TokensFarm

TOKENSFARM

SMART CONTRACT AUDIT

zokyo

January 11th 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.

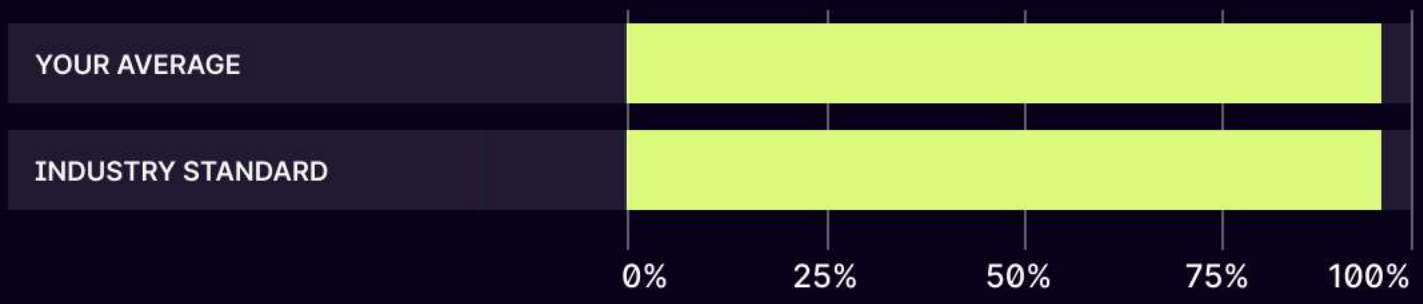
TECHNICAL SUMMARY

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

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

Contract Status

Testable Code



95% of the code is testable, which corresponds the standard of 95%.

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 TokensFarm 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 TokensFarm repository: https://github.com/Tokensfarm/tokensfarm-contracts

Initial commit: 43e0e617143d209dacb6f8c71e31c53434a08ef4

Final commit: 59df44392e1b7b80d713b85b8f6a747a2a2f7cb9

Third iteration final commit: 944d561b63411241a49f25086858a9dab280c1c9
Fourth iteration final commit: c3c5fecb7afcf3a8bcb78cb5898ef932472e7e8a

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

- PerpetualTokensFarmSDK.sol
- PerpetualTokensFarm.sol
- TokensFarmSDK.sol
- VestingFarmFactory.sol
- LinearVestingFarm.sol
- MerkleLinearVesting.sol
- TokensFarm.sol
- TokensFarmFactory.sol
- TokensFarmSDKFactory.sol
- IterativeVestingFarm.sol
- MerkleIterativeVesting.sol
- ReferralDashboard.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 TokensFarm 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

During the audit, Zokyo Security checked the whole set of contracts within the scope. The contracts consist of staking farm contracts, SDK version of the contracts, and factories for creating and managing instances of the staking farm contracts.

The goal of the audit was to ensure the correctness of the staking and reward mechanism, ensure the security of users' funds, validate the contract code against the list of common security vulnerabilities, and ensure that Solidity best practices are applied to reduce gas expenditure.

There were several high- and medium-severity issues found, as well as some informational ones. The issues were connected with the correctness of the work with Ether, creating and withdrawing stakes. Other issues were connected with the necessity of adding extra validations, gas optimization, and logic clarifications. Nevertheless, all the issues were successfully resolved or verified by the TokensFarm team. After all the fixes were implemented, the contracts passed all security tests.

It should be mentioned that all the contracts are upgradable, which means that the admin of the contract can upgrade their logic at any time.

The overall security of the contracts is high enough. The TokensFarm team has prepared a solid set of tests to ensure the correctness of the contract logic. Zokyo Security has prepared our own set of unit tests as well to validate crucial business logic scenarios. It should also be mentioned that due to the complexity and the size of the contracts, they lack readbility. We recommend TokensFarm to prepare a detailed documentation on the logic of the contracts.

During the second iteration of the audit, there were some additional fixes performed by the TokensFarm team. For example, a wrong address of the user was verified in the finalizeDeposit() function. This issue occurred because some variables have similar naming, e.g. `user` and `_user`. Nevertheless, the issue was fixed and the auditors verified its correctness.

On the third iteration of the audit, the TokensFarm team presented migration functionality and some minor upgrades. The migration functionality allows protocol admins to migrate stakes from the previous epoch to the current epoch so that users continue to receive rewards. The team of auditors re-ran the developed tests suite, added new tests to check the migration mechanism, and did an extensive review of both new functionality and all interactions with the original functionality. The auditors found several issues connected with extra gas expenditure, incorrectly implemented epoch start checks, and a possible denial of service. It needs to be mentioned that the code has become a bit tangled after adding migrations and optimizing deposit functions. Yet, the main reason is keeping the contracts size acceptable for EVM, so there is no security influence. The auditors verified the logic of the migration and rewards update with the TokensFarm team.

Executive Summary

During the fourth audit iteration, the auditors have checked all the changes in the contracts from the previous scope and audited a new contract, ReferralDashboards. No critical issues were found. There were 2 high-, 2 medium-severity, and several information issues detected in the new contract. The first high issue was connected to the ability of the owner to set a large minimum stake period. The TokensFarm team has verified that it is unnecessary to add a limitation on a period that can be set. Another high-severity issue was found in the ReferralDashboards smart contract and was connected to the correctness of the changing info about the referral. The issue was successfully fixed by the Tokensfarm team. Medium issues were connected to the safety of the ERC20 transfer and an unnecessary payable modifier. Both of them were fixed as well. A complete list of all the detected issues can be seen in the Complete Analysis chapter of this report.

The overall security of the protocol is still high enough. The contracts have a good natspec documentation, though there are some informational issues connected to the optimization of the contracts, which should be resolved. In order to ensure high security of the contracts, Zokyo Security has also updated all written unit tests and prepared new ones.

PROTOCOL OVERVIEW

The TokensFarm protocol is a staking protocol that allows users to lock staking tokens and receive reward tokens over time. The protocol consists of two versions of staking: original and SDK staking. Both original and SDK contracts are the composites of TokensFarm, PerpetualTokensFarm, and TokensFarmFactory contracts.

The main difference between original and SDK contracts is that SDK contracts don't actually store any staking tokens transferred from users to the contract's balance. Instead, all the crucial functions such as deposit(), makeDepositRequest(), finalizeDeposit(), noticeReducedStake(), and noticeReducedStakeWithoutStakeId() can be called only by the Contract Admin for users. This way, there is no risk of stealing staking tokens from the contract's balance.

The TokensFarm contract allows users to deposit their staking tokens and start earning rewards on their stakes. Each deposit is divided into stakes. In case there is a warm-up period, the user has to make a deposit request and finalize it after the warm-up period is over. Users can withdraw their stake at any time in case an early withdrawal is allowed. In this case, users still have to wait for a minimal time to stake in order to receive the earned rewards. Otherwise, the rewards can be burnt or redistributed based on the contract's options. There is a single reward period, during which users can deposit and earn rewards. While the reward period is still on, it can be extended by funding or redistributing more rewards.

The PerpetualTokensFarm contract is similar to the TokensFarm contract. The only difference is that the reward period is separated into epochs. Users can deposit and earn rewards only in the current epoch. Once the epoch is over, the owner of the contract can create a new epoch.

The SDK version of the contracts also allows users to earn rewards based on their stakes. The reward period mechanism is similar to the original version of the contracts. The main difference is that users don't have to transfer their tokens to the staking contract. Instead, the admin creates and withdraws stakes for users.

The newly added migration functionality allows stakes migration from the finished epoch to the current epoch for all users.

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

CRITICAL-1 | RESOLVED

New epoch can't be started.
PerpetualTokensFarmSDK.sol
In order to start a new epoch, `allStakesAreMigrated` should be equal to true. (startNewEpoch(), line 576). However, in the initialize() function, it is set as false (Line 282). Due to this, epoch 1 can't be started. Also, during the execution of the startNewEpoch() function, `allStakesAreMigrated` should be set as false (Line 617).
The issue was found during the 3rd iteration of the audit in the newly added migration functionality.

Recommendation:
Set `allStakesAreMigrated` as true during initializing and as false during startNewEpoch() execution.

Post-audit:
The issue was found simultaneously with the TokensFarm team, thus by the time of the reporting, the issue was meant to be resolved. Nevertheless, the team of auditors has checked the solution as well.

HIGH-1 | RESOLVED

Usage of msg.value in the loop.
PerpetualTokensFarmSDK.sol, function noticeReducedStakeWithoutStakeId(), line 1433. TokensFarmSDK.sol, function noticeReducedStakeWithoutStakeId(), line 1451. The _payoutRewardsAndUpdateState() function is executed multiple times in the loop. There is a call of the _erc20Transfer() function in this function, where msg.value is used and added to the `totalFeeCollectedETH` storage variable. This way, the same msg.value is added multiple times. This can potentially prevent collecting ETH fees. Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#msgvalue-inside-a-loop

Recommendation:
Avoid using msg.value in the loop. Add msg.value only once to `totalFeeCollectedETH`.

HIGH-2 | RESOLVED

The value in the `idInList` mapping can be wrong.
TokensFarm.sol: function finaliseDeposit(), line 1103. TokensFarmSDK.sol: function finaliseDeposit(), line 1061. PerpetualTokensFarmSDK.sol: function finaliseDeposit(), line 1114. PerpetualTokensFarm.sol: function finaliseDeposit(), line 1203. In case the user doesn't have any deposit requests, they get removed from the `waitingList`, and their ID in the waiting list is given to the last user in the `waitingList`. However, the value in the `idInList` mapping is not updated for `lastUserInWaitingListArray`. Due to this, the finalization of the request for `lastUserInWaitingListArray` can be blocked.

Recommendation:
Update the value in the `idInList` mapping for `lastUserInWaitingListArray` with `deletedUserId`. Delete the value from the `idInList` mapping for the user. Take into account that `lastUserInWaitingListArray` can be equal to `user` in case there is only one address in the waiting list.

HIGH-3 | RESOLVED

Users can withdraw the same stake more than once with the emergency withdrawal function.

TokensFarm.sol: function emergencyWithdraw().
PerpetualTokensFarm.sol: function emergencyWithdraw(). Users can execute the emergencyWithdraw() function with the same stake, even when the amount of the stake is `0`. Due to this, the `participants` array is updated every time, deleting the first user from the array. This way, users can delete all users from the `participants` array and block all withdrawal functions to them.

Recommendation:
Do not let users conduct an emergency withdrawal of the same stake more than once.

HIGH-4 | RESOLVED

The wrong stake amount is stored.

PerpetualTokensFarm.sol: function _deposit(), line 1083.
The `_amount` parameter is stored in `stake.amount`. In case there is a stake fee, a greater amount would be stored instead of the value after taking the fee.

Recommendation:
Store `stakedAmount` in `stake.amount`.

User's stake can be reduced before stakes are finalized.

PerpetualTokensFarmSDK.sol

TokensFarmSDK.sol

During the execution of the makeDepositRequest() function, the value in the `totalActiveStakeAmount[user]` mapping is updated. This value is used in the noticeReducedStakeWithoutStakeId() function to verify that the user has a sufficient stake amount. In case this function is called before the finalization of the stake, the user would be able to withdraw their non-finalized stake. Even though there is a validation that the stake is finalized (TokensFarmSDK.sol, line 1442), the issue still can occur when the user has only one stake that is not yet finalized. The issue is marked as medium since only the owner or the contract admin can execute these functions.

Post-audit:

The deposits that are not finalized can still be processed, potentially breaking the `totalDeposits` variable and user's `totalActiveStakeAmount`.

Consider such a scenario:

1) The user has performed 3 different stakes with the following amounts:
   a) Stake `0` with amount = 1 token.
   b) Stake `1` with amount = 2 token.
   c) Stake `2` with amount = 3 token.
2) Stake `0` and `2` are finalized, leaving stake `1` not finished.
   a) `totalDeposits` is equal 1 + 3 = 4. (Since only stakes `0` and `2` are finalized.)
3) The user withdraws the amount of 3 tokens with the noticeReducedStakeWithoutStakeId() function. In this case:
   a) the amount of stake `0` will be equal 0.
   b) the amount of stake `1` (not finished) will also be equal 0.
   c) the amount of stake `2` will still be equal to 3.
   d) `totalDeposits` will be equal to 1 (Despite the fact that there is a finalized stake `2` with the amount of 3).

After this, the finalization of stake `1` will increase the `totalDeposits` despite the fact that the amount of stake `1` is equal to 0 (Because stakedAmount is also stored in the deposit request structure). And once `totalDeposits` is increased, the user will also be able to finalize stake `2`.

Post-audit 2:
A validation was added: in case `lastFinalisedStake[user]` > 0, stakeId to finalize should be equal to `lastFinalisedStake[user] + 1`. Yet, there is still a case, when the user can finalize the stake with the ID `0`, then the stake with the ID `2` and won't be able to finalize the stake with the ID `1`. Also, the user can finalize any stake at the very first time, thus not start with the stake `0`.

Post-audit 3:
Stakes can now be finalized only in the correct order.

MEDIUM-2 | RESOLVED

Possible Denial-of-Service
PerpetualTokensFarm.sol: function migrateUserStakes(), line 539.
PerpetualTokensFarmSDK.sol: function migrateUserStakes(), line 538.
There is a "require" statement that validates that the epoch ID of the user's stake strictly equals to `epochId` - 1. Yet, there might be a case when the user has a stake, whose epoch ID is lower than `epochId` - 1. Thus, the migration would be blocked until the user withdraws such a stake.

The issue was detected during the third iteration of the audit in the newly added migration functionality.

Recommendation:
Consider changing "require" to "if" so that the migration is not blocked in such a scenario.

LOW-1 | RESOLVED

Unnecessary validation.
PerpetualTokensFarm.sol: function finaliseDeposit(), line 1156.
TokensFarm.sol: function finaliseDeposit(), line 1058.
PerpetualTokensFarmSDK.sol: function finaliseDeposit(), line 1076.

In PerpetualTokensFarm.sol and TokensFarm.sol, the `if` statement will never return false since if the caller is not the owner, the transaction will revert to the `onlyOwner` modifier. In the PerpetualTokensFarmSDK.sol contract, the function can be executed either by the owner or the contract admin. In case the function is called by the contract admin, the value of the local variable won't be assigned to the `_user` parameter and will be equal to msg.sender (which is the contract admin).

Recommendation:
Remove the unnecessary validation.

Post-audit:
In PerpetualTokensFarmSDK.sol, the validation was removed. In other contracts, functions can now be called by the user, so the validation is necessary now.

LOW-2 | VERIFIED

Internal functions are never used.
TokensFarmFactory.sol, TokensFarmSDKFactory.sol: functions _getFarmArray(), _getFarmImplementation().
Functions are internal and are not used within the contract. However, they increase the size of the contract.

Recommendation:
Remove the unused functions.

Post-audit.
Functions will be used in the future updates of the contracts.

Reduce without stake ID might revert in case not the first stake was reduced with the ID.
TokensFarmSDK.sol, PerpetualTokensFarmSDK.sol: function noticeReducedStakeWithoutStakeId().
In case the user has more than one stake and withdraws any stake but the first one, `lastStakeConsumed[_user]` will be equal to this stake. When the user decides to withdraw stakes using the noticeReducedStakeWithoutStakeId() function, it might revert in line 1463 since all the stakes before `lastStakeConsumed[_user]` will be skipped. The issue is marked as low since the user can still withdraw their stakes separately with the noticeReducedStake() function.

Recommendation:
Make sure that noticeReducedStakeWithoutStakeId() processes all actual stakes.

Post-audit.
The validation was added to ensure that`stakeId` is less or equal to `lastStakeConsumed[_user]` and greater or equal to `lastStakeConsumed[_user] + 1`.
However, there are cases now when the user cannot withdraw all of their stake. For example:
1) The user has 4 stakes.
2) The user withdraws a part of stake `0`.
3) The user withdraws their stake `1`.
4) The user withdraws their stakes `2` and `3` without stake ID.
5) The user can't withdraw the rest of stake `0` due to "Must consume the next stake, can not skip".

Post-audit 2.
It is now verified that stakes can be reduced only in the correct order.

LOW-4 | RESOLVED

Enormous gas expenditure
PerpetualTokensFarm.sol, PerpetualTokensFarmSDK.sol: function migrateUserStakes().
The lastStakeMigrated storage variable is written on every step of the double cycle, which leads to an extremely high gas expenditure.
Also, it is especially crucial since the variable is not used in the contracts at all.

The issue was detected during the third iteration of the audit in the newly added migration functionality.

Recommendation:
Review the logic of the lastStakeMigrated usage, add conditions to provide storage updates just once for the last ID. E.g., consider adding a couple of conditions to write only the last ID in the double cycle, or consider moving the creation of the storage pointer for StakeInfo out of the cycle.

Post-audit.
The variable was removed from the contracts.

INFO-1 | RESOLVED

The visibility of the variables is not explicitly marked.
PerpetualTokensFarm.sol: `idInList`.
PerpetualTokensFarmSDK.sol: `idInList`.
TokensFarm.sol: `idInList`.
TokensFarmSDK.sol: `idInList`.
For the improved code readability, it is recommended to explicitly mark the visibility for all storage variables and constants.

Recommendation:
Mark the visibility of all variables and constants in the contracts.

INFO-2 | VERIFIED

Storage constants should be used.
PerpetualTokensFarm.sol: lines 244, 245, 535, 536, 637, 654, 1066, 1562.
PerpetualTokensFarmSDK.sol: lines 240, 535, 616, 1559, 1387.
TokensFarm.sol: lines 233, 234, 525, 545, 987, 1188, 1558.
TokensFarmSDK.sol: lines 243, 530, 1588, 1396.

Number 40 and 100 should be moved to a separate storage constant.

Recommendation:
Move the numbers used in the code to storage constants.

From the client:
In order not to exceed the contract size limit, constants won't be used.

INFO-3 | RESOLVED

Unnecessary adding of 0.
TokensFarm.sol: function deposit(), line 1214.
TokensFarmSDK.sol: function deposit(), line 1159.
Adding `warmupPeriod` in both cases has no effect since it was previously checked that `warmupPeriod` is equal to 0.

Recommendation:
Remove adding `warmupPeriod`.

INFO-4 | RESOLVED

Finalizing pending deposit requests can be blocked.
TokensFarm.sol: function finaliseDeposit(), line 1063.
TokensFarmSDK.sol: function finaliseDeposit(), line 1026.
PerpetualTokensFarmSDK.sol: function finaliseDeposit(), line 1081.
PerpetualTokensFarm.sol: function finaliseDeposit(), line 1174.
There is a validation that `warmupPeriod` is not equal to `0` in these functions. However, in case the owner sets `warmupPeriod` as `0` with the setWarmup() function, all pending deposit requests will be blocked, preventing users from depositing and withdrawing their funds. The issue is marked as informational since only the owner can change `warmupPeriod`.

Recommendation:
Verify that users' funds can't get blocked due to the changes of the warmup period.

INFO-5 | RESOLVED

View function can be optimized.
TokensFarm.sol: function getAllPendingStakes(), line 690.
TokensFarmSDK.sol: function getAllPendingStakes(), line 649.
PerpetualTokensFarm.sol: function getAllPendingStakes(), line 777.
The function performs iteration through the whole participants array, which may consume more gas than allowed per transaction. In order to reduce gas spendings, the `waitingList` array can be used, which already contains all the users who have current deposit requests.

Recommendation:
Use the waitingList array instead of `participants`.

Redistributing rewards calculates more total rewards than there are on the contract's balance.

TokensFarm.sol: function withdraw(), line 1425.
TokensFarmSDK.sol: function _payoutRewardsAndUpdateState(), line 1246.
PerpetualTokensFarm.sol: function withdraw(), line 1437.
PerpetualTokensFarmSDK.sol: function _payoutRewardsAndUpdateState(), line 1245.

In case user's pending reward is to be redistributed, the _fundInternal() function is called, which increases the `totalRewards` storage variable. However, the actual reward balance doesn't increase since the same reward token is funded. This way, there can be a situation when there are not enough rewards to pay to users. The issue is marked as informational since it doesn't prevent the withdrawal of stakes but can prevent collecting rewards for users.

Recommendation:
Update `totalRewards` correctly in case of the redistribution of pending rewards.

From the client:
The `totalRewards` variable doesn't affect any calculations. It is an intended functionality to increase this variable during every redistribution.

INFO-7 | VERIFIED

The `start` parameter is unnecessary
PerpetualTokensFarm.sol, PerpetualTokensFarmSDK.sol: function migrateUserStakes(). Providing this parameter is unnecessary since it must always be equal to the `lastUserMigrated` storage variable. In order to simplify the execution of this function, it is recommended to remove this parameter.

The issue was found during the third iteration of the audit in the newly added migration functionality.

Recommendation:
Remove the `start` parameter and use `lastUserMigrated` instead.

From the client:
The TokensFarm team prefers current functionality because the backend provides the start and end so it will be easier for BE to determine the failure in case that tx reverts.

INFO-8 | VERIFIED

The owner can withdraw reward tokens
PerpetualTokensFarm.sol, TokensFarm.sol: function withdrawTokensIfStuck().
There is no validation that the provided `_erc20` is not equal to `erc20` (Reward token). Thus, in case the admin has withdrawn reward tokens, there might be not enough rewards for users to pay out. The issue is marked as informational since stakes can still be withdrawn with the emergencyWithdraw() function, though the capability of the admin to withdraw rewards should still be mentioned in the report.
The issue was detected during the third iteration of the audit in the newly added migration functionality.

Recommendation:
Restrict the admin from withdrawing reward tokens.

From the client:
This functionality is meant for the reward token because there is a case where it can be funded more than needed, so it has to remain as is.

INFO-9 | RESOLVED

New epoch can be started while migration is in progress

PerpetualTokensFarm.sol, PerpetualTokensFarmSDK.sol. It is possible for the owner to start a new epoch while the migration is in progress. In this case, the previous migration won't be finished correctly and the new migration might not be started in the future. The issue is marked as informational since only the owner can execute these functions, and in this scenario, users will still be able to withdraw their stakes. The issue was found during the third iteration of the audit in the newly added migration functionality.

Recommendation:

Verify that a new epoch can't be started while migration is in progress.

INFO-10 | VERIFIED

The pool is updated twice and extra operations are performed

PerpetualTokensFarmSDK.sol, migrateUserStakes()

The pool is updated directly in the function and through the call of _payoutRewardsAndUpdateState() (with the _amount = `0` parameter). That is unnecessary gas spending.

The same applies to all math operations in _payoutRewardsAndUpdateState() (in the if(_unStake) statement), where a lot of operations are performed with re-writing storage after the operations with `0` _amount.

The issue was detected during the third iteration of the audit in the newly added migration functionality.

Recommendation:

Review the logic of the pool update to check if there are no side effects of the double pool updates and if additional gas spending is necessary.

Post-audit

1) It was confirmed by the TokensFarm team that the fix does not affect methods with _payoutRewardsAndUpdateState() calls.
2) The necessity of the updatePool() within the double migration cycle was confirmed.
3) The correctness of the logic was confirmed for the PerpetualTokensFarm.sol contract.

4TH ITERATION COMPLETE ANALYSIS

HIGH-1 | VERIFIED

The owner can block funds.
PerpetetualTokensFarm.sol
The owner can set a large duration value with the the setMinTimeToStake() function, in which case users' funds will be stuck. In order to avoid this, there should be a limitation in the function preventing the owner from setting a large minimum time to stake.

Recommendation:
Add a limitation for the duration parameter in the setMinTimeToStake() function, e.g. 1 year.

From the client.
In case of a hacker attack, if min time to stake will be set to large values, the owner of the farm will be able to set it back to a normal value.

HIGH-2 | RESOLVED

Total pending amount to withdraw is not updated.
ReferralDashboard.sol: changeReferralInfo().
The value from the `referrer2TotalPendingAmountToWithdraw` mapping indicates the total amount of tokens that a referral can withdraw. However, when changing the information about the referral, the value from the mapping is not updated. In case the amount of the reward was increased, the referral won't be able to withdraw it due to the if() check. The issue is marked as high since even though the admin can still fix it by lowering the amount of the reward for the referral, it still requires additional efforts from the admin and leads to the loss of funds by the referral.

Recommendation:
Update the value from the `referrer2TotalPendingAmountToWithdraw` mapping accordingly in the changeReferralInfo() function.

Post-audit.
The value in the mapping has been updated.

MEDIUM-1 | RESOLVED

Ether could be stuck on contract.
ReferralDashboard.sol: withdraw().
The withdraw() function is payable, but there is no interaction with msg.value in it. Thus, users can accidentally send Ether in msg.value. In this case, Ether will be stuck on the contract.

Recommendation.
Remove payable from the function.

MEDIUM-2 | RESOLVED

safeTransfer() should be used.
PerpetetrualTokensFarm.sol, PerpetualTokensFarmSDK.sol, TokensFarm.sol, TokensFarmSDK.sol
In these contracts, tokens are used with a regular transfer() method from the OpenZeppeling IERC20 interface. Though, in general, the ERC20 token may have no return value for these methods (see USDT implementation) and lead to the failure of calls and contract blocking. In case SafeERC20 is used in the contract for IERC20, the transfer() function should be changed to safeTransfer().

Recommendation.
Replace the transfer() function with safeTransfer().

INFO-1 | UNRESOLVED

Unnecessary variable creation.
In the case of using interfaces to invoke its functions, there is no need to create a variable for it. Instead, use interfaces directly like IERC20(address).transfer(). The following lines should be changed:

1) IterativeVestingFarm.sol: lines 864-866.
2) LinearVestingFarm.sol: lines 793-795.
3) MerkleIterativeVesting.sol: lines 1112-1114.
4) TokensFarmFactory.sol: lines 196-197, 286-288, 290-291, 452-453, 474-475, 496-497, 518-519, 540-541, 562-563, 585-586, 608-609, 760-763, 808-809, 812-814, 1226-1227
5) TokensFarmSDKFactory.sol: lines 181-185, 289-291, 293-294, 461-462, 478-479, 628-631, 671-672, 675-677, 708-710, 756-758, 777-778, 796-797, 815-816, 834-835, 853-854, 877-878, 896-897, 914-915, 932-933, 951-952, 983-984, 1001-1002, 1099-1100,

Recommendation:
Use interfaces directly.
From the client. Due to syntax reasons, the current implementation won't be changed.

INFO-2 | VERIFIED

The owner has full contract functionality.
The functions that are critical to the protocol like emergencyWithdraw and pauseFarm, which gives the owner the ability to withdraw tokens from the contract. In case of compromising the private key of the owner, the exploiter has full control of protocol. To prevent this, consider creating roles or address variables, which should be settled in the constructor for different addresses for different functionality like farmController, withdrawAdmin, etc. so that even if the owner's private key is compromised, users' funds are safe.

Recommendation:
Create roles for different contract functionality for different addresses.

From the client.
All farms are deployed by the factory contract, setting the factory as the owner of the farm. All crucial functions are executed by the Congress, which is a multisig wallet.

Gas optimization suggestions.

1) The function can be marked as external.
ReferralDashboard.sol: fund(), getAllReferrerInfo().

2) The memory parameter can be marked as calldata since it is not changed during the function execution.
ReferralDashboard.sol:

- changeReferralInfo(), addNewReferral(): _farmType, _stakeTokenSymbol, _stakeTokenIcon1, _stakeTokenIcon2.

3) MerkleIterativeVesting.sol
Line 421, 448, 901, 910 .... Converting address to address is redundant, since it only increases gas expenditure and has no impact on the values.

Post-audit.

As passing 4 string parameters causes a “Stuck too deep” error, the previous suggestion wasn’t full. In order to keep thefunction gas-optimized and avoid this error, it is recommended to define a struct with 4-string variables and pass this struct as a calldata argument. During unit-testing it was verified that such a struct saves around 120 gas units compared to memory variables.

From the client.

Since backend and frontend are already implemented, the interface function could not be changed.

Post-audit.

Point 1 was fixed only.

INFO-4 | VERIFIED

Name and return parameters mismatch.
PerpetualTokensFarm.sol. The function is named getTotalRewardsLockedUnlocked, but it returns unlocked and locked reward.

Recommendation.
Rename the function or change the order of return parameters.
From the client. Since backend and frontend are already implemented, the function interface could not be changed.

INFO-5 | UNRESOLVED

Unreachable code or unnecessary validations.
1) TokensFarmSDKFactory.sol: deployAndFundTokensFarmSDK(), _fundInternalFarm();
VestingFarmFactory.sol:_fundInternalVestingFarm(), _getFarmArray().

1.1) Validation “List of params is not initialised” (deployAndFundTokensFarmSDK(), line 424) is unreachable because if params are not initialized then the reward token is a zero address and not a Contract. Before that validation with the revert message “List of params is not initialized”, there is another validation for sufficient balance of the reward token for payoffs and there is a call IERC20(rewardToken).balanceOf(). Thus, if rewardToken is a zero address, the function call will revert on the operation not even reaching next validations for parameters.

1.2) Validation “RewardToken address can't be 0×0 address” ” (_fundInternalFarm(), line 279 is unreachable because if params are not initialized, the reward token is a zero address and not a Contract. There are 3 calls of the _fundInternalFarm() method, and before every of these calls, there are calls of the _sufficientFunds() method for checking whether the balance of the reward token is sufficient for payoffs and there is a call IERC20(rewardToken).balanceOf(). Thus, if rewardToken is a zero address, the function call will revert even before calling the _fundInternalFarm method, so the revert on “RewardToken address can't be 0×0 address” will never happen.

1.3) Validation “RewardToken address can't be 0×0 address” in VestingTokensFarm
” (_fundInternalVestingFarm(), line 246 is unreachable because if given as a function parameter reward token is being zero address and not being Contract, then the method call will revert before that operation. There are 3 calls of the _fundInternalVestingFarm() method, and before every of these calls, there are calls of the _sufficientFunds() method for checking whether the balance of the reward token is sufficient for payoffs and there is a call IERC20(rewardToken).balanceOf(). Thus, if rewardToken is zero address, the function call will revert even before calling the _fundInternalVestingFarm method, so the revert on “RewardToken address can't be 0×0 address” will never happen.

1.4) Validation "Farm index is out of range” in VestingTokensFarm ” (_getFarmArray(), line 195 will never revert since before the function calls to _getFarmArray(), there are exact checks for the index not to be out of range. If it is, is will revert before the call to _getFarmArray() (This happens in getLastDeployedFarm(), line 990; getDeployedFarms(), line 1025)

2) ReferalDashboard.sol, withdraw(), line 551.

Since the reward amount can’t be 0, it is redundant to validate that the value from mapping `referral2NextReferralToBePaid` is greater than 0. In case the value from mapping `referral2NextReferralToBePaid` equals 0, this means that all referrals are paid and the function will revert at the first require.

3) There is no reason to check the uint256 value for positivity because uint256 by default is equal to 0 and cannot be lower than 0. Such validation will never revert. The following contracts contain unnecessary validations of uint256 values:

- MerkleIterativeVesting.sol: setMerkleRoot(), Line 282
- MerkleLinearVesting.sol: setMerkleRoot(), Line 242
- TokensFarmSDKFactory.sol: withdrawStuckTokensFromFactory(), Line 1093
- VestingFarmFactory.sol: _fundInternalVestingFarm(), Line 250; withdrawStuckTokensFromFactory(), Line 839;
- TokensFarm.sol: setTheRestOfParameters(), Line 295, 299; setMinTimeToStake(), Line 507; setStakeFeePercent(), Line 542 (part with “_stakeFeePercent >= 0”); setFlatFeeAmountDeposit(), Line 583; setFlatFeeAmountWithdraw(), Line 603; setCoolDown(), Line 638; setWarmup(), Line 658;

- VestingFarmFactory.sol: isEnumInRange(), Line 176 (part with "_farmIndex >= 0");
- PerpetualTokensFarmSDK.sol: startNewEpoch(), Line 585 (part with "_rewardFeePercent >= 0"); setRewardFeePercent(), Line 669 (part with "_rewardFeePercent >= 0")

4) Validation of initializable values.
- TokensFarm.sol: _addPool(), Line 372.
- PerpetualTokensFarm.sol: _addPool(), Line 392.

Validating that `tokenStaked` is equal to zero address is redundant since this variable cannot be set before this function is called as it is used only in the initializer.
- TokensFarm.sol: makeWithdrawRequest(), Line 1222; withdraw(): Lines 1304, 1351.

Validating that contract is fully initialized is unnecessary in these functions since in order to call them, the deposit function should be called first to validate that the contract is initialized.

Recommendation.
Remove unreachable validations.

Post-audit.
Point 2 was fixed only.

INFO-6 | VERIFIED

Methods for setting parameters do not have validation checks.
TokensFarmSDKFactory.sol: sendParameters(), prepareParameters()

1.1) The sendParameters() method does not have any validation checks despite the fact that this method includes setting important parameters such as staking token, reward token, reward per second, etc.
1.2) The prepareParameters() method does not have any validation checks either.

Recommendation.
Add validations for important parameters.

From the client. Since these parameters are already validated on the farm contracts, validating it on factory is unnecessary.

INFO-7 | VERIFIED

Methods are not used.
TokensFarmSDKFactory.sol: _getFarmArray(), _getFarmImplementation()
TokensFarmFactory.sol: _getFarmAdday(), _getFarmImplementation()
VestingFarmFactory.sol:, _getFarmImplementation()

1.1) _getFarmArray() is declared in the TokensFarmSDKFactory, TokensFarmFactory, and VestingFarmFactory contracts but is only used in the VestingFarmFactory contract.
1.2) _getFarmImplementation() is declared in TokensFarmSDKFactory, TokensFarmFactory, and VestingFarmFactory bun never used in any of these contracts.

Recommendation.
Remove unused internal methods.

INFO-8 | UNRESOLVED

Check that endTime should be greater than startTime can be bypassed.
MerkleIterativeVesting.sol, setStartTime(), line 321.
There is no need to check that _startTime is less than endtime because endTime will be calculated based on _startTime. Check that endTime > _startTime can be bypassed if the function is called twice. In the first call, endTime will be updated with _startTime + duration of vesting, and in the second call, we can set _startTime with a greater value than the original endTime.

Example. Current: startTime = 1, endTime = 6.
We call setStartTime(_startTime = 5). New value: startTime 5, endTime = 10. And now we can call setStartTime(_startTime = 9), which is greater than the initial endTime value. This makes the requirement endTime > _startTime completely pointless.

Recommendation.
Either implement a validation that can't be bypassed or remove this restriction and allow setting the start time regardless of the end time.

	PerpetualTokensFarmSDK.sol	PerpetualTokensFarm.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

	TokensFarmSDK.sol	TokensFarm.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

	TokensFarmFactory.sol	TokensFarmSDKFactory.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

	VestingFarmFactory.sol	IterativeVestingFarm.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

	LinearVestingFarm.sol	MerkleIterativeVesting.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

	MerkleLinearVesting.sol	ReferralDashboard.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

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by Zokyo Security

As a part of our work assisting TokensFarm 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 the review of the TokensFarm contract requirements for details about issuance amounts and how the system handles these.

Contract: PerpetualTokensFarm

Deposit/Request deposit/Finalize deposit request

Deposit

- ✓ Should deposit tokens for user (136ms)
- ✓ Should revert deposit if warmup period is on (50ms)
- ✓ Should revert deposit if farm has ended (58ms)
- ✓ Should update firstDepositAt only at the first deposit (108ms)
- ✓ Should collect flat fee during depositing (62ms)
- ✓ Should revert deposit if msg.value != flatFeeAmountDeposit (53ms)
- ✓ Should collect stake fee (71ms)
- ✓ Should revert if amount == 0

Make deposit request

- ✓ Should make deposit request (72ms)
- ✓ Should revert make deposit request if warmup is off
- ✓ Should revert make deposit request if reward will end after warmup
- ✓ Should update firstDepositAt only once during making a deposit request (78ms)
- ✓ Should collect flat fee during making a deposit request (47ms)
- ✓ Should revert make deposit request if msg.value != flatFeeAmountDeposit
- ✓ Should collect stake fee (48ms)
- ✓ Should revert if amount equals 0

Finalize deposit

- ✓ Should finalize user s deposit request (78ms)
- ✓ Should revert finalizing deposit request if warmup period is 0 (47ms)
- ✓ Should revert finalizing deposit request if warmup period is not yet finished (41ms)
- ✓ Should finalize one of user s deposit requests (160ms)
- ✓ Should remove user from waiting list (198ms)
- ✓ Should revert finalizing if user has not deposit requests (45ms)

Withdraw/Make withdraw request/Emergency withdraw
Withdraw
✓ Should withdraw (107ms)
✓ Should revert if minimal time to stake is not respected (45ms)
✓ Should burn user's pending if minimal time to stake is not respected and penalty should be burnt (92ms)
✓ Should redistribute user's pending if minimal time to stake is not respected and penalty should be redistributed (79ms)
✓ Should burn user's pending if staking is ended and penalty should be redistributed (97ms)
✓ Should cover ETH commission if flat fee is allowed (71ms)
✓ Should revert if msg.value != flatFeeAmountWithdraw when flat fee is allowed (67ms)
✓ Should collect rewards without fee (73ms)
✓ Should revert withdraw if amount is greater than staked amount (41ms)
Make withdraw request
✓ Should make withdrawal request (82ms)
✓ Should revert make withdrawal request if whole stake amount is already withdrawn (69ms)
✓ Should revert make withdrawal request if cooldown is 0 (43ms)
✓ Should revert make withdrawal request if stake is not finalized (46ms)
✓ Should revert make withdrawal request if minimal time to stake was not respected (51ms)
✓ Should revert make withdrawal request if amount is greater than stake amount (51ms)
✓ Should make withdrawal request for different stakes (103ms)
✓ Should finalize withdrawal request (95ms)
✓ Should revert finalizing withdrawal request if request wasn't made for provided id (136ms)
✓ Should not finalize withdrawal request if cooldown has not passed yet (67ms)
Emergency withdraw
✓ Should withdraw in case of emergency (142ms)
✓ Should revert emergency withdraw if minimal time to stake is not respected (40ms)
Perpetual functionality
✓ Should deposit and claim rewards in correct epoch (197ms)
✓ Should revert start new epoch if current epoch is not ended
✓ Should revert start new epoch if start time is less than block.timestamp
✓ Should revert start new epoch if reward fee percent is greater than 100

Contract: PerpetualTokensFarm

Deposit/Request deposit/finalize deposit
Deposit
✓ Should deposit tokens for user (53ms)
✓ Should revert deposit if warmup period is on
✓ Should revert deposit if farm has ended
✓ Should update firstDepositAt only at the first deposit (41ms)
✓ Should collect flat fee during depositing
✓ Should revert deposit if msg.value != flatFeeAmountDeposit

✓ Should not update ATH stake if deposited amount is less than ATH stake amount (43ms)
✓ Should revert if epoch hasn't started yet

Request deposit

✓ Should make deposit request
✓ Should revert make deposit request if warmup is off
✓ Should revert make deposit request if reward will end after warmup
✓ Should update firstDepositAt only once during making a deposit request
✓ Should collect flat fee during making a deposit request
✓ Should revert make deposit request if msg.value != flatFeeAmountDeposit

Finalize deposit

✓ Should finalize user's deposit request (50ms)
✓ Should revert finalizing deposit request if warmup period is 0
✓ Should revert finalizing deposit request if warmup period is not yet finished
✓ Should finalize one of user's deposit requests (104ms)
✓ Should remove user from waiting list (127ms)
✓ Should revert finalizing if user has not deposit requests

Notice reduced stake with stake id

✓ Should notice reduced amount (87ms)
✓ Should revert notice reduced stake if deposit request is not finalized
✓ Should revert notice reduced stake if withdraw amount > stake amount
✓ Should revert if minimal time to stake is not respected
✓ Should burn user's pending if minimal time to stake is not respected and penalty should be burnt (78ms)
✓ Should redistribute user's pending if minimal time to stake is not respected and penalty should be redistributed (62ms)
✓ Should burn user's pending if staking is ended and penalty should be redistributed (90ms)
✓ Should cover ETH commission if flat fee is allowed (52ms)
✓ Should revert if msg.value != flatFeeAmountWithdraw when flat fee is allowed (44ms)

Notice reduce stake without stake id

✓ Should reduce user's stakes amount (81ms)
✓ Should revert if user tries to withdraw more than he deposited
✓ Should reduce stake with id and then the rest of stake without id (161ms)
✓ Should not reduce stakes in not finalized stakes (161ms)
✓ Should revert if user has only one stake and it is not finalized yet
✓ Should withdraw a single user's stake (51ms)

Perpetual functionality

✓ Should deposit and claim rewards in correct epoch (142ms)
✓ Should revert start new epoch if current epoch is not ended
✓ Should revert start new epoch if start time is less than block.timestamp
✓ Should revert start new epoch if reward fee percent is greater than 100

Contract: TokensFarm

Deposit/Request deposit/Finalize deposit request

Deposit

- ✓ Should deposit tokens for user (71ms)
- ✓ Should revert deposit if warmup period is on
- ✓ Should revert deposit if farm has ended
- ✓ Should update firstDepositAt only at the first deposit (63ms)
- ✓ Should collect flat fee during depositing (46ms)
- ✓ Should revert deposit if msg.value != flatFeeAmountDeposit
- ✓ Should collect stake fee (45ms)
- ✓ Should revert if amount == 0

Make deposit request

- ✓ Should make deposit request 53ms)
- ✓ Should revert make deposit request if warmup is off
- ✓ Should revert make deposit request if reward will end after warmup
- ✓ Should update firstDepositAt only once during making a deposit request (60ms)
- ✓ Should collect flat fee during making a deposit request (40ms)
- ✓ Should revert make deposit request if msg.value != flatFeeAmountDeposit
- ✓ Should collect stake fee (47ms)
- ✓ Should revert if amount equals 0

Finalize deposit

- ✓ Should finalize user s deposit request (66ms)
- ✓ Should revert finalizing deposit request if warmup period is 0 (54ms)
- ✓ Should revert finalizing deposit request if warmup period is not yet finished (47ms)
- ✓ Should finalize one of user s deposit requests (149ms)
- ✓ Should remove user from waiting list (174ms)
- ✓ Should revert finalizing if user has not deposit requests (41ms)

Withdraw/Make withdraw request/Emergency withdraw

Withdraw

- ✓ Should withdraw (91ms)
- ✓ Should revert if minimal time to stake is not respected (39ms)
- ✓ Should burn user s pending if minimal time to stake is not respected and penalty should be burnt (81ms)
- ✓ Should redistribute user s pending if minimal time to stake is not respected and penalty should be redistributed (72ms)
- ✓ Should burn user s pending if staking is ended and penalty should be redistributed (102ms)
- ✓ Should not pay pendingReward if it is 0 (68ms)
- ✓ Should cover ETH commission if flat fee is allowed (67ms)
- ✓ Should revert if msg.value != flatFeeAmountWithdraw when flat fee is allowed (56ms)
- ✓ Should collect rewards without fee (69ms)

✓ Should revert withdraw if amount is greater than staked amount (38ms)

Make withdraw request

✓ Should make withdrawal request (87ms)
✓ Should revert make withdrawal request if whole stake amount is already withdrawn (76ms)
✓ Should revert make withdrawal request if cooldown is 0 (47ms)
✓ Should revert make withdrawal request if stake is not finalized (49ms)
✓ Should revert make withdrawal request if minimal time to stake was not respected (43ms)
✓ Should revert make withdrawal request if amount is greater than stake amount (56ms)
✓ Should make withdrawal request for different stakes (118ms)
✓ Should finalize withdrawal request (102ms)
✓ Should revert finalizing withdrawal request if request wasn t made for provided id (123ms)
✓ Should not finalize withdrawal request if cooldown has not passed yet (80ms)

Emergency withdraw

✓ Should withdraw in case of emergency (140ms)
✓ Should revert emergency withdraw if minimal time to stake is not respected (38ms)

Contract: TokensFarmSDK

Deposit/Request deposit/Finalize deposit

Deposit

✓ Should deposit tokens for user (53ms)
✓ Should revert deposit if warmup period is on
✓ Should revert deposit if farm has ended
✓ Should update firstDepositAt only at the first deposit (40ms)
✓ Should collect flat fee during depositing
✓ Should revert deposit if msg.value != flatFeeAmountDeposit
✓ Should not update ATH stake if deposited amount is less than ATH stake amount (41ms)

Request deposit

✓ Should make deposit request
✓ Should revert make deposit request if warmup is off
✓ Should revert make deposit request if reward will end after warmup
✓ Should update firstDepositAt only once during making a deposit request
✓ Should collect flat fee during making a deposit request
✓ Should revert make deposit request if msg.value != flatFeeAmountDeposit

Finalize deposit

✓ Should finalize user s deposit request (51ms)
✓ Should revert finalizing deposit request if warmup period is 0
✓ Should revert finalizing deposit request if warmup period is not yet finished
✓ Should finalize one of user s deposit requests (107ms)
✓ Should remove user from waiting list (167ms)
✓ Should revert finalizing if user has not deposit requests

Notice reduced stake with stake id
✓ Should notice reduced amount (87ms)
✓ Should revert notice reduced stake if deposit request is not finalized
✓ Should revert notice reduced stake if withdraw amount > stake amount (40ms)
✓ Should revert if minimal time to stake is not respected
✓ Should burn user's pending if minimal time to stake is not respected and penalty should be burnt (83ms)
✓ Should redistribute user's pending if minimal time to stake is not respected and penalty should be redistributed (61ms)
✓ Should burn user's pending if staking is ended and penalty should be redistributed (81ms)
✓ Should not pay pendingReward if it is 0 (60ms)
✓ Should cover ETH commission if flat fee is allowed (47ms)
✓ Should revert if msg.value != flatFeeAmountWithdraw when flat fee is allowed (48ms)
Notice reduce stake without stake id
✓ Should reduce user's stakes amount (91ms)
✓ Should revert if user tries to withdraw more than he deposited
✓ Should reduce stake with id and then the rest of stake without id (156ms)
✓ Should not reduce stakes in not finalized stakes (162ms)
✓ Should revert if user has only one stake and it is not finalized yet
✓ Should withdraw a single user's stake (54ms)

165 passing (12s)

Tests prepared for the third audit iteration for migration logic testing:

Contract: PerpetualTokensFarm
Migration
✓ Should migrate from one epoch to another (126ms)
✓ Should migrate stakes for several users (480ms)

Contract: PerpetualTokensFarmSDK
Migration
✓ Should migrate from one epoch to another (126ms)
✓ Should migrate stakes for several users (480ms)

Contract: TokensFarm
Withdraw/Make withdraw request/Emergency withdraw
✓ Should withdraw rewards (137ms)

Contract: TokensFarmSDK
Notice reduced stake with stake id
✓ Should withdraw rewards (137ms)

6 passing (3s)

Contract: TokensFarm

Checks

✓ Should withdraw rewards with update before pending calculation all in one transaction (145ms)

Contract: PerpetualTokensFarm

Checks

✓ Should have exact match for pending calculation in one transaction (134ms)

Contract: PerpetualTokensFarm

Initializer and setters

✓ Should revert if start time is in the past
✓ Should revert if try to initialize second time
✓ Should revert if try to set new warmup and cooldown to initialized epoch
✓ Should set new min time to stake
✓ Should set new flat fee
✓ Should set new fee collector

Deposit/Request deposit/Finalize deposit request

Deposit

✓ Should get rewards after deposit (155ms)
✓ Should revert if contract is not initialized
✓ Should revert if epoch is not initialized
✓ Should revert if epoch is not started

Withdraw/Make withdraw request/Emergency withdraw

Withdraw

✓ Should withdraw rewards when flat fee is > 0
✓ Should withdraw rewards after migration to new epoch

Make withdraw request

✓ Should make withdrawal request for different stakes (216ms)
✓ Should finalize withdrawal request (249ms)
✓ Should revert finalizing withdrawal request if request wasn't made for provided id (296ms)
✓ Should not finalize withdrawal request if cooldown has not passed yet (141ms)
✓ Should revert if try to withdraw without stake
✓ Should make withdrawal request in new epoch

Emergency withdraw

✓ Should withdraw in case of emergency in new epoch

Perpetual functionality

✓ Should revert start new epoch if caller is not the owner
✓ Should revert start new epoch if start time is in the past (47ms)
✓ Should revert if old epoch is not initialized
✓ Should revert if all stakes are not migrated

Migration

✓ Should migrate from one epoch to another (258ms)
✓ Should migrate stakes for several users (794ms)

- ✓ Should revert if try to migrate from the wrong user
- ✓ Should revert if there is only one epoch
- ✓ Should revert if end is higher than actual amount of users
- ✓ Should revert if try to migrate before epoch is started

Contract: PerpetualTokensFarmSDK

Initialization and setters

- ✓ Should revert if start time is in the past
- ✓ Should revert if try to initialize rest of parameters second time
- ✓ Should set new min time to stake
- ✓ Should set new reward fee percent
- ✓ Should set new flat fee for deposit
- ✓ Should set new flat fee for withdraw
- ✓ Should set new fee collector
- ✓ Should set new contract admin
- ✓ Should activate / deactivate farm

Deposit/Request deposit/finalize deposit

Deposit

- ✓ Should revert if try to deposit when contract is not fully initialized

Withdraw

- ✓ Should withdraw rewards
- ✓ Should withdraw collected fees
- ✓ Should withdraw collected fees in eth
- ✓ Should withdraw stuck tokens
- ✓ Should revert withdraw stuck tokens if token to withdraw is staking token
- ✓ Should withdraw rewards with penalty included

Migration

- ✓ Should migrate from one epoch to another (206ms)
- ✓ Should migrate stakes for several users (593ms)

Modifiers check

- ✓ Should revert if try to withdrawn without depositing
- ✓ Should revert if try to use admin only function from third party user
- ✓ Should revert if try to use admin or factory only function from third party user

Contract: TokensFarm

Initialization and setters

- ✓ Should revert if reward token is zero address (71ms)
- ✓ Should revert if reward per second token is zero (58ms)
- ✓ Should revert if start time is in the past (67ms)
- ✓ Should revert if stake fee is higher than 100 (62ms)
- ✓ Should revert if reward fee is higher than 100 (63ms)
- ✓ Should revert if stakingToken is address zero (64ms)
- ✓ Should revert if rest of parameters are already set (84ms)
- ✓ Should revert if fee collector is address zero (61ms)

- ✓ Should setMinTimeToStake to new amount
- ✓ Should revert if try to set wrong stake fee percent
- ✓ Should revert if try to set wrong reward fee percent
- ✓ Should set new flat deposit fee
- ✓ Should set new flat withdraw fee
- ✓ Should set new fee collector

Fund

- ✓ Should revert if amount is too small
- ✓ Should revert if staking is over (42ms)

Deposit/Request deposit/Finalize deposit request

Finalize deposit

- ✓ Should revert finalizing deposit request if user didn't deposit before

Withdraw/Make withdraw request/Emergency withdraw

- ✓ Should withdraw rewards without update (461ms)
- ✓ Should withdraw rewards with update before pending calculation (406ms)
- ✓ Should withdraw collected fees
- ✓ Should withdraw collected fees eth
- ✓ Should not withdraw rewards if contract is not initialized completely (108ms)
- ✓ Should withdraw rewards with flat fee withdrawn (263ms)
- ✓ Should revert if minimal time staked is not passed (99ms)
- ✓ Should revert withdraw rewards with flat fee withdrawn if fee is not included (95ms)
- ✓ Should withdraw stuck tokens (86ms)

Withdraw

- ✓ Should withdraw partial amount of stake when there is more than 1 user (176ms)
- ✓ Should burn user's pending if staking is ended and penalty should be redistributed (375ms)

Contract: TokensFarmSDK

Initializer and setters

- ✓ Should revert if start time is in the past
- ✓ Should revert if contract is already initialized
- ✓ Should set new min time to stake
- ✓ Should set new reward fee percent
- ✓ Should set new flat fee for deposit
- ✓ Should set new flat fee for withdraw
- ✓ Should set new fee collector
- ✓ Should set new contract admin
- ✓ Should manipulate pause of farm

Notice reduce stake without stake id

- ✓ Should withdraw rewards (316ms)
- ✓ Should withdraw with penalty
- ✓ Should revert if try to withdraw without paying flat fee
- ✓ Should withdraw without fees and penalty
- ✓ Should withdraw without fees but with penalty

- ✓ Should withdraw rewards for other user
- ✓ Should withdraw rewards with penalty
- ✓ Should revert if try to withdraw rewards without waiting min time to stake
- ✓ Should withdraw collected fee
- ✓ Should withdraw collected fee in eth
- ✓ Should withdraw stuck tokens

Modifier check

- ✓ Should revert if deposit is used when it is nit existed
- ✓ Should revert if admin function is used from third party user

TokensFarmFactory

initialization

- ✓ # initialize (330ms)
- ✓ Check getters after initialization
- ✓ revert when passing zero address as farmImplementation
- ✓ revert when passing zero address as perpetualImplementation
- ✓ revert when passing zero address as proxyAdmin
- ✓ revert when passing zero address as feeCollector

Deployment and funding

- ✓ revert on getLastDeployedSpecificFarm when Params are not set
- ✓ # prepareParameters
- ✓ # getLastDeployedFarm
- ✓ revert on 'Parameters are not loaded' when deployAndFundTokensFarm before initialization
- ✓ # deployAndFundTokensFarm (142ms)
- ✓ revert when timeStart < block.timestamp
- ✓ revert when not enough tokens left in factory to fund (46ms)
- ✓ # fundTheSpecificFarm (63ms)
- ✓ should revert when fundTheSpecificFarm with ZeroAddress as Farm (44ms)
- ✓ should revert when fundTheSpecificFarm with ZeroAddress as RewardToken
- ✓ should revert when fundTheSpecificFarm with zero amount (39ms)
- ✓ # revert with 'Wrong confirmation param 2
- ✓ # deployAndFundPerpetualFarm (223ms)
- ✓ # revert on deployAndFundPerpetualFarm when not enough tokens left in factory to fund (69ms)
- ✓ # withdrawCollectedFeesETHOnFarm (74ms)
- ✓ revert on withdrawCollectedFeesETHOnFarm when passing zero address as farm (68ms)
- ✓ # withdrawCollectedFeesERCOnFarm (99ms)
- ✓ revert on withdrawCollectedFeesERCOnFarm when passing zero address as farm (82ms)
- ✓ revert when not mantainer trying to call setter

- ✓ # setMinTimeToStakeOnFarm
- ✓ revert when setMinTimeToStakeOnFarm on ZeroAddress as Farm
- ✓ # setIsEarlyWithdrawAllowedOnFarm
- ✓ revert when setIsEarlyWithdrawAllowedOnFarm on ZeroAddress as Farm
- ✓ # setStakeFeePercentOnFarm
- ✓ revert when setStakeFeePercentOnFarm on ZeroAddress as Farm
- ✓ # setRewardFeePercentOnFarm
- ✓ revert when setRewardFeePercentOnFarm on ZeroAddress as Farm
- ✓ # migrateUserStake
- ✓ revert when migrateUserStake on ZeroAddress as Farm
- ✓ # setFlatFeeAmountDepositOnFarm
- ✓ revert when setFlatFeeAmountDepositOnFarm on ZeroAddress as Farm
- ✓ # setFlatFeeAmountWithdrawOnFarm
- ✓ revert when setFlatFeeAmountWithdrawOnFarm on ZeroAddress as Farm
- ✓ # setIsFlatFeeAllowedOnFarm
- ✓ revert when setIsFlatFeeAllowedOnFarm on ZeroAddress as Farm
- ✓ # setCoolDownOnTokensFarm
- ✓ revert when setCoolDownOnTokensFarm on ZeroAddress as Farm
- ✓ # setWarmupOnTokensFarm
- ✓ revert when setWarmupOnTokensFarm on ZeroAddress as Farm
- ✓ # withdrawTokensIfStuckOnFarm
- ✓ revert when withdrawTokensIfStuckOnFarm with Zero Address as Farm
- ✓ revert when withdrawTokensIfStuckOnFarm with Zero Address as pulling token
- ✓ revert when withdrawTokensIfStuckOnFarm with Zero Address as beneficiary
- ✓ revert when withdrawTokensIfStuckOnFarm with Zero amount of tokens
- ✓ revert when setCoolDownOnTokensFarm on ZeroAddress as Farm
- ✓ # setWarmupOnTokensFarm
- ✓ revert when setWarmupOnTokensFarm on ZeroAddress as Farm
- ✓ # withdrawStuckTokensFromFactory
- ✓ revert when withdrawStuckTokensFromFactory with Zero Address as pulling token
- ✓ revert when withdrawStuckTokensFromFactory with Zero Address as beneficiary
- ✓ revert when withdrawStuckTokensFromFactory with Zero amount of tokens
- ✓ # revert when depositing with warmup is on (54ms)
- ✓ # finaliseDepositOnTokensFarm
- ✓ revert when finaliseDepositOnTokensFarm with Zero Address as Farm
- ✓ # startNewEpochOnSpecificFarm
- ✓ revert on getLastDeployedSpecificFarm when passing zero as startTime
- ✓ revert on getLastDeployedSpecificFarm when passing zero address as farm
- ✓ revert on getLastDeployedSpecificFarm when passing zero address as reward token
- ✓ revert on startNewEpochOnSpecificFarm when not enough tokens left in factory to fund
- ✓ # setIsAllowedOnFarm
- ✓ revert on setIsAllowedOnFarm when passing zero address as farm
- ✓ # setFeePercentsOnFarm

✓ revert on setFeePercentsOnFarm when passing zero address as farm
✓ # setFlatFeeAmountOnFarm
✓ revert on setFlatFeeAmountOnFarm when passing zero address as farm
✓ # setWarmupCoolDownOnTokensFarm
✓ revert on setWarmupCoolDownOnTokensFarm when passing zero address as farm
✓ revert when getLastDeployedFarm with index out of bonds
✓ # getLastDeployedFarm
✓ # getDeployedSpecificFarm
✓ revert when getDeployedFarms with index out of bonds
✓ revert when getDeployedFarms with index out of bonds
✓ revert when getDeployedFarms with One of the index is out of range
✓ revert when setFarmImplementation with index out of bonds
✓ revert when setFarmImplementation with Zero Address as Farm
✓ revert when setProxyAdmin with Zero Address as Admin
✓ revert when setFeeCollector with Zero Address as Admin
✓ revert when setAllImplementationAtOnce with Zero Address as _farmImplementation
✓ revert when setAllImplementationAtOnce with Zero Address as _perpetualFarmImplementation

Setters

✓ # setAllImplementationAtOnce
✓ # setTokensFarmImplementation
✓ # setPerpetualTokensFarmImplementation
✓ # setProxyAdmin
✓ # setFeeCollector (47ms)
✓ revert on setFeeCollector when passing zero address as farm (48ms)

TokensFarmSDKFactory

initialization

✓ # initialize
✓ See if _perpetualFarmImplementation is initialised properly
✓ See if _proxyAdmin is initialised properly
✓ See if _feeCollector is initialised properly

initialization

✓ # initialize
✓ revert when passing zero address as farmImplementation
✓ revert when passing zero address as perpetualFarmImplementation
✓ revert when passing zero address as proxyAdmin
✓ revert when passing zero address as feeCollector

Deployment

1) should revert on 'List of params is not initialised' when deployAndFundTokensFarmSDK

✓ # getLastDeployedFarm before deployments
✓ revert when not admin sending parameters
✓ # deployAndFundTokensFarmSDK (106ms)
✓ # pauseSpecificFarm (111ms)
✓ revert on pauseSpecificFarm when passing zero address as farm (97ms)
✓ # unpauseSpecificFarm
✓ revert on 'Wrong confirmation param 1' when fund TokensFarmSDK
✓ revert when passing zero address as staking token
✓ revert when passing zero address as rewards token
✓ revert on deployAndFundPerpetualFarm when params are not set
✓ # deployAndFundPerpetualFarm (105ms)
✓ # migrateUserStake
✓ revert when migrateUserStake on ZeroAddress as Farm
✓ revert on deployAndFundPerpetualFarm when passing wrong confirmation param
✓ # startNewEpochOnSpecificFarm
✓ revert on startNewEpochOnSpecificFarm when passing zero as startTime
✓ revert on startNewEpochOnSpecificFarm when passing zero address as farm
✓ revert on startNewEpochOnSpecificFarm when passing zero address as reward token
✓ # getLastDeployedFarm after deployments
✓ # getDeployedSpecificFarm
✓ revert when getDeployedFarms with index out of bonds
✓ revert when getDeployedFarms with One of the index is out of range
✓ revert on startNewEpochOnSpecificFarm when not enough tokens left in factory to fund
✓ # deployAndFundPerpetualFarm with startTime = 0 (88ms)
✓ revert when finaliseDepositOnTokensFarm with Zero Address as Farm
✓ revert on finaliseDepositOnTokensFarm
✓ # getLastDeployedFarm

Funding

✓ revert when passing zero address as farm
✓ revert when passing zero address as reward token
✓ revert when passing zero amount as amount to fund
✓ revert from address that is not maintainer
✓ # fundTheSpecificFarm (38ms)

Withdrawal

✓ # withdrawCollectedFeesETHOnFarm
✓ # withdrawCollectedFeesERCOnFarm (56ms)
✓ # withdrawCollectedFeesETHOnFarm (55ms)
✓ # withdrawTokensIfStuckOnFarm
✓ revert when withdrawTokensIfStuckOnFarm with Zero amount of tokens
✓ revert on withdrawStuckTokensFromFactory when passing zero address as reward token
✓ # withdrawStuckTokensFromFactory

Setters

✓ # setFarmImplementation
✓ # setPerpetualFarmImplementation

- ✓ # activateOrDeactivateFarm
- ✓ # setMinTimeToStakeOnFarm
- ✓ # setMinTimeToStakeOnFarm
- ✓ # setIsEarlyWithdrawAllowedOnFarm
- ✓ # setWarmupOnTokensFarm
- ✓ # setRewardFeePercentOnFarm
- ✓ # setFlatFeeAmountWithdrawOnFarm
- ✓ # setIsFlatFeeAllowedOnFarm
- ✓ # setFarmManagerOnFarm
- ✓ # setFarmManagerOnFarm
- ✓ # setCurrentFeeCollectorOnFarm
- ✓ # setAllImplementationAtOnce
- ✓ revert when setAllImplementationAtOnce with Zero Address as _farmImplementation
- ✓ revert when setAllImplementationAtOnce with Zero Address as _perpetualFarmImplementation

VestingFarmFactory

Check if requires from program is working

- ✓ # Initialize
- ✓ Check getters after initialize
- ✓ revert when passing zero address as iterativeFarmImplementation
- ✓ revert when passing zero address as Proxy admin
- ✓ revert when passing zero address as linearFarmImplementation
- ✓ revert when passing zero address as merkleLinearFarmImplementation
- ✓ revert when passing zero address as merkleIterativeFarmImplementation
- ✓ revert when passing zero address as proxyAdmin
- ✓ revert when passing zero address as _maintainersRegistry
- ✓ revert when passing zero address as feeCollector

Linear Farm deployment and funding

- ✓ return zero address on getLastDeployedFarm when no farm were deployed
- ✓ # deployLinearVestingFarm (61ms)
- ✓ revert when getLastDeployedFarm when farm index is out of range
- ✓ # setStartTimeOnSpecificFarm (39ms)
- ✓ revert when calling setStartTimeOnSpecificFarm with zero address as farm

✓ revert when calling setStartTimeOnSpecificFarm with startTime=0
✓ # deployLinearVestingFarm with startTime=0 (76ms)
✓ # addMoreUsersOnSpecificRegularFarm (43ms)
✓ revert when calling addMoreUsersOnSpecificRegularFarm with zero address as farm
✓ revert when calling setEndTimeRegularLinear with endTime=0
✓ revert when calling setEndTimeRegularLinear with zero address as farm
✓ # setEndTimeRegularLinear
✓ # setFarmImplementation
✓ revert when setFarmImplementation with zero address as farm
✓ # getDeployedFarms
✓ revert on getDeployedFarms when startIndex >= endIndex
✓ revert on getDeployedFarms when farmIndex is out of range
✓ revert on getDeployedFarms when endInxed is greater than amount of farms deployed
✓ revert when setFarmImplementation with farm index out of range
✓ # setProxyAdmin
✓ revert when setProxyAdmin with zero address as proxy admin
✓ # setFeeCollector
✓ revert when setFeeCollector with zero address as proxy admin
✓ revert when not TokensFarmCongress calls fundAndOrActivateSpecificVestingFarm
✓ revert when calling fundAndOrActivateSpecificVestingFarm with zero address as farm (40ms)
✓ revert when calling fundAndOrActivateSpecificVestingFarm with zero address as reward token
✓ # fundAndOrActivateSpecificVestingFarm (48ms)
1) revert on 'Amount must be over or equal to 0' when calling fundAndOrActivateSpecificVestingFarm with zero amount
✓ # pauseSpecificFarm
✓ revert when calling pauseSpecificFarm with zero address as farm
✓ # addMoreUsersOnSpecificRegularFarm
✓ # removeUserOnSpecificRegularFarm (43ms)
✓ revert when calling removeUserOnSpecificRegularFarm with zero address as user (40ms)
✓ revert when calling removeUserOnSpecificRegularFarm with user not being added before
✓ # withdrawLeftOverTokensOnSpecificVestingFarm (40ms)
✓ revert when withdrawLeftOverTokensOnSpecificVestingFarm with zero address as farm
✓ revert when withdrawLeftOverTokensOnSpecificVestingFarm with zero address as collector
✓ # emergencyAssetsWithdrawalOnSpecificVestingFarm (64ms)
✓ revert when calling emergencyAssetsWithdrawalOnSpecificVestingFarm with zero address as farm (56ms)
✓ revert when calling emergencyAssetsWithdrawalOnSpecificVestingFarm with zero address as token to withdraw (57ms)
✓ revert when calling withdrawStuckTokensFromFactory with zero address as token to withdraw (58ms)
✓ # withdrawStuckTokensFromFactory (53ms)
✓ revert when calling withdrawStuckTokensFromFactory with zero address as token to withdraw (56ms)
✓ revert when calling withdrawStuckTokensFromFactory with zero address as beneficiary (56ms)

2) revert when calling withdrawStuckTokensFromFactory with zero amount of token to withdraw
Iterative farm deployment and funding
✓ # deployIterativeVestingFarm (83ms)
✓ # addMoreUsersOnSpecificRegularFarm
✓ revert when not TokensFarmCongress cals fundAndOrActivateSpecificVestingFarm
✓ # fundAndOrActivateSpecificVestingFarm (42ms)
✓ # pauseSpecificFarm
✓ # addMoreUsersOnSpecificRegularFarm
✓ revert when calling removeUserOnSpecificRegularFarm with user not being added before
✓ revert when calling removeUserOnSpecificRegularFarm with zero address as farm
✓ # withdrawLeftOverTokensOnSpecificVestingFarm
✓ # emergencyAssetsWithdrawalOnSpecificVestingFarm (64ms)
Merkle Iterative farm deployment and funding
✓ # deployAndFundMerkleIterativeVestingFarm (122ms)
✓ revert when calling deployAndFundMerkleIterativeVestingFarm with budget > contract balance
✓ # setMerkleRootForFarm (42ms)
✓ # pauseSpecificFarm
✓ # fundAndOrActivateSpecificVestingFarm (45ms)
Merkle Linear Farm deployment and funding
✓ # deployAndFundMerkleLinearVestingFarm (106ms)
✓ # getLastDeployedFarm
✓ # setEndTimeMerkleLinear
✓ revert when calling setEndTimeMerkleLinear with zero address as farm
✓ # setMerkleRootForFarm (43ms)
✓ revert when calling setMerkleRootForFarm with zero address as farm (39ms)
✓ revert when not tokensFarmCongress calls fundAndOrActivateSpecificVestingFarm (63ms)
✓ # fundAndOrActivateSpecificVestingFarm (46ms)
✓ # setAllImplementationAtOnce
✓ revert when calling setAllImplementationAtOnce with zero address as LinearFarm
✓ revert when calling setAllImplementationAtOnce with zero address as IterativeFarm
✓ revert when calling setAllImplementationAtOnce with zero address as MerkleLinear
✓ revert when calling setAllImplementationAtOnce with zero address as MerkleIterative
✓ revert when deploying farm with zero address as reward token
3) revert when deploying farm with zero amount of reward token
✓ # pauseSpecificFarm
✓ revert when removeUserFromMerkleFarm which is not part of farm (45ms)
✓ # removeUserFromMerkleFarm
✓ revert when calling removeUserFromMerkleFarm with zero address as farm
✓ # withdrawLeftOverTokensOnSpecificVestingFarm
✓ # emergencyAssetsWithdrawalOnSpecificVestingFarm (59ms)

245 passing (12s)
3 failing

Contract: ReferralDashboard

initialize

- ✓ Address of token can not be 0×0 (156ms)
- ✓ Address of congress can not be 0×0 (103ms)
- ✓ Address of maintainers reg can not be 0×0 (94ms)
- ✓ Admin cannot be 0×0 (190ms)

addNewReferral

- ✓ only admin can add referral
- ✓ referral address cannot be zero (53ms)
- ✓ farm address cannot be zero
- ✓ referral reward amount cannot be zero
- ✓ referral reward percent cannot be more than 100
- ✓ add new referral (142ms)

removeReferal

- ✓ cannot remove referral for non existing referrer
- ✓ cannot remove non existing referral (117ms)
- ✓ cannot remove already paid referral (165ms)
- ✓ remove referral when referrer has 1 referal (142ms)
- ✓ remove zero referral when referrer has 2 referals (234ms)

changeReferralInfo

- ✓ referrer can not be 0×0
- ✓ referrer has to exist
- ✓ info can be changed only for existing referral (96ms)
- ✓ cannot change reward to zero (121ms)
- ✓ cannot change reward percent to more than 100% (97ms)
- ✓ cannot change info for paid referral (192ms)
- ✓ changes info for referral [lower reward amount]
- ✓ changes info for referral [greater reward amount] (137ms)
- ✓ changes info for referral [same reward amount] (114ms)
- ✓ changes info for referral (129ms)

withdraw

- ✓ strart referal has to be in order (89ms)
- ✓ end referal cannot exceed referal number (94ms)
- ✓ strart referal has to be greater than end referal (94ms)
- There is nothing to withdraw
- ✓ referrer can withdraw his reward (114ms)

emergency withdraw

- ✓ only token congress can call emergency withdraw (82ms)
- ✓ beneficiary cannot be zero address (88ms)
- ✓ amount to withdraw cannot exceed contract balance (92ms)
- ✓ withdraw all contract balance when with zero amount argument (111ms)
- ✓ withdraw half of the contract balance (108ms)

ichange user access
✓ cannot add admin twice
✓ sets admin
✓ cannot remove admin role from not admin
✓ revoke admin role from admin
fund
✓ cannot fund contract with zero amount
✓ fund contract (54ms)
getters
✓ getAllReferrerInfo (141ms)
✓ getAllReferrers (127ms)

41 passing (4s)

Contract: IterativeVestingFarm
initialize
✓ Address of token can not be 0×0 (324ms)
✓ Start time can not be in the past (189ms)
✓ Cannot set zero number of portions (170ms)
✓ Cannot set zero distribution interval (183ms)
✓ Early claim available percentage should be above 0 and below 100 (177ms)
✓ Before start claim percentage should be above 0 and below 100 (144ms)
✓ Initial Fund Percent should be above 5 and up to 100 (224ms)
✓ Partial funding must be false when initial fund percent is 100 and true otherwise (441ms)
User functions
✓ Cannot add users to an active farm (295ms)
✓ Arrays should have same length when adding users (255ms)
✓ Should add new users and rewards (412ms)
✓ Cannot remove users from an active farm (252ms)
✓ Cannot remove users who are paid (467ms)
✓ Should remove users (350ms)
Activation functions
✓ Should fund farm with non-partial funding (340ms)
✓ Should fund farm with partial funding (345ms)
✓ Cannot fund less than initialFundPercent while partial funding (310ms)
✓ Should pause farm (266ms)
Setter functions
✓ Cannot set start time when start time has already arrived (225ms)
✓ Cannot set start time in the past (213ms)
✓ Cannot set start time after end time (219ms)
✓ Should set start time (305ms)

Getter functions
- ✓ Should get total rewards, that are locked, unlocked and withdrawn (642ms)
- ✓ Should get info of user (373ms)
- ✓ Should get amount of payouts without burned part (293ms)

Withdraw functions
- ✓ Cannot claim rewards with insufficient funds (293ms)
- ✓ Cannot claim rewards with zero claim percent (336ms)
- ✓ Cannot claim rewards from an inactive farm (289ms)
- ✓ Cannot claim rewards if user has been paid out (351ms)
- ✓ Should claim whole rewards (420ms)
- ✓ Cannot withdraw rewards with insufficient funds (283ms)
- ✓ Cannot withdraw rewards if farm has not started yet (300ms)
- ✓ Cannot withdraw rewards from an inactive farm (312ms)
- ✓ Cannot withdraw rewards if user has been paid out (357ms)
- ✓ Should withdraw rewards (360ms)
- ✓ Cannot remove superfluous rewards if farm is not finished yet (227ms)
- ✓ Cannot remove superfluous rewards if there's none of them (253ms)
- ✓ Should remove superfluous rewards (293ms)
- ✓ Cannot withdraw assets from an active farm (308ms)
- ✓ Cannot withdraw assets if asset token address is zero (215ms)
- ✓ Cannot withdraw assets to zero address (263ms)
- ✓ Should withdraw assets (315ms)

Contract: LinearVestingFarm

initialize
- ✓ Address of token can not be 0×0 (121ms)
- ✓ Start time can not be in the past (107ms)
- ✓ End time can not be before start time (102ms)
- ✓ Early claim available percentage should be above 0 and below 100 (108ms)
- ✓ Before start claim percentage should be above 0 and below 100 (114ms)
- ✓ Initial Fund Percent should be above 5 and up to 100 (158ms)
- ✓ Partial funding must be false when initial fund percent is 100 and true otherwise (302ms)

User functions
- ✓ Cannot add users to an active farm (208ms)
- ✓ Arrays should have same length when adding users (201ms)
- ✓ Should add new users and rewards (322ms)
- ✓ Cannot remove users from an active farm (224ms)
- ✓ Cannot remove users who are paid (309ms)
- ✓ Should remove users (308ms)

Activation functions
- ✓ Should fund farm with non-partial funding (265ms)
- ✓ Should fund farm with partial funding (284ms)
- ✓ Cannot fund less than initialFundPercent while partial funding (297ms)
- ✓ Should pause farm (236ms)

Setter functions
- ✓ Cannot set start time when start time has already arrived (191ms)
- ✓ Cannot set start time in the past (188ms)
- ✓ Cannot set start time after end time (204ms)
- ✓ Should set start time (238ms)
- ✓ Cannot set end time in the past (237ms)
- ✓ Should set end time (232ms)

Getter functions
- ✓ Should get last reward time (217ms)
- ✓ Should get total amount of user rewards (266ms)
- ✓ Should get total rewards, that are locked, unlocked and withdrawns (360ms)
- ✓ Should get info of user (328ms)
- ✓ Should get amount of payouts without burned part (275ms)

Withdraw functions
- ✓ Cannot claim rewards with insufficient funds (294ms)
- ✓ Cannot claim rewards with zero claim percent (351ms)
- ✓ Cannot claim rewards from an inactive farm (233ms)
- ✓ Cannot claim rewards if user has been paid out (278ms)
- ✓ Should claim whole rewards (375ms)
- ✓ Cannot withdraw rewards with insufficient funds (301ms)
- ✓ Cannot withdraw rewards if farm has not started yet (268ms)
- ✓ Cannot withdraw rewards from an inactive farm (248ms)
- ✓ Cannot withdraw rewards if user has been paid out (296ms)
- ✓ Should withdraw rewards (345ms)
- ✓ Cannot remove superfluous rewards if farm is not finished yet (198ms)
- ✓ Cannot remove superfluous rewards if there's none of them (247ms)
- ✓ Should remove superfluous rewards (283ms)
- ✓ Cannot withdraw assets from an active farm (305ms)
- ✓ Cannot withdraw assets if asset token address is zero (204ms)
- ✓ Cannot withdraw assets to zero address (235ms)
- ✓ Should withdraw assets (285ms)

87 passing (24s)

Contract: MerkleIterativeVesting

no partial fund main flow
- ✓ initialize (234ms)
- ✓ fundAndOrActivate (201ms)
- ✓ user cannot witdraw before vesting start time (95ms)
- ✓ user withdraw amount cannot be zero (71ms)
- ✓ not a user cannot withdraw portion with merkleproof of correct user (76ms)
- ✓ withdraw 1 period out of 10 (803ms)
- ✓ user cannot withdraw vested portion at the same period twice (92ms)
- ✓ withdraw all rewards for all users period by period (5438ms)

partial fund main flow
- ✓ initialize (158ms)
- ✓ allowance of tokens
- ✓ cannot start farm without sufficient fund percent (58ms)
- ✓ start and fund farm with sufficient fund percent (144ms)
- ✓ user cannot witdraw before vesting start time
- ✓ user withdraw amount cannot be zero
- ✓ not a user cannot withdraw portion with merkleproof of correct user (45ms)
- ✓ withdraw 1 period out of 10 (951ms)
- user cannot wtihdraw when there is no reward
- ✓ GETTER: getTotalRewardsLockedUnlockedAndWithdrawn()
- ✓ user cannot withdraw vested portion at the same period twice (71ms)
- ✓ withdraw 5 periods rewards for all users period by period (2523ms)
- ✓ when farm runs out fund, users cannot withdraw rewards (86ms)
- farm is not active when it run out of funds
- ✓ add rest of funds to farm (93ms)
- ✓ withdraw rest of rewards for all users after all time has passed (666ms)

Contract: MerkleIterativeVesting Unit

Initialize
- ✓ Root cannot be zero (131ms)
- ✓ Tokens address can not be 0×0 (97ms)
- ✓ Start time can not be in the past (103ms)
- ✓ Portion of distribution has to be over 0 (109ms)
- ✓ Total rewards can not be equal or under 0 (110ms)
- ✓ Claim percentage should be above 0 and below 100 (125ms)
- ✓ Claim percentage should be above 0 and below 100 (89ms)
- ✓ Initial Fund Percent should be above 5 and up to 100 (299ms)
- ✓ Delay between two vesting days needs to be over 0 (133ms)

setStartTime
- ✓ start time cannot be in the past
- ✓ Cannot change start time after vesting has started
- ✓ start time cannot be after end time (191ms)
- ✓ sets start time (73ms)
- ✓ sets start time [number of porions = 1] (170ms)

removeUser
- ✓ cannot remove user that doesn't exist
- ✓ remove user (49ms)
- ✓ cannot remove user when farm is active (152ms)
- ✓ user can not be kicked if he is already paid (454ms)

setMerkleRoot
- ✓ admin can set merkle root as zero
- ✓ cannot set merkle root as zero
- ✓ cannot set merkle root when farm is active (138ms)

collect leftovers
✓ admin cannot collect leftovers before end of farm (336ms)
✓ admin cannot collect pending rewards as leftovers (332ms)
✓ admin can collect lefovers (333ms)
emergencyAssetsWithdrawal
✓ cannot withdraw while farm is active (298ms)
✓ asset address cannot be zero (273ms)
✓ collector address cannot be zero (372ms)
✓ admin can perform emergency withdraw (371ms)
claimWholeRewards with burn fee
✓ cannot claim if it is not available at current farm (185ms)
✓ user cannot claimWholeRewards when Vesting Farm is not activated (340ms)
✓ claimWholeRewards with burn fees (320ms)
✓ user are not in vesting after claimWholeRewards (401ms)
✓ only registred user can claimWholeRewards (293ms)
✓ claim whole reward after vesting is over [partial vesting] (333ms)
fundAndOrActivate
✓ function takes no token when sufficient balance is present (262ms)
✓ fund with correct amount when passed value is zero (243ms)
✓ fund partcially correctly (267ms)
✓ 50% partcial funding 25% present on contract, can fund extra 25% to start vesting
✓ reverts with not sufficient funds (178ms)
getTotalRewardsLockedUnlockedAndWithdrawn
✓ check value before vesting start (213ms)
✓ check value after vesting start (172ms)
✓ check value after some vested portions (3098ms)
✓ check value 1 portion total (882ms)
areUsersOnFarm
✓ areUsersOnFarm

64 passing (25s)

Contract: MerkleLinearVesting Unit
Initialize
✓ Root cannot be zero (207ms)
✓ Tokens address can not be 0×0 (124ms)
✓ Start time can not be in the past (151ms)
✓ Total rewards can not be equal or under 0 (91ms)
✓ Claim percentage should be above 0 and below 100 (113ms)
✓ Claim percentage should be above 0 and below 100 (119ms)
✓ Initial Fund Percent should be above 5 and up to 100 (304ms)
setStartTime
✓ start time cannot be in the past
✓ Cannot change start time after vesting has started

- ✓ start time cannot be after end time (154ms)
- ✓ sets start time

removeUser

- ✓ cannot remove user that doesn't exist (87ms)
- ✓ remove user (76ms)
- ✓ cannot remove user when farm is active (173ms)
- ✓ user can not be kicked if he is already paid (352ms)

setMerkleRoot

- ✓ admin can set merkle root as zero (65ms)
- ✓ cannot set merkle root as zero (43ms)
- ✓ cannot set merkle root when farm is active (134ms)

collect leftovers

- ✓ admin cannot collect leftovers before end of farm (355ms)
- ✓ admin cannot collect pending rewards as leftovers (349ms)
- ✓ admin can collect lefovers (339ms)

emergencyAssetsWithdrawal

- ✓ cannot withdraw while farm is active (268ms)
- ✓ asset address cannot be zero (288ms)
- ✓ collector address cannot be zero (299ms)
- ✓ admin can perform emergency withdraw (348ms)

claimWholeRewards with burn fee

- ✓ cannot claim if it is not available at current farm (345ms)
- ✓ user cannot claimWholeRewards when Vesting Farm is not activated (506ms)
- ✓ claimWholeRewards with burn fees (562ms)
- ✓ user are not in vesting after claimWholeRewards (656ms)
- ✓ only registred user can claimWholeRewards (329ms)
- ✓ claim whole reward after vesting is over [partial vesting] (331ms)

Contract: MerkleLinearVesting

Contract: MerkleLinearVesting

- ✓ initialize (126ms)
- ✓ fundAndOrActivate (194ms)
- ✓ user cannot witdraw before vesting start time (43ms)
- ✓ user withdraw amount cannot be zero (50ms)
- ✓ not a user cannot withdraw portion with merkleproof of correct user (43ms)
- ✓ withdraw 1 period out of 10 (757ms)
- ✓ user cannot withdraw vested portion at the same time twice (112ms)
- ✓ withdraw all rewards for all users period by period (7521ms)

partial fund main flow

- ✓ initialize (334ms)
- ✓ allowance of tokens (55ms)
- ✓ cannot start farm without sufficient fund percent (39ms)
- ✓ start and fund farm with sufficient fund percent (159ms)
- ✓ user cannot witdraw before vesting start time

✓ user withdraw amount cannot be zero (45ms)
✓ not a user cannot withdraw portion with merkleproof of correct user (77ms)
✓ withdraw 1 period out of 10 (933ms)
✓ GETTER: getTotalRewardsLockedUnlockedAndWithdrawn()
✓ user cannot withdraw vested portion at the same time twice (52ms)
✓ withdraw 5 periods rewards for all users period by period (3211ms)
✓ when farm runs out fund, users cannot withdraw rewards (114ms)
✓ add rest of funds to farm (119ms)
✓ withdraw rest of rewards for all users after all time has passed (924ms)

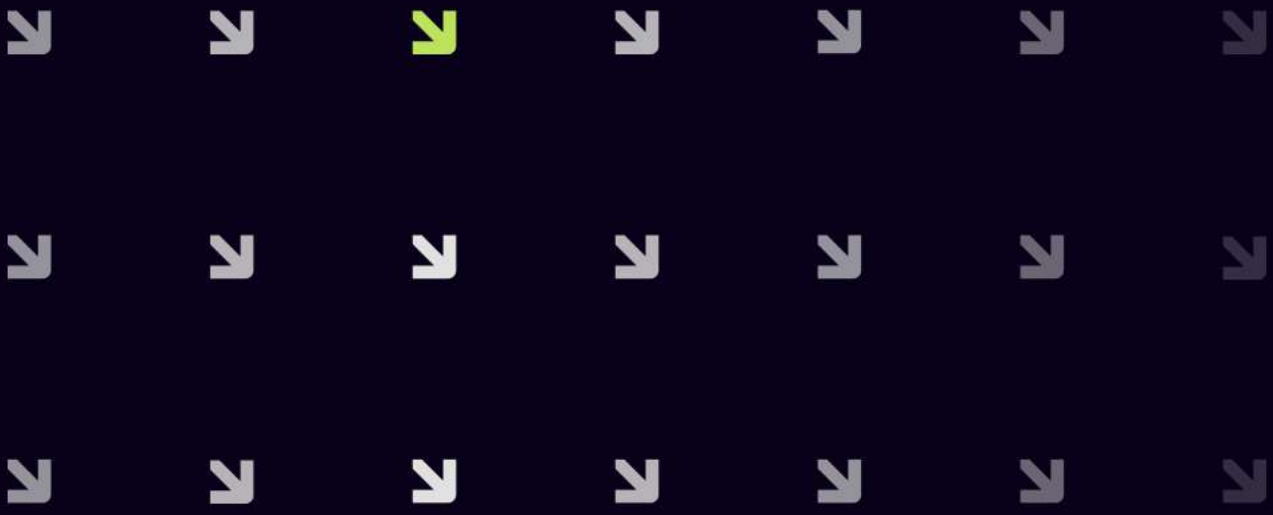
53 passing (23s)

FILE	% STMTS	% BRANCH	% FUNCS
ReferralDashboard.sol	100	98.44	100
TokensFarmFactory.sol	95.79	91.67	94.87
TokensFarmSDKFactory.sol	93.09	85.19	89.74
PerpetualTokensFarm.sol	98.96	82	100
PerpetualTokensFarmSDK.sol	95.6	75.6	100
TokensFarm.sol	100	86.41	100
TokensFarmSDK.sol	97.19	78.68	100
VestingFarmFactory. sol	96.69	89.8	93.1
IterativeVestingFarm .sol	99.5	91.53	100
LinearVestingFarm.sol	100	93.88	100
MerkleIterativeVesting.sol	94.64	91.18	96
MerkleLinearVesting.sol	90.11	81.03	92.59
All Files	**95.75**	**86.75**	**94.8**

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



zokyo