SMART CONTRACTS REVIEW

December 4th 2024 | v. 1.0

Security Audit Score

PASS

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

ZOKYO AUDIT SCORING WELF

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: 0 points

Starting with a perfect score of 100:
- 2 Critical issues: 2 resolved = 0 points deducted
- 0 High issue: 0 points deducted
- 1 Medium issue: 1 resolved = 0 points deducted
- 3 Low issues: 3 resolved = 0 points deducted
- 4 Informational issues: 3 resolved and 1 acknowledged = 0 points deducted

Thus, the score is 100

TECHNICAL SUMMARY

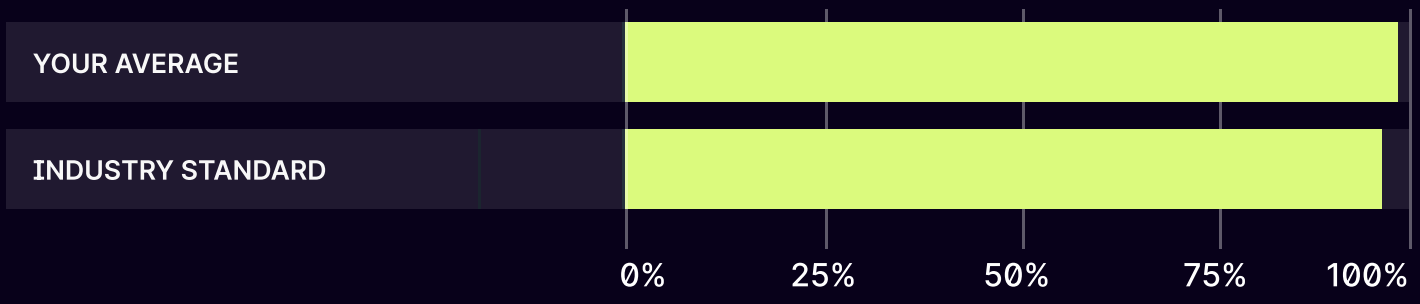
This document outlines the overall security of the WELF smart contract/s evaluated by the Zokyo Security team.

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

Contract Status

There were 2 critical issues found during the review. (See Complete Analysis)

Testable Code



95,67% 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 WELF 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 WELF repository:
Repo: https://github.com/Welf-global/welf-smart-contracts
Last commit - 17d9a1675638e43e0d1b3469e559e840271c77d8

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

- ./platform/token/ERC20FixedSupply.sol
- ./platform/token/IplatformToken/IERC20FixedSupply.sol
- ./platform/access_controller/PlatformAccessController.sol
- ./platform/admin_panel/PlatformAdminPanel.sol
- ./platform/admin_panel/Iplatform_admin_panel/IPlatformAdminPanel.sol
- ./platform/vesting/ERC20Vesting.sol
- ./platform/vesting/IPlatformVesting/IPlatformVesting.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 WELF 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 Foundry 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

Vesting system for managing token distributions, specifically for an ERC20 token ($WELF), where tokens are gradually released to users over time. It is controlled by a platform admin and ensures that funds are unlocked in stages according to a set schedule. The contract stores multiple "vesting pools," each with its own parameters such as total token amount, start time, and how many stages (ticks) the vesting period has. It allows the admin to add users to specific pools, and manage liquidity (tokens allocated for vesting).

Users are able to claim a portion of their allocated tokens based on the vesting schedule, with some funds available immediately after the **Token Generation Event (TGE)** and the rest unlocked incrementally over time. The contract tracks each user's claimed amount and ensures that the correct amount is transferred when they claim tokens. The admin has the ability to adjust the vesting conditions, remove users, and modify liquidity.

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 WELF team and the WELF 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	Missing _vestingCount Increment	Critical	Resolved
2	Incorrect Vesting Amount Comparison	Critical	Resolved
3	TGE Date Update Affects Previous Vestings	Medium	Resolved
4	Excessive Loops Around the Codebase	Low	Resolved
5	Incorrect Comparison for Offset Validation	Low	Resolved
6	Unused ReentrancyGuard Import	Low	Resolved
7	Unnecessary Use of SafeMath	Informational	Resolved
8	Unused event	Informational	Resolved
9	Could use delete keyword	Informational	Resolved
10	Risk of Centralisation	Informational	Acknowledged

CRITICAL-1 | RESOLVED

Missing `_vestingCount` Increment

The `_vestingCount` variable is not incremented after being used to assign an index in the `addUserVesting` function of ERC20Vesting. This results in vestings being assigned the same index, leading to inconsistent and corrupted mappings.
Additionally, `addUserVesting` function assigns the `index` for the new vesting to `_vestingCount` without incrementing it. This could lead to overwriting the most recently added vesting entry.

Recommendation:

Increment `_vestingCount` and set `index` to `_vestingCount + 1` to ensure a unique index is used for the new vesting.

CRITICAL-2 | RESOLVED

Incorrect Vesting Amount Comparison

In the `updateUserVesting` function of ERC20Vesting, the condition `if(user.claimedAmount < newAmount)` is incorrect. This condition erroneously checks if the `claimedAmount` is less than `newAmount`. The intended logic should verify that the `claimedAmount` does not exceed the `newAmount`. Using the wrong comparison allows reducing the vesting amount below what has already been claimed, potentially resulting in an underflow condition when calculating the claim amount and it will always revert.

Recommendation:

Update the condition to `if(user.claimedAmount > newAmount) revert()` to correctly enforce that the new vesting amount must be greater than or equal to the already claimed amount.

MEDIUM-1 | RESOLVED

TGE Date Update Affects Previous Vestings

The `setTgeDate` function of ERC20Vesting updates the `tgeStartDate` each time it is called. However, this update applies to all vesting schedules, including previous ones, potentially causing confusion or lack of transparency for users.

Recommendation:

Cache the `tgeStartDate` for each vesting schedule rather than updating the global `tgeStartDate` each time. This will ensure transparency for users and provide a clear reference to the TGE date used at the time of their vesting. Consider storing the TGE date in the vesting mapping (`_vestingMap`) for each user.

LOW-1 | RESOLVED

Excessive Loops Around the Codebase

There are multiple instances of loops in the codebase like insertVestingList, insertWalletListToVesting, removeWalletListFromVesting, insertAdminList, removeAdminList that iterate over potentially large datasets, E.g. insertVestingList, insertWalletListToVesting, removeWalletListFromVesting, insertAdminList, removeAdminList. Excessive use of loops can lead to high gas costs and inefficient execution, especially when dealing with large amounts of data. This can result in performance issues and may even cause transactions to fail due to gas limit exceedance.

Recommendation:

Optimize loops by reducing their frequency and considering alternative methods.

LOW-2 | RESOLVED

Incorrect Comparison for Offset Validation

In the `airdrop` function of ERC20Vesting, the check `if(offset > _vestingCount)` is used to ensure the offset is within bounds. However, it should be `if(offset >= _vestingCount)` because if the offset is equal to `_vestingCount`, the index will be zero.

```
492:    function airdrop(uint256 batchSize, uint256 offset) external onlyPlatformAdmin {
493:        if(offset > _vestingCount)
494:            revert OutOfBounds();
495:
496:        uint256 index = _vestingCount - offset;
```

Recommendation:

Update the comparison to `if(offset >= _vestingCount)` to ensure the correct behavior when the offset is equal to `_vestingCount`.

LOW-3 | RESOLVED

Unused ReentrancyGuard Import

The ERC20Vesting.sol and ERC20FixedSupply.sol contract imports ReentrancyGuard from OpenZeppelin, but it is not utilized anywhere in the contract. This could indicate unnecessary code and an unused import, potentially increasing the contract's size without providing any security benefits.

Recommendation:

Apply the `nonReentrant` modifier to the relevant functions.

INFORMATIONAL-1 | RESOLVED

Unnecessary Use of SafeMath

The ERC20Vesting contract imports `SafeMath` from the OpenZeppelin library, but in Solidity 0.8.0 and later, arithmetic operations are checked for overflow and underflow by default. Using `SafeMath` is redundant and not necessary, as the compiler automatically handles these checks.

Recommendation:
Remove the `SafeMath` import and refactor the contract to remove any reliance on it, as Solidity 0.8+ provides built-in overflow and underflow checks.

INFORMATIONAL-2 | RESOLVED

Unused event

The ERC20Vesting.sol contract declares the IncreaseLiquidity event but the vent is not used in the contract.

Recommendation:
Remove the unused event.

INFORMATIONAL-3 | RESOLVED

Could use delete keyword

The removeWalletListFromVesting() function of ERC20Vesting is intended to remove the user's vestings by setting each element of _vestingToUser[vestingId] to the default value. However, it could use the “delete” keyword to remove the entire struct storage instead of setting each element to the default value.

Recommendation:
Consider using the “delete” keyword like “delete _vestingToUser[vestingId]”.

Risk of Centralisation

The current implementation grants significant control to the owner through multiple functions that can alter the contract's state and behavior. This centralization places considerable trust in a single entity, increasing the risk of potential misuse.
I
f the owner's private key is compromised, an attacker could execute any function accessible to the owner, potentially leading to fund loss, contract manipulation, or service disruption.

Recommendation:
To enhance security and reduce the risk of a single point of failure, it is recommended to implement a multi-signature wallet for executing owner functions.

Client Comment: As with propchain it is planned to use Fireblocks SaaS solution to distribute key-shards between multiple management and operational members. Their approval is mandatory and configured in the company’s workspace. Transactions can therefore only be signed after all necessary team members have reviewed and approved a transaction via their defined company devices.

./platform/token/ERC20FixedSupply.sol
./platform/token/IplatformToken/IERC20FixedSupply.sol
./platform/access_controller/PlatformAccessController.sol
./platform/admin_panel/PlatformAdminPanel.sol
./platform/admin_panel/Iplatform_admin_panel/IPlatformAdminPanel.sol
./platform/vesting/ERC20Vesting.sol
./platform/vesting/IPlatformVesting/IPlatformVesting.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

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by Zokyo Security

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

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

Ran 4 tests for test/PlatformAccessControllerTest.t.sol:PlatformAccessControllerTest
[PASS] testInitiatePlatformAccessController() (gas: 13406)
[PASS] testInitiatePlatformAccessControllerAlreadyInitialized() (gas: 11138)
[PASS] testIsAdmin() (gas: 25287)
[PASS] testOnlyPlatformAdminModifier() (gas: 23900)
Suite result: ok. 4 passed; 0 failed; 0 skipped; finished in 16.38ms (755.48µs CPU time)

Ran 9 tests for test/ERC20FixedSupply.t.sol:ERC20FixedSupplyTest
[PASS] testBeforeTokenTransferWithAntisnipe() (gas: 294227)
[PASS] testConstructor() (gas: 22393)
[PASS] testConstructorInvalidAddressRevert() (gas: 214377)
[PASS] testConstructorZeroAmountRevert() (gas: 111410)
[PASS] testSetAntisnipeAddress() (gas: 42701)
[PASS] testSetAntisnipeAddressInvalidAddressRevert() (gas: 17206)
[PASS] testSetAntisnipeDisable() (gas: 41597)
[PASS] testSetAntisnipeDisableAlreadyDisabledRevert() (gas: 42903)
[PASS] testTotalSupply() (gas: 11052)
Suite result: ok. 9 passed; 0 failed; 0 skipped; finished in 151.56ms (92.84ms CPU time)

Ran 5 tests for test/PlatformAccessControllerTestFuzz.t.sol:PlatformAccessControllerFuzzTestFuzz
[PASS] testFuzzInitiatePlatformAccessController(address) (runs: 268, µ: 316316, ~: 316316)
[PASS] testFuzzInitiatePlatformAccessControllerAlreadyInitialized(address) (runs: 268, µ: 11384, ~: 11384)
[PASS] testFuzzIsAdmin(address) (runs: 268, µ: 24163, ~: 24163)
[PASS] testFuzzMsgSender(address) (runs: 268, µ: 9877, ~: 9877)
[PASS] testFuzzOnlyPlatformAdminModifier(address) (runs: 268, µ: 23677, ~: 23677)
Suite result: ok. 5 passed; 0 failed; 0 skipped; finished in 620.43ms (609.99ms CPU time)

Ran 8 tests for test/PlatformAdminPanel.t.sol:PlatformAdminPanelTest
[PASS] testConstructor() (gas: 13351)
[PASS] testInsertAdminList() (gas: 70126)
[PASS] testInsertAdminListEmptyList() (gas: 11758)
[PASS] testOnlyRootAdminModifierFails() (gas: 14784)
[PASS] testRemoveAdminList() (gas: 57660)
[PASS] testRemoveAdminListEmptyList() (gas: 11780)
[PASS] testSetRootAdmin() (gas: 19147)
[PASS] testSetRootAdminZeroAddress() (gas: 11518)
Suite result: ok. 8 passed; 0 failed; 0 skipped; finished in 9.58ms (1.71ms CPU time)

Ran 3 tests for test/PlatformAdminPanelTestFuzz.t.sol:PlatformAdminPanelTestFuzz
[PASS] testFuzzInsertAdminList(address[]) (runs: 268, μ: 3442787, ~: 3422824)
[PASS] testFuzzIsAdmin(address) (runs: 268, μ: 29192, ~: 16960)
[PASS] testFuzzSetRootAdmin(address) (runs: 268, μ: 17635, ~: 17635)
Suite result: ok. 3 passed; 0 failed; 0 skipped; finished in 3.87s (3.86s CPU time)

Ran 39 tests for test/ERC20Vesting.t.sol:ERC20VestingTest
[PASS] testAddUserVesting() (gas: 459729)
[PASS] testAddUserVestingAlreadyActive1() (gas: 250362)
[PASS] testAddUserVestingAlreadyActive2() (gas: 314798)
[PASS] testAddUserVestingInvalidCreationData() (gas: 61597)
[PASS] testAddUserVestingZeroAddress() (gas: 18616)
[PASS] testAirdropOutOfBounds() (gas: 213097)
[PASS] testAmountForClaim() (gas: 560207)
[PASS] testClaimNoClaimableAmount() (gas: 419325)
[PASS] testConstructorValidAddress() (gas: 3251100)
[PASS] testConstructorZeroAddressRevert() (gas: 62771)
[PASS] testDecreaseLiquidityInsufficientBalance() (gas: 1250647)
[PASS] testDecreaseLiquidityZeroAmount() (gas: 1248825)
[PASS] testDistributeAmountOutOfBounds() (gas: 213096)
[PASS] testInsertVestingList() (gas: 209629)
[PASS] testInsertVestingListEmptyArray() (gas: 19685)
[PASS] testInsertVestingListValidInput() (gas: 209563)
[PASS] testInsertWalletListToVesting() (gas: 295497)
[PASS] testInsertWalletListToVestingAlreadyActive() (gas: 664808)
[PASS] testInsertWalletListToVestingArraySizeMismatch() (gas: 378382)
[PASS] testInsertWalletListToVestingEmptyArray() (gas: 212639)
[PASS] testInsertWalletListToVestingInvalidVestingId() (gas: 234883)
[PASS] testInsertWalletListToVestingUserAlreadyActive() (gas: 462887)
[PASS] testRemoveActiveVesting() (gas: 180285)
[PASS] testRemoveInactiveVesting() (gas: 194130)

[PASS] testRemoveVestingExceedsClaimedAmount() (gas: 727239)
[PASS] testRemoveVestingOutOfBounds() (gas: 20428)
[PASS] testRemoveVestingValidId() (gas: 180302)
[PASS] testRemoveWalletListFromVestingEmptyArray() (gas: 17599)
[PASS] testRemoveWalletListFromVestingUserNotActive() (gas: 465111)
[PASS] testSetTgeDate() (gas: 42333)
[PASS] testSetTgeDateEmitEvent() (gas: 42874)
[PASS] testSetTgeDateInvalidTimestamp() (gas: 18046)
[PASS] testSetTgeDateValidTimestamp() (gas: 42376)
[PASS] testSetVestingTicksMissing() (gas: 251461)
[PASS] testSetVestingValidations() (gas: 260220)
[PASS] testUpdateTokenAddress() (gas: 27023)
[PASS] testUpdateUserVestingNotActive() (gas: 262792)
[PASS] testVestingByAddressActiveUser() (gas: 463007)
[PASS] testVestingByAddressInactiveUser() (gas: 24332)
Suite result: ok. 39 passed; 0 failed; 0 skipped; finished in 284.58ms (192.28ms CPU time)

Ran 4 tests for test/ERC20FixedSupplyTestFuzz.t.sol:ERC20FixedSupplyTestFuzz
[PASS] testFuzzApproveAndTransferFrom(address,address,uint256) (runs: 268, μ: 73117, ~: 73099)
[PASS] testFuzzConstructor(uint256) (runs: 268, μ: 1538131, ~: 1538131)
[PASS] testFuzzSetAntisnipeAddress(address) (runs: 268, μ: 43515, ~: 43515)
[PASS] testFuzzTransfer(address,uint256) (runs: 268, μ: 57393, ~: 56908)
Suite result: ok. 4 passed; 0 failed; 0 skipped; finished in 12.97s (12.91s CPU time)

Ran 19 tests for test/ERC20VestingFuzz.t.sol:ERC20VestingTestFuzz
[PASS] testFuzzAddUserVesting(address,uint256,uint256) (runs: 262, μ: 456043, ~: 456803)
[PASS] testFuzzAddUserVestingAlreadyActive(address,uint256,uint256,uint256,uint256,uint256) (runs: 262, μ: 251409, ~: 251941)
[PASS] testFuzzAddUserVestingAlreadyActive(uint256,uint256,uint256,uint256,uint256) (runs: 262, μ: 316528, ~: 317060)
[PASS] testFuzzAddUserVestingZeroAddress(uint256,uint256,uint256,uint256,uint256) (runs: 262, μ: 20009, ~: 20009)
[PASS] testFuzzAirdropOutOfBounds(uint256,uint256,uint256) (runs: 268, μ: 8464960, ~: 7973824)
[PASS] testFuzzAmountForClaim(uint256,uint256,uint256,uint256,uint256,uint256) (runs: 268, μ: 501436, ~: 503422)
[PASS] testFuzzClaimNoClaimableAmount(uint256,uint256,uint256,uint256) (runs: 268, μ: 432068, ~: 432739)
[PASS] testFuzzInsertVestingList(uint256,uint256,uint256,uint256,uint256) (runs: 262, μ: 210949, ~: 211481)
[PASS] testFuzzInsertWalletListToVestingAlreadyActive(address,address) (runs: 268, μ: 665868, ~: 665868)

[PASS] testFuzzInsertWalletListToVestingEmptyArray(uint256) (runs: 257, μ: 213327, ~: 213327)
[PASS] testFuzzInsertWalletListToVestingInvalidVestingId(uint256) (runs: 268, μ: 236527, ~: 236527)
[PASS] testFuzzInsertWalletListToVestingUserAlreadyActive(address) (runs: 268, μ: 463678, ~: 463678)
[PASS] testFuzzRemoveActiveVesting(uint256,uint256,uint256,uint256,uint256) (runs: 262, μ: 180770, ~: 181180)
[PASS] testFuzzRemoveInactiveVesting(uint256,uint256,uint256,uint256,uint256) (runs: 262, μ: 194990, ~: 195522)
[PASS] testFuzzRemoveVestingOutOfBounds(uint256) (runs: 268, μ: 21153, ~: 21153)
[PASS] testFuzzRemoveVestingValidId(uint256,uint256,uint256,uint256,uint256,uint256) (runs: 256, μ: 180718, ~: 181266)
[PASS] testFuzzRemoveWalletListFromVestingEmptyArray() (gas: 17576)
[PASS] testFuzzSetTgeDate(uint256) (runs: 268, μ: 42914, ~: 42914)
[PASS] testFuzzUpdateUserVestingNotActive(uint256,uint256,uint256,uint256,uint256,uint256) (runs: 259, μ: 265458, ~: 265843)

Suite result: ok. 19 passed; 0 failed; 0 skipped; finished in 9.14s (9.85s CPU time)

Ran 8 test suites in 13.95s (27.07s CPU time): 91 tests passed, 0 failed, 0 skipped (91 total tests)

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

FILE	% STMTS	% BRANCH	% FUNCS	% LINES
PlatformAccessController.sol	100	100	100	100
PlatformAdminPanel.sol	100	87.5	100	100
ERC20FixedSupply.sol	100	100	100	100
ERC20Vesting.sol	82.69	76.09	90.91	81.87
All Files	**95.67**	**90.98**	**97.72**	**95.46**

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

zokyo