

# RollBitPepe (\$ROB)

v0.8.19+commit.7dd6d404 v0.8.13

→ Low-Risk

Low-risk code

→ Medium-Risk

Medium-risk code

+ High-Risk

High-risk code

RollBitPepe

0x617B678DDbde93Fd165FBe7Be29CE1485b1f0baA

[Disclaimer

AuditBlock is not liable for any financial losses incurred as a result of its services. The information provided in this contract audit should not be considered financial advice. Please conduct your own research to make informed decisions.

## Executive Summary

## **Project Name**

RollBitPepe

Overview

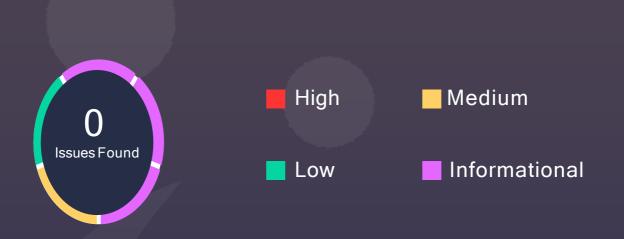
The RollBitPepe ecosystem is designed to provide players with an immersive and rewarding experience. The ecosystem is constantly evolving, and new features and benefits are being added all the time

### Method

Manual Review, Functional Testing, Automated Testing etc.

## **Scope of Audit**

The scope of this audit was to analyze the contract codebase for quality, security, and correctness.



	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	0	0	0
Partially Resolved Issues	0	0	0	0
Resolved Issues	0	0	0	0

# Smart Contract Weakness Classification (SWC) Vulnerabilities for Attacks

Tautology or contradiction Re-entrancy Timestamp Dependence Missing Zero Address Validation Gas Limit and Loops Return values of low-level calls **Exception Disorder** Revert/require functions Gasless Send Private modifier Use of tx.origin Using block.timestamp Multiple Sends Compiler version not fixed Address hardcoded Using SHA3 Divide before multiply Using suicide Integer overflow/underflow Using throw Using inline assembly Dangerous strict equalities

## **Types of Severities**

## High

A high-severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

## Medium

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

### Low

Low-level severity issues can cause minor impact and or are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

## Informational

These are severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

## **Techniques and Methods**

The overall quality of code.

- · Use of best practices.
- · Code documentation and comments match logic and expected behavior.
- Token distribution and calculations are as per the intended behavior mentioned in the whitepaper.
- implementation of ERC-20 token standards.
- · Efficient use of gas.
- · Code is safe from re-entrance and other vulnerabilities.

The following techniques, methods, and tools were used to review all the smart contracts.

#### **Structural Analysis**

In this step, we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

### **Static Analysis**

Static analysis of smart contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

#### Code Review / Manual Analysis

Manual analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, and their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

#### **Gas Consumption**

In this step, we have checked the behavior of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

#### **Tools and Platforms Used for Audit**

Remix IDE, Truffle, Truffle Team, Solhint, Mythril, Slither, Solidity statistic analysis.

## Phase 1

Project - RollBitPepe

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

1. Ownership Methods (manipulate ownership)

```
transferOwnership(initialOwner);
}
function mint(address account, uint256 amount) external onlyOwner {
    _mint(account, amount);
}

function changeOwner(address newOwner) external onlyOwner {
    transferOwnership(newOwner);
```

#### Description

Our auditor found this issue manually. It is not a bug or vulnerability. We are simply acknowledging that the contract already uses OpenZeppelin's ownership support contract, and the contractor again defines the initial ownership slot. This may cause damage or danger.

#### Recommendation

It is important to note that not define the ownership with external calls. You can use the Openzeppelin ownership contract. Which you have already used! However, staying away from calling ownership with external calls. Just use a modifier for that. Using modifiers is good practice.

**Status** 

Acknowledged

**Informational Severity Issues** 

No issues found

## Phase 2

```
(contracts/RollBitPepe.sol#9) shadows:
        - ERC20.name()
(node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#62-64)
        - IERC20Metadata.name()
(node modules/@openzeppelin/contracts/token/ERC20/extensions/IERC20Metada
(contracts/RollBitPepe.sol#10) shadows:
       - ERC20.symbol()
(node modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#70-72)
(node modules/@openzeppelin/contracts/token/ERC20/extensions/IERC20Metada
Reference: https://github.com/crytic/slither/wiki/Detector-
Documentation#local-variable-shadowing
(node modules/@openzeppelin/contracts/access/Ownable.sol#4)
(node modules/@openzeppelin/contracts/token/ERC20/IERC20.sol#4)
(node modules/@openzeppelin/contracts/token/ERC20/extensions/ERC20Burnabl
(node modules/@openzeppelin/contracts/token/ERC20/extensions/IERC20Metada
Reference: https://github.com/crytic/slither/wiki/Detector-
```

Pragma version^0.8.0

(node\_modules/@openzeppelin/contracts/access/Ownable.sol#4) allows old versions
Pragma version^0.8.0

(node\_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#4) allows old versions

Pragma version^0.8.0

(node\_modules/@openzeppelin/contracts/token/ERC20/IERC20.sol#4) allows old versions

Pragma version^0.8.0

(node\_modules/@openzeppelin/contracts/token/ERC20/extensions/ERC20Burnable.sol#
4) allows old versions

Pragma version^0.8.0

(node\_modules/@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol
#4) allows old versions

Pragma version^0.8.0 (node\_modules/@openzeppelin/contracts/utils/Context.sol#4) allows old versions

Pragma version^0.8.13 (contracts/RollBitPepe.sol#2) allows old versions solc-0.8.17 is not recommended for deployment

Reference: <a href="https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity">https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity</a>

## **Functional Testing**

## Some of the tests performed are mentioned below:

- ✓ Should revert when non-owner calls the changedOwner
- Should be applied initial supply
- ✓ Should revert when none-owner calls transform-from
- ✓ Should work mint method correctly
- Should mint to any address

## **Closing Summary**

In this report, we have considered the security of RollBitPepe. We performed our audit according to the procedure described above.

Several issues were identified during the audit process, and their severity levels have been classified. Recommendations and best practices have also been provided to enhance code quality and security posture. The team has acknowledged all identified issues.

## **Disclaimer**

AuditBlock does not provide security warranties, investment advice, or endorsements of any platform. This audit does not guarantee the security or correctness of the audited smart contracts. The statements made in this document should not be interpreted as investment or legal advice. The authors are not liable for any decisions made based on the information in this document. Securing smart contracts is an ongoing process. A single audit is not sufficient. We recommend that the platform's development team implement a bug bounty program to encourage further analysis of the smart contract by other third parties

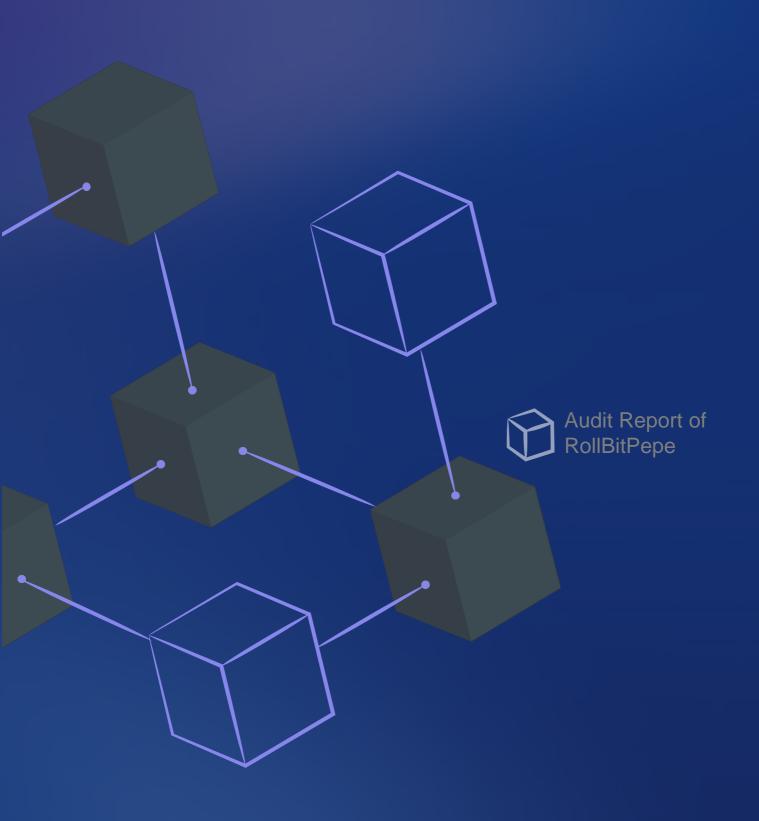
## **AuditBlock**

AuditBlock is a blockchain security company that provides professional services and solutions for securing blockchain projects. They specialize in smart contract audits on various blockchains and offer a range of services









<u>auditblock@gmail.com</u>

