# EOSAFE: Security Analysis of EOSIO Smart Contracts

**Ningyu He*[1]**, Ruiyi Zhang*[2], Haoyu Wang[3], Lei Wu[4], Xiapu Luo[5], Yao Guo[1], Ting Yu[6], Xuxian Jiang[2]

[1] Key Lab on HCST (MOE), Peking University, China
[2] PeckShield, Inc.
[3] Beijing University of Posts and Telecommunications, China
[4] School of Cyber Science and Technology, Zhejiang University, China
[5] The Hong Kong Polytechnic University, China
[6] Qatar Computing Research Institute, Qatar

PEKING UNIVERSITY

# Motivation

## EOSIO market cap & DApp transactions

- EOSIO adopts **DPoS** consensus, which supports **higher TPS**
- Highest market cap: nearly **18 billion USD** in Apr. 2018
- The DApp ecosystem was extremely active


EOS (EOS), the blockchain platform well-known for its delegated proof-of-stake (DPoS) consensus model which works to speed up transactions and block creation at the expense of some level of decentralization, has recently surpassed Ethereum (ETH) in daily decentralized application (dApp) users and transaction volume.

*https://defirepublic.com/eos-surpasses-ethereum-in-daily-dapp-users-and-transaction-volume/*

# Attack Events

## EOSIO smart contracts have been attacked

| Date | Victim | Vulnerability | Financial Loss | Reference |
|---|---|---|---|---|
| **Oct. 2018** | EOSCast | *Fake EOS* | 60K EOS | *https://blog.peckshield.com/2018/11/02/eos/* |
| **Oct. 2018** | EOSBet | *Fake Receipt* | 140K EOS | *https://blog.peckshield.com/2018/10/26/eos/* |
| **Dec. 2018** | EOS BetDice | *Rollback* | 195K EOS | *https://hacked.slowmist.io/* |

**Representative** attack events against EOSIO smart contracts

**More than *100 attacks* according to our collected dataset.**

# Vulnerability Detectors

## Tools in Ethereum & EOSIO

- There are dozens of existed tools targeting on automatically detecting vulnerabilities in Ethereum smart contracts, like Osiris (https://github.com/christoftorres/Osiris), Securify2 (https://github.com/eth-sri/securify2), teEther (https://github.com/nescio007/teether) and so on.

- However, there are few tools that are able to identify vulnerabilities in EOSIO smart contracts

# Background

## Dispatcher: `apply` function

- EOSIO smart contracts is written in C++ and compiled into Wasm bytecode
- The `apply` is a dispatcher, handling received *invocation* or *notification*
- The signature of `apply` is fixed
  - *receiver*: current account
  - *code*: the account in which the code is executed.
  - *action*: name of invoked function

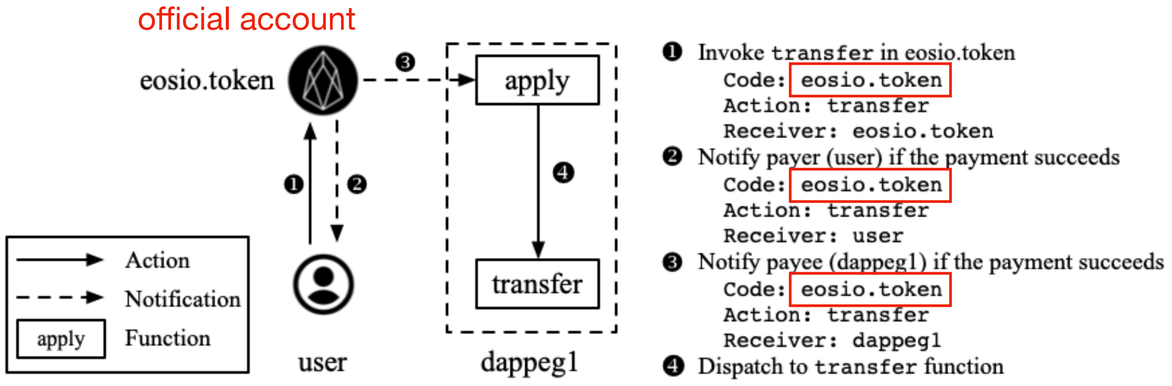
```
void apply (uint64_t receiver, uint64_t code, uint64_t action) {
    if (action == N(onerror)) {
        check(code == N(eosio), " exception captured" );
    }
    auto self = receiver;
    if ((code == self || code == N(eosio.token))) {
        switch (action) {
            case N(transfer): // action == N(transfer)
            // deal with:
            // 1. direct invocation to transfer function
            // 2. notification emitted from transfer ...
        }
    }
}
```

Source code of dispatcher `apply`

# Background

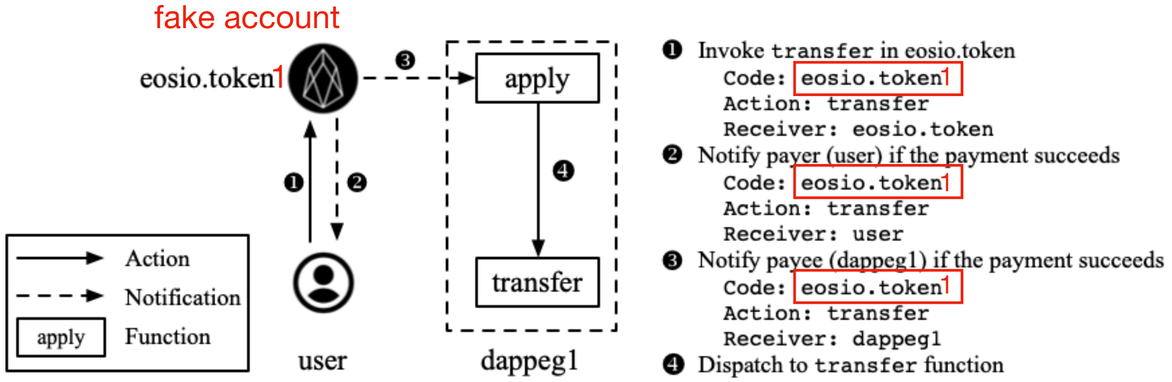
## Transfer official/fake EOS in EOSIO

- STEP 1: `user` invokes the `transfer` in eosio.token to make a request
- STEP 2&3: eosio.token will *notify* both of them if the request is approved
- STEP 4: the `apply` in `dapp` will dispatch the notification to its `transfer`




Normal situation

- STEP 1: `user` invokes the `transfer` in eosio.token1 to make a request
- STEP 2&3: eosio.token1 will *notify* both of them if the request is approved
- STEP 4: the `apply` in `dapp` will dispatch the notification to its `transfer`. However, the fake EOS is worthless




Fake EOS attack

# Goal

## Concerned vulnerabilities

| Vulnerability | Root Cause | Consequence |
|---|---|---|
| **Fake EOS** | Lack of verification on `code` | The malicious user could bypass the verification with worthless token |
| **Fake receipt** | Lack of verification on `to` in `transfer` | The malicious user could fool the victim without any cost |
| **Rollback** | Adopting problematic *bet-reveal* model in gambling smart contract | The malicious user could forecast the random number in advance |
| **Missing permission check** | Lack of verification on carried authority table | The malicious user can perform behaviors on behalf of others without authority |

**We concerned four types of vulnerabilities that are fully imported by programming error in smart contracts**

# Challenge #1

**Existing tools on detecting vulnerabilities in Ethereum smart contracts cannot be directly adopted to detect vulnerabilities in EOSIO smart contracts**

| | Ethereum (EVM bytecode) | EOSIO (Wasm bytecode) |
|---|---|---|
| **Amount of Instructions** | 154<br>(32 PUSH, 16 DUP, 16 SWAP) | 172 |
| **Types of Instructions** | - | - Float number related instructions<br>- Type cast related instructions<br>- `br_table` instruction |
| **Control Flow** | Relative simple | *Multi-level nested loops* |
| **Complexity of Vulnerabilities** | Do not rely on too much semantic information, like reentrancy, overflow, blockchain-state dependency vulnerabilities | *Heavily rely on semantic information*, like fake EOS, fake receipt and rollback vulnerabilities |

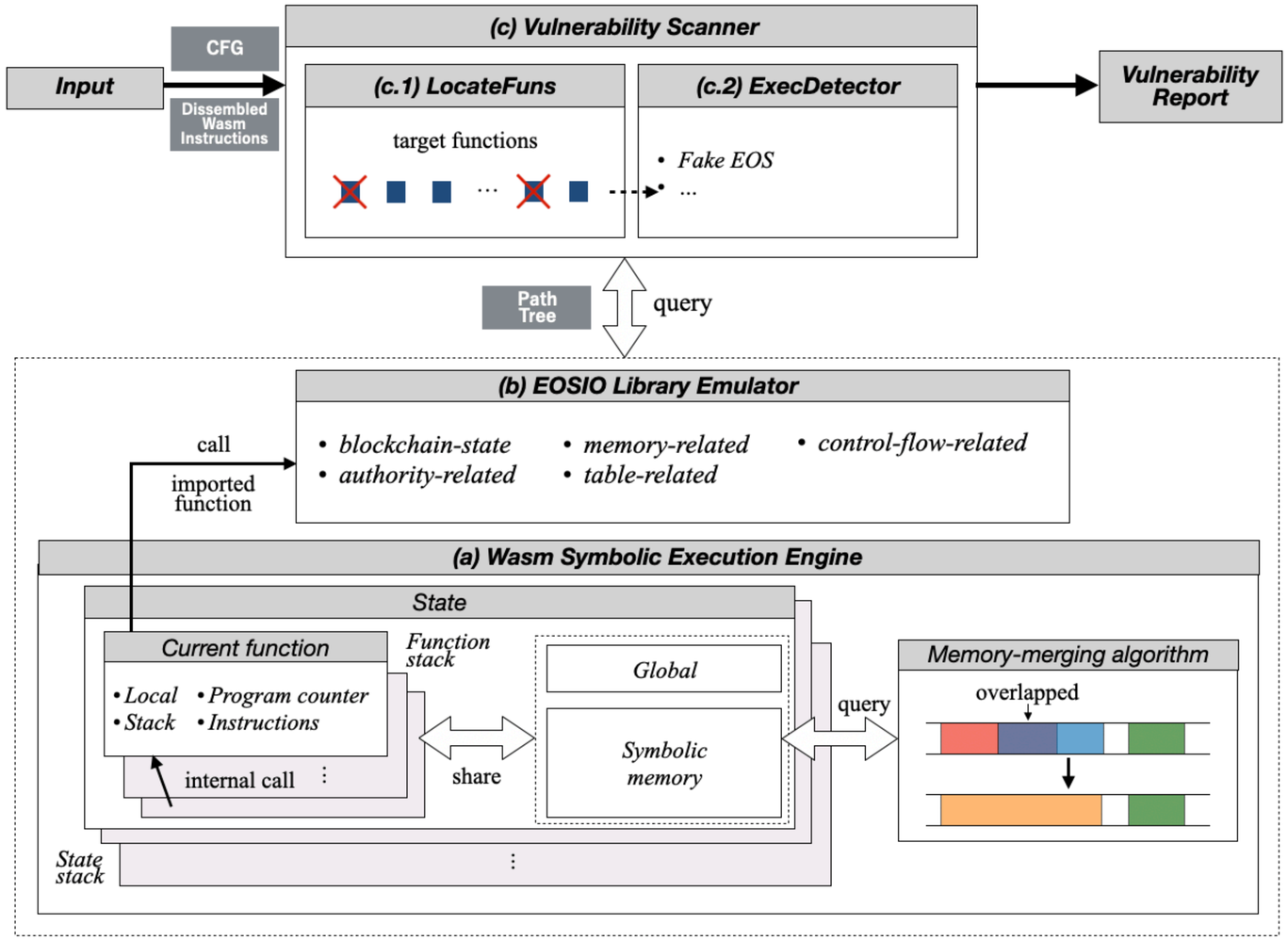
# Challenge #2

**Symbolic execution technology would bring in some typical challenges**

- **Memory modeling**: if the address is a symbol, how to effectively and precisely retrieve and store data?
- **External dependency**: the source code, even the bytecode, of the library functions cannot be obtained, how to emulate these functions' behavior?
- **Path explosion**: the path explosion can significantly consume the resources and increase the analyzing time, especially when encountering the multi-level nested loops.

# Design of EOSAFE

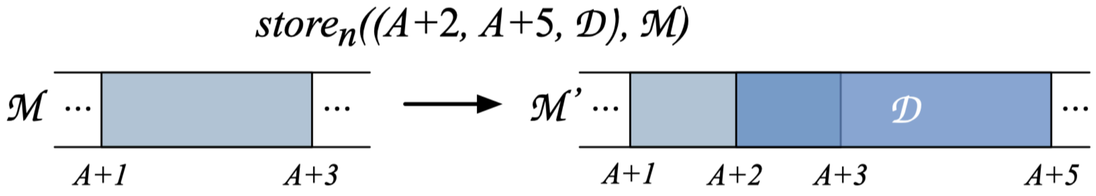
- **Wasm Symbolic Execution Engine**
- **EOSIO Library Emulator**
- **Vulnerability Scanner**



# Design of EOSAFE

## Wasm Symbolic Execution Engine

- Implementation:
  - We implement *all 172 instructions* with the help of Bit Vector in z3
  - Each feasible path corresponds to a state, maintaining the data structure in Wasm
- How to address **path explosion**?
  - We adopt the `timeout` and `depth` option
  - *Simple but efficient*
- How to address **memory modeling**?
  - Using a mapping structure to mimic the sparse linear array
  - We propose a *memory-merging algorithm* to overcome the *memory overlapped problem* within this structure




Interval (A+2, A+3) is overlapped after applying the store instruction

# Design of EOSAFE

## EOSIO Library Emulator

- To address the **external dependency**:
  - We divide commonly used external functions into five categories
  - We manually model these functions behavior

**Library functions are divided into categories.**

| Category | Imported Function Examples |
|---|---|
| blockchain-state | tapos_block_num |
| | current_time |
| memory-related | memcpy |
| | memmov |
| control-flow-related | eosio_exit |
| | eosio_assert |
| authority-related | require_auth |
| | require_auth_2 |
| table-related | db_get_i64 |
| | db_update_i64 |

# Design of EOSAFE

## Vulnerability Scanner

- Each vulnerability scanner consists of two steps:
  - locating suspicious functions
  - detecting vulnerabilities
- We implement four types of scanners corresponds to vulnerabilities we concerned
- This design means good scalability

$$Eeq^{i}_{\texttt{apply}}(action, \texttt{"transfer"}) \wedge$$
$$\forall a \in (\mathcal{A} - \{self\}).\neg Eeq^{i}_{\texttt{apply}}(code, a)$$

The formal definition of fake EOS vulnerability detection logic. Specifically, if a path can lead to the transfer function and does not check the `code` field, we regard this contract is vulnerable to fake EOS vulnerability.

# Evaluation

## Benchmark & Comparison with the KLEE solution

**A Comparison of EOSAFE and KLEE on the benchmark. (TP – True Positive, FP – False Positive, TN – True Negative, FN – False Negative, SR – Success Rate, P – Precision, R – Recall, F1 – F1 Measure)**

| Type | # Samples(Vul/Non-Vul) | EOSAFE | | | | | | | | KLEE | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | TP | FP | TN | FN | SR** | P | R | F1 | TP | FP | TN | FN | SR** | P | R | F1 |
| Fake EOS | 14 (7/7) | 7 | 0 | 7 | 0 | 100.00% | 100.00% | 100.00% | 100.00% | 5 | 0 | 7 | 2 | 50.00% | 100.00% | 71.43% | 83.33% |
| Fake Receipt | 10 (5/5) | 5 | 0 | 5 | 0 | 100.00% | 100.00% | 100.00% | 100.00% | 0 | 0 | 5 | 5 | 0.00% | - | - | - |
| Rollback | 18 (9/9) | 8 | 0 | 9 | 1 | 94.44% | 100.00% | 88.89% | 94.12% | 0 | 0 | 9 | 9 | 0.00% | - | - | - |
| Permission | 10 (6/4)* | 6 | 0 | 4 | 0 | 100.00% | 100.00% | 100.00% | 100.00% | 5 | 0 | 4 | 1 | 90.00% | 100.00% | 83.33% | 90.91% |
| ***Total*** | **52 (27/25)** | **26** | **0** | **25** | **1** | **98.08%** | **100.00%** | **96.30%** | **98.11%** | **10** | **0** | **25** | **17** | **30.77%** | **100.00%** | **37.04%** | **54.05%** |

* 4 pairs of the missing permission check samples are manually crafted. ** Timeout or memory error caused path explosion will be regarded as failed cases.

- From official announcement, cryptocurrency forums, and technical articles and blogs, we collected *52 victim smart contracts, which are vulnerable to at least one of our concerned vulnerabilities, as benchmark*
- The benchmark is released at: ***https://github.com/HNYuuu/EOSafe-bench mark***

- We compare EOSAFE with KLEE, which takes converted C code as input:
  - EOSAFE outperformed KLEE significantly, especially in recall
  - KLEE failed to perform the detection in most cases due to path explosion
  - The conversion from Wasm to C code requires extra human efforts on each case

# Evaluation

## On dataset

**Vulnerability detection results in the wild.**

| Type | # Candidates | # Vulnerable (%*) | # Unique | # Vulnerable (%*) |
|---|---|---|---|---|
| Fake EOS | 47,396 | 1,457 (2.71%) | 4,678 | 272 (4.88%) |
| Fake Receipt | 47,396 | 7,143 (13.31%) | 4,678 | 2,192 (39.33%) |
| Rollback | 17,394 | 1,149 (2.14%) | 913 | 84 (1.51%) |
| Permission | 53,666 | 8,373 (15.60%) | 5,574 | 662 (11.88%) |
| ***Total*** | **53,666** | **13,752 (25.63%)** | **5,574** | **2,759 (49.50%)** |

*The percent is calculated based on all the EOSIO smart contracts with their versions.

- We collected all 53,666 smart contracts corresponding to 5,574 accounts as of Nov. 15th, 2019
- The result suggests:
  - Nearly half of smart contracts were once vulnerable to at least one kind of vulnerability, fake receipt is the most widespread one.
  - As for affected versions, missing permission check vulnerability impacts more than others.

# Evaluation

## Time to fix vulnerability

The time to fix the vulnerabilities.

| Type | # Unique (Vul) | # Latest with Vul (%) | # Patched (%) | Patch Time |
|---|---|---|---|---|
| Fake EOS | 272 | 207 (76.10%) | 65 (23.90%) | 14.85d |
| Fake Receipt | 2,192 | 1,735 (79.15%) | 457 (20.85%) | 24.01d |
| Rollback | 84 | 28 (33.33%) | 56 (66.67%) | 4.24d |
| Permission | 662 | 313 (47.28%) | 349 (53.72%) | 4.38d* |
| ***Total*** | **2,759** | **2,080 (75.39%)** | **679 (24.61%)** | **16.84d** |

*The average patch time for missing permission check is calculated on the action level.

- On average, patching a vulnerability requires **more than two weeks**
- The rollback vulnerability is patched most timely, because it has strong connections with the gambling DApps
- More than **3/4** of smart contracts were still vulnerable, especially the fake receipt vulnerability

# Take-away messages

- We proposed the **first symbolic execution engine** for Wasm native code
- Based on the engine, we implemented an accurate and scalable framework: EOSAFE, which **outperforms the KLEE solution** on detecting vulnerabilities in EOSIO smart contracts
- We constructed and **released a manually checked benchmark**. Evaluation on the benchmark illustrates **promising performance** of EOSAFE
- We applied EOSAFE on over 53K smart contracts, which reveals **serious security issues** in the ecosystem, e.g., over 25% of the smart contracts are vulnerable

# Thanks!

Ningyu HE: *ningyu.he@pku.edu.cn*