WRIOS

ALL YOU NEED IS A BROWSER!

A BLOCKCHAIN-POWERED USER-CENTRIC WEB 3.0 PLATFORM

SMART CONTRACTS SECURITY ANALYSIS

webRunes, 2018

Abstract

In this report, we consider the security of the wr.io project. Our task is to find and describe security issues in the smart contracts.

Summary

In this report we have considered the security of wr.io smart contracts. We performed our audit according to the Procedure described below.
The audit showed some code style issues. We recommend addressing them.

General recommendations

The contracts code is of good code quality. Code style can be improved, however these are minor issues that do not influence code operation.

.

If webRunes decides to improve the code, we recommend following best practices for Pragmas version and following Solidity Style Guide for Code style issues.

The text below is details the statements made in Summary and General recommendations.

Procedure

In our audit, we consider the following crucial features of the smart contract code:
1. Whether the code is secure.
2. Whether the code corresponds to the specifications.
3. Whether the code meets best practices in efficient use of gas, code readability, etc.

We perform our audit according to the following procedure:

automated analysis

- ❏ we scan project's smart contracts with several publicly available automated Solidity analysis tools such as Remix and Solhint
- ❏ we manually verify (reject or confirm) all the issues found by tools

manual audit

- ❏ we manually analyze smart contracts for security vulnerabilities
- ❏ we check smart contracts logic and compare it with the one described in the whitepaper
- ❏ we check ERC20 compliance
- ❏ we run tests and check code coverage report
- ❏ we reflect all the gathered information in the report

Checked vulnerabilities

We have scanned the mart contracts for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that we considered (the full list includes them but is not limited to them):

Reentrancy
Timestamp Dependence
Gas Limit and Loops
DoS with (Unexpected) Throw
DOS with (Unexpected) revert
DoS with Block Gas Limit
Transaction-Ordering Dependence
Use of tx.origin
Exception disorder
Gasless send
Balance equality
Byte array
Transfer forwards all gas
ERC20 API violation
Malicious libraries
Compiler version not fixed
Redundant fallback function
Send instead of transfer
Unchecked external call
Unchecked math
Unsafe type inference
Implicit visibility level
Address hardcoded
Using delete for arrays
Integer overflow/underflow
Locked money
Private modifier
Revert/require functions
Using var
Visibility
Using blockhash
Using SHA3
Using suicide
Using throw

Project overview

Project description

In our analysis we consider wr.io technical specification and smart contracts code (https://github.com/daonomic/contracts-wrio/tree/master/contracts).

Project architecture

For the audit, we have been provided with the following set of solidity files:
DaonomicCrowdsale.sol
RefundableDaonomicCrowdsale.sol (inherits DaonomicCrowdsale contract)
WgdSale.sol (inherits WhitelistDaonomicCrowdsale and RefundableDaonomicCrowdsale contracts from OpenZeppelin library)
WgdToken.sol (inherits StandardBurnableToken contract from OpenZeppelin library)
Whitelist.sol
WhitelistDaonomicCrowdsale.sol (inherits Ownable contract from OpenZeppelin library, inherits DaonomicCrowdsale contract)
WhitelistImpl.sol (inherits Ownable contract from OpenZeppelin library, inherits Whitelist contracts)
The files are the part of the truffle project and an npm package. The project also contains tests and deploy scripts.
The project successfully compiles with truffle compile command.
The project successfully passes all the tests (truffle test command).

Manual analysis

The contracts were completely manually analyzed, their logic was checked and compared with the one described in the documentation. Besides, the results of the automated analysis were manually verified. All confirmed issues are described below. We have also checked the issues found by Remix and Solhint.

General (in all contracts):

- **line 1:** Contracts should be deployed with the same compiler version and flags that they have been tested the most with. Locking the pragma helps ensure that contracts do not accidentally get deployed using, for example, the latest compiler which may have higher risks of undiscovered bugs.

DaonomicCrowdsale.sol

- **lines 28, 70:** address(0) is always used as token address on Purchase event instead of the token address for the token being purchased, can be changed or removed.

- **lines 28, 74:** txId is a parameter of the Purchase event that is unused and always sent as empty string - can be removed.

- **lines 32, 36:** The events RateAdd and RateRemove are defined, but RateAdd is only used once in the constructor of WgdSale and RateRemove is not used at all. Please make sure that this is necessary and used functionality.

- **lines 78, 109:** _postValidatePurchase() function is not overridden in functions that inherit DaonomicCrowdsale and can be removed.

- **lines 115, 154:** For consistency, the functions that are to be overridden should not contain empty blocks.

RefundableDaonomicCrowdsale.sol

No issues found.

WgdSale.sol

- **lines 22 - 43:** Please set explicit visibility for the parameters.

- **line 41:** The getRate() will return rate only if it is called with the zero address as parameter. Please check if this is the intended functionality and comment on why this functionality is necessary (for transparency).

- **line 166 (suggestion):** The getStage() function uses gas every time to check for multiple conditions. The current stage can be set as a variable and the information about the stage can be stored in an array where the stage numbers are indexes and the values are structures containing stage limit and stage rate. A similar structure could be implemented for getAmountBonus().

WgdToken.sol

- **line 12:** Please set explicit visibility for the TOTAL parameter.

Whitelist.sol

No issues found.

WhitelistDaonomicCrowdsale.sol

- **line 20:** Solidity style issue - visibility should be first in list of modifiers, please change for consistency.
- **line 32:** Please use view instead of constant for consistency.

WhitelistImpl.sol

- **line 9:** Please set specific visibility to the whitelist variable.

- **line 16, 22 (suggestion):** The two functions addToWhitelist and removeFromWhitelist can be replaced with one - changeWhitelistStatus and the status can be passed as parameter.