Audited by CRED SHIELDS

CredShields

Smart Contract Audit

November 7, 2025 • CONFIDENTIAL

Description

This document details the process and result of the Smart Contract audit performed by CredShields Technologies PTE. LTD. on behalf of Vouch between Oct 22nd, 2025, and Oct 28th, 2025. A retest was performed on Nov 4th, 2025.

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

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

Prepared for
Vouch

Table of Contents

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

Vouch engaged CredShields to perform a smart contract audit from Oct 22nd, 2025, to Oct 28th, 2025. During this timeframe, fifteen (15) vulnerabilities were identified. **A retest was performed on Nov 4th, 2025, and all the bugs have been addressed.**

During the audit, three (3) vulnerabilities were found with a severity rating of either High or Critical. These vulnerabilities represent the greatest immediate risk to "Vouch" 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
Smart Contract	1	2	2	5	2	3	**15**

Table: Vulnerabilities Per Asset in Scope

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

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

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

2.1 Preparation Phase

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

A testing window from Oct 22nd, 2025, to Oct 28th, 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
https://github.com/Vouchrun/vouch-staking/tree/ce58d0c34d1e49231df478fcfddc691baf54b34b

2.1.2 Documentation

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

2.1.3 Audit Goals

CredShields employs a combination of in-house tools and thorough manual review processes to deliver comprehensive smart contract security audits. The majority of the audit involves manual inspection of the contract's source code, guided by OWASP's Smart Contract Security Weakness Enumeration (SCWE) framework and an extended, self-developed checklist built from industry best practices. The team focuses on deeply understanding the contract's core logic, designing targeted test cases, and assessing business logic for potential vulnerabilities across OWASP's identified weakness classes.

CredShields aligns its auditing methodology with the OWASP Smart Contract Security projects, including the Smart Contract Security Verification Standard (SCSVS), the Smart Contract Weakness Enumeration (SCWE), and the Smart Contract Secure Testing Guide (SCSTG). These frameworks, actively contributed to and co-developed by the CredShields team, aim to bring consistency, clarity, and depth to smart contract security assessments. By adhering to these OWASP standards, we ensure that each audit is performed against a transparent, community-driven, and technically robust baseline. This approach enables us to deliver structured, high-quality audits that address both common and complex smart contract vulnerabilities systematically.

2.2 Retesting Phase

Vouch 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 asset and OWASP SCWE classification. Each asset section includes a summary highlighting the key risks and observations. The table in the executive summary presents the total number of identified security vulnerabilities per asset, categorized by risk severity based on the OWASP Smart Contract Security Weakness Enumeration framework.

3.1 Findings Overview

3.1.1 Vulnerability Summary

During the security assessment, fifteen (15) security vulnerabilities were identified in the asset.

VULNERABILITY TITLE	SEVERITY	SCWE | Vulnerability Type
User Can Drain Unlimited Holder Rewards When Unlock Period Is Zero	Critical	Inconsistent Accounting (SCWE-010)
Outdated Holder Reward Accumulators Lead To Incorrect Payouts	High	Inconsistent Accounting (SCWE-010)
Controller Cannot Recover or Wrap Received PLS	High	Missing functionality (SCWE-006)
Admin Allocation Update Can Distort Past Reward Distribution	Medium	Inconsistent Accounting (SCWE-010)
Lack of Emergency Recovery Mechanisms in Reward Pool Contracts Leading to Permanent Fund Loss	Medium	Inconsistent Accounting (SCWE-010)
Unchecked External Call Return Value in _updateRewardPoolBudgets Function	Low	Unchecked return value (SCWE-048)

Use safeTransfer instead of transfer	Low	Missing best practices
Missing VOUCH Token Validation in Liquidity Pool Creation	Low	Missing Input Validation (SC04-Lack Of Input Validation)
Missing zero address validations	Low	Lack of Input Validation (SC04-Lack Of Input Validation)
Missing events in important functions	Low	Missing Event Emission (SCWE-063)
Dead Code	Informational	Code With No Effects (SCWE-062)
Incorrect Natspec Documentation in Unstake Function Misrepresents Reward Claiming Behavior	Informational	Gas optimization (SCWE-082)
Gas Optimization in Increments	Gas	Gas optimization (SCWE-082)
Splitting Require/Revert Statements	Gas	Gas optimization (SCWE-082)
Custom error to save gas	Gas	Gas optimization (SCWE-082)

Table: Findings in Smart Contracts

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

Vouch is actively partnering with CredShields from this engagement to validate the discovered vulnerabilities' remediations. **A retest was performed on Nov 4th, 2025, and all the issues have been addressed.**

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

VULNERABILITY TITLE	SEVERITY	REMEDIATION STATUS
User Can Drain Unlimited Holder Rewards When Unlock Period Is Zero	Critical	Fixed [Nov 4th, 2025]
Outdated Holder Reward Accumulators Lead To Incorrect Payouts	High	Fixed [Nov 4th, 2025]
Controller Cannot Recover or Wrap Received PLS	High	Fixed [Nov 4th, 2025]
Admin Allocation Update Can Distort Past Reward Distribution	Medium	Fixed [Nov 4th, 2025]
Lack of Emergency Recovery Mechanisms in Reward Pool Contracts Leading to Permanent Fund Loss	Medium	Fixed [Nov 4th, 2025]
Unchecked External Call Return Value in _updateRewardPoolBudgets Function	Low	Fixed [Nov 4th, 2025]
Use safeTransfer instead of transfer	Low	Fixed [Nov 4th, 2025]
Missing VOUCH Token Validation in Liquidity Pool Creation	Low	Fixed [Nov 4th, 2025]
Missing zero address validations	Low	Fixed [Nov 4th, 2025]
Missing events in important functions	Low	Fixed [Nov 4th, 2025]
Dead Code	Informational	Fixed [Nov 4th, 2025]

Incorrect Natspec Documentation in Unstake Function Misrepresents Reward Claiming Behavior	Informational	Fixed [Nov 4th, 2025]
Gas Optimization in Increments	Gas	Fixed [Nov 4th, 2025]
Splitting Require/Revert Statements	Gas	Fixed [Nov 4th, 2025]
Custom error to save gas	Gas	Fixed [Nov 4th, 2025]

Table: Summary of findings and status of remediation

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

Bug ID #C001 [Fixed]

User Can Drain Unlimited Holder Rewards When Unlock Period Is Zero

Vulnerability Type
Inconsistent Accounting (SCWE-010)

Severity
Critical

Description
In the _startUnlock function, when the stakingToken is vouchToken and the standardUnlockPeriod is set to 0, the contract immediately transfers the user's stake without reducing principalStakedToken[address(vouchToken)] and userTotalVouchStaked[user]. Normally, these two mappings are used to track global and per-user VOUCH stake balances for holder reward distribution. Failing to update them leaves the system believing that the user still holds their stake, even though the tokens were withdrawn. This discrepancy allows the user to continue earning unlimited holder rewards without having any tokens staked. Additionally, future holder reward calculations that rely on principalStakedToken become inaccurate, creating state inconsistency in _calculateHolderRewardDeltas and _distributeHolderRewardDividends.

Affected Code
- https://github.com/Vouchrun/vouch-staking/blob/ce58d0c34d1e49231df478fcfddc691baf54b34b/contracts/VouchStaking.sol#L484-L489

Impacts
A malicious user can fully unstake their tokens while retaining userTotalVouchStaked credit, continuously claiming holder rewards indefinitely. This leads to total depletion of holder reward balances and invalid reward share calculations for other users.

Remediation
When standardUnlockPeriod == 0, ensure that principalStakedToken[address(vouchToken)] and userTotalVouchStaked[user] are both reduced before transferring tokens out.

Retest

This issue has been fixed by properly updating the principalStakedToken[address(vouchToken)] and userTotalVouchStaked[user] when claiming the principle staked using the startUnlock function while the standardUnlockPeriod was set to 0.

Bug ID #H001 [Fixed]

Outdated Holder Reward Accumulators Lead To Incorrect Payouts

Vulnerability Type
Inconsistent Accounting (SCWE-010)

Severity
High

Description
Several reward claim functions — specifically claim(uint256 _pid), claimHolderRewards(), claimAll(), and pendingHolderRewards(address) — distribute or calculate holder rewards based on outdated accumulator states. The holder reward accumulators (accVouchHolderRewardsPerShare, accVpIsHolderRewardsPerShare, accPIsHolderRewardsPerShare) are only updated inside _distributeHolderRewardDividends. However, none of the affected functions invoke _distributeHolderRewardDividends before performing reward calculations or transfers. As a result, users may receive stale or incomplete rewards since the latest unaccounted token deltas are never incorporated into the accumulators before payout.

Affected Code
- https://github.com/Vouchrun/vouch-staking/blob/ce58d0c34d1e49231df478fcfddc691baf54b34b/contracts/VouchStaking.sol#L394
- https://github.com/Vouchrun/vouch-staking/blob/ce58d0c34d1e49231df478fcfddc691baf54b34b/contracts/VouchStaking.sol#L411
- https://github.com/Vouchrun/vouch-staking/blob/ce58d0c34d1e49231df478fcfddc691baf54b34b/contracts/VouchStaking.sol#L429
- https://github.com/Vouchrun/vouch-staking/blob/ce58d0c34d1e49231df478fcfddc691baf54b34b/contracts/VouchStaking.sol#L958

Impacts
Users receive inaccurate reward amounts - This can lead to unfair reward distribution, accounting mismatches, and potential disputes over the correctness of user balances

Remediation
Call _distributeHolderRewardDividends before any holder reward computation or transfer in the affected functions to synchronize accumulator state.

Retest

This issue is fixed by calling _distributeHolderRewardDividends function before calculating the rewards to transfer.

Bug ID #H002 [Fixed]

Controller Cannot Recover or Wrap Received PLS

Vulnerability Type
Missing functionality (SCWE-006)

Severity
High

Description
The HolderRewardsVault contract includes a receive() payable function that allows the contract to accept native PLS transfers. In normal operation, this vault is expected to hold reward tokens and allow the controller to transfer them via pullTokenTo(...). However, the contract does not implement any mechanism to manage or recover received native PLS. Specifically, unlike LPRewardPool, it lacks a sync() or _wrapIfNeeded() function to convert incoming PLS to its wrapped equivalent (WPLS) or to withdraw the native balance. As a result, any PLS accidentally sent to this contract remains permanently locked. The root cause is the absence of native token handling logic within HolderRewardsVault.

Affected Code
- https://github.com/Vouchrun/vouch-staking/blob/ce58d0c34d1e49231df478fcfddc691baf54b34b/contracts/HolderRewardsVault.sol#L41

Impacts
Any PLS sent directly to HolderRewardsVault becomes irretrievable, resulting in a permanent loss of assets.

Remediation
Add a function callable by controller to wrap or withdraw any received PLS, similar to the _wrapIfNeeded() approach in LPRewardPool.

Retest
This issue is fixed by implementing the suggested fix.

Bug ID #M001 [Fixed]

Admin Allocation Update Can Distort Past Reward Distribution

Vulnerability Type
Inconsistent Accounting (SCWE-010)

Severity
Medium

Description
In the set(uint256 _pid, uint256 _allocPoint, bool _active) function, only the pool corresponding to the given _pid is updated using _calcAccLiquidityRewardsPerShare (for liquidity pools) or _calcAccStandardTriple (for standard pools). However, all other pools sharing the same rewardPool rely on the same global totalAllocPoint to calculate their proportional share of emissions. When one pool's allocPoint changes, the relative reward share of every other pool in that rewardPool changes retroactively.

Because those other pools' accumulators (liqAccVouchPerShare, etc.) are not updated before the change, they end up using the new allocation ratio for past unaccounted time. This introduces a temporal inconsistency where past emissions are distributed using future ratios, leading to over- or under-payment across pools.

Affected Code

- https://github.com/Vouchrun/vouch-staking/blob/ce58d0c34d1e49231df478fcfddc691baf54b34b/contracts/VouchStaking.sol#L843

Impacts
All pools mapped to the same rewardPool become desynchronized, causing inaccurate emission accounting. Users interacting after an allocation change may claim either inflated or reduced rewards depending on pool order and timing of interaction. This can accumulate reward imbalances over time and erode the fairness of the distribution model.

Remediation
Before changing a pool's allocation, update all pools associated with the same rewardPool to settle their pending emissions under the previous ratio.

Retest

All pools linked to the same rewardPool are now updated before changing allocPoint, preventing allocation-based reward inconsistencies.

Bug ID #M002 [Fixed]

Lack of Emergency Recovery Mechanisms in Reward Pool Contracts Leading to Permanent Fund Loss

Vulnerability Type
Missing functionality (SCWE-006)

Severity
Medium

Description
The reward pool contracts LPRewardPool, StakingRewardPool, and HolderRewardsVault all accept native tokens through payable receive functions but completely lack emergency recovery mechanisms. This design flaw creates permanent fund loss scenarios when users invoke the exit function without claiming rewards first, as unclaimed rewards become permanently trapped. Mathematical rounding errors and contract upgrade scenarios further exacerbate this issue, leaving residual funds stranded across all three reward pools with no administrative recourse for recovery.

The absence of token sweep functions or emergency withdrawal capabilities means any funds not perfectly accounted for in reward distribution calculations become permanently locked. This includes orphaned rewards from exited users, calculation dust amounts, and any mistakenly sent tokens.

Affected Code

- https://github.com/Vouchrun/vouch-staking/blob/ce58d0c34d1e49231df478fcfddc691baf54b34b/contracts/StakingRewardPool.sol#L56
- https://github.com/Vouchrun/vouch-staking/blob/ce58d0c34d1e49231df478fcfddc691baf54b34b/contracts/LPRewardPool.sol#L58
- https://github.com/Vouchrun/vouch-staking/blob/ce58d0c34d1e49231df478fcfddc691baf54b34b/contracts/HolderRewardsVault.sol#L41

Impacts
Permanent loss of user rewards occurs when participants use the exit function during emergencies without claiming accumulated rewards first.

Remediation

Implement emergency recovery functions in all reward contracts with proper access controls.

Retest
This issue has been fixed by introducing emergency recovery functions.

Bug ID #L001 [Fixed]

Unchecked External Call Return Value in _updateRewardPoolBudgets Function

Vulnerability Type
Unchecked return value (SCWE-048)

Severity
Low

Description
The _updateRewardPoolBudgets function contains an error handling vulnerability where return values from external calls are completely ignored. The function executes a low-level call to the sync() function on reward pool contracts but fails to check whether the call actually succeeded or failed. This silent failure mode means that if the sync() call reverts due to contract pauses, gas limitations, implementation bugs, or interface mismatches, the function continues execution with potentially stale token balance data.

Affected Code
- https://github.com/Vouchrun/vouch-staking/blob/ce58d0c34d1e49231df478fcfddc691baf54b34b/contracts/VouchStaking.sol#L1496C9-L1497C11

Impacts
Inaccurate reward emission rates affect all users across multiple pools, leading to economic miscalculations and unpredictable reward distributions.

Remediation
Implement proper boolean success checking with require statement.

Retest
This vulnerability is fixed.

Bug ID #L002 [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/Vouchrun/vouch-staking/blob/ce58d0c34d1e49231df478fcfddc691baf54b34b/contracts/StakingRewardPool.sol#L79
- https://github.com/Vouchrun/vouch-staking/blob/ce58d0c34d1e49231df478fcfddc691baf54b34b/contracts/StakingRewardPool.sol#L83
- https://github.com/Vouchrun/vouch-staking/blob/ce58d0c34d1e49231df478fcfddc691baf54b34b/contracts/StakingRewardPool.sol#L87
- https://github.com/Vouchrun/vouch-staking/blob/ce58d0c34d1e49231df478fcfddc691baf54b34b/contracts/StakingRewardPool.sol#L122-L124

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() and instead of transfer().

Retest
This issue is fixed by calling safeTransfer() instead of the transfer()

Bug ID #L003 [Fixed]

Missing VOUCH Token Validation in Liquidity Pool Creation

Vulnerability Type
Lack of Input Validation (SC04-Lack Of Input Validation)

Severity
Low

Description
The addLiquidityPool function lacks validation to prevent VOUCH tokens from being added as liquidity pools, creating fundamental conflicts in the protocol's economic model. This oversight allows administrators to accidentally or maliciously configure VOUCH tokens as liquidity pool staking tokens, enabling double reward accumulation where the same VOUCH stake earns both liquidity pool triple rewards and global holder rewards simultaneously. The absence of this validation breaks the clear separation between standard pools with unlock periods and liquidity pools with instant withdrawals, allowing users to bypass intended economic safeguards.

Affected Code
- https://github.com/Vouchrun/vouch-staking/blob/ce58d0c34d1e49231df478fcfddc691baf54b34b/contracts/VouchStaking.sol#L810C5-L835C6

Impacts
Double reward accumulation corrupts economic model and dilutes legitimate user rewards. Accounting system conflicts lead to inaccurate holder distributions and principal tracking. Unlock mechanism bypass enables instant VOUCH withdrawals defeating standard pool safeguards.

Remediation
Add explicit validation preventing VOUCH tokens in liquidity pools.

Retest
A validation check is now been added to prevent VOUCH from being used as a liquidity pool token, removing the possibility of double rewards and model conflicts.

Bug ID #L004 [Fixed]

Missing zero address validations

Vulnerability Type
Missing Input Validation (SC04-Lack Of 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 Code

- https://github.com/Vouchrun/vouch-staking/blob/ce58d0c34d1e49231df478fcfddc691baf54b34b/contracts/StakingRewardPool.sol#L53
-

Impacts
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.

Bug ID #L005 [Fixed]

Missing events in important functions

Vulnerability Type
Missing Event Emission (SCWE-063)

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
- https://github.com/Vouchrun/vouch-staking/blob/ce58d0c34d1e49231df478fcfddc691baf54b34b/contracts/HolderRewardsVault.sol#L37-L39
- https://github.com/Vouchrun/vouch-staking/blob/ce58d0c34d1e49231df478fcfddc691baf54b34b/contracts/LPRewardPool.sol#L43-L46
- https://github.com/Vouchrun/vouch-staking/blob/ce58d0c34d1e49231df478fcfddc691baf54b34b/contracts/StakingRewardPool.sol#L58-L60

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 in important functions.

Bug ID #I001 [Fixed]

Dead Code

Vulnerability Type
Code With No Effects (SCWE-062)

Severity
Informational

Description
It is recommended to keep the production repository clean to prevent confusion and the introduction of vulnerabilities. The functions and parameters, contracts, and interfaces that are never used or called externally or from inside the contracts should be removed when the contract is deployed on the mainnet.

Affected Code

- https://github.com/Vouchrun/vouch-staking/blob/ce58d0c34d1e49231df478fcfddc691baf54b34b/contracts/VouchStaking.sol#L647C5-L653C6
- https://github.com/Vouchrun/vouch-staking/blob/ce58d0c34d1e49231df478fcfddc691baf54b34b/contracts/VouchStaking.sol#L418C5-L420C6

Impacts
This does not impact the security aspect of the Smart contract but prevents confusion when the code is sent to other developers or auditors to understand and implement.
This reduces the overall size of the contracts and also helps in saving gas.

Remediation
If the library functions are not supposed to be used anywhere, consider removing them from the contract.

Retest
This issue is fixed by removing these functions from the contract.

Bug ID #I002 [Fixed]

Incorrect Natspec Documentation in Unstake Function Misrepresents Reward Claiming Behavior

Vulnerability Type
Poor Governance Documentation (SCWE-015)

Severity
Informational

Description
The natspec documentation for the unstake function incorrectly states that it handles holder rewards for VOUCH pools, creating a fundamental mismatch between documented behavior and actual implementation. The function explicitly reverts with "standard: unlock only" for standard pool types, which are the exclusive pool type capable of containing VOUCH tokens for holder rewards. This documentation error misleads developers and users into believing unstake processes holder rewards when in reality the function only operates on liquidity pools and never interacts with VOUCH tokens or holder reward distributions.

Affected Code
- https://github.com/Vouchrun/vouch-staking/blob/ce58d0c34d1e49231df478fcfddc691baf54b34b/contracts/VouchStaking.sol#L307C8-L309C41

Impacts
Developer confusion and integration errors due to mismatched documentation. Incorrect user expectations about reward claiming behavior.

Remediation
Correct natspec to accurately reflect liquidity pool-only operation. Clarify reward claiming separation between pool types. Update documentation to match actual function constraints.

Retest
The natspec documentation has been corrected to accurately reflect that unstake applies only to liquidity pools and does not handle VOUCH holder rewards.

Bug ID #G001 [Fixed]

Gas Optimization in Increments

Vulnerability Type
Gas optimization (SCWE-082)

Severity
Gas

Description
The contract uses two for loops**,** which use post increments for the variable "**i**".
The contract can save some gas by changing this to **++i**.
++i costs less gas compared to **i++** or **i += 1** for unsigned integers. In **i++**, the compiler has to create a temporary variable to store the initial value. This is not the case with **++i** in which the value is directly incremented and returned, thus, making it a cheaper alternative.

Vulnerable Code

- https://github.com/Vouchrun/vouch-staking/blob/ce58d0c34d1e49231df478fcfddc691baf54b34b/contracts/VouchStaking.sol#L430
- https://github.com/Vouchrun/vouch-staking/blob/ce58d0c34d1e49231df478fcfddc691baf54b34b/contracts/VouchStaking.sol#L1079

Impacts
Using **i++** instead of **++i** costs the contract deployment around 600 more gas units.

Remediation
It is recommended to switch to **++i** and change the code accordingly so the function logic remains the same and meanwhile saves some gas.

Retest
This issue has been fixed.

Bug ID #G002 [Fixed]

Splitting Require/Revert Statements

Vulnerability Type
Gas Optimization (SCWE-082)

Severity
Gas

Description
Require/Revert statements when combined using operators in a single statement usually lead to a larger deployment gas cost but with each runtime calls, the whole thing ends up being cheaper by some gas units.

Affected Code

- https://github.com/Vouchrun/vouch-staking/blob/ce58d0c34d1e49231df478fcfddc691baf54b34b/contracts/VouchStaking.sol#L319
- https://github.com/Vouchrun/vouch-staking/blob/ce58d0c34d1e49231df478fcfddc691baf54b34b/contracts/VouchStaking.sol#L752
- https://github.com/Vouchrun/vouch-staking/blob/ce58d0c34d1e49231df478fcfddc691baf54b34b/contracts/VouchStaking.sol#L779
- https://github.com/Vouchrun/vouch-staking/blob/ce58d0c34d1e49231df478fcfddc691baf54b34b/contracts/VouchStaking.sol#L816
- https://github.com/Vouchrun/vouch-staking/blob/ce58d0c34d1e49231df478fcfddc691baf54b34b/contracts/VouchStaking.sol#L817

Impacts
The multiple conditions in one **require/revert** statement combine require/revert statements in a single line, increasing deployment costs and hindering code readability.

Remediation
It is recommended to separate the **require/revert** statements with one statement/validation per line.

Retest
This issue has been fixed.

Bug ID #G003 [Fixed]

Custom error to save gas

Vulnerability Type
Gas Optimization (SCWE-082)

Severity
Gas

Description
During code analysis, it was observed that the smart contract is using the revert() statements for error handling. However, since Solidity version 0.8.4, custom errors have been introduced, providing a better alternative to the traditional revert(). Custom errors allow developers to pass dynamic data along with the revert, making error handling more informative and efficient. Furthermore, using custom errors can result in lower gas costs compared to the revert() statements.

Affected Code
- https://github.com/Vouchrun/vouch-staking/blob/ce58d0c34d1e49231df478fcfddc691baf54b34b/contracts/StakingRewardPool.sol#L89
- https://github.com/Vouchrun/vouch-staking/blob/ce58d0c34d1e49231df478fcfddc691baf54b34b/contracts/VouchStaking.sol#L317

Impacts
Custom errors allow developers to provide more descriptive error messages with dynamic data. This provides better insights into the cause of the error, making it easier for users and developers to understand and address issues.

Remediation
It is recommended to replace all the instances of revert() statements with error() to save gas..

Retest
This issue has 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