIAN	0	31

Dynamic Seed Strategy:

- Fetch secret value
- Solve hash checks

Instrumented EVM Environment :

- simulate block properties
- Exploit lottery games
  - block properties as random seed

Impact of Vulnerabilities Identified

Information of typical contracts exploited by EthPloit

Contract Information				Exploit results		Number of Test Cases			
Contract	Address	#Tx	Highest Balance	Vulnerability	Teether/MAIAN	Normal	No EVM	No Seeds	No Taint
TestR	0xaf53...	6	0.5 ETH, $269.2	Exposed Secret	×/√	13.0	18.1	-	8.9
BLITZ_GAME	0x35b5...	4	6.0 ETH, $572.6	Exposed Secret	×/×	49.6	50.0	-	169.0
Who_Wants...	0xfc62...	10	4.0 ETH, $546.3	Exposed Secret	×/×	46.2	28.0	-	61.5
Game	0xe37b...	6	3.0 ETH, $445.9	Exposed Secret	×/×	50.2	37.8	-	65.5
GPUMining	0xa965...	346	1.2 ETH, $712.3	Unchecked Transfer Value	×/×	188.1	660.6	319.7	332.9
HRKD	0x0a70...	307	50.1 ETH, $11k	Unchecked Transfer Value	×/×	48.4	-	29.2	20.1
Slotthereum	0xb43b...	76	0.4 ETH, $92.4	Unchecked Transfer Value	×/×	52.9	87.4	214.6	57.2
Divs4D	0x3983...	161	4.1 ETH, $905.3	Unchecked Transfer Value	×/×	10.7	-	18.9	29.1
DailyRoi	0x77e4...	4,488	397.1 ETH, $87k	Unchecked Transfer Value	×/×	11.6	-	10.3	10.7
Dividend	0xe3ac...	47	140.5 ETH, $66k	Unchecked Transfer Value	×/√	134.7	47.8	-	333.3
HOTTO	0x612f...	132	1.1 ETH, $320.1	Bad Access Control	×/√	18.2	23.8	-	15.3
Crypto...Network	0x781f...	52K	1.3 ETH, $541.8	Bad Access Control	×/√	28.8	40.0	21.4	89.7

Exposed Secret exploited in total: 32 contract, lost 37.3 ETH, about $6,485

Unchecked Transfer Value & Vulnerable Access Control affect lots of widely used contracts, e.g., DailyRoi: 4,888 transactions, maximum balance of 397.1 ETH ($87k)

Evaluation of Core Techniques

Benchmarks : newly discovered 554 exploitable contracts

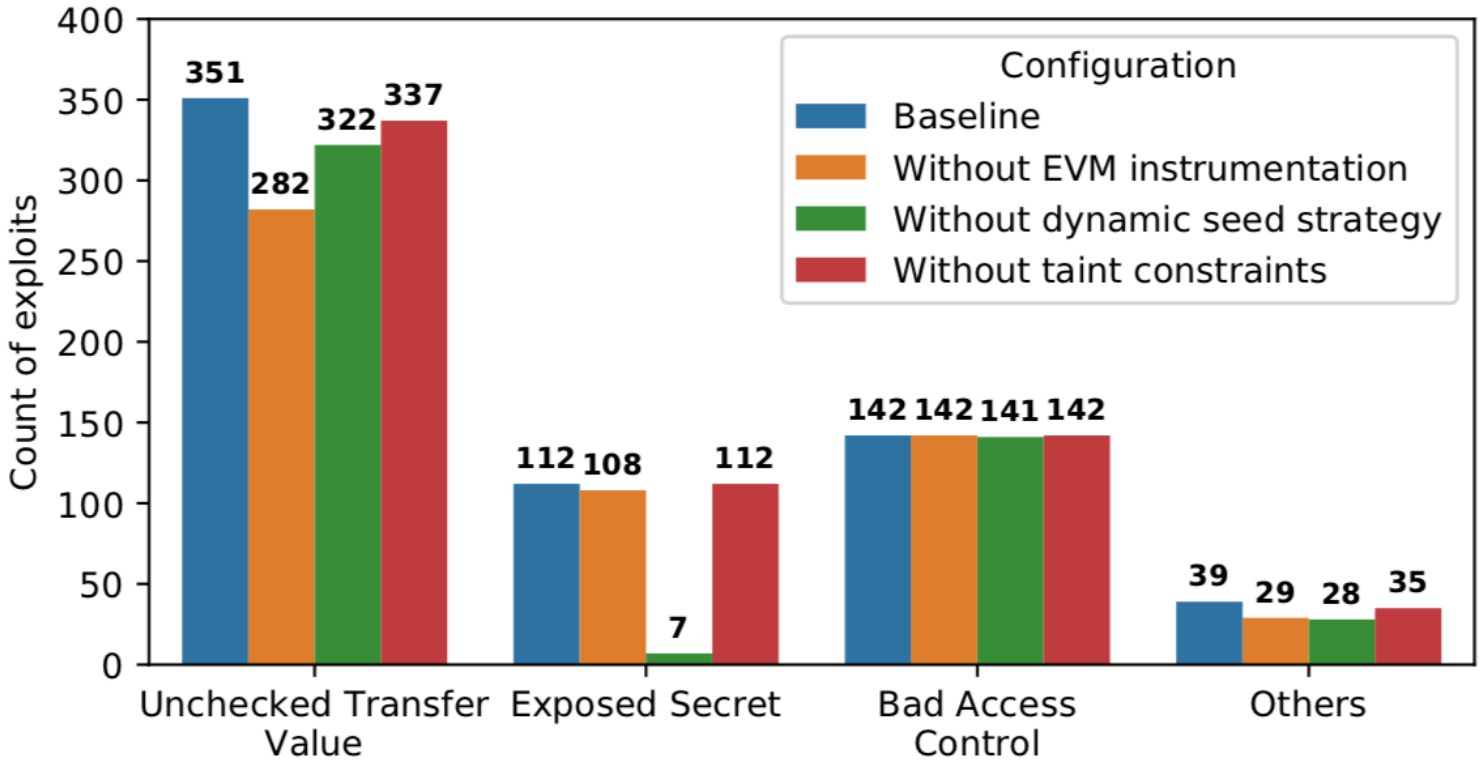
Four different configuration of EthPloit:

- 1 Without EVM instrumentation
- 2 Without dynamic seed strategy
- 3 Without taint constraints
- **Baseline:** All techniques are enabled

Benchmark is tested for 10 times under each configuration, respectively

Evaluation of Core Techniques

Number of generated exploits under various configuration

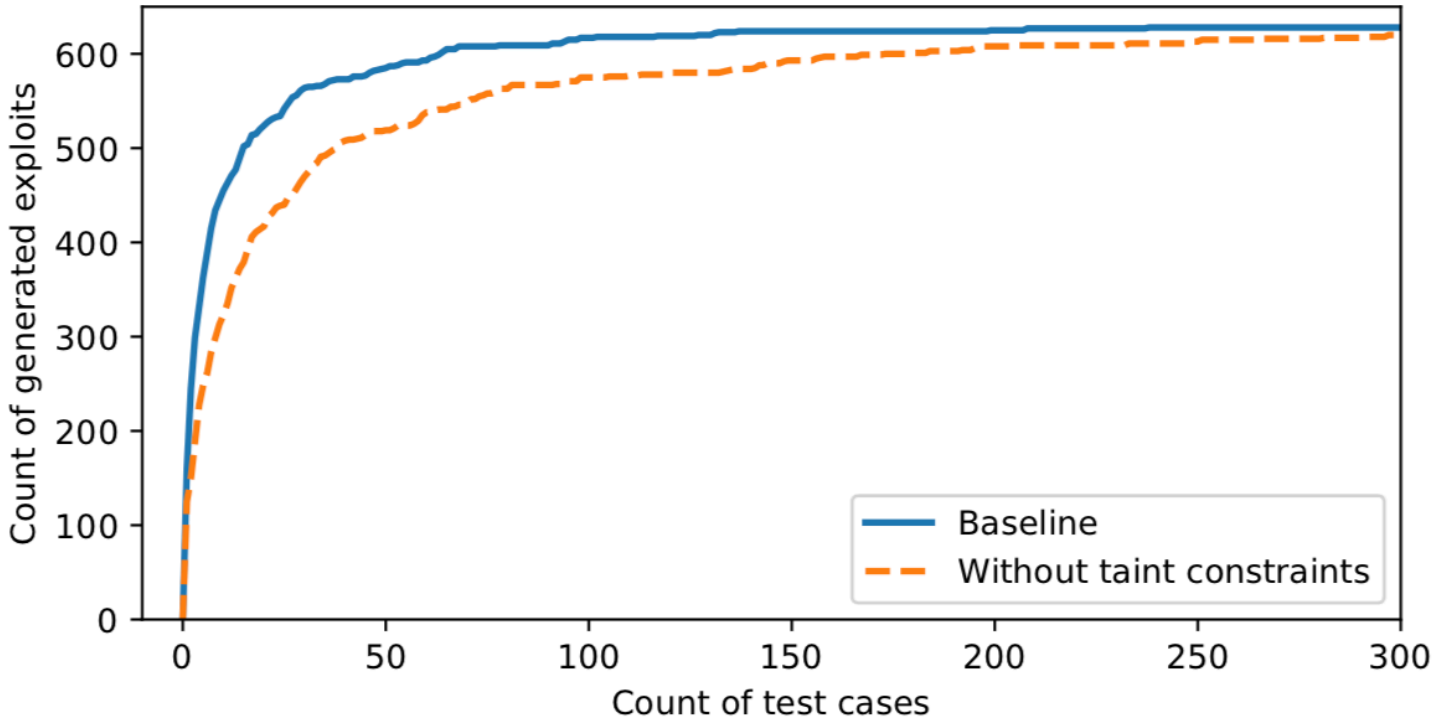


Without dynamic seed strategy — *EthPloit* miss **most** Exposed Secret

Without EVM instrumentation — *EthPloit* miss **69** Unchecked Transfer Value

Evaluation of Core Techniques

Count of exploits with regards to count of test cases



Use the number of test cases to represent fuzzing efficiency

- The overall fuzzing efficiency is damaged when taint analysis is removed
- With taint constraints, over **90%** exploits can be found in **100** test cases

Contents

Conclusion

Design EthPloit

1. Automatically generate exploits of contracts
2. Deploy light-weight approaches to solve:
   - Unsolvable Constraints
   - Blockchain Effects
3. Fuzz **45,308** contracts in real world
4. Introduce a new vulnerability: **Exposed Secret**

In memory of medical staff who bravely fight COVID

During the new coronavirus infection in 2020:

- Li Wenliang and 8 other doctors died of illness
- More than 3,000 health workers infected

Pay the highest respect to all the medical staff !

Thank you & Question ?