BALLS Token

Smart Contract Security Assessment

Oct 9, 2024

VERACITY

Disclaimer

Veracity Security (“Veracity”) has conducted an independent audit to verify the integrity of and highlight any vulnerabilities or errors, intentional or unintentional, that may be present in the codes that were provided for the scope of this audit. This audit report does not constitute agreement, acceptance or advocacy for the Project that was audited, and users relying on this audit report should not consider this as having any merit for financial advice in any shape, form or nature.

The contracts audited do not account for any economic developments that may be pursued by the Project in question, and that the veracity of the findings thus presented in this report relate solely to the proficiency, competence, aptitude and discretion of our independent auditors, who make no guarantees nor assurance that the contracts are completely free of exploits, bugs, vulnerabilities or deprecation of technologies. Further, this audit report shall not be disclosed nor transmitted to any persons or parties on any objective, goal or justification without due written assent, acquiescence or approval by the Veracity team.

All information provided in this report does not constitute financial or investment advice, nor should it be used to signal that any person reading this report should invest their funds without sufficient individual due diligence regardless of the findings presented in this report. Information is provided ‘as is’, and Veracity is under no covenant to the completeness, accuracy or solidity of the contracts audited. In no event will Veracity or its partners, employees, agents or parties related to the provision of this audit report be liable to any parties for, or lack thereof, decisions and/or actions with regards to the information provided in this audit report.

Cryptocurrencies and any technologies by extension directly or indirectly related to cryptocurrencies are highly volatile and speculative by nature. All reasonable due diligence and safeguards may yet be insufficient, and users should exercise considerable caution when participating in any shape or form in this nascent industry.

The audit report has made all reasonable attempts to provide clear and articulate recommendations to the Project team with respect to the rectification, amendment and/or revision of any highlighted issues, vulnerabilities or exploits within the contracts provided. It is the sole responsibility of the Project team to sufficiently test and perform checks, ensuring that the contracts are functioning as intended, specifically that the functions contained within said contracts have the desired intended effects, functionalities and outcomes of the Project team.

1 Overview

This report has been prepared for **BALLS Token project** Veracity provides an examination of the smart contracts to look for vulnerabilities, logic errors or other issues from both an internal and external perspective. The scope of this audit is the initial raise contract Deposit.sol, which includes industry standard libraries from OpenZepplin.

1.1 Summary

Name	BALLS Token
URL	https://buy.balls.xyz/
Platform	Ethereum
Language	Solidity

BALLS Token consists of 2 contracts:

balls_eth.sol	https://github.com/balls-xyz/balls-contracts/blob/main/contracts/balls_eth.sol
balls_token.sol	https://github.com/balls-xyz/balls-contracts/blob/main/contracts/balls_token.sol

1.2 Testing

Following an initial pass on all contracts, we performed a series of tests. However it is not possible to catch all scenarios with these tests. Veracity has implemented a suite of audit tests that also exercise the primary functions of each contract to ensure that no transaction or fund locking occurs.

Tests have been implemented with the Foundry fuzz testing framework and some of the issues discovered are listed in the tables below. No further critical issues were discovered during this secondary process.

1.3 Final Contracts Assessed

Following deployment of the contracts assessed, Veracity compares the contracts that have been deployed, and wired with the contracts that have been audited to guarantee no tampering has been possible between audit report issue and project start.

This gives project owners and community members confidence that what has been deployed matches the findings and resolution status described in this document.

https://github.com/balls-xyz/balls-contracts/

Github hash: fc154b8

Deployment network: Ethereum x 1

Project wallet address:

Links to verified contracts (2):

Name	Address	Network	Matched
balls_eth.sol	ETH:	Ethereum,	**NO**
balls_token.sol			

1.4 Findings Summary

Individual issues found have been categorised based on criticality as high, medium, low or informational. The client is required to respond to each issue individually, although it may be by design and therefore simply acknowledged. Additional recommendations may apply to all contracts, but are replicated for each for resolution.

For example an issue relating to centralisation of financial risk may apply to all administration functions, but will be included only once per contract. The table below shows the collected number of issues found and the resolution statuses across all contracts in the project.

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change)
● High	1	1	0	0
● Medium	2	2	0	0
● Low	3	3	0	0
● Informational	0	0	0	0
Total	6	0	0	0

1.4.1 Status Classifications

Severity	Description
● High	Exploits, vulnerabilities or errors that will certainly or probabilistically lead towards loss of funds, control, or impairment of the contract and its functions. Issues under this classification are recommended to be fixed with utmost urgency.
● Medium	Bugs or issues with that may be subject to exploit, though their impact is somewhat limited. Issues under this classification are recommended to be fixed as soon as possible.
● Low	Effects are minimal in isolation and do not pose a significant danger to the project or its users. Issues under this classification are recommended to be fixed nonetheless.
● Informational	Consistency, syntax or style best practices. Generally pose a negligible level of risk, if any.
● Optimization	Suboptimal implementations that may result in additional gas consumption, unnecessary computation or avoidable inefficiencies.

1.4.2 Collected Issues and Statuses

ID	Contract	Severity	Summary	Status
01	balls_eth.sol	LOW	The withdrawFunds() function is unnecessary as the contract receives USDT, not native Ether (ETH).	RESOLVED
02	balls_eth.sol and balls_token.sol	MEDIUM	The contract has fallback() and receive() functions allowing it to receive native funds (e.g., ETH) unintentionally. Users might accidentally send native currency, which won't be recoverable as the contract lacks logic to handle or refund such funds.	RESOLVED
03	balls_token.sol	LOW	The Wrapped interface is declared but never used in the contract.	RESOLVED
04	balls_token.sol	HIGH	Multiplying howManyWholePairTokens by ERC20(pairToken).decimals() is incorrect. The decimals() function returns the number of decimals (e.g., 18), but the intended behavior likely requires multiplying by 10**decimals() to account for the token's precision.	RESOLVED
05	balls_eth.sol and balls_token.sol	LOW	The ERC20 Token; line in the contract is redundant. This declaration is not used anywhere within the contract, and it does not follow proper Solidity declaration syntax, leading to potential confusion without affecting the functionality.	RESOLVED
06	balls_eth.sol and balls_token.sol	MEDIUM	The line below performs multiplication after division: require(balanceOf[_to] <= maxWalletPercent*(totalSupply/100),...	RESOLVED

2. Findings

The contract(s) assessed have been largely authored from scratch rather than using industry tested implementations for ERC20. Standard interfaces have been included inline which adds risk of errors, however code comparison shows no errors have been introduced during this process. This can result in the introduction of vulnerabilities or bugs that have not been seen or addressed in previous projects. However our team has made recommendations and several code sweeps to mitigate the effect of not using industry standard libraries. The following sections outline issues found with individual contracts.

2.1 BallsToken, BallsEth

This report has been prepared for the **BALLS Token project**. Veracity provides an examination of the smart contracts to look for vulnerabilities, logic errors or other issues from both an internal and external perspective.

2.1.1 Privileged Roles

The following functions can be called by the admin or manager role of both contracts:

BallsToken, BallsEth contracts have the following privileged functions:

- setLPtoken
- createPool
- createPoolsetLPtokenAndEnableTrading
- configImmuneToMaxWallet
- configureTrading

2.1.2 Initial Token Allocation

BallsToken, BallsEth

1 Million tokens are allocated to the contract deployer (declared as admin).

2.1.3 Taxes, Rules, Initial Variables.

This is not applicable to this contract.

2.1.4 Deposit Issues & Recommendations

Issue Number: 1

Title: Unnecessary Variable `order[]` Declaration

Severity: Low

Contract: balls_token.sol

Files:
https://github.com/balls-xyz/balls-contracts/blob/main/contracts/balls_eth.sol#L41

Summary: The `order[]` array is declared, costs gas, but is private and never used.

Proposed Fix:
Remove the `order[]` array and its usage in the constructor to save gas.

Resolution:

Proposed fix implemented.

Issue Number: 2

Severity: Medium

Contract: balls_eth.sol and balls_token.sol

Files:
https://github.com/balls-xyz/balls-contracts/blob/main/contracts/balls_eth.sol
https://github.com/balls-xyz/balls-contracts/blob/main/contracts/balls_token.sol

Summary:

The contract has `fallback()` and `receive()` functions allowing it to receive native funds (e.g., ETH) unintentionally. Users might accidentally send native currency, which won't be recoverable as the contract lacks logic to handle or refund such funds.

Code:

```
fallback() external payable {}
receive() external payable {}
```

Proposed Fix:
Remove the `fallback()` and `receive()` functions to prevent the contract from accepting native funds unintentionally, ensuring users don't lose funds due to accidental transfers.

Resolution:

Proposed fix implemented.

Issue Number: 3

Severity: Low

Contract: balls_token.sol

Files:

https://github.com/balls-xyz/balls-contracts/blob/main/contracts/balls_token.sol

Summary:
The `Wrapped` interface is declared but never used in the contract.

Code:

```
interface Wrapped {
    function deposit() external payable;
    function withdraw(uint) external;
}
```

Proposed Fix:
Remove the `Wrapped` interface to reduce unnecessary code.

Resolution:

Proposed fix implemented.

Issue Number: 4

Severity: High

Contract: balls_token.sol

File:

https://github.com/balls-xyz/balls-contracts/blob/main/contracts/balls_token.sol#L69

https://github.com/balls-xyz/balls-contracts/blob/main/contracts/balls_token.sol#L85

Summary:
Multiplying `howManyWholePairTokens` by `ERC20(pairToken).decimals()` is incorrect. The `decimals()` function returns the number of decimals (e.g., `18`), but the intended behavior likely requires multiplying by `10**decimals()` to account for the token's precision.

Code:

```
howManyWholePairTokens *= ERC20(pairToken).decimals();
```

Proposed Fix:

Replace with:

```
howManyWholePairTokens *= 10 ** ERC20(pairToken).decimals();
```

Resolution:

Proposed fix implemented.

Issue Number: 5

Title: Redundant `ERC20 Token` Declaration

Severity: Low

Contract: balls_eth.sol and balls_token.sol

Files:
https://github.com/balls-xyz/balls-contracts/blob/main/contracts/balls_eth.sol
https://github.com/balls-xyz/balls-contracts/blob/main/contracts/balls_token.sol

Summary:
The `ERC20 Token;` line in the contract is redundant. This declaration is not used anywhere within the contract, and it does not follow proper Solidity declaration syntax, leading to potential confusion without affecting the functionality.

Proposed Fix:
Remove the redundant `ERC20 Token;` declaration to clean up the contract code and eliminate any unnecessary components that could lead to misunderstanding or clutter in the codebase.

Resolution:

Proposed fix implemented.

Issue Number: 6

Title: Loss of precision due to division before multiplication in both contracts

Severity: Medium

Contract: balls_eth.sol and balls_token.sol

Files:
https://github.com/balls-xyz/balls-contracts/blob/main/contracts/balls_eth.sol#L134
https://github.com/balls-xyz/balls-contracts/blob/main/contracts/balls_token.sol#L143

Summary:
The line below performs multiplication after division:

```
require(balanceOf[_to] <= maxWalletPercent*(totalSupply/100),...
```

Proposed Fix:
To ensure no precision is lost, the multiplication should occur before the division. By doing so, the integer division occurs as the last step, minimizing precision loss.

```
require(balanceOf[_to] <= (maxWalletPercent * totalSupply) /
100,...
```

Resolution:

Proposed fix implemented.