SolidityScan | Powered by CRED SHIELDS

Security Assessment Report

AIBToken

31 Oct 2025

This security assessment report was prepared by SolidityScan.com, a cloud-based Smart Contract Scanner.

Table of Contents

MISSING @INHERITDOC ON OVERRIDE FUNCTIONS
MISSING NATSPEC COMMENTS IN ASSEMBLY BLOCKS
MISSING NATSPEC COMMENTS IN SCOPE BLOCKS
MISSING NATSPEC DESCRIPTIONS FOR PUBLIC VARIABLE DECLARATIONS
MISSING @NOTICE IN NATSPEC COMMENTS FOR CONSTRUCTORS
MISSING @NOTICE IN NATSPEC COMMENTS FOR FUNCTIONS
MISSING UNDERSCORE IN NAMING VARIABLES
NAME MAPPING PARAMETERS
REVERT STATEMENTS WITHIN EXTERNAL AND PUBLIC FUNCTIONS CAN BE USED TO PERFORM DOS ATTACKS
USE CALL INSTEAD OF TRANSFER OR SEND
USE SCIENTIFIC NOTATION
IN-LINE ASSEMBLY DETECTED
AVOID RE-STORING VALUES
AVOID ZERO-TO-ONE STORAGE WRITES
CACHE ADDRESS(THIS) WHEN USED MORE THAN ONCE
CHEAPER CONDITIONAL OPERATORS
CHEAPER INEQUALITIES IN IF()
DEFAULT INT VALUES ARE MANUALLY RESET
DEFINE CONSTRUCTOR AS PAYABLE
REVERTING FUNCTIONS CAN BE PAYABLE
GAS INEFFICIENCY DUE TO MULTIPLE OPERANDS IN SINGLE IF/ELSEIF CONDITION
OPTIMIZING ADDRESS ID MAPPING
PUBLIC CONSTANTS CAN BE PRIVATE
SPLITTING REVERT STATEMENTS
STORAGE VARIABLE CACHING IN MEMORY

UNNECESSARY DEFAULT VALUE INITIALIZATION
USE SELFBALANCE() INSTEAD OF ADDRESS(THIS).BALANCE

05 Scan History

06 Disclaimer

01. Vulnerability Classification and Severity

Description

To enhance navigability, the document is organized in descending order of severity for easy reference. Issues are categorized as *Fixed*, *Pending Fix*, or *Won't Fix*, indicating their current status. *Won't Fix* denotes that the team is aware of the issue but has chosen not to resolve it. Issues labeled as *Pending Fix* state that the bug is yet to be resolved. Additionally, each issue's severity is assessed based on the risk of exploitation or the potential for other unexpected or unsafe behavior.

Critical

The issue affects the contract in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss.

High

High-severity vulnerabilities pose a significant risk to both the Smart Contract and the organization. They can lead to user fund losses, may have conditional requirements, and are challenging to exploit.

Medium

The issue affects the ability of the contract to operate in a way that doesn't significantly hinder its behavior.

Low

The issue has minimal impact on the contract's ability to operate.

Informational

The issue does not affect the contract's operational capability but is considered good practice to address.

Gas

This category deals with optimizing code and refactoring to conserve gas.

02. Executive Summary

AIBToken

0×4fb0112f59DF1aa06B1e099fD624D479cfDc9789

https://bscscan.com/address/0×4fb0112f59DF1aa06B1e099fD6...

Language	Audit Methodology	Contract Type
Solidity	**Static Scanning**	-

Website	Publishers/Owner Name	Organization
-	-	-

Contact Email

-

Security Score is AVERAGE

The SolidityScan score is calculated based on lines of code and weights assigned to each issue depending on the severity and confidence. To improve your score, view the detailed result and leverage the remediation solutions provided.

This report has been prepared for AIBToken using SolidityScan to scan and discover vulnerabilities and safe coding practices in their smart contract including the libraries used by the contract that are not officially recognized. The SolidityScan tool runs a comprehensive static analysis on the Solidity code and finds vulnerabilities ranging from minor gas optimizations to major vulnerabilities leading to the loss of funds. The coverage scope pays attention to all the informational and critical vulnerabilities with over 700+ modules. The scanning and auditing process covers the following areas:

Various common and uncommon attack vectors will be investigated to ensure that the smart contracts are secure from malicious actors. The scanner modules find and flag issues related to Gas optimizations that help in reducing the overall Gas cost It scans and evaluates the codebase against industry best practices and standards to ensure compliance It makes sure that the officially recognized libraries used in the code are secure and up to date.

The SolidityScan Team recommends running regular audit scans to identify any vulnerabilities that are introduced after AIBToken introduces new features or refactors the code.

03. Threat Summary

Threat Score

High Risk

32.5/100

THREAT SUMMARY

Your smart contract has been assessed and assigned a **High Risk** threat score. The score indicates the likelihood of risk associated with the contract code.

Contract's source code is verified.

Source code verification provides transparency for users interacting with smart contracts. Block explorers validate the compiled code with the one on the blockchain. This also gives users a chance to audit the contracts, ensuring that the deployed code matches the intended functionality and minimizing the risk of malicious or erroneous contracts.

The contract can mint new tokens.

Minting functions are often utilized to generate new tokens, which can be allocated to specific addresses, such as user wallets or the contract owner's wallet. This feature is commonly employed in various decentralized finance (DeFi) and non-fungible token (NFT) projects to facilitate token issuance and distribution. The Presence of Minting Function module is designed to quickly identify the presence and implementation of minting functions in a smart contract. Mint functions play a crucial role in creating new tokens and transferring them to the designated user's or owner's wallet. This process significantly contributes to increasing the overall circulation of the tokens within the ecosystem.

The tokens cannot be burned in this contract.

The token contract incorporates a burn function that enables the intentional reduction of token amounts, consequently diminishing the total supply. The execution of this burn function contributes to the creation of scarcity within the token ecosystem, as the overall availability of the token decreases.

The contract can be compiled with a more recent Solidity version

The contract should be written using the latest Solidity pragma version as it comes with numerous bug fixes. Utilizing an outdated version exposes the contract to vulnerabilities associated with known issues that have been addressed in subsequent updates. Therefore, it is essential to stay current with the latest Solidity version to ensure the robustness and security of the contract against potential vulnerabilities.

This is not a proxy-based upgradable contract.

The Proxy-Based Upgradable Contract module is dedicated to identifying the presence of upgradeable contracts or proxy patterns within a smart contract. The utilization of upgradeable contracts or proxy patterns enables contract owners to make dynamic changes to various aspects, including functions, token circulation, and distribution, without requiring a complete redeployment of the contract.

Owners can blacklist tokens or users.

This module is designed to identify whether the owner of a smart contract has the capability to blacklist specific tokens or users. In a scenario where owners possess the authority to blacklist, all transactions related to the blacklisted entities will be immediately halted. Ownership privileges that include the ability to blacklist tokens or users can be a critical feature in certain use cases, providing the owner with control over potential malicious activities, compliance issues, or other concerns. However, in situations where this authority is abused or misapplied, it can lead to unintended consequences and user dissatisfaction.

Is ERC-20 token.

A token is expected to adhere to the established standards of the ERC-20 token specification, encompassing the inclusion of all necessary functions with standardized names and arguments as defined by the ERC-20 standard.

This is not a Pausable contract.

Pausable contracts refer to contracts that can be intentionally halted by their owners, temporarily preventing token holders from engaging in buying or selling activities. This pause mechanism allows contract owners to exert control over the token's functionality, introducing a temporary suspension in trading activities for various reasons such as security concerns, updates, or regulatory compliance adjustments.

Critical functions that add, update, or delete owner/admin addresses are detected.

A smart contract within the Web3 ecosystem that incorporates critical administrative functions can potentially compromise the transparency and intended objectives of the contract. It is imperative to conduct a thorough examination of these functions, especially in the realm of Web3 smart contracts. Minimizing administrative functions in a token contract within the Web3 framework can significantly reduce the likelihood of complications and enhance overall efficiency and clarity.

The contract cannot be self-destructed by owners.

The SELFDESTRUCT opcode is a critical operation in Ethereum smart contracts, allowing a contract to autonomously terminate itself. When invoked, this opcode deallocates the contract, freeing up storage and computational resources on the Ethereum blockchain. Notably, the remaining Ether in the contract is sent to a specified address, ensuring a responsible handling of funds.

The contract is not vulnerable to ERC-20 approve Race condition vulnerability.

The ERC-20 race condition arises when two or more transactions attempt to interact with the same ERC-20 token contract concurrently. This scenario can result in conflicts and unexpected behavior due to the non-atomic nature of certain operations in the contract. Atomicity refers to the concept that an operation is indivisible and occurs as a single, uninterruptible unit.

The contract's owner was found.

Renounced ownership indicates that the contract is truly decentralized, as the owner has relinquished control, ensuring that the contract's functionality and rules cannot be altered by administrators or any central authority.

Addresses contain more than 20% of circulating token supply.

Users with token balances exceeding 5% of the circulating token supply are critical to monitor, as their actions can significantly influence the token's price and ecosystem. Proper token distribution helps maintain a healthy market by preventing concentration of power and promoting fair participation.

The contracts are using functions that can only be called by the owners.

An overpowered owner risk occurs when a contract has numerous functions that only the owner can execute. This can lead to centralization issues and potential abuse, as the owner has disproportionate control over the contract's operations.

The contract does not have a cooldown feature.

Cooldown functions, a crucial aspect in the smart contract landscape, are employed to temporarily suspend trading activities or other contract workflows. The mechanism introduces a time-based delay, effectively preventing users from repeatedly executing transactions or engaging in rapid buying and selling of tokens. Cooldown functions are used to halt trading or other contract workflows for a certain amount of time so as to prevent users from repeatedly executing transactions or buying and selling tokens.

Owners can whitelist tokens or users.

This empowers the contract owner to selectively grant privileges to users, such as exemption from fees or access to unique contract features.

Owners can set or update Fees in the contract.

In the context of smart contracts, fees are essential components that may be associated with various functionalities, such as transactions, token transfers, or other specific actions. The ability for owners to set or update fees is particularly valuable in scenarios where fee adjustments are needed to align with market conditions, regulatory requirements, or project-specific considerations. The Owners Can Set or Update Fees module focuses on identifying the capability within a smart contract for owners to establish or modify fees. This feature allows contract owners to have control over the fee structure within the contract, providing flexibility and adaptability to changing circumstances.

Hardcoded addresses were not found.

The inclusion of a fixed or hardcoded address within a smart contract has the potential to pose significant challenges in the future, particularly concerning the contract's adaptability and upgradability. This static reference to an address may impede the seamless implementation of updates or modifications to the contract, hindering its ability to evolve in response to changing requirements. Such rigidity may result in complications and obstacles when attempting to enhance or alter the smart contract's functionality over time.

The contract does not have any owner-controlled functions modifying token balances.

The Owners Updating Token Balance module is focused on identifying situations where a smart contract has functions controlled by owners that allow them to update token balances for other users or the contract. If a contract permits owners to manipulate token balances, it can have significant implications on user holdings and overall contract integrity. In some scenarios, contracts may provide owners with functions that enable the manual adjustment of token balances. While this feature can be legitimate for specific use cases, such as token distribution or rewards, it also introduces potential risks. Allowing owners to arbitrarily update token balances may lead to vulnerabilities, manipulation, or unintended changes in the token ecosystem.

Owner's wallet contains 100000000.0 tokens which is more than 100.0% of the circulating token supply.

A check on the owner's wallet balance exceeding a specific token amount can indicate a centralization risk, where the owner may have disproportionate control over the token supply, potentially leading to manipulation or abuse.

No such functions retrieving ownership were found.

The Function Retrieving Ownership module serves the purpose of swiftly and efficiently retrieving ownership-related information within a smart contract. This functionality is vital for projects seeking to access and manage ownership data seamlessly. Utilizing this module, developers can streamline the process of obtaining ownership details, contributing to the effective administration of ownership-related functions within the ecosystem.

IS SPAM CONTRACT

A spam NFT is an NFT that is considered low-quality or deceptive, cluttering the marketplace and potentially misleading users.

Absence of Malicious Typecasting.

Malicious typecasting, particularly the conversion of uint160 values to addresses, is a tactic often used by scammers to create deceptive addresses that can bypass standard detection mechanisms, facilitating fraudulent activities.

LIQUIDITY BURN STATUS

The liquidity burn status indicates whether the LP tokens for the scanned contract have been permanently removed or remain accessible. If burnt, the LP tokens are sent to an irrecoverable address, ensuring that the liquidity cannot be withdrawn, offering permanent stability and security to the project. If not burnt, the LP tokens could still be accessed and withdrawn, which might expose investors to risks of liquidity manipulation or removal.

LIQUIDITY LOCK STATUS

The liquidity status determines whether the liquidity for the scanned contract is securely locked or accessible. If locked, LP tokens are stored in a time-locked contract, preventing any withdrawals until the lock expires. This helps protect investors from sudden liquidity removal. If not locked, LP tokens remain accessible, allowing project developers or liquidity providers to withdraw liquidity at any time, potentially posing risks to investors.

No such functions having totalSupply function update were found.

A fixed supply token is critical when the token's value is tied to scarcity or when precise control over inflation or deflation is required. Without a fixed supply, the contract could introduce unexpected inflation, devalue the token, or erode trust in the token's consistency.

No such functions having gas abuse via malicious minting.

Gas abuse refers to patterns within smart contracts that manipulate gas consumption in ways that unnecessarily increase transaction costs for users. This can occur through various mechanisms designed to exploit gas inefficiencies or inflate gas usage, shifting the financial burden onto users without their knowledge.

Valid token name or symbol.

The token name or symbol contains potentially harmful content, such as HTML tags or JavaScript code. If these unsanitized strings are displayed by user interfaces, they could execute scripts in users' browsers, posing a significant risk of Cross-Site Scripting (XSS).

No such functions having addresses with special access.

Special permissions granted to non-owner addresses allow them to execute specific functions with elevated access. This can introduce security risks, as these privileged addresses may perform critical operations that impact the contract's state or user funds. If not properly managed or monitored, these permissions could lead to unauthorized or malicious actions, compromising the contract's integrity.

No hidden owner detected

The Hidden Owner check identifies whether there are any hidden owner roles within the contract. Hidden ownership can allow unauthorized access and control over contract functions, which poses a risk to users and stakeholders.

The token is not a counterfeit token

The contract is found to have the token symbol identical to that of official tokens, thereby falling under the category of counterfeit tokens. These counterfeit tokens can mislead users into believing they are interacting with legitimate, well-known cryptocurrencies, potentially leading to financial losses and damaging the reputation of the official token.

Absence of external call risk in critical functions.

This check identifies risks associated with external calls within critical functions. External calls can introduce vulnerabilities such as unexpected state changes, or dependencies on external contracts, which may compromise the integrity and reliability of the function’s execution.

Issue Type

PRESENCE OF MINTING FUNCTION

Action Taken

Pending Fix

Description

Minting functions are often utilized to generate new tokens, which can be allocated to specific addresses, such as user wallets or the contract owner's wallet. This feature is commonly employed in various decentralized finance (DeFi) and non-fungible token (NFT) projects to facilitate token issuance and distribution. The Presence of Minting Function module is designed to quickly identify the presence and implementation of minting functions in a smart contract. Mint functions play a crucial role in creating new tokens and transferring them to the designated user's or owner's wallet. This process significantly contributes to increasing the overall circulation of the tokens within the ecosystem.

Remediation

It is important to ensure the minting function includes proper access control mechanisms to restrict token minting to authorized addresses only, thereby preventing unauthorized or excessive token creation.

File Location	Line No.
contracts/AIB_ERC20.sol	L181 - L190

Issue Type

SOLIDITY PRAGMA VERSION

Action Taken

Pending Fix

Description

The contract should be written using the latest Solidity pragma version as it comes with numerous bug fixes. Utilizing an outdated version exposes the contract to vulnerabilities associated with known issues that have been addressed in subsequent updates. Therefore, it is essential to stay current with the latest Solidity version to ensure the robustness and security of the contract against potential vulnerabilities.

Remediation

Update the Solidity pragma version to the latest stable version to benefit from the latest bug fixes and security enhancements.

File Location	Line No.
contracts/AIB_ERC20.sol	L2 - L2

Issue Type

OWNERS CANNOT BLACKLIST TOKENS OR USERS

Action Taken

Pending Fix

Description

This module is designed to identify whether the owner of a smart contract has the capability to blacklist specific tokens or users. In a scenario where owners possess the authority to blacklist, all transactions related to the blacklisted entities will be immediately halted. Ownership privileges that include the ability to blacklist tokens or users can be a critical feature in certain use cases, providing the owner with control over potential malicious activities, compliance issues, or other concerns. However, in situations where this authority is abused or misapplied, it can lead to unintended consequences and user dissatisfaction.

Remediation

Ensure that ownership privileges do not include the ability to blacklist tokens or users, as this feature can lead to centralization concerns and potential misuse. Instead, implement transparent governance mechanisms for handling compliance issues or malicious activities.

File Location	Line No.
contracts/AIB_ERC20.sol	L7 - L7

Issue Type

CRITICAL ADMINISTRATIVE FUNCTIONS

Action Taken

Pending Fix

Description

A smart contract within the Web3 ecosystem that incorporates critical administrative functions can potentially compromise the transparency and intended objectives of the contract. It is imperative to conduct a thorough examination of these functions, especially in the realm of Web3 smart contracts. Minimizing administrative functions in a token contract within the Web3 framework can significantly reduce the likelihood of complications and enhance overall efficiency and clarity.

Remediation

Review and minimize critical administrative functions within the smart contract to only include essential functionalities necessary for contract management and operation.

File Location	Line No.
contracts/AIB_ERC20.sol	L5 - L5

Issue Type

OVERPOWERED OWNERS

Action Taken

Pending Fix

Description

An overpowered owner risk occurs when a contract has numerous functions that only the owner can execute. This can lead to centralization issues and potential abuse, as the owner has disproportionate control over the contract's operations.

Remediation

Review and minimize the number of critical functions accessible to owners, ensuring that these functions are necessary for contract management and do not pose undue risk to users' funds in the event of compromise or misuse. Implement multi-signature or governance mechanisms for critical actions to distribute authority and mitigate risk.

File Location	Line No.
contracts/AIB_ERC20.sol	L92 - L104
contracts/AIB_ERC20.sol	L106 - L112
contracts/AIB_ERC20.sol	L114 - L118
contracts/AIB_ERC20.sol	L120 - L123
contracts/AIB_ERC20.sol	L125 - L128
contracts/AIB_ERC20.sol	L130 - L136
contracts/AIB_ERC20.sol	L138 - L144

File Location	Line No.
contracts/AIB_ERC20.sol	L147 - L149
contracts/AIB_ERC20.sol	L151 - L153
contracts/AIB_ERC20.sol	L173 - L179

Issue Type

OWNERS WHITELISTING TOKENS/USERS

Action Taken

Pending Fix

Description

This empowers the contract owner to selectively grant privileges to users, such as exemption from fees or access to unique contract features.

Remediation

Ensure that the contract owner's ability to whitelist tokens or users is limited to necessary and legitimate use cases, such as granting special privileges or access to specific features, and implement strict access controls to prevent unauthorized whitelisting actions.

File Location	Line No.
contracts/AIB_ERC20.sol	L7 - L7

Issue Type

OWNERS CAN SET/UPDATE FEES

Action Taken

Pending Fix

Description

In the context of smart contracts, fees are essential components that may be associated with various functionalities, such as transactions, token transfers, or other specific actions. The ability for owners to set or update fees is particularly valuable in scenarios where fee adjustments are needed to align with market conditions, regulatory requirements, or project-specific considerations. The Owners Can Set or Update Fees module focuses on identifying the capability within a smart contract for owners to establish or modify fees. This feature allows contract owners to have control over the fee structure within the contract, providing flexibility and adaptability to changing circumstances.

Remediation

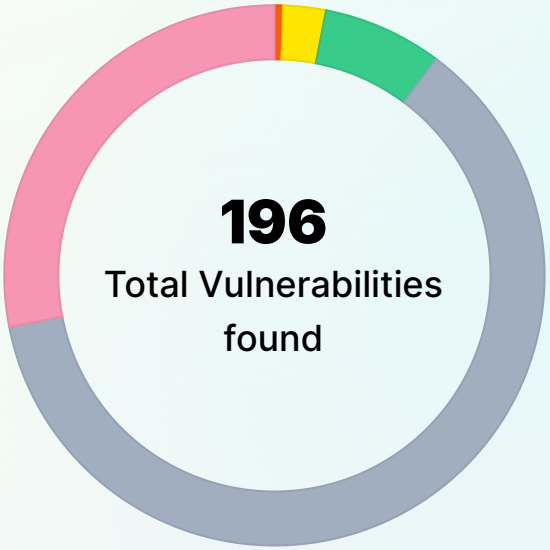
To remove or disable the functions that allow the contract owner to update fees, ensuring that fee structures remain fixed and cannot be altered post-deployment, you would typically need to modify the smart contract code itself. Specifically, you would locate the functions responsible for setting or updating fees and either remove them entirely or modify them to prevent changes.

File Location	Line No.
contracts/AIB_ERC20.sol	L138 - L144

04. Findings Summary

0×4fb0112f59DF1aa06B1e099fD624D479cfDc9789
BINANCE (Bsc Mainnet) | View on Bscscan

Security Score	Scan duration	Lines of code
67.38/100	3 secs	201



0	1	5	14	121	55
Crit	High	Med	Low	Info	Gas

This audit report has not been verified by the SolidityScan team. To learn more about our published reports. click here

ACTION TAKEN

0	0	0	197
Fixed	False Positive	Won't Fix	Pending Fix

S. No.	Severity	Bug Type	Instances	Detection Method	Status
H001	High	UNCHECKED TRANSFER	1	Automated	Pending Fix
M001	Medium	PRECISION LOSS DURING DIVISION BY LARGE NUMBERS	4	Automated	Pending Fix
M002	Medium	USING EXTCODESIZE TO CHECK FOR EXTERNALLY OWNED ACCOUNTS	1	Automated	Pending Fix
L001	Low	USE OF FLOATING PRAGMA	1	Automated	Pending Fix
L002	Low	LACK OF ZERO VALUE CHECK IN TOKEN TRANSFERS	1	Automated	Pending Fix
L003	Low	MISSING EVENTS	4	Automated	Pending Fix
L004	Low	MISSING ZERO ADDRESS VALIDATION	6	Automated	Pending Fix
L005	Low	OUTDATED COMPILER VERSION	1	Automated	Pending Fix
L006	Low	USE OWNABLE2STEP	1	Automated	Pending Fix

S. No.	Severity	Bug Type	Instances	Detection Method	Status
I001	Informational	CONTRACT NAME SHOULD USE PASCALCASE	1	Automated	*Pending Fix*
I002	Informational	MISSING @AUTHOR IN NATSPEC COMMENTS FOR CONTRACT DECLARATION	1	Automated	*Pending Fix*
I003	Informational	MISSING @DEV IN NATSPEC COMMENTS FOR CONTRACT DECLARATION	1	Automated	*Pending Fix*
I004	Informational	MISSING @DEV IN NATSPEC COMMENTS FOR FUNCTIONS	22	Automated	*Pending Fix*
I005	Informational	MISSING INDEXED KEYWORDS IN EVENTS	1	Automated	*Pending Fix*
I006	Informational	MISSING @INHERITDOC ON OVERRIDE FUNCTIONS	22	Automated	*Pending Fix*
I007	Informational	MISSING NATSPEC COMMENTS IN ASSEMBLY BLOCKS	1	Automated	*Pending Fix*
I008	Informational	MISSING NATSPEC COMMENTS IN SCOPE BLOCKS	19	Automated	*Pending Fix*
I009	Informational	MISSING NATSPEC DESCRIPTIONS FOR PUBLIC VARIABLE DECLARATIONS	14	Automated	*Pending Fix*
I010	Informational	MISSING @NOTICE IN NATSPEC COMMENTS FOR CONSTRUCTORS	1	Automated	*Pending Fix*
I011	Informational	MISSING @NOTICE IN NATSPEC COMMENTS FOR FUNCTIONS	22	Automated	*Pending Fix*

S. No.	Severity	Bug Type	Instances	Detection Method	Status
I012	Informational	MISSING UNDERSCORE IN NAMING VARIABLES	5	Automated	*Pending Fix*
I013	Informational	NAME MAPPING PARAMETERS	2	Automated	*Pending Fix*
I014	Informational	REVERT STATEMENTS WITHIN EXTERNAL AND PUBLIC FUNCTIONS CAN BE USED TO PERFORM DOS ATTACKS	8	Automated	*Pending Fix*
I015	Informational	USE CALL INSTEAD OF TRANSFER OR SEND	1	Automated	*Pending Fix*
I016	Informational	USE SCIENTIFIC NOTATION	1	Automated	*Pending Fix*
I017	Informational	IN-LINE ASSEMBLY DETECTED	1	Automated	*Pending Fix*
G001	Gas	AVOID RE-STORING VALUES	6	Automated	*Pending Fix*
G002	Gas	AVOID ZERO-TO-ONE STORAGE WRITES	4	Automated	*Pending Fix*
G003	Gas	CACHE ADDRESS(THIS) WHEN USED MORE THAN ONCE	3	Automated	*Pending Fix*
G004	Gas	CHEAPER CONDITIONAL OPERATORS	1	Automated	*Pending Fix*
G005	Gas	CHEAPER INEQUALITIES IN IF()	5	Automated	*Pending Fix*
G006	Gas	DEFAULT INT VALUES ARE MANUALLY RESET	2	Automated	*Pending Fix*
G007	Gas	DEFINE CONSTRUCTOR AS PAYABLE	1	Automated	*Pending Fix*
G008	Gas	REVERTING FUNCTIONS CAN BE PAYABLE	10	Automated	*Pending Fix*
G009	Gas	GAS INEFFICIENCY DUE TO MULTIPLE OPERANDS IN SINGLE IF/ELSEIF CONDITION	9	Automated	*Pending Fix*
G010	Gas	OPTIMIZING ADDRESS ID MAPPING	2	Automated	*Pending Fix*
G011	Gas	PUBLIC CONSTANTS CAN BE PRIVATE	1	Automated	*Pending Fix*

S. No.	Severity	Bug Type	Instances	Detection Method	Status
G012	Gas	SPLITTING REVERT STATEMENTS	6	Automated	*Pending Fix*
G013	Gas	STORAGE VARIABLE CACHING IN MEMORY	4	Automated	*Pending Fix*
G014	Gas	UNNECESSARY DEFAULT VALUE INITIALIZATION	1	Automated	*Pending Fix*
G015	Gas	USE SELFBALANCE() INSTEAD OF ADDRESS(THIS).BALANCE	1	Automated	*Pending Fix*

05. Vulnerability Details

Issue Type

UNCHECKED TRANSFER

S. No.	Severity	Detection Method	Instances
H001	High	Automated	1

Bug ID	File Location	Line No.	Action Taken
SSB_4658143_85	--	--	*Pending Fix*

Upgrade your Plan to view the full report

1 High Issues Found

Please upgrade your plan to view all the issues in your report.

Upgrade

Issue Type

PRECISION LOSS DURING DIVISION BY LARGE NUMBERS

S. No.	Severity	Detection Method	Instances
M001	Medium	Automated	4

Bug ID	File Location	Line No.	Action Taken
SSB_4658143_152	--	--	*Pending Fix*
SSB_4658143_153	--	--	*Pending Fix*
SSB_4658143_154	--	--	*Pending Fix*
SSB_4658143_155	--	--	*Pending Fix*

Issue Type

USE OF FLOATING PRAGMA

S. No.	Severity	Detection Method	Instances
L001	Low	Automated	1

Bug ID	File Location	Line No.	Action Taken
SSB_4658143_101	--	--	*Pending Fix*

Upgrade your Plan to view the full report

1 Low Issues Found

Please upgrade your plan to view all the issues in your report.

Upgrade

Issue Type

CONTRACT NAME SHOULD USE PASCALCASE

S. No.	Severity	Detection Method	Instances
I001	Informational	Automated	1

Bug ID	File Location	Line No.	Action Taken
SSB_4658143_84	--	--	*Pending Fix*

Upgrade your Plan to view the full report

1 Informational Issues Found

Please upgrade your plan to view all the issues in your report.

Upgrade

Issue Type

AVOID RE-STORING VALUES

S. No.	Severity	Detection Method	Instances
G001	Gas	Automated	6

Description

The function is found to be allowing re-storing the value in the contract's state variable even when the old value is eq ual to the new value. This practice results in unnecessary gas consumption due to the `Gsreset` operation (2900 ga s), which could be avoided. If the old value and the new value are the same, not updating the storage would avoid thi s cost and could instead incur a `Gcoldsload` (2100 gas) or a `Gwarmaccess` (100 gas), potentially saving gas.

Bug ID	File Location	Line No.	Action Taken
SSB_4658143_124	contracts/AIB_ERC20.sol	L92 - L104	*Pending Fix*
SSB_4658143_125	contracts/AIB_ERC20.sol	L106 - L112	*Pending Fix*
SSB_4658143_126	contracts/AIB_ERC20.sol	L125 - L128	*Pending Fix*
SSB_4658143_127	contracts/AIB_ERC20.sol	L130 - L136	*Pending Fix*
SSB_4658143_128	contracts/AIB_ERC20.sol	L138 - L144	*Pending Fix*
SSB_4658143_129	contracts/AIB_ERC20.sol	L173 - L179	*Pending Fix*

Issue Type

AVOID ZERO-TO-ONE STORAGE WRITES

S. No.	Severity	Detection Method	Instances
G002	Gas	Automated	4

Description

Writing a storage variable from zero to a non-zero value costs 22,100 gas (20,000 for the write and 2,100 for cold access), making it one of the most expensive operations. This is why patterns like OpenZeppelin's `ReentrancyGuard` use `1` and `2` instead of `0` and `1`—to avoid the high cost of zero-to-non-zero writes. Non-zero to non-zero updates cost only 5,000 gas.

Bug ID	File Location	Line No.	Action Taken
SSB_4658143_183	contracts/AIB_ERC20.sol	L100 - L100	*Pending Fix*
SSB_4658143_184	contracts/AIB_ERC20.sol	L101 - L101	*Pending Fix*
SSB_4658143_185	contracts/AIB_ERC20.sol	L110 - L110	*Pending Fix*
SSB_4658143_186	contracts/AIB_ERC20.sol	L126 - L126	*Pending Fix*

Issue Type

CACHE ADDRESS(THIS) WHEN USED MORE THAN ONCE

S. No.	Severity	Detection Method	Instances
G003	Gas	Automated	3

Description

The repeated usage of `address(this)` within the contract could result in increased gas costs due to multiple executions of the same computation, potentially impacting efficiency and overall transaction expenses.

Bug ID	File Location	Line No.	Action Taken
SSB_4658143_29	contracts/AIB_ERC20.sol	L61 - L61	*Pending Fix*
SSB_4658143_30	contracts/AIB_ERC20.sol	L74 - L74	*Pending Fix*
SSB_4658143_31	contracts/AIB_ERC20.sol	L152 - L152	*Pending Fix*

Issue Type

CHEAPER CONDITIONAL OPERATORS

S. No.	Severity	Detection Method	Instances
G004	Gas	Automated	1

Description

During compilation, `x != 0` is cheaper than `x > 0` for unsigned integers in solidity inside conditional statements.

Bug ID	File Location	Line No.	Action Taken
SSB_4658143_68	contracts/AIB_ERC20.sol	L84 - L84	*Pending Fix*

Issue Type

CHEAPER INEQUALITIES IN IF()

S. No.	Severity	Detection Method	Instances
G005	Gas	Automated	5

Description

The contract was found to be doing comparisons using inequalities inside the if statement.
When inside the `if` statements, non-strict inequalities `(>=, <=)` are usually cheaper than the strict equalities `(>, <)`.

Bug ID	File Location	Line No.	Action Taken
SSB_4658143_20	contracts/AIB_ERC20.sol	L84 - L84	*Pending Fix*
SSB_4658143_21	contracts/AIB_ERC20.sol	L93 - L93	*Pending Fix*
SSB_4658143_22	contracts/AIB_ERC20.sol	L96 - L96	*Pending Fix*
SSB_4658143_23	contracts/AIB_ERC20.sol	L107 - L107	*Pending Fix*
SSB_4658143_24	contracts/AIB_ERC20.sol	L217 - L217	*Pending Fix*

Issue Type

DEFAULT INT VALUES ARE MANUALLY RESET

S. No.	Severity	Detection Method	Instances
G006	Gas	Automated	2

Description

The contract is found to inefficiently reset integer variables to their default value of zero using manual assignment. In Solidity, manually setting a variable to its default value does not free up storage space, leading to unnecessary gas c onsumption. Instead, using the `.delete` keyword can achieve the same result while also freeing up storage space on the Ethereum blockchain, resulting in gas cost savings.

Bug ID	File Location	Line No.	Action Taken
SSB_4658143_27	contracts/AIB_ERC20.sol	L116 - L116	*Pending Fix*
SSB_4658143_28	contracts/AIB_ERC20.sol	L121 - L121	*Pending Fix*

Issue Type

DEFINE CONSTRUCTOR AS PAYABLE

S. No.	Severity	Detection Method	Instances
G007	Gas	Automated	1

Description

Developers can save around 10 opcodes and some gas if the constructors are defined as payable.
However, it should be noted that it comes with risks because payable constructors can accept ETH during deployment.

Bug ID	File Location	Line No.	Action Taken
SSB_4658143_100	contracts/AIB_ERC20.sol	L47 - L63	*Pending Fix*

Issue Type

REVERTING FUNCTIONS CAN BE PAYABLE

S. No.	Severity	Detection Method	Instances
G008	Gas	Automated	10

Description

If a function modifier such as `onlyOwner` is used, the function will revert if a normal user tries to pay the function. Marking the function as payable will lower the gas cost for legitimate callers because the compiler will not include checks for whether a payment was provided.

Bug ID	File Location	Line No.	Action Taken
SSB_4658143_32	contracts/AIB_ERC20.sol	L92 - L104	*Pending Fix*
SSB_4658143_33	contracts/AIB_ERC20.sol	L106 - L112	*Pending Fix*
SSB_4658143_34	contracts/AIB_ERC20.sol	L114 - L118	*Pending Fix*
SSB_4658143_35	contracts/AIB_ERC20.sol	L120 - L123	*Pending Fix*
SSB_4658143_36	contracts/AIB_ERC20.sol	L125 - L128	*Pending Fix*
SSB_4658143_37	contracts/AIB_ERC20.sol	L130 - L136	*Pending Fix*
SSB_4658143_38	contracts/AIB_ERC20.sol	L138 - L144	*Pending Fix*
SSB_4658143_39	contracts/AIB_ERC20.sol	L147 - L149	*Pending Fix*
SSB_4658143_40	contracts/AIB_ERC20.sol	L151 - L153	*Pending Fix*
SSB_4658143_41	contracts/AIB_ERC20.sol	L173 - L179	*Pending Fix*

Issue Type

GAS INEFFICIENCY DUE TO MULTIPLE OPERANDS IN SINGLE IF/ELSEIF CONDITION

S. No.	Severity	Detection Method	Instances
G009	Gas	Automated	9

Description

The contract is found to use multiple operands within a single `if` or `else if` statement, which can lead to unnecessary gas consumption due to the way the EVM evaluates compound boolean expressions. Each operand in a compound condition is evaluated even if the first condition fails, unless short-circuiting occurs, and the combined logic can result in more complex bytecode and higher gas usage compared to using nested `if` statements. This inefficiency is particularly relevant in functions that are called frequently or within loops.

Bug ID	File Location	Line No.	Action Taken
SSB_4658143_188	contracts/AIB_ERC20.sol	L93 - L95	*Pending Fix*
SSB_4658143_189	contracts/AIB_ERC20.sol	L96 - L98	*Pending Fix*
SSB_4658143_190	contracts/AIB_ERC20.sol	L107 - L109	*Pending Fix*
SSB_4658143_191	contracts/AIB_ERC20.sol	L193 - L195	*Pending Fix*
SSB_4658143_192	contracts/AIB_ERC20.sol	L196 - L198	*Pending Fix*
SSB_4658143_193	contracts/AIB_ERC20.sol	L202 - L204	*Pending Fix*
SSB_4658143_194	contracts/AIB_ERC20.sol	L206 - L208	*Pending Fix*
SSB_4658143_195	contracts/AIB_ERC20.sol	L212 - L214	*Pending Fix*

Bug ID	File Location	Line No.	Action Taken
SSB_4658143_196	contracts/AIB_ERC20.sol	L227 - L229	⚠ *Pending Fix*

Issue Type

OPTIMIZING ADDRESS ID MAPPING

S. No.	Severity	Detection Method	Instances
G010	Gas	Automated	2

Description

Combining multiple address/ID mappings into a single mapping using a struct enhances storage efficiency, simplifies code, and reduces gas costs, resulting in a more streamlined and cost-effective smart contract design.
It saves storage slot for the mapping and depending on the circumstances and sizes of types, it can avoid a Gsset (2 0000 gas) per mapping combined. Reads and subsequent writes can also be cheaper when a function requires both values and they fit in the same storage slot.

Bug ID	File Location	Line No.	Action Taken
SSB_4658143_42	contracts/AIB_ERC20.sol	L14 - L14	*Pending Fix*
SSB_4658143_43	contracts/AIB_ERC20.sol	L15 - L15	*Pending Fix*

Issue Type

PUBLIC CONSTANTS CAN BE PRIVATE

S. No.	Severity	Detection Method	Instances
G011	Gas	Automated	1

Description

Public constant variables cost more gas because the EVM automatically creates getter functions for them and adds entries to the method ID table. The values can be read from the source code instead.

Bug ID	File Location	Line No.	Action Taken
SSB_4658143_83	contracts/AIB_ERC20.sol	L26 - L26	*Pending Fix*

Issue Type

SPLITTING REVERT STATEMENTS

S. No.	Severity	Detection Method	Instances
G012	Gas	Automated	6

Description

The contract is using multiple conditions in a single `if` statement followed by a revert. This costs some extra gas.

Bug ID	File Location	Line No.	Action Taken
SSB_4658143_55	contracts/AIB_ERC20.sol	L93 - L95	*Pending Fix*
SSB_4658143_56	contracts/AIB_ERC20.sol	L96 - L98	*Pending Fix*
SSB_4658143_57	contracts/AIB_ERC20.sol	L107 - L109	*Pending Fix*
SSB_4658143_58	contracts/AIB_ERC20.sol	L193 - L195	*Pending Fix*
SSB_4658143_59	contracts/AIB_ERC20.sol	L196 - L198	*Pending Fix*
SSB_4658143_60	contracts/AIB_ERC20.sol	L206 - L208	*Pending Fix*

Issue Type

STORAGE VARIABLE CACHING IN MEMORY

S. No.	Severity	Detection Method	Instances
G013	Gas	Automated	4

Description

The contract is using the state variable multiple times in the function.
`SLOADs` are expensive (100 gas after the 1st one) compared to `MLOAD` / `MSTORE` (3 gas each).

Bug ID	File Location	Line No.	Action Taken
SSB_4658143_51	contracts/AIB_ERC20.sol	L92 - L104	*Pending Fix*
SSB_4658143_51	contracts/AIB_ERC20.sol	L92 - L104	*Pending Fix*
SSB_4658143_52	contracts/AIB_ERC20.sol	L192 - L199	*Pending Fix*
SSB_4658143_53	contracts/AIB_ERC20.sol	L222 - L244	*Pending Fix*

Issue Type

UNNECESSARY DEFAULT VALUE INITIALIZATION

S. No.	Severity	Detection Method	Instances
G014	Gas	Automated	1

Description

The contract was found to be initializing the value of the variable to it’s default value.
This is redundant and not required.

Bug ID	File Location	Line No.	Action Taken
SSB_4658143_182	contracts/AIB_ERC20.sol	L24 - L24	*Pending Fix*

Issue Type

USE SELFBALANCE() INSTEAD OF ADDRESS(THIS).BALANCE

S. No.	Severity	Detection Method	Instances
G015	Gas	Automated	1

Description

In Solidity, efficient use of gas is paramount to ensure cost-effective execution on the Ethereum blockchain. Gas can be optimized when obtaining contract balance by using `selfbalance()` rather than `address(this).balance` because it bypasses gas costs and refunds, which are not required for obtaining the contract's balance.

Bug ID	File Location	Line No.	Action Taken
SSB_4658143_187	contracts/AIB_ERC20.sol	L152 - L152	*Pending Fix*

06. Scan History

Critical High Medium Low Informational Gas

No	Date	Security Score	Scan Overview
1.	2025-10-31	**67.38**	0 1 5 14 121 55

07. Disclaimer

The Reports neither endorse nor condemn any specific project or team, nor do they guarantee the security of any specific project. The contents of this report do not, and should not be interpreted as having any bearing on, the economics of tokens, token sales, or any other goods, services, or assets.

The security audit is not meant to replace functional testing done before a software release.

There is no warranty that all possible security issues of a particular smart contract(s) will be found by the tool, i.e., It is not guaranteed that there will not be any further findings based solely on the results of this evaluation.

Emerging technologies such as Smart Contracts and Solidity carry a high level of technical risk and uncertainty. There is no warranty or representation made by this report to any Third Party in regards to the quality of code, the business model or the proprietors of any such business model, or the legal compliance of any business.

In no way should a third party use these reports to make any decisions about buying or selling a token, product, service, or any other asset. It should be noted that this report is not investment advice, is not intended to be relied on as investment advice, and has no endorsement of this project or team. It does not serve as a guarantee as to the project's absolute security.

The assessment provided by SolidityScan is subject to dependencies and under continuing development. 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. SolidityScan owes no duty to any third party by virtue of publishing these Reports.

As one audit-based assessment cannot be considered comprehensive, we always recommend proceeding with several independent manual audits including manual audit and a public bug bounty program to ensure the security of the smart contracts.