NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

School of Computer Science and Engineering
College of Engineering

Gas-Aware Instrumentation for Solidity Smart Contract Runtime Monitoring

OBJECTIVE:

- Study different vulnerabilities in smart contracts and understand their causes.
- Investigate the use of different tools to build a proof-of-concept to automate vulnerabilities' discovery in smart contracts.

PROBLEM:

Poorly written smart contracts are susceptible to hacks. One notorious example is the **"DAO"** attack, which results in more than **$45 million USD** stolen. The example we are examining is the "batchTransfer" for Beauty Token (BEC). By exploiting the integer overflow, attackers can generate an extremely large amount of tokens, and deposit them into a normal address.

```
1     function batchTransfer(address[] memory _receivers, uint256 _value)
      public whenNotPaused returns (bool) {
2           checkTotalAmount();
3           uint cnt = _receivers.length;
4           uint256 amount = uint256(cnt) *  value;
5           require(cnt > 0 && cnt <= 20);
6           require( value > 0 && balances[msg.sender] >= amount);
7           balances[msg.sender] = balances[msg.sender].sub(amount);
8           for (uint i = 0; i < cnt; i++) {
9                 addOwner(_receivers[i]);
10                balances[_receivers[i])=balances[_receivers[i]].add(_value);
11                emit Transfer(msg.sender, _receivers[i], _value);
12          }
13          checkTotalAmount();
14          return true;
15    }
```

If '_value' is large, it may cause integer overflow.

↓

Unexpected Result: amount is lesser than '_value'
This line will pass with no error.

↓

The *receiver* can get large amount of tokens when it is not supposed to.

```
1   function checkTotalAmount() public returns(bool success)
2   {
3       uint256 sum = 0;
4       uint256 balance = 0;
5       address addr;
6       for (uint256 i = 0; i < _totalHolders; ++i)
7       {
8           addr = checkHolders[i];
9           balance = balances[address];
10          require(sum + balance >= sum);
11          sum += balance;
12          emit TotalBalance(sum, balance);
13      }
14      require(sum == totalSupply);
15      return true;
16  }
```

SOLUTION:

In this project, the first tool we are using is the Solidity Instrumentation Framework (SIF) to insert code for run-time monitoring. In this case, we inserted the "checkTotalAmount" function.
It ensures the total supply of tokens is equal to the sum of tokens of each address.

Other tools used:
1. Truffle Suite for Automation of Tests
2. Modified Solidity Compiler
3. Modified Ethereum Virtual Machine

Check that there is no overflow or underflow of tokens

Check that there is no integer overflow for addition of sum and tokens of any account

↓

If there is any arithmetic error, the transaction is reverted.

RESULTS:

BECToken: Successfully prevent attack from occurring
Useless Ethereum Token: Provide useful logging information to prevent attack
Working Proof of Concept for full automation of Run-Time Analysis

STUDENT: ROYSTON BEH ZHI YANG
SUPERVISOR: AST PROF LI YI

FYP AY19-20 SEMESTER 2
SCSE19-0028