Audited by

CRED SHIELDS

CredShields

Smart Contract Audit

November 12th, 2024

Description

This document details the process and result of the Smart Contract audit performed by CredShields Technologies PTE. LTD. on behalf of HoldPlatform between Oct 20th, 2024, and Oct 24th, 2024. A retest was performed on Nov 7th, 2024.

Author
Shashank (Co-founder, CredShields) shashank@CredShields.com

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

Prepared for
HoldPlatform

Table of Contents

1. Executive Summary --------------------

HoldPlatform engaged CredShields to perform a smart contract audit from Oct 20th, 2024, to Oct 24th, 2024. During this timeframe, Seven (7) vulnerabilities were identified. **A retest was performed on Nov 7th, 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 "HoldPlatform" 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
HoldPlatform V2 Smart Contracts	0	0	3	1	1	2	**7**
	0	**0**	**1**	**1**	**1**	**2**	**7**

Table: Vulnerabilities Per Asset in Scope

The CredShields team conducted the security audit to focus on identifying vulnerabilities in HoldPlatform V2 Smart Contract's scope during the testing window while abiding by the policies set forth by HoldPlatform'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 HoldPlatform'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 HoldPlatform can implement both manual and automated procedures to eliminate entire classes of vulnerabilities in the future. By taking a proactive approach, HoldPlatform can future-proof its security posture and protect its assets.

2. The Methodology --------------------

HoldPlatform engaged CredShields to perform a HoldPlatform V2 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 Oct 20th, 2024, to Oct 24th, 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
https://optimistic.etherscan.io/address/0x26c50AF4725a77172A60439EC1957252F2Fe4094#code%23F1%23L861

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

HoldPlatform 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

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, Seven (7) security vulnerabilities were identified in the asset.

VULNERABILITY TITLE	SEVERITY	SWC | Vulnerability Type
Missing Handling of Fees on Transfer in ERC20 Token Transfers	Medium	Calculation Inaccuracy
Inadequate Handling of Non-Reverting ERC20 Transfers	Medium	Token Interaction
Missing Validation for Token Leads to User Manipulating Contract’s TVL	Medium	Price Manipulation
Missing Events in Important Functions	Low	Missing Best Practices
Require with Empty Message	Informational	Code optimization
Cheaper Inequalities in require()	Gas	Gas Optimization
Cheaper Conditional Operators	Gas	Gas Optimization

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	Not 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 block.blockhash(), msg.gas, throw, sha3(), callcode(), suicide() 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	tx.origin is not used anywhere in the code
SWC-116	Block values as a proxy for time	Not Vulnerable	Block.timestamp is not used
SWC-117	Signature Malleability	Not Vulnerable	Not used anywhere
SWC-118	Incorrect Constructor Name	Not Vulnerable	All the constructors are created using the constructor keyword rather than functions.
SWC-119	Shadowing State Variables	Not Vulnerable	Not applicable as this won't work during compile time after version 0.6.0
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

4. Remediation Status -----------------

HoldPlatform is actively partnering with CredShields from this engagement to validate the discovered vulnerabilities' remediations. **A retest was performed on Nov 7th, 2024, and all the issues have been addressed.**

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

VULNERABILITY TITLE	SEVERITY	REMEDIATION STATUS
Missing Handling of Fees on Transfer in ERC20 Token Transfers	Medium	**Won't Fix** [07/11/2024]
Inadequate Handling of Non-Reverting ERC20 Transfers	Medium	**Partially Fix** [07/11/2024]
Missing Validation for Token Leads to User Manipulating Contract's TVL	Medium	**Won't Fix** [07/11/2024]
Missing Events in Important Functions	Low	**Won't Fix** [07/11/2024]
Require with Empty Message	Informational	**Won't Fix** [07/11/2024]
Cheaper Inequalities in require()	Gas	**Fixed** [07/11/2024]

Table: Summary of findings and status of remediation

5. Bug Reports ------------------------

Bug ID #1 [Won't Fix]

Missing Handling of Fees on Transfer in ERC20 Token Transfers

Vulnerability Type
Calculation Inaccuracy

Severity
Medium

Description
The contract contains a vulnerability related to transferring ERC20 tokens without considering the possibility of fees charged on transfer. Some ERC20 tokens implement a fee mechanism, where a certain percentage of tokens is deducted as a fee during each transfer. However, the contract does not account for this possibility when transferring tokens using the safeTransferFrom/TransferFrom function.

Affected Code

- https://optimistic.etherscan.io/address/0x26c50AF4725a77172A60439EC1957252F2Fe4094#code#F1#L1024

Impacts
Failure to account for transfer fees can lead to accounting errors and financial losses. When tokens with transfer fees are transferred, the actual received amount may be less than expected due to the deduction of fees. As a result, the contract's internal accounting and balance tracking may become inaccurate, leading to discrepancies in token balances and potential financial losses for users.

Remediation
To address this vulnerability it is recommended to add a mechanism to calculate the balance of the contract before and after the transfer is completed.

Retest

Client's comment: Parameter no. 27 contains the token sync, which reflects the real token balance within the smart contract. The real token balance must be greater than or equal to the token balance.

Credshields Comment: Not every token charges a fee on transfer, the parameter no. 27 syncs the token that is received from the user (which is after deducting the fees), and parameter no. 25,26 calculates the token amount from the user provided token (which is before deducting the fees).
Reference: weird-erc20

Bug ID #2 [Partially Fixed]

Inadequate Handling of Non-Reverting ERC20 Transfers

Vulnerability Type
Token Interaction

Severity
Medium

Description
The functions in the contract make calls to the transfer() functions of ERC20 tokens without verifying the return value. Some ERC20 tokens, such as ZRX and EURS, do not revert on transfer failures but instead return false. This behaviour, while technically compliant with the ERC20 standard, is not handled correctly in the contract. If a transfer fails and returns false, the contract will proceed as if the transfer succeeded.

Affected Code

- https://optimistic.etherscan.io/address/0x26c50AF4725a77172A60439EC1957252F2Fe4094#code#F1#L1500
- https://optimistic.etherscan.io/address/0x26c50AF4725a77172A60439EC1957252F2Fe4094#code#F1#L1634
- https://optimistic.etherscan.io/address/0x26c50AF4725a77172A60439EC1957252F2Fe4094#code#F1#L1689
- https://optimistic.etherscan.io/address/0x26c50AF4725a77172A60439EC1957252F2Fe4094#code#F1#L1701
- https://optimistic.etherscan.io/address/0x26c50AF4725a77172A60439EC1957252F2Fe4094#code#F1#L1709

Impacts
Users attempting to withdraw tokens might not actually transferred to their, resulting in incorrect balances and potential loss of funds..

Remediation
Update the functions to check the return value of the transfer calls and revert the transaction if the return value is false. This can be done by wrapping the calls in a require statement. Or using safeTransferFrom()/safeTransfer() from OpenZeppelin Library.

Retest

If importing the OpenZeppelin library causes an error, then implement a mechanism to check the return value of the transfer call.
Eg.

```
(bool success, ) = token.transfer(s.user, s.payment_queue);
 require (success, “Transfer failed”);
```

Bug ID #3 [Won’t Fix]

Missing validation for token leads to user manipulate contract’s TVL

Vulnerability Type
Price Manipulation

Severity
Medium

Description
The smart contract allows users to input the TokenPrice during the Holdplatform() function execution. This user-provided value is used to calculate s.usdvalue_deposit and subsequently impacts the _syncTVL() function, which updates the totalusdvalue. Since the TokenPrice is not verified against a reliable price oracle, users can manipulate this value to inaccurately reflect the USD value of their deposits. This manipulation can lead to incorrect calculations of the total value locked (TVL).

Affected Code

- https://optimistic.etherscan.io/address/0x26c50AF4725a77172A60439EC1957252F2Fe4094#code#F1#L976
- https://optimistic.etherscan.io/address/0x26c50AF4725a77172A60439EC1957252F2Fe4094#code#F1#L1376

Impacts
The manipulation of TokenPrice can result in an inaccurate representation of the contract's financial status, potentially misleading stakeholders and users about the platform's value.

Remediation
Implement a mechanism to fetch and verify the TokenPrice from a trusted and decentralized price oracle. This ensures that the price used in calculations reflects the true market value.

Retest
If the deployer can fetch the price directly from the etherscan and avoid taking such inputs from users.

Bug ID #4 [Won’t Fix]

Missing Events in Important Functions

Vulnerability Type
Missing Best Practices

Severity
Low

Description
Events are inheritable members of contracts. When you call them, they cause the arguments to be stored in the transaction’s log—a special data structure in the blockchain. These logs are associated with the address of the contract which can then be used by developers and auditors to keep track of the transactions.

The contract was found to be missing these events on certain critical functions which would make it difficult or impossible to track these transactions off-chain.

Affected Code
The following functions were affected -

- https://optimistic.etherscan.io/address/0x26c50AF4725a77172A60439EC1957252F2Fe4094#code#F1#L877-#L904
- https://optimistic.etherscan.io/address/0x26c50AF4725a77172A60439EC1957252F2Fe4094#code#F1#L1250-#1299
- https://optimistic.etherscan.io/address/0x26c50AF4725a77172A60439EC1957252F2Fe4094#code#F1#L1305-#1358
- https://optimistic.etherscan.io/address/0x26c50AF4725a77172A60439EC1957252F2Fe4094#code#F1#L1562-#1605
- https://optimistic.etherscan.io/address/0x26c50AF4725a77172A60439EC1957252F2Fe4094#code#F1#L1748-#1793
- https://optimistic.etherscan.io/address/0x26c50AF4725a77172A60439EC1957252F2Fe4094#code#F1#L1802-#1819
- https://optimistic.etherscan.io/address/0x26c50AF4725a77172A60439EC1957252F2Fe4094#code#F1#L1867-#1878

Impacts

Events are used to track the transactions off-chain and missing these events on critical functions makes it difficult to audit these logs if they're needed at a later stage.

Remediation
Consider emitting events for important functions to keep track of them.

Retest
Client Comments : I actually wanted to add events to each function but due to the 24,576-byte limit, it can't

Bug ID #5 [Won't Fix]

Require with Empty Message

Vulnerability Type
Code optimization

Severity
Informational

Description
During analysis; multiple require statements were detected with empty messages. The statement takes two parameters, and the message part is optional. This is shown to the user when and if the require statement evaluates to false. This message gives more information about the conditional and why it gave a false response.

Affected Code

- https://optimistic.etherscan.io/address/0x26c50AF4725a77172A60439EC1957252F2Fe4094#code#F1#L861
- https://optimistic.etherscan.io/address/0x26c50AF4725a77172A60439EC1957252F2Fe4094#code#F1#L920
- https://optimistic.etherscan.io/address/0x26c50AF4725a77172A60439EC1957252F2Fe4094#code#F1#L987
- https://optimistic.etherscan.io/address/0x26c50AF4725a77172A60439EC1957252F2Fe4094#code#F1#L997
- https://optimistic.etherscan.io/address/0x26c50AF4725a77172A60439EC1957252F2Fe4094#code#F1#L1002
- https://optimistic.etherscan.io/address/0x26c50AF4725a77172A60439EC1957252F2Fe4094#code#F1#L1010
- https://optimistic.etherscan.io/address/0x26c50AF4725a77172A60439EC1957252F2Fe4094#code#F1#L1011
- https://optimistic.etherscan.io/address/0x26c50AF4725a77172A60439EC1957252F2Fe4094#code#F1#L1024
- https://optimistic.etherscan.io/address/0x26c50AF4725a77172A60439EC1957252F2Fe4094#code#F1#L1377
- https://optimistic.etherscan.io/address/0x26c50AF4725a77172A60439EC1957252F2Fe4094#code#F1#L1378

- https://optimistic.etherscan.io/address/0x26c50AF4725a77172A60439EC1957252F2Fe4094#code#F1#L1383
- https://optimistic.etherscan.io/address/0x26c50AF4725a77172A60439EC1957252F2Fe4094#code#F1#L1384
- https://optimistic.etherscan.io/address/0x26c50AF4725a77172A60439EC1957252F2Fe4094#code#F1#L1498
- https://optimistic.etherscan.io/address/0x26c50AF4725a77172A60439EC1957252F2Fe4094#code#F1#L1627
- https://optimistic.etherscan.io/address/0x26c50AF4725a77172A60439EC1957252F2Fe4094#code#F1#L1633
- https://optimistic.etherscan.io/address/0x26c50AF4725a77172A60439EC1957252F2Fe4094#code#F1#L1674
- https://optimistic.etherscan.io/address/0x26c50AF4725a77172A60439EC1957252F2Fe4094#code#F1#L1679
- https://optimistic.etherscan.io/address/0x26c50AF4725a77172A60439EC1957252F2Fe4094#code#F1#L1688
- https://optimistic.etherscan.io/address/0x26c50AF4725a77172A60439EC1957252F2Fe4094#code#F1#L1700
- https://optimistic.etherscan.io/address/0x26c50AF4725a77172A60439EC1957252F2Fe4094#code#F1#L1708
- https://optimistic.etherscan.io/address/0x26c50AF4725a77172A60439EC1957252F2Fe4094#code#F1#L1836
- https://optimistic.etherscan.io/address/0x26c50AF4725a77172A60439EC1957252F2Fe4094#code#F1#L1898
- https://optimistic.etherscan.io/address/0x26c50AF4725a77172A60439EC1957252F2Fe4094#code#F1#L1907
- https://optimistic.etherscan.io/address/0x26c50AF4725a77172A60439EC1957252F2Fe4094#code#F1#L1912
- https://optimistic.etherscan.io/address/0x26c50AF4725a77172A60439EC1957252F2Fe4094#code#F1#L1913
- https://optimistic.etherscan.io/address/0x26c50AF4725a77172A60439EC1957252F2Fe4094#code#F1#L1922
- https://optimistic.etherscan.io/address/0x26c50AF4725a77172A60439EC1957252F2Fe4094#code#F1#L2003
- https://optimistic.etherscan.io/address/0x26c50AF4725a77172A60439EC1957252F2Fe4094#code#F1#L2008
- https://optimistic.etherscan.io/address/0x26c50AF4725a77172A60439EC1957252F2Fe4094#code#F1#L2014
- https://optimistic.etherscan.io/address/0x26c50AF4725a77172A60439EC1957252F2Fe4094#code#F1#L2021

Impacts

Having a short descriptive message in the require statement gives users and developers more details as to why the conditional statement failed and helps in debugging the transactions.

Remediation

It is recommended to add a descriptive message, no longer than 32 bytes, inside the require statement to give more detail to the user about why the condition failed.

Retest

Client's comment: I actually wanted to add notes to each empty message, but due to the 24,576-byte limit, I am currently updating them as additional notes only.

Bug ID #6 [Won't Fix]

Cheaper Inequalities in require()

Vulnerability Type

Gas Optimization

Severity

Gas

Description

The contract was found to be performing comparisons using inequalities inside the require statement. When inside the require statements, non-strict inequalities (>=, <=) are usually costlier than strict equalities (>, <).

Affected Code

- https://optimistic.etherscan.io/address/0x26c50AF4725a77172A60439EC1957252F2Fe4094#code#F1#L2013-#2017
- https://optimistic.etherscan.io/address/0x26c50AF4725a77172A60439EC1957252F2Fe4094#code#F1#L2019-#2022

Impacts

Using non-strict inequalities inside "require" statements costs more gas.

Remediation

It is recommended to go through the code logic, and, **if possible**, modify the non-strict inequalities with the strict ones to save gas as long as the logic of the code is not affected.

Retest

Client's comment: I have updated it according to your recommendations.
Smart Contract: 0xaAC85d8cCe339D60285fe58c896180b5b24BA1d4 NOT WORK

Bug ID #7 [Fixed]

Cheaper Conditional Operators

Vulnerability Type
Gas Optimization

Severity
Gas

Description
Upon reviewing the code, it has been observed that the contract uses conditional statements involving comparisons with unsigned integer variables. Specifically, the contract employs the conditional operators x != 0 and x > 0 interchangeably. However, it's important to note that during compilation, x != 0 is generally more cost-effective than x > 0 for unsigned integers within conditional statements.

Affected Code

- https://optimistic.etherscan.io/address/0x26c50AF4725a77172A60439EC1957252F2Fe4094#code#F1#L659
- https://optimistic.etherscan.io/address/0x26c50AF4725a77172A60439EC1957252F2Fe4094#code#F1#L800

Impacts
Employing x != 0 in conditional statements can result in reduced gas consumption compared to using x > 0. This optimization contributes to cost-effectiveness in contract interactions.

Remediation
Whenever possible, use the x != 0 conditional operator instead of x > 0 for unsigned integer variables in conditional statements.

Retest
This issue has been fixed as recommended.

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

CRED SHIELDS

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

Audited by CRED SHIELDS