# sGUARD : Towards Fixing Vulnerable Smart Contracts Automatically

Tai D. Nguyen, Long H. Pham, Jun Sun
Singapore Management University, Singapore

**2021 IEEE Symposium on Security and Privacy (S&P)**

# Background

- **Smart Contract**： implement a set of rules for managing digital assets in Ethereum accounts.

```
pragma solidity ^0.4.11;
contract MyContract {
    uint i = (10 + 2) * 2;
}
```

solidity source code

```
6060604052601860005534156010576fe5b5b603380601e6000396000f30060606040525bfe00a165627a7a72305820e8d51d91
f3af019d36e0e5d9d96443cdedaffd6764df9527ba3d510872b554f50029
```

bytecode

```
"opcodes":
    "PUSH1 0x60 PUSH1 0x40 MSTORE PUSH1 0x18 PUSH1 0x0 SSTORE CALLVALUE ISZERO PUSH1 0x10 JUMPI
INVALID JUMPDEST JUMPDEST PUSH1 0x33 DUP1 PUSH1 0x1E PUSH1 0x0 CODECOPY PUSH1 0x0 RETURN STOP PUSH1
0x60 PUSH1 0x40 MSTORE JUMPDEST INVALID STOP LOG1 PUSH6 0x627A7A723058 SHA3 0xe8 0xd5 0x1d SWAP2
RETURN 0xaf ADD SWAP14 CALLDATASIZE 0xe0 0xe5 0xd9 0xd9 PUSH5 0x43CDEDAFFD PUSH8 0x64DF9527BA3D5108
PUSH19 0xB554F500290000000000000000000000000000"
```

opcode

# Background

## Reentrancy vulnerability

when a smart contract C invokes a function of another contract D and subsequently a call back to contract C is made while it is in an inconsistent state

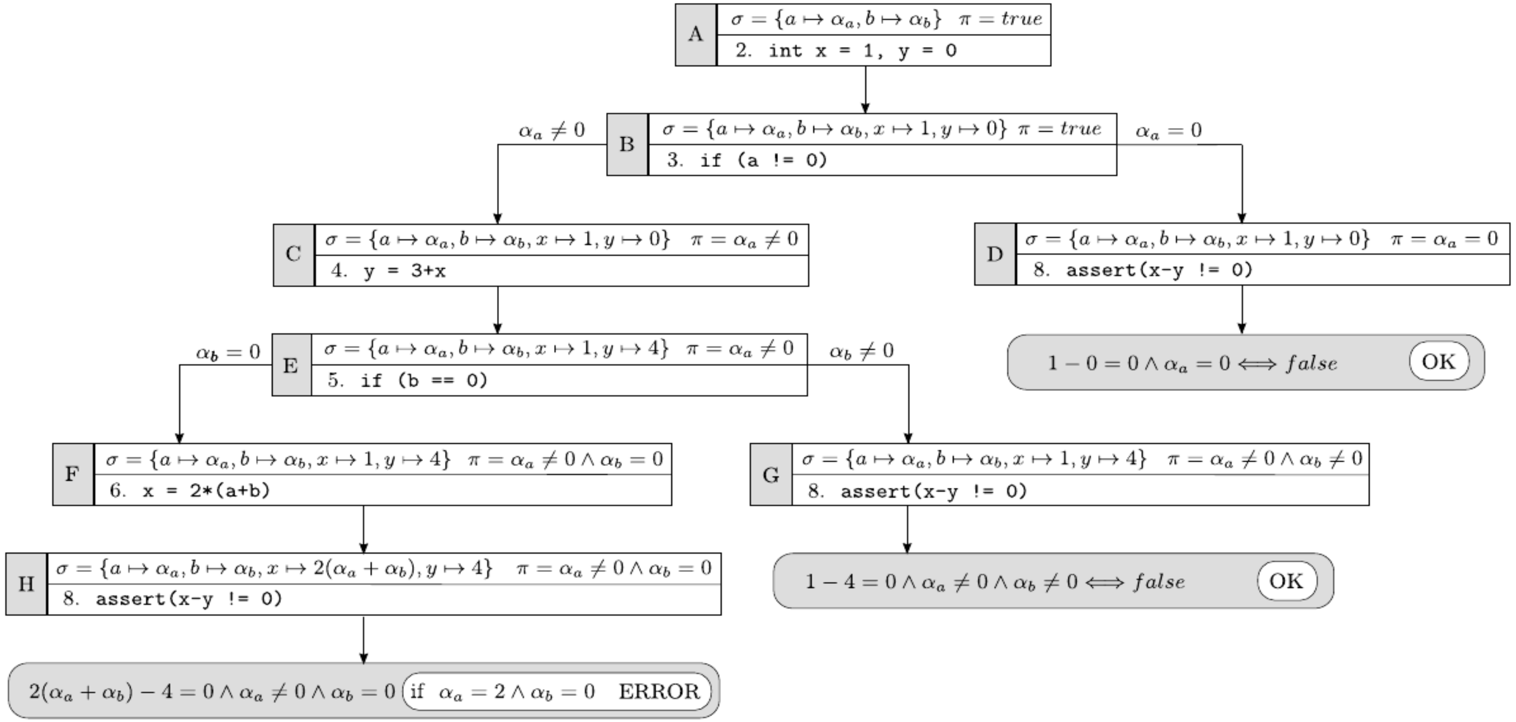
```
function getThisWeekBurnedAmount() public view returns(uint) {
  uint thisWeekStartTime = getThisWeekStartTime();
  uint total = 0;
  for (uint i = numOfBurns; i >= 1; i--) {
    if (burnTimestampArr[i - 1] < thisWeekStartTime) break;
    total += burnAmountArr[i - 1];
  }
  return total;
}

function getThisWeekBurnAmountLeft() public view returns(uint) {
  return weeklyLimit - getThisWeekBurnedAmount();
}

function burn(uint amount) external payable {
  require(amount <= getThisWeekBurnAmountLeft());
  require(IERC20(tokenAddress).transferFrom(msg.sender, BURN_ADDRESS, amount));
  ++numOfBurns;
}
```

# Background

## Symbolic execution

symbolic execution can explore multiple paths that a program could take under different inputs. The key idea is to allow a program to take *symbolic values* as inputs. Execution is performed maintaining for each explored control path a formula that describes the conditions satisfied by the branches taken along that path, and a symbolic memory store that maps variables to symbolic expressions or values

```
void aFunction(int a, int b) {
    int x = 1, y = 0;

    if(a != 0) {
        y = 3 + x;

        if(b == 0) {
            x = 2 * (a + b);
        }
    }

    assert(x - y != 0);
}
```



# sGUARD Overview

## Identify

a **finite** set of symbolic execution traces , static analysis (bytecode)

## Patch

specific fixing pattern, four types vulnerability (source code)

**Sound**

**Preciseness**

**Efficiency**

# Problem & Challenge

## Weak specifications

A test suite is taken as the correctness specification

## Time overhead

## Gas overhead

# Main Idea

- **Enumerating Symbolic Traces**
- **Dependency Analysis**
- **Fix the Smart Contract**

# Enumerate symbolic traces

**Symbolic trace** $\langle s_0, op_0, \cdots, s_n, op_n, s_{n+1} \rangle$

$\mathbf{s_i}$ $(pc_i, S_i^s, M_i^s, R_i^s)$

$\mathbf{op_i}$ $\Big\downarrow$

$\mathbf{s_{i+1}}$ $(pc_{i+1}, S_{i+1}^s, M_{i+1}^s, R_{i+1}^s)$

$$\frac{}{(pc, S, M, R) \rightsquigarrow \square}\ \text{STOP} \qquad \frac{S_1, x = S.pop()}{(pc, S, M, R) \rightsquigarrow (pc+1, S_1, M, R)}\ \text{POP} \qquad \frac{S_1, x = S.pop() \quad z = op(x) \quad S_2 = S_1.push(z)}{(pc, S, M, R) \rightsquigarrow (pc+1, S_2, M, R)}\ \text{UNARY-OP}$$

$$\frac{\begin{array}{c} S_1, x = S.pop() \quad S_2, y = S_1.pop() \\ z = op(x, y) \quad S_3 = S_2.push(z) \end{array}}{(pc, S, M, R, pc) \rightsquigarrow (pc+1, S_3, M, R, pc+1)}\ \text{BINARY-OP} \qquad \frac{\begin{array}{c} S_1, x = S.pop() \quad S_2, y = S_1.pop() \quad S_3, m = S_2.pop() \\ z = op(x, y, m) \quad S_4 = S_3.push(z) \end{array}}{(pc, S, M, R) \rightsquigarrow (pc+1, S_4, M, R)}\ \text{TERNARY-OP}$$

$$\frac{S_1, p = S.pop() \quad v = M[p] \quad S_2 = S_1.push(v)}{(pc, S, M, R) \rightsquigarrow (pc+1, S_2, M, R)}\ \text{MLOAD} \qquad \frac{S_1, p = S.pop() \quad S_2, v = S_1.pop() \quad M_1 = M[p \leftarrow v]}{(pc, S, M, R) \rightsquigarrow (pc+1, S_2, M_1, R)}\ \text{MSTORE}$$

$$\frac{S_1, p = S.pop() \quad v = R[p] \quad S_2 = S_1.push(v)}{(pc, S, M, R) \rightsquigarrow (pc+1, S_2, M, R)}\ \text{SLOAD} \qquad \frac{S_1, p = S.pop() \quad S_2, v = S_1.pop() \quad R_1 = R[p \leftarrow v]}{(pc, S, M, R) \rightsquigarrow (pc+1, S_2, M, R_1)}\ \text{SSTORE}$$

$$\frac{v = S.get(i) \quad S_1 = S.push(v)}{(pc, S, M, R) \rightsquigarrow (pc+1, S_1, M, R)}\ \text{DUP-I} \qquad \frac{v_0 = S.get(0) \quad v_i = S.get(i) \quad S_1 = S[0 \leftarrow v_i] \quad S_2 = S_1[i \leftarrow v_0]}{(pc, S, M, R) \rightsquigarrow (pc+1, S_2, M, R)}\ \text{SWAP-I}$$

$$\frac{S_1, lbl = S.pop() \quad S_2, c = S_1.pop() \quad c \neq 0}{(pc, S, M, R) \rightsquigarrow (lbl, S_2, M, R)}\ \text{JUMPI-T} \qquad \frac{S_1, lbl = S.pop() \quad S_2, c = S_1.pop() \quad c = 0}{(pc, S, M, R) \rightsquigarrow (pc+1, S_2, M, R)}\ \text{JUMPI-F}$$

$$\frac{S_1, lbl = S.pop()}{(pc, S, M, R) \rightsquigarrow (lbl, S_1, M, R)}\ \text{JUMP} \qquad \frac{res = call() \quad S_1 = S.push(res)}{(pc, S, M, R) \rightsquigarrow (pc+1, S_1, M, R)}\ \text{CALL}$$

$$\frac{S_1, p = S.pop() \quad S_2, n = S_1.pop() \quad v = sha3(M[p, p+n]) \quad S_3 = S_2.push(v)}{(pc, S, M, R) \rightsquigarrow (pc+1, S_3, M, R)}\ \text{SHA3}$$

# Enumerate symbolic traces

**Symbolic trace** $\langle s_0, op_0, \cdots, s_n, op_n, s_{n+1} \rangle$

## Without loops

apply the symbolic rules **iteratively**

## With loops

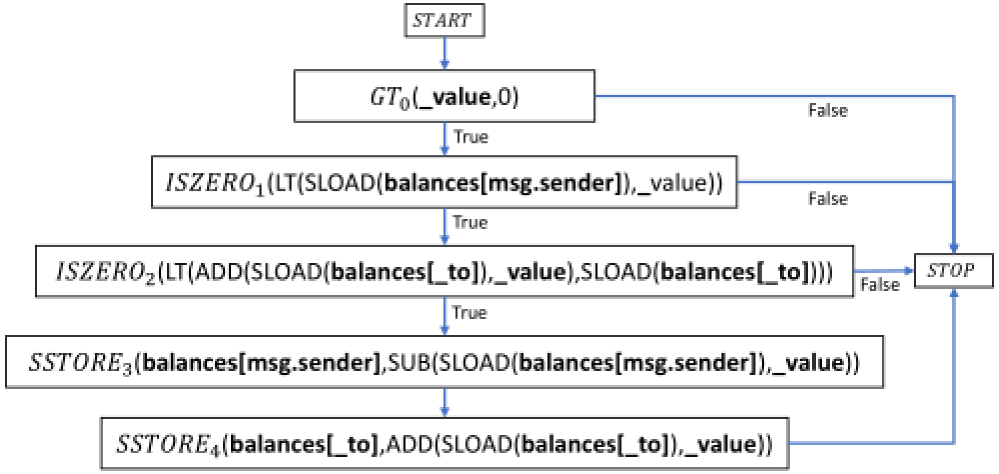
Establish a **bound** on the number of iterations

# Dependency analysis

## Control dependency

## Data dependency

```
function transfer(address _to, uint _value) public {
  if (_value <= 0) revert();
  if (balances[msg.sender] < _value) revert();
  if (balances[_to] + _value < balances[_to]) revert();
  balances[msg.sender] = balances[msg.sender] - _value;
  balances[_to] = balances[_to] + _value;
}
```



# Fix the smart contract

**check** whether each **symbolic trace** suffers from any of the vulnerabilities

introduce **runtime checks** so as to prevent the vulnerability

1、overflow/underflow : safe_add、safe_sub

2、reentrancy：modifier（mutex)

3、tx.origin： msg.sender

# Intra-function reentrancy vulnerability

- ➢the no writes after call (NW)
- ➢**dependency from $op_c$ to $op_s$**

```
uint numWithdraw = 0;
function withdraw() external {
    uint256 amount = balances[msg.sender];
    balances[msg.sender] = 0;
    (bool ret, ) = msg.sender.call.value(amount)("");
    require(ret);
    numWithdraw ++;
}
```

false positive

```
function getThisWeekBurnedAmount() public view returns(
    uint) {
  uint thisWeekStartTime = getThisWeekStartTime();
  uint total = 0;
  for (uint i = numOfBurns; i >= 1; (i = sub_uint256(i,
      1))) {
    if (burnTimestampArr[sub_uint256(i, 1)] <
        thisWeekStartTime) break;
    total = add_uint256(total, burnAmountArr[
        sub_uint256(i, 1)]);
  }
  return total;
}

function getThisWeekBurnAmountLeft() public view
    returns(uint) {
  return sub_uint256(weeklyLimit,
      getThisWeekBurnedAmount());
}

function burn(uint amount) external payable
    nonReentrant {
  require(amount <= getThisWeekBurnAmountLeft());
  require(IERC20(tokenAddress).transferFrom(msg.sender,
      BURN_ADDRESS, amount));
  (numOfBurns = add_uint256(numOfBurns, 1));
}
```

Intra-function reentrancy vulnerability

# Fix Intra-function reentrancy vulnerability

## Patched by sGUARD

```
function getThisWeekBurnedAmount() public view returns(
    uint) {
  uint thisWeekStartTime = getThisWeekStartTime();
  uint total = 0;
  for (uint i = numOfBurns; i >= 1; (i = sub_uint256(i,
      1))) {
    if (burnTimestampArr[sub_uint256(i, 1)] <
        thisWeekStartTime) break;
    total = add_uint256(total, burnAmountArr[
        sub_uint256(i, 1)]);
  }
  return total;
}

function getThisWeekBurnAmountLeft() public view
    returns(uint) {
  return sub_uint256(weeklyLimit,
      getThisWeekBurnedAmount());
}

function burn(uint amount) external payable
    nonReentrant {
  require(amount <= getThisWeekBurnAmountLeft());
  require(IERC20(tokenAddress).transferFrom(msg.sender,
      BURN_ADDRESS, amount));
  (numOfBurns = add_uint256(numOfBurns, 1));
}
```

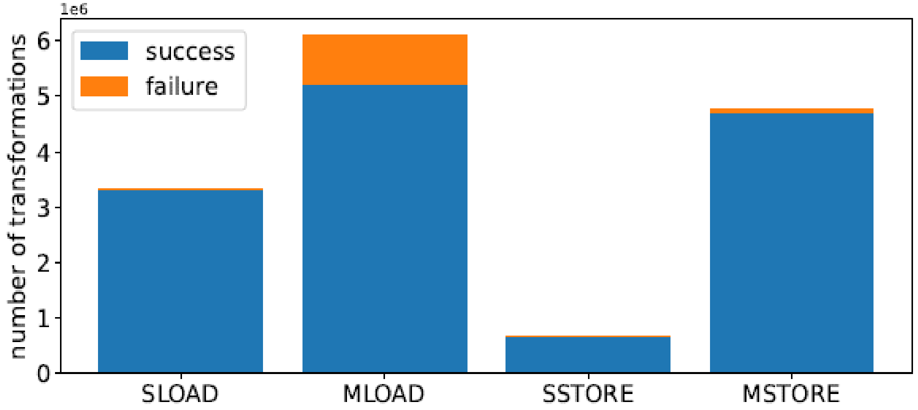
OpenZeppelin

**The standard for secure blockchain applications**

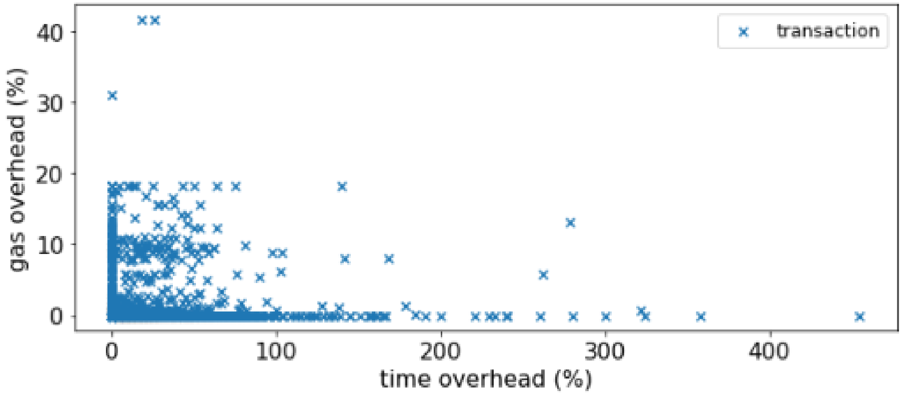
OpenZeppelin provides security products to build, automate, and operate decentralized applications. We also protect leading organizations by performing security audits on their systems and products.

```
modifier nonReentrant() {
    _nonReentrantBefore();
    _;
    _nonReentrantAfter();
}

function _nonReentrantBefore() private {
    // On the first call to nonReentrant, _status will be _NOT_ENTERED
    require(_status != _ENTERED, "ReentrancyGuard: reentrant call");

    // Any calls to nonReentrant after this point will fail
    _status = _ENTERED;
}

function _nonReentrantAfter() private {
    // By storing the original value once again, a refund is triggered (see
    // https://eips.ethereum.org/EIPS/eip-2200)
    _status = _NOT_ENTERED;
}
```

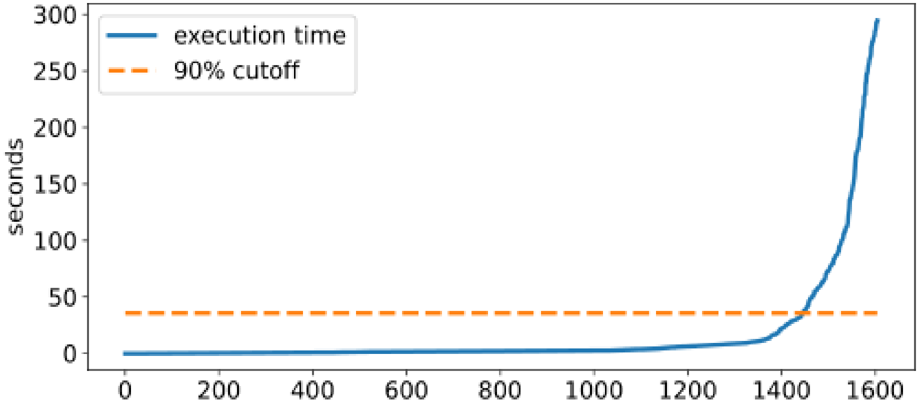
# Overall Evaluation Results




precision




overhead




efficiency

# Conclusion

- **Precisely identify four types of vulnerability**
- **Fix contracts with introducing only minor overhead**