



CredShields

# Smart Contract Audit

March 20th, 2025 • CONFIDENTIAL

## Description

This document details the process and result of the Smart Contract audit performed by CredShields Technologies PTE. LTD. on behalf of ePound between March 18th, 2025, and March 19th, 2025.

## Author

Shashank (Co-founder, CredShields) [shashank@CredShields.com](mailto:shashank@CredShields.com)

## Reviewers

Aditya Dixit (Research Team Lead), Shreyas Koli(Auditor), Naman Jain (Auditor), Sanket Salavi (Auditor), Yash Shah (Auditor)

## Prepared for

ePound

# Table of Contents

Table of Contents	2
1. Executive Summary -----	3
State of Security	4
2. The Methodology -----	5
2.1 Preparation Phase	5
2.1.1 Scope	5
2.1.2 Documentation	5
2.1.3 Audit Goals	6
2.2 Retesting Phase	6
2.3 Vulnerability classification and severity	6
2.4 CredShields staff	8
3. Findings Summary -----	9
3.1 Findings Overview	9
3.1.1 Vulnerability Summary	9
3.1.2 Findings Summary	10
4. Remediation Status -----	13
5. Bug Reports -----	14
Bug ID #1	14
Use Ownable2Step	14
Bug ID #2	15
Outdated Pragma	15
Bug ID #3	16
Large Number Literals	16
6. The Disclosure -----	17

# 1. Executive Summary -----

ePound engaged CredShields to perform a smart contract audit from March 18th, 2025, to March 19th, 2025. During this timeframe, 3 vulnerabilities were identified.

During the audit, 0 vulnerabilities were found with a severity rating of either High or Critical. These vulnerabilities represent the greatest immediate risk to "ePound" 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
ePound Smart Contract	0	0	0	2	0	1	<b>3</b>
	<b>0</b>	<b>0</b>	<b>0</b>	<b>2</b>	<b>0</b>	<b>1</b>	<b>3</b>

*Table: Vulnerabilities Per Asset in Scope*

The CredShields team conducted the security audit to focus on identifying vulnerabilities in the ePound Smart Contract's scope during the testing window while abiding by the policies set forth by the ePound 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 ePound'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 ePound can implement both manual and automated procedures to eliminate entire classes of vulnerabilities in the future. By taking a proactive approach, ePound can future-proof its security posture and protect its assets.

## 2. The Methodology -----

ePound engaged CredShields to perform the ePound 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 March 18th, 2025, to March 19th, 2025, 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://bscscan.com/address/0x42a53b1fd6bd9279c6e5d88d53361bb524591472#code">https://bscscan.com/address/0x42a53b1fd6bd9279c6e5d88d53361bb524591472#code</a>

#### 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

ePound 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, and 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	● None	● 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 about the engagement or this document.



## 3. Findings Summary -----

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, 3 security vulnerabilities were identified in the asset.

VULNERABILITY TITLE	SEVERITY	SWC   Vulnerability Type
Use Ownable2Step	Low	Missing Best Practices
Outdated Pragma	Low	Outdated Compiler Version
Large Number Literals	Gas	Gas Optimization

*Table: Findings in 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>	Vulnerable	Bug ID #2
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 <b>block.blockhash()</b> , <b>msg.gas</b> , <b>throw</b> , <b>sha3()</b> , <b>callcode()</b> , <b>suicide()</b> 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	<code>Jump</code> 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	<code>abi.encodePacked()</code> 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 -----

ePound is actively partnering with CredShields from this engagement to validate the remediation of the discovered vulnerabilities. The table shows the remediation status of each finding.

VULNERABILITY TITLE	SEVERITY	REMEDIATION STATUS
Use Ownable2Step	Low	<b>Won't Fix</b> [ March 19th, 2025 ]
Outdated Pragma	Low	<b>Won't Fix</b> [ March 19th, 2025 ]
Large Number Literals	Gas	<b>Won't Fix</b> [ March 19th, 2025 ]

*Table: Summary of findings and status of remediation*

## 5. Bug Reports -----

Bug ID #1

### Use Ownable2Step

#### Vulnerability Type

Missing Best Practices

#### Severity

Low

#### Description

The "Ownable2Step" pattern is an improvement over the traditional "Ownable" pattern, designed to enhance the security of ownership transfer functionality in a smart contract. Unlike the original "Ownable" pattern, where ownership can be transferred directly to a specified address, the "Ownable2Step" pattern introduces an additional step in the ownership transfer process. Ownership transfer only completes when the proposed new owner explicitly accepts the ownership, mitigating the risk of accidental or unintended ownership transfers to mistyped addresses.

#### Affected Code

- <https://bscscan.com/address/0x42a53b1fd6bd9279c6e5d88d53361bb524591472#code#L537>

#### Impacts

Without the "Ownable2Step" pattern, the contract owner might inadvertently transfer ownership to an unintended or mistyped address, potentially leading to a loss of control over the contract. By adopting the "Ownable2Step" pattern, the smart contract becomes more resilient against external attacks aimed at seizing ownership or manipulating the contract's behavior.

#### Remediation

It is recommended to use either Ownable2Step or Ownable2StepUpgradeable depending on the smart contract.

#### Retest:

-

Bug ID #2

## Outdated Pragma

### Vulnerability Type

Outdated Compiler Version ([SWC-102](#))

### Severity

Low

### Description

The smart contract is using an outdated version of the Solidity compiler specified by the pragma directive i.e. 0.8.24. Solidity is actively developed, and new versions frequently include important security patches, bug fixes, and performance improvements. Using an outdated version exposes the contract to known vulnerabilities that have been addressed in later releases. Additionally, newer versions of Solidity often introduce new language features and optimizations that improve the overall security and efficiency of smart contracts.

### Affected Code

- <https://bscscan.com/address/0x42a53b1fd6bd9279c6e5d88d53361bb524591472#code#L5>

### Impacts

The use of an outdated Solidity compiler version can have significant negative impacts. Security vulnerabilities that have been identified and patched in newer versions remain exploitable in the deployed contract.

Furthermore, missing out on performance improvements and new language features can result in inefficient code execution and higher gas costs.

### Remediation

It is suggested to use the 0.8.28 pragma version.

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

### Retest

-

Bug ID #3

## Large Number Literals

### Vulnerability Type

Gas Optimization

### Severity

Gas

### Description

Solidity supports multiple rational and integer literals, including decimal fractions and scientific notations. The use of very large numbers with too many digits was detected in the code that could have been optimized using a different notation also supported by Solidity.

### Affected Code

- <https://bscscan.com/address/0x42a53b1fd6bd9279c6e5d88d53361bb524591472#code#L540>

### Impacts

Having a large number literals in the code increases the gas usage of the contract during its deployment and when the functions are used or called from the contract.

It also makes the code harder to read and audit and increases the chances of introducing code errors.

### Remediation

Scientific notation in the form of  $2e10$  is also supported, where the mantissa can be fractional, but the exponent has to be an integer. The literal  $MeE$  is equivalent to  $M * 10^{**E}$ . Examples include  $2e10$ ,  $2e10$ ,  $2e-10$ ,  $2.5e1$ , as suggested in official solidity documentation. <https://docs.soliditylang.org/en/latest/types.html#rational-and-integer-literals>

It is recommended to use numbers in the form  $"35 * 1e7 * 1e18"$  or  $"35 * 1e25"$ .

The numbers can also be represented by using underscores between them to make them more readable such as  $"35\_00\_00\_000"$

### Retest

-



## 6. The Disclosure -----

The Reports provided by CredShields are 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.

# YOUR **SECURE FUTURE** STARTS HERE



At CredShields, we're more than just auditors. We're your strategic partner in ensuring a secure Web3 future. Our commitment to your success extends beyond the report, offering ongoing support and guidance to protect your digital assets

Q Audited by

