SMART CONTRACTS REVIEW

zokyo

April 19th 2024 | v. 1.0

Security Audit Score

PASS

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

ZOKYO AUDIT SCORING WAVEFRONT

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 issuei: 0 points deducted
- 2 Medium issues: 2 acknowledged = - 6 points deducted
- 3 Low issues: 1 acknowledged and 2 resolved = - 2 points deducted
- 3 Informational issue: 3 resolved = 0 points deducted

Thus, 100 - 6 - 2 = 92

TECHNICAL SUMMARY

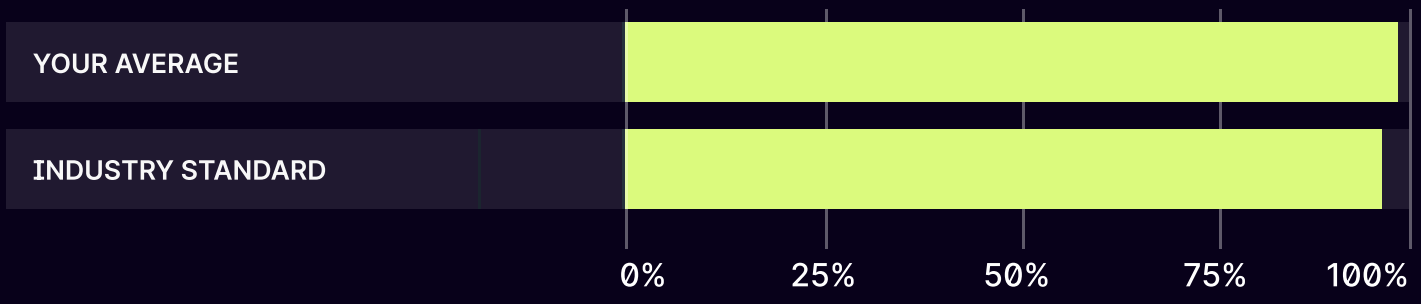
This document outlines the overall security of the Wavefront smart contract/s evaluated by the Zokyo Security team.

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

Contract Status

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

Testable Code



95.92% of the code is testable, which is above the industry standard of 95%.

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 Wavefront 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 5

Executive Summary 7

Structure and Organization of the Document 8

Complete Analysis 9

Code Coverage and Test Results for all files written by Zokyo Security 15

AUDITING STRATEGY AND TECHNIQUES APPLIED

The source code of the smart contract was taken from the Wavefront repository:
Repo: https://github.com/Heesho/wavefront/blob/main/contracts

Last commit: 0d9c02bba961e5e5b1fc766c8239fe86007681dc

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

- ./MemeFactory.sol
- ./WaveFrontFactory.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 Wavefront 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. Part of this work includes writing a test suite using the Hardhat testing framework. 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	Testing contract/s logic against common and uncommon attack vectors.
04	Thorough manual review of the codebase line by line.

Executive Summary

The Zokyo team has performed a security audit of the provided codebase. The contracts submitted for auditing are well-crafted and organized. Detailed findings from the audit process are outlined in the "Complete Analysis" section.

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 Wavefront team and the Wavefront 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	Calculations Not Compatible With USDC/USDT	Medium	Acknowledged
2	Incorrect Accounting For Fee-On-Transfer Tokens	Medium	Acknowledged
3	createMeme Can be Called From The MemeFactory Directly Bypassing All Checks	Low	Resolved
4	contribute() function shouldn't work after endTimestamp	Low	Acknowledged
5	Unsafe transfer	Low	Resolved
6	Storage variables are read multiple times in the functions	Informational	Resolved
7	Incorrect Comments	Informational	Resolved
8	memeFactory variable could be setas immutable	Informational	Resolved

MEDIUM-1 | ACKNOWLEDGED

Calculations Not Compatible With USDC/USDT

The base token is supposed to be a stable coin in the MemeFactory contract , and it is safe to assume that these tokens can be USDC or USDT , i.e. tokens with 6 decimals but the calculation assumes the base token to be of 18 decimals.
Inside the buy() function we calculate how much meme tokens to mint for a specified amount of base tokens . The new reserveBase is calculated with a precision of 1e6 and based on that the newReserveMeme is calculated where reserveMeme and reserveBase is taken into account and reserveMeme is in 1e18 whereas reserveBase is in 1e6 leading to a an incorrect amountOut.

Recommendation:
Normalise the decimals of the base tokens to 18 decimals prior to calculating reserves and amount.

Client comment: wETH will be used as a base token.

MEDIUM-2 | ACKNOWLEDGED

Incorrect Accounting For Fee-On-Transfer Tokens

The base token is supposed to be a stable coin like USDT , though not currently but it is possible in the future that the fee on transfer mechanism for the token is turned on . If so , the accounting would be incorrect in the contribute() function . This is because totalContributed and account_BaseContributed would be lower than the actual amount sent to the contract since a part of the amount would be sent out as fee.

Recommendation:
Check the balance of the base token before and after the transfer and account for the difference.

Client comment: wETH will be used as a base token.

LOW-1 | RESOLVED

createMeme Can be Called From The MemeFactory Directly Bypassing All Checks

A new meme is supposed to be deployed via the createMeme() function in the WaveFrontFactory.sol which checks the name , symbol and minAmountIn before deploying a meme , but these checks can be bypassed if the meme is directly deployed via calling the createMeme function in the MemeFactory which has no checks.

Recommendation:
Make sure only the WaveFrontFactory can call the createMeme function.

LOW-2 | RESOLVED

contribute() function shouldn't work after endTimestamp

According to the comment of the contribute() function, it is intended to allow users to contribute during the pre-market phase.
But it allows users to call after **endTimstamp** if the market is not opened.

Recommendation:
Add a check if block.timestamp has passed endTimestamp.

Client comment: It's intended behavior.

LOW-3 | RESOLVED

Unsafe transfer

MemeFactory.sol L108, L137, L166, L314, L321, L323, L359, L376, L391, L463, L487
The **transfer()** and **transferFrom()** functions return a bool in the IERC20 implementation but in the above lines, the return values are not checked. If the token to be transferred is a contract which doesn't return a bool on its transfer and transferFrom functions, the transaction will be failed.

Recommendation:
Use safeTransfer and safeTransferFrom instead.

INFORMATIONAL-1 | RESOLVED

Storage variables are read multiple times in the functions

In the following lines, the storage variables are read multiple times in a single function. Reading storage variables multiple times in a single function costs more gas than defining a local variable.

MemeFactory.sol L303 - L304 : **reserveBase** variable
MemeFactory.sol L304 - L305 : **reserveMeme** variable
MemeFactory.sol L347 - L348 : **reserveBase** variable
MemeFactory.sol L346 - L347 : **reserveMeme** variable
MemeFactory.sol L442 - L443 : **maxSupply**, **reserveMeme** variables
MemeFactory.sol L599, L614 : **maxSupply** variables
MemeFactory.sol L648 - L649 : **lastMeme** variable
WaveFrontFactory.sol L112 - L116 : **index** variable

MemeFactory.sol L323 - L324 : Calling the same read function (**treasury**()) of an external contract twice.

Recommendation:
Add a local variable instead of reading a storage variable multiple times.

INFORMATIONAL-2 | RESOLVED

Incorrect Comments

The code mentions that base token can be ETH but there is no support for native.

INFORMATIONAL-3 | RESOLVED

memeFactory variable could be setas immutable

In the WaveFrontFactory contract, the memeFactory variable is only set in the constructor.

Recommendation:
Set memeFactory variable immutable.

	./MemeFactory.sol ./WaveFrontFactory.sol
Reentrance	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

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by Zokyo Security

As a part of our work assisting Wavefront in verifying the correctness of their contract/s code, our team was responsible for writing integration tests using the Hardhat testing framework.

The tests were based on the functionality of the code, as well as a review of the Wavefront contract/s requirements for details about issuance amounts and how the system handles these.

MemeFactory and WaveFrontFactory Test Suite

Contribution Phase

✓ Should allow contributing base tokens before the pre-market phase ends
✓ Should revert when contributing zero amount

MemeFactory Contract

✓ Should deploy MemeFactory
✓ Should create a new Meme

WaveFrontFactory Contract

✓ Should deploy WaveFrontFactory
✓ Should create a new Meme through WaveFrontFactory (95ms)
✓ Should set a new treasury address
✓ Should update the minimum amount in for creating a meme
✓ Should revert when creating a Meme with insufficient amountIn
✓ Should revert when creating a Meme with an existing symbol
✓ Should revert when creating a Meme with invalid name length
✓ Should revert when setting treasury to zero address
✓ Should revert when a non-owner tries to update minAmountIn
✓ Should revert when creating a Meme with an empty name
✓ Should revert when creating a Meme with an empty symbol
✓ Should revert when creating a Meme with a symbol length exceeding the limit
✓ Should revert when a non-owner tries to set the treasury address
✓ Should successfully create a Meme with sufficient amountIn (79ms)
✓ Should allow the owner to increase minAmountIn

router tests

✓ User0 creates meme1 and meme2, and User0 and User1 contribute and redeem contributions (553ms)
✓ Buying, selling, claiming fees, and status updates for meme1 (370ms)
✓ Additional tests for selling and error handling (105ms)

Quoting and Trading

✓ should quote buy and sell prices accurately (79ms)

Meme and Account Data
✓ should retrieve meme and account data correctly (152ms)
Borrowing, Repaying, and Transferring
✓ should handle borrowing, repaying, and transferring correctly (79ms)

Meme Contract

updateStatus
✓ Should revert on zero address
✓ Should revert on empty status
✓ Should revert on status exceeding max length
✓ Should successfully update status and emit an event (79ms)
donate
✓ User0 donates 1 WETH
✓ Should revert on zero donation amount
getAccountCredit
✓ Should return 0 if account balance is 0
getAccountTransferrable
✓ Should return account balance if account_Debt is 0

33 passing (3s)

The resulting code coverage (i.e., the ratio of tests-to-code) is as follows:

FILE	% STMTS	% BRANCH	% FUNCS	% LINES	%UNCOVERED LINES
Base.sol	100%	50%	100%	100%	
FixedPointMathLib.sol	100%	100%	54.55%	54.55%	... 231,240,249
MemeFactory.sol	100%	66.28%	100%	99.29%	448
WaveFrontFactory.sol	100%	100%	100%	100%	
WaveFrontMulticall.sol	76.19%	66.28%	58.33%	88.73%	... 211,212,213
WaveFrontRouter.sol	94.12%	80%	77.78%	94.44%	148,149,150
All Files	**95.92%**	**72.41%**	**82.35%**	**94.44%**	

We are grateful for the opportunity to work with the Wavefront 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 Wavefront team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

zokyo