CredShields

Smart Contract Audit

November 24, 2025 • NON-CONFIDENTIAL

Description

This document details the process and result of the Smart Contract audit performed by CredShields Technologies PTE. LTD. on behalf of Avail between November 3, 2025, and November 5, 2025. A retest was performed on November 21, 2025.

Author

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

Reviewers

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

Prepared for

Avail

Table of Contents

1. Executive Summary

Avail engaged CredShields to perform a smart contract audit from November 3, 2025, to November 5, 2025. During this timeframe, 7 vulnerabilities were identified. A retest was performed on November 21, 2025, 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 "Avail" 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
Vault Contracts	1	0	3	0	3	0	7

Table: Vulnerabilities Per Asset in Scope

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

2. The Methodology

Avail engaged CredShields to perform a Vault 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 November 3, 2025, to November 5, 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
Audited Commit: https://github.com/availproject/ca-sc/tree/c33f5221fdbe2435d01c11a430c48a9bd7854197

2.1.2 Documentation

Avail team shared the following documentation to help Credshields with the audit - https://arcananetwork.notion.site/Chain-Abstraction-Technical-Paper-1f4f11ed080480808d32cc1f11371b52

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

Avail is actively partnering with CredShields to validate the remediations implemented towards the discovered vulnerabilities.

2.3 Vulnerability classification and severity

CredShields follows OWASP's Risk Rating Methodology to determine the risk associated with discovered vulnerabilities. This approach considers two factors - Likelihood and Impact - which are evaluated with three possible values - **Low**, **Medium**, and **High**, based on factors such as Threat

agents, Vulnerability factors, and Technical and Business Impacts. The overall severity of the risk is calculated by combining the likelihood and impact estimates.

Overall Risk Severity				
Impact	HIGH	Medium	High	Critical
	MEDIUM	Low	Medium	High
	LOW	None	Low	Medium
		LOW	MEDIUM	HIGH
Likelihood				

Overall, the categories can be defined as described below -

1. **Informational**

   We prioritize technical excellence and pay attention to detail in our coding practices. Our guidelines, standards, and best practices help ensure software stability and reliability. Informational vulnerabilities are opportunities for improvement and do not pose a direct risk to the contract. Code maintainers should use their own judgment on whether to address them.

2. **Low**

   Low-risk vulnerabilities are those that either have a small impact or can't be exploited repeatedly or those the client considers insignificant based on their specific business circumstances.

3. **Medium**

   Medium-severity vulnerabilities are those caused by weak or flawed logic in the code and can lead to exfiltration or modification of private user information. These vulnerabilities

can harm the client's reputation under certain conditions and should be fixed within a specified timeframe.

4. **High**

   High-severity vulnerabilities pose a significant risk to the Smart Contract and the organization. They can result in the loss of funds for some users, may or may not require specific conditions, and are more complex to exploit. These vulnerabilities can harm the client's reputation and should be fixed immediately.

5. **Critical**

   Critical issues are directly exploitable bugs or security vulnerabilities that do not require specific conditions. They often result in the loss of funds and Ether from Smart Contracts or users and put sensitive user information at risk of compromise or modification. The client's reputation and financial stability will be severely impacted if these issues are not addressed immediately.

6. **Gas**

   To address the risk and volatility of smart contracts and the use of gas as a method of payment, CredShields has introduced a "Gas" severity category. This category deals with optimizing code and refactoring to conserve gas.

2.4 CredShields staff

The following individual at CredShields managed this engagement and produced this report:

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

Please feel free to contact this individual with any questions or concerns you have about the engagement or this document.

3. Findings Summary

This chapter contains the results of the security assessment. Findings are sorted by their severity and grouped by asset and OWASP SCWE classification. Each asset section includes a summary highlighting the key risks and observations. The table in the executive summary presents the total number of identified security vulnerabilities per asset, categorized by risk severity based on the OWASP Smart Contract Security Weakness Enumeration framework.

3.1 Findings Overview

3.1.1 Vulnerability Summary

During the security assessment, 7 security vulnerabilities were identified in the asset.

VULNERABILITY TITLE	SEVERITY	SCWE | Vulnerability Type
C001 — Anyone Can Mint Unlimited Token	Critical	Access Control (SCWE-016)
M001 — Solver Can Overpay And Lock Excess ETH	Medium	Business Logic (SC03-LogicErrors)
M002 — Admin Withdrawal Can Block ETH Settlement	Medium	Denial of Service (SCWE-087)
M003 — Fee-on-Transfer Tokens Cause Incorrect Deposit Accounting	Medium	Business Logic (SC03-LogicErrors)
G001 — Gas Optimization in Require/Revert Statements	Gas	Gas Optimization (SCWE-082)
G002 — Cheaper Inequalities in require()	Gas	Gas Optimization (SCWE-082)
G003 — Gas Optimization in Increments	Gas	Gas Optimization (SCWE-082)

Table: Findings in Smart Contracts

4. Remediation Status ——————————————————

Avail is actively partnering with CredShields from this engagement to validate the discovered vulnerabilities' remediations. A retest was performed on November 21, 2025, and all the issues have been addressed.

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

VULNERABILITY TITLE	SEVERITY	REMEDIATION STATUS
C001 — Anyone Can Mint Unlimited Tokens	Critical	Fixed [Nov 21, 2025]
M001 — Solver Can Overpay And Lock Excess ETH	Medium	Fixed [Nov 21, 2025]
M002 — Admin Withdrawal Can Block ETH Settlement	Medium	Fixed [Nov 21, 2025]
M003 — Fee-on-Transfer Tokens Cause Incorrect Deposit Accounting	Medium	Fixed [Nov 21, 2025]
G001 — Gas Optimization in Require/Revert Statements	Gas	Fixed [Nov 21, 2025]
G002 — Cheaper Inequalities in require()	Gas	Won't Fix [Nov 21, 2025]
G003 — Gas Optimization in Increments	Gas	Fixed [Nov 21, 2025]

Table: Summary of findings and status of remediation

5. Bug Reports

Bug ID #C001 [Fixed]

Anyone Can Mint Unlimited Token

Vulnerability Type
Access Control (SCWE-016)

Severity
Critical

Description
The mint(address to, uint256 amount) function is declared external and callable by any address. It directly calls _mint() without any role or ownership checks, allowing arbitrary users to create unlimited tokens. The root cause is the absence of access control on the mint function.

Affected Code

- https://github.com/availproject/ca-sc/blob/c33f5221fdbe2435d01c11a430c48a9bd7854197/contracts/USDC.sol#L8

Impacts
Any attacker can mint unlimited Token, destroying token integrity and collapsing its value.

Remediation
Restrict minting to an authorized role or owner.

Retest
The file has been removed entirely.

Bug ID #M001 [Fixed]

Solver Can Overpay And Lock Excess ETH

Vulnerability Type
Business Logic (SC03-LogicErrors)

Severity
Medium

Description
In the fulfil(...) function, solver sends ETH via msg.value, which initializes a local gasBalance variable. For each ETH transfer, the code deducts the transfer amount from gasBalance and enforces that gasBalance >= value. However, there is no final validation ensuring that the total ETH required equals msg.value, nor any logic to refund leftover ETH after all transfers.
As a result, if the total amount to distribute is less than msg.value, the excess remains in the contract. The root cause is the absence of a final reconciliation step between the provided ETH and the total ETH required for fulfilment.

Affected Code

- https://github.com/availproject/ca-sc/blob/c33f5221fdbe2435d01c11a430c48a9bd7854197/contracts/Vault.sol#L230

Impacts
Callers who overpay in msg.value, whether by error or through imprecise estimation lose the surplus ETH, which becomes trapped in the contract. Only an admin can later retrieve it via withdraw(), creating a loss scenario for solvers or users and potential operational friction.

Remediation
Refund any surplus ETH to the caller at the end of the function.

Retest
This issue has been fixed by adding refund logic.

Bug ID #M002 [Fixed]

Admin Withdrawal Can Block ETH Settlement

Vulnerability Type
Denial of Service (SCWE-087)

Severity
Medium

Description
The settle(...) function distributes ETH and ERC20 tokens to solvers as authorized by an admin signature. When distributing ETH, the function executes (bool sent,) = solver.call{value: amount}(""). This transfer relies on the contract holding a sufficient ETH balance. However, if an admin withdraws ETH from the contract before settlements occur, the contract may not have enough ETH to complete the distribution.
Additionally, the contract lacks a receive() or fallback() function, preventing anyone from replenishing ETH directly via a simple transfer. The root cause is the absence of a payable entry point for ETH replenishment combined with no mechanism to verify sufficient ETH balance before attempting transfers.

Affected Code

- https://github.com/availproject/ca-sc/blob/c33f5221fdbe2435d01c11a430c48a9bd7854197/contracts/Vault.sol#L307

Impacts
If admins withdraw ETH and the contract balance becomes insufficient, subsequent settle() calls involving ETH will revert, halting all settlements that require ETH payouts. This results in a full denial of service for ETH-based settlements and disrupts protocol operations until ETH is sent via deposit(...) function.

Remediation
Add a payable receive() function to allow ETH top-ups and optionally validate sufficient balance before settlement.

Retest
This issue has been fixed by removing withdraw logic.

Bug ID #M003 [Fixed]

Fee-on-Transfer Tokens Cause Incorrect Deposit Accounting

Vulnerability Type
Business Logic (SC03-LogicErrors)

Severity
Medium

Description
The deposit() function assumes all ERC20 tokens transfer the full requested amount. For fee-on-transfer tokens, the actual amount received by the Vault is less than request.sources[chainIndex].value, causing an incorrect deposit record. The contract sets depositNonce and requestState before verifying how many tokens were actually received, resulting in inaccurate accounting.

Affected Code
- https://github.com/availproject/ca-sc/blob/c33f5221fdbe2435d01c11a430c48a9bd7854197/contracts/Vault.sol#L175-L180

Impacts
Vault records deposits as successful even when it receives fewer tokens due to transfer fees. This creates fund mismatches, breaks accounting integrity, and may cause downstream payout or redemption failures.

Remediation
Check actual token balance before and after the transfer. Compare the difference with the expected amount and revert if mismatched, or record the actual received amount. Reject or explicitly handle fee-on-transfer tokens.

Retest
This issue is fixed by adding a mandatory check of the account balance before and after the transfer, which reverses the transfer if the amounts do not match.

Bug ID #G001 [Fixed]

Gas Optimization in Require/Revert Statements

Vulnerability Type
Gas Optimization (SCWE-082)

Severity
Gas

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

Affected Code

- https://github.com/availproject/ca-sc/blob/c33f5221fdbe2435d01c11a430c48a9bd7854197/contracts/Vault.sol#L291-L294
- https://github.com/availproject/ca-sc/blob/c33f5221fdbe2435d01c11a430c48a9bd7854197/contracts/Vault.sol#L296-L299

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

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

Retest
This issue has been fixed.

Bug ID #G002 [Won't Fix]

Cheaper Inequalities in require()

Vulnerability Type
Gas Optimization (SCWE-082)

Severity
Gas

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

Affected Code

- https://github.com/availproject/ca-sc/blob/c33f5221fdbe2435d01c11a430c48a9bd7854197/contracts/Vault.sol#L222-L225

Impacts
Using non-strict inequalities inside "require" statements costs more gas.

Remediation
It is recommended to go through the code logic, and, **if possible**, modify the non-strict inequalities with the strict ones to save gas as long as the logic of the code is not affected.

Retest
-

Bug ID#G003 [Fixed]

Gas Optimization in Increments

Vulnerability Type
Gas optimization (SCWE-082)

Severity
Gas

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

Affected Code

- https://github.com/availproject/ca-sc/blob/c33f5221fdbe2435d01c11a430c48a9bd7854197/contracts/Vault.sol#L187

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.

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

20A TANJONG PAGAR ROAD, SINGAPORE (088443)

info@credshields.com