MixBytes()

Open Enterprise Rewards Smart Contract Audit Report

Introduction

General provisions

Aragon is software allowing to freely organize and collaborate without borders or intermediaries. Create global, bureaucracy-free organizations, companies, and communities.

Autark is an Aragon Network organization building open source tools that serve digital cooperatives and aims to revolutionize work by leveraging the corresponding challenges.

With this in mind, MixBytes team was willing to contribute to Aragon ecosystem development by providing security assessment of the Open Enterprise Suite smart contracts created by Autark, as well as the StandardBounties and AragonApp smart contracts.

Scope of the audit

Code written by: Autark

Audited commit: Rewards.sol version 1508acf.

Security Assessment Principles

Classification of Issues

- CRITICAL: Bugs that enable theft of ether/tokens, lock access to funds without possibility to restore it, or lead to any other loss of ether/tokens to be transferred to any party (for example, dividends).
- MAJOR: Bugs that can trigger a contract failure, with further recovery only possible through manual modification of the contract state or contract replacement altogether.
- WARNINGS: Bugs that can break the intended contract logic or enable a DoS attack on the contract.
- COMMENTS: All other issues and recommendations.

Security Assessment Methodology

The audit was performed with triple redundancy by three auditors.

Stages of the audit were as follows:

- “Blind” manual check of the code and model behind the code
- “Guided” manual check of the code
- Check of adherence of the code to requirements of the client
- Automated security analysis using internal solidity security checker
- Automated security analysis using public analysers
- Manual by-checklist inspection of the system
- Discussion and merge of independent audit results
- Report execution

Detected Issues

CRITICAL

1. Rewards.sol#L84

There is no check that the user has not already claimed his reward. As a result, anybody with some reference token amount can claim all reward tokens from the vault. We recommend adding the check.

Fixed at 7dab770

MAJOR

Not found

WARNINGS

1. Rewards.sol#L211

There are no blockchain-enforced guarantees that the vault will be able to distribute the reward in the future (i.e. that the vault will remain solvent). Moreover, there are no guarantees that the app will still have access to the vault in the future.

Acknowledged

2. Rewards.sol#L252

The current implementation of one-time rewards would work only if the balances of the reference token holders and the total supply were monotonically increasing functions. This requirement is not provided by the MiniMeToken. Strictly speaking, the code does not adhere to the Aragon Planning App paper. The simplest way to solve the problem is to implement an ancestor of the MiniMeToken which prevents token transfers (except distribution during creation) and token burning.

Fixed at Rewards.sol#L213-L215

3. Rewards.sol#L256

As an example of the previous warning: suppose a user received newly minted reference tokens, but the total supply remains unchanged (some tokens were destroyed). As a result, the user will get zero payout.

Acknowledged

Client: minting can be disabled in the Token Manager App.

4. Rewards.sol#L253

Check that end balance >= start balance and end supply >= start supply must be used (or `SafeMath::sub`).

`balance` could overflow if somebody spends his tokens during the reward period.

`supply` could overflow if the controller destroys some reference tokens during the reward period (see `MiniMeToken::destroyTokens`).

Fixed at 7dab770

5. Rewards.sol#L94

Even if the vault held enough tokens to send a payout, the payout would not be performed. This issue can affect the last receiver of the reward. We recommend changing the condition to `>=`.

Fixed at 7dab770

6. Rewards.sol#L256

`SafeMath::mul` should be used to avoid overflow during computation of `rewardAmount`.

Fixed at 7dab770

COMMENTS

1. Rewards.sol#L38

Struct could be optimized for saving gas on reward insertion:

- `uint256 value` - unused
- `uint256 occurrences` - since the `MAX_OCCURRENCES = uint8(42)` type could be changed to uint8. Also, this struct member is not used in `getReward`. Could it be removed?
- `uint256 duration`, `uint256 delay` - could be changed to `uint64` or even `uint32` since it is a number of blocks
- `uint256 blockStart` - could be changed to uint64

Moreover, all changed members (and the existing members with the `address` and `bool` types) should be grouped into bunches of 32 bytes.

Acknowledged

2. Rewards.sol#L45

Typo in a word `occurrences`.

Fixed at 7dab770

3. Rewards.sol#L124, Rewards.sol#L184

Duration is not a timestamp or time, but a number of blocks.

Fixed at 7dab770

4. Rewards.sol#L125, Rewards.sol#L186

Delay is not a timestamp or time, but a number of blocks.

Fixed at 7dab770

5. Rewards.sol#L84

Despite the fact that there is no dangerous side effects of calling `claimReward` right now, we recommend adding the explicit modifier `isInitialized` to this function to avoid them in the future.

Fixed at 7dab770

6. Rewards.sol#L231

Check could be moved to the checks block at the beginning of the function to save gas in some situations.

Fixed at 7dab770

7. Rewards.sol#L189

Check that _duration > 0 could be added.

Fixed at 7dab770

8. Rewards.sol#L109 ,

Rewards.sol#L131

We recommend adding the explicit check `isInitialized` .

Fixed at 7dab770

9. Rewards.sol#L59

We recommend at least using a mapping instead of an array (as it is done in Aragon apps). For more details, see aragon/aragon-apps#68 or navigate to #11.

Fixed at 7dab770

10. Rewards.sol#L55 ,

Rewards.sol#L56

Rewards.sol#L59

Rewards.sol#L61

Explicit positions of the storage data are not used. This can make migration of the existing contract instance to a new code version cumbersome. A simple example of storage data explicit positions can be seen here.

Acknowledged

11. Rewards.sol#L253 ,

Rewards.sol#L254 ,

Rewards.sol#L256 ,

Rewards.sol#L87 ,

Rewards.sol#L225 ,

Rewards.sol#L239

We recommend using a `SafeMath` library to prevent overflows and underflows.

Mostly fixed at 7dab770

12. Rewards.sol#L85

We recommend adding the explicit check that the reward exists.

Fixed at 7dab770

13. Rewards.sol#L50

There is no need to have the `claimed` field. We can calculate `claimed` as `timeClaimed != 0`.

Fixed at 7dab770

14. Rewards.sol#L237 ,

Rewards.sol#L247

We recommend marking the Reward parameter with a storage specifier to skip copying the value.

Fixed at 7dab770

15. Rewards.sol#L231

Similarly to dividend payouts in stock assets, after reward creation (at the moment `reward.blockStart + reward.duration` or some blocks before this moment) a user can accumulate a large amount of reference tokens, and right after the `reward.blockStart + reward.duration` moment, dispose of them. At the end, even though the user held the tokens for minimal time, he still received the reward.

Acknowledged

16. Rewards.sol#L90 ,

Rewards.sol#L100

We recommend reverting the transaction as soon as it is known that the reward amount is zero. Otherwise, the blockchain is polluted with the excess state and event.

Fixed at 7dab770

Comments on the dependencies

1. MiniMeToken.sol:463

An unchecked cast. Possible truncation of `_value` can go unnoticed. We suggest adding the `require(_value <= uint128(-1));` check.

2. MiniMeToken.sol:438

We recommend replacing this check with the assert `assert(_block >= checkpoints[0].fromBlock);`. The `getValueAt` code does not have the information to handle such cases, moreover, they are handled in the calling code. If the control reaches the condition and the latter evaluates to `true`, this will indicate a code inconsistency and should not be silenced with `return 0;`.

The same goes for the check at line 432.

3. MiniMeToken.sol

A lot of deprecation warnings during compilation.

CONCLUSION

The fixed contract doesn't have any vulnerabilities according to our analysis.