Audited by CRED SHIELDS

CredShields

Smart Contract Audit

April 15th, 2024 • CONFIDENTIAL

Description

This document details the process and result of the Smart Contract audit performed by CredShields Technologies PTE. LTD. on behalf of Tribally Games between July 16th, 2024, and July 18th, 2024. A retest was performed on August 5th, 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

Tribally Games

Table of Contents

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

Tribally Games engaged CredShields to perform a smart contract audit from July 16th, 2024, to July 18th, 2024. During this timeframe, 6 vulnerabilities were identified. **A retest was performed on August 5th, 2024, and all the bugs have been addressed.**

During the audit, 1 vulnerabilities were found with a severity rating of either High or Critical. These vulnerabilities represent the greatest immediate risk to "Tribally Games" 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
Tribaly Games	0	1	0	4	0	1	**6**
	0	**1**	**0**	**4**	**0**	**1**	**6**

Table: Vulnerabilities Per Asset in Scope

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

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

Tribally Games engaged CredShields to perform a Tribally Games 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 July 16th, 2024, to July 18th, 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://github.com/Tribally-Games/contracts/tree/73ed6a9f1f17d958d6f6dfdf8458377b5d8c925c

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

Tribally Games 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, 6 security vulnerabilities were identified in the asset.

VULNERABILITY TITLE	SEVERITY	SWC | Vulnerability Type
Cross-Chain Signature Replay Attack	High	Cross-Chain Signature Replay
Missing Zero Address Validations	Low	Missing Input Validation
Use safeTransfer/safeTransferFrom instead of transfer/transferFrom	Low	Missing Best Practices
Floating and Outdated Pragma	Low	Floating Pragma (SWC-103)
Missing Events in Important Functions	Low	Missing Best Practices
Cheaper Inequalities in if()	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	Bug ID #4
SWC-103	Floating Pragma	Not Vulnerable	Bug ID #4
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	Bug ID #1
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 ----------

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

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

VULNERABILITY TITLE	SEVERITY	REMEDIATION STATUS
Cross-Chain Signature Replay Attack	High	**Fixed** [Aug 5, 2024]
Missing Zero Address Validations	Low	**Fixed** [Aug 5, 2024]
Use safeTransfer/safeTransferFrom instead of transfer/transferFrom	Low	**Fixed** [Aug 5, 2024]
Floating and Outdated Pragma	Low	**Partially Fixed** [Aug 5, 2024]
Missing Events in Important Functions	Low	**Fixed** [Aug 5, 2024]
Cheaper Inequalities in if()	Gas	**Not Fixed** [Aug 5, 2024]

Table: Summary of findings and status of remediation

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

Bug ID # 1 [Fixed]

Cross-Chain Signature Replay Attack

Vulnerability Type
Cross-Chain Signature Replay

Severity
High

Description
The withdraw() function in the GatewayFacet contract appears to be vulnerable to a cross-chain signature replay attack. This type of attack occurs when a signature from one chain is used on another chain, effectively replaying the action in a different context. In this function, a signature is used to validate the request, but there is no differentiation between chains, allowing attackers to potentially use a valid signature from one chain on another.

Affected Code
- https://github.com/Tribally-Games/contracts/blob/73ed6a9f1f17d958d6f6dfdf8458377b5d8c925c/src/facets/GatewayFacet.sol#L54-L68

Impact
If this vulnerability is exploited, it could lead to unintended transfers of assets. An attacker could replay a legitimate request signature from one chain on another chain, causing assets to be transferred to the recipient unintentionally. This could result in financial losses and unexpected behavior in the contract.

Remediation
To fix this vulnerability, it is recommended to add chain_ID validation while generating and verifying the signature hash.

Retest
This issue is fixed by adding chain_ID in the signature payload.

Bug ID # 2 [Fixed]

Missing Zero Address Validations

Vulnerability Type
Missing Input Validation

Severity
Low

Description
The contracts were found to be setting new addresses without proper validations for zero addresses.
Address type parameters should include a zero-address check otherwise contract functionality may become inaccessible or tokens burned forever.
Depending on the logic of the contract, this could prove fatal and the users or the contracts could lose their funds, or the ownership of the contract could be lost forever.

Affected Variables and Line Numbers
- https://github.com/Tribally-Games/contracts/blob/73ed6a9f1f17d958d6f6dfdf8458377b5d8c925c/src/init/InitDiamond.sol#L11-L25

Impact
If address type parameters do not include a zero-address check, contract functionality may become unavailable or tokens may be burned permanently.

Remediation
Add a zero address validation to all the functions where addresses are being set.

Retest
This issue has been fixed by adding a Zero address validation.

Bug ID # 3 [Fixed]

Use safeTransfer/safeTransferFrom instead of transfer/transferFrom

Vulnerability Type
Missing Best Practices

Severity
Low

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

Affected Code
- https://github.com/Tribally-Games/contracts/blob/73ed6a9f1f17d958d6f6dfdf8458377b5d8c925c/src/libs/LibTribalToken.sol#L10
- https://github.com/Tribally-Games/contracts/blob/73ed6a9f1f17d958d6f6dfdf8458377b5d8c925c/src/libs/LibTribalToken.sol#L14

Impact
Using safeTransferFrom 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() and safeTransferFrom() instead of transfer() and transferFrom().

Retest
This issue has been fixed by replacing transfer() and transferFrom() with safeTransfer() and safeTransferFrom()

Bug ID # 4 [Partially 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., ^0.8.24. 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
- https://github.com/Tribally-Games/contracts/blob/73ed6a9f1f17d958d6f6dfdf8458377b5d8c925c/src/facets/ConfigFacet.sol#L2
- https://github.com/Tribally-Games/contracts/blob/73ed6a9f1f17d958d6f6dfdf8458377b5d8c925c/src/facets/GatewayFacet.sol#L2
- https://github.com/Tribally-Games/contracts/blob/73ed6a9f1f17d958d6f6dfdf8458377b5d8c925c/src/init/InitDiamond.sol#L2
- https://github.com/Tribally-Games/contracts/blob/73ed6a9f1f17d958d6f6dfdf8458377b5d8c925c/src/libs/LibAppStorage.sol#L2
- https://github.com/Tribally-Games/contracts/blob/73ed6a9f1f17d958d6f6dfdf8458377b5d8c925c/src/libs/LibAuth.sol#L2
- https://github.com/Tribally-Games/contracts/blob/73ed6a9f1f17d958d6f6dfdf8458377b5d8c925c/src/libs/LibErrors.sol#L2
- https://github.com/Tribally-Games/contracts/blob/73ed6a9f1f17d958d6f6dfdf8458377b5d8c925c/src/libs/LibTribalToken.sol#L2
- https://github.com/Tribally-Games/contracts/blob/73ed6a9f1f17d958d6f6dfdf8458377b5d8c925c/src/shared/AccessControl.sol#L2
- https://github.com/Tribally-Games/contracts/blob/73ed6a9f1f17d958d6f6dfdf8458377b5d8c925c/src/shared/ReentrancyGuard.sol#L2
- https://github.com/Tribally-Games/contracts/blob/73ed6a9f1f17d958d6f6dfdf8458377b5d8c925c/src/shared/Structs.sol#L2

Impact

Suppose the smart contract gets compiled and deployed with an older or too recent version of the solidity compiler. In that case, 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 low.

Remediation
Keep the compiler versions consistent in all the smart contract files. Do not allow floating pragmas anywhere. It is suggested to use the 0.8.25 pragma version
Reference: https://swcregistry.io/docs/SWC-103

Retest
This issue has been partially fixed by removing the floating pragma. The compiler version is still outdated.

Bug ID # 5 [Fixed]

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://github.com/Tribally-Games/contracts/blob/73ed6a9f1f17d958d6f6dfdf8458377b5d8c925c/src/facets/ConfigFacet.sol#L23-L25

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
This issue has been fixed by emitting events.

Bug ID # 6 [Not Fixed]

Cheaper Inequalities in if()

Vulnerability Type
Gas Optimization

Severity
Gas

Description
The contract was found to be doing comparisons using inequalities inside the "if" statement. When inside the "if" statements, non-strict inequalities (>=, <=) are usually cheaper than the strict equalities (>, <).

Affected Code
- https://github.com/Tribally-Games/contracts/blob/73ed6a9f1f17d958d6f6dfdf8458377b5d8c925c/src/facets/GatewayFacet.sol#L59
- https://github.com/Tribally-Games/contracts/blob/73ed6a9f1f17d958d6f6dfdf8458377b5d8c925c/src/libs/LibAuth.sol#L22

Impact
Using strict inequalities inside "if" statements costs more gas.

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

Retest
This issue has not been fixed.

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

20A TANJONG PAGAR ROAD, SINGAPORE (088443)

info@credshields.com