WeBump

Token

# SMART CONTRACT AUDIT REPORT

IMMUNEBYTES

24 MAY 2023

# Introduction

## About WeBump

WeBump's platform guarantees creators earn royalties for book trades. Smart contracts enable immediate payment transfers, eliminating waiting times. Authors receive higher splits and residuals on book resales. The AMM uses smart contracts and liquidity providers to set book prices after the official sale. Smart contracts allow ebook reselling, swapping, and physical pickup. Decentralization ensures security and unique editions. We prioritize transparency and secure funds for readers and collectors. WeBump's ecosystem eliminates embezzlement risk and builds trust. ERC20 Cards provide liquidity and enable token and ebook swapping. They are stored in crypto wallets, minimizing the chance of loss. Blockchain transactions are instant and transparent.

Visit https://webump.io/ to know more about it.

## About ImmuneBytes

ImmuneBytes is a security start-up that provides professional services in the blockchain space. The team has hands-on experience conducting smart contract audits, penetration testing, and security consulting. ImmuneBytes's security auditors have worked on various A-league projects and understand DeFi projects like AAVE, Compound, 0x Protocol, Uniswap, and dydx.

The team has secured 205+ blockchain projects by providing security services on different frameworks. The ImmuneBytes team helps start-ups with detailed system analysis, ensuring security and managing the overall project.

Visit http://immunebytes.com/ to learn more about the services.

# Documentation Details

The team has provided the following doc for audit:

1. https://github.com/WeBump/WeBumpIo/blob/main/Whitepaper_1.0_WeBump.pdf

## Audit Goals

The audit focused on verifying that the smart contract system is secure, resilient, and working according to its specifications. The audit activities can be grouped into the following three categories:

1. Security: Identifying security-related issues within each contract and the system of contracts.
2. Sound Architecture: Evaluation of the architecture of this system through the lens of established smart contract best practices and general software best practices.
3. Code Correctness and Quality: A full review of the contract source code. The primary areas of focus include

    a. Correctness
    b. Readability
    c. Sections of code with high complexity
    d. Quantity and quality of test coverage

## Audit Process & Methodology

ImmuneBytes team have performed thorough testing of the project, starting with analyzing the code design patterns in which we reviewed the smart contract architecture to ensure it is structured and safely used by third-party smart contracts and libraries.

Our team then performed a formal line-by-line inspection of the Smart Contract to find potential issues like Signature Replay Attacks, Unchecked External Calls, External Contract Referencing, Variable Shadowing, Race conditions, Transaction-ordering dependence, timestamp dependence, DoS attacks, and others.

In the Unit testing phase, we run unit tests written by the developer to verify the functions work as intended. In Automated Testing, we tested the Smart Contract with our in-house developed tools to identify vulnerabilities and security flaws.

A team of independent auditors audited the code, including -

1. Testing the functionality of the Smart Contract to determine proper logic has been followed throughout.
2. Analyzing the complexity of the code by thorough, manual review of the code, line-by-line.
3. Deploying the code on testnet using multiple clients to run live tests.
4. Analyzing failure preparations to check how the Smart Contract performs in case of bugs and vulnerabilities.
5. Checking whether all the libraries used in the code are on the latest version.
6. Analyzing the security of the on-chain data.

## Audit Details

| | |
|---|---|
| **Project Name** | WeBump |
| **Platform** | Polygon |
| **Languages** | Solidity |
| **GitHub Link** | https://gist.github.com/WojcikMM/1c32622dfc0df217172e52ad6a5feaca |
| **Platforms & Tools** | Remix IDE, Truffle, VScode, Contract Library, Slither, SmartCheck, Fuzz |

## Security Level References

Every issue in this report was assigned a severity level from the following:

**CRITICAL**

Issues may result in fund leakage or incorrect fund allocation.

**HIGH**

Issues affecting the business logic, performance, and functionality.

**MEDIUM**

Issues could lead to data loss or other manipulations.

**LOW**

Issues around smart contract code upgradability, libraries, and others.

**INFORMATIONAL**

Issues which can further improve the code on gas optimizations and reusability.

**OPEN**

The issue hasn't been fixed as per our recommendation(s) and still stays pertinent.

**FIXED**

The issue has been agreed upon to be true and fixed as per our recommendation(s).

**ACKNOWLEDGED**

The issue is an intended behaviour and doesn't require a change as per our recommendation(s).

| Issues | Critical | High | Medium | Low | Informational |
|---|---|---|---|---|---|
| Open | - | - | - | - | - |
| Fixed | - | - | - | - | 1 |
| Acknowledged | - | - | - | - | - |

This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

## Finding

| # | Findings | Risk | Status |
|---|---|---|---|
| 1 | Unlocked Pragma statements found in the contract | **Informatory** | **Fixed** |

## Critical Severity Issues

**No issues were found.**

## High Severity Issues

**No issues were found.**

## Medium severity issues

**No issues were found.**

## Low severity issues

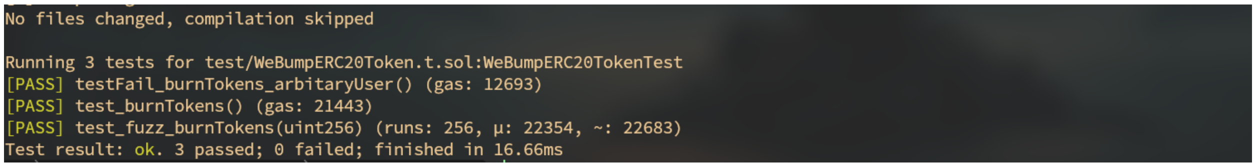
**No issues were found.**

## Informational

| **1. Unlocked Pragma statements found in the contract** |
|---|
| **Description:**<br><br>During the code review, it was found that the contracts included unlocked Pragma solidity version statements. It's not considered a better practice in Smart contract development as it might lead to accidental deployment to a version with unfixed bugs. |
| **Recommendation:**<br>It's always recommended to lock pragma statements to a specific version while writing contracts. |
| **Status: Fixed** |

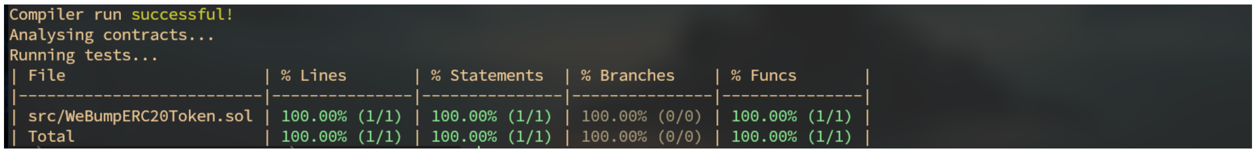
## Automated Test Results

### Auditor Test Cases



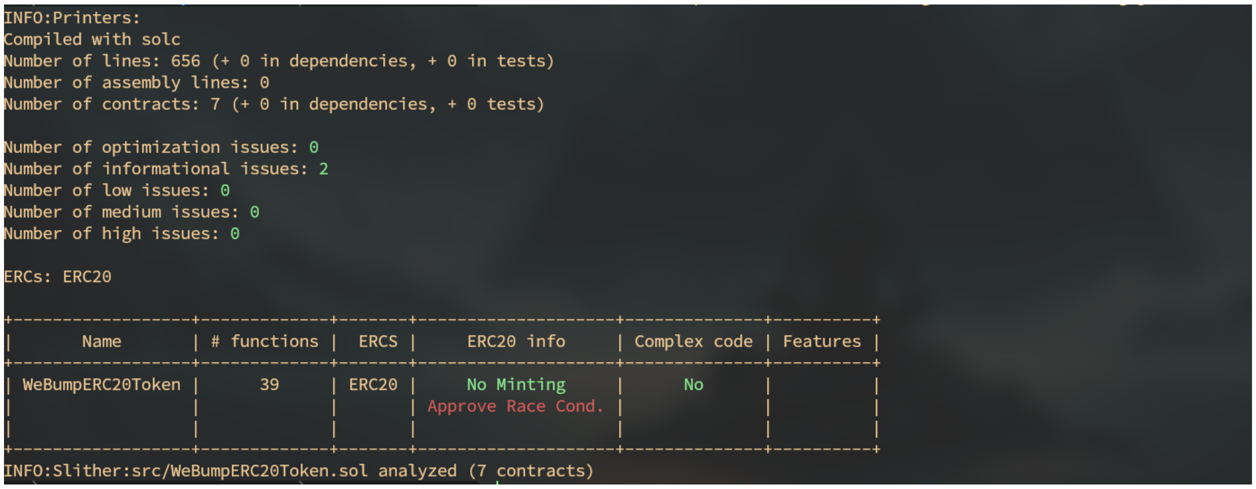
```
No files changed, compilation skipped

Running 3 tests for test/WeBumpERC20Token.t.sol:WeBumpERC20TokenTest
[PASS] testFail_burnTokens_arbitaryUser() (gas: 12693)
[PASS] test_burnTokens() (gas: 21443)
[PASS] test_fuzz_burnTokens(uint256) (runs: 256, μ: 22354, ~: 22683)
Test result: ok. 3 passed; 0 failed; finished in 16.66ms
```

### Code Coverage



```
Compiler run successful!
Analysing contracts...
Running tests...
| File                      | % Lines       | % Statements  | % Branches    | % Funcs       |
|---------------------------|---------------|---------------|---------------|---------------|
| src/WeBumpERC20Token.sol  | 100.00% (1/1) | 100.00% (1/1) | 100.00% (0/0) | 100.00% (1/1) |
| Total                     | 100.00% (1/1) | 100.00% (1/1) | 100.00% (0/0) | 100.00% (1/1) |
```

### Slither



```
INFO:Printers:
Compiled with solc
Number of lines: 656 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 7 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 0
Number of informational issues: 2
Number of low issues: 0
Number of medium issues: 0
Number of high issues: 0

ERCs: ERC20

+------------------+-------------+-------+--------------------+--------------+----------+
|       Name       | # functions |  ERCS |     ERC20 info     | Complex code | Features |
+------------------+-------------+-------+--------------------+--------------+----------+
| WeBumpERC20Token |      39     | ERC20 |     No Minting     |      No      |          |
|                  |             |       | Approve Race Cond. |              |          |
|                  |             |       |                    |              |          |
+------------------+-------------+-------+--------------------+--------------+----------+
INFO:Slither:src/WeBumpERC20Token.sol analyzed (7 contracts)
```

This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

## Concluding Remarks

While conducting the audits of WeBump, it was observed that the contracts contain no Critical, High, Medium, and Low severity issues.

## Disclaimer

ImmuneBytes' audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process; therefore, running a bug bounty program complementing this audit is strongly recommended.

Our team does not endorse the WeBump platform or its product, nor is this audit investment advice.
Notes:

- Please make sure contracts deployed on the mainnet are the ones audited.
- Check for code refactoring by the team on critical issues.

IMMUNEBYTES

AUDITS