CredShields

# Smart Contract Audit

29 January 2026 • CONFIDENTIAL

### Description

This document details the process and result of the Smart Contract audit performed by CredShields Technologies PTE. LTD. on behalf of Akita between 27 January 2026, and 27 January 2026. A retest was performed on 29 January 2026.


### 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

Akita

# Table of Contents

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

Akita engaged CredShields to perform a smart contract audit from 27 January 2026 to 27 January 2026. During this timeframe, 5 vulnerabilities were identified. **A retest was performed on 29 January 2026, 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 "Akita" and should be prioritized for remediation; 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 |
|---|---|---|---|---|---|---|---|
| Akita Contract | 0 | 0 | 0 | 2 | 3 | 0 | **5** |

*Table: Vulnerabilities Per Asset in Scope*

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

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

Akita 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 27 January 2026 to 27 January 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: Akita.sol**<br>**Retested Scope: Akita.sol** |

### 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

Akita 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, 5 security vulnerabilities were identified in the asset.

| Vulnerability Title | Severity | Vulnerability Type | Status |
|---|---|---|---|
| Contract Lacks Ability To Update Recovery Address | Low | Missing functionality (SCWE-006) | Fixed |
| Missing events in important functions | Low | Missing Event Emission (SCWE-063) | Fixed |
| Missing zero address validations | Informational | Missing Input Validation (SC04-Lack Of Input Validation) | Fixed |
| Constructor Is Unnecessary Marked As Payable | Informational | Misconfiguration | Fixed |
| ETH Recovery Function Is Unusable | Informational | Design Issue | Fixed |

*Table: Findings and Remediations*

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

Bug ID #L001 [Fixed]

## Contract Lacks Ability To Update Recovery Address

**Vulnerability Type**
Missing functionality (SCWE-006)

**Severity**
Low

**Description**
The Akita contract defines a privileged `recovery` address that is authorized to withdraw ETH and ERC20 tokens accidentally sent to the contract via `Akita.recoverETH` and `Akita.recoverERC20`. This address is set once during contract deployment through the constructor and is intended to represent a trusted recovery operator. However, the contract does not provide any mechanism to update or rotate the `recovery` address after deployment. As a result, if the private key controlling the recovery address is lost, compromised, or if operational requirements change, the recovery functionality becomes permanently inaccessible or unsafe due to the immutability of the `recovery` variable.

**Affected Code**

- Akita#L26

**Impacts**
If the recovery address becomes unusable or untrusted, ETH or ERC20 tokens sent to the contract in the future cannot be safely recovered, potentially resulting in permanent loss of funds or operational risk.

**Remediation**
Introduce a controlled setter function that allows the recovery address to be updated to a new trusted address.

**Retest**
This issue has been fixed by adding the updateRecorvery function.

Bug ID #L002 [Fixed]

## Missing events in important functions

**Vulnerability Type**
Missing Event Emission (SCWE-063)

**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**

- Akita#L31
- Akita#L40

**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**
It is suggested that we consider emitting events for important functions to keep track of them.

**Retest**
This issue has been fixed by emitting events.

Bug ID #I001 [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**

- Akita#L25

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

**Remediation**
It is recommended to add a zero address validation to all the functions where addresses are being set.

**Retest**
This issue has been fixed

Bug ID #I002 [Fixed]

## Constructor Is Unnecessary Marked As Payable

**Vulnerability Type**
Misconfiguration

**Severity**
Informational

**Description**
The contract is designed as an ERC20 token that mints a fixed supply at deployment and optionally allows the designated recovery address to withdraw ETH or ERC20 tokens that are accidentally sent to the contract. The constructor is responsible for initializing the recovery address, setting token metadata, and minting the total supply to the receiver, and it is executed only once during deployment by the deployer.
However, the constructor is marked as `payable` even though it does not use `msg.value`, store ETH-related state, or perform any logic that depends on receiving ETH. The root cause is the inclusion of the `payable` modifier without a functional requirement, which allows ETH to be sent during deployment without any explicit intention or documentation.

**Affected Code**

- Akita#L25

**Impacts**
Auditors, integrators, or users reviewing the contract may incorrectly assume that ETH sent during deployment has a specific purpose, potentially leading to confusion, incorrect assumptions about token economics, or misinterpretation of the contract's design.

**Remediation**
Remove the `payable` modifier from the constructor to clearly signal that ETH is not intended to be sent or handled during deployment.

**Retest**
This issue has been fixed by removing payable from the constructor.

Bug ID #I003 [Fixed]

## ETH Recovery Function Is Unusable

**Vulnerability Type**
Design Issue

**Severity**
Informational

**Description**
The contract includes a `recoverETH()` function, but it does not implement a `receive()` or `fallback()` function. As a result, the contract cannot accept ETH through normal transfers after deployment. ETH can only be received via the payable constructor or forced transfers, making the ETH recovery logic effectively redundant in normal operation.

**Affected Code**

- Akita#L31

**Impacts**
ETH recovery is unreachable under normal conditions, leading to misleading or dead code.

**Remediation**
Add a `receive()` or `fallback()` function to allow ETH transfers, or remove the `recoverETH()` function if `ETH` handling is not intended.

**Retest**
This issue has been resolved by adding a receive function.

# 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


20A TANJONG PAGAR ROAD, SINGAPORE (088443)

info@credshields.com