Zin Finance

Smart Contract Audit

The BlockChain Auditor

Prepared by: Jorge Martinez

Version: 2
Date: Oct. 5, 2020

Summary of Findings

This document expresses all security concerns of the Zin Finance Smart Contracts as expressed by Jorge Martinez. Careful analysis of the code repository was performed to identify opportunities to improve the security, code efficiency, best practices, and overall functionality of the smart contracts. That being said, this audit should be seen as one step in the development process with the intent of raising awareness of the meticulous work involved in secure development and making no material statements or guarantees to the operational state of the smart contract(s) once they are deployed.

Contract Status: Ready for Deployment

2 Critical Issue(s) were found in the ZinCrowdsale.sol that were resolved.
2 Medium Level Issue(s) were found in ZinCrowdsale.sol that were resolved.
5 Low Level Issue(s) were found in ZinCrowdsale.sol that were resolved.
8 Informational Issues were found in ZinCrowdsale.sol that were acknowledged.

Test Coverage

ZinFinance	Industry Standard
100.0%	95.0%

Initially, there were no unit tests provided. Zin Finance asked us to develop a unit test suite and as a result the current test coverage is now 100%.

The industry standard is 95% but coverage of at least 75% is acceptable but we were able to achieve 100%.

Table of Contents

Table of Contents

Audit Methodology & Techniques

The BlockChain Auditor has the following auditing process:

1. Our audits include
   a. Review of the specifications, source code, and instructions provided to the BlockchainAuditors to clearly identify the desired functionality of the smart contract(s).
   b. Manual line by line review of contract code to spot potential vulnerabilities.
   c. Identification of deviations between desired functionality expressed to the BlockchainAuditors and what the smart contract(s) are doing.
2. Automated static and symbolic analysis, as well as verifying testing coverage using the provided test suite.
   a. Automated static and symbolic analysis help determine what inputs cause each part of the smart contract to execute. Analysis of how much of the code base is tested and comparison to industry standard.
3. Examination of smart contracts and development process as a whole, ensuring best practices are followed, allowing improved efficiency and security based on established industry and academic practices.
4. Specific, itemized, and actionable recommendations to assist in securing the smart contract(s) in question.

Audit Methodology & Techniques

Issues looked for:

- Unchecked External Call
- Costly Loops
- Denial of service / logical oversights
- Overpowered Owner
- Arithmetic Precision
- Relying on tx.origin
- Overflow/Underflow
- Unsafe Type Inference
- Improper Transfer
- In-Loop Transfer
- TimeStamp Dependence
- Code clones, functionality duplication
- Transaction-ordering dependence
- Mishandled exceptions and callstack limits
- Unsafe external calls
- Reentrancy and cross-function vulnerabilities
- Access control
- Centralization of power
- Business logic contradicting the specification
- Arbitrary token minting

Contract Checklist

ZinToken.sol

Contract Vulnerability	
Integer Overflow	Pass
Race Condition	Pass
Denial of Service	Pass
Logical Vulnerability	Pass
Hardcoded Address	Not Applicable
Function Input Parameter Check	Pass
Function Access Control Check	Pass
Random Number Generation	Not Applicable
Random Number Use	Not Applicable
Contract Specification	
Solidity Compiler Version	Fail
Event Use	Pass
Fallback Function Use	Pass
Contructor Use	Pass
Function Visibility Declaration	Pass
Variable Storage Declaration	Pass
Deprecated Keyword Use	Pass
ERC20/223 Standard	Pass
ERC721 Standard	Not Applicable
Business Risk	
Able to Arbitratrily Create Token	Pass
Able to Arbitrarily Destroy Token	Pass
Can Suspend Transactions	Pass
Short Address Attack	Pass
Gas Optimization	
assert()/require()/revert() misused	Pass
Loop Optimization	Pass
Storage Optimization	Pass

Contract Checklist

ZinCrowdsale.sol

Contract Vulnerability	
Integer Overflow	Pass
Race Condition	Pass
Denial of Service	Pass
Logical Vulnerability	Fail
Hardcoded Address	Pass
Function Input Parameter Check	Pass
Function Access Control Check	Pass
Random Number Generation	Not Applicable
Random Number Use	Not Applicable
Contract Specification	
Solidity Compiler Version	Fail
Event Use	Fail
Fallback Function Use	Pass
Contructor Use	Pass
Function Visibility Declaration	Fail
Variable Storage Declaration	Pass
Deprecated Keyword Use	Pass
ERC20/223 Standard	Pass
ERC721 Standard	Not Applicable
Business Risk	
Able to Arbitratrily Create Token	Pass
Able to Arbitrarily Destroy Token	Not Applicable
Can Suspend Transactions	Fail
Short Address Attack	Pass
Gas Optimization	
assert()/require()/revert() misused	Pass
Loop Optimization	Pass
Storage Optimization	Pass

Issue Classification

Critical

These issues in the smart contract can have catastrophic implications that could ruin your reputation, disrupt the contract's functionality, and impact the client and your user's sensitive information.

Informational

This issue relates to style and security best practices but does not pose an immediate risk.

Medium

An issue classified as medium has relatively small risk and isn't exploitable to circumvent desired functionality and could not have financial consequences but could put user's sensitive information at risk.

Acknowledged

The issue remains in the code but is a result of an intentional business or design decision.

Low

An issue classified as informational does not pose an immediate threat to disruption of functionality and could not be exploited on a recurring basis, however, it should be considered for security best practices or code integrity.

Unresolved

Although the client has been informed of the risk, it was decided to accept it because it was not relevant in the functionality of the smart contract.

Undetermined

The impact of the issue is uncertain and more investigation is required to understand the repercussions of the issue.

Mitigated

Actions were taken to minimize the impact or likelihood of the risk.

Detailed Analysis

5.1 ZinCrowdsale.sol

No Way of Ending the Crowdsale Early

Explanation:

The only way the crowdsale can terminate is if the weiRaised goal is reached or if the closing time has transpired. Upon discovery of vulnerability or in the event that the Zin team makes changes to their fund raising objectives, the crowdsale can not terminate prematurely.

Recommendation:

Implement a function to suspend the activity of the contract with proper security measures in case the aforementioned situation arises.

5.2 finalization() ZinCrowdsale.sol 364

Unsafe Use of Arithmetic Expression

Explanation:

Best practice is to use SafeMath library to perform arithmetic operations.

Recommendation:

Use safemath **sub()** function to perform the substraction between totalTokens and usedTokens instead.

5.3 finalize()

ZinCrowdsale.sol 269-278

Reentrancy Vulnerability

Explanation:

This function is not following the checks-effects-interactions pattern where you change the state, emit the event, and then call the function.

Recommendation:

Set **isFinalized** to true before emitting **Finalized()** or use the boolean **token.tranfer()** returns inside of **finalization()** to ensure that the remaining tokens are transferred out of the ZinCrowdsale contract and that you are using the return value from **token.transfer()**.

Note: It is not likely that this would be used as an attack vector.

5.4 finalization()

ZinCrowdsale.sol 365

Unused Return Value

Explanation:

It is best practice to store the return value from a external call in a local or state variable.

Recommendation:

Set **isFinalized** to true using the boolean **token.tranfer()** returns inside of **finalization()** to ensure that the remaining tokens are transferred from the ZinCrowdsale contract before finalizing the sale.

5.5 File Level

All Files 5

Unlocked Pragma

Explanation:

pragma ^0.6.0 is wrong because caller.send.value was changed to caller.send{value} starting in version 0.6.4 and the contract won't compile on a version earlier than 0.6.4.

Recommendation:

Change pragma to 0.6.8, 0.6.10, or 0.6.11.

5.6 finalize()

ZinCrowdsale.sol 269

Function Visibility Declaration

Explanation:

Best practice is to restrict the visibility of variables, mappings, functions etc. as much as possible based on needs of interaction.

Recommendation:

Change the visibility of finalize() from public to external.

5.7 transferOwnership()

ZinCrowdsale.sol 258

Function Visibility Declaration

Explanation:

Best practice is to restrict the visibility of variables, mappings, functions etc. as much as possible based on needs of interaction.

Recommendation:

Change the visibility of transferOwnership() from public to external.

5.8 capReached()

ZinCrowdsale.sol 250

Function Visibility Declaration

Explanation:

Best practice is to restrict the visibility of variables, mappings, functions etc. as much as possible based on needs of interaction.

Recommendation:

Change the visibility of capReached() from public to external.

5.9 ZinCrowdsale.constructor()

ZinCrowdsale.sol 377

Code Integrity

Explanation:

Code implementations should be as close as possible to their associated logical behavior while maintaining correctness.

Recommendation:

The comment says that clients are to receive 10 ZIN tokens for every Ether but with a rate 25000ZINbits/wei they would receive 250. If this is the intended behavior then the comment should reflect that new rate otherwise the rate should be 0.1 ZINbits/wei.

5.10 SafeMath library

All Files

Gas Optimization

Explanation:

Safemath library is being used in both contracts and hardcoded in both contracts.

Recommendation:

Put SafeMath library in its own file and import it in each contract in order to reduce gas costs.

5.11 finalize()

ZinCrowdsale.sol 275

Readability

Explanation:

Events are supposed to be prefixed by the **emit** keyword.

Recommendation:

Prefix **Finalized()** event call with the **emit** keyword.

5.12 transferOwnership()

ZinCrowdsale.sol 261

Readability

Explanation:

Events are supposed to be prefixed by the **emit** keyword.

Recommendation:

Prefix **Finalized()** event call with the **emit** keyword.

5.13 library SafeMath

All Files

Gas Optimization

Explanation:

Unused functions div() and mod() increasing gas costs.

Recommendation:

Remove unused functions from the SafeMath library.

5.14 abstract contract Context

ZinToken.sol

Gas Optimization

Explanation:

Best practice is to make code as modular as possible.

Recommendation:

Put abstract contract Context in its own file and import as needed.

5.15 interface ERC20

ZinToken.sol

Gas Optimization

Explanation:

Best practice is to make code as modular as possible.

Recommendation:

Put interface ERC20 in its own file and import as needed.

5.16 contract ERC20

ZinToken.sol

Gas Optimization

Explanation:

Best practice is to make code as modular as possible.

Recommendation:

Put contract ERC20 in its own file and import as needed.

5.17 contract ZinToken

ZinToken.sol

Gas Optimization

Explanation:

Best practice is to make code as modular as possible.

Recommendation:

Put contract ZinToken in its file.

Notable Concerns & Suggestions

Overall Thoughts

Zin Finance has developed and maintained an excellent codebase that demonstrates competency and care in implementing the functionality needed to serve the Zin Finance platform and users. With the completion of the unit testing suite and the recognition of the contract suggestions The Blockchain Auditor assesses the Zin Finance team as having taken the utmost care in providing secure, efficient and audited smart contract(s).

The Blockchain Auditor is honored to have the opportunity to be partnered with Zin Finance to provide enhanced security and efficiency for the Zin Finance platform.

Not only does the Zin Finance team display excellency in their implementations and demonstrates clear intent on providing the best possible platform for its users, our team also appreciates and is aligned with Zin Finance's goal of bringing high-value opportunities in crypto investing to its larger and non-traditional global audience.

The BlockChain Auditor

- Jorge Martinez

Appendix A

File Fingerprints

ZinToken.sol	8a2d2ad282be9d587cdd9b63807c9b62a3e506ca6929191135ac024390240fd3
ZinCrowdsale.sol	70d2af098bfdeaa5dda448ed18893ce5c0ce6b5d3ac8e5d669086a478de11395