SolidityScan | Powered by CRED SHIELDS

Security Assessment Report

BricaToken

5 Feb 2026

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

Table of Contents

AVOID RE-STORING VALUES
AVOID ZERO-TO-ONE STORAGE WRITES
BYTES CONSTANT MORE EFFICIENT THAN STRING LITERAL
CACHE ADDRESS(THIS) WHEN USED MORE THAN ONCE
CHEAPER CONDITIONAL OPERATORS
CHEAPER INEQUALITIES IN REQUIRE()
DEFINE CONSTRUCTOR AS PAYABLE
FUNCTIONS CAN BE IN-LINED
REVERTING FUNCTIONS CAN BE PAYABLE
GAS OPTIMIZATION FOR STATE VARIABLES
OPTIMIZING ADDRESS ID MAPPING
PUBLIC CONSTANTS CAN BE PRIVATE
SMALLER DATA TYPES COST MORE
USE SELFBALANCE() INSTEAD OF ADDRESS(THIS).BALANCE
VARIABLES DECLARED BUT NEVER USED

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

BricaToken

0×00C0614F8157b867a67d1A2F98F7e3600634C901

https://bscscan.com/address/0×00C0614F8157b867a67d1A2F9...

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

Website	Publishers/Owner Name	Organization
-	-	-

Contact Email

-

Security Score is GREAT

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 BricaToken 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 BricaToken introduces new features or refactors the code.

03. Threat Summary

Threat Score

High Risk

33.1/100

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 cannot 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 cannot 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 not 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 not 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 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 cannot 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 cannot 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 999999989.0 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.

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.

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 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.

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.

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.

The contract is not a honeypot.

A token is honeypot when it exhibits the typical characteristics designed to trap investors’ funds while preventing them from profiting. Common signs include the inability to sell tokens once purchased, restrictive transfer functions coded into the smart contract, excessive or hidden taxes on transactions, or permissions that allow only the contract owner to execute sells.

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.
contract.sol	L6 - L6

Issue Type

ERC20 RACE CONDITION

Action Taken

⚠ *Pending Fix*

Description

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.

Remediation

Implement locking mechanisms or state variables to ensure that only one transaction can modify the token balances or allowances at a time, thereby preventing the race condition.

File Location	Line No.
contract.sol	L87 - L90

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.
contract.sol	L123 - L129
contract.sol	L134 - L143
contract.sol	L149 - L152

Issue Type

FUNCTION RETRIEVING OWNERSHIP

Action Taken

⚠ *Pending Fix*

Description

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.

Remediation

To enhance the security and management of ownership within the smart contract, it is recommended to introduce an ownership transfer mechanism that securely transfers ownership to another address. Additionally, the relinquishOwnership function should be modified to prevent setting the owner to the zero address, which can lead to the contract being left without an owner. Implementing these changes will ensure that ownership-related functions remain secure and manageable within the ecosystem.

File Location	Line No.
contract.sol	L151 - L151

Issue Type

TOKEN SUPPLY NOT FIXED

Action Taken

Pending Fix

Description

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.

Remediation

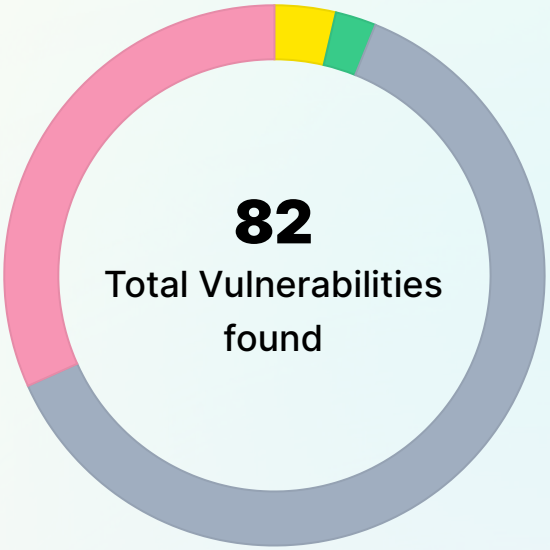
To remediate this issue, ensure that the total supply is assigned a fixed value within the constructor and that no functions within the contract can change this total supply after deployment. Remove or thoroughly secure any minting or burning capabilities to prevent unauthorized changes to the total supply.

File Location	Line No.
contract.sol	L105 - L114

04. Findings Summary

0×00C0614F8157b867a67d1A2F98F7e3600634C901
BINANCE (Bsc Mainnet) | View on Bscscan

Security Score	Scan duration	Lines of code
89.00/100	**9 secs**	**140**



0	0	3	2	51	26
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	1	0	82
Fixed	False Positive	Won't Fix	Pending Fix

S. No.	Severity	Bug Type	Instances	Detection Method	Status
M001	Medium	ACCOUNT EXISTENCE CHECK FOR LOW LEVEL CALLS	1	Automated	Pending Fix
M002	Medium	APPROVE FRONT-RUNNING ATTACK	1	Automated	Pending Fix
M003	Medium	UNVALIDATED CALL DATA IN TRANSFER FUNCTION USING .CALL	1	Automated	Pending Fix
L001	Low	USE OF FLOATING PRAGMA	1	Automated	Pending Fix
L002	Low	OUTDATED COMPILER VERSION	1	Automated	Pending Fix
I001	Informational	MISSING @AUTHOR IN NATSPEC COMMENTS FOR CONTRACT DECLARATION	1	Automated	Pending Fix
I002	Informational	MISSING @DEV IN NATSPEC COMMENTS FOR CONTRACT DECLARATION	1	Automated	Pending Fix
I003	Informational	MISSING @DEV IN NATSPEC COMMENTS FOR FUNCTIONS	12	Automated	Pending Fix
I004	Informational	MISSING @INHERITDOC ON OVERRIDE FUNCTIONS	11	Automated	Pending Fix
I005	Informational	MISSING NATSPEC DESCRIPTIONS FOR PUBLIC VARIABLE DECLARATIONS	5	Automated	Pending Fix
I006	Informational	MISSING @NOTICE IN NATSPEC COMMENTS FOR CONSTRUCTORS	1	Automated	Pending Fix
I007	Informational	MISSING @NOTICE IN NATSPEC COMMENTS FOR FUNCTIONS	10	Automated	Pending Fix

S. No.	Severity	Bug Type	Instances	Detection Method	Status
I008	Informational	MISSING @NOTICE IN NATSPEC COMMENTS FOR MODIFIERS	1	Automated	*Pending Fix*
I009	Informational	MISSING @PARAM IN NATSPEC COMMENTS FOR MODIFIERS	1	Automated	*Pending Fix*
I010	Informational	MISSING UNDERSCORE IN NAMING VARIABLES	2	Automated	*Pending Fix*
I011	Informational	NAME MAPPING PARAMETERS	2	Automated	*Pending Fix*
I012	Informational	UNUSED RECEIVE FALLBACK	1	Automated	*Pending Fix*
I013	Informational	USE CALL INSTEAD OF TRANSFER OR SEND	1	Automated	*Pending Fix*
I014	Informational	USE SCIENTIFIC NOTATION	1	Automated	*Pending Fix*
I015	Informational	VARIABLES SHOULD BE IMMUTABLE	1	Automated	*Pending Fix*
G001	Gas	AVOID RE-STORING VALUES	1	Automated	*Pending Fix*
G002	Gas	AVOID ZERO-TO-ONE STORAGE WRITES	1	Automated	*Pending Fix*
G003	Gas	BYTES CONSTANT MORE EFFICIENT THAN STRING LITERAL	2	Automated	*Pending Fix*
G004	Gas	CACHE ADDRESS(THIS) WHEN USED MORE THAN ONCE	2	Automated	*Pending Fix*
G005	Gas	CHEAPER CONDITIONAL OPERATORS	1	Automated	*Pending Fix*
G006	Gas	CHEAPER INEQUALITIES IN REQUIRE()	3	Automated	*Pending Fix*
G007	Gas	DEFINE CONSTRUCTOR AS PAYABLE	1	Automated	*Pending Fix*
G008	Gas	FUNCTIONS CAN BE IN-LINED	1	Automated	*Pending Fix*
G009	Gas	REVERTING FUNCTIONS CAN BE PAYABLE	3	Automated	*Pending Fix*
G010	Gas	GAS OPTIMIZATION FOR STATE VARIABLES	2	Automated	*Pending Fix*

S. No.	Severity	Bug Type	Instances	Detection Method	Status
G011	Gas	OPTIMIZING ADDRESS ID MAPPING	2	Automated	*Pending Fix*
G012	Gas	PUBLIC CONSTANTS CAN BE PRIVATE	3	Automated	*Pending Fix*
G013	Gas	SMALLER DATA TYPES COST MORE	1	Automated	*Pending Fix*
G014	Gas	USE SELFBALANCE() INSTEAD OF ADDRESS(THIS).BALANCE	1	Automated	*Pending Fix*
G015	Gas	VARIABLES DECLARED BUT NEVER USED	2	Automated	*Pending Fix*

05. Vulnerability Details

Issue Type

ACCOUNT EXISTENCE CHECK FOR LOW LEVEL CALLS

S. No.	Severity	Detection Method	Instances
M001	Medium	Automated	1

Description

The low-level calls such as the `delegatecall`, `call`, or `callcode`, do not validate prior to the call if the destination acc ount exists or not. They will always return true even if the account is non-existent, therefore, giving invalid output.

Bug ID	File Location	Line No.	Action Taken
SSB_5029366_3	contract.sol	L137 - L137	*Pending Fix*

Issue Type

APPROVE FRONT-RUNNING ATTACK

S. No.	Severity	Detection Method	Instances
M002	Medium	Automated	1

Description

The method overrides the current allowance regardless of whether the spender already used it or not, so there is no way to increase or decrease allowance by a certain value atomically unless the token owner is a smart contract, not an account.
This can be abused by a token receiver when they try to withdraw certain tokens from the sender's account. Meanwhile, if the sender decides to change the amount and sends another `approve` transaction, the receiver can n otice this transaction before it's mined and can extract tokens from both transactions, therefore, ending up with toke ns from both the transactions. This is a front-running attack affecting the `ERC20 Approve` function.

Bug ID	File Location	Line No.	Action Taken
SSB_5029366_2	contract.sol	L87 - L90	*Pending Fix*

Issue Type

UNVALIDATED CALL DATA IN TRANSFER FUNCTION USING .CALL

S. No.	Severity	Detection Method	Instances
M003	Medium	Automated	1

Description

The contract is found to be using `.call` for calling `transfer`, but the data is not checked for success correctly. This lack of verification can lead to potential vulnerabilities, as the success of the call is not adequately confirmed. Without proper checks, there is a risk that the transfer operation may fail silently, which could lead to unexpected behaviors or security issues within the contract. To ensure safety, it is essential to check the call data length and decode the data to confirm the success of the call.

Bug ID	File Location	Line No.	Action Taken
SSB_5029366_15	contract.sol	L137 - L139	*Pending Fix*

Issue Type

USE OF FLOATING PRAGMA

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

Description

Solidity source files indicate the versions of the compiler they can be compiled with using a pragma directive at the t op of the solidity file. This can either be a floating pragma or a specific compiler version.
The contract was found to be using a floating pragma which is not considered safe as it can be compiled with all the versions described.

Bug ID	File Location	Line No.	Action Taken
SSB_5029366_10	contract.sol	L6 - L6	*Pending Fix*

Issue Type

OUTDATED COMPILER VERSION

S. No.	Severity	Detection Method	Instances
L002	Low	Automated	1

Description

Using an outdated compiler version can be problematic especially if there are publicly disclosed bugs and issues that affect the current compiler version.

Bug ID	File Location	Line No.	Action Taken
SSB_5029366_17	contract.sol	L6 - L6	*Pending Fix*

Issue Type

MISSING @AUTHOR IN NATSPEC COMMENTS FOR CONTRACT DECLARATION

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

Description

Natspec `@author` tags are missing from contract declarations in the code. This reduces code clarity and makes it difficult to determine the original author or organization responsible for the contract. The absence of an `@author` tag can lead to challenges in tracking ownership, verifying authenticity, and maintaining proper documentation.

Bug ID	File Location	Line No.	Action Taken
SSB_5029366_55	contract.sol	L14 - L184	*Pending Fix*

Issue Type

MISSING @DEV IN NATSPEC COMMENTS FOR CONTRACT DECLARATION

S. No.	Severity	Detection Method	Instances
I002	Informational	Automated	1

Description

Natspec `@dev` tags are missing from contract declarations in the code. This reduces code clarity and makes it difficult for developers to understand the contract's internal logic, implementation details, and potential caveats.

Bug ID	File Location	Line No.	Action Taken
SSB_5029366_16	contract.sol	L14 - L184	*Pending Fix*

Issue Type

MISSING @DEV IN NATSPEC COMMENTS FOR FUNCTIONS

S. No.	Severity	Detection Method	Instances
I003	Informational	Automated	12

Description

Natspec `@dev` comments are missing from a function declaration in the code. The `@dev` tag provides a detailed ex planation of a function's purpose, assumptions, and behavior. Lack of `@dev` documentation can: reduce code maint ainability, making it harder for developers to understand the function's implementation details.

Bug ID	File Location	Line No.	Action Taken
SSB_5029366_60	contract.sol	L74 - L76	*Pending Fix*
SSB_5029366_61	contract.sol	L78 - L80	*Pending Fix*
SSB_5029366_62	contract.sol	L82 - L85	*Pending Fix*
SSB_5029366_63	contract.sol	L87 - L90	*Pending Fix*
SSB_5029366_64	contract.sol	L92 - L99	*Pending Fix*
SSB_5029366_65	contract.sol	L105 - L114	*Pending Fix*
SSB_5029366_66	contract.sol	L123 - L129	*Pending Fix*
SSB_5029366_67	contract.sol	L134 - L143	*Pending Fix*
SSB_5029366_68	contract.sol	L149 - L152	*Pending Fix*
SSB_5029366_69	contract.sol	L158 - L169	*Pending Fix*
SSB_5029366_70	contract.sol	L171 - L177	*Pending Fix*

Bug ID	File Location	Line No.	Action Taken
SSB_5029366_71	contract.sol	L183 - L183	⚠ *Pending Fix*

Issue Type

MISSING @INHERITDOC ON OVERRIDE FUNCTIONS

S. No.	Severity	Detection Method	Instances
I004	Informational	Automated	11

Description

Overridden functions in Solidity should include the `@inheritdoc` tag in their NatSpec comments to ensure documentation consistency and clarity. Failing to include `@inheritdoc` can lead to incomplete or missing documentation, making it difficult for developers and auditors to understand the function's intended behavior. This lack of clarity can increase maintenance overhead and potentially lead to misinterpretations of contract logic.

Bug ID	File Location	Line No.	Action Taken
SSB_5029366_27	contract.sol	L59 - L68	*Pending Fix*
SSB_5029366_28	contract.sol	L74 - L76	*Pending Fix*
SSB_5029366_29	contract.sol	L78 - L80	*Pending Fix*
SSB_5029366_30	contract.sol	L82 - L85	*Pending Fix*
SSB_5029366_31	contract.sol	L87 - L90	*Pending Fix*
SSB_5029366_32	contract.sol	L92 - L99	*Pending Fix*
SSB_5029366_33	contract.sol	L105 - L114	*Pending Fix*
SSB_5029366_34	contract.sol	L149 - L152	*Pending Fix*

Bug ID	File Location	Line No.	Action Taken
SSB_5029366_35	contract.sol	L158 - L169	*Pending Fix*
SSB_5029366_36	contract.sol	L171 - L177	*Pending Fix*
SSB_5029366_37	contract.sol	L183 - L183	*Pending Fix*

Issue Type

MISSING NATSPEC DESCRIPTIONS FOR PUBLIC VARIABLE DECLARATIONS

S. No.	Severity	Detection Method	Instances
I005	Informational	Automated	5

Description

Public variable declarations without NatSpec descriptions are found in the code. NatSpec comments improve readabi lity and provide clear documentation for externally accessible variables. Without them, developers and users may str uggle to understand the role and impact of these variables when interacting with the contract.

Bug ID	File Location	Line No.	Action Taken
SSB_5029366_73	contract.sol	L20 - L20	*Pending Fix*
SSB_5029366_74	contract.sol	L21 - L21	*Pending Fix*
SSB_5029366_75	contract.sol	L22 - L22	*Pending Fix*
SSB_5029366_76	contract.sol	L24 - L24	*Pending Fix*
SSB_5029366_77	contract.sol	L30 - L30	*Pending Fix*

Issue Type

MISSING @NOTICE IN NATSPEC COMMENTS FOR CONSTRUCTORS

S. No.	Severity	Detection Method	Instances
I006	Informational	Automated	1

Description

Natspec `@notice` comments are missing from a constructor declaration in the code. The `@notice` tag provides an overview of what the constructor does, helping users and developers quickly understand its purpose.

Bug ID	File Location	Line No.	Action Taken
SSB_5029366_9	contract.sol	L59 - L68	*Pending Fix*

Issue Type

MISSING @NOTICE IN NATSPEC COMMENTS FOR FUNCTIONS

S. No.	Severity	Detection Method	Instances
I007	Informational	Automated	10

Description

Natspec `@notice` comments are missing from a function declaration in the code. The `@notice` tag provides an over view of what the function does, helping users and developers quickly understand its purpose.

Bug ID	File Location	Line No.	Action Taken
SSB_5029366_41	contract.sol	L74 - L76	*Pending Fix*
SSB_5029366_42	contract.sol	L78 - L80	*Pending Fix*
SSB_5029366_43	contract.sol	L82 - L85	*Pending Fix*
SSB_5029366_44	contract.sol	L87 - L90	*Pending Fix*
SSB_5029366_45	contract.sol	L92 - L99	*Pending Fix*
SSB_5029366_46	contract.sol	L105 - L114	*Pending Fix*
SSB_5029366_47	contract.sol	L149 - L152	*Pending Fix*
SSB_5029366_48	contract.sol	L158 - L169	*Pending Fix*
SSB_5029366_49	contract.sol	L171 - L177	*Pending Fix*
SSB_5029366_50	contract.sol	L183 - L183	*Pending Fix*

Issue Type

MISSING @NOTICE IN NATSPEC COMMENTS FOR MODIFIERS

S. No.	Severity	Detection Method	Instances
I008	Informational	Automated	1

Description

Natspec `@notice` comments are missing from a modifier declaration in the code. The `@notice` tag provides a high-level description of a modifier's purpose, making it easier to understand its function. Lack of `@notice` documentation can reduce code readability, making it harder to understand restrictions applied by modifiers.

Bug ID	File Location	Line No.	Action Taken
SSB_5029366_19	contract.sol	L50 - L53	*Pending Fix*

Issue Type

MISSING @PARAM IN NATSPEC COMMENTS FOR MODIFIERS

S. No.	Severity	Detection Method	Instances
I009	Informational	Automated	1

Description

Natspec `@param` comments are missing from a modifier declaration in the code. This reduces clarity, making it har
der to understand the purpose of modifier parameters, their expected values, and their role in contract execution.

Bug ID	File Location	Line No.	Action Taken
SSB_5029366_72	contract.sol	L50 - L53	*Pending Fix*

Issue Type

MISSING UNDERSCORE IN NAMING VARIABLES

S. No.	Severity	Detection Method	Instances
I010	Informational	Automated	2

Description

Solidity style guide suggests using underscores as the prefix for non-external functions and state variables (private o r internal) but the contract was not found to be following the same.

Bug ID	File Location	Line No.	Action Taken
SSB_5029366_82	contract.sol	L32 - L32	*Pending Fix*
SSB_5029366_83	contract.sol	L33 - L33	*Pending Fix*

Issue Type

NAME MAPPING PARAMETERS

S. No.	Severity	Detection Method	Instances
I011	Informational	Automated	2

Description

After Solidity 0.8.18, a feature was introduced to name mapping parameters. This helps in defining a purpose for each mapping and makes the code more descriptive.

Bug ID	File Location	Line No.	Action Taken
SSB_5029366_51	contract.sol	L32 - L32	*Pending Fix*
SSB_5029366_52	contract.sol	L33 - L33	*Pending Fix*

Issue Type

UNUSED RECEIVE FALLBACK

S. No.	Severity	Detection Method	Instances
I012	Informational	Automated	1

Description

The contract was found to be defining an empty function.
It is not recommended to leave them empty unless there's a specific use case such as to receive Ether via an empty `receive()` function.

Bug ID	File Location	Line No.	Action Taken
SSB_5029366_18	contract.sol	L183 - L183	*Pending Fix*

Issue Type

USE CALL INSTEAD OF TRANSFER OR SEND

S. No.	Severity	Detection Method	Instances
I013	Informational	Automated	1

Description

The contract was found to be using `transfer` or `send` function call. This is unsafe as `transfer` has hard coded gas budget and can fail if the user is a smart contract.

Bug ID	File Location	Line No.	Action Taken
SSB_5029366_14	contract.sol	L127 - L127	*Pending Fix*

Issue Type

USE SCIENTIFIC NOTATION

S. No.	Severity	Detection Method	Instances
I014	Informational	Automated	1

Description

Although the Solidity compiler can optimize exponentiation, it is recommended to prioritize idioms not reliant on com piler optimization. Utilizing scientific notation enhances code clarity, making it more self-explanatory and aligning wit h best practices in Solidity development.

Bug ID	File Location	Line No.	Action Taken
SSB_5029366_40	contract.sol	L62 - L62	*Pending Fix*

Issue Type

VARIABLES SHOULD BE IMMUTABLE

S. No.	Severity	Detection Method	Instances
I015	Informational	Automated	1

Description

Constants and Immutables should be used in their appropriate contexts.
`constant` should only be used for literal values written into the code. `immutable` variables should be used for expressions, or values calculated in, or passed into the constructor.

Bug ID	File Location	Line No.	Action Taken
SSB_5029366_39	contract.sol	L24 - L24	*Pending Fix*

Issue Type

AVOID RE-STORING VALUES

S. No.	Severity	Detection Method	Instances
G001	Gas	Automated	1

Description

The function is found to be allowing re-storing the value in the contract's state variable even when the old value is equal to the new value. This practice results in unnecessary gas consumption due to the `Gsreset` operation (2900 gas), which could be avoided. If the old value and the new value are the same, not updating the storage would avoid this 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_5029366_8	contract.sol	L171 - L177	*Pending Fix*

Issue Type

AVOID ZERO-TO-ONE STORAGE WRITES

S. No.	Severity	Detection Method	Instances
G002	Gas	Automated	1

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_5029366_1	contract.sol	L63 - L63	*Pending Fix*

Issue Type

BYTES CONSTANT MORE EFFICIENT THAN STRING LITERAL

S. No.	Severity	Detection Method	Instances
G003	Gas	Automated	2

Description

The contract was found to be using a string constant. This can be optimized by using `bytes32 constant` to save gas.

Bug ID	File Location	Line No.	Action Taken
SSB_5029366_56	contract.sol	L20 - L20	*Pending Fix*
SSB_5029366_57	contract.sol	L21 - L21	*Pending Fix*

Issue Type

CACHE ADDRESS(THIS) WHEN USED MORE THAN ONCE

S. No.	Severity	Detection Method	Instances
G004	Gas	Automated	2

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_5029366_5	contract.sol	L124 - L124	*Pending Fix*
SSB_5029366_6	contract.sol	L135 - L135	*Pending Fix*

Issue Type

CHEAPER CONDITIONAL OPERATORS

S. No.	Severity	Detection Method	Instances
G005	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_5029366_7	contract.sol	L125 - L125	*Pending Fix*

Issue Type

CHEAPER INEQUALITIES IN REQUIRE()

S. No.	Severity	Detection Method	Instances
G006	Gas	Automated	3

Description

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

Bug ID	File Location	Line No.	Action Taken
SSB_5029366_11	contract.sol	L94 - L94	*Pending Fix*
SSB_5029366_12	contract.sol	L107 - L107	*Pending Fix*
SSB_5029366_13	contract.sol	L163 - L163	*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 deployme
nt.

Bug ID	File Location	Line No.	Action Taken
SSB_5029366_38	contract.sol	L59 - L68	*Pending Fix*

Issue Type

FUNCTIONS CAN BE IN-LINED

S. No.	Severity	Detection Method	Instances
G008	Gas	Automated	1

Description

The internal function was called only once throughout the contract. Internal functions cost more gas due to additiona
l JUMP instructions and stack operations.

Bug ID	File Location	Line No.	Action Taken
SSB_5029366_24	contract.sol	L171 - L177	*Pending Fix*

Issue Type

REVERTING FUNCTIONS CAN BE PAYABLE

S. No.	Severity	Detection Method	Instances
G009	Gas	Automated	3

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_5029366_20	contract.sol	L123 - L129	*Pending Fix*
SSB_5029366_21	contract.sol	L134 - L143	*Pending Fix*
SSB_5029366_22	contract.sol	L149 - L152	*Pending Fix*

Issue Type

GAS OPTIMIZATION FOR STATE VARIABLES

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

Description

Plus equals (`+=`) costs more gas than addition operator. The same thing happens with minus equals (`-=`).

Bug ID	File Location	Line No.	Action Taken
SSB_5029366_25	contract.sol	L110 - L110	*Pending Fix*
SSB_5029366_26	contract.sol	L166 - L166	*Pending Fix*

Issue Type

OPTIMIZING ADDRESS ID MAPPING

S. No.	Severity	Detection Method	Instances
G011	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_5029366_58	contract.sol	L32 - L32	*Pending Fix*
SSB_5029366_59	contract.sol	L33 - L33	*Pending Fix*

Issue Type

PUBLIC CONSTANTS CAN BE PRIVATE

S. No.	Severity	Detection Method	Instances
G012	Gas	Automated	3

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_5029366_79	contract.sol	L20 - L20	*Pending Fix*
SSB_5029366_80	contract.sol	L21 - L21	*Pending Fix*
SSB_5029366_81	contract.sol	L22 - L22	*Pending Fix*

Issue Type

SMALLER DATA TYPES COST MORE

S. No.	Severity	Detection Method	Instances
G013	Gas	Automated	1

Description

Usage of smaller integer types such as `uint8`, `uint16`, `int8`, or `int16` in arithmetic operations incur additional gas costs compared to the default `uint` and `int` types, which are typically `uint256` and `int256` respectively.

Bug ID	File Location	Line No.	Action Taken
SSB_5029366_23	contract.sol	L62 - L62	*Pending Fix*

Issue Type

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

S. No.	Severity	Detection Method	Instances
G014	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_5029366_4	contract.sol	L124 - L124	*Pending Fix*

Issue Type

VARIABLES DECLARED BUT NEVER USED

S. No.	Severity	Detection Method	Instances
G015	Gas	Automated	2

Description

The contract has declared a variable but it is not used anywhere in the code. This represents dead code or missing lo
gic.
Unused variables increase the contract's size and complexity, potentially leading to higher gas costs and a larger att
ack surface

Bug ID	File Location	Line No.	Action Taken
SSB_5029366_53	contract.sol	L20 - L20	*Pending Fix*
SSB_5029366_54	contract.sol	L21 - L21	*Pending Fix*

06. Scan History

Critical High Medium Low Informational Gas

No	Date	Security Score	Scan Overview
1.	2026-02-05	89.00	0 0 3 2 51 26

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.