DApp Developers and Smart Contract Auditors

SMART CONTRACT SECURITY AUDIT

of

AIM TOKEN CONTRACT

Smart Contract Audit of Aim Token

December 25th, 2025 | v. 1.1

TABLE OF CONTENTS

AUDIT INTRODUCTION

Auditing Firm	SecureDApp Auditors
Audit Architecture	SecureDApp Auditing Standard
Language	Solidity
Client Firm	AIM Token
Youtube	-
Twitter	-
Linkedin	-
Facebook	-
Instagram	-
Report Date	December 28th, 2025

AUDIT DOCUMENT

Name	Smart Contract Code Review and Security Analysis Report for AIM Token
Approved By	Himanshu Gautam| CTO at SecureDApp
Type	Custom
Platform	EVM
Language	Solidity
Changelog	Final Review

AUDIT SCOPE

The scope of this report is to audit the smart contract source code of Aim Token contract.
Our client provided us with multiple smart contracts deployed at:

- https://bscscan.com/token/0x7C978c0d413D8abB96bc646f93BD67B135b5e8D0#code

The contract is written in Solidity. After initial research, we agreed to perform the following tests and analyses as part of our well-rounded audit:

- Smart contract behavioural consistency analysis
- Test coverage analysis
- Penetration testing: checking against our database of vulnerabilities and simulating manual attacks against the contracts
- Static analysis
- Manual code review and evaluation of code quality
- Analysis of GAS usage
- Contract analysis with regards to the host network

Final Review Scope

Contract Link	https://bscscan.com/token/0x7C978c0d413D8abB96bc646f93BD67B135b5e8D0#code
Commit Hash	N/A
Functional Requirements	Partial documentation provided.
Technical Requirements	Partial documentation provided.
Contracts Addresses	0x7C978c0d413D8abB96bc646f93BD67B135b5e8D0
Contracts	AIM.sol

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to asset loss or data manipulation.
High	High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g., public access to crucial functions.
Medium	Medium-level vulnerabilities are important to fix; however, they cannot lead to asset loss or data manipulation.
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that cannot have a significant impact on execution.
Informational	Issue listed to improve understanding, readability and quality of code

All statuses which are identified in the audit report are categorized here for the reader to review:

Status Type	Definition
Open	Risks are open.
Acknowledged	Risks are acknowledged, but not fixed.
Resolved	Risks are acknowledged and fixed.

AUDIT SUMMARY

The SecureDApp team has performed a line-by-line manual analysis and automated review of smart contracts. Smart contracts were analyzed mainly for common contract vulnerabilities, exploits, and manipulation hacks. According to the audit:

Status	Critical	High	Medium	Low	Informative
Open	0	0	0	0	0
Acknowledged	2	0	1	2	0
Resolved	0	0	0	0	0

AUDIT METHODOLOGY

SecureDApp scans contracts and reviews codes for common vulnerabilities, exploits, hacks and backdoors. Mentioned are the steps used by SecureDApp to audit smart contracts:

a. Smart contract source code review:
    i. Review of the specifications, sources, and instructions provided to SecureDApp to make sure we understand the audit scope, intended business behaviour, overall architecture, and the project's goal.
    ii. Manual review of code which is the process of reading source code line-by-line to identify potential vulnerabilities.
b. Test coverage analysis: (Unit testing)
    i. Test coverage analysis is the process of determining whether the test cases are covering the code and how much code is exercised when we run those test cases.
c. Static analysis:
    i. Run a suite of vulnerability detectors to find security concerns in smart contracts with different impact levels.
d. Symbolically executed tests: (SMTChecker testing) (Taint analysis)
    i. Symbolic execution is analysing a program to determine what inputs cause each part of a program to execute.
    ii. Check for security vulnerabilities using static and dynamic analysis
e. Property-based analysis (Fuzz tests)(Invariant testing)
    i. Run the execution flow multiple times by generating random sequences of calls to the contract.
    ii. Asserts that all the invariants hold true for all scenarios.
f. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarity, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
g. Specific, itemised, actionable recommendations to help you take steps to secure your smart contracts.

Automated 5S frameworks are used to assess the smart contract vulnerabilities

- Consensys Tools
- SWC Registry
- Solidity Coverage
- Open Zeppelin Code Analyser
- Solidity Shield Scan

We have audited the smart contracts for commonly known and more specific vulnerabilities. Below is the list of smart contract tests, vulnerabilities, exploits, and hacks:

ID	Description	Status
EEA 3.3	Oracle Manipulation	N/A
EEA 3.3	Bad Randomness - VRF	N/A
S60	Assembly Usage	Passed
S59	Dangerous usage of block.timestamp	Passed
EEA 3.7	Front-Running Attacks	N/A
EEA 3.7	Back-Running Attacks	N/A
EEA 3.7	Sandwich Attacks	N/A
DASP	Gas Griefing Attacks	Passed
DASP	Force Feeding	Passed
SCSVS V2	Access Control	Passed
DASP	Short Address Attack	Passed
DASP	Checks Effects Interactions	Passed
EEA 4.1	No Self-destruct	Passed
SCSVS V14	Decentralized Finance Checks	Passed

Slither Tests	Checks for ERC's conformance	Passed
Coverage	Unit tests with 100% coverage	N/A
Gas Reporter	Gas usage & limitations	Passed
Echidna Tests	Malicious input handling	Passed
SWC-101	Integer Overflow and Underflow	Passed
SWC-102	Outdated Compiler Version	Passed
SWC-103	Floating Pragma	Passed
SWC-104	Unchecked Call Return Value	Passed
SWC-105	Unprotected Ether Withdrawal	Passed
SWC-106	Unprotected SELF-DESTRUCT Instruction	Passed
SWC-107	Re-entrancy	Passed
SWC-108	State Variable Default Visibility	Passed
SWC-109	Uninitialized Storage Pointer	Passed
SWC-110	Assert Violation	Passed
SWC-111	Use of Deprecated Solidity Functions	Passed
SWC-112	Delegate Call to Untrusted Callee	Passed

SWC-113	DoS with Failed Call	Passed
SWC-114	Transaction Order Dependence	Passed
SWC-115	Authorization through tx.origin	Passed
SWC-116	Block values as a proxy for time	Passed
SWC-117	Signature Malleability	Passed
SWC-134	Message call with the hardcoded gas amount	Passed
SWC-135	Code With No Effects (Irrelevant/Dead Code)	**Informational**
SWC-136/SCSVS V3	Unencrypted Private Data On-Chain	Passed

SYSTEM OVERVIEW

Privileged roles

1. Contract Owner Role
    a. Set through the constructor on deployment

Risks

1. The impact of the owner role being compromised would have a huge impact on the protocol.
2. Centralisation risk is the most common cause of cryptography asset loss.
3. Compromising the Owner Role may lead to all user’s asset loss.

FINDINGS

Centralization Risk

Centralization risk is the most common cause of dapp's hacks. When a smart contract has an active contract ownership, the risk related to centralization is elevated. There are some well-intended reasons to be an active contract owner, such as:

- Contract owners can be granted the power to pause() or lock() the contract in case of an external attack.
- Contract owners can use functions like, include(), and exclude() to add or remove wallets from fees, swap checks, and transaction limits. This is useful to run a presale, and to list on an exchange.

Authorizing a full centralized power to a single body can be dangerous. Unfortunately, centralization related risks are higher than common smart contract vulnerabilities. Centralization of ownership creates a risk of rug pull scams, where owners cash out tokens in such quantities that they become valueless. The most important question to ask here is, how to mitigate centralization risk? Here's SecureDApp's recommendation to lower the risks related to centralization hacks:

- Smart contract owner's private key must be carefully secured to avoid any potential hack.
- Smart contract ownership should be shared by multi-signature (multi-sig) wallets.
- Smart contract ownership can be locked in a contract, user voting, or community DAO can be introduced to unlock the ownership.

Aim Token Centralization Status

- AIM smart contract has an Owner Role.
- To avoid centralisation risk, consider using Openzeppelin Access Control instead of Ownable.

MANUAL REVIEW

Identifier	Definition	Severity
CEN-01	Excessive Owner Privileges Allow Arbitrary Freezing of Transfers and Accounts	Critical

Contract: AIM.sol

The `AIM` token contract grants the owner **unrestricted administrative authority** over core token transfer mechanics. Through a combination of global transfer controls and address-level blacklisting, the owner can **unilaterally and indefinitely restrict token movement** for all users, including exchanges and liquidity pools.

1. Global Transfer Freeze

2. Arbitrary Account Blocking (Blacklisting)

Impact:

- Suitable **only for closed or permissioned ecosystems**
- The owner can **freeze all trading at any time**, including during active market conditions.
- Liquidity can be effectively **held hostage**, preventing users from selling or transferring tokens.
- Arbitrary wallets, exchanges, or liquidity pools can be **permanently blacklisted**.
- Creates a **single point of failure** and introduces significant custodial risk.
- Strongly resembles **honeypot or rug-pull behavior** from the perspective of automated scanners and exchanges.
- Requires users to place **absolute trust in the owner**, contradicting decentralization principles.

RECOMMENDATION

Remove or severely restrict global transfer control, or enforce a **time-lock** with public notice.

Status: Acknowledged

The client acknowledges the centralization characteristics of the non-upgradable AIM Token contract, confirms careful and secure management of the owner wallet, and accepts the associated trade-offs; no changes are planned as the permissioned design is intentional.

Identifier	Definition	Severity
CEN-02	Centralized Token Supply Allocation at Deployment	Critical

Contract: AIM.sol

Upon deployment, the `AIM` token contract mints the **entire token supply** to the owner's address without any vesting, distribution constraints, or supply-locking mechanisms.

As a result, **100% of the circulating supply is immediately controlled by a single externally owned account (EOA)** — the contract owner.

There are no safeguards such as:

- Token vesting schedules
- Timelocks
- Multi-signature wallets
- Distribution limits
- Emission schedules

This design introduces a **significant centralization and trust risk**, especially when combined with the owner's ability to restrict transfers and blacklist accounts.

Impact

- The owner can **dump the entire supply** at any time, causing extreme price collapse.
- Market participants must **fully trust the owner** not to manipulate supply.
- No guarantees exist regarding fair distribution or long-term token stability.
- Increases the likelihood of:
  - Rug-pull scenarios

    - Market manipulation
    - Exchange delisting or refusal to list

- Automated risk scanners may flag the token as **high-risk or unsafe** due to full supply centralization.

RECOMMENDATION

Distribute tokens through:

- Vesting contracts
- Time-locked allocations
- Emission schedules

Store treasury and team allocations in **multi-signature wallets**.

Publicly disclose tokenomics and distribution plans.

For public deployments, avoid minting the full supply directly to a single owner address.

Consider immutable or verifiable token distribution mechanisms.

Status: Acknowledged

The client acknowledges the centralization characteristics of the non-upgradable AIM Token contract, confirms careful and secure management of the owner wallet, and accepts the associated trade-offs; no changes are planned as the permissioned design is intentional.

Identifier	Definition	Severity
MED-01	Restrictive Minimum Wallet Balance Logic (Anti-Exit Mechanism)	MEDIUM

Contract: AIM.sol

The `AIM` token contract enforces a **minimum wallet balance requirement** on senders, which can prevent users from fully transferring or exiting their token holdings.

Impact

- Users may be **unable to fully exit their positions**, even when sufficient liquidity exists.
- The owner can dynamically raise `minWalletLimit`, effectively **locking user funds**.
- Creates a hidden **anti-exit or soft honeypot mechanism**, particularly when combined with sell limits or blacklisting.
- May cause unexpected transaction failures on decentralized exchanges.
- Significantly increases custodial risk and undermines user trust.

RECOMMENDATION

Remove the minimum wallet balance enforcement entirely for public tokens.

Status: Acknowledged

Identifier	Definition	Severity
LOW-01	Redundant SafeMath Usage and Missing Administrative Events	Low

Contract: AIM.sol

The contract uses the `SafeMath` library despite relying on Solidity version ≥0.8, which already provides built-in overflow and underflow protection. This results in **unnecessary gas overhead** without improving security.

Additionally, several critical administrative functions do not emit events, reducing on-chain transparency and making it difficult for users, auditors, and monitoring tools to track privileged state changes.

Affected functions include:

- `blockAccount`
- `setTransferFlag`
- `setMaxWalletLimit`
- `setExchangeAccount`

Impact:

- Increased gas costs due to redundant arithmetic checks.
- Reduced transparency of owner actions.
- Off-chain monitoring systems may miss critical administrative changes.
- Harder for users to detect potentially malicious or risky behavior.

RECOMMENDATION

Remove `SafeMath` and rely on Solidity's native overflow checks.

Emit events for all administrative actions to improve transparency and auditability.

Status: Acknowledged

Identifier	Definition	Severity
LOW-02	Outdated Solidity Version Constraint and Missing Immutability Declarations	Low

Contract: AIM.sol

The contract is compiled with Solidity version constraint **0.8.13**, which is known to contain several severe compiler issues that have been addressed in later releases. Using an outdated compiler version may expose the contract to avoidable risks.

Additionally, the state variables `_decimals` and `_totalSupply` are assigned once during deployment and never modified thereafter, yet they are not declared as `immutable`. This results in unnecessary storage usage and reduced code clarity.

Impact

- Potential exposure to known compiler-level issues.
- Higher gas costs due to avoidable storage reads.
- Reduced clarity regarding variables intended to remain constant.

RECOMMENDATION

- Upgrade to a newer stable Solidity version (e.g., ≥0.8.20).
- Declare _decimals and _totalSupply as immutable to improve gas efficiency and code safety.

Status: Acknowledged

DISCLAIMER

SecureDApp Auditors provides an easy-to-understand audit of solidity source code (commonly known as smart contracts).

The smart contract for this particular audit was analyzed for common contract vulnerabilities and centralisation exploits. This audit report makes no statements or warranties on the security of the code. This audit report does not provide any warranty or guarantee regarding the absolute bug-free nature of the smart contract analysed, nor do it provide any indication of the client's business, business model or legal compliance. This audit report does not extend to the compiler layer, any other areas beyond the programming language, or other programming aspects that could present security risks. Cryptographic tokens are emergent technologies; they carry high levels of technical risks and uncertainty. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. This audit report could include false positives, false negatives, and other unpredictable results.

CONFIDENTIALITY

This report is subject to the terms and conditions (including, without limitations, description of services, confidentiality, disclaimer and limitation of liability) outlined in the scope of the audit provided to the client. This report should not be transmitted, disclosed, referred to, or relied upon by any individual for any purpose without SecureDApp's prior written consent.

NO FINANCIAL ADVICE

This audit report does not indicate the endorsement of any particular project or team, nor guarantees its security. No third party should rely on the reports in any way, including to make any decisions to buy or sell a product, service or any other asset. The information provided in this report does not constitute investment advice, financial advice, trading advice, or any other sort of advice and you should not treat any of the report's content as such. This audit report should not be used in any way to make decisions around investment or involvement. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. for avoidance of doubt, services, including any associated audit reports or materials, shall not be considered or relied upon as any form of financial, tax, legal, regulatory, or other advice.

TECHNICAL DISCLAIMER

All services, audit reports, smart contract audits, other materials, or any products or results of the use thereof are provided “as is” and “as available” and with all faults and defects without warranty of any kind. To the maximum extent permitted under applicable law, SecureDApp hereby disclaims all warranties, whether expressed, implied, statutory, or otherwise with respect to services, audit report, or other materials. Without limiting the foregoing, SecureDApp specifically disclaims all implied warranties of merchantability, fitness for a particular purpose, title and non-infringement, and all warranties arising from the course of dealing, usage, or trade practice. Without limiting the foregoing, SecureDApp makes no warranty of any kind that all services, audit reports, smart contract audits, or other materials, or any products or results of the use thereof, will meet the client’s or any other individual’s requirements, achieve any intended result, be compatible or work with any software, system, or other services, or be secure, accurate, complete, free of harmful code, or error-free.

TIMELINESS OF CONTENT

The content contained in this audit report is subject to change without any prior notice. SecureDApp does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following the publication.

LINKS TO OTHER WEBSITES

This audit report provides, through hypertext or other computer links, access to websites and social accounts operated by individuals other than SecureDApp. Such hyperlinks are provided for your reference and convenience only and are the exclusive responsibility of such websites’ and social accounts’ owners. You agree that SecureDApp is not responsible for the content or operation of such websites and social accounts and that SecureDApp shall have no liability to you or any other person or entity for the use of third-party websites and social accounts. You are solely responsible for determining the extent to which you may use any content at any other websites and social accounts to which you link from the report.

ABOUT SECUREDAPP

SecureDApp Auditor provides intelligent blockchain solutions. SecureDapp is developing an ecosystem that is seamless and responsive. Some of our services: Blockchain Security, Token Launchpad, NFT Marketplace, etc. SecureDapp's mission is to interconnect multiple services like Blockchain Security, DeFi, Gaming, and Marketplace under one ecosystem that is seamless, multi-chain compatible, scalable, secure, fast, responsive, and easy to use.

SecureDApp is built by a decentralised team of UI experts, contributors, engineers, and enthusiasts from all over the world. Our team currently consists of 6+ core team members and 10+ casual contributors. SecureDApp provides manual, static, and automatic smart contract analysis to ensure that the project is checked against known attacks and potential vulnerabilities.

To learn more, visit : https://securedapp.in/

To view our audit portfolio, visit : github.securedapp.in

To book an audit, message : securedapp.telegram

Securedapp.in

Securedapp_Linkedin

Securedapp_Telegram