TokensFarm

TOKENSFARM

SMART CONTRACT AUDIT

zokyo

September 29th 2022 | v. 2.0

Security Audit Score

PASS

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

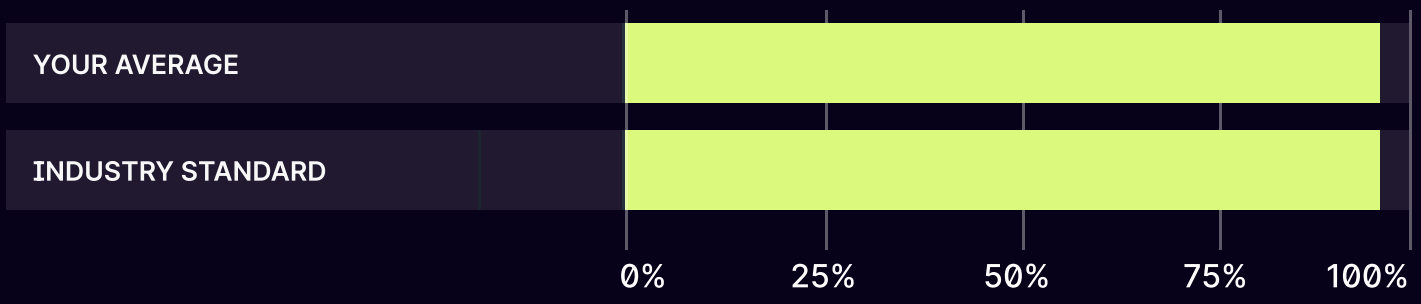
TECHNICAL SUMMARY

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 EVM 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: ttps://github.com/Tokensfarm/tokensfarm-contracts

Initial commit: 43e0e617143d209dacb6f8c71e31c53434a08ef4

Final commit: 59df44392e1b7b80d713b85b8f6a747a2a2f7cb9

Within the scope of this audit, Zokyo auditors have reviewed the following contracts:

- PerpetualTokensFarmSDK.sol
- PerpetualTokensFarm.sol
- TokensFarmSDK.sol
- TokensFarm.sol
- TokensFarmFactory.sol
- TokensFarmSDKFactory.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. A part of this work included writing a unit test suite using the Truffle testing framework. In summary, our strategies consisted mostly 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, the Zokyo Security team has audited the whole set of contracts within the scope. The contracts consist of staking farm contracts, SDK version of contracts and factories for creating and managing instances of staking farm contracts.

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

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 optimizations and logic clarifications. Nevertheless, all the issues were successfully resolved or verified by the TokensFarm team. After all the fixes, the contracts have passed all security tests.

It should be mentioned that all the contracts are upgradable, which means that the admin of the contract can upgrade its 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 contract logic. The Zokyo Security team has prepared our own set of unit-tests as well in order to validate crucial bussiness logic scenarios. It should also be mentioned that due to the complexity and the size of the contracts, they lack readbility. We recommend the TokensFarm team 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 user was verified in function finalizeDeposit(). The issue occured because variables have a similar naming such as `user` and `_user`. Nevertheless, the issue was fixed and the auditors verified its correctness.

PROTOCOL OVERVIEW

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 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 warmup period, the user has to make a deposit request and finalize it after the warmup period is finished. Users can withdraw their stake at any time in case early withdrawal is allowed. In this case, users still have to wait for minimal time to stake in order to receive 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.

STRUCTURE AND ORGANIZATION OF DOCUMENT

For the ease of navigation, document's sections are arranged from the most critical to the least critical. Issues are tagged "Resolved" or "Unresolved" depending on whether they have been fixed or addressed. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

Critical

The issue affects the contract in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss.

High

The issue affects the ability of the contract to compile or operate in a significant way.

Medium

The issue affects the ability of the contract to operate in a way that doesn't significantly hinder its behavior.

Low

The issue has minimal impact on the contract's ability to operate.

Informational

The issue has no impact on the contract's ability to operate.

COMPLETE ANALYSIS

HIGH-1 | RESOLVED

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 is 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 `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 has 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 in 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 emergency withdrawal of the same stake more than once.

HIGH-4 | RESOLVED

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 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 in the cases 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` unfinalized.
   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` (unfinalized) 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 stake with the ID `2` and won't be able to finalize 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.

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 unused functions.

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

LOW-3 | RESOLVED

Reduce without stake id might revert in case not the first stake was reduced with 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: `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.

INFO-1 | RESOLVED

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

Recommendation:
Mark visibility for 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 is 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 storage variable `totalRewards`. 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 ssue is marked as informational since it dosn't prevent thewithdrawal of stakes, but it 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 intended functionality to increase this variable during every redistribution.

	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

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)

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