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 Skype: yogeshpadsala

SMART CONTRACT AUDIT REPORT

For

Fun Token (Order # FO811D7F49704)

Authored By: Yogesh Padsala **Prepared For**: daddytab

Prepared on: 26/05/2018

Table of Content

1. Disclaimer

2. Overview of the audit

3. Attacks made to the contract

4. Critical vulnerabilities found in the contract

5. Medium vulnerabilities found in the contract

6. Low severity vulnerabilities found in the contract

8. Summary of the audit

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 files and it contains approx 1950 lines of Solidity code. All the functions and state variables are well commented using the natspec documentation for the functions which is good to understand quickly how everything is supposed to work.

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 SafeMat. There are some places (as discussed below) where it is not used. So, it is good practice to use that everywhere 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.

It is recommended to use latest solidity compiler 0.4.24. This version of solidity automatically rejects all the short addresses (further discussed below).

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. 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: Congratulations. No critical vulnerabilities found.

5. Medium vulnerabilities found in the contract

5.1: File Name - FUNTokenAll.sol

5.1.1: Compiler version not fixed

=> In this file you have put "pragma solidity ^0.4.22;" 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 ^0.4.22; // bad: compiles with 0.4.22 and above

pragma solidity 0.4.22; // good : compiles with 0.4.22 only

=> If you put (^) symbol then you are able to get compiler version 0.4.22 and above. But if you don't use (^) symbol then you are able to use only 0.4.22 version. And if there is some changes come in compiler and you use old version then some issue may come at deploy time.

=> Also, it is highly recommended to use latest version of solidity compiler (0.4.24).

5.1.2: Short address attack

=> You have not used address validation in some function. As like spender address in approve function, _from address in transferFrom function ,_spender address in increaseApproval and decreaseApproval function ,_address parameter in sendCrowdsaleTokens function

=> Either you can use latest solidity compiler (0.4.24), or you explicitly validate for short address inputs.

5.2: File Name - FUNCrowdsaleAll.sol

5.2.1: Compiler version not fixed

=> As said above, try to use latest solidity compiler and also try to remove caret (^) symbol.

6. Low severity vulnerabilities found

6.1: File Name - FUNTokenAll.sol

6.1.1: Implicit visibility level not specified

=> There are some places where you have not put visibility level, like line numbers: #104,#110,#107

=> Please try to add visibility levels on these lines. This is not huge issue although. If you do not put visibility level then by default it takes “public” but if you want some function or variable to be private, then you must give visibility level.

6.1.2: increaseApproval, transfer and transferFrom functions issue:

=> In these functions, your conditions are good but you are not checking the amount of token, which is 0 or not. These functions are still accepting token value zero (0)

=> This is not a big issue but you can prevent user or DApp to call the smart contract with amount zero, even in error and which would cost gas to users.

6.2: File Name - FUNCrowdsaleAll.sol

6.2.1: Costly loop possibility

=> On line numbers #861, #887, #888, #889, #890, #891, #953, #866, #838, #818, #1180, #1120, #964, #917

=> There are loops in the code where data come from outside so you have to take caution when you send the data

=> Incorrect data can raise situation where this loop go in infinite mod, which would drain all ether for gas price.

6.2.2: Unchecked Math**:**

=> You are using SafeMath library, which is good thing. But at line numbers #1171,#819,#300,#983, SafeMath is not being used.

=> As said earlier, it is recommended to use OpenZeppelin's SafeMath library everywhere to mitigate underflow and overflow risk.

6.2.3: Implicit visibility level not specified

=> There are some places where you have not put visibility level, like line numbers: #971,#1322,#1029,#1030

=> Although this is not a big problem, but it is good practice to specify visibility level everywhere.

8. 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. This is good practice to check those from DApp, as it gives clear alert message to end users.

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