



CredShields

Smart Contract Audit

Sept 24th, 2022 • CONFIDENTIAL

Description

This document details the process and result of the Wahed Token smart contract audit performed by CredShields Technologies PTE. LTD. on behalf of Wahed Projects between Aug 9th, 2022, and Aug 15th, 2022. And a retest was performed on 20th Sept 2022.

Author

Shashank (Co-founder, CredShields)

shashank@CredShields.com

Reviewers

Aditya Dixit (Research Team Lead)

aditya@CredShields.com

Prepared for

Wahed Projects

Table of Contents

1. Executive Summary	3
State of Security	4
2. Methodology	5
2.1 Preparation phase	5
2.1.1 Scope	6
2.1.2 Documentation	6
2.1.3 Audit Goals	6
2.2 Retesting phase	8
2.3 Vulnerability classification and severity	8
2.4 CredShields staff	12
3. Findings	13
3.1 Findings Overview	13
3.1.1 Vulnerability Summary	13
3.1.2 Findings Summary	15
4. Remediation Status	19
5. Bug Reports	20
Bug ID#1	20
Floating Pragma	20
Bug ID#2	22
Functions should be declared External	22
Bug ID#3	24
Missing Multiple Zero Address Validations	24
Bug ID#4	26
Missing Events on important functions	26
Bug ID#5	28
Missing Constant Attribute in Variables	28
Bug ID#6	30
Gas Optimization in year and month	30
Bug ID#7	32

Gas Optimization due to redundant codes	32
Bug ID#8	34
Missing Timelock on critical state changes	34
Bug ID#9	36
Missing NatSpec Comments	36
6. Disclosure	37

1. Executive Summary

Wahed Projects engaged CredShields to perform a smart contract audit from Aug 9th, 2022, to Aug 15th, 2022. During this timeframe, eight (8) vulnerabilities were identified. **A retest was performed on 20th Sept 2022 and none of the bugs are in pending fix state.**

During the audit, zero (0) vulnerability was found that had a severity rating of either High or Critical. These vulnerabilities represent the greatest immediate risk to "Wahed Projects" and should be prioritized for remediation, and fortunately, none were found.

The table below shows the in-scope assets and breakdown of findings by severity per asset. Section 2.3 contains more information on how severity is calculated.

Assets in Scope	Critical	High	Medium	Low	info	Gas	Σ
Wahed Token	0	0	0	4	1	4	9
	0	0	0	4	1	4	9

Table: Vulnerabilities Per Asset in Scope

The CredShields team conducted the security audit to focus on identifying vulnerabilities in Wahed Project's scope during the testing window while abiding by the policies set forth by Wahed Project's team.



State of Security

Maintaining a healthy security posture requires constant review and refinement of existing security processes. Running a CredShields continuous audit allows Wahed Project's internal security team and development team to not only uncover specific vulnerabilities but gain a better understanding of the current security threat landscape.

We recommend running regular security assessments to identify any vulnerabilities introduced after Wahed Projects introduces new features or refactors the code.

Reviewing the remaining resolved reports for a root cause analysis can further educate Wahed Project's internal development and security teams and allow manual or automated procedures to be put in place to eliminate entire classes of vulnerabilities in the future. This proactive approach helps contribute to future-proofing the security posture of Wahed Projects assets.

2. Methodology

Wahed Project's engaged CredShields to perform a Wahed Token smart contract audit. The following sections cover how the engagement was put together and executed.

2.1 Preparation phase

CredShields team read all the provided documents and comments in the smart-contract code to understand the contract's features and functionalities. The team reviewed all the functions and prepared a mind map to review for possible security vulnerabilities in the order of the function with more critical and business-sensitive functionalities for the refactored code.

The team deployed a self-hosted version of the smart contract to verify the assumptions and validation of the vulnerabilities during the audit phase.

A testing window from Aug 9th, 2022, to Aug 15th, 2022, was agreed upon during the preparation phase.

2.1.1 Scope

During the preparation phase, the following scope for the engagement was agreed-upon:

IN SCOPE ASSETS
https://bscscan.com/address/0x733708a9869066a82b741410855aa756646c0f18

Table: List of Files in Scope

2.1.2 Documentation

N/A - Documentation was not required as the code was self sufficient for understanding the project.

2.1.3 Audit Goals

CredShields' methodology uses individual tools and methods; however, tools are just used for aids. The majority of the audit methods involve manually reviewing the smart contract source code. The team followed the standards of the [SWC registry](#) for testing along with an extended self-developed checklist based on industry standards, but it was not limited to it. The team focused heavily on understanding the core concept behind all the functionalities along with preparing test and edge cases. Understanding the business logic and how it could have been exploited.

The audit's focus was to verify that the smart contract system is secure, resilient, and working according to its specifications. Breaking the audit activities into the following three categories:

- **Security** - Identifying security-related issues within each contract and the system of contracts.
- **Sound Architecture** - Evaluation of the architecture of this system through the lens of established smart contract best practices and general software best practices.
- **Code Correctness and Quality** - A full review of the contract source code. The primary areas of focus include:
 - Correctness
 - Readability
 - Sections of code with high complexity
 - Improving scalability
 - Quantity and quality of test coverage

2.2 Retesting phase

Wahed Projects is actively partnering with CredShields to validate the remediations implemented towards the discovered vulnerabilities.

2.3 Vulnerability classification and severity

Discovering vulnerabilities is important, but estimating the associated risk to the business is just as important.

To adhere to industry guidelines, CredShields follows OWASP's Risk Rating Methodology. This is calculated using two factors - **Likelihood** and **Impact**. Each of these parameters can take three values - **Low**, **Medium**, and **High**.

These depend upon multiple factors such as Threat agents, Vulnerability factors (Ease of discovery and exploitation, etc.), and Technical and Business Impacts. The likelihood and the impact estimate are put together to calculate the overall severity of the risk.

CredShields also define an **Informational** severity level for vulnerabilities that do not align with any of the severity categories and usually have the lowest risk involved.

Overall Risk Severity				
Impact	HIGH	Medium	High	Critical
	MEDIUM	Low	Medium	High
	LOW	Note	Low	Medium
		LOW	MEDIUM	HIGH
	Likelihood			

Overall, the categories can be defined as described below -

1. Informational

We believe in the importance of technical excellence and pay a great deal of attention to its details. Our coding guidelines, practices, and standards help ensure that our software is stable and reliable.

Informational vulnerabilities should not be a cause for alarm but rather a chance to improve the quality of the codebase by emphasizing readability and good practices. They do not represent a direct risk to the Contract but rather suggest improvements and the best practices that can not be categorized under any of the other severity categories.

Code maintainers should use their own judgment as to whether to address such issues.

2. Low

Vulnerabilities in this category represent a low risk to the Smart Contract and the organization. The risk is either relatively small and could not be exploited on a recurring basis, or a risk that the client indicates is not important or significant, given the client's business circumstances.

3. Medium

Medium severity issues are those that are usually introduced due to weak or erroneous logic in the code.

These issues may lead to exfiltration or modification of some of the private information belonging to the end-user, and exploitation would be detrimental to the client's reputation under certain unexpected circumstances or conditions. These conditions are outside the control of the adversary.

These issues should eventually be fixed under a certain timeframe and remediation cycle.

4. High

High severity vulnerabilities represent a greater risk to the Smart Contract and the organization. These vulnerabilities may lead to a limited loss of funds for some of the end-users.

They may or may not require external conditions to be met, or these conditions may be manipulated by the attacker, but the complexity of exploitation will be higher.

These vulnerabilities, when exploited, will impact the client's reputation negatively.

They should be fixed immediately.

5. Critical

Critical issues are directly exploitable bugs or security vulnerabilities. These issues do not require any external conditions to be met.

The majority of vulnerabilities of this type involve a loss of funds and Ether from the Smart Contracts and/or from their end-users.

The issue puts the vast majority of, or large numbers of, users' sensitive information at risk of modification or compromise.

The client's reputation will suffer a severe blow, or there will be serious financial repercussions.

Considering the risk and volatility of smart contracts and how they use gas as a method of payment to deploy the contracts and interact with them, gas optimization becomes a major point of concern. To address this, CredShields also introduces another severity category called "**Gas Optimization**" or "**Gas**". This category deals with code optimization techniques and refactoring due to which Gas can be conserved.

2.4 CredShields staff

The following individual at CredShields managed this engagement and produced this report:

- **Shashank, Co-founder CredShields**
 - shashank@CredShields.com

Please feel free to contact this individual with any questions or concerns you have around the engagement or this document.

3. Findings

This chapter contains the results of the security assessment. Findings are sorted by their severity and grouped by the asset and SWC classification. Each asset section will include a summary. The table in the executive summary contains the total number of identified security vulnerabilities per asset per risk indication.

3.1 Findings Overview

3.1.1 Vulnerability Summary

During the security assessment, six (6) security vulnerabilities were identified in the asset.

VULNERABILITY TITLE	SEVERITY	SWC Vulnerability Type
Floating Pragma	Low	Floating Pragma (SWC-103)
Functions should be declared External	Gas	Gas Optimization
Missing Multiple Zero Address Validations	Low	Missing Input Validation
Missing Events on important functions	Low	Missing Best Practices
Missing Constant Attribute in Variables	Gas	Gas Optimization
Gas Optimization in year and month	Gas	Gas Optimization
Gas Optimization due to redundant codes	Gas	Gas Optimization
Missing Timelock on critical state changes	Low	Missing best practices

Missing NatSpec Comments	Informational	Missing best practices
--------------------------	---------------	------------------------

Table: Findings in Smart Contracts



3.1.2 Findings Summary

SWC ID	SWC Checklist	Test Result	Notes
SWC-100	Function Default Visibility	Not Vulnerable	Not applicable after v0.5.X (Currently using solidity v >= 0.8.6)
SWC-101	Integer Overflow and Underflow	Not Vulnerable	The issue persists in versions before v0.8.X.
SWC-102	Outdated Compiler Version	Not Vulnerable	Version 0^8.0 and above is used
SWC-103	Floating Pragma	Vulnerable	Contract uses floating pragma
SWC-104	Unchecked Call Return Value	Not Vulnerable	call() is not used
SWC-105	Unprotected Ether Withdrawal	Not Vulnerable	Appropriate function modifiers and require validations are used on sensitive functions that allow token or ether withdrawal.
SWC-106	Unprotected SELFDESTRUCT Instruction	Not Vulnerable	selfdestruct() is not used anywhere
SWC-107	Reentrancy	Not Vulnerable	No notable functions were vulnerable to it.
SWC-108	State Variable Default Visibility	Not Vulnerable	Not Vulnerable
SWC-109	Uninitialized Storage Pointer	Not Vulnerable	Not vulnerable after compiler version, v0.5.0

SWC-110	Assert Violation	Not Vulnerable	Asserts are not in use.
SWC-111	Use of Deprecated Solidity Functions	Not Vulnerable	None of the deprecated functions like <code>block.blockhash()</code> , <code>msg.gas</code> , <code>throw</code> , <code>sha3()</code> , <code>callcode()</code> , <code>suicide()</code> are in use
SWC-112	Delegatecall to Untrusted Callee	Not Vulnerable	Not Vulnerable.
SWC-113	DoS with Failed Call	Not Vulnerable	No such function was found.
SWC-114	Transaction Order Dependence	Not Vulnerable	Not Vulnerable.
SWC-115	Authorization through tx.origin	Not Vulnerable	<code>tx.origin</code> is not used anywhere in the code
SWC-116	Block values as a proxy for time	Not Vulnerable	<code>Block.timestamp</code> is used but the contract doesn't make strict check so not an issue.
SWC-117	Signature Malleability	Not Vulnerable	Not used anywhere
SWC-118	Incorrect Constructor Name	Not Vulnerable	All the constructors are created using the <code>constructor</code> keyword rather than functions.
SWC-119	Shadowing State Variables	Not Vulnerable	Not applicable as this won't work during compile time after version <code>0.6.0</code>
SWC-120	Weak Sources of Randomness from Chain Attributes	Not Vulnerable	Random generators are not used.
SWC-121	Missing Protection against Signature Replay Attacks	Not Vulnerable	No such scenario was found

SWC-122	Lack of Proper Signature Verification	Not Vulnerable	Not used anywhere
SWC-123	Requirement Violation	Not Vulnerable	Not vulnerable
SWC-124	Write to Arbitrary Storage Location	Not Vulnerable	No such scenario was found
SWC-125	Incorrect Inheritance Order	Not Vulnerable	No such scenario was found
SWC-126	Insufficient Gas Griefing	Not Vulnerable	No such scenario was found
SWC-127	Arbitrary Jump with Function Type Variable	Not Vulnerable	Jump is not used.
SWC-128	DoS With Block Gas Limit	Not Vulnerable	Not Vulnerable.
SWC-129	Typographical Error	Not Vulnerable	No such scenario was found
SWC-130	Right-To-Left-Override control character (U+202E)	Not Vulnerable	No such scenario was found
SWC-131	Presence of unused variables	Not Vulnerable	No such scenario was found
SWC-132	Unexpected Ether balance	Not Vulnerable	No such scenario was found
SWC-133	Hash Collisions With Multiple Variable Length Arguments	Not Vulnerable	abi.encodePacked() or other functions are not used.
SWC-134	Message call with hardcoded gas amount	Not Vulnerable	Not used anywhere in the code
SWC-135	Code With No Effects	Not Vulnerable	No such scenario was found
SWC-136	Unencrypted Private Data On-Chain	Not Vulnerable	No such scenario was found

