CERTIK

League of Traders - token farm - audit

Security Assessment

CertiK Assessed on Jun 2nd, 2025

CertiK Assessed on Jun 2nd, 2025

League of Traders - token farm - audit

The security assessment was prepared by CertiK, the leader in Web3.0 security.

Executive Summary

TYPES	ECOSYSTEM	METHODS
Staking	Binance Smart Chain (BSC)	Formal Verification, Manual Review, Static Analysis

LANGUAGE	TIMELINE	KEY COMPONENTS
Solidity	Delivered on 06/02/2025	N/A

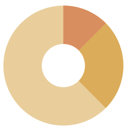
CODEBASE	COMMITS
source View All in Codebase Page	6122cd26871c1c0ae9292ac3cc253f4f5704ac45 View All in Codebase Page

Highlighted Centralization Risks

Privileged role can mint tokens

Has blacklist/whitelist

Vulnerability Summary



8	6	0	2	0
Total Findings	Resolved	Partially Resolved	Acknowledged	Declined

🟧	1	Centralization	1 Acknowledged	Centralization findings highlight privileged roles & functions and their capabilities, or instances where the project takes custody of users' assets.
🟥	0	Critical		Critical risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.
🟧	0	Major		Major risks may include logical errors that, under specific circumstances, could result in fund losses or loss of project control.
🟨	2	Medium	2 Resolved	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
🟨	5	Minor	4 Resolved, 1 Acknowledged	Minor risks can be any of the above, but on a smaller scale. They generally do not compromise the overall integrity of the project, but they may be less efficient than other solutions.

■ 0	Informational	Informational errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

TABLE OF CONTENTS | LEAGUE OF TRADERS - TOKEN FARM - AUDIT

Summary

Executive Summary

Vulnerability Summary

Codebase

Audit Scope

Approach & Methods

Findings

LOT-03 : Centralization Risks

LOT-08 : Extending Current Staking Duration Does Not Change Eligible Rewards

LOT-09 : `_beforeTokenTransfer()` is not implemented in BEP20

LOT-04 : SafeTransfer should be used in place of transfer

LOT-05 : Incompatibility With Deflationary Tokens

LOT-06 : Mismatch between Reward Decay Schedule and Staking Duration Could Leave Unclaimed Rewards in the Contract

LOT-07 : `getAPY()` can be misleading when staked token and reward token are different

LOT-10 : Inconsistent logic

Optimizations

LOT-01 : User-Defined Getters

Appendix

Disclaimer

CODEBASE | LEAGUE OF TRADERS - TOKEN FARM - AUDIT

Repository

source

Commit

6122cd26871c1c0ae9292ac3cc253f4f5704ac45

AUDIT SCOPE | LEAGUE OF TRADERS - TOKEN FARM - AUDIT

3 files audited ● 2 files with Acknowledged findings ● 1 file with Resolved findings

ID	Repo	File	SHA256 Checksum
● LTT	League-of-Traders/lot-contract	LotToken.sol	b9c1b65cf6cfe37ea4b4294cf94d24dd221de3e55e564f187d8d71b0f3a2f727
● TFT	League-of-Traders/lot-contract	TokenFarm.sol	aa2b22663fd6dd30e79a82442ec511425a8e74659d7c11ec1962e018048d24df
● LTL	League-of-Traders/lot-contract	projects/token-farm/contracts/LotToken.sol	f068d44461be49056878fb7258d4fe19c8f8168474f9186aff31abf162fe383b

APPROACH & METHODS | LEAGUE OF TRADERS - TOKEN FARM - AUDIT

This report has been prepared for League of Traders to discover issues and vulnerabilities in the source code of the League of Traders - token farm - audit project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Formal Verification, Manual Review, and Static Analysis techniques.

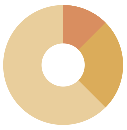
The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Testing the smart contracts against both common and uncommon attack vectors;
- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

FINDINGS | LEAGUE OF TRADERS - TOKEN FARM - AUDIT



8	0	1	0	2	5	0
Total Findings	Critical	Centralization	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for League of Traders - token farm - audit. Through this audit, we have uncovered 8 issues ranging from different severity levels. Utilizing the techniques of Formal Verification, Manual Review & Static Analysis to complement rigorous manual code reviews, we discovered the following findings:

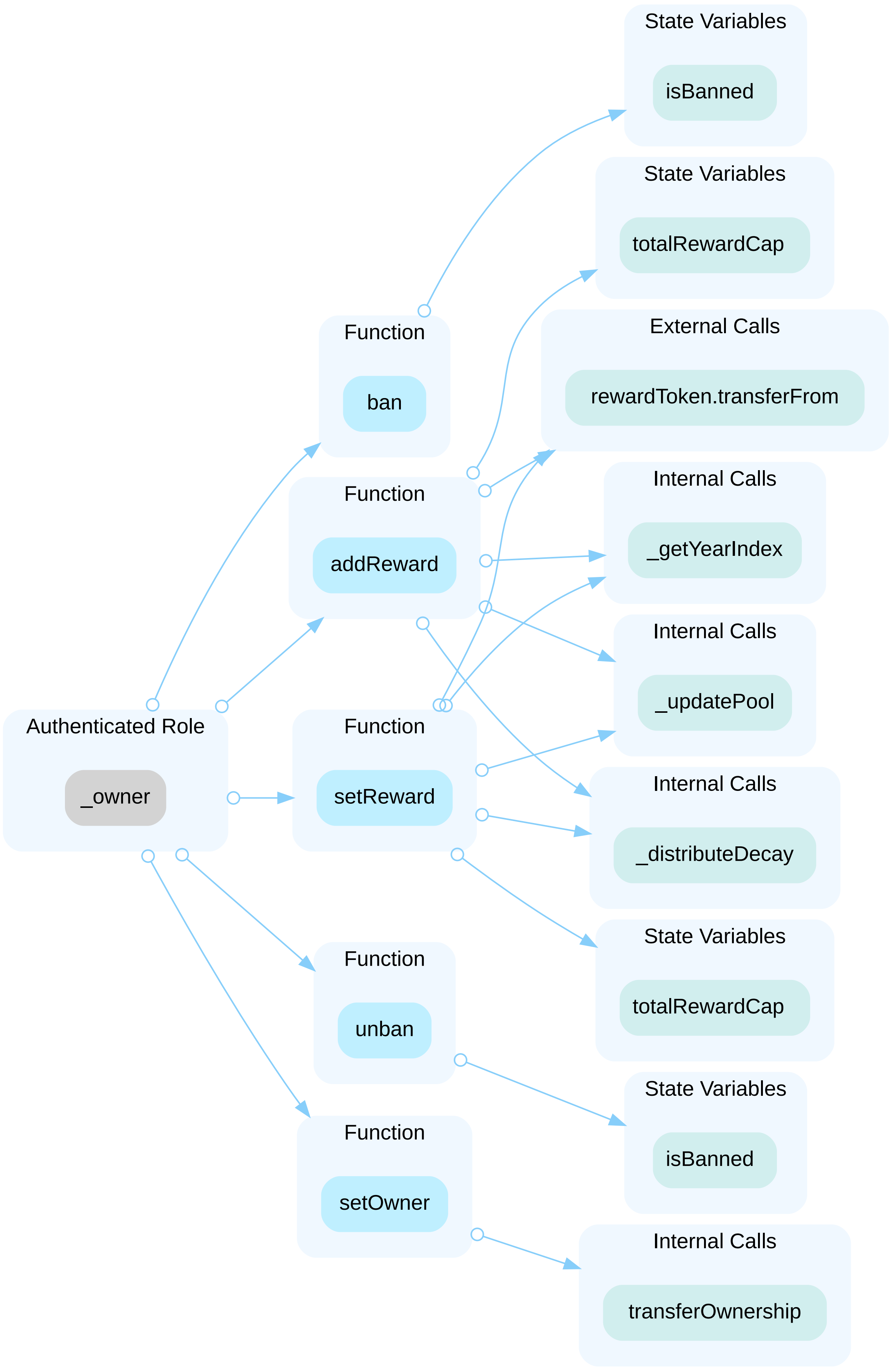
ID	Title	Category	Severity	Status
LOT-03	**Centralization Risks**	**Centralization**	**Centralization**	**● Acknowledged**
LOT-08	Extending Current Staking Duration Does Not Change Eligible Rewards	Logical Issue	Medium	● Resolved
LOT-09	`_beforeTokenTransfer()` Is Not Implemented In BEP20	Logical Issue, Inconsistency	Medium	● Resolved
LOT-04	SafeTransfer Should Be Used In Place Of Transfer	Logical Issue	Minor	● Resolved
LOT-05	Incompatibility With Deflationary Tokens	Volatile Code	Minor	● Resolved
LOT-06	Mismatch Between Reward Decay Schedule And Staking Duration Could Leave Unclaimed Rewards In The Contract	Logical Issue	Minor	● Acknowledged
LOT-07	`getAPY()` Can Be Misleading When Staked Token And Reward Token Are Different	Logical Issue	Minor	● Resolved
LOT-10	Inconsistent Logic	Inconsistency	Minor	● Resolved

LOT-03 | CENTRALIZATION RISKS

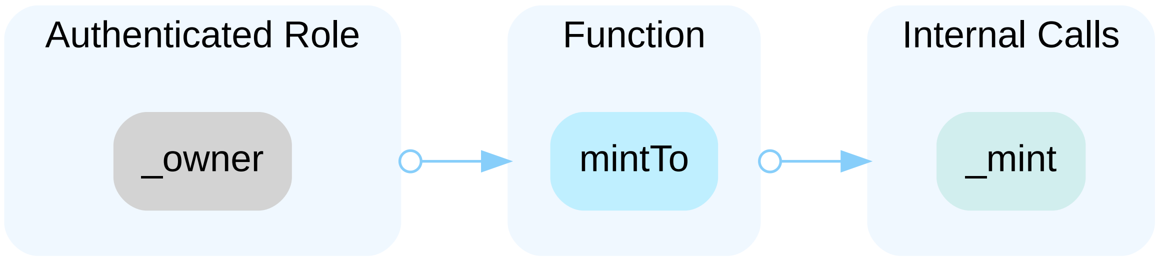
Category	Severity	Location	Status
Centralization	● **Centralization**	**LotToken.sol (source): 9; TokenFarm.sol (source): 73, 100, 117, 398, 404**	● **Acknowledged**

Description

In the contract `TimeBasedStaking`, the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and set / add rewards, ban or unban any address from staking tokens, and change owner address.


State Variables
isBanned
State Variables
totalRewardCap
External Calls
rewardToken.transferFrom
Function
ban
Function
addReward
Internal Calls
_getYearIndex
Internal Calls
_updatePool
Authenticated Role
_owner
Function
setReward
Internal Calls
_distributeDecay
State Variables
totalRewardCap
Function
unban
State Variables
isBanned
Function
setOwner
Internal Calls
transferOwnership

In the contract `LotToken`, the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and mint tokens to any address. Note that the inherited `BEP20` contract also contains a `mint()` function that allows the `_owner` to mint arbitrary amount of token to itself.



In a subsequent commit 91b1da32c6c379d5cfc481a3b81449ba4e5c8077, additional `onlyOwner` functions `setTransferAllowedTimestamp()`, `addToWhitelist()`, and `removeFromWhitelist()` are added. Any compromise to the `_owner` account may allow the hacker to set transfer allowed timestamp and add/remove users from the whitelist.

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets. Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND

- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

Alleviation

[LoT Team, 06/04/2025]: We acknowledge the centralization risks outlined regarding the privileged `_owner` role in both `TimeBasedStaking` and `LotToken` contracts. To mitigate these concerns, we plan to adopt the following strategy:

Immediately after deployment, the owner role will be transferred to a Safe (Gnosis Safe) multisignature wallet, which requires multiple signers to authorize sensitive operations.

[CertiK, 06/04/2025]: It is suggested to implement the aforementioned methods to avoid centralized failure. Also, CertiK strongly encourages the project team to periodically revisit the private key security management of all addresses related to centralized roles.

LOT-08 | EXTENDING CURRENT STAKING DURATION DOES NOT CHANGE ELIGIBLE REWARDS

Category	Severity	Location	Status
Logical Issue	● Medium	TokenFarm.sol (source): 185, 202~209	● Resolved

Description

In the `stake()` function, the check in line 185 enforces that the initial stake cannot be 0, but subsequent stakes can be 0. When subsequent stake is 0, it effectively increases the lock up period to a later `newLockupEnd`. However, the longer stake duration does not change the weight of the existing stake, as the increase in `s.weight` equals `0` in line 209. This means that when a user stakes 0 token and extend the stake duration, he does not benefit from the higher weight associated with a longer duration. More broadly, even when the new stake amount is not zero, it increases the `s.lockupEndTimestamp` for both the existing staked amount and the newly staked amount, but the existing staked amount does not enjoy the higher weight associated with a longer stake duration, but will be subject to the longer staking period before being able to withdraw.

Proof of Concept

The following test shows that extending stake duration versus longer upfront stake have different rewards.

```
    it("should compare rewards between extended stake vs upfront longer stake",
async () => {

      // confirms that both users have the same reward token balance before staking
      let userRewardBefore = await rewardToken.balanceOf(user.address);
      let otherRewardBefore = await rewardToken.balanceOf(other.address);
      console.log("userRewardBefore", userRewardBefore);

      expect(userRewardBefore).to.equal(otherRewardBefore);


      // First user: stake 100 tokens for 30 days, then extend to 60 days
      const initialStake = parseEther("100");
      await staking.connect(user).stake(initialStake, 30);

      // Second user: stake 100 tokens for 60 days upfront
      await staking.connect(other).stake(initialStake, 60);

      // Wait for 30 days
      await ethers.provider.send("evm_increaseTime", [days(29)]);
      await ethers.provider.send("evm_mine");

      // First user extends stake to 60 days
      await staking.connect(user).stake(0, 30);

      // Confirm that the user cannot withdraw now
      await
expect(staking.connect(user).withdraw(initialStake)).to.be.revertedWith("Locked");

      // Wait another 30 days
      await ethers.provider.send("evm_increaseTime", [days(45)]);
      await ethers.provider.send("evm_mine");

      // Get rewards for both users
      await staking.connect(user).withdraw(initialStake);
      await staking.connect(other).withdraw(initialStake);

      let userReward = await rewardToken.balanceOf(user.address);
      let otherReward = await rewardToken.balanceOf(other.address);

      expect(otherReward).to.be.greaterThan(userReward);
      console.log("additional reward for other user", otherReward - userReward);
    });
```

test result:

```
userRewardBefore 0n
additional reward for other user 6757971841704718417048n
      ✔ should compare rewards between extended stake vs upfront longer stake
```

Recommendation

We'd like to understand if the intended design allows extending staking duration after initial stake. We recommend that the client consider either separating different staking operations by the same user, so each staking transaction could have their own `lockupEndTimestamp` , or if extending duration is allowed, factor in the incremental weight from the original stake.

Alleviation

[League of Traders, 06/04/2025]: Issue acknowledged. Changes have been reflected in the commit hash: https://github.com/League-of-Traders/lot-contract/commit/200fd06ae3d5116205f51c6af3be08f07090d496

LOT-09 | `_beforeTokenTransfer()` IS NOT IMPLEMENTED IN BEP20

Category	Severity	Location	Status
Logical Issue, Inconsistency	● Medium	projects/token-farm/contracts/LotToken.sol (update): 44 ~51	● Resolved

Description

The token transfer restrictions in the PR https://github.com/League-of-Traders/lot-contract/pull/2 are implemented using the `_beforeTokenTransfer()` function. This assumes that the `transfer()` and `transferFrom()` functions of the inherited token contract contains the `_beforeTokenTransfer()` function call. However, in the `bsc-library`, the `BEP20` contract (https://github.com/League-of-Traders/lot-contract/blob/main/projects/bsc-library/contracts/BEP20.sol) does not contain the `_beforeTokenTransfer()` function at all, which means the intended transfer restrictions would not apply to the `LotToken` contract.

Recommendation

We recommend the team to review and update the `BEP20` dependency to ensure that it has the `_beforeTokenTransfer()` function, and include additional tests to confirm that the transfer restrictions function as intended for the `LotToken` contract.

Alleviation

[League of Traders, 06/09/2025]: Issue acknowledged. Changes have been reflected in the commit hash: https://github.com/League-of-Traders/lot-contract/commit/8104527c89f298f0766bcb5e84420542f6775dfc

LOT-04 | SAFETRANSFER SHOULD BE USED IN PLACE OF TRANSFER

Category	Severity	Location	Status
Logical Issue	● Minor	TokenFarm.sol (source): 194, 215, 233, 244, 262, 392	● Resolved

Description

SafeTransfer should be used in place of Transfer for Solidity contracts to ensure robust security and error handling. Unlike the basic Transfer function, SafeTransfer incorporates safeguards by validating success of the transfer.

Recommendation

We recommend using `SafeTransfer` / `SafeTransferFrom` instead of `transfer` for token transfer in Solidity contracts.

Alleviation

[League of Traders, 06/04/2025]: Issue acknowledged. Changes have been reflected in the commit hash: https://github.com/League-of-Traders/lot-contract/commit/086d1abc1179b5c317219224ac2481dc12f1058f

LOT-05 | INCOMPATIBILITY WITH DEFLATIONARY TOKENS

Category	Severity	Location	Status
Volatile Code	● Minor	TokenFarm.sol (source): 110, 128, 215, 244, 262, 392~393	● Resolved

Description

When transferring deflationary ERC20 tokens, the input amount may not be equal to the received amount due to the charged transaction fee. The `TimeBasedStaking` contract assumes that the amount received by the contract equals the transfer amount. If that assumption does not hold, core logic such as staking / unstaking / claiming could break due to insufficient token balance.

Recommendation

We advise the client to regulate the tokens supported and add necessary mitigation mechanisms to keep track of accurate balances if there is a need to support deflationary tokens.

Alleviation

[League of Traders, 06/05/2025]: Issue acknowledged. Changes have been reflected in the commit hash: https://github.com/League-of-Traders/lot-contract/commit/7620192316ceb6079a52ec16200968cc252bd93e

[Certik, 06/05/2025]: The mitigation mechanism is capable of handling fee-on-transfer staked tokens. We still advise the client to regulate the tokens supported, in case more variations in token behavior causes unintended consequences.

LOT-06 | MISMATCH BETWEEN REWARD DECAY SCHEDULE AND STAKING DURATION COULD LEAVE UNCLAIMED REWARDS IN THE CONTRACT

Category	Severity	Location	Status
Logical Issue	● Minor	TokenFarm.sol (source): 30, 151~159	● Acknowledged

Description

The `_distributeDecay()` function distributes rewards in a `for` loop of size `100`, which means that rewards will only be fully distributed after 100 years. When users stake tokens, the maximum lockup time period is 4 years. The mismatch means that users will almost certainly be leaving some unclaimed rewards in the contract when they withdraw their staked tokens. After users withdraw, they would not be able to claim any of those unclaimed rewards.

Proof of Concept

The following test shows that there's left over rewards after the `MAX_LOCKUP_DAYS` of 4 years

```
  describe("Reward Distribution", () => {
    it("should have unclaimed rewards remaining after 4 years due to decay
schedule", async () => {
      // Initial stake
      await staking.connect(user).stake(stakeAmount, 365 * 4);

      // Move time forward 4 years
      const fourYears = days(365 * 4);
      await ethers.provider.send("evm_increaseTime", [fourYears]);
      await ethers.provider.send("evm_mine");

      // Calculate total rewards distributed in 4 years
      const startTime = await staking.startTimestamp();
      const endTime = startTime + BigInt(fourYears);
      const distributedRewards = await
staking.getTotalRewardFromTimestamp(startTime, endTime);

      // Get total reward cap
      const totalRewardCap = await staking.totalRewardCap();

      // Calculate remaining rewards
      const remainingRewards = totalRewardCap - distributedRewards;

      // Verify that there are still rewards remaining
      expect(remainingRewards).to.be.gt(0);

      // Calculate theoretical remaining rewards after 4 years
      // Year 1: 1/3 of total
      // Year 2: 1/3 of remaining (2/3 of total)
      // Year 3: 1/3 of remaining (4/9 of total)
      // Year 4: 1/3 of remaining (8/27 of total)
      // Remaining after 4 years should be (2/3)^4 of total
      const expectedRemaining = (totalRewardCap * 16n) / 81n; // (2/3)^4 = 16/81

      // Allow for some small rounding differences
      expect(remainingRewards).to.be.closeTo(expectedRemaining, totalRewardCap /
1000n);
    });
  });
```

test result:

```
Reward Distribution
      ✔ should have unclaimed rewards remaining after 4 years due to decay schedule
```

Recommendation

We advise the client to be transparent about this feature if this is the intended design. Alternatively, consider revising the reward decay schedule to align it with the maximum staking duration.

Alleviation

[League of Traders, 06/04/2025]:

We would like to clarify that this behavior is intentional by design.

When a user withdraws, their reward distribution stops, but the `totalRewardCap` continues to be distributed to all active participants based on the geometric decay schedule.

All stakers are entitled to claim the portion of rewards that were emitted during their participation period, proportionally to their ve-weight.

Any remaining rewards after a users withdrawal are not considered Unclaimed in a global sense they continue to be allocated to future participants under the decay model.

We will consider adding clearer documentation and UI guidance to ensure users fully understand this mechanism before staking.

LOT-07 | `getAPY()` CAN BE MISLEADING WHEN STAKED TOKEN AND REWARD TOKEN ARE DIFFERENT

Category	Severity	Location	Status
Logical Issue	● Minor	TokenFarm.sol (source): 339~344	● Resolved

Description

The `getAPY()` function is intended to show the current annual percentage yield. The function calculates the total amount of reward tokens for the next year and divide it by the total weighted stake. This can be misleading, because if the `stakingToken` and `rewardToken` are different tokens and have different value, the ratio does not actually reflect the "yield" in dollar terms, but only a ratio between the estimated amount of reward token divided by weighted amount of staked tokens. Additionally, only the 4 year lockup gives full weight, and the decay schedule means that for the remaining 3 years after the first year, the APY would be meaningfully lower.

Recommendation

We recommend the client to distinguish the case when the `stakingToken` and `rewardToken` are different tokens and when they are the same tokens. The `getAPY()` function is only accurate whey are the same token. A more suitable function name should be used when they are different tokens, to prevent misleading users into believing it reflects actual APY. Users should also be aware that the current year APY assumes 4 year lockup, and actual APY from subsequent years after the first year will be meaningfully lower.

Alleviation

[League of Traders, 06/04/2025]: Issue acknowledged. Changes have been reflected in the commit hash: https://github.com/League-of-Traders/lot-contract/commit/7af98fbc771170c4af397a72e2781edede97f9e0

LOT-10 | INCONSISTENT LOGIC

Category	Severity	Location	Status
Inconsistency	● Minor	projects/token-farm/contracts/LotToken.sol (update): 28	● Resolved

Description

In the PR https://github.com/League-of-Traders/lot-contract/pull/2, it is stated that:

- transferAllowedTimestamp can be updated under specific conditions:
    - If current time is before the original timestamp and no ETA has been set, it can be updated freely.
    - Otherwise, it can only be updated once more, and only before the calculated ETA.

The second bullet point indicates that the update needs to happen before the calculated ETA, if the current time has passed the original timestamp. However, in the corresponding commit, the implementation is as follows:

```
28  require(newTimestamp <= ETA, "Too late to update");
```

It compares the `newTimestamp` instead of the current `block.timestamp` with the `ETA`. This implementation limits the `newTimestamp` instead of the update timing, as the PR and the `require` statement error message indicates.

Additionally, the second bullet point also states that it can be updated only "once" more. The actual implementation of the code does not enforce only updating "once". In the second time this function is called, the `if` statements in both line 22 and line 25 will be false, and the `require` statement in line 28 does not have to revert in the second update.

Recommendation

If the intention is to limit the timing of the update once it has passed the original timestamp, we recommend updating line 28 to compare `block.timestamp` with `ETA`. If the intention is to restrict the `newTimestamp`, we recommend modifying the error message and the PR to reflect the actual behavior of the code.

Additionally, given that the intention is that this function can only be called once, there should be additional logic in this function to enforce this restriction.

Alleviation

[League of Traders, 06/09/2025]: Issue acknowledged. Changes have been reflected in the commit hash: https://github.com/League-of-Traders/lot-contract/commit/f1f957b9d21b797e50d653711accf0515fe6a293

OPTIMIZATIONS | LEAGUE OF TRADERS - TOKEN FARM - AUDIT

ID	Title	Category	Severity	Status
LOT-01	User-Defined Getters	Gas Optimization	Optimization	● Resolved

LOT-01 | USER-DEFINED GETTERS

Category	Severity	Location	Status
Gas Optimization	● Optimization	TokenFarm.sol (source): 357~359	● Resolved

Description

The linked functions are equivalent to the compiler-generated getter functions for the respective variables.

Recommendation

We advise using compiler-generated getter functions as they are less prone to error and more maintainable than manually written ones.

Alleviation

[League of Traders, 06/04/2025]: Issue acknowledged. Changes have been reflected in the commit hash: https://github.com/League-of-Traders/lot-contract/commit/80374464f5c34b871c3987c14484bacc31f8a422

APPENDIX | LEAGUE OF TRADERS - TOKEN FARM - AUDIT

Finding Categories

Categories	Description
Gas Optimization	Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.
Inconsistency	Inconsistency findings refer to different parts of code that are not consistent or code that does not behave according to its specification.
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases and may result in vulnerabilities.
Logical Issue	Logical Issue findings indicate general implementation issues related to the program logic.
Centralization	Centralization findings detail the design choices of designating privileged roles or other centralized controls over the code.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

DISCLAIMER | CERTIK

UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

Elevating Your Entire Web3 Journey

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.