CredShields

Smart Contract Audit

Oct 16th, 2023 • CONFIDENTIAL

Description

This document details the process and result of the Bridge Contracts audit performed by CredShields Technologies PTE. LTD. on behalf of Archethic Foundation between Oct 4th, 2023, and Oct 10th, 2023. A retest was performed on Oct 12th, 2023.

Author

Shashank (Co-founder, CredShields)

shashank@CredShields.com

Reviewers

Aditya Dixit (Research Team Lead)

aditya@CredShields.com

Prepared for

Archethic Foundation

Table of Contents

CRED SHiELDS

1. Executive Summary

Archethic Foundation engaged CredShields to perform a smart contract audit from Oct 4th, 2023, to Oct 10th, 2023. During this timeframe, Ten (10) vulnerabilities were identified. **A retest was performed on Oct 12th, 2023, and all the bugs have been addressed.**

During the audit, One (1) vulnerability was found with a severity rating of either High or Critical. These vulnerabilities represent the greatest immediate risk to "Archethic Foundation" and should be prioritized for remediation.

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
Bridge Contracts	0	1	0	3	3	3	**10**
	0	**1**	**0**	**3**	**3**	**3**	**10**

Table: Vulnerabilities Per Asset in Scope

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

2. Methodology

Archethic Foundation engaged CredShields to perform an Archethic Foundation 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 4th, 2023, to Oct 10th, 2023, 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://github.com/archethic-foundation/bridge-contracts/tree/0b7543e643a568c97a8befc50e2a5c4424c421d4/evm/contracts

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

Archethic Foundation 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

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, Ten (10) security vulnerabilities were identified in the asset.

VULNERABILITY TITLE	SEVERITY	SWC | Vulnerability Type
Strict Equality in _enoughFunds can cause funds to get stuck in Contracts	High	Improper Validation
Use safeTransfer instead of transfer	Low	Missing best practices
Floating and Outdated Pragma	Low	Floating Pragma (SWC-103)
Use Ownable2Step	Low	Missing Best Practices
Missing NatSpec Comments	Informational	Missing best practices
Missing State Variable Visibility	Informational	Missing Best Practices

Require With Empty Message	Informational	Missing Best Practices
Variables should be Immutable	Gas	Gas Optimization
Internal Function Never Used	Gas	Gas Optimization
Use of SafeMath	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

Archethic Foundation is actively partnering with CredShields from this engagement to validate the discovered vulnerabilities' remediations. **A retest was performed on Oct 12th, 2023, and all the issues have been addressed.**

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

VULNERABILITY TITLE	SEVERITY	REMEDIATION STATUS
Strict Equality in _enoughFunds can cause funds to get stuck in Contracts	High	**Fixed [12/10/2023]**
Use safeTransfer instead of transfer	Low	**Fixed [12/10/2023]**
Floating and Outdated Pragma	Low	**Fixed [12/10/2023]**
Use Ownable2Step	Low	**Fixed [12/10/2023]**
Missing NatSpec Comments	Informational	**Fixed [12/10/2023]**
Missing State Variable Visibility	Informational	**Fixed [12/10/2023]**
Require With Empty Message	Informational	**Fixed [12/10/2023]**
Variables should be Immutable	Gas	**Fixed**

		[12/10/2023]
Internal Function Never Used	Gas	**Fixed [12/10/2023]**
Use of SafeMath	Gas	**Fixed [12/10/2023]**

Table: Summary of findings and status of remediation

5. Bug Reports

Bug ID#1 [Fixed]

Strict Equality in _enoughFunds can cause funds to get stuck in Contracts

Vulnerability Type
Improper Validation

Severity
High

Description
In the withdraw & refund function of the contract, there is a strict balance equality check using address(this).balance == amount. This check ensures that the contract's balance must be exactly equal to the amount for a successful withdrawal & refund. However, this strict check can lead to unexpected behavior if someone sends an extra amount to the contract, resulting in a failed withdrawal.
The issue arises from the strict equality check (==) without considering a scenario where the contract may receive additional funds, causing the balance to be greater than the expected withdrawal amount.

Affected Code

- https://github.com/archethic-foundation/bridge-contracts/blob/0b7543e643a568c97a8befc50e2a5c4424c421d4/evm/contracts/HTLC/HTLCBase.sol#L50-L67
- https://github.com/archethic-foundation/bridge-contracts/blob/0b7543e643a568c97a8befc50e2a5c4424c421d4/evm/contracts/HTLC/HTLCBase.sol#L73-L86
- https://github.com/archethic-foundation/bridge-contracts/blob/0b7543e643a568c97a8befc50e2a5c4424c421d4/evm/contracts/HTLC/HTLC_ETH.sol#L37-L39

Impacts

If someone accidentally or intentionally sends more funds to the contract than the amount expected for a withdrawal & refund, the contract will not permit refunds. leaving the funds stuck in the contract.

Remediation

To address this issue and allow for flexibility in handling excess funds, it is recommended to implement a partial balance check in the _enoughFunds().

Retest

This is fixed. The strict validations have been updated to “address(this).balance >= amount”

Bug ID#2 [Fixed]

Use safeTransfer instead of transfer

Vulnerability Type
Missing best practices

Severity
Low

Description
The transfer() method is used instead of safeTransfer(), presumably to save gas however OpenZeppelin's documentation discourages the use of transfer(), use safeTransfer() whenever possible because safeTransfer auto-handles boolean return values whenever there's an error.

Affected Code

- https://github.com/archethic-foundation/bridge-contracts/blob/0b7543e643a568c97a8befc50e2a5c4424c421d4/evm/contracts/HTLC/ChargeableHTLC_ERC.sol#L32-L41
- https://github.com/archethic-foundation/bridge-contracts/blob/0b7543e643a568c97a8befc50e2a5c4424c421d4/evm/contracts/HTLC/HTLC_ERC.sol#L14-L20
- https://github.com/archethic-foundation/bridge-contracts/blob/0b7543e643a568c97a8befc50e2a5c4424c421d4/evm/contracts/Pool/ERCPool.sol#L29-L38

Impacts
Using safeTransfer has the following benefits -

- It checks the boolean return values of ERC20 operations and reverts the transaction if they fail,
- at the same time allowing you to support some non-standard ERC20 tokens that don't have boolean return values.
- It additionally provides helpers to increase or decrease an allowance, to mitigate an attack possible with vanilla approve.

Remediation

Consider using safeTransfer() instead of transfer(). Also, add a nonReentrant modifier to prevent reentrancy attacks and unintentional results.

Retest

The contracts are now using safeTransfer instead of transfer.

Bug ID #3 [Fixed]

Floating and Outdated Pragma

Vulnerability Type

Floating Pragma (SWC-103)

Severity

Low

Description

Locking the pragma helps ensure that the contracts do not accidentally get deployed using an older version of the Solidity compiler affected by vulnerabilities.

The contract allowed floating or unlocked pragma to be used, i.e.,^0.8.13. This allows the contracts to be compiled with all the solidity compiler versions above the limit specified. The following contracts were found to be affected -

Affected Code

- All Solidity Files

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 informational.

Remediation

Keep the compiler versions consistent in all the smart contract files. Do not allow floating pragmas anywhere. It is suggested to use the 0.8.21 pragma version

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

Retest

The pragma version has been fixed and updated to 0.8.21.

Bug ID #4 [Fixed]

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://github.com/archethic-foundation/bridge-contracts/blob/0b7543e643a568c97a8befc50e2a5c4424c421d4/evm/contracts/Pool/PoolBase.sol#L6
- https://github.com/archethic-foundation/bridge-contracts/blob/0b7543e643a568c97a8befc50e2a5c4424c421d4/evm/contracts/Migrations.sol#L4

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:

This is fixed. The contracts are now using Ownable2StepUpgradeable and OwnableUpgradeable.

Bug ID#5 [Fixed]

Missing NatSpec Comments

Vulnerability Type
Missing best practices

Severity
Informational

Description
Solidity contracts use a special form of comments to document code. This special form is named the Ethereum Natural Language Specification Format (NatSpec).
The document is divided into descriptions for developers and end-users along with the title and the author.
The contracts in the scope were missing these comments.

Impacts
Without Natspec comments, it can be challenging for other developers to understand the code's intended behavior and purpose. This can lead to errors or bugs in the code, making it difficult to maintain and update the codebase. Additionally, it can make it harder for auditors to evaluate the code for security vulnerabilities, increasing the risk of potential exploits.

Remediation
Developers should review their codebase and add Natspec comments to all relevant functions, variables, and events. Natspec comments should include a description of the function or event, its parameters, and its return values.

Retest:
NatSpec comments have been added to the code.

Bug ID #6 [Fixed]

Missing State Variable Visibility

Vulnerability Type
Missing Best Practices

Severity
Informational

Description
In Solidity, the visibility of state variables is important as it determines how those variables can be accessed and modified by other contracts or functions.
The contract defined state variables that were missing a visibility modifier.

Affected Code

- https://github.com/archethic-foundation/bridge-contracts/blob/0b7543e643a568c97a8befc50e2a5c4424c421d4/evm/contracts/Pool/PoolBase.sol#L23-L27

Impacts
If the visibility of a state variable is accidentally left out, it can cause unexpected behavior and security vulnerabilities. For example, if a state variable is supposed to be private and is accidentally declared without any visibility keyword, it will be treated as "internal" by default, which may lead to it being accessible by other contracts or functions outside the intended scope. This can lead to a potential attack vector for malicious actors.

Remediation
Explicitly define visibility for all state variables. These variables can be specified as public, internal, or private.

Retest
The state variables have been updated with appropriate visibility.

Bug ID #7 [Fixed]

Require With Empty Message

Vulnerability Type

Missing Best Practices

Severity

informational

Description

During code analysis, it has been observed that some require statements lack descriptive messages, which provide crucial information to users when conditions are not met. These messages, limited to 32 bytes, improve user understanding of why a transaction was reverted.

Vulnerable Code

- https://github.com/archethic-foundation/bridge-contracts/blob/0b7543e643a568c97a8befc50e2a5c4424c421d4/evm/contracts/HTLC/ChargeableHTLC_ETH.sol#L31-L36
- https://github.com/archethic-foundation/bridge-contracts/blob/0b7543e643a568c97a8befc50e2a5c4424c421d4/evm/contracts/HTLC/ChargeableHTLC_ETH.sol#L40
- https://github.com/archethic-foundation/bridge-contracts/blob/0b7543e643a568c97a8befc50e2a5c4424c421d4/evm/contracts/HTLC/HTLC_ETH.sol#L29-L30
- https://github.com/archethic-foundation/bridge-contracts/blob/0b7543e643a568c97a8befc50e2a5c4424c421d4/evm/contracts/HTLC/HTLC_ETH.sol#L34-L35

Impacts

Users may be left without clear context when a transaction is reverted due to unmet conditions, leading to confusion and frustration

Remediation

Add concise, informative messages to require statements, explaining why the condition failed. Ensure messages are clear and within the 32-byte limit.

Retest

The require statements are now defined with appropriate messages.

Bug ID #8 [Fixed]

Variables should be Immutable

Vulnerability Type
Gas Optimization

Severity
Gas

Description:
Declaring state variables that are not updated following deployment as immutable can save gas costs in smart contract deployments and function executions. Immutable state variables are those that cannot be changed once they are initialized, and their values are set permanently.

By declaring state variables as immutable, the compiler can optimize their storage in a way that reduces gas costs. Specifically, the compiler can store the value directly in the bytecode of the contract, rather than in storage, which is a more expensive operation.

Affected Code:

- https://github.com/archethic-foundation/bridge-contracts/blob/0b7543e643a568c97a8befc50e2a5c4424c421d4/evm/contracts/HTLC/SignedHTLC_ERC.sol#L13
- https://github.com/archethic-foundation/bridge-contracts/blob/0b7543e643a568c97a8befc50e2a5c4424c421d4/evm/contracts/HTLC/SignedHTLC_ETH.sol#L13

Impacts:
Gas usage is increased if the variables that are not updated outside of the constructor are not set as immutable.

Remediation:
An "`immutable`" attribute should be added in the parameters that are never updated outside of the constructor to save the gas.

Retest

The variables that are not modified anywhere in the code except the constructor have been updated as immutable.

Bug ID #9 [Fixed]

Internal Function Never Used

Vulnerability Type
Gas Optimization

Severity
Gas

Description:
In the provided smart contract, there are internal functions declared but not used within the contract's functions or by any other contracts. These unused internal functions consume gas when deploying and executing the contract. Additionally, having unused functions can potentially confuse auditors and developers trying to understand the contract's logic.

Affected Code:

- https://github.com/archethic-foundation/bridge-contracts/blob/0b7543e643a568c97a8befc50e2a5c4424c421d4/evm/contracts/HTLC/HTLC_ETH.sol#L23-L25

Impacts:
The impact of having unused internal functions in the contract is relatively low. It primarily affects the gas consumption during contract deployment and execution. It doesn't directly lead to vulnerabilities or security risks, but it can make the contract's code less readable and efficient.

Remediation:
To improve code quality and reduce unnecessary gas consumption, it's recommended to remove any internal functions that are not used within the contract or by any other contracts in the system

Retest

The unused function is removed from the contract.

Bug ID #10 [Fixed]

Use of SafeMath

Vulnerability Type
Gas Optimization

Severity
Gas

Description:
SafeMath library is found to be used in the contract. This increases gas consumption more than traditional methods and validations if done manually.
Also, Solidity **0.8.0** and above includes checked arithmetic operations by default, and this renders SafeMath unnecessary.

Affected Code:

- https://github.com/archethic-foundation/bridge-contracts/blob/0b7543e643a568c97a8befc50e2a5c4424c421d4/evm/contracts/HTLC/ChargeableHTLC_ERC.sol#L5
- https://github.com/archethic-foundation/bridge-contracts/blob/0b7543e643a568c97a8befc50e2a5c4424c421d4/evm/contracts/HTLC/ChargeableHTLC_ETH.sol#L4
- https://github.com/archethic-foundation/bridge-contracts/blob/0b7543e643a568c97a8befc50e2a5c4424c421d4/evm/contracts/HTLC/SignedHTLC_ERC.sol#L6
- https://github.com/archethic-foundation/bridge-contracts/blob/0b7543e643a568c97a8befc50e2a5c4424c421d4/evm/contracts/HTLC/SignedHTLC_ETH.sol#L6
- https://github.com/archethic-foundation/bridge-contracts/blob/0b7543e643a568c97a8befc50e2a5c4424c421d4/evm/contracts/Pool/PoolBase.sol#L7

Impacts:
This increases the gas usage of the contract.

Remediation:

We do not recommend using the SafeMath library for all arithmetic operations. It is good practice to use explicit checks where it is really needed and to avoid extra checks where overflow/underflow is impossible.
It is recommended to upgrade to the latest compiler because versions above 0.8.0+ automatically check for overflows and underflows.

Retest:
Safemath has been removed from the contracts. This is fixed.

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.