Audited by CRED SHIELDS

CredShields

Smart Contract Audit

14th August, 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 July 28th, 2025, and August 6th, 2025. A retest was performed on August 13th, 2025.

Author

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

Reviewers

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

Prepared for

Vouch

Table of Contents

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

Vouch engaged CredShields to perform a smart contract audit from July 28th, 2025, to August 6th, 2025. During this timeframe, 24 vulnerabilities were identified. **A retest was performed on August 13th, 2025, and all the bugs have been addressed.**

During the audit, 2 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
Vouch Token and Distribution Contracts	2	0	5	7	0	10	**24**
	2	**0**	**5**	**7**	**0**	**10**	**24**

Table: Vulnerabilities Per Asset in Scope

The CredShields team conducted the security audit to focus on identifying vulnerabilities in the Vouch Token and Distribution 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 Vouch Token and Distribution 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 28th, 2025, to August 6th, 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-token/tree/d4cf5a1a3e844d00cc0715ade1143d10e8324633

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, 24 security vulnerabilities were identified in the asset.

VULNERABILITY TITLE	SEVERITY	SCWE | Vulnerability Type
Attacker Can Claim Rewards Twice Via Cross-Function Reentrancy	Critical	Reentrancy (SCWE-046)
Failed Voter Transfers Permanently Lock User Funds	Critical	Fund Stuck
Hardcoded Swap Deadline Enables Transaction Manipulation by Block Builders	Medium	Transaction Manipulation (SCWE-028)
Zero Slippage Setting Enables Unfavorable Swap Execution	Medium	Slippage Risk (SCWE-028)
Externally Owned Account Can Bypass Launch Lock via tx.origin	Medium	Improper Authorization (SCWE-016)
Incorrect Shareholder Removal Causes Duplicates and Accounting Inconsistency	Medium	Business Logic (SC03-LogicErrors)
Admin Can Set Critical Recipients to Zero Address	Medium	Uncaught Exceptions (SCWE-004)

Lack of Admin Controls to Update Critical External Addresses	Low	Maintainability Risk / Centralization
Admin Can Only Remove Last Pair Due to Inefficient Array Management	Low	Data Structure Mismanagement
Fallback Distribution Lacks Threshold Check for All Token Balances	Low	Missing Validation (SCWE-004)
Constructor Accepts Expired Lock Time Without Validation	Low	Missing Validation (SCWE-004)
Missing zero address validations	Low	Missing Input Validation (SC04-Lack Of Input Validation)
Outdated Pragma	Low	Outdated Compiler Version (SCWE-061)
Missing events in important functions	Low	Missing Best Practices (SCWE-063)
Boolean Equality	Gas	Gas Optimization (SCWE-082)
Large Number Literals	Gas	Gas Optimization (SCWE-082)
Cheaper Inequalities in if()	Gas	Gas Optimization (SCWE-082)
Cheaper Inequalities in require()	Gas	Gas Optimization (SCWE-082)
Cheaper conditional operators	Gas	Gas Optimization (SCWE-082)
Gas Optimization for State Variables	Gas	Gas Optimization (SCWE-082)
Gas Optimization in Require/Revert Statements	Gas	Gas Optimization (SCWE-082)
Gas Optimization in Increments	Gas	Gas Optimization (SCWE-082)

Public constants can be private	Gas	Gas Optimization (SCWE-082)
State Variable Can Be Marked As Constants	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 August 13th, 2025, and all the issues have been addressed.**

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

VULNERABILITY TITLE	SEVERITY	REMEDIATION STATUS
Attacker Can Claim Rewards Twice Via Cross-Function Reentrancy	Critical	Fixed [August 13, 2025]
Failed Voter Transfers Permanently Lock User Funds	Critical	Won't Fix [August 13, 2025]
Hardcoded Swap Deadline Enables Transaction Manipulation by Block Builders	Medium	Won't Fix [August 13, 2025]
Zero Slippage Setting Enables Unfavorable Swap Execution	Medium	Won't Fix [August 13, 2025]
Externally Owned Account Can Bypass Launch Lock via tx.origin	Medium	Won't Fix [August 13, 2025]
Incorrect Shareholder Removal Causes Duplicates and Accounting Inconsistency	Medium	Fixed [August 13, 2025]
Admin Can Set Critical Recipients to Zero Address	Medium	Fixed [August 13, 2025]
Lack of Admin Controls to Update Critical External Addresses	Low	Won't Fix [August 13, 2025]
Admin Can Only Remove Last Pair Due to Inefficient Array Management	Low	Fixed [August 13, 2025]
Fallback Distribution Lacks Threshold Check for All Token Balances	Low	Fixed [August 13, 2025]
Constructor Accepts Expired Lock Time Without Validation	Low	Won't Fix [August 13, 2025]

Missing zero address validations	Low	Partially Fixed [August 13, 2025]
Outdated Pragma	Low	Won't Fix [August 13, 2025]
Missing events in important functions	Low	Won't Fix [August 13, 2025]
Boolean Equality	Gas	Won't Fix [August 13, 2025]
Large Number Literals	Gas	Won't Fix [August 13, 2025]
Cheaper Inequalities in if()	Gas	Won't Fix [August 13, 2025]
Cheaper Inequalities in require()	Gas	Won't Fix [August 13, 2025]
Cheaper conditional operators	Gas	Won't Fix [August 13, 2025]
Gas Optimization for State Variables	Gas	Won't Fix [August 13, 2025]
Gas Optimization in Require/Revert Statements	Gas	Won't Fix [August 13, 2025]
Gas Optimization in Increments	Gas	Won't Fix [August 13, 2025]
Public constants can be private	Gas	Won't Fix [August 13, 2025]
State Variable Can Be Marked As Constants	Gas	Won't Fix [August 13, 2025]

Table: Summary of findings and status of remediation

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

Bug ID #C001 [Fixed]

Attacker Can Claim Rewards Twice Via Cross-Function Reentrancy

Vulnerability Type
Reentrancy (SCWE-046)

Severity
Critical

Description
The triggerTokenSends() function is protected by nonReentrant, but the receive function is not. During triggerTokenSends(), when a voter is paid with call{value: amountPlsPerVoter}(""), a malicious voter contract can invoke the contract's receive() function before triggerTokenSends completes. Since receive repeats the reward distribution logic and also transfers tokens, this allows the attacker to trigger a second distribution in the same transaction. The root cause is the absence of a reentrancy guard or state-locking in receive, combined with both functions modifying and reading the same reward balances and distribution timestamps.

Affected Code
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/VotersDistributor.sol#L67-L157

Impacts
An attacker controlling a voter address can reenter through receive while rewards are being distributed, causing them and other voters to receive distributions twice in a single execution, depleting contract funds.

Remediation
It is recommended to add a nonReentrant guard to receive() and update lastDistributionTime before any external calls in triggerTokenSends().

Retest
This issue has been fixed by updating the triggerTokenSends before making any low-level calls.

Bug ID #C002 [Won't Fix]

Failed Voter Transfers Permanently Lock User Funds

Vulnerability Type
Fund Stuck

Severity
Critical

Description
In the receive() function, ETH (PLS token) is distributed to each voter using a low-level call. If a voter is a smart contract without a receive() or fallback() function, or if any call fails for other reasons (e.g., out-of-gas, reverts), the ETH transfer to that address fails. Although execution continues after a failed call, the function has no mechanism to track or retry failed distributions. Consequently, any voter who fails to receive ETH in that transaction has no way to claim their funds later. The root cause is the lack of a fallback mechanism for failed transfers and reliance on immediate delivery to untrusted addresses.

Affected Code

- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/VotersDistributor.sol#L96

Impacts
Any voter who fails to receive the native PLS distribution—due to missing fallback function, gas limits, or transient errors—will permanently lose their entitlement, with no option to retry or manually claim their share. This results in irreversible loss of funds for affected participants and unequal treatment of voters.

Remediation
Replace direct ETH transfer with a claim-based mechanism that records each voter's entitlement, allowing them to manually claim funds at their discretion.

Retest
Client's comment: This is the desired behavior.

Bug ID #M001 [Won't Fix]

Hardcoded Swap Deadline Enables Transaction Manipulation by Block Builders

Vulnerability Type
Transaction Manipulation (SCWE-028)

Severity
Medium

Description
The contract performs a token-to-ETH & ETH-to-token swap via router.swapExactTokensForETHSupportingFeeOnTransferTokens, passing block.timestamp as the deadline. This hardcoded deadline does not protect against transaction withholding by miners or block builders. Since the contract does not validate a user-specified deadline, there is no mechanism to prevent delayed execution during unfavorable market conditions. The root cause is relying on block.timestamp for deadline enforcement instead of accepting an externally defined cutoff time.

Affected Code
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/Vouch.sol#L271
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L513
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L720
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/LiquidityManager.sol#L217

Impacts
A malicious block builder can intentionally delay the inclusion of the transaction, executing it when prices have shifted significantly, causing users to receive less ETH than expected.

Remediation
Accept a uint256 deadline parameter and enforce it at the call site to limit the transaction execution window.

Retest

Client's comment: Acknowledged. It is not possible for us to pass a deadline value within the standard ERC20 transfer function. The _transferFrom function which is triggered by the transfer function, does many swaps, and therefore, we must use block.timestamp or some value that cannot revert. Additionally, due to Vouch's tax fees, it isn't very advantageous for any block builder to attempt to take advantage of this for some reason.

Bug ID #M002 [Won't Fix]

Zero Slippage Setting Enables Unfavorable Swap Execution

Vulnerability Type
Slippage Risk (SCWE-028)

Severity
Medium

Description
The contract performs a token-to-ETH & ETH-to-token swap using router.swapExactTokensForETHSupportingFeeOnTransferTokens, where the amountOutMin parameter is hardcoded to 0. This disables slippage protection, allowing the swap to proceed regardless of how much ETH is received. Without a minimum expected output, the transaction can be executed even under extreme price impact or sandwich attacks, leading to substantial user loss. The root cause is setting amountOutMin to zero instead of dynamically calculating it based on price feeds or a tolerated slippage percentage.

Affected Code

- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/Vouch.sol#L268
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L513
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/LiquidityManager.sol#L253-L254
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/LiquidityManager.sol#L217

Impacts
The contract may perform token swaps at significantly unfavorable rates, exposing the protocol to value loss from front-running or volatile market conditions.

Remediation
Calculate amountOutMin using current price data and apply a slippage tolerance before executing the

Retest

Client's comment: The team acknowledges that internally calculating getAmountsOut for slippage protection is ineffective against MEV attacks, as bots can manipulate pool reserves before the call, producing an artificially low expected output and rendering the slippage check meaningless. In a V2-only PulseChain environment with few price oracles, this makes preventing such attacks impractical. As a mitigation, most tax tokens set amountOutMin to zero and rely on keeping swap sizes small and tax fees high enough to make MEV exploitation unprofitable.

Bug ID #M003 [Won't Fix]

Externally Owned Account Can Bypass Launch Lock via tx.origin

Vulnerability Type
Improper Authorization (SCWE-016)

Severity
Medium

Description
In the _transfer function, the contract checks if launchedAt > 0 or if tx.origin is an admin or equal to the airDropper. This logic uses tx.origin instead of msg.sender to validate ownership or authorization. Since tx.origin always returns the original external account that started the transaction, it can be manipulated through intermediate contract calls. A malicious contract can call the token contract on behalf of an unsuspecting user, allowing an unauthorized actor to bypass pre-launch restrictions.

Affected Code

- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/Vouch.sol#L148

Impacts
An attacker can exploit a user's wallet by having them unknowingly initiate a transaction that triggers _transfer, thereby bypassing the launch check and performing token transfers before intended launch conditions are met.

Remediation
It is recommended to replace tx.origin with msg.sender to ensure that only direct callers are validated as admins or airdroppers.

Retest
Client's comment: Acknowledged. This is due to the unique deployment, and setup process, along with the fact that the onlyAdmin address is a GnosisSafe... I must utilize tx.origin in the pre-launch phase so that the safe's signer address is allowed to trigger transfers to add liquidity pre-launch. msg.sender will not work here, due to the additional transfers that take place when adding liquidity, transfers to the router, for example.

Bug ID #M004 [Fixed]

Incorrect Shareholder Removal Causes Duplicates and Accounting Inconsistency

Vulnerability Type
Business Logic (SC03-LogicErrors)

Severity
Medium

Description
The removeShareholder function attempts to remove a shareholder from the shareholders array and update three separate index mappings: shareholderVouchIndexes, shareholderVpIsIndexes, and shareholderPIsIndexes. However, it performs multiple overwrites using shareholders[shareholders.length - 1] without verifying index uniqueness or checking for overlapping positions. When removing multiple shareholders sequentially, the last element of the array is repeatedly copied into multiple intermediate positions, corrupting the array and duplicating entries. Only a single .pop() operation is called, reducing the array length by 1 even though multiple entries have been invalidated.

For example, consider the array:
[1, 2, 3, 4, 5, 6, 7, 8, 9]

If removeShareholder(2), removeShareholder(3), and removeShareholder(4) are called in sequence:

1. For removeShareholder(2), index 1 (2) is overwritten with 9:
   → [1, 9, 3, 4, 5, 6, 7, 8, 9]
2. For removeShareholder(3), index 2 (3) is also overwritten with 9:
   → [1, 9, 9, 4, 5, 6, 7, 8, 9]
3. For removeShareholder(4), index 3 (4) becomes 9:
   → [1, 9, 9, 9, 5, 6, 7, 8, 9]

Yet the array length is only reduced once, resulting in:
[1, 9, 9, 9, 5, 6, 7, 8] – with three duplicate 9s and stale index mappings.

Affected Code

- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L883-L894

Impacts

This leads to multiple protocol issues such as incorrect accounting of total shareholders and misleading governance participation counts or reward distributions.

Remediation

Ensure only one overwrite per removal, and perform array clean-up and pop() once per shareholder index.

Eg:

```
   function removeShareholder(address shareholder) internal {
      shareholders[ shareholderVouchIndexes[ shareholder ]] = shareholders[ shareholders.length -
1];
      shareholderVouchIndexes[ shareholders[ shareholders.length - 1]] =
shareholderVouchIndexes[ shareholder ];

+      shareholders.pop();

      shareholders[ shareholderVplsIndexes[ shareholder ]] = shareholders[ shareholders.length -
1];       shareholderVplsIndexes[ shareholders[ shareholders.length - 1]] =
shareholderVplsIndexes[ shareholder ];

+      shareholders.pop();

      shareholders[ shareholderPlsIndexes[ shareholder ]] = shareholders[ shareholders.length - 1];
      shareholderPlsIndexes[ shareholders[ shareholders.length - 1]] =
shareholderPlsIndexes[ shareholder ];

+      shareholders.pop();
   }
```

Retest

This issue has been fixed by updating the function logic to properly remove the share holder from the mapping.

Bug ID #M005 [Fixed]

Admin Can Set Critical Recipients to Zero Address

Vulnerability Type
Uncaught Exceptions (SCWE-004)

Severity
Medium

Description
In DaoDistributor.setRecipientAddresses, the function allows the admin to update several key recipient addresses including validators, feePool, safu, and daoTreasury. However, there is no check to prevent setting any of these addresses to the zero address. If any are accidentally or maliciously set to address(0), future transfers or distributions to those entities will fail or be permanently lost. The root cause is missing validation for non-zero input addresses in a critical configuration function.

Affected Code

- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L259-L275

Impacts
Setting a recipient to the zero address may lead to lost funds, halted reward flows, or broken downstream functionality depending on how each address is used.

Remediation
Add a require() check to ensure that none of the input addresses are the zero address.

Retest
This issue has been fixed by adding zero address validations.

Bug ID #L001 [Won't Fix]

Lack of Admin Controls to Update Critical External Addresses

Vulnerability Type
Maintainability Risk / Centralization

Severity
Low

Description
The contract hardcodes several external addresses and contract instances without providing administrative functions to update them. While constants may be appropriate for immutable values, some addresses are declared as mutable but remain fixed due to missing setter functions. This design choice prevents the system from adapting to changes such as third-party contract upgrades, address deprecations, or infrastructure migrations.

Affected Code

- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/Vouch.sol#L24
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/VotersDistributor.sol#L35-L36
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/LiquidityManager.sol#L18-L22
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/LiquidityManager.sol#L25-L26
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/LPLocker.sol#L30
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L18-L19
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L21

Impacts
If any hardcoded external dependency becomes deprecated or compromised, developers cannot redirect interactions, leading to degraded functionality or service disruption.

Remediation

It is recommended to add administrative functions to update all mutable external address and contract references.

Retest
Client's comment: Acknowledged. These addresses are immutable contracts.

Bug ID #L002 [Fixed]

Admin Can Only Remove Last Pair Due to Inefficient Array Management

Vulnerability Type
Data Structure Mismanagement

Severity
Low

Description
The contract provides addPair() and removeLastPair() functions to manage an array of pair addresses. However, the removal function only supports deleting the last entry using pop(). This constraint limits administrative flexibility, as there is no way to remove a specific pair without reconstructing the array or relying on insertion order.

Affected Code
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/Vouch.sol#L354-L357

Impacts
Administrators cannot efficiently remove arbitrary pair entries, which may lead to operational inefficiencies or the persistence of invalid data in the contract state.

Remediation
It is recommended to replace the current removal logic with a swap-and-pop approach that enables indexed removal from the array.

Retest
This issue has been fixed by adding another function to remove pairs from the array.

Bug ID #L003 [Fixed]

Fallback Distribution Lacks Threshold Check for All Token Balances

Vulnerability Type
Missing Validation (SCWE-004)

Severity
Low

Description
The receive() function proceeds with token distribution even when only one asset meets its respective threshold, unlike triggerTokenSends, which requires all thresholds to be met. This inconsistent behavior may lead to partial distributions.

Affected Code

- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/VotersDistributor.sol#L67-L108

Impacts
If only one asset meets its threshold, the fallback function may distribute it while skipping others, leading to inconsistent reward expectations across interfaces and functions.

Remediation
Add a comprehensive threshold check to the receive() function, mirroring the validation logic used in triggerTokenSends.

```
+ if (address(this).balance < plsDistributionThreshold ||
+     vpls.balanceOf(address(this)) < vplsDistributionThreshold ||
+     vouch.balanceOf(address(this)) < vouchDistributionThreshold) {
+     return;
+ }
```

Retest
This issue has been fixed by adding a validation.

Bug ID #L004 [Won't Fix]

Constructor Accepts Expired Lock Time Without Validation

Vulnerability Type
Missing Validation (SCWE-004)

Severity
Low

Description
In the constructor, the _lockTime parameter is only checked to ensure it does not exceed one year from block.timestamp. However, there is no validation to ensure _lockTime is set after the current time. As a result, an already expired lock time can be set at deployment, unintentionally or maliciously bypassing time-based logic reliant on lockTime.

Affected Code

- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/LPLocker.sol#L64

Impacts
If _lockTime is set to a past timestamp, lock enforcement mechanisms may be bypassed immediately upon deployment, weakening any time-based constraints expected by the contract.

Remediation
Ensure _lockTime is strictly greater than block.timestamp in the constructor.

```
+ require(_lockTime > block.timestamp, "Lock time must be in the future");
```

Retest
Client's comment: Acknowledged, this is alright.

Bug ID #L005 [Partially 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-token-d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/Vouch.sol#L295
- https://github.com/Vouchrun/VOUCH-token-d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/Vouch.sol#L299
- https://github.com/Vouchrun/VOUCH-token-d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/Vouch.sol#L323
- https://github.com/Vouchrun/VOUCH-token-d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/Vouch.sol#L327
- https://github.com/Vouchrun/VOUCH-token-d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/LiquidityManager.sol#L157
- https://github.com/Vouchrun/VOUCH-token-d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L275
- https://github.com/Vouchrun/VOUCH-token-d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L279
- https://github.com/Vouchrun/VOUCH-token-d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L283

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 partially addressed.

Bug ID #L006 [Won't Fix]

Outdated Pragma

Vulnerability Type
Outdated Compiler Version (SCWE-061)

Severity
Low

Description
The smart contract is using an outdated version of the Solidity compiler specified by the pragma directive i.e. 0.8.20. Solidity is actively developed, and new versions frequently include important security patches, bug fixes, and performance improvements. Using an outdated version exposes the contract to known vulnerabilities that have been addressed in later releases. Additionally, newer versions of Solidity often introduce new language features and optimizations that improve the overall security and efficiency of smart contracts.

Affected Code

- https://github.com/Vouchrun/VOUCH-token-d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/LPLocker.sol#L3
- https://github.com/Vouchrun/VOUCH-token-d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/Utilities.sol#L2
- https://github.com/Vouchrun/VOUCH-token-d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/VotersDistributor.sol#L2
- https://github.com/Vouchrun/VOUCH-token-d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/Vouch.sol#L2
- https://github.com/Vouchrun/VOUCH-token-d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/LiquidityManager.sol#L3
- https://github.com/Vouchrun/VOUCH-token-d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/ValidatorFeeDepositor.sol#L2
- https://github.com/Vouchrun/VOUCH-token-d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L2

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

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

Remediation
It is suggested to use the 0.8.29 pragma version.
Reference: https://scs.owasp.org/SCWE/SCSVS-CODE/SCWE-061/

Retest
Client's comment: Acknowledged.

Bug ID #L007 [Won't Fix]

Missing events in important functions

Vulnerability Type
Missing Best Practices (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-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/LPLocker.sol#L237-L241
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/LPLocker.sol#L243-L261
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/VotersDistributor.sol#L39-L45
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/VotersDistributor.sol#L47-L49
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/VotersDistributor.sol#L67-L108
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/Vouch.sol#L293-L295
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/Vouch.sol#L297-L299
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/Vouch.sol#L301-L304
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/Vouch.sol#L306-L309

- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/Vouch.sol#L311-L319
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/Vouch.sol#L321-L323
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/Vouch.sol#L325-L327
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/LiquidityManager.sol#L229-L242
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/ValidatorFeeDepositor.sol#L30-L32
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/ValidatorFeeDepositor.sol#L34-L44
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L241-L253
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L255-L257
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L259-L275
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L277-L279
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L281-L283
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L293-L309
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L311-L327
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L329-L343
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L801-L805
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L896-L912

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's comment: Acknowledged.

Bug ID #G001 [Won't Fix]

Boolean Equality

Vulnerability Type
Gas Optimization (SCWE-082)

Severity
Gas

Description
The contract was found to be equating variables with a boolean constant inside a "require()" statement which is not recommended and is unnecessary. Boolean constants can be used directly in conditionals.

Affected Code

- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/Vouch.sol#L246
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/Vouch.sol#L247

Impacts
Equating the values to boolean constants in conditions cost gas and can be used directly.

Remediation
It is recommended to use boolean constants directly. It is not required to equate them to true or false.

Retest
Client's comment: Acknowledged.

Bug ID #G002 [Won't Fix]

Large Number Literals

Vulnerability Type
Gas Optimization (SCWE-082)

Severity
Gas

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

Affected Code

- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L222
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L289

Impacts
Having a large number literals in the code increases the gas usage of the contract during its deployment and when the functions are used or called from the contract.
It also makes the code harder to read and audit and increases the chances of introducing code errors.

Remediation
Scientific notation in the form of 2e10 is also supported, where the mantissa can be fractional, but the exponent has to be an integer. The literal MeE is equivalent to M * 10**E. Examples include 2e10, 2e10, 2e-10, 2.5e1, as suggested in official solidity documentation. https://docs.soliditylang.org/en/latest/types.html#rational-and-integer-literals
It is recommended to use numbers in the form "35 * 1e7 * 1e18" or "35 * 1e25".
The numbers can also be represented by using underscores between them to make them more readable such as "35_00_00_000"

Retest
Client's comment: Acknowledged.

Bug ID #G003 [Won't Fix]

Cheaper Inequalities in if()

Vulnerability Type
Gas Optimization (SCWE-082)

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/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/VotersDistributor.sol#L68
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/VotersDistributor.sol#L111
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/VotersDistributor.sol#L116
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/VotersDistributor.sol#L117
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/VotersDistributor.sol#L118
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/Vouch.sol#L175
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/LiquidityManager.sol#L191
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/LiquidityManager.sol#L198
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L420
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L571
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L578

- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L581
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L897
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L902
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L917
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L929
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L941
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L965
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L969
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L993
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L997
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L1021
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L1025
-

Impacts
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
Client’s comment: Acknowledged.

Bug ID #G004 [Won't Fix]

Cheaper Inequalities in require()

Vulnerability Type
Gas Optimization (SCWE-082)

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://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/LPLocker.sol#L64
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/LPLocker.sol#L120
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/LPLocker.sol#L204
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/LPLocker.sol#L219
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/Utilities.sol#L181
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/Utilities.sol#L191
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/Vouch.sol#L202
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/Vouch.sol#L307
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/Vouch.sol#L336
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/Vouch.sol#L337
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/LiquidityManager.sol#L144
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/LiquidityManager.sol#L145

- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L289

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: Acknowledged.

Bug ID #G005 [Won't Fix]

Cheaper conditional operators

Vulnerability Type
Gas Optimization (SCWE-082)

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://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/LPLocker.sol#L204
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/LPLocker.sol#L225
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/LPLocker.sol#L226
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/LPLocker.sol#L227
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/LPLocker.sol#L228
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/LPLocker.sol#L229
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/LPLocker.sol#L230
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/Utilities.sol#L214
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/VotersDistributor.sol#L56
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/VotersDistributor.sol#L57
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/VotersDistributor.sol#L58

- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/Vouch.sol#L148
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/Vouch.sol#L175
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/LiquidityManager.sol#L168
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/LiquidityManager.sol#L170
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/LiquidityManager.sol#L175
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/LiquidityManager.sol#L177
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/LiquidityManager.sol#L191
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/LiquidityManager.sol#L193
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/LiquidityManager.sol#L198
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/LiquidityManager.sol#L200
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L540
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L954
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L980
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L1008
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L571
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L574
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L578
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L581
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L897
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L902
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L904

- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L929
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L941
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L965
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L969
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L993
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L997
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L1021
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L1025

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

Client's comment: Acknowledged.

Bug ID #G006 [Won't Fix]

Gas Optimization for State Variables

Vulnerability Type
Gas Optimization (SCWE-082)

Severity
Gas

Description
Plus equals (+=) costs more gas than the addition operator. The same thing happens with minus equals (-=). Therefore, x +=y costs more gas than x = x + y.

Affected Code

- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/LPLocker.sol#208

Impacts
Writing the arithmetic operations in x = x + y format will save some gas.

Remediation
It is suggested to use the format x = x + y in all the instances mentioned above.

Retest
Client's comment: Acknowledged.

Bug ID #G007 [Won't Fix]

Gas Optimization in Require/Revert Statements

Vulnerability Type
Gas Optimization (SCWE-082)

Severity
Gas

Description
The **require/revert** statement takes an input string to show errors if the validation fails.
The strings inside these functions that are longer than **32 bytes** require at least one additional MSTORE, along with additional overhead for computing memory offset and other parameters. For this purpose, having strings lesser than 32 bytes saves a significant amount of gas.

Affected Code

- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/LPLocker.sol#L64
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/LPLocker.sol#L206
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/LPLocker.sol#L255
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/Utilities.sol#L203
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/VotersDistributor.sol#L56
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/VotersDistributor.sol#L57
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/VotersDistributor.sol#L58
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/LiquidityManager.sol#L141
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/LiquidityManager.sol#L147
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/LiquidityManager.sol#L153

- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L303
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L321
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L338
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L373
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L955
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L981
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L1009
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L959
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L985
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L1013

Impacts
Having longer require/revert strings than **32 bytes** cost a significant amount of gas.

Remediation
It is recommended to shorten the strings passed inside **require/revert** statements to fit under **32 bytes**. This will decrease the gas usage at the time of deployment and at runtime when the validation condition is met.

Retest
Client's comment: Acknowledged.

Bug ID #G008 [Won't Fix]

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-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/Utilities.sol#L346
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/VotersDistributor.sol#L94
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/VotersDistributor.sol#L141
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/Vouch.sol#L213
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/Vouch.sol#L22
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/Vouch.sol#L234
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L367

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
Client's comment: Acknowledged.

Bug ID #G009 [Won't Fix]

Public constants can be private

Vulnerability Type
Gas Optimization (SCWE-082)

Severity
Gas

Description
Public constant variables cost more gas because the EVM automatically creates getter functions for them and adds entries to the method ID table. The values can be read from the source code instead.

Affected Code

- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/Utilities.sol#L333
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/Utilities.sol#L334C
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/Utilities.sol#L335
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/Vouch.sol#L13

Impacts
Public constants are more costly due to the default getter functions created for them, increasing the overall gas cost.

Remediation
If reading the values for the constants is not necessary, consider changing the public visibility to private.

Retest
Client's comment: Acknowledged.

Bug ID #G010 [Won't Fix]

State Variable Can Be Marked As Constants

Vulnerability Type
Gas Optimization (SCWE-082)

Severity
Gas

Description
The contract has defined state variables whose values are never modified throughout the contract. The variables whose values never change should be marked as constant to save **gas**.

Affected Code

- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/LPLocker.sol#L15
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/LPLocker.sol#L16
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/LPLocker.sol#L17
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/LPLocker.sol#L19
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/LPLocker.sol#L30
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/Vouch.sol#L46
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/LiquidityManager.sol#L18
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/LiquidityManager.sol#L19
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/LiquidityManager.sol#L20
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/LiquidityManager.sol#L21
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/LiquidityManager.sol#L22
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/LiquidityManager.sol#L25

- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/LiquidityManager.sol#L26
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L19
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L34
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L35
- https://github.com/Vouchrun/VOUCH-token/blob/d4cf5a1a3e844d00cc0715ade1143d10e8324633/contracts/DaoDistributor.sol#L37

Impacts
Not marking unchanging state variables as constant in the contract can waste gas.

Remediation
Make sure that the values stored in the variables flagged above do not change throughout the contract. If this is the case, then consider setting these variables as **constant**.

Retest
Client's comment: Acknowledged.

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