RD AUDITORS

AIOW SMART CONTRACT, CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: AIOW
Prepared on: 17th Sept 2021
Platform: ETH
Language: Solidity

TABLE OF CONTENTS

THIS DOCUMENT MAY CONTAIN CONFIDENTIAL INFORMATION ABOUT ITS SYSTEMS AND INTELLECTUAL PROPERTY OF THE CUSTOMER AS WELL AS INFORMATION ABOUT POTENTIAL VULNERABILITIES AND METHODS OF THEIR EXPLOITATION.

THE REPORT CONTAINING CONFIDENTIAL INFORMATION CAN BE USED INTERNALLY BY THE CUSTOMER OR IT CAN BE DISCLOSED PUBLICLY AFTER ALL VULNERABILITIES ARE FIXED - UPON THE DECISION OF THE CUSTOMER.

Document

Name	Smart Contract Code Review and Security Analysis Report of AIOW
Platform	ETH / Solidity
File 1	AIOW.sol
MD5 hash	0715B2491AB7506552165917B4D09D24
SHA256 hash	5435DD6177608060F11F7633FAA532E4C46A9A9672EF3F6C4FBCE31B62C471A3
File 2	PriceFeedMock.sol
MD5 hash	192AD574680FC7A80FC121E652F015EE
SHA256 hash	6FD28156B4DBFC5138689A44EF2E93211A549A6A879D36C05932768BDD246A96
File 3	TokenSalePresale.sol
MD5 hash	522B2DABED5A8334FBDB15A210ABD0C9
SHA256 hash	39628E86F92C92DBFF40BB58525EAE2DA9706CF993EFAE1AF491EDACBAB50D41
File 4	TokenSale.sol
MD5 hash	94242461670B87B145F8713CC8CDB63E
SHA256 hash	D26E2CA6325E2632056FAB2FADE9C1E61856C619A93A15B2797BEF3C8A81753A
File 5	TokenSaleStrategic.sol
MD5 hash	A03E594041A584CF7B3228248C1044AE
SHA256 hash	787B144168E3DA235F664E940D6B5F2A350F63E3BEC167DFF373A55B9A760966

File 6	TokenSalePrivate.sol
MD5 hash	0753101D9B319AADFAF0CF3403819FC5
SHA256 hash	6A151BB6BE3AD581B130E92E47BD5F17D09813F22F88787C0B2ACA2D7FF68625
Date	17/09/2021

Introduction

RD Auditors (Consultant) were contracted by AIOW (Customer) to conduct a Smart Contracts Code Review and Security Analysis. This report represents the findings of the security assessment of the customer`s smart contracts and its code review conducted between
11 - 17 Sept 2021.


This contract consists of six files.

Project Scope

The scope of the project is a smart contract.

We have scanned this smart contract for commonly known and more specific vulnerabilities, below are those considered (the full list includes but is not limited to):

- Reentrancy

- Timestamp Dependence

- Gas Limit and Loops

- DoS with (Unexpected) Throw

- DoS with Block Gas Limit

- Transaction-Ordering Dependence

- Byte array vulnerabilities

- Style guide violation

- Transfer forwards all gas

- ERC20 API violation

- Malicious libraries

- Compiler version not fixed

- Unchecked external call - Unchecked math

- Unsafe type inference

- Implicit visibility level

Executive Summary

According to the assessment, the customer's solidity smart contract is well-secured.

Insecure	Poorly secured	Secure	Well-secured

You are here
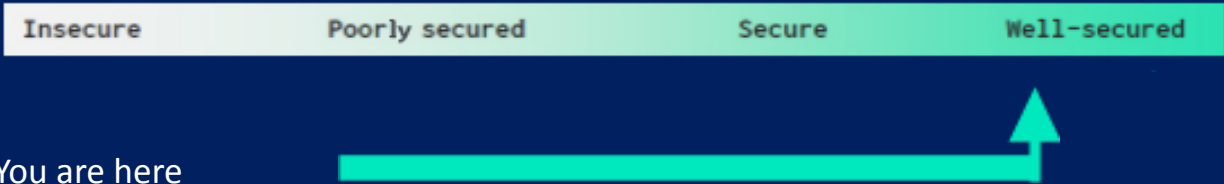
Automated checks are with smartDec, Mythril, Slither and remix IDE. All issues were performed by our team, which included the analysis of code functionality, manual audit found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the audit overview section. The general overview is presented in the AS-IS section and all issues found are located in the audit overview section.

We found 0 critical, 0 high, 0 medium, 0 low and 0 very low level issues.

Code Quality

Please find a link that, within this report IERC20, ownable taken from the popular open source.

The libraries within this smart contract are part of a logical algorithm. A library is a different type of smart contract that contains reusable code. Once deployed on the blockchain (only once), it is assigned to a specific address and its properties/methods can be reused many times by other contracts.

The AIOW team has not provided scenario and unit test scripts, which would help to determine the integrity of the code in an automated way.

Overall, the code is almost commented. Commenting can provide rich documentation for functions, return variables and more. Use of Ethereum Natural Language Specification Format (NatSpec) for commenting is recommended.

Documentation

The hash of that file is mentioned in the table. As mentioned above, It's recommended to write comments in the smart contract code, so anyone can quickly understand the programming flow as well as complex code logic.

Comments are very helpful in understanding the overall architecture of the protocol. It also provides a clear overview of the system components, including helpful details, like the lifetime of the background script.

Use of Dependencies

As per our observation, the libraries are used in this smart contract infrastructure. Those were based on well known industry standard open source projects and even core code blocks that are written well and systematically.

AS-IS Overview

AIOW

File And Function Level Report

File: AIOW.sol

Contract:	AIOW
Inherit:	Ownable
Observation:	Passed
Test Report:	Passed
Score:	Passed
Conclusion:	Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	_beforeTokenTransfer	write	Passed	All Passed	No Issue	Passed
2	calcBalanceUnlocked	read	Passed	All Passed	No Issue	Passed
3	calcBalanceLockedOfInboundRule	read	Passed	All Passed	No Issue	Passed
4	calcBalanceLocked	read	Passed	All Passed	No Issue	Passed
5	registerTransferRule	write	Passed	All Passed	No Issue	Passed
6	balanceOf	read	Passed	All Passed	No Issue	Passed
7	allowance	read	Passed	All Passed	No Issue	Passed
8	transfer	write	Passed	All Passed	No Issue	Passed
9	transferFrom	write	Passed	All Passed	No Issue	Passed
10	_transfer	write	Passed	All Passed	No Issue	Passed
11	approve	write	Passed	All Passed	No Issue	Passed
12	increaseAllowance	write	Passed	All Passed	No Issue	Passed
13	decreaseAllowance	write	Passed	All Passed	No Issue	Passed
14	_approve	write	Passed	All Passed	No Issue	Passed
15	balanceStatsOf	read	Passed	All Passed	No Issue	Passed
16	inboundRuleStatsOf	read	Passed	All Passed	No Issue	Passed
17	getInboundTransferRules	read	Passed	All Passed	No Issue	Passed

File: PriceFeedMock.sol

Contract: PriceFeedMock
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	updatePrice	write	Passed	All Passed	No Issue	Passed
2	latestRoundData	read	Passed	All Passed	No Issue	Passed

File: TokenSalePresale.sol

Contract: TokenSalePresale
Import: TokenSale
Inherit: TokenSale
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

File: TokenSale.sol

Contract: TokenSale
Inherit: Ownable,PriceFetcher
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	getTokensLeftForSale	read	Passed	All Passed	No Issue	Passed
2	ethToToken	read	Passed	All Passed	No Issue	Passed
3	tokenToEth	read	Passed	All Passed	No Issue	Passed
4	isOpen	read	Passed	All Passed	No Issue	Passed
5	hasEnded	read	Passed	All Passed	No Issue	Passed
6	getMaxTokensAllowedToBuyOf	read	Passed	All Passed	No Issue	Passed
7	getMinTokensAllowedToBuyOf	read	Passed	All Passed	No Issue	Passed
8	buyTokens	write	Passed	All Passed	No Issue	Passed
9	withdrawEth	write	Passed	All Passed	No Issue	Passed
10	transferUnsoldTokens	write	Passed	All Passed	No Issue	Passed

File: TokenSaleStrategic.sol

Contract: TokenSaleStrategic
Import: TokenSale
Inherit: TokenSale,Whitelist
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	whitelistAdd	write	Passed	All Passed	No Issue	Passed
2	_whitelistAdd	write	Passed	All Passed	No Issue	Passed
3	whitelistBatchAdd	write	Passed	All Passed	No Issue	Passed
4	whitelistRemove	write	Passed	All Passed	No Issue	Passed
5	buyTokens	write	Passed	All Passed	No Issue	Passed

File: TokenSalePrivate.sol

Contract: TokenSalePrivate
Import: TokenSale
Inherit: TokenSale,Whitelist
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	buyTokens	write	Passed	All Passed	No Issue	Passed

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to lost tokens etc.
High	High level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g. public access to crucial functions.
Medium	Medium level vulnerabilities are important to fix; however, they cannot lead to lost tokens.
Low	Low level vulnerabilities are most related to outdated, unused etc. These code snippets cannot have a significant impact on execution.
Lowest Code Style/ Best Practice	Lowest level vulnerabilities, code style violations and information statements cannot affect smart contract execution and can be ignored.

Audit Findings

Critical

No critical severity vulnerabilities were found.

High

No high severity vulnerabilities were found.

Medium

No medium severity vulnerabilities were found.

Low

No low severity vulnerabilities were found.

Very Low

No very low severity vulnerabilities were found.

Conclusion

We were given a contract file and have used all possible tests based on the given object. The contract is written systematically, so it is ready to go for production.

Since possible test cases can be unlimited and developer level documentation (code flow diagram with function level description) not provided, for such an extensive smart contract protocol, we provide no such guarantee of future outcomes. We have used all the latest static tools and manual observations to cover maximum possible test cases to scan everything.

The security state of the reviewed contract is now "well secured"

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review:

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis:

Our audit techniques included manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's web site to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

Documenting Results:

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyse the feasibility of an attack in a live system.

Suggested Solutions:

We search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinised by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

Disclaimers

RD Auditors Disclaimer

The smart contracts given for audit have been analysed in accordance with the best industry practices at the date of this report, in relation to: cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment and functionality (performing the intended functions).

Because the total number of test cases are unlimited, the audit makes no statements or warranties on the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only - we recommend proceeding with several independent audits and a public bug bounty program to ensure security of smart contracts.

Technical Disclaimer

Smart contracts are deployed and executed on the blockchain. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to hacks. Thus, the audit can't guarantee explicit security of the audited smart contracts.

RD
AUDITORS

Email: info@rdauditors.com

Website: www.rdauditors.com