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

Crypto Asset Ratings (Order #FO810808)

Prepared By: Yogesh Padsala **Prepared For**: Crypto Asset Ratings

Prepared on: 08/06/2018

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 2 file and both contains approx 1100 lines of Solidity code. All the functions and state variables are 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: File Name: ANSAToken.sol

4.1.1: Overflow and underflow:

=> approve and mint functions accept negative values. You need to check the value is not negative.

=> Or better, please use OpenZeppelin's SafeMath library to mitigate this issue.

https://github.com/OpenZeppelin/openzeppelin-solidity/blob/master/contracts/math/SafeMath.sol

4.2: File name: AnsaCrowdsale.sol

4.2.1: Overflow and underflow:

=> Some of the functions accepts negative values. So, you need to check the value of those functions as it should not be less than zero. Or better, use safemath as much as possible.

1)tokenDistribution function "_value" variable accepting nagetive value. (line number :-776)

2)changeMinInvestment function "_minInvestment" variable accepting nagetive value.(line number :-752)

3)changeRate function "_rate" variable accepting nagetive value.(line number :-727)

4)changeEndtime function "_endTime" variable accepting nagetive value.(line number :-718)

5)addToWhitelist function "_stage" variable accepting nagetive value.(line number :-673)

6)startPhase3 function "_endTime" and "_startTime" variable accepting nagetive value.(line number :-655)

7)startPhase2 function "_endTime" and "_startTime" variable accepting nagetive value.(line number :-646)

8) changeAdminCharges function "p1" ,"p2" and "p3" variable accepting negative value.(line number :-738).

5. Medium vulnerabilities found in the contract

5.1: File Name: ANSAToken.sol

5.1.1: Short address attack:

=> There are some places you do not check the value of address as like:

1) In Transfer function:- "_to" parametere (line number :- #68).

2) In Transfer function:-"_to" parametere (line number :- #81).

3) In transferToAddress function:-"_to" parametere (line number :- #95).

4) In transferToContract function :- "_to" parametere (line number :- #102).

5) In isContract function :- "_addr" parametere (line number :- #116).

6) In transferFrom function :- "_from" and "_to" parameteres (line number :- #138).

7) In approve function :- "_spender" parametere (line number :-#150).

8) In mint function :- "_to" parametere (line number :- #233).

9) In Transfer function:-"_to" parametere (line number :- #266).

10) In Transfer function:-"_to" parametere (line number :- #280).

5.1.2: Compiler version not fixed

=> In this file you have put "pragma solidity ^0.4.23;" 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 ^0.4.23; // bad: compiles w 0.4.23 and above

pragma solidity 0.4.23; // good : compiles w 0.4.23 only

=> If you put (^) symbol then you are able to get compiler version 0.4.23 and above. But if you don't use (^) symbol then you are able to use only 0.4.23 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: File name: AnsaCrowdsale.sol

5.2.1: Short address attack:

=> There are some places you do not check the value of address as like:

1)in returnInvestoramount function:-"_beneficiary" parametere (line number :- #372).

2)in deposit function:-"investor" parametere (line number :- #411).

3)in closeAfterWhitelisted function:-"_beneficiary" parametere (line number :- #414).

4)in refund function:-"investor" parametere (line number :- #425).

5)in buyTokens function:-"_beneficiary" parametere (line number :- #528).

6)in addToWhitelist function:-"_beneficiary" parametere (line number :- #673)

=> To check validity, you can add following condition in all those functions:

1) require(Address parameter != address(0));

2) require (Address parameter != 0x0);

PS: Please replace “Address Parameter” keywords with your actual keywords based on any particular function.

5.2.2: Compiler version not fixed

=> Please try to put latest solidity compiler version (0.4.24)

6. Low severity vulnerabilities found

6.1: File Name: ANSAToken.sol

6.1.1: Implicit visibility level

=> At line #135,#60, 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.

6.2: File name: AnsaCrowdsale.sol

=> Please try to set visibility level at line number #492,#493,#494 and #47

7. Summary of the Audit

Overall the code is well commented.

The tokens will be locked up until 1 year. So it will not allow transfer of the token in 12 months after the ICO. This logic is implemented in ANSAToken.sol file.

Our final recommendation would be to pay more attention to the visibility of the functions 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