Audited by CRED SHIELDS

CredShields

Smart Contract Audit

August 28, 2025 • 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 August 22nd, 2025, and August 25th, 2025. A retest was performed on August 26th, 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)

Prepared for

Avail

Table of Contents

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

Avail engaged CredShields to perform a smart contract audit from August 22nd, 2025, to August 25th, 2025. During this timeframe, 4 vulnerabilities were identified. **A retest was performed on August 26th, 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
XARMigration Contract	1	0	0	0	1	2	**4**
	1	**0**	**0**	**0**	**1**	**2**	**4**

Table: Vulnerabilities Per Asset in Scope

The CredShields team conducted the security audit to focus on identifying vulnerabilities in the XARMigration 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 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 August 22nd, 2025, to August 25th, 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/availproject/xar-migration/tree/dbeed79672bf17e8aaf8ea14e0be8a34ead4cfc0

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

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, 4 security vulnerabilities were identified in the asset.

VULNERABILITY TITLE	SEVERITY	SCWE | Vulnerability Type
Owner Can Drain User Migration Funds	Critical	Overpowered Owner
Rounding Error Due to Split Withdrawals Without Minimum Deposit Enforcement	Informational	Business Logic (SC03-LogicErrors)
Splitting Require/Revert Statements	Gas	Gas Optimization (SCWE-082)
Cheaper Inequalities in require()	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 August 26th, 2025, and all the issues have been addressed.**

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

VULNERABILITY TITLE	SEVERITY	REMEDIATION STATUS
Owner Can Drain User Migration Funds	Critical	Won't Fix [August 26, 2025]
Rounding Error Due to Split Withdrawals Without Minimum Deposit Enforcement	Informational	Won't Fix [August 26, 2025]
Splitting Require/Revert Statements	Gas	Won't Fix [August 26, 2025]
Cheaper Inequalities in require()	Gas	Won't Fix [August 26, 2025]

Table: Summary of findings and status of remediation

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

Bug ID #C001 [Won't Fix]

Owner Can Drain User Migration Funds

Vulnerability Type
Overpowered Owner

Severity
Critical

Description
The XARMigration contract is designed to safeguard user deposits of XAR and provide a time-locked conversion to AVAIL. However, the drain function allows the owner to arbitrarily transfer any ERC-20 tokens, including deposited XAR and locked AVAIL, to themselves. This creates a direct backdoor for governance to seize all user migration funds. The issue stems from the lack of restriction on which token can be drained, combined with unrestricted owner control.

Affected Code
- https://github.com/availproject/migration/blob/dbeed79672bf17e8aaf8ea14e0be8a34ead4cfc0/src/XARMigration.sol#L81-L83

Impacts
Users depositing XAR for migration may lose all their tokens if the owner invokes drain on either the XAR or AVAILcontracts. This fully compromises the trust model and defeats the purpose of a secure migration contract.

Remediation
Restrict drain to only allow recovery of unrelated tokens and explicitly disallow transferring XAR or AVAIL.

Eg:

```
-   function drain(IERC20 token, uint256 amount) external onlyOwner {
-       token.safeTransfer(msg.sender, amount);
-   }
+   function drain(IERC20 token, uint256 amount) external onlyOwner {
```

```
+       require(token != XAR && token != AVAIL, "cannot drain core assets");
+       token.safeTransfer(msg.sender, amount);
+   }
```

Retest

Client's Comments: This is a requirement. We want to be able to burn all XAR tokens and withdraw any unclaimed AVAIL tokens in the future.

Bug ID #I001 [Won't Fix]

Rounding Error Due to Split Withdrawals Without Minimum Deposit Enforcement

Vulnerability Type
Business Logic (SC03-LogicErrors)

Severity
Informational

Description
The withdraw function performs split payouts at six months and one year. Because each payout converts XAR to AVAILwith integer division separately, users can be underpaid compared to the expected floor(totalDeposited / XAR_PER_AVAIL).

This effect is most severe for small deposits, where rounding truncation can eliminate the payout entirely. For example:

- With 6 XAR deposited, and XAR_PER_AVAIL = 4, the entitlement should be floor(6/4) = 1 AVAIL.
- However, split withdrawals give (3/4) + (3/4) = 0 + 0 = 0 AVAIL.
- The user receives 0 instead of 1 AVAIL, losing their entire entitlement.

Affected Code
- https://github.com/availproject/migration/blob/dbeed79672bf17e8aaf8ea14e0be8a34ead4cfc0/src/XARMigration.sol#L74-L77

Impacts
Users with small deposits may end up receiving no AVAIL at all despite being entitled to at least one unit. Even for larger deposits, the rounding logic consistently underpays by up to one AVAIL, leaving residual "dust" locked in the contract. This systematic shortfall undermines fairness and can reduce user trust in the migration process.

Remediation
Enforce a minimum deposit amount to prevent small deposits that can entirely vanish in rounding.

Retest

Client's Comments: This is a rare case and can be ignored.

Bug ID #G001 [Won't Fix]

Splitting Require/Revert Statements

Vulnerability Type
Gas Optimization (SCWE-082)

Severity
Gas

Description
Require/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/availproject/migration/blob/dbeed79672bf17e8aaf8ea14e0be8a34ead4cfc0/src/XARMigration.sol#L41

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

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

Retest
Client's Comments: This is a rare case and can be ignored.

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/migration/blob/dbeed79672bf17e8aaf8ea14e0be8a34ead4cfc0/src/XARMigration.sol#L41

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

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

Retest:
Client's Comments: This is a rare case and can be ignored.

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