Audited by CRED SHIELDS

CredShields

Smart Contract Audit

February 7th, 2025 • CONFIDENTIAL

Description

This document details the process and result of the Smart Contract audit performed by CredShields Technologies PTE. LTD. on behalf of w3.labs between January 29th, 2025, and February 3rd, 2025. A retest was performed on February 5th, 2025.

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

w3.labs

Table of Contents

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

w3.labs engaged CredShields to perform a smart contract audit from January 29th, 2025, to February 3rd, 2025. During this timeframe, 13 vulnerabilities were identified. **A retest was performed on February 5th, 2025, and all the bugs have been addressed.**

During the audit, 0 vulnerabilities were found with a severity rating of either High or Critical. These vulnerabilities represent the greatest immediate risk to "w3.labs" and should be prioritized for remediation, and fortunately, none were found.

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
w3.labs Staking Contracts	0	0	4	1	1	7	**13**
	0	**0**	**4**	**1**	**1**	**7**	**13**

Table: Vulnerabilities Per Asset in Scope

The CredShields team conducted the security audit to focus on identifying vulnerabilities in w3.labs Staking Contract's scope during the testing window while abiding by the policies set forth by w3.labs'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 w3.labs'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 w3.labs can implement both manual and automated procedures to eliminate entire classes of vulnerabilities in the future. By taking a proactive approach, w3.labs can future-proof its security posture and protect its assets.

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

w3.labs engaged CredShields to perform a Staking Contracts 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 January 29th, 2025, to February 3rd, 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/w3labsxyz/ethereum-staking-contracts/tree/efe4bb7a4d2d04332cbeebae74a27547d18fc62f

2.1.2 Documentation

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

2.1.3 Audit Goals

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

2.2 Retesting Phase

w3.labs 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, 13 security vulnerabilities were identified in the asset.

VULNERABILITY TITLE	SEVERITY	SWC | Vulnerability Type
Overflow leads to approval DoS	Medium	Denial of Service
Overflow leads to deposit DoS	Medium	Denial of Service
Overflow leads to unbound request DoS	Medium	Denial of Service
Reorg attack	Medium	Blockchain Reorg
Missing Events	Low	Missing Best Practices
Dead Code	Informational	Code With No Effects - SWC-135
Cheaper Conditional Operators	Gas	Gas Optimization
Unused Imports	Gas	Gas Optimization
Cheaper Inequalities in if()	Gas	Gas Optimization

Splitting Revert Statements	Gas	Gas Optimization
Gas Optimization in Increments	Gas	Gas Optimization
Gas Optimization for State Variables	Gas	Gas Optimization
Public Constants can be Private	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	Vulnerable	Bug IDs #1, #2, #3
SWC-102	Outdated Compiler Version	Not Vulnerable	Version 0^.8.0 and above is used
SWC-103	Floating Pragma	Not Vulnerable	Contract is not using 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	Vulnerable	Bug ID #6
SWC-136	Unencrypted Private Data On-Chain	Not Vulnerable	No such scenario was found

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

w3.labs is actively partnering with CredShields from this engagement to validate the discovered vulnerabilities' remediations. **A retest was performed on February 5th, 2025, and all the issues have been addressed.**

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

VULNERABILITY TITLE	SEVERITY	REMEDIATION STATUS
Overflow leads to approval DoS	Medium	**Fixed** [Feb 5, 2025]
Overflow leads to deposit DoS	Medium	**Fixed** [Feb 5, 2025]
Overflow leads to unbound request DoS	Medium	**Fixed** [Feb 5, 2025]
Reorg attack	Medium	**Fixed** [Feb 5, 2025]
Missing Events	Low	**Fixed** [Feb 5, 2025]
Dead Code	Informational	**Fixed** [Feb 5, 2025]
Cheaper Conditional Operators	Gas	**Fixed** [Feb 5, 2025]
Unused Imports	Gas	**Fixed** [Feb 5, 2025]
Cheaper Inequalities in if()	Gas	**Partially Fixed** [Feb 5, 2025]
Splitting Revert Statements	Gas	**Fixed** [Feb 5, 2025]
Gas Optimization in Increments	Gas	**Fixed** [Feb 5, 2025]

Gas Optimization for State Variables	Gas	**Fixed** [Feb 5, 2025]
Public Constants can be Private	Gas	**Partially Fixed** [Feb 5, 2025]

Table: Summary of findings and status of remediation

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

Bug ID #1 [Fixed]

Overflow leads to approval DoS

Vulnerability Type
Denial of Service

Severity
Medium

Description
The approveStakeQuota() function processes deposit approvals and updates the _depositDataCount variable, which is a uint16. The function enforces a check to prevent numberOfDepositData from exceeding type(uint16).max, but it does not account for cumulative additions over multiple calls. If the _depositDataCount approaches uint16.max over time, adding another valid deposit batch can cause an overflow, reverting the transaction and permanently preventing any further deposits. Since this function is required for deposit processing, reaching an overflow state will lead to a denial of service (DoS) for all future deposits.

Affected Code
- https://github.com/w3labsxyz/ethereum-staking-contracts/blob/efe4bb7a4d2d04332cbeebae74a27547d18fc62f/src/StakingVault.sol#L330

Impacts
Once _depositDataCount overflows, any attempt to approve new deposits will revert, making further deposits impossible.

Remediation
Change _depositDataCount to a larger type to prevent overflow. Alternatively, introduce a safeguard to check if _depositDataCount + numberOfDepositData exceeds type(uint16).max before updating the variable.

Retest
This issue has been resolved by updating to uint32.

Client's Comments: Ethereum currently has a total supply of around 120M Ether. Our vault now has a capacity for uint32, i.e., ~4B deposits or ~128B Ether. So the Ether supply would need to be 1000x for enough Ether to exist, then some single Entity would need that amount.

Bug ID #2 [Fixed]

Overflow leads to deposit DoS

Vulnerability Type
Denial of Service

Severity
Medium

Description
The receive() function handles direct ETH deposits into the staking vault and updates _numberOfDeposits, which is a uint16. The function calculates numberOfNewDeposits based on msg.value and adds it to _numberOfDeposits. However, there is no validation to check whether _numberOfDeposits + numberOfNewDeposits exceeds type(uint16).max. Over time, as deposits accumulate, _numberOfDeposits can overflow, leading to a revert and permanently blocking any further deposits. Since deposits are fundamental to the staking mechanism, this results in a denial of service (DoS) for all depositors.

Affected Code

- https://github.com/w3labsxyz/ethereum-staking-contracts/blob/efe4bb7a4d2d04332cbeebae74a27547d18fc62f/src/StakingVault.sol#L357-L357

Impacts
Once _numberOfDeposits overflows, any attempt to deposit ETH into the contract will fail, preventing depositors from staking further.

Remediation
Change _numberOfDeposits to a larger type to prevent overflow. Additionally, implement a validation check to ensure _numberOfDeposits + numberOfNewDeposits does not exceed type(uint16).max before updating the variable.

Retest
This issue has been resolved by updating to uint32.
Client's Comments: Ethereum currently has a total supply of around 120M Ether. Our vault now has a capacity for uint32, i.e., ~4B deposits or ~128B Ether. So the Ether supply would need to be 1000x for enough Ether to exist, then some single Entity would need that amount.

Bug ID #3 [Fixed]

Overflow leads to unbound request DoS

Vulnerability Type
Denial of Service

Severity
Medium

Description
The requestUnbondings() function processes unbonding requests and updates _numberOfUnbondings, which is a uint16. While the function includes a check to prevent numberOfUnbondings from exceeding type(uint16).max, it does not account for cumulative additions over multiple calls. If _numberOfUnbondings approaches uint16.max over time, adding another valid unbonding request can cause an overflow, reverting the transaction and permanently preventing further unbondings. Since unbonding is a critical operation for users to retrieve staked funds, this leads to a denial of service (DoS) scenario.

Affected Code
- https://github.com/w3labsxyz/ethereum-staking-contracts/blob/efe4bb7a4d2d04332cbeebae74a27547d18fc62f/src/StakingVault.sol#L429
- https://github.com/w3labsxyz/ethereum-staking-contracts/blob/efe4bb7a4d2d04332cbeebae74a27547d18fc62f/src/StakingVault.sol#L461

Impacts
Once _numberOfUnbondings overflows, any attempt to request new unbondings will revert, making it impossible for stakers to exit their positions.

Remediation
Change _numberOfUnbondings to a larger type to prevent overflow. Alternatively, introduce a safeguard to check if _numberOfUnbondings + numberOfUnbondings exceeds type(uint16).max before updating the variable.

Retest
This issue has been resolved by updating to uint32.
Client's Comments: Ethereum currently has a total supply of around 120M Ether. Our vault now has a capacity for uint32, i.e., ~4B deposits or ~128B Ether. So the Ether supply would need to be 1000x for enough Ether to exist, then some single Entity would need that amount.

Bug ID #4 [Fixed]

Reorg attack

Vulnerability Type
Blockchain Reorg

Severity
Medium

Description
The createVault() function in the contract is responsible for deploying new vaults using the new opcode, which internally uses the create opcode for contract creation. This mechanism is vulnerable to a **reorg attack**, a type of attack that exploits 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 particularly relevant for Ethereum-based blockchains, which experience occasional reorgs. Optimistic rollups such as Optimism and Arbitrum are also prone to reorgs, especially when fraud proofs are discovered, leading to reverted blocks.

Attack Scenario:

1. Alice calls createVault() and deploys a vault.
2. Before the transaction is finalized, she transfers funds to the vault.
3. A reorg occurs, reverting Alice's vault creation.
4. Bob, seeing the intended vault address, quickly calls createVault(), getting the same address in the reorged chain.
5. When Alice's fund transfer is reprocessed, it goes to Bob's vault instead.

Affected Code

- https://github.com/w3labsxyz/ethereum-staking-contracts/blob/efe4bb7a4d2d04332cbeebae74a27547d18fc62f/src/StakingHub.sol#L105

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 issue has been fixed.

Bug ID #5 [Fixed]

Missing Events

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

- https://github.com/w3labsxyz/ethereum-staking-contracts/blob/efe4bb7a4d2d04332cbeebae74a27547d18fc62f/src/StakingHub.sol#L188-L192
- https://github.com/w3labsxyz/ethereum-staking-contracts/blob/efe4bb7a4d2d04332cbeebae74a27547d18fc62f/src/StakingHub.sol#L195-L199
- https://github.com/w3labsxyz/ethereum-staking-contracts/blob/efe4bb7a4d2d04332cbeebae74a27547d18fc62f/src/StakingHub.sol#L202-L204
- https://github.com/w3labsxyz/ethereum-staking-contracts/blob/efe4bb7a4d2d04332cbeebae74a27547d18fc62f/src/StakingHub.sol#L207-L214
- https://github.com/w3labsxyz/ethereum-staking-contracts/blob/efe4bb7a4d2d04332cbeebae74a27547d18fc62f/src/StakingVault.sol#L241-L247
- https://github.com/w3labsxyz/ethereum-staking-contracts/blob/efe4bb7a4d2d04332cbeebae74a27547d18fc62f/src/StakingVault.sol#L261-L284
- https://github.com/w3labsxyz/ethereum-staking-contracts/blob/efe4bb7a4d2d04332cbeebae74a27547d18fc62f/src/StakingVault.sol#L341-L378
- https://github.com/w3labsxyz/ethereum-staking-contracts/blob/efe4bb7a4d2d04332cbeebae74a27547d18fc62f/src/StakingVault.sol#L391-L432
- https://github.com/w3labsxyz/ethereum-staking-contracts/blob/efe4bb7a4d2d04332cbeebae74a27547d18fc62f/src/StakingVault.sol#L679-L683

Impacts

Events are used to track the transactions off-chain and missing these events on critical functions makes it difficult to audit these logs if they're needed at a later stage.

Remediation
Consider emitting events for important functions to keep track of them.

Retest
This issue has been fixed.

Bug ID #6 [Fixed]

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/w3labsxyz/ethereum-staking-contracts/blob/efe4bb7a4d2d04332cbeebae74a27547d18fc62f/src/EIP7002.sol#L16
- https://github.com/w3labsxyz/ethereum-staking-contracts/blob/efe4bb7a4d2d04332cbeebae74a27547d18fc62f/src/EIP7002.sol#L20
- https://github.com/w3labsxyz/ethereum-staking-contracts/blob/efe4bb7a4d2d04332cbeebae74a27547d18fc62f/src/EIP7002.sol#L24
- https://github.com/w3labsxyz/ethereum-staking-contracts/blob/efe4bb7a4d2d04332cbeebae74a27547d18fc62f/src/BeaconChain.sol#L11

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

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

Retest
This issue has been fixed.

Bug ID #7 [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/w3labsxyz/ethereum-staking-contracts/blob/efe4bb7a4d2d04332cbeebae74a27547d18fc62f/src/StakingVault.sol#L224

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.

Bug ID #8 [Fixed]

Unused Imports

Vulnerability Type
Gas Optimization

Severity
Gas

Description
The contract EIP7002.sol was importing contract Math.sol which was not used anywhere in the code. This increases the gas cost and overall contract's complexity.

Affected Code

- https://github.com/w3labsxyz/ethereum-staking-contracts/blob/efe4bb7a4d2d04332cbeebae74a27547d18fc62f/src/EIP7002.sol#L4

Impacts
Unused imports in smart contracts can lead to an increase in the size of the code, making it more difficult to verify and potentially slowing down its execution. Moreover, having unused code in a smart contract can also increase the attack surface by potentially introducing vulnerabilities that can be exploited by malicious actors. This can lead to security issues and compromise the integrity of the contract.

Additionally, including unused imports in smart contracts can also increase deployment and gas costs, making it more expensive to deploy and run the contract on the Ethereum network.

Remediation
It is recommended to remove the import statement if the external contracts or libraries are not used anywhere in the contract.

Retest
This issue has been fixed.

Bug ID #9 [Partially Fixed]

Cheaper Inequalities in if()

Vulnerability Type
Gas Optimization

Severity
Gas

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

Affected Code

- https://github.com/w3labsxyz/ethereum-staking-contracts/blob/efe4bb7a4d2d04332cbeebae74a27547d18fc62f/src/BeaconChain.sol#L56
- https://github.com/w3labsxyz/ethereum-staking-contracts/blob/efe4bb7a4d2d04332cbeebae74a27547d18fc62f/src/StakingVault.sol#L202
- https://github.com/w3labsxyz/ethereum-staking-contracts/blob/efe4bb7a4d2d04332cbeebae74a27547d18fc62f/src/StakingVault.sol#L224
- https://github.com/w3labsxyz/ethereum-staking-contracts/blob/efe4bb7a4d2d04332cbeebae74a27547d18fc62f/src/StakingVault.sol#L346
- https://github.com/w3labsxyz/ethereum-staking-contracts/blob/efe4bb7a4d2d04332cbeebae74a27547d18fc62f/src/StakingVault.sol#L399
- https://github.com/w3labsxyz/ethereum-staking-contracts/blob/efe4bb7a4d2d04332cbeebae74a27547d18fc62f/src/StakingVault.sol#L472
- https://github.com/w3labsxyz/ethereum-staking-contracts/blob/efe4bb7a4d2d04332cbeebae74a27547d18fc62f/src/StakingVault.sol#L520
- https://github.com/w3labsxyz/ethereum-staking-contracts/blob/efe4bb7a4d2d04332cbeebae74a27547d18fc62f/src/StakingVault.sol#L589

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
This issue has been partially fixed.
Client Comment: one possibility realized here, the rest required strict inequalities.

Bug ID #10 [Fixed]

Splitting Revert Statements

Vulnerability Type
Gas Optimization

Severity
Gas

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

Affected Code

- https://github.com/w3labsxyz/ethereum-staking-contracts/blob/efe4bb7a4d2d04332cbeebae74a27547d18fc62f/src/StakingVault.sol#L242-L244
- https://github.com/w3labsxyz/ethereum-staking-contracts/blob/efe4bb7a4d2d04332cbeebae74a27547d18fc62f/src/StakingVault.sol#L393-L395
- https://github.com/w3labsxyz/ethereum-staking-contracts/blob/efe4bb7a4d2d04332cbeebae74a27547d18fc62f/src/StakingVault.sol#L443-L445
- https://github.com/w3labsxyz/ethereum-staking-contracts/blob/efe4bb7a4d2d04332cbeebae74a27547d18fc62f/src/StakingVault.sol#L590-L592

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

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

Retest
This issue has been fixed.

Bug ID #11 [Fixed]

Gas Optimization in Increments

Vulnerability Type
Gas optimization

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/w3labsxyz/ethereum-staking-contracts/blob/efe4bb7a4d2d04332cbeebae74a27547d18fc62f/src/StakingVault.sol#L273
- https://github.com/w3labsxyz/ethereum-staking-contracts/blob/efe4bb7a4d2d04332cbeebae74a27547d18fc62f/src/StakingVault.sol#L310
- https://github.com/w3labsxyz/ethereum-staking-contracts/blob/efe4bb7a4d2d04332cbeebae74a27547d18fc62f/src/StakingVault.sol#L360
- https://github.com/w3labsxyz/ethereum-staking-contracts/blob/efe4bb7a4d2d04332cbeebae74a27547d18fc62f/src/StakingVault.sol#L408
- https://github.com/w3labsxyz/ethereum-staking-contracts/blob/efe4bb7a4d2d04332cbeebae74a27547d18fc62f/src/StakingVault.sol#L449
- https://github.com/w3labsxyz/ethereum-staking-contracts/blob/efe4bb7a4d2d04332cbeebae74a27547d18fc62f/src/StakingVault.sol#L556
- https://github.com/w3labsxyz/ethereum-staking-contracts/blob/efe4bb7a4d2d04332cbeebae74a27547d18fc62f/src/StakingVault.sol#L601

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

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

Retest

This issue has been fixed.

Bug ID #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/w3labsxyz/ethereum-staking-contracts/blob/efe4bb7a4d2d04332cbeebae74a27547d18fc62f/src/StakingVault.sol#L330
- https://github.com/w3labsxyz/ethereum-staking-contracts/blob/efe4bb7a4d2d04332cbeebae74a27547d18fc62f/src/StakingVault.sol#L357
- https://github.com/w3labsxyz/ethereum-staking-contracts/blob/efe4bb7a4d2d04332cbeebae74a27547d18fc62f/src/StakingVault.sol#L358
- https://github.com/w3labsxyz/ethereum-staking-contracts/blob/efe4bb7a4d2d04332cbeebae74a27547d18fc62f/src/StakingVault.sol#L427
- https://github.com/w3labsxyz/ethereum-staking-contracts/blob/efe4bb7a4d2d04332cbeebae74a27547d18fc62f/src/StakingVault.sol#L459
- https://github.com/w3labsxyz/ethereum-staking-contracts/blob/efe4bb7a4d2d04332cbeebae74a27547d18fc62f/src/StakingVault.sol#L428
- https://github.com/w3labsxyz/ethereum-staking-contracts/blob/efe4bb7a4d2d04332cbeebae74a27547d18fc62f/src/StakingVault.sol#L429
- https://github.com/w3labsxyz/ethereum-staking-contracts/blob/efe4bb7a4d2d04332cbeebae74a27547d18fc62f/src/StakingVault.sol#L460
- https://github.com/w3labsxyz/ethereum-staking-contracts/blob/efe4bb7a4d2d04332cbeebae74a27547d18fc62f/src/StakingVault.sol#L461
- https://github.com/w3labsxyz/ethereum-staking-contracts/blob/efe4bb7a4d2d04332cbeebae74a27547d18fc62f/src/StakingVault.sol#L476
- https://github.com/w3labsxyz/ethereum-staking-contracts/blob/efe4bb7a4d2d04332cbeebae74a27547d18fc62f/src/StakingVault.sol#L488
- https://github.com/w3labsxyz/ethereum-staking-contracts/blob/efe4bb7a4d2d04332cbeebae74a27547d18fc62f/src/StakingVault.sol#L500

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.

Bug ID #13 [Partially Fixed]

Public Constants can be Private

Vulnerability Type
Gas Optimization

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/w3labsxyz/ethereum-staking-contracts/blob/efe4bb7a4d2d04332cbeebae74a27547d18fc62f/src/BeaconChain.sol#L11
- https://github.com/w3labsxyz/ethereum-staking-contracts/blob/efe4bb7a4d2d04332cbeebae74a27547d18fc62f/src/BeaconChain.sol#L20
- https://github.com/w3labsxyz/ethereum-staking-contracts/blob/efe4bb7a4d2d04332cbeebae74a27547d18fc62f/src/BeaconChain.sol#L21
- https://github.com/w3labsxyz/ethereum-staking-contracts/blob/efe4bb7a4d2d04332cbeebae74a27547d18fc62f/src/BeaconChain.sol#L24
- https://github.com/w3labsxyz/ethereum-staking-contracts/blob/efe4bb7a4d2d04332cbeebae74a27547d18fc62f/src/BeaconChain.sol#L27
- https://github.com/w3labsxyz/ethereum-staking-contracts/blob/efe4bb7a4d2d04332cbeebae74a27547d18fc62f/src/StakingHub.sol#L16
- https://github.com/w3labsxyz/ethereum-staking-contracts/blob/efe4bb7a4d2d04332cbeebae74a27547d18fc62f/src/StakingHub.sol#L19
- https://github.com/w3labsxyz/ethereum-staking-contracts/blob/efe4bb7a4d2d04332cbeebae74a27547d18fc62f/src/StakingVault.sol#L45
- https://github.com/w3labsxyz/ethereum-staking-contracts/blob/efe4bb7a4d2d04332cbeebae74a27547d18fc62f/src/StakingVault.sol#L48
- https://github.com/w3labsxyz/ethereum-staking-contracts/blob/efe4bb7a4d2d04332cbeebae74a27547d18fc62f/src/StakingVault.sol#L51

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
This issue has been partially fixed.
Client Comment: Turned public constants private (or internal, where necessary).

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