Cyberscope

# Audit Report

## zkSwap Finance Governance Staking

January 2024

| | |
|---|---|
| Network | zkSync |
| Address | 0x4Ca2aC3513739ceBF053B66a1d59C88d925f1987 |
| Audited by | © cyberscope |

# Table of Contents

# Review

| | |
|---|---|
| **Explorer** | https://explorer.zksync.io/address/0x4ca2ac3513739cebf053b66a1d59c88d925f1987 |

## Audit Updates

| | |
|---|---|
| **Initial Audit** | 22 Jan 2024 |
| **Acknowledged Phase** | 24 Jan 2024 |

## Source Files

| Filename | SHA256 |
|---|---|
| **yZFToken.sol** | 0df67efc8b32d08a2dcc82a5633024d6d0cc8eefff2bc9844b89e6bc97b3d3a6 |
| **ZFGovernanceStaking.sol** | 5f4907032640ca430b234d89c78058f886dde6a03acd20daee1747995b430063 |
| **IZFToken.sol** | 589e755e2ff327f6fb0c3dd929bf8af3af7bd7d459d786682b24ac476200c709 |
| **@openzeppelin/contracts/utils/Strings.sol** | 0519199dbc635f98ce2e4537986604ee618bca665c65e9a1738702dfacf72010 |
| **@openzeppelin/contracts/utils/StorageSlot.sol** | b4a5fb7ab93bfeda06509eafbd5f71fde0e0de84b6d9129553bd535a42166c15 |
| **@openzeppelin/contracts/utils/ShortStrings.sol** | ddd52921d2996abf2e3d9c1c4f6d00194a3e3b278a164948f995862371444a55 |
| **@openzeppelin/contracts/utils/Nonces.sol** | 1c16c3cf8bb0679cbd47cddd8b141fea193e76966c94c858c5bccc94b8695030 |
| **@openzeppelin/contracts/utils/Context.sol** | 847fda5460fee70f56f4200f59b82ae622bb03c79c77e67af010e31b7e2cc5b6 |

| | |
|---|---|
| **@openzeppelin/contracts/utils/math/SignedMath.sol** | 768c28e3a33c3312e57ae8a1caaec2893bc89ac6e386621de018f85e9a2d6e99 |
| **@openzeppelin/contracts/utils/math/Math.sol** | a6ee779fc42e6bf01b5e6a963065706e882b016affbedfd8be19a71ea48e6e15 |
| **@openzeppelin/contracts/utils/cryptography/MessageHashUtils.sol** | 2fd5c641cf452efd15f784827cb2835664970d7fbc166bf80824ed27011cc374 |
| **@openzeppelin/contracts/utils/cryptography/EIP712.sol** | 27dac0732a0154f432c0a7a1d1f067ab51116105e157d0e5d68d040fd83954d5 |
| **@openzeppelin/contracts/utils/cryptography/ECDSA.sol** | 37828cb50b47bcc51c7b770bde15d5885d871ef1e67028057a0b788c3568726e |
| **@openzeppelin/contracts/token/ERC20/IERC20.sol** | 6f2faae462e286e24e091d7718575179644dc60e79936ef0c92e2d1ab3ca3cee |
| **@openzeppelin/contracts/token/ERC20/ERC20.sol** | ddff96777a834b51a08fec26c69bb6ca2d01d150a3142b3fdd8942e07921636a |
| **@openzeppelin/contracts/token/ERC20/extensions/IERC20Permit.sol** | 912509e0e9bf74e0f8a8c92d031b5b26d2d35c6d4abf3f56251be1ea9ca946bf |
| **@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol** | 1d079c20a192a135308e99fa5515c27acfbb071e6cdb0913b13634e630865939 |
| **@openzeppelin/contracts/token/ERC20/extensions/ERC20Permit.sol** | 677cb995a34f0cc937f3d77d4626c46fbf47cdef4c9cc0314c27672c0459cf80 |
| **@openzeppelin/contracts/token/ERC20/extensions/ERC20FlashMint.sol** | 58f4f4e5b759b5709a7ba705dfe60a26a60fb18154bff8cdf145a7c4e8c4c368 |
| **@openzeppelin/contracts/token/ERC20/extensions/ERC20Burnable.sol** | 2e6108a11184dd0caab3f3ef31bd15fed1bc7e4c781a55bc867ccedd8474565c |
| **@openzeppelin/contracts/interfaces/draft-IERC6093.sol** | 4aea87243e6de38804bf8737bf86f750443d3b5e63dd0fd0b7ad92f77cdbd3e3 |

| | |
|---|---|
| **@openzeppelin/contracts/interfaces/IERC5267.sol** | efd1ebd1e04b6ef9c3b8781a097588f83da954323f438d54a71dc06508e6c7b8 |
| **@openzeppelin/contracts/interfaces/IERC3156FlashLender.sol** | 3fb668ca6aaf756f5db9049abd2a18f638ff70307ca7ce59f85e772bae17380d |
| **@openzeppelin/contracts/interfaces/IERC3156FlashBorrower.sol** | 06a759fc3607f87bfb716c95ac2f67c64b1485703bdd9467f1fea7ecc1180215 |
| **@openzeppelin/contracts/access/Ownable.sol** | 38578bd71c0a909840e67202db527cc6b4e6b437e0f39f0c909da32c1e30cb81 |

# Overview

This document presents the overview of the smart contract audit conducted for the "zkSwap Finance Governance Staking" contract. The purpose of this audit is to identify and address security vulnerabilities, provide recommendations for code improvements, and ensure the robustness of the codebase.

## ZFGovernanceStaking Contract

The "ZFGovernanceStaking" contract is a decentralized finance (DeFi) smart contract designed for staking. Key features of this contract include:

**Token Staking**

Allows users to stake "ZFToken" tokens and earn rewards over time.

**Reward Calculation**

Implements a mechanism to calculate rewards based on the amount of time tokens are staked, using `zfPerSecond` rate and total Supply.

**Flexible Time Frame**

Utilizes `startTimestamp` and `endTimestamp` to define the active period for reward accumulation.

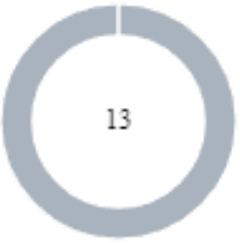
**Withdrawal Fees**:

Incorporates a withdrawal fee mechanism, controlled by `withdrawFeeFactor`, allowing the contract owner to set withdrawal fees.

**Owner Controls**

Provides the contract owner with various administrative functions, including setting reward rates, withdrawal fees, and modifying the staking period's start and end times.

# Findings Breakdown




| | |
|---|---|
| Critical | 0 |
| Medium | 0 |
| Minor / Informative | 13 |

| Severity | Unresolved | Acknowledged | Resolved | Other |
|---|---|---|---|---|
| Critical | 0 | 0 | 0 | 0 |
| Medium | 0 | 0 | 0 | 0 |
| Minor / Informative | 11 | 2 | 0 | 0 |

# Diagnostics

Critical Medium Minor / Informative

| Severity | Code | Description | Status |
|---|---|---|---|
| ● | CCR | Contract Centralization Risk | Acknowledged |
| ● | IDI | Immutable Declaration Improvement | Unresolved |
| ● | ITSV | Inadequate Time Setting Validation | Unresolved |
| ● | LTA | Locked Tokens Accumulation | Acknowledged |
| ● | MVN | Misleading Variable Name | Unresolved |
| ● | MEE | Missing Events Emission | Unresolved |
| ● | RES | Redundant Event Statement | Unresolved |
| ● | RSML | Redundant SafeMath Library | Unresolved |
| ● | RSW | Redundant Storage Writes | Unresolved |
| ● | L04 | Conformance to Solidity Naming Conventions | Unresolved |
| ● | L07 | Missing Events Arithmetic | Unresolved |
| ● | L16 | Validate Variable Setters | Unresolved |
| ● | L19 | Stable Compiler Version | Unresolved |

# CCR - Contract Centralization Risk

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | ZFGovernanceStaking.sol#L112,117,121,126 |
| **Status** | Acknowledged |

## Description

The contract's functionality and behavior are heavily dependent on external parameters or configurations. While external configuration can offer flexibility, it also poses several centralization risks that warrant attention. Centralization risks arising from the dependence on external configuration include Single Point of Control, Vulnerability to Attacks, Operational Delays, Trust Dependencies, and Decentralization Erosion.

Specifically, the contract owner has the authority to make changes to key parameters of the smart contract, that can heavily impact on its functionality.

```
function setWithdrawFeeFactor(uint8 _factor) external onlyOwner
{
    require(_factor < withdrawFeeFactorMax,
"setWithdrawFeeFactor: max Factor");
    withdrawFeeFactor = _factor;
}

function setRewardRate(uint256 _zfPerSecond) external onlyOwner
{
    zfPerSecond = _zfPerSecond;
}
```

## Recommendation

To address this finding and mitigate centralization risks, it is recommended to evaluate the feasibility of migrating critical configurations and functionality into the contract's codebase itself. This approach would reduce external dependencies and enhance the contract's self-sufficiency. It is essential to carefully weigh the trade-offs between external configuration flexibility and the risks associated with centralization.

## Team Update

The team has acknowledged that this is not a security issue and states: *Currently, the zkSwap Finance Staking contract is under a 48-hour timelock, and all information is transparent, public, and verifiable. More information can be found at https://docs.zkswap.finance/contracts-and-audits/smart-contracts*

# IDI - Immutable Declaration Improvement

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | ZFGovernanceStaking.sol#L34 |
| **Status** | Unresolved |

## Description

The contract declares state variables that their value is initialized once in the constructor and are not modified afterwards. The `immutable` is a special declaration for this kind of state variables that saves gas when it is defined.

```
token
```

## Recommendation

By declaring a variable as immutable, the Solidity compiler is able to make certain optimizations. This can reduce the amount of storage and computation required by the contract, and make it more gas-efficient.

# ITSV - Inadequate Time Setting Validation

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | ZFGovernanceStaking.sol#L126 |
| **Status** | Unresolved |

## Description

The functions `setStartTime` and `setEndTime` allow the contract owner to set the start and end times without sufficient validation. Specifically, there is no check in `setEndTime` to ensure that the `endTimestamp` is set to a value greater than `startTimestamp`. Similarly, in `setStartTime`, there is no safeguard against setting a `startTimestamp` that could be retrospectively later than `endTimestamp`. This oversight can lead to illogical time settings where the end time is earlier than the start time, or the start time is reset to a point beyond the end time, potentially causing functional discrepancies and confusion in the contract's operation.

```
function setStartTime(uint256 _startTime) external onlyOwner {
    startTimestamp = _startTime;
    lastRewardTime = _startTime;
}

function setEndTime(uint256 _endTime) external onlyOwner {
    endTimestamp = _endTime;
}
```

## Recommendation

To resolve this issue and enhance the contract's integrity, it is recommended to incorporate validation checks in both functions. These checks will enforce a logical chronological order, preventing the contract owner from setting times that could disrupt the normal functioning of the contract or confuse users. Implementing these validations is crucial for maintaining the contract’s operational consistency and protecting it against both inadvertent misconfigurations and potential misuse.

# LTA - Locked Tokens Accumulation

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | ZFGovernanceStaking.sol#L106 |
| **Status** | Unresolved |

## Description

The contract's withdrawal mechanism calculates the withdrawal amount based on the user's shares and applies a withdrawal fee. However, due to the integer arithmetic used in Solidity, fractional parts of tokens resulting from this fee calculation are not transferred to the user and are effectively left in the contract. Over time, especially with numerous transactions, these residual amounts could accumulate, leading to a growing pool of locked tokens within the contract's balance.

The presence of these locked tokens in the contract's balance could inadvertently inflate the apparent value of each share when users make subsequent withdrawals. This situation might lead to an unintentional distribution of these locked tokens to users withdrawing their stakes, thereby slightly increasing their withdrawal amounts. While this might seem beneficial to users in the short term, it represents a deviation from the intended token distribution and staking mechanics as designed.

```
function withdraw(uint256 _shares) nonReentrant public {
    // Harvest
    uint256 pending = pendingZF();
    if (pending > 0) {
        IZFToken(token).mint(address(this), pending);
        lastRewardTime = block.timestamp;
    }

    uint256 _withdrawAmount =
(balance().mul(_shares)).div(totalSupply);
    _withdrawAmount =
_withdrawAmount.mul(withdrawFeeFactor).div(withdrawFeeFactorMax
);

    _burn(msg.sender, _shares);
    IERC20(token).safeTransfer(msg.sender, _withdrawAmount);
}
```

## Recommendation

The issue around locked tokens necessitates a careful review, particularly if their accumulation is unintended. It is advisable to implement a mechanism that facilitates the transfer of these residual tokens to the users, ensuring that all distributed rewards align with the user's proportional share in the staking pool. This adjustment would prevent the unintended accumulation of tokens in the contract and maintain the equitable distribution of rewards.

On the other hand, if the retention of locked tokens is an intentional design choice, potentially aimed at enhancing the reward mechanism, this strategy must be transparently communicated to all stakeholders. Clear communication is essential in maintaining user trust and confidence in the contract's operations. Users should be fully informed about how their stakes are being managed and how these design choices might impact their rewards.

## Team Update

The team has acknowledged that this is intended and states: Users that unstake need to pay a penalty, which will then be distributed to loyal stakers still in the pool. This incentivizes long-term stakers and platform loyalty.

# MVN - Misleading Variable Name

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | ZFGovernanceStaking.sol#L25 |
| **Status** | Unresolved |

## Description

The definition of variable `withdrawFeeFactor = 99;` suggests the withdrawal fee is 1%. However, the actual functionality of the variable indicates that it represents the percentage of the withdrawal amount that the user receives (99% in this case), rather than the fee itself. The comment accompanying the variable declaration adds to the confusion by implying that the fee is 1%. This discrepancy between the variable name, its value, and the associated comment can lead to misunderstandings about the contract's fee structure. Misinterpretation of such a critical variable could affect user trust and the overall perception of the contract's transparency.

```
uint8 public withdrawFeeFactor = 99; // 1%

function withdraw(uint256 _shares) nonReentrant public {
    ...
    uint256 _withdrawAmount =
(balance().mul(_shares)).div(totalSupply);
    _withdrawAmount =
_withdrawAmount.mul(withdrawFeeFactor).div(withdrawFeeFactorMax
);
    ...
```

## Recommendation

To enhance clarity and reduce potential confusion, it is recommended to rename the `withdrawFeeFactor` variable to more accurately reflect its functionality. This modification will improve the readability and understanding of the contract's code, thereby fostering greater trust and transparency with its users.

# MEE - Missing Events Emission

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | ZFGovernanceStaking.sol#L112,117,121,126 |
| **Status** | Unresolved |

## Description

The contract performs actions and state mutations from external methods that do not result in the emission of events. Emitting events for significant actions is important as it allows external parties, such as wallets or dApps, to track and monitor the activity on the contract. Without these events, it may be difficult for external parties to accurately determine the current state of the contract.

```
function setWithdrawFeeFactor(uint8 _factor) external onlyOwner
{
    require(_factor < withdrawFeeFactorMax,
"setWithdrawFeeFactor: max Factor");
    withdrawFeeFactor = _factor;
}

function setRewardRate(uint256 _zfPerSecond) external onlyOwner
{
    zfPerSecond = _zfPerSecond;
}

function setStartTime(uint256 _startTime) external onlyOwner {
    startTimestamp = _startTime;
    lastRewardTime = _startTime;
}

function setEndTime(uint256 _endTime) external onlyOwner {
    endTimestamp = _endTime;
}
```

## Recommendation

It is recommended to include events in the code that are triggered each time a significant action is taking place within the contract. These events should include relevant details such as the user's address and the nature of the action taken. By doing so, the contract will be more transparent and easily auditable by external parties. It will also help prevent potential issues or disputes that may arise in the future.

# RES - Redundant Event Statement

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | ZFGovernanceStaking.sol#L28,29 |
| **Status** | Unresolved |

## Description

Event `Deposit` and `Withdraw` are declared but never used within the contract's functions. Events in Solidity are crucial for emitting information from the contract to the outside world, particularly to off-chain applications and user interfaces. They are essential for tracking contract activity and providing transparency in transactions. The declaration of these events without corresponding `emit` statements in functions where deposits and withdrawals occur represents a redundancy in the contract code. This omission could lead to a lack of vital transactional data being available to users or off-chain services, impacting their ability to monitor and respond to deposit and withdrawal actions effectively.

```
event Deposit(address indexed user, uint256 amount);
event Withdraw(address indexed user, uint256 amount);
```

## Recommendation

It is recommended that the contract be updated to include emit statements for the `Deposit` and `Withdraw` events in the respective functions where these actions take place. If the contract's design intentionally omits the use of these events, it would be advisable to remove their declarations to streamline the contract and avoid confusion. In the case of intentional omission, ensuring that adequate mechanisms are in place for tracking and recording deposit and withdrawal actions is essential. Proper implementation and use of events are vital for maintaining the integrity and transparency of smart contract operations.

## RSML - Redundant SafeMath Library

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | ZFGovernanceStaking.sol |
| **Status** | Unresolved |

## Description

SafeMath is a popular Solidity library that provides a set of functions for performing common arithmetic operations in a way that is resistant to integer overflows and underflows.

Starting with Solidity versions that are greater than or equal to 0.8.0, the arithmetic operations revert to underflow and overflow. As a result, the native functionality of the Solidity operations replaces the SafeMath library. Hence, the usage of the SafeMath library adds complexity, overhead and increases gas consumption unnecessarily.

```
library SafeMath {...}
```

## Recommendation

The team is advised to remove the SafeMath library. Since the version of the contract is greater than `0.8.0` then the pure Solidity arithmetic operations produce the same result.

If the previous functionality is required, then the contract could exploit the `unchecked { ... }` statement.

Read more about the breaking change on https://docs.soliditylang.org/en/v0.8.16/080-breaking-changes.html#solidity-v0-8-0-breaking-changes.

## RSW - Redundant Storage Writes

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | ZFGovernanceStaking.sol#L112,117,121,126 |
| **Status** | Unresolved |

## Description

The contract modifies the state of the following variables without checking if their current value is the same as the one given as an argument. As a result, the contract performs redundant storage writes, when the provided parameter matches the current state of the variables, leading to unnecessary gas consumption and inefficiencies in contract execution.

```
function setWithdrawFeeFactor(uint8 _factor) external onlyOwner
{
    require(_factor < withdrawFeeFactorMax,
"setWithdrawFeeFactor: max Factor");
    withdrawFeeFactor = _factor;
}

function setRewardRate(uint256 _zfPerSecond) external onlyOwner
{
    zfPerSecond = _zfPerSecond;
}
```

## Recommendation

The team is advised to implement additional checks within to prevent redundant storage writes when the provided argument matches the current state of the variables. By incorporating statements to compare the new values with the existing values before proceeding with any state modification, the contract can avoid unnecessary storage operations, thereby optimizing gas usage.

## L04 - Conformance to Solidity Naming Conventions

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | ZFGovernanceStaking.sol#L54,58,70,97,112,117,121,126 |
| **Status** | Unresolved |

### Description

The Solidity style guide is a set of guidelines for writing clean and consistent Solidity code. Adhering to a style guide can help improve the readability and maintainability of the Solidity code, making it easier for others to understand and work with.

The followings are a few key points from the Solidity style guide:

1. Use camelCase for function and variable names, with the first letter in lowercase (e.g., myVariable, updateCounter).
2. Use PascalCase for contract, struct, and enum names, with the first letter in uppercase (e.g., MyContract, UserStruct, ErrorEnum).
3. Use uppercase for constant variables and enums (e.g., MAX_VALUE, ERROR_CODE).
4. Use indentation to improve readability and structure.
5. Use spaces between operators and after commas.
6. Use comments to explain the purpose and behavior of the code.
7. Keep lines short (around 120 characters) to improve readability.

```
address _user
uint256 _from
uint256 _to
uint256 _amount
uint256 _shares
uint8 _factor
uint256 _zfPerSecond
uint256 _startTime
uint256 _endTime
```

## Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.
Find more information on the Solidity documentation
https://docs.soliditylang.org/en/v0.8.17/style-guide.html#naming-convention.

## L07 - Missing Events Arithmetic

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | ZFGovernanceStaking.sol#L118,123,127 |
| **Status** | Unresolved |

### Description

Events are a way to record and log information about changes or actions that occur within a contract. They are often used to notify external parties or clients about events that have occurred within the contract, such as the transfer of tokens or the completion of a task.

It's important to carefully design and implement the events in a contract, and to ensure that all required events are included. It's also a good idea to test the contract to ensure that all events are being properly triggered and logged.

```
zfPerSecond = _zfPerSecond
lastRewardTime = _startTime
endTimestamp = _endTime
```

### Recommendation

By including all required events in the contract and thoroughly testing the contract's functionality, the contract ensures that it performs as intended and does not have any missing events that could cause issues with its arithmetic.

## L16 - Validate Variable Setters

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | ZFGovernanceStaking.sol#L34 |
| **Status** | Unresolved |

### Description

The contract performs operations on variables that have been configured on user-supplied input. These variables are missing of proper check for the case where a value is zero. This can lead to problems when the contract is executed, as certain actions may not be properly handled when the value is zero.

```
token = _token
```

### Recommendation

By adding the proper check, the contract will not allow the variables to be configured with zero value. This will ensure that the contract can handle all possible input values and avoid unexpected behavior or errors. Hence, it can help to prevent the contract from being exploited or operating unexpectedly.

## L19 - Stable Compiler Version

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | ZFGovernanceStaking.sol#L3 |
| **Status** | Unresolved |

### Description

The `^` symbol indicates that any version of Solidity that is compatible with the specified version (i.e., any version that is a higher minor or patch version) can be used to compile the contract. The version lock is a mechanism that allows the author to specify a minimum version of the Solidity compiler that must be used to compile the contract code. This is useful because it ensures that the contract will be compiled using a version of the compiler that is known to be compatible with the code.

```
pragma solidity ^0.8.0;
```

### Recommendation

The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest version that provides all the required functionality for the codebase. As a result, the project will be compiled in a well-tested LTS (Long Term Support) environment.

# Functions Analysis

| Contract | Type | Bases | | |
|---|---|---|---|---|
| | **Function Name** | **Visibility** | **Mutability** | **Modifiers** |
| | | | | |
| **ZFToken** | Implementation | ERC20, Ownable | | |
| | | Public | ✓ | - |
| | mint | Public | ✓ | onlyMinter |
| | burn | Public | ✓ | - |
| | addMinter | Public | ✓ | onlyOwner |
| | removeMinter | Public | ✓ | onlyOwner |
| | getMinterLength | Public | | - |
| | isMinter | Public | | - |
| | getMinter | Public | | - |
| | | | | |
| **Strings** | Library | | | |
| | toString | Internal | | |
| | toStringSigned | Internal | | |
| | toHexString | Internal | | |
| | toHexString | Internal | | |
| | toHexString | Internal | | |
| | equal | Internal | | |
| | | | | |
| **StorageSlot** | Library | | | |

| | | | | |
|---|---|---|---|---|
| | getAddressSlot | Internal | | |
| | getBooleanSlot | Internal | | |
| | getBytes32Slot | Internal | | |
| | getUint256Slot | Internal | | |
| | getStringSlot | Internal | | |
| | getStringSlot | Internal | | |
| | getBytesSlot | Internal | | |
| | getBytesSlot | Internal | | |
| | | | | |
| **ShortStrings** | Library | | | |
| | toShortString | Internal | | |
| | toString | Internal | | |
| | byteLength | Internal | | |
| | toShortStringWithFallback | Internal | ✓ | |
| | toStringWithFallback | Internal | | |
| | byteLengthWithFallback | Internal | | |
| | | | | |
| **Nonces** | Implementation | | | |
| | nonces | Public | | - |
| | _useNonce | Internal | ✓ | |
| | _useCheckedNonce | Internal | ✓ | |
| | | | | |
| **Context** | Implementation | | | |

| | | | | |
|---|---|---|---|---|
| | _msgSender | Internal | | |
| | _msgData | Internal | | |
| | _contextSuffixLength | Internal | | |
| | | | | |
| **SignedMath** | Library | | | |
| | max | Internal | | |
| | min | Internal | | |
| | average | Internal | | |
| | abs | Internal | | |
| | | | | |
| **Math** | Library | | | |
| | tryAdd | Internal | | |
| | trySub | Internal | | |
| | tryMul | Internal | | |
| | tryDiv | Internal | | |
| | tryMod | Internal | | |
| | max | Internal | | |
| | min | Internal | | |
| | average | Internal | | |
| | ceilDiv | Internal | | |
| | mulDiv | Internal | | |
| | mulDiv | Internal | | |
| | sqrt | Internal | | |

| | | | | |
|---|---|---|---|---|
| | sqrt | Internal | | |
| | log2 | Internal | | |
| | log2 | Internal | | |
| | log10 | Internal | | |
| | log10 | Internal | | |
| | log256 | Internal | | |
| | log256 | Internal | | |
| | unsignedRoundsUp | Internal | | |
| | | | | |
| **MessageHashUtils** | Library | | | |
| | toEthSignedMessageHash | Internal | | |
| | toEthSignedMessageHash | Internal | | |
| | toDataWithIntendedValidatorHash | Internal | | |
| | toTypedDataHash | Internal | | |
| | | | | |
| **EIP712** | Implementation | IERC5267 | | |
| | | Public | ✓ | - |
| | _domainSeparatorV4 | Internal | | |
| | _buildDomainSeparator | Private | | |
| | _hashTypedDataV4 | Internal | | |
| | eip712Domain | Public | | - |
| | _EIP712Name | Internal | | |
| | _EIP712Version | Internal | | |

| | | | | |
|---|---|---|---|---|
| | | | | |
| **ECDSA** | Library | | | |
| | tryRecover | Internal | | |
| | recover | Internal | | |
| | tryRecover | Internal | | |
| | recover | Internal | | |
| | tryRecover | Internal | | |
| | recover | Internal | | |
| | _throwError | Private | | |
| | | | | |
| **IERC20** | Interface | | | |
| | totalSupply | External | | - |
| | balanceOf | External | | - |
| | transfer | External | ✓ | - |
| | allowance | External | | - |
| | approve | External | ✓ | - |
| | transferFrom | External | ✓ | - |
| | | | | |
| **ERC20** | Implementation | Context, IERC20, IERC20Metadata, IERC20Errors | | |
| | | Public | ✓ | - |
| | name | Public | | - |
| | symbol | Public | | - |

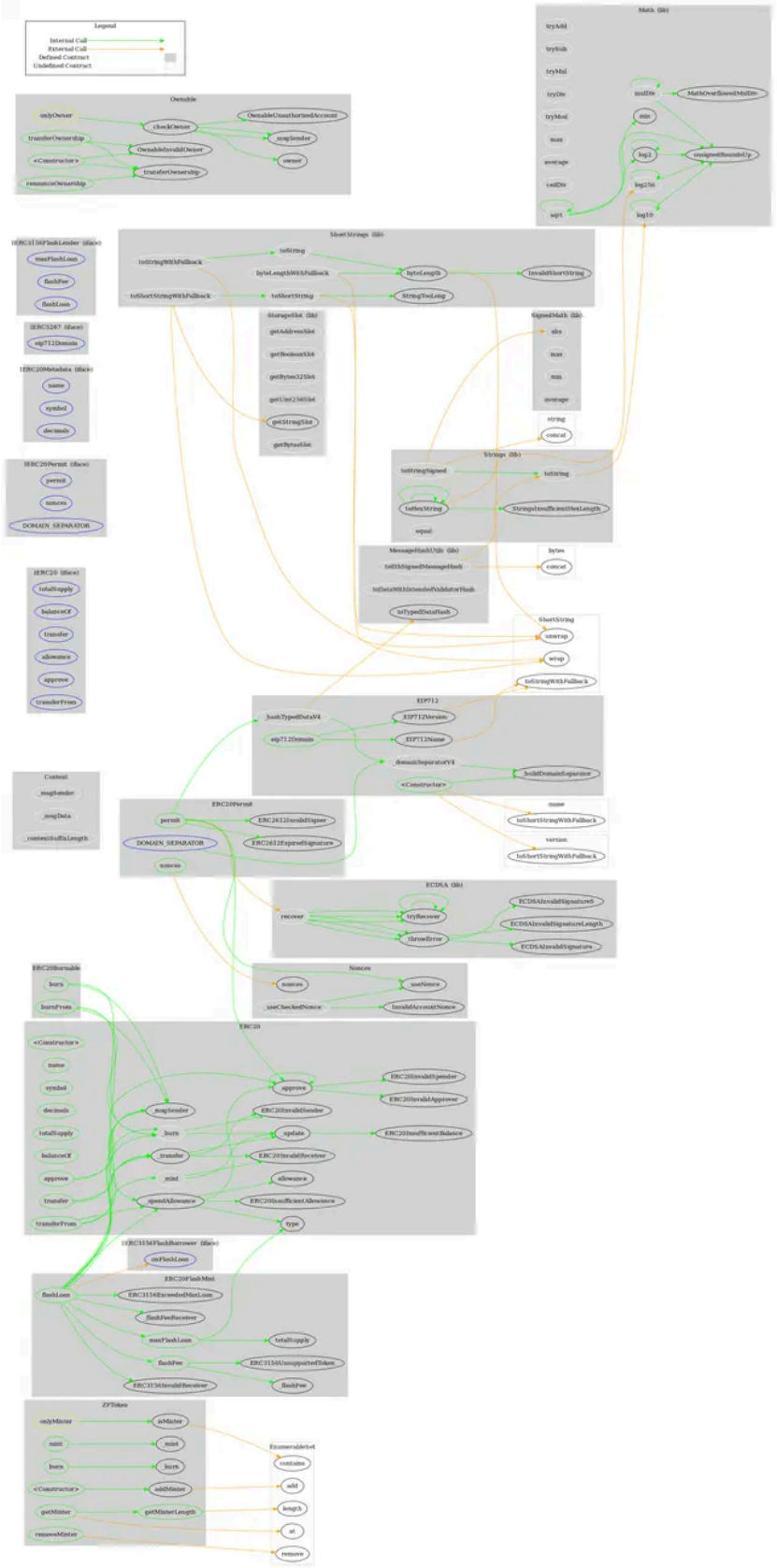
| | | | | |
|---|---|---|---|---|
| | decimals | Public | | - |
| | totalSupply | Public | | - |
| | balanceOf | Public | | - |
| | transfer | Public | ✓ | - |
| | allowance | Public | | - |
| | approve | Public | ✓ | - |
| | transferFrom | Public | ✓ | - |
| | _transfer | Internal | ✓ | |
| | _update | Internal | ✓ | |
| | _mint | Internal | ✓ | |
| | _burn | Internal | ✓ | |
| | _approve | Internal | ✓ | |
| | _approve | Internal | ✓ | |
| | _spendAllowance | Internal | ✓ | |
| | | | | |
| **IERC20Permit** | Interface | | | |
| | permit | External | ✓ | - |
| | nonces | External | | - |
| | DOMAIN_SEPARATOR | External | | - |
| | | | | |
| **IERC20Metadata** | Interface | IERC20 | | |
| | name | External | | - |
| | symbol | External | | - |

| | | | | |
|---|---|---|---|---|
| | decimals | External | | - |
| | | | | |
| **ERC20Permit** | Implementation | ERC20, IERC20Permit, EIP712, Nonces | | |
| | | Public | ✓ | EIP712 |
| | permit | Public | ✓ | - |
| | nonces | Public | | - |
| | DOMAIN_SEPARATOR | External | | - |
| | | | | |
| **ERC20FlashMint** | Implementation | ERC20, IERC3156FlashLender | | |
| | maxFlashLoan | Public | | - |
| | flashFee | Public | | - |
| | _flashFee | Internal | | |
| | _flashFeeReceiver | Internal | | |
| | flashLoan | Public | ✓ | - |
| | | | | |
| **ERC20Burnable** | Implementation | Context, ERC20 | | |
| | burn | Public | ✓ | - |
| | burnFrom | Public | ✓ | - |
| | | | | |
| **IERC20Errors** | Interface | | | |
| | | | | |
| **IERC721Errors** | Interface | | | |

| | | | | |
|---|---|---|---|---|
| | | | | |
| **IERC1155Errors** | Interface | | | |
| | | | | |
| **IERC5267** | Interface | | | |
| | eip712Domain | External | | - |
| | | | | |
| **IERC3156FlashLender** | Interface | | | |
| | maxFlashLoan | External | | - |
| | flashFee | External | | - |
| | flashLoan | External | ✓ | - |
| | | | | |
| **IERC3156FlashBorrower** | Interface | | | |
| | onFlashLoan | External | ✓ | - |
| | | | | |
| **Ownable** | Implementation | Context | | |
| | | Public | ✓ | - |
| | owner | Public | | - |
| | _checkOwner | Internal | | |
| | renounceOwnership | Public | ✓ | onlyOwner |
| | transferOwnership | Public | ✓ | onlyOwner |
| | _transferOwnership | Internal | ✓ | |

# Inheritance Graph

# Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
Ownable
onlyOwner
checkOwner
OwnableUnauthorizedAccount
transferOwnership
OwnableInvalidOwner
msgSender
«Constructor»
owner
renounceOwnership
transferOwnership
Math (lib)
tryAdd
trySub
tryMul
tryDiv
tryMod
max
average
ceilDiv
sqrt
mulDiv
MathOverflowedMulDiv
min
log2
unsignedRoundsUp
log256
log10
IERC3156FlashLender (iface)
maxFlashLoan
flashFee
flashLoan
ShortStrings (lib)
toString
toStringWithFallback
byteLength
InvalidShortString
byteLengthWithFallback
toShortString
StringTooLong
toShortStringWithFallback
IERC5267 (iface)
eip712Domain
StorageSlot (lib)
getAddressSlot
getBooleanSlot
getBytes32Slot
getUint256Slot
getStringSlot
getBytesSlot
SignedMath (lib)
abs
max
min
average
string
concat
IERC20Metadata (iface)
name
symbol
decimals
IERC20Permit (iface)
permit
nonces
DOMAIN_SEPARATOR
Strings (lib)
toStringSigned
toString
toHexString
StringsInsufficientHexLength
equal
MessageHashUtils (lib)
toEthSignedMessageHash
toDataWithIntendedValidatorHash
toTypedDataHash
bytes
concat
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
ShortString
unwrap
wrap
toStringWithFallback
EIP712
_hashTypedDataV4
eip712Domain
EIP712Version
EIP712Name
domainSeparatorV4
«Constructor»
buildDomainSeparator
Context
_msgSender
_msgData
_contextSuffixLength
ERC20Permit
permit
ERC2612InvalidSigner
DOMAIN_SEPARATOR
ERC2612ExpiredSignature
nonces
name
toShortStringWithFallback
version
toShortStringWithFallback
ECDSA (lib)
recover
tryRecover
throwError
ECDSAInvalidSignature
ECDSAInvalidSignatureLength
ECDSAInvalidSignature
ERC20Burnable
burn
burnFrom
Nonces
nonces
useNonce
useCheckedNonce
InvalidAccountNonce
ERC20
«Constructor»
name
symbol
decimals
totalSupply
balanceOf
approve
transfer
transferFrom
_msgSender
_burn
transfer
_mint
spendAllowance
approve
update
type
ERC20InvalidSpender
ERC20InvalidApprover
ERC20InvalidSender
ERC20InsufficientBalance
ERC20InvalidReceiver
allowance
ERC20InsufficientAllowance
IERC3156FlashBorrower (iface)
onFlashLoan
ERC20FlashMint
flashLoan
ERC3156ExceededMaxLoan
flashFeeReceiver
maxFlashLoan
totalSupply
flashFee
ERC3156UnsupportedToken
ERC3156InvalidReceiver
flashFee
ZFToken
onlyMinter
isMinter
mint
mint
burn
burn
«Constructor»
addMinter
getMinter
getMinterLength
removeMinter
EnumerableSet
contains
add
length
at
remove

# Summary

zkSwap Finance GovernanceStaking contract implements a staking mechanism. This audit investigates security issues, business logic concerns and potential improvements.

# Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope’s prior written consent. This report is not nor should be considered an “endorsement” or “disapproval” of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk Cyberscope’s position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

# About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.

**The Cyberscope team**

https://www.cyberscope.io