



CredShields

# DAML Smart Contract Audit

February 19, 2026 • CONFIDENTIAL

## Description

This document details the process and result of the DAML Smart Contract audit performed by CredShields Technologies PTE. LTD. on behalf of Asterizm between January 14, 2026, and January 21, 2026. A retest was performed on February 16, 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

Asterizm

# Table of Contents

<b>Table of Contents</b>	<b>2</b>
<b>1. Executive Summary -----</b>	<b>3</b>
State of Security	4
<b>2. The Methodology -----</b>	<b>5</b>
2.1 Preparation Phase	5
2.1.1 Scope	5
2.1.2 Documentation	6
2.1.3 Audit Goals	6
2.2 Retesting Phase	6
2.3 Vulnerability classification and severity	7
2.4 CredShields staff	8
<b>3. Findings Summary -----</b>	<b>9</b>
3.1 Findings Overview	9
3.1.1 Vulnerability Summary	9
<b>5. Bug Reports -----</b>	<b>10</b>
Bug ID #C001 [Fixed]	10
Client Owner Can Not Create Locked Amulet Vault Proposal	10
Bug ID #H001 [Fixed]	12
Sender Can Relay Message Without Paying Relayer Fee	12
Bug ID #M001 [Acknowledged]	13
clientOwner Can Liquidate Locked Amulet Without Transfer Execution	13
Bug ID #M002 [Fixed]	14
Wallet provider can be arbitrarily selected in payment	14
Bug ID #M003 [Acknowledged]	15
Sender Can Overpay Relayer Fee Without Refund	15
Bug ID #I001 [Fixed]	16
Manager Can Misconfigure Factory Initialization Parameters	16
<b>6. The Disclosure -----</b>	<b>17</b>



# 1. Executive Summary -----

Asterizm engaged CredShields to perform a Daml smart contract audit from January 14, 2026, to January 21, 2026. During this timeframe, 6 vulnerabilities were identified. **A retest was performed on February 16, 2026, and all the bugs have been addressed.**

During the audit, 2 vulnerabilities were found with a severity rating of either High or Critical. These vulnerabilities represent the greatest immediate risk to "Asterizm" 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	Σ
Asterizm SDK Contracts	1	1	3	0	1	<b>6</b>

*Table: Vulnerabilities Per Asset in Scope*

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



## 2. The Methodology -----

Asterizm engaged CredShields to perform the Asterizm SDK Contracts audit. The following sections cover how the engagement was put together and executed.

### 2.1 Preparation Phase

The CredShields team meticulously reviewed all Daml templates, choices, and data structures to gain a thorough understanding of the contract's business logic and authorization rules. They examined all choice controllers and observers, creating a mind map to systematically identify potential security vulnerabilities—such as unauthorized access, non-deterministic behavior, or privacy leaks—prioritizing those that were more critical and business-sensitive for the refactored code. To confirm their findings, the team deployed the package to a Daml Sandbox environment and performed verifications and validations using Daml Script to simulate real-world ledger interactions.

A testing window from January 14, 2026, to January 21, 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
<b>Audited Scope:</b> <a href="https://github.com/Asterizm-Protocol/asterizm-contracts-canton/tree/1ada974130fdc910dc87b4b271988b3cab0b4833">https://github.com/Asterizm-Protocol/asterizm-contracts-canton/tree/1ada974130fdc910dc87b4b271988b3cab0b4833</a>
<b>Retested Scope:</b> <a href="https://github.com/Asterizm-Protocol/asterizm-contracts-canton/tree/9708861832e87aff306688681d3b11ad881cf927">https://github.com/Asterizm-Protocol/asterizm-contracts-canton/tree/9708861832e87aff306688681d3b11ad881cf927</a>



## 2.1.2 Documentation

- <https://docs.asterizm.io/>

## 2.1.3 Audit Goals

CredShields employs a combination of in-house tools and thorough manual review processes to deliver comprehensive Daml smart contract security audits. The majority of the audit involves manual inspection of the Daml source code, focusing specifically on ledger-level authorization and data privacy models. Our review is guided by an extended, self-developed checklist built from industry best practices for functional languages and privacy-first distributed ledgers. The team focuses on deeply understanding the contract's choice controllers, signatory requirements, and observer rights, designing targeted Daml Scripts to test for potential vulnerabilities such as unauthorized access or non-deterministic logic.

CredShields aligns its auditing methodology with global security standards, adapting principles from the Smart Contract Security Verification Standard ([SCSVS](#)) and the Smart Contract Weakness Enumeration ([SCWE](#)) to the specific architecture of the Daml Ledger Model. These frameworks, which the CredShields team actively contributes to, are tailored during the audit to address Daml-specific risks like disclosure of sensitive data and transaction contention. By adhering to these robust baselines, we ensure that each audit is performed against a transparent and technically sound framework. This approach enables us to deliver structured, high-quality audits that address both common ledger risks and complex multi-party business logic vulnerabilities systematically.

## 2.2 Retesting Phase

Asterizm 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 and best practices ensure software stability and maintainability. Informational findings include code style improvements, unused Template variables, or lack of documentation for complex Choices. These are opportunities for improvement that do not pose a direct risk to the ledger's state. Code maintainers should use their own judgment on whether to address them.

### 2. Low



Low-risk vulnerabilities are those with a minimal impact on the ledger or those that cannot be exploited repeatedly. This includes minor redundancies in Signatory logic or issues the client considers insignificant based on their specific business circumstances. While they don't break the contract, they may lead to inefficient ledger usage.

### 3. Medium

Medium-severity vulnerabilities arise from weak or flawed logic that could lead to the exfiltration of private data or unauthorized modification of non-critical contract state. This involves improper use of Observers, leading to unintended data disclosure (divulgence). 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 contract and the organization. They can result in the loss of assets (fungible or non-fungible) for some users or allow a party to bypass authorization for critical business choices. These issues often involve complex logic flaws in Choice Controllers and can severely harm the client's reputation. Immediate remediation is recommended.

### 5. Critical

Critical issues are directly exploitable bugs that do not require specific conditions. These include vulnerabilities that allow unauthorized parties to act as Signatories, result in a permanent loss of assets/funds, or expose sensitive PII (Personally Identifiable Information) across the entire network. If left unaddressed, these issues will severely impact the client's reputation and financial stability.

## 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, 6 security vulnerabilities were identified in the asset.

Vulnerability Title	Severity	Vulnerability Type	Status
Client Owner Can Not Create Locked Amulet Vault Proposal	Critical	Missing Functionality	Fixed
Sender Can Relay Message Without Paying Relay Fee	High	Business Logic Error	Fixed
clientOwner Can Liquidate Locked Amulet Without Transfer Execution	Medium	Missing State Validation	Acknowledged
Wallet provider can be arbitrarily selected in payment	Medium	Missing validation	Fixed
Sender Can Overpay Relay Fee Without Refund	Medium	Business Logic Error	Acknowledged
Manager Can Misconfigure Factory Initialization Parameters	Informational	Logical Error	Fixed

*Table: Findings and Remediations*



## 5. Bug Reports -----

Bug ID #C001[Fixed]

### Client Owner Can Not Create Locked Amulet Vault Proposal

#### Vulnerability Type

Missing Functionality

#### Severity

Critical

#### Description

The `Client.CreateAmuletVaultProposal` choice is intended to allow the contract owner to initiate the creation of an Amulet vault-related proposal, which is expected to support both standard vault proposals and proposals that include a lock mechanism. The surrounding contract parameters and comments indicate that Amulet vault functionality is part of the intended design and workflow.

However, the current implementation only creates an `AmuletVault.AmuletVaultProposal` and lacks any functionality to create an `AmuletVault.AmuletVaultWithLockProposal`. The root cause is an incomplete implementation where the Amulet vault feature is only partially wired, and the vault contract itself is not used anywhere else in the system, as confirmed by the client.

#### Affected Code

- <https://github.com/Asterizm-Protocol/asterizm-contracts-canton/blob/1ada974130fdc910dc87b4b271988b3cab0b4833/sdk/src/Asterizm/Client/Client.daml>

#### Impacts

The system cannot support locked Amulet vault workflows, preventing intended use cases such as time-locked or condition-locked vaults. This results in reduced functionality, potential misalignment with business requirements, and future integration risk if downstream components assume the presence of locked vault proposals.

#### Remediation

Implement support for creating `AmuletVaultWithLockProposal` or remove unused Amulet vault logic to ensure the contract behavior matches the intended system design.



## Retest

Fixed in [970886](#).



Bug ID #H001[Fixed]

## Sender Can Relay Message Without Paying Relay Fee

### Vulnerability Type

Business Logic Error

### Severity

High

### Description

The `Client.SendMessageWithValue` choice is designed to allow a sender to relay a message while providing a `value` intended to cover the relayer fee defined in `customRelayContract.fee`. The expected behavior is that the relayer is compensated for message delivery, either by deducting the fee from the user-provided amount or by transferring the fee on-chain as part of the transaction flow.

However, while the implementation checks that `value >= customRelayContract.fee`, the relayer fee is neither paid on-chain nor deducted from the user's amount. The value is only recorded in the `SendMessageEvent` and no accounting, transfer, or balance adjustment occurs. The root cause is the absence of on-ledger fee settlement logic despite the presence of a fee validation check.

### Affected Code

- <https://github.com/Asterizm-Protocol/asterizm-contracts-canton/blob/1ada974130fdc910dc87b4b271988b3cab0b4833/sdk/src/Asterizm/Client/Client.daml#L26-L63>

### Impacts

Relayers are not economically compensated by the protocol, undermining the incentive model and potentially causing relayers to stop servicing messages. At the same time, the system creates a false assurance that fees are enforced, leading to inconsistent business logic and operational risk.

### Remediation

Explicitly deduct and transfer the relayer fee on-chain, or redesign the flow to clearly separate off-chain fee settlement from on-chain validation.

### Retest

This vulnerability is not fixed.



Bug ID #M001[Acknowledged]

## clientOwner Can Liquidate Locked Amulet Without Transfer Execution

### Vulnerability Type

Missing State Validation

### Severity

Medium

### Description

The `AmuletVaultWithLock_Liquidate` choice is intended to liquidate locked Amulet funds after a cross-chain transfer has been successfully executed. The expected flow is that liquidation is only permitted once the associated transfer has completed and the system has marked the transfer as executed, ensuring correct sequencing and preventing premature fund movements.

However, the current implementation does not validate whether the related transfer has already been executed before allowing liquidation. The choice proceeds directly to unlocking the locked Amulet and transferring funds to the vault without checking execution state. The root cause is the absence of a guard condition tying liquidation eligibility to a completed transfer state.

### Affected Code

- <https://github.com/Asterizm-Protocol/asterizm-contracts-canton/blob/1ada974130fdc910dc87b4b271988b3cab0b4833/sdk/src/Asterizm/AmuletVaultWithLock.daml#L263-L285>

### Impacts

Locked funds can be liquidated prematurely or out of sequence, breaking transfer invariants and potentially resulting in incorrect fund routing or double-spend-like scenarios across the transfer lifecycle. This undermines the integrity of the cross-chain settlement process and increases the risk of financial inconsistencies.

### Remediation

Require validation that the associated transfer has been executed before allowing liquidation to proceed.

### Retest

Client's comment: yes, liquidation is possible at any time, even if the user hasn't received funds on the Ethereum side (which is extremely unlikely). It's strange to restrict liquidation — it's an emergency withdrawal mechanism.



Bug ID #M002 [Fixed]

## Wallet provider can be arbitrarily selected in payment

### Vulnerability Type

Missing validation

### Severity

Medium

### Description

`AmuletVaultUserService_MakePayment` accepts a `walletProvider` argument from the caller and uses it directly as the transfer provider. This value is not checked against the `walletProvider` stored on the `AmuletVaultUserService` template. The choice is controlled by user and the provided `walletProvider`, so any party can pass itself as the provider. As a result, the actual provider recorded in the transfer can differ from the one the client intended or configured. This breaks the expected binding between the user service and its configured wallet provider. It could allow payments to be routed through an unintended provider without any on-ledger check.

### Affected Code

- <https://github.com/Asterizm-Protocol/asterizm-contracts-canton/blob/1ada974130fdc910dc87b4b271988b3cab0b4833/sdk/src/Asterizm/AmuletVaultWithLock.daml#L93-L137>

### Impacts

Payments can be executed under a different provider than the one configured for the user service. This can affect fee calculations, auditing, compliance checks, or any off-chain assumptions tied to the provider identity.

### Remediation

Enforce `walletProvider == self.walletProvider` inside the choice. If dynamic providers are required, store an allow-list and validate membership, or update the template via an explicit admin choice rather than accepting arbitrary input.

### Retest

This issue is [fixed](#).



Bug ID #M003 [Acknowledged]

## Sender Can Overpay Relay Fee Without Refund

### Vulnerability Type

Business Logic Error

### Severity

Medium

### Description

The `Client.SendMessageWithValue` choice allows a sender to relay a message by providing a `value` intended to cover the relay fee defined in `customRelayContract.fee`. The expected flow is that users submit the exact fee required to compensate the relay for message delivery.

However, the implementation only enforces `value >= customRelayContract.fee` and forwards the entire `value` into the `SendMessageEvent` without refunding any excess or constraining the amount. The root cause is the absence of logic to either validate that value equals the exact fee or to refund any overpayment to the sender.

### Affected Code

- <https://github.com/Asterizm-Protocol/asterizm-contracts-canton/blob/1ada974130fdc910dc87b4b271988b3cab0b4833/sdk/src/Asterizm/Client/Client.daml#L41>

### Impacts

Bob submits a transaction with a `value` greater than the relay fee, unintentionally overpaying. The excess amount is permanently transferred to the relay flow with no refund mechanism, leading to direct and unnecessary loss of funds for the sender.

### Remediation

Enforce exact fee payment or explicitly refund any excess value above `customRelayContract.fee` to the sender.

### Retest

Client's comment: yes, the sender can pay more commission to the relay than specified because they set that value manually in order to cover gas price in Ethereum (this is done in other networks too).



Bug ID #I001[Fixed]

## Manager Can Misconfigure Factory Initialization Parameters

### Vulnerability Type

Logical Error

### Severity

Informational

### Description

The `Factory.Initialize` choice is intended to be executed by the factory manager to deploy and initialize core system contracts, including the custom relayer contract and the local chain configuration. The `Factory` template already defines manager as a contract parameter and enforces it as the sole signatory and controller for initialization.

However, the `Initialize` choice redundantly accepts `manager : Party` again as an input parameter, even though this value is neither necessary nor authoritative compared to the template-level `manager`. The root cause is parameter duplication, which introduces ambiguity between the template state and the choice arguments and increases the risk of misconfiguration or incorrect assumptions by integrators.

### Affected Code

- <https://github.com/Asterizm-Protocol/asterizm-contracts-canton/blob/1ada974130fdc910dc87b4b271988b3cab0b4833/sdk/src/Asterizm/Initializer.daml#L19>

### Impacts

Initialization logic becomes harder to reason about and more error-prone, increasing operational risk. Downstream developers or integrators may incorrectly assume the passed-in manager parameter is validated or used, leading to inconsistent expectations and potential deployment mistakes.

### Remediation

Remove the redundant `manager` parameter from the `Initialize` choice and rely exclusively on the template-level `manager` for authorization and configuration.

### Retest

Fixed in [970886](#).



## 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 Daml-based distributed applications and private/public ledgers 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's assessment of Daml templates, ledger-level privacy controls, or multi-party workflows.



# 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

