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

# Roman Grushkovsky (Order # FO6EC8199FC4)

**Prepared By**: Yogesh Padsala **Prepared For**: Roman Grushkovsky

**Prepared on**: 30/04/2018

# 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 files and it contains approx 500 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 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 – URUN_pre_sale.sol**

**4.1.1:** Unchecked math.

=> We found that you are not using safe math library in this file on line numbers #108 and #109

=> As said earlier, it is recommended to use OpenZeppelin's SafeMath library to mitigate this issue.

# 5. Medium vulnerabilities found in the contract

**5.1: File Name – URUN_token.sol**

**5.1.1:** Compiler version not fixed

=> In this file you have put "pragma solidity ^0.4.19;" 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 ^0.4.19; // bad: compiles w 0.4.19 and above

pragma solidity 0.4.19; // good : compiles w 0.4.19 only

=> If you put (^) symbol then you are able to get compiler version 0.4.19 and above. But if you don't use (^) symbol then you are able to use only 0.4.19 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 (0.4.23).

# 6. Low severity vulnerabilities found

**6.1: File Name – URUN_pre_sale.sol**

**6.1.1:** constant functions

=> This is not a big issue but just for your information, you are using “constant” for viewing data. 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 you can visit following link:

https://ethereum.stackexchange.com/questions/25200/solidity-what-is-the-difference-between-view-and-constant/25202

**6.1.2**: Implicit visibility level not specified

=> There are some places where you have not put visibility level, like line numbers: #113, #114, #118, #116, #108, #109

=> 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.

**6.2: File Name - URUN_token.sol**

**6.2.1:** Implicit visibility level not specified on line #113

**6.2.2**: Using the approve function of the ERC-20 standard

=> You are using “approve” function for allowance directly. Sometimes if you give allowance 0 then it also execute without any error and remove all the allowance and set it 0. This is not a very big issue, so again if you think that is not big problem according to your business logic, then you can safely ignore this.

# 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.

In URUN_pre_sale.sol file, 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.