Global Digital Opportunities

GDO Infotech Private Limited
Amreli – Surat – Poland – UK – Canada
Web: www.gdo.co.in **Email**: contact@gdo.co.in
Ph: +91 9712144344

SMART CONTRACT AUDIT REPORT

For

BitDollars Token (Order # FO6D6DB888B3)

Prepared By: Yogesh Padsala

Prepared For: BitDollars Token

Prepared on: 07/06/2018

Table of Content

1. Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only.

2. Overview of the audit

The project has 1 file BitDollars.sol. It contains approx 631 lines of Solidity code. All the functions and state variables are not well commented using the natspec documentation.

3. Attacks made to the contract

In order to check for the security of the contract, we tested several attacks in order to make sure that the contract is secure and follows best practices.

3.1: Over and under flows

An overflow happens when the limit of the type variable uint256, 2 ** 256, is exceeded. What happens is that the value resets to zero instead of incrementing more. On the other hand, an underflow happens when you try to subtract 0 minus a number bigger than 0. For example, if you subtract 0 - 1 the result will be = 2 ** 256 instead of -1. This is quite dangerous.

This contract **does** check for overflows and underflows by using OpenZeppelin's SafeMath to mitigate this attack.

3.2: Short address attack

If the token contract has enough amount of tokens and the buy function doesn’t check the length of the address of the sender, the Ethereum’s virtual machine will just add zeros to the transaction until the address is complete.

This contract isn’t vulnerable to this attack since it doesn't have any Buy function but also it **does NOTHING to prevent** the *short address attack*

during **ICO** or in an **exchange** (it will just depend if the ICO contract or DApp to check the length of data. If they don't, then short address attacks would drain out this coin from the exchange).

3.3: Visibility & Delegatecall

It is also known as, The Parity Hack, which occurs while misuse of Delegatecall.

No such issues found in this smart contract and visibility also properly addressed. There are some places where there is no visibility defined. Smart Contract will assume “Public” visibility if there is no visibility defined. It is good practice to explicitly define the visibility, but again, the contract is not prone to any vulnerability due to this in this case.

3.4: Reentrancy / TheDAO hack

Reentrancy occurs in this case: any interaction from a contract (A) with another contract (B) and any transfer of Ether hands over control to that contract (B). This makes it possible for B to call back into A before this interaction is completed.

Use of “require” function in this smart contract mitigated this vulnerability.

3.5: Forcing ether to a contract

While implementing “selfdestruct” in smart contract, it sends all the ether to the target address. Now, if the target address is a contract address, then the fallback function of target contract does not get called. And thus Hacker can bypass the “Required” conditions. Here, the Smart Contract’s balance has never been used as guard, which mitigated this vulnerability.

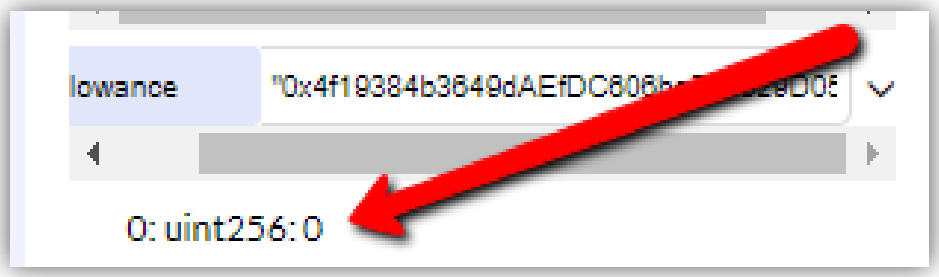
4. Critical vulnerabilities found in the contract

4.1: Underflow & Overflow attack:

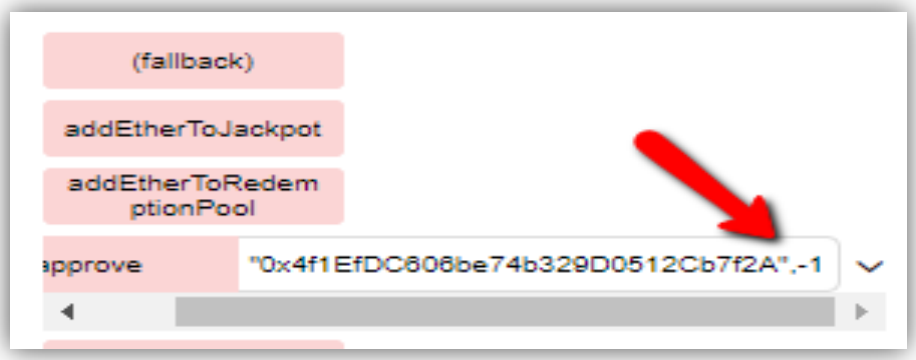
=>In your contract some functions accept negative value.

=>Function name: - approve, debug_setTotalCreated ,increaseApproval.

- **Approve**
  - Allowance value in beginning.

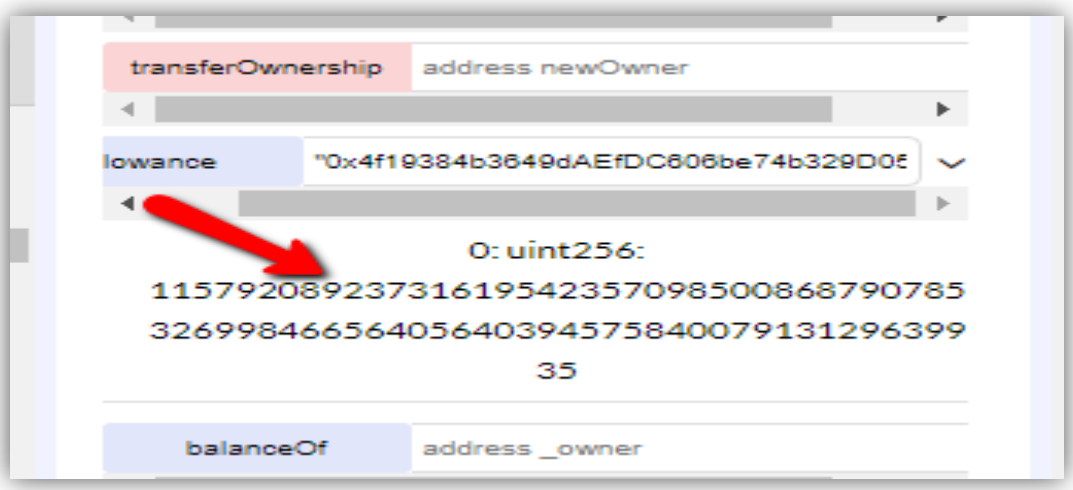


  - Now calling approve function with negative value.

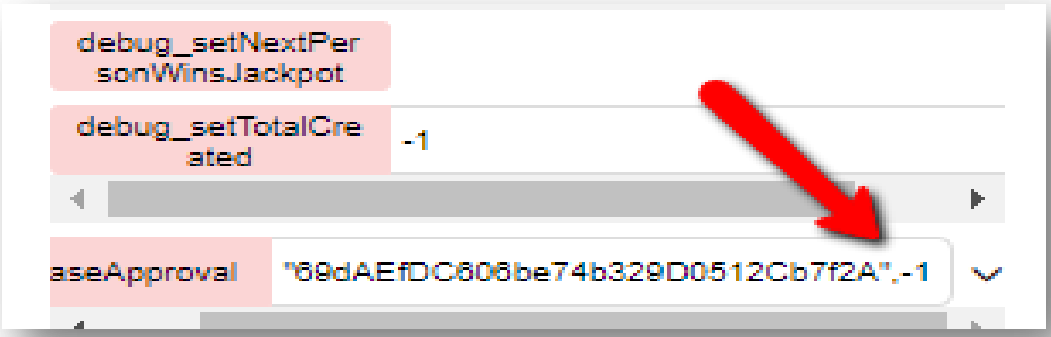


  - Transaction Hash:- https://rinkeby.etherscan.io/tx/0xa5832538ef8d3868cd2e53da070534ce6a33944adc72ad2ab3c6f3f25deba617.

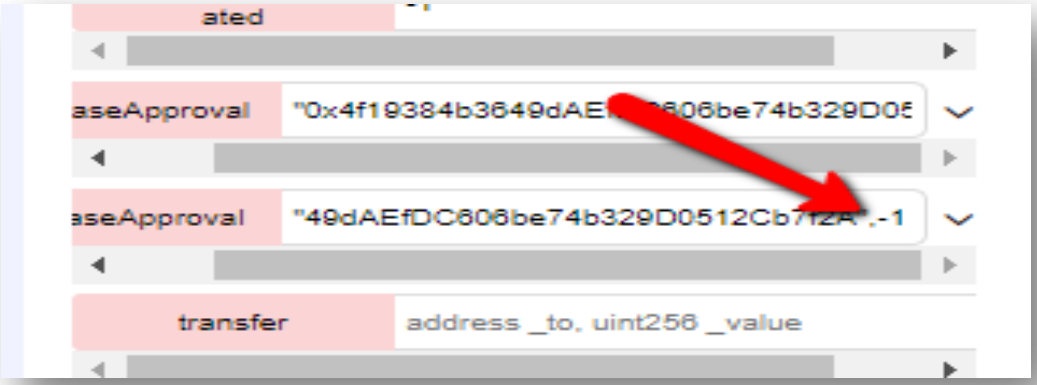
- Allowance after negative approves.



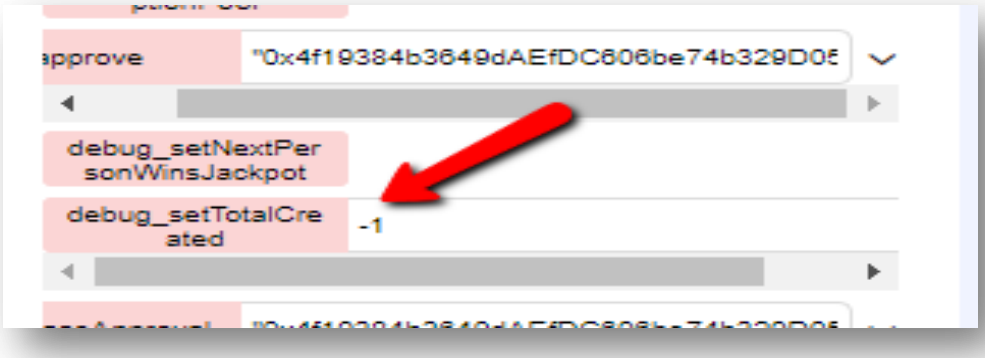
- Calling decreaseApproval with negative value.



- Transaction hash:- https://rinkeby.etherscan.io/tx/0xe8c84bd2daea7abd53db14e4481252e884d9fe4416bafbc313b2a31c5727e445.
- Calling increaseApproval with negative value.



- Transaction hash:-
- https://rinkeby.etherscan.io/tx/0xd4ba1920d20083832e035472a9fcc545e4e2f4c56893f8ae613e9b8d78cc28ae.
- Calling debug_setTotalCreated function with negative value.



- Transaction hash:-
- https://rinkeby.etherscan.io/tx/0x652635a65070af95f8ba7af78f9f847a08575c12b7b3fcfd0b2f0220052ac3a1.

Solution:-

```
 * race condition is to first reduce the spender's allowance to 0 and set the desired value afte
 * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
 * @param _spender The address which will spend the funds.
 * @param _value The amount of tokens to be spent.
 */
function approve(address _spender, uint256 _value) public returns (bool) {
  allowed[msg.sender][_spender] = _value;
  emit Approval(msg.sender, _spender, _value);
  return true;
}

/**
 * @dev Function to check the amount of tokens that an owner allowed to a spender.
```

- In approve, increaseApproval and decreaseApproval functions you have to put one condition.

  require(_value <= balances[msg.sender]);

- By this way user get approve of his balance not more than then his balance.
- In debug_setTotalCreated function you have to take care of variable "_value" value when you call it.

4.2: Short address attack

=>In your contract, some functions are not checking the validity of address variable.

=>Function name: - transferFrom, approve, debug_setTotalCreated and increaseApproval.

```
 * @dev Transfer tokens from one address to another
 * @param _from address The address which you want to send tokens from
 * @param _to address The address which you want to transfer to
 * @param _value uint256 the amount of tokens to be transferred
 */
function transferFrom(address _from, address _to, uint256 _value) public returns (bool) {
  require(_to != address(0));
  require(_value <= balances[_from]);
  require(_value <= allowed[_from][msg.sender]);

  balances[_from] = balances[_from].sub(_value);
  balances[_to] = balances[_to].add(_value);
  allowed[_from][msg.sender] = allowed[_from][msg.sender].sub(_value);
  emit Transfer(_from, _to, _value);
  return true;
}

/**
 * @dev Approve the passed address to spend the specified amount of tokens on behalf of msg.sender.
 *
```

- You are not checking the validity of "from" variable.
- Anyone can request these function with short address.

Solution:-

- Add only one line in these functions.

- **require(address parameter != address(0));**

5. Medium vulnerabilities found in the contract

5.1: Compiler version not fixed

=> In this file you have put "pragma solidity ^0.4.21;" which is not good way to define compiler version.

=> Solidity source files indicate the versions of the compiler they can be compiled with.

pragma solidity ^0.4.21; // bad: compiles w 0.4.21 and above

pragma solidity 0.4.21; // good : compiles w 0.4.21 only

=> If you put (^) symbol then you are able to get compiler version 0.4.21 and above. But if you don't use (^) symbol then you are able to use only 0.4.21 version. And if there is some changes come in compiler and you use old version then some issue may come at deploy time.

=> And try to use latest version of solidity compiler (0.4.24).

5.2: Unchecked math:

=> You are using safemath library that is good thing.

=>But line number #428 and #431 you are not using safemath library.

=>You can make your contract safe from underflow and overflow attack when you use safemath for mathematic calculation.

```
423         msg.sender.transfer(amountToSend);
424
425         // 3.2 Deduct BitDollars from the seller
426
427             // 3.2.1 Burn the BitDollars by subtracting the sender balance
428             balances[msg.sender] -= amountToSell;
429
430             // 3.2.2 Deduct BitDollars from the totalSupply counter
431             totalSupply_ -= amountToSell;
432
433
```

Solution:-

Here you can put these two lines like this way

1) For #428

   balances[msg.sender]=balances[msg.sender].sub(amountToSell);

2) For #431

   totalSupply_=totalSupply_.sub(amountToSell);

5.3: vulnerability in transfer and transferFrom function.

=>In these function you are not checking the value of "_value" parameter.

=>Right now anyone can call these functions with the 0 value.

```
130   * @param _value The amount to be transferred.
131   */
132   function transfer(address _to, uint256 _value) public returns (bool) {
133     require(_to != address(0));
134     require(_value <= balances[msg.sender]);
135
136     balances[msg.sender] = balances[msg.sender].sub(_value);
137     balances[_to] = balances[_to].add(_value);
138     emit Transfer(msg.sender, _to, _value);
139     return true;
140   }
141
142   /**
```

=>You are not checking the _value whether it is more then 0 or not.

Solution:-

- Try to add this line in these two function.

  require(_value > 0);

6. Low severity vulnerabilities found

6.1: Implicit visibility level

=> At line #116, #118 you did not specify the visibility level.

=> This is not a big issue in solidity. But it is good practice to define the visibility level. If you do not specify it, then it automatically takes public, but just in case if you want to make variable or function private, then you must specify that.

```
114   using SafeMath for uint256;
115
116   mapping(address => uint256) balances;
117
118   uint256 totalSupply_;
119
120   /**
```

Solution:-

1) For #116
    a. mapping(address => uint256) public balances;
2) For #118
    a. uint256 public totalSupply_;

7. Summary of the Audit

Overall the code is well commented.

Our final recommendation would be to pay more attention to the visibility of the functions , hardcoded address and mapping since it’s quite important to define who’s supposed to executed the functions and to follow best practices regarding the use of assert, require etc. (which you are doing ;)).

Try to check the address and value of token externally before sending to the solidity code.

you are using constant function for viewing the information it's ok now because constant is alias of the view. But it's good thing to use view function for viewing smart contract information. For more details: https://ethereum.stackexchange.com/questions/25200/solidity-what-is-the-difference-between-view-and-constant/25202