Abstract

In this report, we consider the security of GO Cubo Lodge Club, who engaged ForsetiDev team on 9 December 2017 to perform smart contracts audit of GO Token. A audit was conducted on the commit version `aa40f0f`

Analysis technique

We used several publicly available automated Solidity analysis tools, as well as proceed manual analysis. All the issues found by tools were manually checked (rejected or confirmed). Contracts were manually analyzed, their logic was checked and compared with the one described in the whitepaper.

Bugs classification

CRITICAL - problems leading to stealing funds from any of the participants, or making them inaccessible by anyone

SEVERE - problems that can stop, freeze or break the internal logic of the contract

WARNING - non-critical problems that cannot break the contract, but contract code does not match declared in WhitePaper logic

Notes - any other findings .

Automated Analysis

Oyente

Timestamp Dependency

CommonCrowdsale : line 357

Timestamp Dependency : True

GOTokenCrowdsale : line 357

Timestamp Dependency:True

Securify

Transaction Reordering

Transactions May Affect Ether Receiver

Matched lines: L.501

Transactions May Affects Ether Amount

Matched lines: L.501

All the issues found by tools were manually checked (rejected or confirmed). Cases, when these issues lead to actual bugs or vulnerabilities, are described in the next section.

Manual Analysis

Severe

CommonCrowdsaele.sol, line 499 :

function createTokens()

There is no check if hardcap will be exceeded,after current transaction, when calling createTokens() function, only check if invested <hardcap already.For example if there is 10 eth until hardcap, investor can send 20 eth transaction, which will pass all checks, despite hardcap will be exceeded. You can exceed maxInvestedLimit in the same way (investedInWei >= maxInvestedLimit). Its recommended to check whether current transaction exceeding hardcap, and if so, send change back to investor

Undesirable loops

CommonCrowdsale.sol, line 431 :

function payExtraTokens(**uint** count)

Loops are undesirable and quite dangerous in solidity, we recommend avoid them where it possible.In this case, we recommend modifying this function in a way, that investor should initiate payExtraTokens by himself

CommonCrowdsale.sol, line 412 :

function end()

In this case its possible to declare variable “foo” and add it to function

```
uint256 foo;
……..
function addMilestone(uint periodInDays, uint discount) public onlyOwner {
 milestones.push(Milestone(periodInDays, discount));
 foo+= periodInDays;
}
```

and increment foo+=periodInDays

to modify end() function this way

```
function end() public constant returns(uint) {
 uint last = start+foo;
 return last;
}
```

Compiler version not fixed

```
pragma solidity ^0.4.17; // bad: compiles w 0.4.17 and above
pragma solidity 0.4.17; // good : compiles w 0.4.17 only
```

It is recommended to follow the latter example, as future compiler versions may handle certain language constructions in a way the developer did not foresee.

Investor may exceed the investment limit

CommonCrowdsale.sol, line 486

```
function calculateAndTransferTokens(address to, uint investedInWei) internal {
 invested = invested.add(msg.value);
 uint tokens =  investedInWei.mul(price.mul(PERCENT_RATE)).div(PERCENT_RATE.sub(getDiscount())).div(1 ether);
 mint(to, tokens);
 if(investedInWei >= maxInvestedLimit) token.lock(to);
}
```

Current function only check one-time investment.
Theoretically, if maxInvestedLimit = 1000 I can invest 10 times for 999 dollars , and still not be blocked.
To freeze tokens of an investor, that exceeded the limit , mapping of balances(address => uint) must be stored and then calculateAndTransferTokens function would be :

```
function calculateAndTransferTokens(address to, uint investedInWei) internal {
 invested = invested.add(msg.value);
 uint tokens = investedInWei.mul(price.mul(PERCENT_RATE)).div(PERCENT_RATE.sub(getDiscount())).div(1 ether);
 mint(to, tokens);
 if(investedInWei >= maxInvestedLimit) token.lock(to);
}
```

Warnings

ShortAdressAttack

BasicToken.sol, line 70 :

Its recomended to modify transfer function to protect from ShortAddressAttack (http://vessenes.com/the-erc20-short-address-attack-explained/)

Redundant fallback function

StantardToken.sol, line 171 :

```
function () public payable {
 revert();
}
```

Contracts should reject unexpected payments. Before Solidity 0.4.0, it was done manually:

```
function () payable { throw ; }
```

Starting from Solidity 0.4.0, contracts without a fallback function automatically revert payments, making the code above redundant.

Unchecked math

CommonCrowdsale.sol, line 472 :

```
function getDiscount() public constant returns(uint) {
prevTimeLimit += milestone.periodInDays * 1 days;
```

Solidity is prone to integer over- and underflow. Overflow leads to unexpected effects and can lead to loss of funds if exploited by a malicious account.

Notes

Constant functions

CommonCrowdsale.sol, line 472 :

```
function tokenHoldersCount() public constant returns(uint) {}
```

The function is declared as `constant`. Currently, for functions the `constant` modifier is a synonym for `view` (which is the preferred option). Consider using `view` for functions and `constant` for state variables.

Redundant assignment

CommonCrowdsale.sol, line 361 :

```
function tokenHoldersCount() public constant returns(uint) {
 uint length = tokenHolders.length;
 return length;
}
```

Redundant assignment uint length = tokenHolders.length , but further , different approach is used.

```
function milestonesCount() public constant returns(uint) {
 return milestones.length;
}
```

Redundant assignment

CommonCrowdsale.sol, line 306

Variables (hard cap, price,start,wallet e.t.c) declared twice. Its recommended assigning values only in **GOTokenCrowdsale** contract.

Unnecessary function call

CommonCrowdsale.sol, line 429

function payExtraTokens(**uint** count)
token.mint(this, targetValue);
token.transfer(tokenHolder, targetValue)
Its possible to call token.mint(tokenHolder, targetValue); at first place

Recommendations

To add soft cap variable and refund function

No soft cap means that there is no possibility to add refund functionality. We recommend adding refund functions when soft cap is not reached