



CredShields

# Smart Contract Audit

February 11, 2026 • CONFIDENTIAL

## Description

This document details the process and result of the Smart Contract audit performed by CredShields Technologies PTE. LTD. on behalf of Mercuri Finance between January 30, 2026, and February 1, 2026. A retest was performed on February 10, 2026.

## Author

Shashank (Co-founder, CredShields) [shashank@CredShields.com](mailto:shashank@CredShields.com)

## Reviewers

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

## Prepared for

Mercuri Finance

# Table of Contents

<b>Table of Contents</b>	<b>2</b>
<b>1. Executive Summary -----</b>	<b>4</b>
State of Security	5
<b>2. The Methodology -----</b>	<b>6</b>
2.1 Preparation Phase	6
2.1.1 Scope	6
2.1.2 Documentation	6
2.1.3 Audit Goals	7
2.2 Retesting Phase	7
2.3 Vulnerability classification and severity	7
2.4 CredShields staff	9
<b>3. Findings Summary -----</b>	<b>10</b>
3.1 Findings Overview	10
3.1.1 Vulnerability Summary	10
<b>5. Bug Reports -----</b>	<b>12</b>
Bug ID #C001[Fixed]	12
Authorized Manager Or Owner Can Bypass Protocol Performance Fees Vault Owner	12
Bug ID #C002[Fixed]	14
Protocol Can Charge Performance Fees On Vault Owner's Principal Amount	14
Bug ID #C003[Fixed]	15
Unrestricted Liquidity Decrease Breaks Protocol Performance Fee Accounting	15
Bug ID #M001[Fixed]	16
Vault Can Revert Withdrawals Non Payable Owner	16
Bug ID #L001[Fixed]	17
Swap Router Can Force WETH Conversion Vault Owner	17
Bug ID #L002[Fixed]	18
Unapproved Manager Can Gain Operational Control Vault Owner	18
Bug ID #L003[Fixed]	19
Contract Owner Can Become Permanently Irrecoverable ManagerRegistry	19
Bug ID #L004[Fixed]	20
Swap Fee Upper Bound Not Enforced	20
Bug ID #L005[Fixed]	21
Floating and Outdated Pragma	21
Bug ID #I001[Fixed]	23
Authorized Manager Can Mint Position With Unbounded Slippage Vault Owner	23
Bug ID #I002[Fixed]	24



Misspelled Event Name Hinders Monitoring and Automation (ManagerApprovalUpdated)	24
Bug ID #I003 [Fixed]	25
Missing zero address validations	25
Bug ID #I004 [Fixed]	26
Use Ownable2Step	26
Bug ID #G001 [Fixed]	27
Cheaper conditional operators	27
<b>6. The Disclosure -----</b>	<b>29</b>



# 1. Executive Summary -----

Mercuri Finance engaged CredShields to perform a smart contract audit from January 30, 2026, to February 1, 2026. During this timeframe, 14 vulnerabilities were identified. A retest was performed on February 10, 2026, and all the bugs have been addressed.

During the audit, 3 vulnerabilities were found with a severity rating of either High or Critical. These vulnerabilities represent the greatest immediate risk to "Mercuri Finance" 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	$\Sigma$
Mercuri Protocol Contracts	3	0	1	5	4	1	<b>14</b>

*Table: Vulnerabilities Per Asset in Scope*

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



## 2. The Methodology -----

Mercuri Finance engaged CredShields to perform a Mercuri Protocol 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 30, 2026, to February 1, 2026, 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

###### **Audited Scope:**

<https://github.com/mercuri-finance/mercuri-protocol/tree/2cf6c35126fc9ce4e29d129b01c8db88c41f12c8>

###### **Retested Scope:**

<https://github.com/mercuri-finance/mercuri-protocol/tree/73376145277cf272c28ddcf9f9fa73313aabb4cf>



## 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

Mercuri Finance 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	<span style="color: yellow;">●</span> Medium	<span style="color: red;">●</span> High	<span style="color: darkred;">●</span> Critical
	MEDIUM	<span style="color: green;">●</span> Low	<span style="color: yellow;">●</span> Medium	<span style="color: red;">●</span> High
	LOW	<span style="color: grey;">●</span> None	<span style="color: green;">●</span> Low	<span style="color: yellow;">●</span> 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](mailto: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, 14 security vulnerabilities were identified in the asset.

Vulnerability Title	Severity	Vulnerability Type	Status
Authorized Manager Or Owner Can Bypass Protocol Performance Fees Vault Owner	Critical	<a href="#">Fee Bypass</a>	Fixed
Protocol Can Charge Performance Fees On Vault Owner's Principal Amount	Critical	<a href="#">Business Logic (SC03-LogicErrors)</a>	Fixed
Unrestricted Liquidity Decrease Breaks Protocol Performance Fee Accounting	Critical	<a href="#">Business Logic (SC03-LogicErrors)</a>	Fixed
Vault Can Revert Withdrawals Non Payable Owner	Medium	<a href="#">Denial of Service (SCWE-087)</a>	Fixed
Swap Router Can Force WETH Conversion Vault Owner	Low	<a href="#">Business Logic (SC03-LogicErrors)</a>	Fixed
Unapproved Manager Can Gain Operational Control Vault Owner	Low	<a href="#">Access Control (SCWE-016)</a>	Fixed
Contract Owner Can Become Permanently Irrecoverable ManagerRegistry	Low	<a href="#">Missing functionality (SCWE-006)</a>	Fixed
Swap Fee Upper Bound Not Enforced	Low	<a href="#">Lack of Input Validation (SC04-Lack Of Input Validation)</a>	Fixed
Floating and Outdated Pragma	Low	<a href="#">Floating Pragma (SCWE-060)</a>	Fixed



Authorized Manager Can Mint Position With Unbounded Slippage Vault Owner	Informational	<a href="#">Missing Slippage Protection</a>	Fixed
Misspelled Event Name Hinders Monitoring and Automation (ManagerApprovalUpdated)	Informational	<a href="#">Naming Inconsistency</a>	Fixed
Missing zero address validations	Informational	<a href="#">Missing Input Validation (SC04-Lack Of Input Validation)</a>	Fixed
Use Ownable2Step	Informational	<a href="#">Missing Best Practices</a>	Fixed
Cheaper conditional operators	Gas	<a href="#">Gas Optimization (SCWE-082)</a>	Fixed

*Table: Findings and Remediations*



## 5. Bug Reports -----

Bug ID #C001[Fixed]

### **Authorized Manager Or Owner Can Bypass Protocol Performance Fees Vault Owner**

#### **Vulnerability Type**

Fee Bypass

#### **Severity**

Critical

#### **Description**

The vault lifecycle is designed so that protocol performance fees are charged on swap fees before liquidity is fully removed, using a strict sequence enforced in `Vault.closePosition` and `Vault.withdrawAll`. Authorized actors are expected to collect swap fees while liquidity is active, apply performance fees via `_applyPerformanceFee`, then remove liquidity and collect principal. However, the public `Vault.collect` function can be called independently by any authorized manager or owner at any time. The root cause is that `Vault.collect` does not enforce liquidity state or fee application ordering, allowing it to be invoked after liquidity has been reduced to zero and before protocol fees are applied.

#### **Affected Code**

- [Vault.sol#L382C4-L394C6](#)

#### **Impacts**

User, acting as an authorized manager, calls `decreaseLiquidity` to reduce liquidity to zero, then calls `collect` to retrieve both principal and accrued fees into the vault balance, and finally triggers `emergencyWithdrawAll`. As a result, user successfully withdraws all assets without `_applyPerformanceFee` ever being executed, causing the protocol to lose its entitled performance fees.

#### **Remediation**

Restrict `collect` so it cannot be used to withdraw principal or bypass protocol fees, by making it internal-only and callable exclusively within fee-safe execution flows.

#### **Retest**

This issue has been fixed by removing the vulnerable functionality.



Bug ID #C002 [Fixed]

## Protocol Can Charge Performance Fees On Vault Owner's Principal Amount

### Vulnerability Type

Business Logic ([SC03-LogicErrors](#))

### Severity

Critical

### Description

The vault applies protocol performance fees through the internal `_applyPerformanceFee` function, which is intended to charge fees exclusively on swap fees generated by an active Uniswap V3 position. The intended flow assumes that swap fees have already been collected into the vault balance while liquidity is active, and that no principal is included in the fee base at that moment. However, `_applyPerformanceFee` calculates fees using the entire on-chain token balances of the vault via `IERC20(token).balanceOf(address(this))`. The root cause is the absence of accounting separation between collected swap fees and other token balances, such as idle principal not yet minted, principal returned from previous positions, or fresh deposits made after a position was minted.

### Affected Code

- [Vault.sol#L509C3-L527C1](#)

### Impacts

User deposit additional funds into the vault after an initial position is opened but before fees are applied. When `_applyPerformanceFee` is later executed, the protocol charges fees on user's newly deposited principal as if it were earned performance fees, resulting in direct and irreversible loss of user funds and incorrect protocol revenue attribution.

### Remediation

Track and apply performance fees only on the actual swap fees earned by the active position, excluding idle balances and user principal.

### Retest

This issue has been fixed by applying performance fees only on the actual swap fees earned by the active position.



Bug ID #C003 [Fixed]

## Unrestricted Liquidity Decrease Breaks Protocol Performance Fee Accounting

### Vulnerability Type

Business Logic ([SC03-LogicErrors](#))

### Severity

Critical

### Description

The vault exposes `decreaseLiquidity` to the owner or manager without enforcing the mandatory fee-settlement lifecycle, resulting in two concrete scenarios.

**(a) Full liquidity removal scenario:** the owner removes 100% liquidity via `decreaseLiquidity` before any fee collection; swap fees are never collected while liquidity is `active` and `_applyPerformanceFee` is never executed, so those fees are later withdrawn as principal, fully bypassing protocol performance fees.

**(b) Partial liquidity removal scenario:** the owner removes part of the liquidity; the removed portion of principal is moved into `tokensOwed`, and when `_collectFees` is later called while liquidity remains `active`, that principal is misclassified as swap fees and added to `accruedFees`, causing `_applyPerformanceFee` to charge protocol fees on principal. Both scenarios stem from the same root cause: liquidity can be decreased outside the enforced, order-dependent lifecycle that separates swap fees from principal and applies protocol fees correctly.

### Affected Code

- [Vault.sol#L359C4-L369C6](#)

### Impacts

Protocol performance fees can be bypassed or incorrectly charged on principal, leading to protocol revenue loss and incorrect user accounting.

### Remediation

It is suggested that enforcing mandatory swap fee collection and protocol fee application before any liquidity decrease.

### Retest

This issue has been [fixed](#).



Bug ID #M001[Fixed]

## Vault Can Revert Withdrawals Non Payable Owner

### Vulnerability Type

Denial of Service ([SCWE-087](#))

### Severity

Medium

### Description

The `_withdrawTokenOrETH` function is used to transfer all vault-held assets to the immutable owner address during normal and emergency withdrawal flows. When withdrawing `WETH`, the function forcibly unwraps `WETH` into native `ETH` and then transfers `ETH` to the owner using a low-level call. This design assumes that the owner address can always receive `ETH`. However, if the owner is a non-payable smart contract or a contract with restrictive fallback logic, the `ETH` transfer will revert. The root cause is the unconditional `WETH` unwrap and `ETH` transfer without providing an ERC20-based alternative path.

### Affected Code

- [Vault.sol#L274C1-L288C6](#)

### Impacts

If the vault owner is a non-payable contract, any withdrawal involving `WETH` will revert, permanently blocking asset recovery and causing a denial of service on withdrawals.

### Remediation

Introduce an explicit boolean configuration that allows the owner to choose whether withdrawals should unwrap `WETH` to `ETH` or transfer `WETH` directly.

### Retest

This issue has been fixed by following recommended remediation.



Bug ID #L001[Fixed]

## Swap Router Can Force WETH Conversion Vault Owner

### Vulnerability Type

Business Logic ([SC03-LogicErrors](#))

### Severity

Low

### Description

The receive function is designed to accept `ETH` only from trusted sources and immediately wrap it into `WETH` when sent by the swap router. This logic assumes that `ETH` sent by the router is always expected and relevant to vault operations. However, the vault may manage pools where neither `token0` nor `token1` is `WETH`. In such cases, receiving `ETH` from the router is unnecessary and inconsistent with the vault's asset model. The root cause is the absence of validation ensuring that `ETH` is only accepted and wrapped when `WETH` is actually one of the pool tokens.

### Affected Code

- [Vault.sol#L498C7-L500C10](#)

### Impacts

If the vault manages a `non-WETH` pool, the swap router can still send `ETH` to the vault, which is forcibly wrapped into `WETH` and held as an unsupported idle asset.

### Remediation

Restrict `ETH` acceptance from the swap router to vaults where both `token0` and `token1` is not `WETH`.

### Retest

This issue has been fixed by following recommended remediation.



Bug ID #L002 [Fixed]

## Unapproved Manager Can Gain Operational Control Vault Owner

### Vulnerability Type

Access Control ([SCWE-016](#))

### Severity

Low

### Description

The `VaultFactory.createVault` function is used by vault owners to deploy a new vault and assign an initial operational manager responsible for liquidity management actions. The design intent, as documented in `Vault.sol`, is that the initial manager is protocol-approved via a `ManagerRegistry`, while subsequent manager changes are at the owner's discretion and not `registry-enforced`. However, `VaultFactory.createVault` accepts an arbitrary manager address without validating it against the `ManagerRegistry`. The root cause is the absence of a registry approval check during vault creation, allowing any address to be set as the initial manager despite documented trust assumptions.

### Affected Code

- [VaultFactory.sol#L126](#)

### Impacts

Alice deploys a vault expecting protocol-approved automation, but mistakenly or maliciously assigns Bob as manager. Bob gains immediate authority to rebalance liquidity, collect fees, and close positions, potentially executing strategies that conflict with Alice's expectations and risk profile.

### Remediation

Enforce manager approval during vault creation by validating the provided manager against the `ManagerRegistry` before deploying the vault.

### Retest

This issue has been [fixed](#).



Bug ID #L003 [Fixed]

## Contract Owner Can Become Permanently Irrecoverable ManagerRegistry

### Vulnerability Type

Missing functionality ([SCWE-006](#))

### Severity

Low

### Description

The `ManagerRegistry` contract is designed to maintain a global list of protocol-approved managers, with privileged control restricted to the contract owner via the `onlyOwner` modifier. This ownership model assumes the owner can continue to administer approvals throughout the contract's lifetime. However, the contract does not implement any mechanism to transfer or update ownership after deployment. The root cause is the absence of an `updateOwner` or equivalent ownership transfer function, making the initially assigned owner immutable.

### Affected Code

- [ManagerRegistry.sol#L9](#)

### Impacts

If the owner's private key is lost, compromised, or the owning entity changes operational control, the protocol permanently loses the ability to approve or revoke managers, blocking governance actions and future protocol maintenance.

### Remediation

Add a controlled ownership transfer function to allow updating the registry `owner` to a new trusted address and it should be callable by `onlyOwner` modifier.

### Retest

This issue has been fixed.



Bug ID #L004 [Fixed]

## Swap Fee Upper Bound Not Enforced

### Vulnerability Type

Lack of Input Validation ([SC04-Lack Of Input Validation](#))

### Severity

Low

### Description

`setProtocolFees()` permits any `uint16 performanceFeeBps` without bounding it to a `BPS` maximum (typically 10,000). Comments state fees are interpreted in basis points, but there is no `require(performanceFeeBps <= 10_000)`. Downstream vault logic that assumes `BPS <= 10_000` may overcharge fees, confiscate the entire performance amount, or revert due to arithmetic assumptions.

### Affected Code

- [VaultFactory.sol#L94](#)

### Impacts

`Owner` can set an excessively high fee (e.g., 65,535 BPS). Depending on Vault math, this may either `confiscate >100%` of proceeds or cause operations to revert, effectively bricking fee collection or position closure across all vaults.

### Remediation

Enforce an upper bound consistent with BPS semantics.

### Retest

This issue has been fixed.



Bug ID #L005 [Fixed]

## Floating and Outdated Pragma

### Vulnerability Type

Floating Pragma ([SCWE-060](#))

### Severity

Low

### Description

Locking the pragma helps ensure that the contracts do not accidentally get deployed using an older version of the Solidity compiler affected by vulnerabilities.

The contract allowed floating or unlocked pragma to be used, i.e.,  $\geq 0.8.20$ . This allows the contracts to be compiled with all the solidity compiler versions above the limit specified. The following contracts were found to be affected -

### Affected Code

- [IERC20.sol#L2](#)
- [INonfungiblePositionManager.sol#L2](#)
- [ISwapRouter02.sol#L2](#)
- [IWETH.sol#L2](#)
- [ManagerRegistry.sol#L2](#)
- [SafeERC20.sol#L2](#)
- [FeeConfig.sol#L2](#)
- [Vault.sol#L2](#)
- [VaultFactory.sol#L2](#)

### Impacts

If the smart contract gets compiled and deployed with an older or too recent version of the solidity compiler, there's a chance that it may get compromised due to the bugs present in the older versions or unidentified exploits in the new versions.

Incompatibility issues may also arise if the contract code does not support features in other compiler versions, therefore, breaking the logic.

The likelihood of exploitation is low.

### Remediation

Keep the compiler versions consistent in all the smart contract files. Do not allow floating pragmas anywhere. It is suggested to use the 0.8.32 pragma version

Reference: <https://scs.owasp.org/SCWE/SCSVS-CODE/SCWE-060/>



**Retest**

This issue has been fixed.



Bug ID #I001[Fixed]

## Authorized Manager Can Mint Position With Unbounded Slippage Vault Owner

### Vulnerability Type

Missing Slippage Protection

### Severity

Informational

### Description

The `Vault.mintPosition` function is used by the owner or delegated manager to create a new Uniswap V3 liquidity position using the vault's token balances. The function validates pool parameters, fee tier, and recipient address, and relies on the caller-provided `MintParams` to control execution conditions. However, while `amount0Min` and `amount1Min` are part of the Uniswap V3 mint interface, `Vault.mintPosition` does not enforce any minimum values for these fields. The root cause is the absence of slippage sanity checks, allowing the position to be minted with zero minimum amounts.

### Affected Code

- [Vault.sol#L300](#)
- [Vault.sol#L440](#)
- [Vault.sol#L335](#)

### Impacts

If a manager submits mint parameters with `amount0Min == 0` and `amount1Min == 0`, the position can be minted under extreme price movement or sandwich conditions, resulting in significantly worse execution than expected and potential loss of user funds due to excessive slippage.

### Remediation

Enforce non-zero minimum amounts in `mintPosition` to ensure basic slippage protection during liquidity provisioning.

### Retest

The issue has been [fixed](#).



Bug ID #I002 [Fixed]

## **Misspelled Event Name Hinders Monitoring and Automation (ManagerApprovalUpdated)**

### **Vulnerability Type**

Naming Inconsistency

### **Severity**

Informational

### **Description**

The event name is declared as `ManagertApprovalUpdated` (note the trailing 't'), which is likely a typo for `ManagerApprovalUpdated`. Tooling and off-chain consumers may rely on canonical naming, and this misspelling can cause missed subscriptions or fragile adapters.

### **Affected Code**

- [ManagerRegistry.sol#L40](#)

### **Impacts**

Off-chain monitoring, alerting, or compliance automation may fail to detect approval changes, degrading observability and timely response to admin actions.

### **Remediation**

Correct the event name to `ManagerApprovalUpdated`. If maintaining backward compatibility on an already-deployed contract, add a second correctly named event and emit both, deprecating the misspelled one in client code.

### **Retest**

This issue has been fixed.



Bug ID #I003 [Fixed]

## Missing zero address validations

### Vulnerability Type

Missing Input Validation ([SC04-Lack Of Input Validation](#))

### Severity

Informational

### 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

- [VaultFactory.sol#L71](#)
- [Vault.sol#L166](#)
- [ManagerRegistry.sol#L38](#)

### Impacts

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

### Remediation

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

### Retest

This issue has been fixed.



Bug ID #I004 [Fixed]

## Use Ownable2Step

### Vulnerability Type

Missing Best Practices

### Severity

Informational

### Description

The "Ownable2Step" pattern is an improvement over the traditional "Ownable" pattern, designed to enhance the security of ownership transfer functionality in a smart contract. Unlike the original "Ownable" pattern, where ownership can be transferred directly to a specified address, the "Ownable2Step" pattern introduces an additional step in the ownership transfer process. Ownership transfer only completes when the proposed new owner explicitly accepts the ownership, mitigating the risk of accidental or unintended ownership transfers to mistyped addresses.

### Affected Code

- [VaultFactory.sol#L16](#)

### Impacts

Without the "Ownable2Step" pattern, the contract owner might inadvertently transfer ownership to an unintended or mistyped address, potentially leading to a loss of control over the contract. By adopting the "Ownable2Step" pattern, the smart contract becomes more resilient against external attacks aimed at seizing ownership or manipulating the contract's behavior.

### Remediation

It is recommended to use either Ownable2Step or Ownable2StepUpgradeable depending on the smart contract.

### Retest:

This issue has been fixed.



Bug ID #G001[Fixed]

## 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  $x \neq 0$  and  $x > 0$  interchangeably. However, it's important to note that during compilation,  $x \neq 0$  is generally more cost-effective than  $x > 0$  for unsigned integers within conditional statements.

### Affected Code

- [Vault.sol#L191](#)
- [Vault.sol#L198](#)
- [Vault.sol#L313](#)
- [Vault.sol#L346](#)
- [Vault.sol#L317](#)
- [Vault.sol#L350](#)
- [Vault.sol#L522](#)
- [Vault.sol#L523](#)
- [Vault.sol#L542](#)

### Impacts

Employing  $x \neq 0$  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  $x \neq 0$  conditional operator instead of  $x > 0$  for unsigned integer variables in conditional statements.

### Retest

The issue has been [fixed](#).





## 6. The Disclosure

---

The Reports provided by CredShields are not an endorsement or condemnation of any specific project or team and do not guarantee the security of any specific project. The contents of this report are not intended to be used to make decisions about buying or selling tokens, products, services, or any other assets and should not be interpreted as such.

Emerging technologies such as Smart Contracts and Solidity carry a high level of technical risk and uncertainty. CredShields does not provide any warranty or representation about the quality of code, the business model or the proprietors of any such business model, or the legal compliance of any business. The report is not intended to be used as investment advice and should not be relied upon as such.

CredShields Audit team is not responsible for any decisions or actions taken by any third party based on the report.



# Your **Secure Future** Starts Here



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

