SolidityScan | Powered by CRED SHIELDS

Security Assessment Report

BrainrotLoanVault

24 Oct 2026

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

AICPA SOC | SCS Advocates OWASP OWASP Smart Contract Top 10 Security Risks & Vulnerabilities

Table of Contents

01 Vulnerability Classification and Severity

02 Executive Summary

03 Threat Summary

04 Findings Summary

05 Vulnerability Details

- UNCHECKED ARRAY LENGTH
- PRECISION LOSS DURING DIVISION BY LARGE NUMBERS
- USE OF FLOATING PRAGMA
- MISSING EVENTS
- MISSING ZERO ADDRESS VALIDATION
- NONREENTRANT MODIFIER PLACEMENT
- OUTDATED COMPILER VERSION
- ADDING A RETURN STATEMENT WHEN THE FUNCTION DEFINES A NAMED RETURN VARIABLE IS REDUNDANT
- AVOID ARITHMETIC DIRECTLY WITHIN ARRAY INDICES
- BLOCK VALUES AS A PROXY FOR TIME
- MISSING @AUTHOR IN NATSPEC COMMENTS FOR CONTRACT DECLARATION
- MISSING @DEV IN NATSPEC COMMENTS FOR CONTRACT DECLARATION
- MISSING @DEV IN NATSPEC COMMENTS FOR FUNCTIONS
- MISSING INDEXED KEYWORDS IN EVENTS
- MISSING @INHERITDOC ON OVERRIDE FUNCTIONS
- 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

NAME MAPPING PARAMETERS
REVERT STATEMENTS WITHIN EXTERNAL AND PUBLIC FUNCTIONS CAN BE USED TO PERFORM DOS ATTACKS
UNNAMED FUNCTION PARAMETERS
ASSIGNING TO STRUCTS CAN BE MORE EFFICIENT
AVOID RE-STORING VALUES
AVOID ZERO-TO-ONE STORAGE WRITES
GAS OPTIMIZATION FOR THIS KEYWORD
CHEAPER INEQUALITIES IN IF()
DEFAULT INT VALUES ARE MANUALLY RESET
DEFINE CONSTRUCTOR AS PAYABLE
EMIT USED IN LOOP
FUNCTIONS CAN BE IN-LINED
REVERTING FUNCTIONS CAN BE PAYABLE
FUNCTION SHOULD RETURN STRUCT
GAS INEFFICIENCY DUE TO MULTIPLE OPERANDS IN SINGLE IF/ELSEIF CONDITION
GAS OPTIMIZATION FOR STATE VARIABLES
GAS OPTIMIZATION IN INCREMENTS
PUBLIC CONSTANTS CAN BE PRIVATE
SPLITTING REVERT STATEMENTS
STORAGE VARIABLE CACHING IN MEMORY
STORING STORAGE VARIABLES IN MEMORY
SUPERFLUOUS EVENT FIELDS
UNNECESSARY CHECKED ARITHMETIC IN LOOP
UNUSED IMPORTS
USE SOLADY LIB WHEN POSSIBLE

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

BrainrotLoanVault

Uploaded Solidity File(s)

Language	Audit Methodology	Website
Solidity	**Static Scanning**	-
Publishers/Owner Name	**Organization**	**Contact Email**
-	-	-



Security Score is GREAT

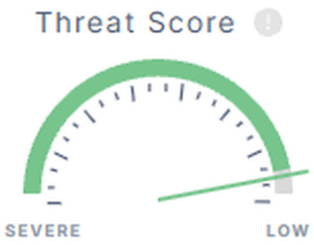
The SolidityScan score is calculated based on lines of code and weights assigned to each issue depending on the severity and confidence. To improve your score, view the detailed result and leverage the remediation solutions provided.

This report has been prepared for BrainrotLoanVault 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 BrainrotLoanVault introduces new features or refactors the code.

03. Threat Summary



Low Risk

94/100

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

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 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 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.

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.

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.

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.

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.

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

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.
BrainrotLoanVault.sol	L2 - L2

Issue Type

PAUSABLE CONTRACTS

Action Taken

⚠ *Pending Fix*

Description

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.

Remediation

Implement the pause() and unpause() functions in your contract and secure them with the onlyOwner modifier. This ensures that only the contract owner has the authority to change the paused state of the contract. Additionally, verify that the owner is correctly set and that the access control mechanisms are thoroughly tested to prevent any unauthorized actions.

File Location	Line No.
BrainrotLoanVault.sol	L416 - L418
BrainrotLoanVault.sol	L420 - L422

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.
BrainrotLoanVault.sol	L6 - L6
BrainrotLoanVault.sol	L7 - L7

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.

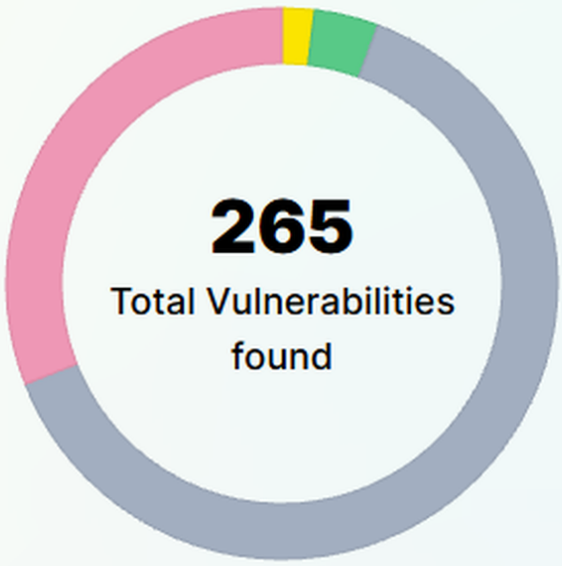
File Location	Line No.
BrainrotLoanVault.sol	L171 - L173
BrainrotLoanVault.sol	L175 - L177
BrainrotLoanVault.sol	L363 - L365
BrainrotLoanVault.sol	L367 - L376

File Location	Line No.
BrainrotLoanVault.sol	L378 - L382
BrainrotLoanVault.sol	L384 - L393
BrainrotLoanVault.sol	L396 - L400
BrainrotLoanVault.sol	L403 - L407
BrainrotLoanVault.sol	L410 - L413
BrainrotLoanVault.sol	L416 - L418
BrainrotLoanVault.sol	L420 - L422
BrainrotLoanVault.sol	L425 - L427

04. Findings Summary

BrainrotLoanVault
File Scan

Security Score	Scan duration	Lines of code
88.16/100	7 secs	422



0	0	5	10	168	82
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	272
Fixed	False Positive	Won't Fix	Pending Fix

S. No.	Severity	Bug Type	Instances	Detection Method	Status
M001	Medium	UNCHECKED ARRAY LENGTH	4	Automated	Pending Fix
M002	Medium	PRECISION LOSS DURING DIVISION BY LARGE NUMBERS	1	Automated	Pending Fix
L001	Low	USE OF FLOATING PRAGMA	1	Automated	Pending Fix
L002	Low	MISSING EVENTS	3	Automated	Pending Fix
L003	Low	MISSING ZERO ADDRESS VALIDATION	3	Automated	Pending Fix
L004	Low	NONREENTRANT MODIFIER PLACEMENT	2	Automated	Pending Fix
L005	Low	OUTDATED COMPILER VERSION	1	Automated	Pending Fix
I001	Informational	ADDING A RETURN STATEMENT WHEN THE FUNCTION DEFINES A NAMED RETURN VARIABLE IS REDUNDANT	1	Automated	Pending Fix
I002	Informational	AVOID ARITHMETIC DIRECTLY WITHIN ARRAY INDICES	1	Automated	Pending Fix
I003	Informational	BLOCK VALUES AS A PROXY FOR TIME	3	Automated	Pending Fix
I004	Informational	MISSING @AUTHOR IN NATSPEC COMMENTS FOR CONTRACT DECLARATION	2	Automated	Pending Fix
I005	Informational	MISSING @DEV IN NATSPEC COMMENTS FOR CONTRACT DECLARATION	2	Automated	Pending Fix
I006	Informational	MISSING @DEV IN NATSPEC COMMENTS FOR FUNCTIONS	40	Automated	Pending Fix
I007	Informational	MISSING INDEXED KEYWORDS IN EVENTS	2	Automated	Pending Fix

S. No.	Severity	Bug Type	Instances	Detection Method	Status
I008	Informational	MISSING @INHERITDOC ON OVERRIDE FUNCTIONS	38	Automated	*Pending Fix*
I009	Informational	MISSING NATSPEC COMMENTS IN SCOPE BLOCKS	6	Automated	*Pending Fix*
I010	Informational	MISSING NATSPEC DESCRIPTIONS FOR PUBLIC VARIABLE DECLARATIONS	13	Automated	*Pending Fix*
I011	Informational	MISSING @NOTICE IN NATSPEC COMMENTS FOR CONSTRUCTORS	1	Automated	*Pending Fix*
I012	Informational	MISSING @NOTICE IN NATSPEC COMMENTS FOR FUNCTIONS	40	Automated	*Pending Fix*
I013	Informational	NAME MAPPING PARAMETERS	3	Automated	*Pending Fix*
I014	Informational	REVERT STATEMENTS WITHIN EXTERNAL AND PUBLIC FUNCTIONS CAN BE USED TO PERFORM DOS ATTACKS	14	Automated	*Pending Fix*
I015	Informational	UNNAMED FUNCTION PARAMETERS	2	Automated	*Pending Fix*
G001	Gas	ASSIGNING TO STRUCTS CAN BE MORE EFFICIENT	2	Automated	*Pending Fix*
G002	Gas	AVOID RE-STORING VALUES	5	Automated	*Pending Fix*
G003	Gas	AVOID ZERO-TO-ONE STORAGE WRITES	2	Automated	*Pending Fix*
G004	Gas	GAS OPTIMIZATION FOR THIS KEYWORD	2	Automated	*Pending Fix*
G005	Gas	CHEAPER INEQUALITIES IN IF()	13	Automated	*Pending Fix*
G006	Gas	DEFAULT INT VALUES ARE MANUALLY RESET	1	Automated	*Pending Fix*
G007	Gas	DEFINE CONSTRUCTOR AS PAYABLE	1	Automated	*Pending Fix*
G008	Gas	EMIT USED IN LOOP	1	Automated	*Pending Fix*
G009	Gas	FUNCTIONS CAN BE IN-LINED	1	Automated	*Pending Fix*

S. No.	Severity	Bug Type	Instances	Detection Method	Status
G010	Gas	REVERTING FUNCTIONS CAN BE PAYABLE	11	Automated	*Pending Fix*
G011	Gas	FUNCTION SHOULD RETURN STRUCT	1	Automated	*Pending Fix*
G012	Gas	GAS INEFFICIENCY DUE TO MULTIPLE OPERANDS IN SINGLE IF/ELSEIF CONDITION	13	Automated	*Pending Fix*
G013	Gas	GAS OPTIMIZATION FOR STATE VARIABLES	3	Automated	*Pending Fix*
G014	Gas	GAS OPTIMIZATION IN INCREMENTS	5	Automated	*Pending Fix*
G015	Gas	PUBLIC CONSTANTS CAN BE PRIVATE	7	Automated	*Pending Fix*
G016	Gas	SPLITTING REVERT STATEMENTS	1	Automated	*Pending Fix*
G017	Gas	STORAGE VARIABLE CACHING IN MEMORY	1	Automated	*Pending Fix*
G018	Gas	STORING STORAGE VARIABLES IN MEMORY	3	Automated	*Pending Fix*
G019	Gas	SUPERFLUOUS EVENT FIELDS	1	Automated	*Pending Fix*
G020	Gas	UNNECESSARY CHECKED ARITHMETIC IN LOOP	6	Automated	*Pending Fix*
G021	Gas	UNUSED IMPORTS	1	Automated	*Pending Fix*
G022	Gas	USE SOLADY LIB WHEN POSSIBLE	1	Automated	*Pending Fix*