



Powered by
 CRED
SHIELDS

Security Assessment Report

OnChainGM

28 Feb 2025

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

INCORRECT ACCESS CONTROL

EVENT BASED REENTRANCY

USE OF FLOATING PRAGMA

HARD-CODED ADDRESS DETECTED

ARBITRUM OPCODE DIFFERENCES

BLOCK VALUES AS A PROXY FOR TIME

NAME MAPPING PARAMETERS

REQUIRE INSTEAD OF REVERT

CUSTOM ERRORS TO SAVE GAS

DEFINE CONSTRUCTOR AS PAYABLE

PUBLIC CONSTANTS CAN BE PRIVATE

STORAGE VARIABLE CACHING IN MEMORY

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



OnChainGM

0xb08432846fae0f5b1b18e54b73630adc9aa6ac5a

<https://arbscan.io/address/0xb08432846fae0f5b1b18e54b736...>

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 OnChainGM 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 (100+) 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 OnChainGM introduces new features or refactors the code.

03. Threat Summary

Threat Score !



THREAT SUMMARY

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.



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 cannot 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 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.

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 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 WALLET 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.



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.



NO INTERACTION WITH THE SMART CONTRACT IN 30 DAYS

This check identifies if there has been any interaction with the smart contract within the past 30 days. Contracts that remain inactive for prolonged periods may indicate abandonment or lack of use, which could affect users' confidence and the contract's ongoing functionality.



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

Action Taken

 Pending Fix

HARDCODED ADDRESSES

Description

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.

Remediation

To remediate, consider using configuration variables or function parameters instead of hardcoded addresses to allow for flexibility and easier updates in the future. If a fixed address is necessary, ensure that it is clearly documented and easily modifiable in the contract.

contract.sol 

L17 - L17

```
16     constructor() {
17         feeRecipient = 0x7500A83DF2aF99B2755c47B6B321a8217d876a85;
18         GM_FEE = 0.000029 ether;
19     }
```

04. Findings Summary



0xb08432846fae0f5b1b18e54b73630adc9aa6ac5a
ARBITRUM (Arbiscan Mainnet) | [View on Arbiscan](#)



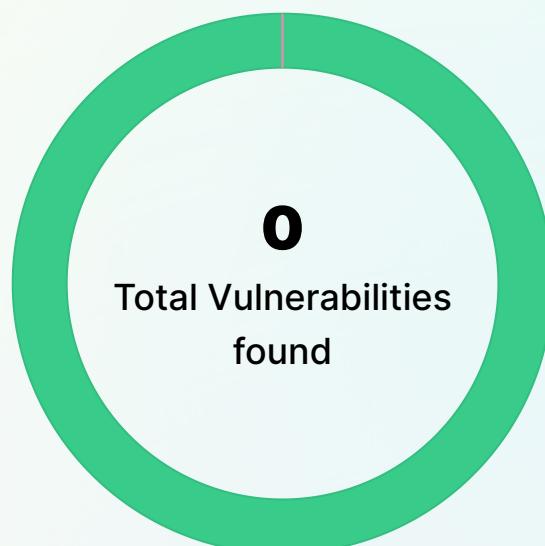
Security Score
97.60/100



Scan duration
10 secs



Lines of code
41



0

Crit

0

High

0

Med

1

Low

0

Info

-1

Gas



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

ACTION TAKEN

0

Fixed

19

False Positive

1

Won't Fix

0

Pending Fix

S. No.	Severity	Bug Type	Instances	Detection Method	Status
C001	● Critical	INCORRECT ACCESS CONTROL	1	Automated	False Positive
L001	● Low	EVENT BASED REENTRANCY	1	Automated	Pending Fix
L002	● Low	USE OF FLOATING PRAGMA	1	Automated	False Positive
I001	● Informational	HARD-CODED ADDRESS DETECTED	1	Automated	False Positive
I002	● Informational	ARBITRUM OPCODE DIFFERENCES	3	Automated	False Positive
I003	● Informational	BLOCK VALUES AS A PROXY FOR TIME	3	Automated	False Positive
I004	● Informational	NAME MAPPING PARAMETERS	1	Automated	False Positive
I005	● Informational	REQUIRE INSTEAD OF REVERT	3	Automated	False Positive
G001	● Gas	CUSTOM ERRORS TO SAVE GAS	3	Automated	False Positive
G002	● Gas	DEFINE CONSTRUCTOR AS PAYABLE	1	Automated	False Positive
G003	● Gas	PUBLIC CONSTANTS CAN BE PRIVATE	1	Automated	False Positive
G004	● Gas	STORAGE VARIABLE CACHING IN MEMORY	2	Automated	False Positive

05. **Vulnerability** Details

Issue Type

INCORRECT ACCESS CONTROL

S. No.	Severity	Detection Method	Instances
C001	● Critical	Automated	1

Upgrade your Plan to view the full report

1 Critical Issues Found

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

 **Upgrade**

yes

Issue Type

EVENT BASED REENTRANCY

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

Bug ID

File Location

Line No.

Action Taken

Upgrade your Plan to view the full report

1 Low Issues Found

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

 **Upgrade**

Issue Type

HARD-CODED ADDRESS DETECTED

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

Bug ID

File Location

Line No.

Action Taken

Upgrade your Plan to view the full report

1 Informational Issues Found

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

 **Upgrade**

Issue Type

CUSTOM ERRORS TO SAVE GAS

S. No.	Severity	Detection Method	Instances
G001	● Gas	Automated	3

Description

The contract was found to be using `revert()` statements. Since Solidity `v0.8.4`, custom errors have been introduced which are a better alternative to the revert. This allows the developers to pass custom errors with dynamic data while reverting the transaction and also making the whole implementation a bit cheaper than using `revert`.

Bug ID	File Location	Line No.	Action Taken
SSB_3414860_17	contract.sol	L24 - L24	 False Positive
contract.sol 			L24 - L24
<pre>23 if (msg.value != GM_FEE) { 24 revert("Incorrect ETH fee"); 25 } 26</pre>			

Comments

Fixed

Bug ID	File Location	Line No.	Action Taken
SSB_3414860_18	contract.sol	L29 - L29	 False Positive

contract.sol 

L29 - L29

```
28     if (!(block.timestamp >= lastGMTimestamp[msg.sender] + TIME_LIMIT || lastGMTimestamp[msg.sender] == 0)) {
29         revert("Wait 24 hours");
30     }
31
```

Comments

Fixed

Bug ID	File Location	Line No.	Action Taken
SSB_3414860_19	contract.sol	L38 - L38	 False Positive

contract.sol 

L38 - L38

```
37     if (!success) {
38         revert("Fee transfer failed");
39     }
40
```

Comments

Fixed

Issue Type

DEFINE CONSTRUCTOR AS PAYABLE

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

Description

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

Bug ID	File Location	Line No.	Action Taken
SSB_3414860_4	contract.sol	L16 - L19	✖ False Positive

```
contract.sol ↗          L16 - L19

15
16     constructor() {
17         feeRecipient = 0x7500A83DF2aF99B2755c47B6B321a8217d876a85;
18         GM_FEE = 0.000029 ether;
19     }
20
21     // Gas optimized GM function with timestamp check
```

Comments

Fixed

Page 24

SolidityScan • A security assessment report

Issue Type

PUBLIC CONSTANTS CAN BE PRIVATE

S. No.	Severity	Detection Method	Instances
G003	● Gas	Automated	1

Description

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

Bug ID	File Location	Line No.	Action Taken
SSB_3414860_15	contract.sol	L8 - L8	 False Positive

contract.sol 

L8 - L8

```
7     uint256 public immutable GM_FEE;
8     uint256 public constant TIME_LIMIT = 24 hours;
9
10    // Mapping to store last GM timestamp for each user
```

Comments

Fixed

Issue Type

STORAGE VARIABLE CACHING IN MEMORY

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

Description

The contract is using the state variable multiple times in the function.

`SLOADs` are expensive (100 gas after the 1st one) compared to `MLOAD / MSTORE` (3 gas each).

Bug ID	File Location	Line No.	Action Taken
SSB_3414860_5	contract.sol	L45 - L52	✖ False Positive

contract.sol [🔗](#) L45 - L52

```
44     // View function to check remaining time
45     function timeUntilNextGM(address user) external view returns (uint256) {
46         if (lastGMTimestamp[user] == 0) return 0;
47
48         uint256 timePassed = block.timestamp - lastGMTimestamp[user];
49         if (timePassed >= TIME_LIMIT) return 0;
50
51         return TIME_LIMIT - timePassed;
52     }
53 }
```

Comments

Fixed

Bug ID	File Location	Line No.	Action Taken
SSB_3414860_5	contract.sol	L45 - L52	 False Positive

[contract.sol](#) ↗

L45 - L52

```
44     // View function to check remaining time
45     function timeUntilNextGM(address user) external view returns (uint256) {
46         if (lastGMTimestamp[user] == 0) return 0;
47
48         uint256 timePassed = block.timestamp - lastGMTimestamp[user];
49         if (timePassed >= TIME_LIMIT) return 0;
50
51         return TIME_LIMIT - timePassed;
52     }
53 }
```

Comments

Fixed

06. Scan History

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

No	Date	Security Score	Scan Overview					
1.	2025-02-28	97.60	● 0	● 0	● 0	● 1	● 0	● -1

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.