SolidityScan | Powered by CRED SHIELDS

Security Assessment Report

Codeup

1 Oct 2024

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

Verified Report

This automated audit report has been verified by the SolidityScan team. To learn more about our published reports click here.

Table of Contents

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

Codeup

Published on 01 Oct 2024

0×1B82536914F3fD162e1EDa12ddB2E05cd961Fe57

https://arbiscan.io/address/0×1B82536914F3fD162e1EDa12ddB...

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 Codeup 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 Codeup introduces new features or refactors the code.

03. Threat Summary

Threat Score

LOW RISK

100/100

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.

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.

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 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.

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.

The Overpowered Owners module is dedicated to identifying situations where contract owners are endowed with excessive privileges through critical functions. Granting too many privileges to owners, especially via critical functions, might pose a significant risk to users' funds if the owners are compromised or if a rug-pulling attack occurs. In the context of smart contracts, owners often have access to critical functions that can impact the contract's functionality, token distribution, or other essential aspects. While providing owners with necessary permissions is crucial for contract management, it is equally important to avoid overempowering owners to mitigate potential risks.

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.

04. Findings Summary

0×1B82536914F3fD162e1EDa12ddB2E05cd961Fe57
ARBITRUM (Arbiscan Mainnet) | View on Arbiscan

Security Score	Scan duration	Lines of code
98.48/100	6 secs	1117

0	0	2	0	7	4
Crit	High	Med	Low	Info	Gas

This audit report has been verified by the SolidityScan team. To learn more about our published reports. click here

ACTION TAKEN

0	0	0	13
Fixed	False Positive	Won't Fix	Pending Fix

S. No.	Severity	Bug Type	Instances	Detection Method	Status
M001	Medium	PRECISION LOSS DURING DIVISION BY LARGE NUMBERS	2	Automated	Pending Fix
I001	Informational	IF-STATEMENT REFACTORING	1	Automated	Pending Fix
I002	Informational	MISSING UNDERSCORE IN NAMING VARIABLES	4	Automated	Pending Fix
I003	Informational	NAME MAPPING PARAMETERS	2	Automated	Pending Fix
G001	Gas	GAS OPTIMIZATION IN INCREMENTS	1	Automated	Pending Fix
G002	Gas	UNNECESSARY MEMORY OPERATIONS WITH AN IMMUTABLE VARIABLE	3	Automated	Pending Fix

05. Vulnerability Details

Issue Type

PRECISION LOSS DURING DIVISION BY LARGE NUMBERS

S. No.	Severity	Detection Method	Instances
M001	Medium	Automated	2

Description

In Solidity, when dividing large numbers, precision loss can occur due to limitations in the Ethereum Virtual Machine (EVM). Solidity lacks native support for decimal or fractional numbers, leading to truncation of division results to integers. This can result in inaccuracies or unexpected behaviors, especially when the numerator is not significantly larger than the denominator.

Bug ID	File Location	Line No.	Action Taken
SSB_1336915_31	contracts/Codeup.sol	L159 - L159	*Pending Fix*

contracts/Codeup.sol L159 - L159

```
158         require(block.timestamp > startUNIX, NotStarted());
159         uint256 gameETH = tokenAmount / gameETHPrice;
160         _checkValue(gameETH);
161         address user = msg.sender;
```

Bug ID	File Location	Line No.	Action Taken
SSB_1336915_32	contracts/Codeup.sol	L224 - L224	⚠ *Pending Fix*

contracts/Codeup.sol L224 - L224

```
223
224         uint256 gameETH = amount / gameETHPrice;
225         _checkValue(gameETH);
226         uint256 totalInvestedBefore = totalInvested;
```

Issue Type

IF-STATEMENT REFACTORING

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

Description

In Solidity, we aim to write clear, efficient code that is both easy to understand and maintain. If statements can be co nverted to ternary operators. While using ternary operators instead of if/else statements can sometimes lead to more concise code, it's crucial to understand the trade-offs involved.

Bug ID	File Location	Line No.	Action Taken
SSB_1336915_30	contracts/Codeup.sol	L340 - L344	*Pending Fix*

contracts/Codeup.sol L340 - L344

```
339
340             if (count == 40) {
341                 return true;
342             } else {
343                 return false;
344             }
345         }
346
```

Issue Type

MISSING UNDERSCORE IN NAMING VARIABLES

S. No.	Severity	Detection Method	Instances
I002	Informational	Automated	4

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_1336915_13	contracts/Codeup.sol	L35 - L35	*Pending Fix*

contracts/Codeup.sol L35 - L35

```
34        /// @notice CodeupERC20 token amount for winner
35        uint256 private constant TOKEN_AMOUNT_FOR_WINNER = 1 ether;
36        /// @notice Token amount in ETH needed for first luqidity
37        uint256 private constant MAX_FIRST_LIQUIDITY_AMOUNT = 0.001 ether;
```

Bug ID	File Location	Line No.	Action Taken
SSB_1336915_14	contracts/Codeup.sol	L37 - L37	*Pending Fix*

contracts/Codeup.sol L37 - L37

```
36        /// @notice Token amount in ETH needed for first luqidity
37        uint256 private constant MAX_FIRST_LIQUIDITY_AMOUNT = 0.001 ether;
38        /// @notice Amount of game token for first liquidity
39        uint256 private constant FIRST_LIQUIDITY_GAME_TOKEN = 10 ether;
```

Bug ID	File Location	Line No.	Action Taken
SSB_1336915_15	contracts/Codeup.sol	L39 - L39	*Pending Fix*

contracts/Codeup.sol L39 - L39

```
38        /// @notice Amount of game token for first liquidity
39        uint256 private constant FIRST_LIQUIDITY_GAME_TOKEN = 10 ether;
40        /// @notice Withdraw commission 33% for rewards pool, 33% for liquidity pool
41        uint256 private constant WITHDRAW_COMMISSION = 66;
```

Bug ID	File Location	Line No.	Action Taken
SSB_1336915_16	contracts/Codeup.sol	L41 - L41	⚠ *Pending Fix*

contracts/Codeup.sol L41 - L41

```
40        /// @notice Withdraw commission 33% for rewards pool, 33% for liquidity pool
41        uint256 private constant WITHDRAW_COMMISSION = 66;
42
43        /// @notice UniswapV2Router address
```

Issue Type

NAME MAPPING PARAMETERS

S. No.	Severity	Detection Method	Instances
I003	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_1336915_17	contracts/Codeup.sol	L67 - L67	*Pending Fix*

contracts/Codeup.sol L67 - L67

```
66        /// @notice account claim status
67        mapping(address => bool) public isClaimed;
68        /// @notice User's tower info
69        mapping(address => Tower) public towers;
```

Bug ID	File Location	Line No.	Action Taken
SSB_1336915_18	contracts/Codeup.sol	L69 - L69	⚠ *Pending Fix*

contracts/Codeup.sol L69 - L69

```
68        /// @notice User's tower info
69        mapping(address => Tower) public towers;
70
71        /// @notice Error messages
```

Issue Type

GAS OPTIMIZATION IN INCREMENTS

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

Description

`++i` costs less gas compared to `i++` or `i += 1` for unsigned integers. In `i++`, the compiler has to create a temporary variable to store the initial value. This is not the case with `++i` in which the value is directly incremented and returned, thus, making it a cheaper alternative.

Bug ID	File Location	Line No.	Action Taken
SSB_1336915_19	contracts/Codeup.sol	L336 - L336	*Pending Fix*

contracts/Codeup.sol L336 - L336

```
335         uint8 count;
336         for (uint8 i = 0; i < 8; i++) {
337             count += builders[i];
338         }
```

Issue Type

UNNECESSARY MEMORY OPERATIONS WITH AN IMMUTABLE VARIABLE

S. No.	Severity	Detection Method	Instances
G002	Gas	Automated	3

Description

`Immutable` variables are different from storage variables. Their instances get replaced in the code with their value. Hence caching operation is unnecessary and costs extra gas.

Bug ID	File Location	Line No.	Action Taken
SSB_1336915_2	contracts/Codeup.sol	L268 - L268	*Pending Fix*

contracts/Codeup.sol L268 - L268

```
267         address currentContract = address(this);
268         address wethMemory = weth;
269         address codeupERC20Memory = codeupERC20;
270         address routerMemory = uniswapV2Router;
```

Bug ID	File Location	Line No.	Action Taken
SSB_1336915_3	contracts/Codeup.sol	L269 - L269	*Pending Fix*

contracts/Codeup.sol L269 - L269

```
268         address wethMemory = weth;
269         address codeupERC20Memory = codeupERC20;
270         address routerMemory = uniswapV2Router;
271         uint256 wethBalance = IERC20(wethMemory).balanceOf(currentContract);
```

Bug ID	File Location	Line No.	Action Taken
SSB_1336915_4	contracts/Codeup.sol	L270 - L270	*Pending Fix*

contracts/Codeup.sol L270 - L270

```
269         address codeupERC20Memory = codeupERC20;
270         address routerMemory = uniswapV2Router;
271         uint256 wethBalance = IERC20(wethMemory).balanceOf(currentContract);
272
```

06. Scan History

Critical High Medium Low Informational Gas

No	Date	Security Score	Scan Overview
1.	2024-09-30	98.48	0 0 2 0 7 4

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.