Audited by CRED SHIELDS

CredShields

Smart Contract Audit

April 15th, 2024 • CONFIDENTIAL

Description

This document details the process and result of the Smart Contract audit performed by CredShields Technologies PTE. LTD. on behalf of Vouch between September 18th, 2024, and September 25th, 2024. A retest was performed on September 27th, 2024.

Author

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

Reviewers

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

Prepared for

Vouch

Table of Contents

1. Executive Summary ——————————- **3**

State of Security 4

2. The Methodology ——————————- **5**

2.1 Preparation phase 5

2.1.1 Scope 5

2.1.2 Documentation 5

2.1.3 Audit Goals 6

2.2 Retesting phase 6

2.3 Vulnerability classification and severity 6

2.4 CredShields staff 8

3. Findings Summary ——————————- **9**

3.1 Findings Overview 9

3.1.1 Vulnerability Summary 9

3.1.2 Findings Summary 11

4. Remediation Status ——————————- **14**

5. Bug Reports —————————— **16**

Bug ID #1 [Fixed] 16

Timelock contract can be made unusable 16

Bug ID #2 [Fixed] 17

Reorg Attack in factory 17

Bug ID #3 [Fixed] 19

No function to change submitBalancesEnabled value 19

Bug ID #4 [Fixed] 20

Missing Validation for _block variable in submitBalances() 20

Bug ID #5 [Fixed] 21

Missing Zero Address Validations 21

Bug ID #6 [Won't fix] 22

Missing Events in Important Functions 22

Bug ID #7 [Won't fix] 26

Outdated Pragma 26

Bug ID #8 [Won't fix] 28

Dead Code 28

Bug ID #9 [Won't fix] 30

Cheaper Inequalities in if() 30

Bug ID #10 [Fixed] 32

Cheaper Conditional Operators 32

Bug ID #11 [Fixed] 34
Custom error to save gas 34
Bug ID #12 [Fixed] 35
Gas Optimization for State Variables 35
6. The Disclosure ——————————- 36

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

Vouch engaged CredShields to perform a smart contract audit from September 18th, 2024, to September 25th, 2024. During this timeframe, 12 vulnerabilities were identified. **A retest was performed on September 27th, 2024, and all the bugs have been addressed.**

During the audit, 1 vulnerabilities were found with a severity rating of either High or Critical. These vulnerabilities represent the greatest immediate risk to "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
LSD Contract	0	1	1	5	1	4	**12**
	0	**1**	**1**	**5**	**1**	**4**	**12**

Table: Vulnerabilities Per Asset in Scope

The CredShields team conducted the security audit to focus on identifying vulnerabilities in the LSD 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 an LSD Contract 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 September 18th, 2024, to September 25th, 2024, was agreed upon during the preparation phase.

2.1.1 Scope

During the preparation phase, the following scope for the engagement was agreed upon:

IN SCOPE ASSETS
https://github.com/Vouchrun/pls-lsd-contracts/tree/741acbcc9e35fe2916667c6128d9f722fcb2d60b

2.1.2 Documentation

The following documentations were referred to during the audit as provided by the Vouch's team:

- https://vouch.run/docs/introduction/intro_to_LSD.html
- https://lsaas-docs.stafi.io/docs/developethlsd/contract.html

2.1.3 Audit Goals

CredShields uses both in-house tools and manual methods for comprehensive smart contract security auditing. The majority of the audit is done by manually reviewing the contract source code, following SWC registry standards, and an extended industry standard self-developed checklist. The team places emphasis on understanding core concepts, preparing test cases, and evaluating business logic for potential vulnerabilities.

2.2 Retesting phase

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 the asset and SWC classification. Each asset section will include a summary. The table in the executive summary contains the total number of identified security vulnerabilities per asset per risk indication.

3.1 Findings Overview

3.1.1 Vulnerability Summary

During the security assessment, 12 security vulnerabilities were identified in the asset.

VULNERABILITY TITLE	SEVERITY	SWC | Vulnerability Type
Timelock contract can be made unusable	High	Missing Input Validation
Reorg Attack in factory	Medium	Blockchain Reorg
No function to change submitBalancesEnabled value	Low	Business Logic Issue
Missing Validation for _block variable in submitBalances()	Low	Missing Input Validation
Missing Zero Address Validations	Low	Missing Input Validation
Missing Events in Important Functions	Low	Missing Best Practices
Outdated Pragma	Low	Outdated Pragma (SWC-102)
Dead Code	Informational	Code With No Effects (SWC-135)

Cheaper Inequalities in if()	Gas	Gas Optimization
Cheaper Conditional Operators	Gas	Gas Optimization
Custom error to save gas	Gas	Gas Optimization
Gas Optimization for State Variables	Gas	Gas Optimization

Table: Findings in Smart Contracts

3.1.2 Findings Summary

SWC ID	SWC Checklist	Test Result	Notes
SWC-100	Function Default Visibility	Not Vulnerable	Not applicable after v0.5.X (Currently using solidity v >= 0.8.6)
SWC-101	Integer Overflow and Underflow	Not Vulnerable	The issue persists in versions before v0.8.X.
SWC-102	Outdated Compiler Version	Not Vulnerable	Bug ID #7
SWC-103	Floating Pragma	Not Vulnerable	Contract was not using a floating pragma
SWC-104	Unchecked Call Return Value	Not Vulnerable	call() is not used
SWC-105	Unprotected Ether Withdrawal	Not Vulnerable	Appropriate function modifiers and require validations are used on sensitive functions that allow token or ether withdrawal.
SWC-106	Unprotected SELFDESTRUCT Instruction	Not Vulnerable	selfdestruct() is not used anywhere
SWC-107	Reentrancy	Not Vulnerable	No notable functions were vulnerable to it.
SWC-108	State Variable Default Visibility	Not Vulnerable	Not Vulnerable
SWC-109	Uninitialized Storage Pointer	Not Vulnerable	Not vulnerable after compiler version, v0.5.0
SWC-110	Assert Violation	Not Vulnerable	Asserts are not in use.
SWC-111	Use of Deprecated Solidity Functions	Not Vulnerable	None of the deprecated functions like block.blockhash(), msg.gas, throw, sha3(), callcode(), suicide() are in use

SWC-112	Delegatecall to Untrusted Callee	Not Vulnerable	Not Vulnerable.
SWC-113	DoS with Failed Call	Not Vulnerable	No such function was found.
SWC-114	Transaction Order Dependence	Not Vulnerable	Not Vulnerable.
SWC-115	Authorization through tx.origin	Not Vulnerable	tx.origin is not used anywhere in the code
SWC-116	Block values as a proxy for time	Not Vulnerable	Block.timestamp is not used
SWC-117	Signature Malleability	Not Vulnerable	Not used anywhere
SWC-118	Incorrect Constructor Name	Not Vulnerable	All the constructors are created using the constructor keyword rather than functions.
SWC-119	Shadowing State Variables	Not Vulnerable	Not applicable as this won't work during compile time after version 0.6.0
SWC-120	Weak Sources of Randomness from Chain Attributes	Not Vulnerable	Random generators are not used.
SWC-121	Missing Protection against Signature Replay Attacks	Not Vulnerable	No such scenario was found
SWC-122	Lack of Proper Signature Verification	Not Vulnerable	Not used anywhere
SWC-123	Requirement Violation	Not Vulnerable	Not vulnerable
SWC-124	Write to Arbitrary Storage Location	Not Vulnerable	No such scenario was found
SWC-125	Incorrect Inheritance Order	Not Vulnerable	No such scenario was found
SWC-126	Insufficient Gas Griefing	Not Vulnerable	No such scenario was found
SWC-127	Arbitrary Jump with Function Type Variable	Not Vulnerable	Jump is not used.

SWC-128	DoS With Block Gas Limit	Not Vulnerable	Not Vulnerable.
SWC-129	Typographical Error	Not Vulnerable	No such scenario was found
SWC-130	Right-To-Left-Override control character (U+202E)	Not Vulnerable	No such scenario was found
SWC-131	Presence of unused variables	Not Vulnerable	No such scenario was found
SWC-132	Unexpected Ether balance	Not Vulnerable	No such scenario was found
SWC-133	Hash Collisions With Multiple Variable Length Arguments	Not Vulnerable	abi.encodePacked() or other functions are not used.
SWC-134	Message call with hardcoded gas amount	Not Vulnerable	Not used anywhere in the code
SWC-135	Code With No Effects	Not Vulnerable	Bug ID #8
SWC-136	Unencrypted Private Data On-Chain	Not Vulnerable	No such scenario was found

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

Vouch is actively partnering with CredShields from this engagement to validate the discovered vulnerabilities' remediations. **A retest was performed on September 27th, 2024, and all the issues have been addressed.**

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

VULNERABILITY TITLE	SEVERITY	REMEDIATION STATUS
Timelock contract can be made unusable	High	**Fixed** [Sep 27th, 2024]
Reorg Attack in factory	Medium	**Fixed** [Sep 27th, 2024]
No function to change submitBalancesEnabled value	Low	**Fixed** [Sep 27th, 2024]
Missing Validation for _block variable in submitBalances()	Low	**Fixed** [Sep 27th, 2024]
Missing Zero Address Validations	Low	**Fixed** [Sep 27th, 2024]
Missing Events in Important Functions	Low	**Won't Fix** [Sep 27th, 2024]
Outdated Pragma	Low	**Won't Fix** [Sep 27th, 2024]
Dead Code	Informational	**Won't Fix** [Sep 27th, 2024]
Cheaper Inequalities in if()	Gas	**Won't Fix** [Sep 27th, 2024]
Cheaper Conditional Operators	Gas	**Fixed** [Sep 27th, 2024]
Custom error to save gas	Gas	**Fixed** [Sep 27th, 2024]

Gas Optimization for State Variables	Gas	**Fixed** [Sep 27th, 2024]

Table: Summary of findings and status of remediation

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

Bug ID #1 [Fixed]

Timelock contract can be made unusable

Vulnerability Type
Missing Input Validation

Severity
High

Description
The createLsdNetworkWithTimelock() function in the LsdNetworkFactory.sol contract lacks proper validation for the _minDelay parameter. This parameter determines the minimum delay required for executing administrative actions within the network. Without constraints, a whitelisted user can accidentally set _minDelay to an excessively high value, such as type(uint256).max, effectively locking the network's administrative functions. This oversight allows the creation of a timelock contract with a timelock that is impractically long, preventing any timely updates or changes to the configuration.

Affected Code
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/LsdNetworkFactory.sol#L211-L223

Impacts
By setting an extremely high _minDelay, a whitelisted user can render the network contract unusable for all participants. This means that any administrative actions, such as updating critical parameters or replacing addresses, would be delayed for an impractically long period.

Remediation
To mitigate this vulnerability, it is essential to implement input validation for the _minDelay parameter. The contract should enforce a reasonable upper limit on _minDelay, ensuring it remains within a practical range.
For example, use _maxDealy variable to validate the value.

Retest
This issue has been fixed by setting a max delay of 30 days.

Bug ID #2 [Fixed]

Reorg Attack in factory

Vulnerability Type
Blockchain Reorg

Severity
Medium

Description
The functions in LsdNetworkFactory are used to deploy new contracts using the new opcode which uses create opcode. This mechanism is vulnerable to a reorg attack, a type of attack that occurs during blockchain reorganization events. During a reorg, the blockchain network can temporarily reorganize its blocks, replacing old blocks with new ones that are consistent with network consensus.
In the event of a reorg, an attacker can exploit this mechanism to create a contract with the same address to which another user has already transferred funds. This is especially relevant for ethereum based blockchains, which have been observed to experience significant reorgs. Optimistic rollups such as Optimism and Arbitrum are also prone to reorgs, particularly when fraud proofs are discovered, leading to reverted blocks.

Attack Scenario:

1. Alice deploys a contrcat using the functions in LsdNetworkFactory and then sends funds to the contract.
2. Bob (another whitelister or if onlyCreationWhitelister set to false), observing that the network block is in the process of reorganization, quickly calls the same function using the same contract creation code and target address.
3. The reorg causes Alice's original contract deployment to be replaced by Bob's transaction, allowing Bob to control the contract with the same address that Alice later transfers funds.
4. Alice's transactions go through, but the contract is now controlled by Bob, leading to the transfer of funds to Bob's controlled contract.

Affected Code

- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/LsdNetworkFactory.sol#L206
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/LsdNetworkFactory.sol#L219-L220
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/LsdNetworkFactory.sol#L233

- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/LsdNetworkFactory.sol#L341
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/LsdNetworkFactory.sol#L345-L350

Impacts
Users may unintentionally transfer funds to contracts controlled by malicious users due to reorg-based manipulation which will lead to loss of funds.

Remediation
It is recommended to use create2 to ensure deterministic contract creation. The create2 opcode generates the contract address based on the deployer's address, a salt, and the bytecode. Including msg.sender as part of the salt ensures that an attacker cannot easily predict or duplicate contract addresses.

Retest
This vulnerability has been fixed by using salt & msg.sender while generating the address.

Bug ID #3 [Fixed]

No function to change submitBalancesEnabled value

Vulnerability Type
Business Logic Issue

Severity
Low

Description
The submitBalances() function in the NetworkBalances contract includes a check for the submitBalancesEnabled flag to determine if balance submissions are allowed. This flag is initialized to true in the init() function, but there is no mechanism provided to change its state to false. As a result, the check for submitBalancesEnabled is effectively redundant, as it cannot be toggled to restrict balance submissions.

Affected Code
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/NetworkBalances.sol#L108-L110

Impacts
The inability to disable balance submissions could lead to operational inefficiencies and a lack of control over the submission process.

Remediation
Introduce a function that allows an authorized actor, such as an admin, to toggle the submitBalancesEnabled flag.

Retest
This issue has been fixed by introducing the new function setSubmitBalancesEnabled().

Bug ID #4 [Fixed]

Missing Validation for _block variable in submitBalances()

Vulnerability Type
Missing Input Validation

Severity
Low

Description
The submitBalances() function in the NetworkBalances contract does not adequately validate the _block parameter to ensure it is within a reasonable range relative to the current Ethereum network block number. Although the function checks that the submitted block number is greater than the last recorded block number, it does not prevent the submission of an excessively high block number. This lack of validation could allow to disrupt the expected sequence of balance updates by submitting a block number that is far ahead of the current network block.

Affected Code
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/NetworkBalances.sol#L103-L124

Impacts
Submitting an excessively high block number could lead to significant discrepancies in the network's balance records, potentially causing confusion and errors in balance-related calculations.

Remediation
It is recommended to add a validation for _block or use an incremental _block variable.

Retest
This issue has been fixed by adding validation for _block.

Bug ID #5 [Fixed]

Missing Zero Address Validations

Vulnerability Type
Missing Input Validation

Severity
Low

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

Affected Variables and Line Numbers

- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/LsdNetworkFactory.sol#L199-L209
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/LsdNetworkFactory.sol#L211-L223
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/LsdNetworkFactory.sol#L225-L237

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 by adding a zero address validation.

Bug ID #6 [Won't fix]

Missing Events in Important Functions

Vulnerability Type
Missing Best Practices

Severity
Low

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

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

Affected Code
The following functions were affected -

- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/NetworkWithdraw.sol#L67-L88
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/NetworkWithdraw.sol#L135-L137
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/NetworkWithdraw.sol#L139-L147
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/NetworkWithdraw.sol#L149-L154
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/NetworkWithdraw.sol#L156-L165
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/NetworkWithdraw.sol#L421-L433
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/NetworkWithdraw.sol#L435-L448
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/LsdToken.sol#L25-L31

- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/LsdToken.sol#L35-L49
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/NodeDeposit.sol#L47-L61
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/NodeDeposit.sol#L103-L105
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/NodeDeposit.sol#L107-L109
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/NodeDeposit.sol#L111-L119
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/NodeDeposit.sol#L121-L123
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/NodeDeposit.sol#L125-L127
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/NodeDeposit.sol#L129-L140
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/NodeDeposit.sol#L142-L145
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/FeePool.sol#L26-L34
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/NetworkProposal.sol#L40-L62
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/NetworkProposal.sol#L101-L107
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/NetworkProposal.sol#L109-L115
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/NetworkProposal.sol#L127-L129
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/NetworkProposal.sol#L131-L148
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/NetworkProposal.sol#L150-L158
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/NetworkProposal.sol#L160-L168
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/NetworkProposal.sol#L170-L186
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/LsdNetworkFactory.sol#L56-L77
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/LsdNetworkFactory.sol#L106-L112
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/LsdNetworkFactory.sol#L114-L116

- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/LsdNetworkFactory.sol#L118-L120
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/LsdNetworkFactory.sol#L122-L124
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/LsdNetworkFactory.sol#L126-L128
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/LsdNetworkFactory.sol#L130-L132
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/LsdNetworkFactory.sol#L134-L136
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/LsdNetworkFactory.sol#L138-L140
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/LsdNetworkFactory.sol#L142-L148
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/LsdNetworkFactory.sol#L154-L171
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/LsdNetworkFactory.sol#L173-L175
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/LsdNetworkFactory.sol#L177-L179
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/LsdNetworkFactory.sol#L181-L183
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/LsdNetworkFactory.sol#L185-L187
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/LsdNetworkFactory.sol#L189-L191
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/LsdNetworkFactory.sol#L193-L195
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/LsdNetworkFactory.sol#L340-L342
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/LsdNetworkFactory.sol#L344-L356
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/UserDeposit.sol#L33-L46
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/UserDeposit.sol#L72-L74
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/UserDeposit.sol#L76-L78
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/NetworkBalances.sol#L36-L41
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/NetworkBalances.sol#L88-L90

- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/NetworkBalances.sol#L92-L97

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: Considering contracts as a database and logic processor, it has exposed enough events to the external environment that DApps depend on.

Bug ID #7 [Won't fix]

Outdated Pragma

Vulnerability Type
Floating Pragma (SWC-103)

Severity
Low

Description
The smart contract is using an outdated version of the Solidity compiler specified by the pragma directive i.e. 0.8.19. 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/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/NetworkWithdraw.sol#L2
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/LsdToken.sol#L2
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/Timelock.sol#L2
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/NodeDeposit.sol#L2
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/FeePool.sol#L2
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/NetworkProposal.sol#L2
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/LsdNetworkFactory.sol#L2
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/UserDeposit.sol#L2

- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/NetworkBalances.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 0.8.25 pragma version.
Reference: https://swcregistry.io/docs/SWC-103

Retest
Client's Comment: We have seen the new versions of Solidity and no vulnerabilities have been found, for stability, we choose not to upgrade to the latest one right now, we will upgrade when it matures.

Bug ID #8 [Won't fix]

Dead Code

Vulnerability Type
Code With No Effects - SWC-135

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/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/FeePool.sol#L36
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/LsdNetworkFactory.sol#L79
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/NetworkBalances.sol#L43
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/NetworkProposal.sol#L64
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/NetworkWithdraw.sol#L90
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/NodeDeposit.sol#L63
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/UserDeposit.sol#L48

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

Client's Comment: We leave the _reinit() method as an entry point for initiating the state for future upgrades.

Bug ID #9 [Won't fix]

Cheaper Inequalities in if()

Vulnerability Type
Gas Optimization

Severity
Gas

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

Affected Code

- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/NetworkWithdraw.sol#L127
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/NetworkWithdraw.sol#L150
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/NetworkWithdraw.sol#L160
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/NetworkWithdraw.sol#L175
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/NetworkWithdraw.sol#L223
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/NetworkWithdraw.sol#L326
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/NetworkWithdraw.sol#L330
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/NetworkWithdraw.sol#L335
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/NetworkWithdraw.sol#L343
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/NetworkWithdraw.sol#L371
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/NodeDeposit.sol#L112

- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/NodeDeposit.sol#L115
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/NetworkProposal.sol#L46
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/NetworkProposal.sol#L132
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/NetworkProposal.sol#L171
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/LsdNetworkFactory.sol#L155
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/UserDeposit.sol#L86
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/UserDeposit.sol#L97
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/UserDeposit.sol#L117
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/NetworkBalances.sol#L93
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/NetworkBalances.sol#L121

Impacts
Using non-strict inequalities inside “if” 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: This trick may not significantly reduce gas, but it compromises code readability and potentially leads to unexpected behaviors, especially considering the threshold value.

Bug ID #10 [Fixed]

Cheaper Conditional Operators

Vulnerability Type
Gas Optimization

Severity
Gas

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

Affected Code

- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/NetworkWithdraw.sol#L175
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/NetworkWithdraw.sol#L223
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/NetworkWithdraw.sol#L310
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/NetworkWithdraw.sol#L343
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/NetworkWithdraw.sol#L371
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/NodeDeposit.sol#L180
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/UserDeposit.sol#L97

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

Remediation

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

Retest

This issue has been fixed as mentioned in the Remediation.

Bug ID #11 [Fixed]

Custom error to save gas

Vulnerability Type
Gas Optimization

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/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/NetworkWithdraw.sol#L283
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/NetworkWithdraw.sol#L317
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/NodeDeposit.sol#L306

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 as mentioned in the Remediation.

Bug ID #12 [Fixed]

Gas Optimization for State Variables

Vulnerability Type
Gas Optimization

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/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/NetworkWithdraw.sol#L408
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/NetworkWithdraw.sol#L441
- https://github.com/Vouchrun/pls-lsd-contracts/blob/741acbcc9e35fe2916667c6128d9f722fcb2d60b/contracts/LsdNetworkFactory.sol#L147

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
This issue has been fixed as mentioned in the Remediation.

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