SMART CONTRACTS REVIEW

zokyo

April 11th 2025 | v. 1.0

Security Audit Score

PASS

Zokyo Security has concluded that these smart contracts passed a security audit.

ZOKYO AUDIT SCORING FLOWZED

1. Severity of Issues:
    - Critical: Direct, immediate risks to funds or the integrity of the contract. Typically, these would have a very high weight.
    - High: Important issues that can compromise the contract in certain scenarios.
    - Medium: Issues that might not pose immediate threats but represent significant deviations from best practices.
    - Low: Smaller issues that might not pose security risks but are still noteworthy.
    - Informational: Generally, observations or suggestions that don't point to vulnerabilities but can be improvements or best practices.
2. Test Coverage: The percentage of the codebase that's covered by tests. High test coverage often suggests thorough testing practices and can increase the score.
3. Code Quality: This is more subjective, but contracts that follow best practices, are well-commented, and show good organization might receive higher scores.
4. Documentation: Comprehensive and clear documentation might improve the score, as it shows thoroughness.
5. Consistency: Consistency in coding patterns, naming, etc., can also factor into the score.
6. Response to Identified Issues: Some audits might consider how quickly and effectively the team responds to identified issues.

HYPOTHETICAL SCORING CALCULATION:

Let's assume each issue has a weight:
- Critical: -30 points
- High: -20 points
- Medium: -10 points
- Low: -5 points
- Informational: -1 point

Starting with a perfect score of 100:
- 0 Critical issues: 0 points deducted
- 0 High issues: 0 points deducted
- 3 Medium issues: 3 resolved = 0 points deducted
- 4 Low issues: 3 resolved and 1 acknowledged = - 1 points deducted
- 0 Informational issues: 0 points deducted

Thus, 100 - 1 = 99

TECHNICAL SUMMARY

This document outlines the overall security of the FlowZed smart contract/s evaluated by the Zokyo Security team.

The scope of this audit was to analyze and document the FlowZed smart contract/s codebase for quality, security, and correctness.

Contract Status

There were 0 critical issues found during the review. (See Complete Analysis)

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract/s but rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that can withstand the Ethereum network's fast-paced and rapidly changing environment, we recommend that the FlowZed team put in place a bug bounty program to encourage further active analysis of the smart contract/s.

Table of Contents

AUDITING STRATEGY AND TECHNIQUES APPLIED

The source code of the smart contract was taken from the FlowZed repository:
Repo: https://github.com/0xFlowZed/sc-audit/commit/9414477aab6477054e9753cf3aead2d092eb84c4

Last commit: 1fbc370c5b580a2a83d07474c5bd4b95c1bd1281

Within the scope of this audit, the team of auditors reviewed the following contract(s):

- ./zedpass1×0.sol
- ./flowzed1×0.sol

During the audit, Zokyo Security ensured that the contract:

- Implements and adheres to the existing standards appropriately and effectively;
- The documentation and code comments match the logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices, efficiently using resources without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the most recent vulnerabilities;
- Meets best practices in code readability, etc.

Zokyo Security has followed best practices and industry-standard techniques to verify the implementation of FlowZed smart contract/s. To do so, the code was reviewed line by line by our smart contract developers, who documented even minor issues as they were discovered. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

01 Due diligence in assessing the overall code quality of the codebase.

02 Cross-comparison with other, similar smart contract/s by industry leaders.

03 Thorough manual review of the codebase line by line.

Executive Summary

FlowZed is utilizing blockchain technology to create waves in the fashion industry, enter The Merkle Tree collection. This is a collection of 5,050 1 of 1 physical silk jackets, which are fully backed by NFTs adhering to the ERC-721A standard. These contracts are deployed to the Ethereum mainnet, and access to purchase is only conditional if the user has a soulbound invite token. These invite tokens are known as ZedPasses which are handled by a separate ZedPass contract which complies with the ERC-1155 standard.

Zokyo was responsible for the security review for the flowzed1×0.sol and zedpass1×0.sol contracts over two days. Users can interact with the platform by first obtaining a zedpass which is either obtained via publicMint in the flowzed contract, or personally minted by an admin of the zedpass contract via the relative minting functionalities. The user can then redeem and order their physical silk jacket by navigating to the web3 gateway, which will trigger the contract owner's wallet to call print() in the flowzed contract. Alternatively you can send a zedpass nomination to an alternative address (the purpose behind ERC-1155 ID zero token), which will become a zedpass (upgraded to an ERC-1155 ID one token via safeTransfer method).

Overall, the code is well written and well engineered in order to achieve the goals of Flowzed. However, it's recommended that additional natspec is introduced as in line documentation so that function intention is abundantly clear. In addition to this whilst unit tests are present, we also recommend additional testing for the contracts, including the use of negative/positive testing and unit testing geared towards the security, such as asserting reentrancies do in fact fail as an example. The issues discovered during the security review ranged between Medium and Low in severity. This is mostly due to the stringent access controls laid out by the code and the heavily reliant on trusted dependencies such as the libraries used from the OpenZeppelin contracts, which is always advised to enhance security. Nonetheless, the issues discovered were mostly around bypassing, usage of outdated solidity conventions resulting in impact in addition to unintended reverts and uninitialized variables. We advise the development team to carefully review these findings and implement the suggested fixes for which a fix review will take place.

STRUCTURE AND ORGANIZATION OF THE DOCUMENT

For the ease of navigation, the following sections are arranged from the most to the least critical ones. Issues are tagged as “Resolved” or “Unresolved” or “Acknowledged” depending on whether they have been fixed or addressed. Acknowledged means that the issue was sent to the FlowZed team and the FlowZed team is aware of it, but they have chosen to not solve it. The issues that are tagged as “Verified” contain unclear or suspicious functionality that either needs explanation from the Client or remains disregarded by the Client. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise 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

The issue affects the ability of the contract to compile or operate in a significant way.

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 has no impact on the contract’s ability to operate.

COMPLETE ANALYSIS

FINDINGS SUMMARY

#	Title	Risk	Status
1	Outdated Transfer Method in FZ1×0.withdraw() May Lead To Out Of Gas Errors	Medium	Resolved
2	publicMint and invalidateCurrentEdition in FZ1×0 Do Not Refund Excess Tokens Resulting In A Loss Of Funds	Medium	Resolved
3	Mints Per Tx Can Be Bypassed Using A Multicall	Medium	Resolved
4	colourOf Not Updated When Current Edition Is Invalidated	Low	Acknowledged
5	Else Block Would Always Revert In safeTransferFrom	Low	Resolved
6	publicMint in FZ1×0 Will Revert If zedPass Is Not Set	Low	Resolved
7	The Admin Wallet Should Be Different From The Printer's Wallet As A Full Contract Takeover Could Occur If The Admins Wallet Is Compromised Via Frontend	Low	Resolved

Outdated Transfer Method in FZ1×0.withdraw() May Lead To Out Of Gas Errors

Description:
The flowzed contract allows for users to public mint tokens and use those tokens in order to print jackets. The `withdraw` function allows the contract administrators to collect the proceeds of such sales however, this method uses an outdated transfer function which may have gas implications especially integrating with other contracts

Impact:
The `transfer` method when transferring the token balance to the fzWallet hardcodes 2,300 gas units. As a result, should the gas amount fluctuate, this may deny the fzWallet user from extracting the proceeds from sales to their wallet through an out of gas error.

Recommendation:
It's recommended that a low level `call()` is used to extract native tokens where the success output variable is validated to ensure that the transfer was successful. This can be done by modifying the `withdraw` function to the following:

```
    function withdraw() external onlyOwner {
        (bool success,) = address(_fzWallet).call{value:
address(this).balance}("");
        require(success, "Failure to withdraw from the FZ1x0 contract");
```

publicMint and invalidateCurrentEdition in FZ1×0 Do Not Refund Excess Tokens Resulting In A Loss Of Funds

Description:
The `invalidateCurrentEdition` and `publicMint` functions are both payable functions accepting Ether in order to pay for the new edition or pay for tokens themselves respectively however, there is no functionality which will refund user funds should they accidentally overpay.

Impact:
Users who accidentally overpay when calling the aforementioned functions will experience the immediate loss of funds.

Recommendation:
In the case of publicMint, it is recommended that `msg.value` is stored in memory, then subtract the _price * quantity after asserting that there is sufficient funds. As for invalidateCurrentEdition, initialise a new variable which contains `msg.value` then subtract the _newEditionPrice. In both situations, the excess is then returned to the user.

Mints Per Tx Can Be Bypassed Using A Multicall

Description:
The publicMint() function has a _maxMintPerTx restriction which is currently set to `10` , this means within a single call to publicMint() a user should not be able to mint more than `10` tokens , but this can be bypassed if the user calls publicMint() within a multicall which can call publicMint() any number of times within the same multicall transaction.

Impact:
The mint per transaction limit can be bypassed by batching multiple publicMint() transactions in a single multicall.

Recommendation:
To have a strict restriction on mints per tx it's better to only allow minting of tokens once in a block, this can be done by storing the block number in publicMint() in a per user mapping and checking if the precious mint's block number is the same as the current.

colourOf Not Updated When Current Edition Is Invalidated

Description:
When current edition is invalidated using invalidateCurrentEdition() , the state of the tokenId is reset and the edition is incremented , but _colourOf[tokenId] is not being reset to Default , meaning it would still be the colour that was assigned during print operation.
Impact:
_colourOf[] would still store the colour of the invalidated edition while it should be set to Default.

Recommendation:
Reset the _colourOf[] to Default in invalidateCurrentEdition()

Client comment: This is by design - colourOf affects the displayed image in the NFT. If the holder needs a new edition issued, e.g. because of damage, we don't feel their chosen colour customisation should be reset on the on-chain image in the meantime before they print again.

We're also happy that the customer can later change colour on re-print. Better to have the potential in the contract to give the customer what they might want

LOW-2 | RESOLVED

Else Block Would Always Revert In safeTransferFrom

Description:
In the safeTransferFrom implementation of the zedpass token if the tokenId passed is 1 then it reaches the else block →

```
else {
            super.safeTransferFrom(from, to, id, amount, data);
        }
```

But this would always revert since in the _update function implementation if the transfer is a normal transfer and the tokenId is more than `0` i.e. 1 then it should revert.

Impact:
The else branch in the safeTransferFrom function would always revert due to the condition in the _update function.

Recommendation:
The else block in safeTransferFrom Can be removed.

LOW-3 | RESOLVED

publicMint in FZ1×0 Will Revert If zedPass Is Not Set

Description:
The Zedpass has `mint0` called when a user calls public mint; however, there exists a scenario where the contract could revert should it not be initialized.

Impact:
The contract will be denied sales if the Zedpass is not initialized.

Recommendation:
It's recommended that the Zedpass contract is passed in as constructor parameters so that the variable is initialized on deployment.

The Admin Wallet Should Be Different From The Printer's Wallet As A Full Contract Takeover Could Occur If The Admins Wallet Is Compromised Via Frontend

Description:
Users leverage the print function in order to place their order and receive their physical jacket. Currently this utilizes the modifier onlyAdmin where the owner or the admin can call this function and place the order for the user however, additional separation of powers may be required as this is a function which users are interacting with via the web interface.

Impact:
Should the wallet used to call the aforementioned function be compromised through the web interface when attempting to place an order, the entire contract may be compromised.

Recommendation:
It's recommended that the protocol developers create a separate address called the "printers" address. The only permission this address has is to call `print` for users and should be used when users place orders for their physical jacket - even if this wallet is compromised via the frontend, this will not result in a full contract takeover.

	./zedpass1x0.sol ./flowzed1x0.sol
Re-entrancy	Pass
Access Management Hierarchy	Pass
Arithmetic Over/Under Flows	Pass
Unexpected Ether	Pass
Delegatecall	Pass
Default Public Visibility	Pass
Hidden Malicious Code	Pass
Entropy Illusion (Lack of Randomness)	Pass
External Contract Referencing	Pass
Short Address/ Parameter Attack	Pass
Unchecked CALL Return Values	Pass
Race Conditions / Front Running	Pass
General Denial Of Service (DOS)	Pass
Uninitialized Storage Pointers	Pass
Floating Points and Precision	Pass
Tx.Origin Authentication	Pass
Signatures Replay	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass

We are grateful for the opportunity to work with the FlowZed team.

The statements made in this document should not be interpreted as an investment or legal advice, nor should its authors be held accountable for the decisions made based on them.

Zokyo Security recommends the FlowZed team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

zokyo