Zipper Smart Contract Audit Report

Preamble

This audit report was undertaken by BlockchainLabs.nz for the purpose of providing feedback to Zipper.
It has subsequently been shared publicly without any express or implied warranty.

Solidity contracts were sourced from the public Github repo zipperglobal/zipt_token/ prior to commit 0a3c4238353122cfc76b9fb01cd60b28b8b0c9e4 - we would encourage all community members and token holders to make their own assessment of the contracts.

Scope

The following contract was a subject for static, dynamic and functional analyses:

- ZipToken.sol

There is also javascript used to run the contract functions:

- deploy_zipt.js

Focus areas

The audit report is focused on the following key areas - though this is not an exhaustive list.

Correctness

- No correctness defects uncovered during static analysis?
- No implemented contract violations uncovered during execution?
- No other generic incorrect behaviour detected during execution?
- Adherence to adopted standards such as ERC20?

Testability

- Test coverage across all functions and events?
- Test cases for both expected behaviour and failure modes?
- Settings for easy testing of a range of parameters?
- No reliance on nested callback functions or console logs?
- Avoidance of test scenarios calling other test scenarios?

Security

- No presence of known security weaknesses?
- No funds at risk of malicious attempts to withdraw/transfer?
- No funds at risk of control fraud?
- Prevention of Integer Overflow or Underflow?

Best Practice

- Explicit labeling for the visibility of functions and state variables?
- Proper management of gas limits and nested execution?
- Latest version of the Solidity compiler?

Analysis

- Static analysis

Issues

Severity Description

Minor	A defect that does not have a material impact on the contract execution and is likely to be subjective.
Moderate	A defect that could impact the desired outcome of the contract execution in a specific scenario.
Major	A defect that impacts the desired outcome of the contract execution or introduces a weakness that may be exploited.
Critical	A defect that presents a significant security vulnerability or failure of the contract across a range of scenarios.

Minor

- Unnecessary gas spent when avoiding double distribution - `Best practice` Lines 23
  There is a check if the account has a zero balance. The cost of each check is a 400 gas. Is there any business reason for that? We assumed that this check was implemented to avoid double sending to the account. Consider the following scenario: - The function called twice and between that calls (highly unlikely) the malicious actor transferred tokens to another ... View on GitHub

  ✓ Fixed 3ca6051a

- Add Transfer event to ZipToken after minting tokens - `Best practice`
  It is highly recommended to add `Transfer(0x0, msg.sender, INITIAL_SUPPLY);` event beneath the following line number: #L15 The reason for this is so that token transfers can be seen on Etherscan and other block explorers... View on GitHub

  ✓ Fixed 791425

- Checksum addresses before sending tokens - `Enhancement`
  We recommend validating addresses are in the correct format to avoid any unforeseen problems with sending tokens to the correct addresses when passing in data from ziptc.txt to deploy_zipt.js View on GitHub

  - [x] Fixed 57103b

- Variable is declared but not used - `Best practices`, `Question`
  Check if the `bool public filled = false;` is redundant. View on GitHub

- deploy_zipt.js - variable is declared but not used - `Correctness`
  BATCH_SIZE never used but another variable with the same functionality is declared View on GitHub

- Optimal gas prices for deploy_zipt.js - `Best practices`
  it is worth to check the gas price right before running deploy_zipt.js to avoid both overspending and crash because of low gas price. View on GitHub

Moderate

- None found

Major

- None found

Critical

- None found

Observations

As a part of the deployment of the audited smart contracts the Zipper team will use Javascript to both deploy and distribute the Zipper token. While this process will decrease the amount of gas required, it requires off-chain collection and processing of user data that may not be as immutable as an on-chain approach.

Conclusion

The developers demonstrated an understanding of Solidity and smart contracts. They were receptive to the feedback provided to help improve the robustness of the contracts.

We took part in carefully reviewing all source code provided, including both static and dynamic testing methodology.

Overall we consider the resulting contracts following the audit feedback period adequate and have not identified any potential vulnerabilities. This contract has a low level risk of ETH and ZIPT being hacked or stolen from the inspected contracts.

Disclaimer

Our team uses our current understanding of the best practises for Solidity and Smart Contracts. Development in Solidity and for Blockchain is an emerging area of software engineering which still has a lot of room to grow, hence our current understanding of best practise may not find all of the issues in this code and design.

We have not analysed any of the assembly code generated by the Solidity compiler. We have not verified the deployment process and configurations of the contracts. We have only analysed the code outlined in the scope. We have not verified any of the claims made by any of the organisations behind this code.

Security audits do not warrant bug-free code. We encourge all users interacting with smart contract code to continue to analyse and inform themselves of any risks before interacting with any smart contracts.