Propchain

SMART CONTRACT AUDIT

zokyo

May 16th 2023 | v. 1.0

Security Audit Score

PASS

Zokyo Security has concluded that this smart contract passes security qualifications to be listed on digital asset exchanges.

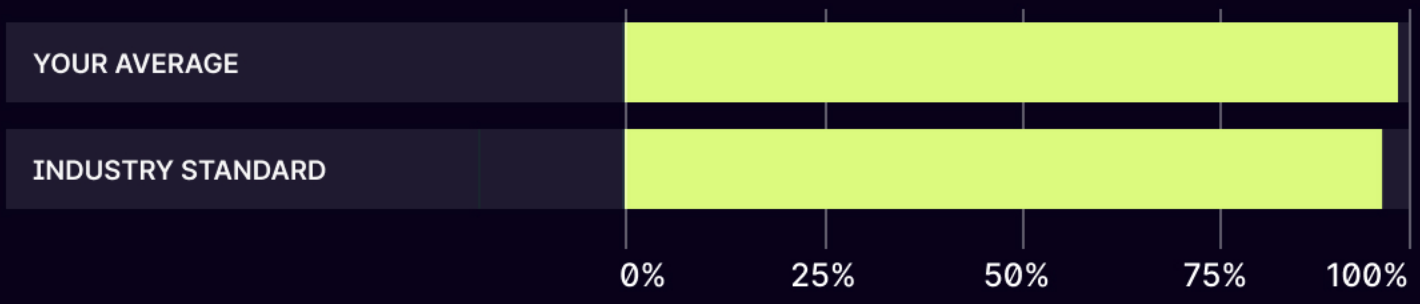
TECHNICAL SUMMARY

This document outlines the overall security of the Propchain smart contracts evaluated by the Zokyo Security team.

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

Contract Status

Testable Code



100% of the code is testable, which is sufficient to correspond the industry standard

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract 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 Propchain team put in place a bug bounty program to encourage further active analysis of the smart contract.

Table of Contents

AUDITING STRATEGY AND TECHNIQUES APPLIED

The source code of the smart contract was taken from the Propchain repository:
https://github.com/propchain-global/propchain-solidity-contracts

Initial commit: 1c8cdafefc8a2ee3e2dba3a617f72e7c9d6615d6
Final commit: e0b5be0fc9340149c32d740a16cfe571e3ed9bc0

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

- PlatformToken.sol
- EIP3009.sol
- PlatformVesting.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 resent vulnerabilities;
- Meets best practices in code readability, etc.

Zokyo Security has followed best practices and industry-standard techniques to verify the implementation of Propchain smart contracts. 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 contracts by industry leaders.

03 Testing contract logic against common and uncommon attack vectors.

04 Thorough manual review of the codebase line by line.

Executive Summary

The Zokyo Security team conducted an audit of the Propchain protocol. The scope of the audit included PlatformToken, PlatformVesting, and EIP3009. PlatformToken represents an ERC20 with additional functionality, such as gasless transactions with off-chain approval via signatures (according to EIP712 and EIP3009 standards) for Staking and Vesting and an anti-snipe system for the token sale period. PlatformVesting, in its turn, is a vesting smart contract that implements a linear schedule model for token distribution. A detailed overview of smart contracts can be seen in the Protocol Overview section.

The audit's goal was to analyze the security of tokens, gasless transaction, and integration of the anti-snipe system, verify the correctness of vesting and tokens distribution, check smart contracts against the list of common vulnerabilities as well as our internal checklist, and validate that contracts correspond to Solidity best practices in terms of code quality and gas optimization.

Auditors have paid particular attention to vesting schedules creation and wallets insertion into vesting schedules. Two high-risk issues were related to the admin functionality of the vesting. The first issue was connected to the ability to remove already disabled vesting. Such functionality could cause storage corruption and prevent smart contracts from operating correctly. The second high-risk issue was connected to the ability to add users to the same vesting without removing a previous user. The normal flow of vesting implies that the admin can insert only one wallet at a time into a particular vesting. In order to replace one wallet with another, an admin has to remove the wallet first and then insert a new one. However, the inserting functionality didn't have the necessary check to validate if a vesting already had an inserted wallet. As a result, several users could have claimed rewards from single staking simultaneously. Other issues were connected to the absence of SafeERC20 usage for token transfers, interaction with non-existing vesting schedules, lack of validations, code optimization, and validation of contracts' logic. The Propchain team has successfully fixed or verified all the issues.

One of the issues was originally connected to the usage of the anti-snipe system. The token can disable the anti-snipe system once and for all. However, it can't be enabled later. The Propchain team has verified that the system is necessary only during the sale period of the token and will be turned off later. It is required behavior to ensure that the token

corresponds to the criteria of listing on most of the exchanges.

The overall security of smart contracts is high enough. Contracts are well-written though lack of natSpec documentation. Both smart contracts are well-tested by the Propchain and Zokyo Security teams. The protocol passed all the security tests once the fixes of the issues were applied.

Protocol Overview

PlatformVesting.sol

The PlatformVesting contract is a smart contract designed to manage vesting pools for the distribution of tokens on a platform. It allows platform administrators to create and manage vesting pools with separate liquidity, whitelists, and parameters. Users can claim their vested tokens according to the vesting schedule defined for each pool. The contract also provides functionality for increasing and decreasing liquidity in the vesting pools, as well as adding and removing users from the whitelists.

PlatformToken.sol

The PlatformToken contract is an ERC20 token with additional functionality, such as restrictions on transfers to non-whitelisted contracts, transfers without approval, and integration with an Antisnipe mechanism. The contract also allows burning tokens and updating addresses for vesting, staking, and cashback contracts.
The Antisnipe mechanism, in the context of the PlatformToken contract, is designed to prevent malicious actors from exploiting the token transfers. It adds an additional layer of security to the token transfer process by validating the transfer conditions before allowing the transfer to proceed.

The Antisnipe mechanism is integrated there by using the IAntisnipe interface and its implementation. The `assureCanTransfer` function validates the transfer conditions and ensures that the transfer is not part of a malicious activity, such as front-running or sniping. If the transfer does not meet the required conditions, the function reverts the transaction, preventing the transfer from occurring.
The PlatformToken contract also provides functions to update the Antisnipe contract address and to disable the Antisnipe mechanism. The platform admin can only call these functions, providing control over the Antisnipe mechanism's usage and configuration.

EIP3009.sol

The EIP3009 contract is an abstract contract that implements the EIP-3009 standard for token transfers with off-chain authorization. It provides a method to transfer tokens with a signed authorization from the token holder, allowing for gasless transactions and more efficient token transfers.

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” depending on whether they have been fixed or addressed. 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

HIGH-1 | RESOLVED

Same vesting can be removed more than once.

PlatformVesting.sol: removeVesting().
The same vesting can be removed more than once with function since the count of vesting doesn't change, making it possible to avoid all the validations in lines 214-215. Thus, the same vesting ID can be passed more than once to the function. Due to this, the variable `totalRemainingAllocatedAmount` will be decreased by values from the same vesting, which might potentially corrupt the variable. Though only an admin can execute this function, the issue is marked as high, as an admin may execute this function with the same ID by mistake.

Recommendation:
Add a validation that vesting with provided ID is not active. It is also recommended to clean values from all mappings with operator delete to properly clean them and decrease the chances of interacting with removed vestings.

Post-audit.
Validation and cleaning values with delete operator were added.

Wallets can be inserted in the same vesting without removal.

PlatformVesting.sol: insertWalletListToVesting().
Based on the logic of this function, only one wallet can participate in a particular vesting simultaneously. In order to change the wallet in vesting, the admin should remove the previous wallet first - using the function removeWalletListFromVesting(). However, the status `active` of `_vestingToUser[vestingId]` is not checked during the execution of the function insertWalletListToVesting(). As a result, two or more wallets can claim vested tokens from a single particular vesting, resulting in the lack of tokens for all vestings and wallets. Though only the admin can insert wallets into vesting, it is still recommended to check if vesting already has an active wallet.

Recommendation:
Check that vesting doesn't have an active wallet before inserting a new one (e.g. by validating that `vl.active` is equal to false).

Post-audit.
Validation for `vl.active` was added.

Incorrect erc20 interface and unchecked transfer.

PlatformToken.sol, IPlatformToken.sol.
PlatformVesting.sol: decreaseLiquidity(), line 291; _claim(), line 393.
IPlatformToken has an incorrect ERC20 transfer() function interface. A contract compiled with Solidity > 0.4.22 interacting with these functions will fail to execute them, as the return value is missing. In addition, this transfer() function without a result verifying a returned value indicates whether the operation succeeded, i.e., it is not reverted on and does not handle errors, and transfer may not be performed. It is recommended to handle the return value to prevent unintended cases and to use `SafeERC20` by OpenZeppelin, which supports both tokens' implementations: those with a return boolean value and those without.

Recommendation:
Use the methods `safeTransfer()` and `safeTransferFrom()` from the library `SafeERC20Upgradeable` by OpenZeppelin.

Post-audit.
Transfer function was removed from IPlatfromToken.sol and in PlatfromVesting.sol.
SafeERC20 is used now.

MEDIUM-2 | RESOLVED

Non-existing vesting might be removed or increased.

PlatformVesting.sol: removeVesting() line 214, modifier existingVesting().
When functions are called, an ID of vesting to be removed is passed as function parameters `vestingId`. This parameter is then validated to be less or equal to `_vestingCount.` However, the actual ID of the last vesting can't be equal to `_vestingCount` due to the design of the function insertVestingList(). The issue is marked as a medium since in case function increaseLiquidity() is called with non-existing ID, value `vesting.unallocatedAmount += amount` will be increased, and funds will be transferred. However, this structure field can be overwritten with an incorrect value when a new batch of vestings is added.

Recommendation:
Change the condition in require statements to "vestingId < _vestingCount".

Post-audit.
Condition in the require statements was changed to "vestingId < _vestingCount".

INFO-1 | RESOLVED

Lack of validation.

PlatformVesting.sol: setTgeDate(), constructor();
PlatformToken.sol: constructor(), setAntisnipeAddress().
Either not all or none of the parameters passed to these functions are validated. It can lead to unexpected behavior in further use, which is why it is recommended to validate parameters. For example, addresses should be validated not to be equal to zero addresses, and numeric parameters should be validated not to be equal to 0.

Recommendation:
Add the necessary validation.

Post-audit.
Validation was added.

INFO-2 | RESOLVED

Requirements instead of custom errors are used.

Since 0.8.4, it is recommended to replace all requirements with the custom errors with meaningful names to reduce contract bytecode, make the code more readable and consistent, and reduce the gas consumption in reverse.
It is worth noting that it can transmit additional parameters to make the errors in specific cases more informative and be used cross-contract.
For example, `error MeaningfulName(parameters if needed)`.

Recommendation:
Use custom errors instead of requires to reduce bytecode, make errors more informative, improve readability and consistency.

Post-audit.
Custom errors are now used in contracts instead of require statements.

INFO-3 | RESOLVED

Redundant require statement.

1) PlatformVesting.sol: removeVesting(), line 210; distributeAmount(), line 392.
The require statement `require(0 <= vestingId, "invalid id");` is redundant, as vestingId is an unsigned integer and will always be greater than or equal to 0.
2) PlatformVesting.sol: removeVesting(), line 249.
The require statement `require(vp.active, "vesting does not exist.")` is redundant, as vp.active state cannot be set to false due to the fact that there is no such functionality in the contract and this require statement will always be true.

Recommendation:
Remove the redundant require statement.

Post-audit.
Redundant require statements were removed.

INFO-4 | RESOLVED

Unused state variable.

PlatformToken.sol: variable `_cashback`, line 29.
EIP3009.sol: _INVALID_SIGNATURE_ERROR, line 21.

Recommendation:
Remove unused state variables **OR** verify if this variable is intended to be used in future updates.

Post-audit.
Unused variables were removed.

INFO-5 | RESOLVED

Missing inheritance.

PlatformVesting.sol → IPlatformVesting.sol;
PlatformToken.sol → IPlatformToken.sol.
The contracts do not inherit from their interfaces, which may lead to inconsistencies and difficulties in understanding the contract's functionality and interactions with other contracts in the project.

Recommendation:
Update the contracts to inherit from the interfaces and ensure that contracts implement all required functions and events defined in the interface.

From the client.
Contracts inherit from their interfaces now.

INFO-6 | RESOLVED

Inaccurate version pragma.

`pragma solidity ^0.8.17` is used in contracts, while the latest is 0.8.19. Older versions may contain bugs and vulnerabilities, and be less optimized in terms of gas. It is recommended to use the latest version of Solidity.

Recommendation:
Specify the latest version of Solidity in the pragma statement.

From the client.
Last Solidity version is used in contracts now.

INFO-7 | RESOLVED

No events emitted in setters.

PlatformToken.sol: updateVestingAddress(), updateStakingAddress(), setAntisnipeDisable(), setAntisnipeAddress();
PlatformVesting.sol: setTgeDate(), updateTokenAddress(), removeVesting(), _setVesting().
Emitting events can be helpful for tracking changes and debugging.

Recommendation:
Emit events in setters.

Post-audit.
Events are emitted in setters now.

Duplicated code.

1) EIP3009.sol: transferWithAuthorization().
In the transferWithAuthorization() function, the return value is true if the recovered signer equals the 'from' address; otherwise, it returns false. However, this check is already performed earlier in the function with a require statement in line 65, which means that if the function execution reaches the return statement, it will always return true. It makes either the `if` or require statement redundant and can be removed.

Recommendation:
Remove the `if` and always return true, or remove require statement and perform validation only in specialTransferFrom().

2) PlatformVesting.sol: decreaseLiquidity(), line 312, 315.
In the `decreaseLiquidity` function, two identical variables `availableSenderBalance` and `ownBalance` are created on lines 312 and 315, both representing the contract's balance of the token. This redundancy can lead to confusion and potential inefficiencies in the code execution, as the contract's balance is queried twice from the token contract.

Recommendation.
Remove one of the redundant variables and use a single variable to represent the contract's token balance. This will simplify the code and reduce the number of external calls to the token contract.

Post-audit.
1) The return value of the function and a check were removed.
2) Variable was removed.

INFO-10 | RESOLVED

Redundant getter.

PlatformVesting.sol: getVestingCount().
Function represents a getter which returns variable `_vestingCount`. However this variable is already public which means that smart contract will have a getter for this function. This makes implementation of getVestingCount() which increases the bytecode of the contract.

Recommendation:
Verify the necessity of function **OR** remove it **OR** make variable private.

Post-audit.
Function was removed.

INFO-11 | RESOLVED

Redundant modifier.

PlatformVesting.sol: distributeAmount().
Function is private and is called only from function airdrop(). Both functions have the modifier “onlyPlatformAdmin”. Thus usage of this modifier in private function is redundant and only increases gas spendings.

Recommendation:
Remove modifier from function distributeAmount().

Post-audit.
Modifier was removed.

INFO-12 | VERIFIED

3rd party dependency.

PlatformToken.sol: antisnipe.
The token depends on the 3rd part contract, Antisnipe. An admin is able to set any address to this variable at any time with setter setAntisnipeAddress(). Since the implementation of Antisnipe is out of the scope of the current audit, the auditors can't claim if the contract Antisnipe is safe to be interacted with and that it is integrated correctly into PlatformToken. Also, dependency is crucial since it performs unknown extra actions during the token transfer.

Recommendation:
Verify what implementation and protocol will be used as Antisnipe.

Post-audit.
The Propchain team has verified that Antisnipe is a necessary part of the token and that it is

INFO-13 | VERIFIED

Addresses of vesting and staking can be changed.

PlatformToken.sol: updateVestingAddress(), updateStakingAddress().
Admin is able to change addresses of vesting and staking at any time. This can lead to inconsistency and unexpected behavior.

Recommendation:
Verify that they should be set more than once. Consider implementing additional validation to provided parameters such as validating that they support the interface of Staking and Vesting.

Post-audit.
The Propchain team has verified that ability to update vesting and staking is necessary as they might be updated in the future.

Incorrect event location.

PlatformVesting.sol: claim(), distributeAmount(), event Claim().
The `Claim` event is only emitted in the `claim` function, but not in the `_claim` internal function. As a result, when the `distributeAmount` function is called, which internally calls the `_claim` function, the `Claim` event is not emitted. This can lead to incomplete or inconsistent information about claims made by users, especially when the `distributeAmount` function is used for airdropping tokens to multiple users.

Recommendation:
Move the `Claim` event emission to the `_claim` internal function to ensure that it is consistently emitted for all claims, regardless of whether made through the `claim` function or the `distributeAmount` function. It will provide a more accurate and complete record of all user claims.

Post-audit. Event `Claim` was moved.

Absence of mapping entry removal.

PlatformVesting.sol: removeWalletListFromVesting(), removeVesting().
In the `removeWalletListFromVesting()` function, the `delete` operator is not used to clean the mapping entries for the removed wallets. Instead, the function only sets the `isActive` field to `false` for both the `UserProperties` and `VestingLink` structs. While this marks the wallet as removed from the vesting, it does not clean the mapping entries, which might lead to unnecessary storage usage.

Recommendation:
Consider adding the `delete` operator to clean the mapping entries for the removed wallets in the `removeWalletListFromVesting()` function **OR** verify if there is a specific reason for not cleaning the mapping entries in this function.

Post-audit.
The Propchain team has verified that it should be possible to "re-add" users later so that users are then able to claim all the payouts they "missed" by not being part of the vesting temporarily.

	PlatformToken.sol	EIP3009.sol
Re-entrancy	Pass	Pass
Access Management Hierarchy	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass
Unexpected Ether	Pass	Pass
Delegatecall	Pass	Pass
Default Public Visibility	Pass	Pass
Hidden Malicious Code	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass
External Contract Referencing	Pass	Pass
Short Address/ Parameter Attack	Pass	Pass
Unchecked CALL Return Values	Pass	Pass
Race Conditions / Front Running	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass
Floating Points and Precision	Pass	Pass
Tx.Origin Authentication	Pass	Pass
Signatures Replay	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass

	PlatformVesting.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 Propchain in verifying the correctness of their contract 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 Propchain contract requirements for details about issuance amounts and how the system handles these.

Propchain audit tests

Platform Vesting

 # Token

- ✓ Should set new vesting address
- ✓ Should revert setting new vesting address as address zero
- ✓ Should set new staking address
- ✓ Should revert setting new staking address as address zero
- ✓ Should burn tokens
- ✓ Should set new antisnipe address
- ✓ Should revert setting new antisnipe address as address zero
- ✓ Should disable antisnipe
- ✓ Should revert disabling antisnipe second time
- ✓ Should revert deploying token with wrong parameters
- ✓ Should transfer signed tokens
- ✓ Should revert if try to transfer from as non-vesting and non-staking address
- ✓ Should revert if try to transfer zero tokens
- ✓ Should revert if try to transfer from with wrong signature
- ✓ Should revert transfer from if valid time is not yet reached
- ✓ Should revert transfer from if valid time is already passed
- ✓ Should revert transfer from if nonce is already used
- ✓ Should check antisnipe if it is set

 # Only admin modifier

- ✓ Should revert setting new vesting address as non-admin
- ✓ Should revert setting new staking address as non-admin
- ✓ Should revert burning tokens as non-admin
- ✓ Should revert setting new antisnipe address as non-admin
- ✓ Should revert disabling antisnipe as non-admin

Platform Vesting

 # Vesting

- ✓ Should create vesting

✓ Should revert create vesting with empty array
✓ Should create vesting with zero amount
✓ Should revert creating vesting with tge amount greater than total amount
✓ Should revert creating vesting with start time in the past
✓ Should revert creating vesting with start time less than tgeStartDate
✓ Should revert creating vesting if tick count equal to zero
✓ Should revert creating vesting if tick duration equal to zero
✓ Should add user to vesting
✓ Should revert if try to add user with arrays length mismatch
✓ Should revert if try to add user with empty arrays
✓ Should revert if try to add user to non existing vesting
✓ Should not add one use to multiple vestings
✓ Should not add multiple users to one vesting
✓ Should increase liquidity
✓ Should revert adding liquidity for non existing vesting
✓ Should revert adding zero liquidity
✓ Should remove vesting without assigned user
✓ Should remove vesting with assigned user
✓ Should revert removing vesting if it already was removed
✓ Should revert removing non existing vesting
✓ Should decrease liquidity (38ms)
✓ Should revert decreasing liquidity for zero amount
✓ Should revert decreasing liquidity for amount greater than balance
✓ Should revert decreasing liquidity if it is already assigned to user
✓ Should revert decreasing liquidity if it is already assigned to vesting
✓ Should remove wallet from vesting
✓ Should revert removing wallet from vesting if user was already removed
✓ Should revert removing zero wallets from vesting

 # Claim

✓ Should claim (43ms)
✓ Should claim with 2 vestings (55ms)
✓ Should claim as airDrop for 2 users (57ms)
✓ Should claim as airDrop for 1 user with smaller batch size even if 2 is added (52ms)
✓ Should revert airdrop if offset is not in the vesting ids range
✓ Should estimate zero rewards if user is not in vesting
✓ Should not revert airdrop if vesting was added but then removed
✓ Should revert claim if there is no funds to be claimed

 # Vesting with start in future and past

✓ Should estimate 0 rewards if vesting start is in future
✓ Should not estimate 0 rewards if vesting start is in past
✓ Should not estimate 0 rewards if vesting start is in past with ticks still in range
✓ Should estimate zero rewards if user already claimed assigned tokens and vesting was in the past

 # Setters
✓ Should revert contract creation with zero address as admin panel
✓ Should set new TgeDate
✓ Should revert if try to set new TgeDate in the past
✓ Should set new Token Address
✓ Should revert updating token address if new address is zero
 # OnlyAdmin Modificator
✓ Should revert if try to set new TgeDate from non admin caller
✓ Should revert if try to set new Token Address from non admin caller
✓ Should revert if try to add vesting from non admin caller
✓ Should revert if try to remove vesting from non admin caller
✓ Should revert if try to increase liquidity from non admin caller
✓ Should revert if try to decrease liquidity from non admin caller
✓ Should revert if try to add user to vesting from non admin caller
✓ Should revert if try to remove user from vesting from non admin caller
✓ Should revert if try to make airdrop from non admin caller
 # Multiple users in one vesting case
✓ User which was overridden cannot be added again
✓ User which was overridden still can claim tokens (44ms)

FILE	% STMTS	% BRANCH	% FUNCS
PlatformToken.sol	100	92.5	100
EIP3009.sol	100	100	100
PlatformVesting.sol	100	94.9	100

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by the Propchain team

As a part of our work assisting Propchain in verifying the correctness of their contract code, our team has checked the complete set of tests prepared by the Propchain team.

We need to mention that the original code has a significant original coverage with testing scenarios provided by the Propchain team. All of them were also carefully checked by the team of auditors.

BulkGiveaway

- ✓ Adding user to giveaway by non-admin fails
- ✓ User is not winner by default
- ✓ Adding user to giveaway by admin succeeds
- ✓ Executing giveaway by non-admin fails
- ✓ Executing giveaway with insufficient balance fails
- ✓ Removing user from giveaway by non-admin fails
- ✓ Removing user from giveaway as admin succeeds
- ✓ Executing giveaway succeeds (46ms)
- ✓ Executing giveaway succeeds for multiple recipients (77ms)
- ✓ Test bulk add winners (67ms)
- ✓ Removing user keeps track of correct order (93ms)
- ✓ Test mass adding via bulk add (1684ms)

LPStaking Test Suite

stake()

- ✓ Should be able to Stake
- ✓ Should approve before staking
- ✓ _amount should not be 0
- ✓ Should be able to get the stake info
- ✓ Each stake should have the different stake ID
- ✓ Should not be able to stake when contract is paused

unstake()

- ✓ Should be able to unstake the staked amount
- ✓ Should be a valid stakeId
- ✓ Should be valid signature
- ✓ Should claim the applicable rewards
- ✓ Should not use the already used signature
- ✓ Should not allow to unstake more than once for stakeId

claimRewards()

- ✓ Should be able to claim rewards (67ms)

✓ Should be a valid stake ID
✓ Should be valid signature
✓ Should wait for the cooling period to claim (47ms)

withdrawLiquidity()

✓ Should be able to withdraw the remaining liquidity
✓ Only paltform admin can withdraw liquidity
✓ Target address should not be zero
✓ Amount should not be zero
✓ Can not withdraw more than available balance

setLPAddress()

✓ Should be able to update the address
✓ Should not have any active stakes
✓ Can not be changed when contract is active
✓ Address should not be zero
✓ Only Platform Admin can Update

setUtilityAddress()

✓ Should be able to update the address
✓ Address should not be zero
✓ Only Platform Admin can Update

setCoolingPeriod()

✓ Should be able to update the cooling period of claim
✓ Only the platform admin can update
✓ Value should not be 0

PlatformAdminPanel

✓ Add admin
✓ Rem admin
✓ Change root admin

PlatformTokenPriceProvider

✓ Usd amount
✓ Token amount

PlatformVesting

✓ Add user to vesting
✓ Claim tge (47ms)
✓ Remove user from vesting
✓ Claim before cliff fails
✓ Claim after cliff succeeds for participants
✓ Claim for re-added member can claim next tick (51ms)
✓ Users removed for multiple ticks can claim skipped ticks after re-add
✓ Users being added late can claim all missed events (TGE + ticks) on first claim
✓ Decrease liquidity
✓ No claim possible after removing vesting
✓ Additional liquidity decrease possible after removing vesting

PlatformVestingAndStaking

✓ Add user to vesting
✓ Claim tge (45ms)
✓ Remove user from vesting
✓ Claim before cliff fails
✓ Claim after cliff succeeds for participants
✓ Claim for re-added member can claim next tick (43ms)
✓ Users removed for multiple ticks can claim skipped ticks after re-add
✓ Users being added late can claim all missed events (TGE + ticks) on first claim
✓ Decrease liquidity
✓ No claim possible after removing vesting
✓ Additional liquidity decrease possible after removing vesting

FILE	% STMTS	% BRANCH	% FUNCS
PlatformToken.sol	41.67	22.5	37.5
EIP3009.sol	0	0	0
PlatformVesting.sol	91.96	53.06	85

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

zokyo