UMA Accelerating Distributor Incremental Audit

OPENZEPPELIN SECURITY | JANUARY 29, 2024 Security Audits Solidity

Table of Contents

Summary

Type
: DeFi

Timeline
: From 2022-12-12
: To 2022-12-15

Languages
: Solidity

Total Issues
: 9 (6 resolved, 2 partially resolved)

Critical Severity Issues
: 0 (0 resolved)

1 (0 resolved)
Low Severity Issues
4 (3 resolved, 1 partially resolved)
Notes & Additional Information
4 (3 resolved, 1 partially resolved)

Scope

We audited a subset of pull request #32 in the across-token repository at the `dde7aedc766bf69f1cb0eb791b60259c705baa26` commit.

In scope were the following contracts:

```
contracts/AcceleratingDistributor.sol
```

The `AcceleratingDistributor` contract allows users to deposit certain tokens and receive rewards in the form of additional tokens. These rewards are distributed to users who have "staked" their tokens in the contract. To encourage users to remain staked for longer periods of time, the rate at which rewards are issued increases over time, up to a maximum value.

The pull request that was reviewed for this audit adds the ability to stake tokens into the `AcceleratingDistributor` contract on behalf of someone else, effectively donating tokens to

Medium Severity

Configuration update retroactively impacts users rewards

Updating the configuration of a staking pool using the `configureStakingToken` function in the `AcceleratingDistributor` contract will immediately affect the rewards received by all users who have staked in the pool. If the `maxMultiplier` or `secondsToMaxMultiplier` parameters are modified, users who have previously staked in the pool will see a change in their outstanding rewards. Specifically:

- Increasing/decreasing the `maxMultiplier` parameter causes an immediate increase/decrease in outstanding rewards, respectively.
- Increasing/decreasing the `secondsToMaxMultiplier` parameter causes an immediate decrease/increase in outstanding rewards, respectively.

This is a direct result of the calculations performed in the `getUserRewardMultiplier` function where the `fractionOfMaxMultiplier` variable is a function of the `secondsToMaxMultiplier` value, and the return value is a function of `fractionOfMaxMultiplier` and `maxMultiplier`. The `getOutstandingRewards` function returns a value called `newUserRewards`, which represents the rewards that a user has received. If this function is called immediately before adjusting the `maxMultiplier` or

other words, adjusting these parameters will immediately affect the outstanding rewards received by a user.

Updating the `baseEmissionRate` does not have any immediate impact on outstanding rewards because within the `baseRewardPerToken` function, `getCurrentTime()` will be equal to `stakingToken.lastUpdateTime` since the `_updateReward` function is called by the `configureStakingToken` function prior to updating the parameters. Thus, the `baseRewardPerToken` function will return `stakingToken.rewardPerTokenStored`, not causing any immediate impact on the outstanding rewards.

Consider implementing a mechanism to checkpoint user rewards such that parameter changes do not retroactively impact outstanding rewards.

Update: *Acknowledged, not resolved. Documentation was added to the* `configureStakingToken` *function in pull request #55 with commit 2110ce2. The documentation clarifies the consequences of updating the* `maxMultiplier` *and* `secondsToMaxMultiplier` *parameters, however no changes were made to contract to prevent parameter changes from impacting outstanding user rewards. The UMA team stated:*

> *We decided not to implement this fix and instead are willing to live with the fact that the admin has full control over the contract's rewards and should set the stakingToken configuration*

> *contract so doing so by changing stakingToken configs doesn't strike us as a different risk profile.*

Low Severity

Dangerous maximum values for reward calculations

The `configureStakingToken` function in the `AcceleratingDistributor` contract contains hard-coded upper limits that ensure `maxMultiplier` is set below `1e36` and `baseEmissionRate` is set below `1e27`. However, these limits are many orders of magnitude above the intended operating parameters for the contract.

According to the Across documentation for rewards locking, the value of `maxMultiplier` should never exceed `3e18` for the initial reward locking program; `1e36` is many orders of magnitude larger.

The rewards locking documentation also states that the initial token supply for rewards is 75,000,000 ACX tokens. This equates to `7.5e25` wei, but the upper limit of `1e27` on `baseEmissionRate` allows a token emission rate *per second* that is two orders of magnitude higher than the total supply to be emitted.

Through an accidental or malicious administrative action, `maxMultiplier` and/or `baseEmissionRate` could be set to values that rapidly deplete the rewards pool, allowing some

Consider setting the upper limits for `maxMultiplier` and `baseEmissionRate` much closer to the expected maximum operating values. For example, setting upper limits no more than 1 order of magnitude above the maximum expected values reduces risk while still allowing some flexibility to change the terms of the rewards program in the future.

Update: *Partially resolved in pull request #4319 with commit 2e9794a. The* `maxMultiplier` *upper limit was reduced from* `1e36` *to* `1e24`*, which allows for multiples of up to 1000000x the base reward rate, and the* `baseEmissionRate` *upper limit was reduced from* `1e27` *to* `1e24`*, which allows for up to 1 million tokens/second to be emitted. These limits are substantially lower than the previous values, but are still high enough that security is being traded for flexibility.*

Lack of input validation

The `AcceleratingDistributor` contract includes some functions that do not properly validate their input parameters. Specifically:

- The `stakeFor` function does not check that the `beneficiary` address is not the zero address.
- The `unstake` and `_stake` functions do not verify that the input `amount` is greater than zero.

the input arguments for the cases listed above.

Update: *Resolved in pull request #47 with commit d38d4cd.*

Missing or incomplete docstrings

Several functions and structs in `AcceleratingDistributor` contract lack complete documentation:

- The members of the `UserDeposit` and `StakingToken` structs are undocumented.
- The `constructor` is undocumented.
- The `getCurrentTime` function is undocumented.
- The `getAverageDepositTimePostDeposit` function is missing a docstring for the `amount` parameter.
- The `_stake` function is undocumented.

Incomplete documentation hinders reviewers' understanding of the code's intention, which is fundamental to correctly assess not only security, but also correctness. Additionally, docstrings improve readability and ease maintenance. They should explicitly explain the purpose or intention of the functions, the scenarios under which they can fail, the roles allowed to call them, the values returned and the events emitted.

Format (NatSpec).

Update: Resolved in pull request #53 with commit 95ae7ff.

Potential underflow error

In the `AcceleratingDistributor` contract, there is no lower bound on the `maxMultiplier` for a staked token. If a value lower than `1e18` is used, the return statement of the `getUserRewardMultiplier` function will fail due to underflow when evaluating `stakingTokens[stakedToken].maxMultiplier - 1e18` even though the return value has a lower bound of 0. This would prevent users from staking tokens, claiming rewards, and withdrawing their staked tokens.

Consider refactoring the return statement of the `getUserRewardMultiplier` function such that it cannot fail due to underflow.

Update: Resolved in pull request #48 with commit 7bd053a. A lower limit of `1e18` has been placed on the `maxMultiplier` value.

Notes & Additional Information

Event parameter will always be zero

in the codebase where the `RewardsWithdrawn` event is emitted, consider removing the redundant event parameter.

Update: *Resolved in pull request #49 with commit f50c6ec.*

Misleading documentation

In the `AcceleratingDistributor` contract, the following instances of misleading documentation were found:

- Lines 165-166: The comment says "if the token is not enabled for staking then we simply send back the full amount of tokens that the contract has", but it is the `lastUpdateTime` value that is checked, not the `enabled` value. Consider changing "if the token is not enabled" to "if the token has not been initialized".

- Within the `across-token/test` directory, the `constants.ts` file uses a `maxMultipler` value of 5 for testing. The associated comment says "At maximum recipient can earn 5x the base rate". However, the Across documentation for reward locking states that 3 is the maximum multiplier value. To avoid confusion, consider updating the test to match the official documentation.

> *This is currently not implemented, because the current maxMultiplier of 3 represents current Across LP staking policy, rather than the technical constraints of the underlying contract.*

Inconsistent ordering

In the `AcceleratingDistributor` contract, ordering generally follows the recommended order in the Solidity Style Guide, which is: type declarations, state variables, events, errors, modifiers, functions. However, within the contract, the event definitions deviate from the style guide, bisecting the functions. Additionally, the `struct` definitions occur after the `rewardToken` state variable is defined.

Furthermore, the ordering of functions is generally well structured with the exception of the `public view` `getCurrentTime` function, which is defined outside of the `view` functions section.

To improve the project's overall legibility, consider standardizing ordering throughout the codebase, as recommended by the Solidity Style Guide.

Update: *Resolved in pull request #52 with commit be635c4. However, in restructuring the layout of the contract, the docstrings for the* `getCurrentTimestamp` *and the* `constructor` *were removed.*

Consider correcting the following typographical errors in `AcceleratingDistributor.sol`:

- Line 122: "dont" should be "don't".
- Line 123: "loose" should be "lose".
- Line 158: "have. i.e" should be "have, i.e.".
- Line 159: "cant" should be "can't".
- Line 165: "if the token" should be "If the token".
- Line 214: "if underflow" should be "on underflow" or "if underflow occurs".
- Line 214: "cant" should be "can't".
- Line 312: "any internal logic..." line is a copy-paste error from line 294.
- Line 331: "users staking duration" should be "user's staking duration" .
- Line 349: "last users average deposit time" should be "user's last average deposit time".
- Line 352: Add a space before `@return`.
- Line 352: "users average deposit time" should be "user's average deposit time".
- Line 359: "users new average deposit time" should be "user's new average deposit time".

Update: *Partially resolved in pull request #51 with commit 5254dec. The following typographical errors are still present:*

- *Line 188: "i.e" should be "i.e.".*
- *Line 396: "last user's" should be "user's".*

No critical or high severity issues were found. Some recommendations were proposed to follow best practices and reduce the potential attack surface. We also recommend implementing monitoring and/or alerting functionality (see Appendix).

Appendix

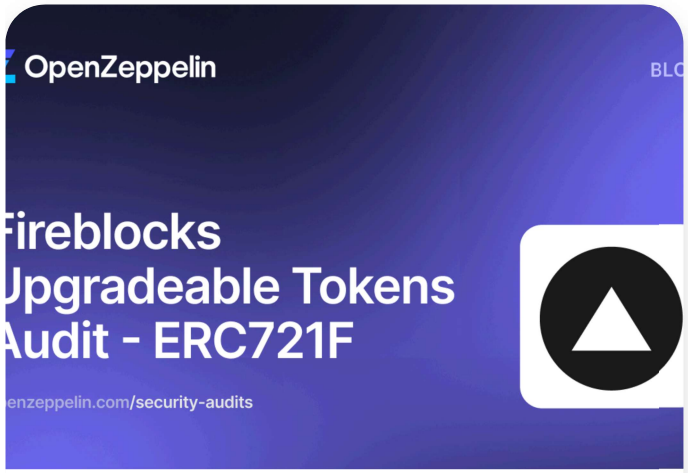
Monitoring Recommendations

While audits help in identifying potential security risks, the UMA team is encouraged to also incorporate automated monitoring of on-chain contract activity into their operations. Ongoing monitoring of deployed contracts helps in identifying potential threats and issues affecting the production environment.

- Per the Across documentation, the reward locking program has an initial supply of 75,000,000 ACX reward tokens. The cumulative rewards will eventually approach this limit, requiring some action to be taken to reduce reward emission or replenish the reward pool with additional tokens. Consider monitoring the balance of ACX tokens remaining in the `AcceleratingDistributor` contract for planning purposes. Additionally, to ensure the outstanding rewards do not exceed the remaining balance of reward tokens available, consider periodically comparing the contract's ACX token balance to the total outstanding rewards across all accounts, reported per user via the `getOutstandingRewards` function.

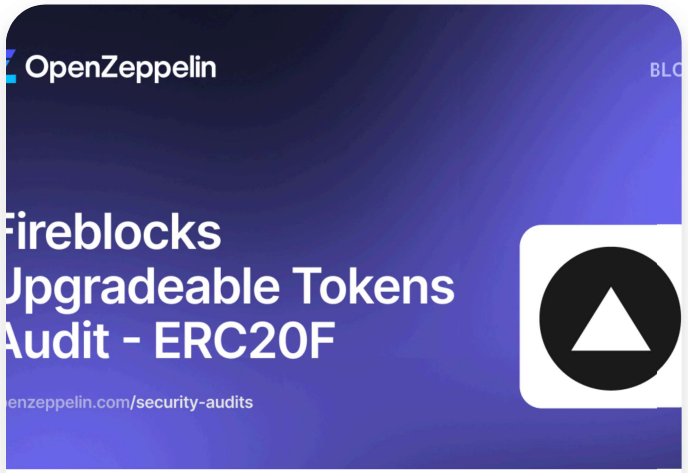
`RecoverToken` events.

Related Posts



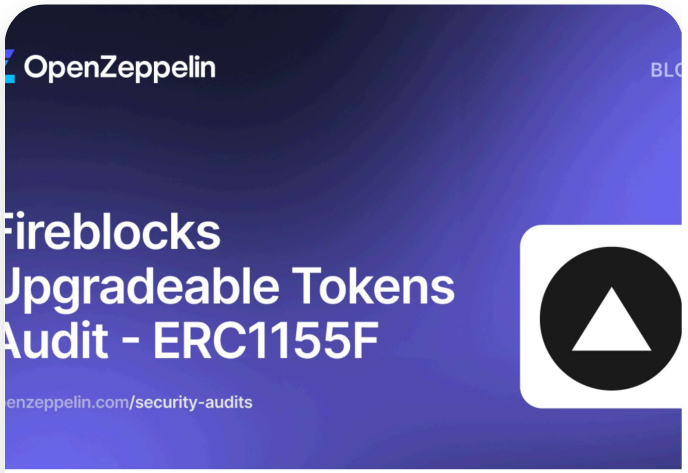
Fireblocks Upgradeable Tokens Audit - ERC721F

We audited the Fireblocks repository at commit e7003dfb. Smart contract Audit.



Fireblocks Upgradeable Tokens Audit - ERC20F

We audited the Fireblocks repository at commit e7003dfb. Smart Contract Audit.



Fireblocks Upgradeable Tokens Audit - ERC1155F

We audited the Fireblocks repository at commit e7003dfb. Smart Contract Audit.