



# CredShields

# Smart Contract Audit

---

**April 6th, 2024 • CONFIDENTIAL**

## **Description**

This document details the process and result of the Smart Contract audit performed by CredShields Technologies PTE. LTD. on behalf of Factiiv between April 4th, 2024, and April 5th, 2024. And a retest was performed on April 5th, 2024.

## **Author**

Shashank (Co-founder, CredShields)

[shashank@CredShields.com](mailto:shashank@CredShields.com)

## **Reviewers**

Aditya Dixit (Research Team Lead)

[aditya@CredShields.com](mailto:aditya@CredShields.com)

## **Prepared for**

Factiiv

# Table of Contents

<b>1. Executive Summary</b>	<b>2</b>
State of Security	3
<b>2. Methodology</b>	<b>4</b>
2.1 Preparation phase	4
2.1.1 Scope	5
2.1.2 Documentation	5
2.1.3 Audit Goals	5
2.2 Retesting phase	6
2.3 Vulnerability classification and severity	6
2.4 CredShields staff	9
<b>3. Findings</b>	<b>10</b>
3.1 Findings Overview	10
3.1.1 Vulnerability Summary	10
3.1.2 Findings Summary	11
<b>4. Remediation Status</b>	<b>14</b>
<b>5. Bug Reports</b>	<b>15</b>
Bug ID #1 [Won't Fix]	15
Outdated Pragma version	15
Bug ID #2 [Won't Fix]	17
Bytes Constants More Efficient Than String Literal	17
<b>6. Disclosure</b>	<b>18</b>

# 1. Executive Summary

Factiiv engaged CredShields to perform a smart contract audit from April 4th, 2024, to April 5th, 2024. During this timeframe, Two (2) vulnerabilities were identified. **A retest was performed on April 5th, 2024, and all the bugs have been addressed.**

During the audit, Zero (0) vulnerabilities were found with a severity rating of either High or Critical. These vulnerabilities represent the greatest immediate risk to "Factiiv Smart Contract" and should be prioritized for remediation, and fortunately, none were found.

The table below shows the in-scope assets and a 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
Factiiv	0	0	0	1	0	1	2
	0	0	0	1	0	1	2

*Table: Vulnerabilities Per Asset in Scope*

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

## State of Security

To maintain a robust security posture, it is essential to continuously review and improve upon current security processes. Utilizing CredShields' continuous audit feature allows both Factiiv's internal security and development teams to not only identify specific vulnerabilities, but also gain a deeper understanding of the current security threat landscape.

To ensure that vulnerabilities are not introduced when new features are added, or code is refactored, we recommend conducting regular security assessments. Additionally, by analyzing the root cause of resolved vulnerabilities, the internal teams at Factiiv can implement both manual and automated procedures to eliminate entire classes of vulnerabilities in the future. By taking a proactive approach, Factiiv can future-proof its security posture and protect its assets.

## 2. Methodology

---

Factiiv engaged CredShields to perform a Smart Contract audit. The following sections cover how the engagement was put together and executed.

### 2.1 Preparation phase

The CredShields team meticulously reviewed all provided documents and comments in the smart-contract code to gain a thorough understanding of the contract's features and functionalities. They meticulously examined all functions and created a mind map to systematically identify potential security vulnerabilities, prioritizing those that were more critical and business-sensitive for the refactored code. To confirm their findings, the team deployed a self-hosted version of the smart contract and performed verifications and validations during the audit phase.

A testing window from April 4th, 2024, to April 5th, 2024, 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
<a href="https://github.com/Factiiv-Projects/FACTTokens/blob/a4221c38d019dbb1ffa1aea3ade29979f5df9174/contracts/">https://github.com/Factiiv-Projects/FACTTokens/blob/a4221c38d019dbb1ffa1aea3ade29979f5df9174/contracts/</a>

*Table: List of Files in Scope*

### 2.1.2 Documentation

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

### 2.1.3 Audit Goals

CredShields uses both in-house tools and manual methods for comprehensive smart contract security auditing. The majority of the audit is done by manually reviewing the contract source code, following SWC registry standards, and an extended industry standard self-developed checklist. The team places emphasis on understanding core concepts, preparing test cases, and evaluating business logic for potential vulnerabilities.

## 2.2 Retesting phase

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

## 2.3 Vulnerability classification and severity

CredShields follows OWASP's Risk Rating Methodology to determine the risk associated with discovered vulnerabilities. This approach considers two factors - Likelihood and Impact - which are evaluated with three possible values - **Low**, **Medium**, and **High**, based on factors such as Threat agents, Vulnerability factors, Technical and Business Impacts. The overall severity of the risk is calculated by combining the likelihood and impact estimates.

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 prioritize technical excellence and pay attention to detail in our coding practices. Our guidelines, standards, and best practices help ensure software stability and reliability. Informational vulnerabilities are opportunities for improvement and do

not pose a direct risk to the contract. Code maintainers should use their own judgment on whether to address them.

## **2. Low**

Low-risk vulnerabilities are those that either have a small impact or can't be exploited repeatedly or those the client considers insignificant based on their specific business circumstances.

## **3. Medium**

Medium-severity vulnerabilities are those caused by weak or flawed logic in the code and can lead to exfiltration or modification of private user information. These vulnerabilities can harm the client's reputation under certain conditions and should be fixed within a specified timeframe.

## **4. High**

High-severity vulnerabilities pose a significant risk to the Smart Contract and the organization. They can result in the loss of funds for some users, may or may not require specific conditions, and are more complex to exploit. These vulnerabilities can harm the client's reputation and should be fixed immediately.

## **5. Critical**

Critical issues are directly exploitable bugs or security vulnerabilities that do not require specific conditions. They often result in the loss of funds and Ether from Smart Contracts or users and put sensitive user information at risk of compromise



or modification. The client's reputation and financial stability will be severely impacted if these issues are not addressed immediately.

## **6. Gas**

To address the risk and volatility of smart contracts and the use of gas as a method of payment, CredShields has introduced a "Gas" severity category. This category deals with optimizing code and refactoring to conserve gas.

## 2.4 CredShields staff

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

- **Shashank, Co-founder CredShields**
  - [shashank@CredShields.com](mailto: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, Two (2) security vulnerabilities were identified in the asset.

VULNERABILITY TITLE	SEVERITY	SWC   Vulnerability Type
Outdated Pragma version	Low	Outdated Pragma
Bytes Constants More Efficient Than String Literal	Gas	Gas Optimization

*Table: Findings in Factiiv Smart Contracts*

### 3.1.2 Findings Summary

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

SWC-110	<a href="#">Assert Violation</a>	Not Vulnerable	Asserts are not in use.
SWC-111	<a href="#">Use of Deprecated Solidity Functions</a>	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	<a href="#">Delegatecall to Untrusted Callee</a>	Not Vulnerable	Not Vulnerable.
SWC-113	<a href="#">DoS with Failed Call</a>	Not Vulnerable	No such function was found.
SWC-114	<a href="#">Transaction Order Dependence</a>	Not Vulnerable	Not Vulnerable.
SWC-115	<a href="#">Authorization through tx.origin</a>	Not Vulnerable	<code>tx.origin</code> is not used anywhere in the code
SWC-116	<a href="#">Block values as a proxy for time</a>	Not Vulnerable	<code>Block.timestamp</code> is not used
SWC-117	<a href="#">Signature Malleability</a>	Not Vulnerable	Not used anywhere
SWC-118	<a href="#">Incorrect Constructor Name</a>	Not Vulnerable	All the constructors are created using the <code>constructor</code> keyword rather than functions.
SWC-119	<a href="#">Shadowing State Variables</a>	Not Vulnerable	Not applicable as this won't work during compile time after version <code>0.6.0</code>
SWC-120	<a href="#">Weak Sources of Randomness from Chain Attributes</a>	Not Vulnerable	Random generators are not used.
SWC-121	<a href="#">Missing Protection against Signature Replay Attacks</a>	Not Vulnerable	No such scenario was found

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

## 4. Remediation Status

Factiiv is actively partnering with CredShields from this engagement to validate the discovered vulnerabilities' remediations. **A retest was performed on April 5th, 2024, and all the issues have been addressed.**

Also, the table shows the remediation status of each finding.

VULNERABILITY TITLE	SEVERITY	REMEDICATION STATUS
Hardcoded Static Address	Informational	<b>Fixed</b> [12/11/2022]
Use of SafeMath	Gas	<b>Fixed</b> [12/11/2022]

*Table: Summary of findings and status of remediation*

## 5. Bug Reports

---

Bug ID #1 [Won't Fix]

### Outdated Pragma version

#### Vulnerability Type

Outdated Pragma

#### Severity

Low

#### Description

Using an outdated compiler version can be problematic, especially if there are publicly disclosed bugs and issues that affect the current compiler version.

The contracts found in the repository were allowing an old compiler version to be used, i.e., 0.8.19.

#### Affected Code

- <https://github.com/Factiiv-Projects/FACTTokens/blob/a4221c38d019dbb1ffa1aea3ade29979f5df9174/contracts/FACTToken.sol#L2>

#### Impacts

If the smart contract gets compiled and deployed with an older or too recent version of the solidity compiler, there's a chance that it may get compromised due to the bugs present in the older versions or unidentified exploits in the new versions.

Incompatibility issues may also arise if the contract code does not support features in other compiler versions, therefore, breaking the logic.

The likelihood of exploitation is really low therefore this is only Low severity.

#### Remediation



Keep the compiler versions updated in all the smart contract files. Do not allow floating pragmas anywhere. It is suggested to use the 0.8.24 pragma version which is stable and not too recent.

Reference: <https://swcregistry.io/docs/SWC-103>

### **Retest**

The current pragma version does cause any exploitation and hence it is safe to use in this case and CredShields team agrees to the decision.

## Bug ID #2 [Won't Fix]

### Bytes Constants More Efficient Than String Literal

#### Vulnerability Type

Gas Optimization

#### Severity

Gas

#### Description

The contract was found to be using `_NAME` string constant. This can be optimized by using `bytes32` constant to save gas.

#### Affected Code

- <https://github.com/Factiiv-Projects/FACTTokens/blob/a4221c38d019dbb1ffa1aea3ade29979f5df9174/contracts/FACTToken.sol#L12>
- <https://github.com/Factiiv-Projects/FACTTokens/blob/a4221c38d019dbb1ffa1aea3ade29979f5df9174/contracts/FACTToken.sol#L13>

#### Impacts

The use of inefficient string constants instead of gas-saving `bytes32` constants raises transaction costs, hindering contract usability and adoption.

#### Remediation

Unless explicitly required, if the string is lesser than 32 bytes, it is recommended to use `bytes32` constant instead of a string constant as it'll save some gas.

#### Retest

This bug is more of a best practice to reduce gas cost and won't cause any exploitation and hence the team decided to keep it as it is and CredShields team agrees to it.

## 6. Disclosure

---

The Reports provided by CredShields is not an endorsement or condemnation of any specific project or team and do not guarantee the security of any specific project. The contents of this report are not intended to be used to make decisions about buying or selling tokens, products, services, or any other assets and should not be interpreted as such.

Emerging technologies such as Smart Contracts and Solidity carry a high level of technical risk and uncertainty. CredShields does not provide any warranty or representation about the quality of code, the business model or the proprietors of any such business model, or the legal compliance of any business. The report is not intended to be used as investment advice and should not be relied upon as such.

CredShields Audit team is not responsible for any decisions or actions taken by any third party based on the report.