

## Security Assessment Report

# MintableERC20

15 Jan 2026

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

# Table of Contents

01 Vulnerability Classification and Severity

02 Executive Summary

03 Threat Summary

04 Findings Summary

05 Vulnerability Details

SUPPORTSINTERFACE() CALLS MAY REVERT

APPROVE FRONT-RUNNING ATTACK

USE OF FLOATING PRAGMA

MISSING ZERO ADDRESS VALIDATION

OUTDATED COMPILER VERSION

CONSTRUCTORS SHOULD EMIT AN EVENT

CONTRACT NAME SHOULD USE PASCALCASE

IF-STATEMENT REFACTORING

INTERFACES SHOULD BE DECLARED IN A SEPARATE FILE

MISSING @AUTHOR IN NATSPEC COMMENTS FOR ABSTRACT CONTRACTS

MISSING @AUTHOR IN NATSPEC COMMENTS FOR CONTRACT DECLARATION

MISSING @DEV IN NATSPEC COMMENTS FOR ABSTRACT CONTRACTS

MISSING @DEV IN NATSPEC COMMENTS FOR CONTRACT DECLARATION

MISSING @DEV IN NATSPEC COMMENTS FOR FUNCTIONS

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

MISSING @NOTICE IN NATSPEC COMMENTS FOR MODIFIERS

MISSING @PARAM IN NATSPEC COMMENTS FOR MODIFIERS

---

REVERT STATEMENTS WITHIN EXTERNAL AND PUBLIC FUNCTIONS CAN BE USED TO PERFORM  
DOS ATTACKS

---

USE SCIENTIFIC NOTATION

---

AVOID RE-STORING VALUES

---

CHEAPER INEQUALITIES IN IF()

---

CHEAPER INEQUALITIES IN REQUIRE()

---

DEFAULT INT VALUES ARE MANUALLY RESET

---

DEFINE CONSTRUCTOR AS PAYABLE

---

FUNCTIONS CAN BE IN-LINED

---

INTERNAL FUNCTIONS NEVER USED

---

LONG REQUIRE/REVERT STRINGS

---

OPTIMIZING ADDRESS ID MAPPING

---

PUBLIC CONSTANTS CAN BE PRIVATE

---

STORAGE VARIABLE CACHING IN MEMORY

---

FUNCTION SHOULD BE EXTERNAL

---

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



### MintableERC20

0x5Bb895538bE44fBa164F4cDe8e27E91811Cc9204

<https://opbnb.bscscan.com/address/0x5Bb895538bE44fBa164F4c...>

Language	Audit Methodology	Contract Type
<b>Solidity</b>	<b>Static Scanning</b>	-
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 MintableERC20 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 MintableERC20 introduces new features or refactors the code.

## 03. Threat Summary

Threat Score ⓘ



Moderate Risk  
67.5 / 100

### THREAT SUMMARY

Your smart contract has been assessed and assigned a **Moderate 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 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 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, uninterrupted unit.



### RENOUNCED OWNERSHIP

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.



### No addresses contain more than 5% 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 not 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 0 tokens which is less than 5% 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.



## CODE INJECTION VIA TOKEN NAME

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.

## 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.

## 04. Findings Summary



0x5Bb895538bE44fBa164F4cDe8e27E91811Cc9204

OPBNB (opBNB Mainnet)

[View on Opbnb](#)

Security Score

95.23/100



Scan duration

13 secs



Lines of code

906



0

Crit

0

High

4

Med

14

Low

180

Info

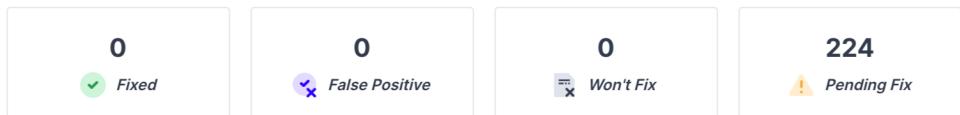
26

Gas



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

## ACTION TAKEN



S. No.	Severity	Bug Type	Instances	Detection Method	Status
M001	<span style="color: yellow;">●</span> Medium	SUPPORTSINTERFACE() CALLS MAY REVERT	1	Automated	Pending Fix
M002	<span style="color: yellow;">●</span> Medium	APPROVE FRONT-RUNNING ATTACK	3	Automated	Pending Fix
L001	<span style="color: green;">●</span> Low	USE OF FLOATING PRAGMA	10	Automated	Pending Fix
L002	<span style="color: green;">●</span> Low	MISSING ZERO ADDRESS VALIDATION	6	Automated	Pending Fix
L003	<span style="color: green;">●</span> Low	OUTDATED COMPILER VERSION	10	Automated	Pending Fix
I001	<span style="color: grey;">●</span> Informational	CONSTRUCTORS SHOULD EMIT AN EVENT	1	Automated	Pending Fix
I002	<span style="color: grey;">●</span> Informational	CONTRACT NAME SHOULD USE PASCALCASE	3	Automated	Pending Fix
I003	<span style="color: grey;">●</span> Informational	IF-STATEMENT REFACTORING	1	Automated	Pending Fix
I004	<span style="color: grey;">●</span> Informational	INTERFACES SHOULD BE DECLARED IN A SEPARATE FILE	5	Automated	Pending Fix
I005	<span style="color: grey;">●</span> Informational	MISSING @AUTHOR IN NATSPEC COMMENTS FOR ABSTRACT CONTRACTS	4	Automated	Pending Fix
I006	<span style="color: grey;">●</span> Informational	MISSING @AUTHOR IN NATSPEC COMMENTS FOR CONTRACT DECLARATION	12	Automated	Pending Fix
I007	<span style="color: grey;">●</span> Informational	MISSING @DEV IN NATSPEC COMMENTS FOR ABSTRACT CONTRACTS	4	Automated	Pending Fix
I008	<span style="color: grey;">●</span> Informational	MISSING @DEV IN NATSPEC COMMENTS FOR CONTRACT DECLARATION	12	Automated	Pending Fix
I009	<span style="color: grey;">●</span> Informational	MISSING @DEV IN NATSPEC COMMENTS FOR FUNCTIONS	13	Automated	Pending Fix

S. No.	Severity	Bug Type	Instances	Detection Method	Status
I010	● Informational	MISSING @INHERITDOC ON OVERRIDE FUNCTIONS	47	Automated	Pending Fix
I011	● Informational	MISSING NATSPEC COMMENTS IN SCOPE BLOCKS	24	Automated	Pending Fix
I012	● Informational	MISSING NATSPEC DESCRIPTIONS FOR PUBLIC VARIABLE DECLARATIONS	3	Automated	Pending Fix
I013	● Informational	MISSING @NOTICE IN NATSPEC COMMENTS FOR CONSTRUCTORS	2	Automated	Pending Fix
I014	● Informational	MISSING @NOTICE IN NATSPEC COMMENTS FOR FUNCTIONS	46	Automated	Pending Fix
I015	● Informational	MISSING @NOTICE IN NATSPEC COMMENTS FOR MODIFIERS	1	Automated	Pending Fix
I016	● Informational	MISSING @PARAM IN NATSPEC COMMENTS FOR MODIFIERS	1	Automated	Pending Fix
I017	● Informational	REVERT STATEMENTS WITHIN EXTERNAL AND PUBLIC FUNCTIONS CAN BE USED TO PERFORM DOS ATTACKS	1	Automated	Pending Fix
I018	● Informational	USE SCIENTIFIC NOTATION	1	Automated	Pending Fix
G001	● Gas	AVOID RE-STORING VALUES	2	Automated	Pending Fix
G002	● Gas	CHEAPER INEQUALITIES IN IF()	1	Automated	Pending Fix
G003	● Gas	CHEAPER INEQUALITIES IN REQUIRE()	1	Automated	Pending Fix
G004	● Gas	DEFAULT INT VALUES ARE MANUALLY RESET	1	Automated	Pending Fix
G005	● Gas	DEFINE CONSTRUCTOR AS PAYABLE	1	Automated	Pending Fix
G006	● Gas	FUNCTIONS CAN BE IN-LINED	2	Automated	Pending Fix
G007	● Gas	INTERNAL FUNCTIONS NEVER USED	4	Automated	Pending Fix
G008	● Gas	LONG REQUIRE/REVERT STRINGS	2	Automated	Pending Fix

S. No.	Severity	Bug Type	Instances	Detection Method	Status
G009	<span style="color: red;">●</span> Gas	OPTIMIZING ADDRESS ID MAPPING	2	Automated	<span style="color: orange;">⚠ Pending Fix</span>
G010	<span style="color: red;">●</span> Gas	PUBLIC CONSTANTS CAN BE PRIVATE	3	Automated	<span style="color: orange;">⚠ Pending Fix</span>
G011	<span style="color: red;">●</span> Gas	STORAGE VARIABLE CACHING IN MEMORY	3	Automated	<span style="color: orange;">⚠ Pending Fix</span>
G012	<span style="color: red;">●</span> Gas	FUNCTION SHOULD BE EXTERNAL	4	Automated	<span style="color: orange;">⚠ Pending Fix</span>
G013	<span style="color: red;">●</span> Gas	VARIABLES DECLARED BUT NEVER USED	1	Automated	<span style="color: orange;">⚠ Pending Fix</span>

## 05. **Vulnerability** Details

Issue Type

**SUPPORTSINTERFACE() CALLS MAY REVERT**

**Reveal Detailed Vulnerabilities**

**1 Medium Issues Found**

Make a one-time payment and get a detailed security report for your smart contract with security scores, bug descriptions & remediations directly in your inbox!

 **Unlock report**

## Issue Type

**USE OF FLOATING PRAGMA**

S. No.	Severity	Detection Method	Instances
L001	● Low	Automated	10

**Reveal Detailed Vulnerabilities****10 Low Issues Found**

Make a one-time payment and get a detailed security report for your smart contract with security scores, bug descriptions & remediations directly in your inbox!

**🔒 Unlock report**

## Issue Type

**CONSTRUCTORS SHOULD EMIT AN EVENT**

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

**Reveal Detailed Vulnerabilities****1 Informational Issues Found**

Make a one-time payment and get a detailed security report for your smart contract with security scores, bug descriptions & remediations directly in your inbox!

**🔒 Unlock report**

### Issue Type

## AVOID RE-STORING VALUES

S. No.	Severity	Detection Method	Instances
<b>G001</b>	<span style="color: red;">● Gas</span>	<b>Automated</b>	<b>2</b>

## **Free Unlimited Gas Issues**

## 2 Gas Issues Found

Sign up for a free trial and optimize your contracts for gas absolutely free!

[Signup for Free Trial](#)

The SolidityScan interface displays a test project named "Test Project\_0098". The main dashboard includes metrics like gas cost (55.67), execution time (5.65 sec), storage (28567), memory (566), and network (75/100). It features tabs for Overview, Vulnerability Scan (selected), and Threat Scan. A sidebar on the left provides links to Home, Projects, Integrations, Private API, Billing, Organisation, and User Guide. A bottom navigation bar includes Request manual audit, Report, and Help.

**Vulnerability Scan**

**Filter Parameter**

**Take action**

**Select Option**

**Variables Declared But Never Used**

**Incorrect Access Control**

**SafeMath Usage**

**Contract Functions**

**Remediation**

**AI Remediation**

**Comments**

**Critical**   **Certain**   L106 - L122   **Won't Fix**

**Vulnerability Description**

**INCORRECT ACCESS CONTROL**

Access control plays an important role in segregation of privileges in smart contracts and other applications. If this is misconfigured or not properly validated on sensitive functions, it may lead to loss of funds, tokens and in some cases compromise of the smart contract. The contract `ProxyContract` is importing an access control library `@openzeppelin/contracts-access/Ownable.sol` but the function `mint` is missing the modifier `onlyOwner`.

**Code**

```
contract(..., owner) {
    constructor(...);
    modifier restricted() {
        require(msg.sender == owner);
    }
    if (msg.sender == owner) ...
    constructor(...);
    owner = msg.sender;
}

function estoppel(isDisabled) public restricted {
    ...
}
```

**False Positive**

**Won't Fix**

**False Positive**

**False Positive**

**Won't Fix**

**Won't Fix**

**Use of SafeMath**

**Use of SafeMath**

## 06. Scan History

● Critical   ● High   ● Medium   ● Low   ● Informational   ● Gas

No	Date	Security Score	Scan Overview
1.	2026-01-15	<b>95.23</b>	● 0 ● 0 ● 4 ● 14 ● 180 ● 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.

