MixBytes()

Open Enterprise Allocations 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: Allocations.sol version 5be80e3.

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

Not found

MAJOR

Not found

WARNINGS

1. Allocations.sol#L463

After some standby period of the contract, all functions with the `transitionsPeriod` modifier will fail with an error due to the lack of gas for creating all periods.

We recommend adding a separate function that will create the missing periods. This function must have the `limit` parameter allowing to create periods in several calls. Also, any unauthorized users should be able to call this function.

Fixed at 26e6d37

2. Allocations.sol#L417

There is no limit on the number of candidates for rewards. If there are a lot of them, then the transaction will end with an error due to lack of gas. In this case, `Payout` will not be added as well.

Fixed at Allocations.sol#L419

3. Allocations.sol#L298

Budget can be allocated to a non-existent account that will be created in the future. We recommend preventing such behaviour and checking for the account.

Fixed at 26e6d37

4. Allocations.sol#L527

The cycle will be aborted and the transaction will be rolled back if there are `candidateAddresses` with `supports` equal to 0. A number of measures in the comments below will help prevent the problem. As an additional measure, you can explicitly check for `supports` in this loop.

Fixed at 26e6d37

5. Allocations.sol#L358

New periods are not being initialized. We suggest adding the `transitionsPeriod` modifier.

Fixed at 26e6d37

6. Allocations.sol#L528

Consider the situation: `_candidateIndex` candidate takes away full payment from some `_payoutId` allocated to him at the moment. Earlier than the `_nextPaymentTime (_accountId, _payoutId, _candidateIndex)` time, an account with the `EXECUTE_ALLOCATION_ROLE` rights is trying to call the `runPayout` transaction. This transaction will end in error on line 528, that in turn will not allow all other candidates to be paid.

Fixed at Allocations.sol#L567

7. Allocations.sol#L425

When calling `runPayout`, `paid` should be passed externally, otherwise this variable will not reflect the true amount of payments in the transaction.

Fixed at a0a5c6c

COMMENTS

1. Allocations.sol#L40-L45

Constants can be calculated in advance.

Fixed at 26e6d37

2. Allocations.sol#L56

Gas consumption required to save `Payout` can be reduced. Place `uint64 recurrences`, `uint64 period`, `uint64 startTime` and `bool distSet` one after another, and they will occupy one storage slot.

Fixed at 26e6d37

3. Allocations.sol#L70

Gas consumption required to save `Account` can be reduced. Place `uint64 payoutsLength`, `address token` and `bool hasBudget` one after another, and they will occupy one storage slot.

Fixed at 26e6d37

4. Allocations.sol#L94

There's no need to use the `uint64` key instead of the `uint` one as element adding requires the same amount of gas as `uint`.

Acknowledged

5. Allocations.sol#L95

Allocations.sol#L97

`AccountsLength`, `periodsLength` and `periodDuration` variables can be placed one after another to save some gas (since they will occupy one storage slot).

Fixed at 26e6d37

6. Allocations.sol#L118

We recommend explicitly checking the given precondition in the code calling `assert` or `require`.

Acknowledged

7. Allocations.sol#L156

Allocations.sol#L170

Allocations.sol#L183

Allocations.sol#L194

Allocations.sol#L206

Allocations.sol#L298

Allocations.sol#L314

Allocations.sol#L330

Allocations.sol#L347

Allocations.sol#L358

Allocations.sol#L391

We recommend adding a check for the account (use `require` with a separate reason indicating the absence of an account). Use a modifier to prevent calls to non-existent accounts.

Almost all fixed at 26e6d37

8. Allocations.sol#L331

Allocations.sol#L348

Allocations.sol#L524

We recommend checking for `Payout` basing on `_accountId` and `_payoutId`. This saves gas and storage and seems like a more logical approach.

Acknowledged

9. Allocations.sol#L170

Allocations.sol#L206

We recommend adding a check for payout (`require` with a separate reason for the absence of payout). Use a modifier to prevent calls to non-existent payout.

Fixed at 26e6d37

10. Allocations.sol#L427

We recommend checking that `_candidateId` does not go beyond `supports` boundaries.

Fixed at 26e6d37

11. Allocations.sol#L427

It makes sense to return from the function if `individualPayout` turned out to be 0.

Fixed at 26e6d37

12. Allocations.sol#L496

It makes sense to add the condition `amount > 0`.

Fixed at 26e6d37

13. Allocations.sol#L207

We recommend adding a check for `_idx` index.

Fixed at 26e6d37

14. Allocations.sol#L50

It is reasonable to make `MAX_SCHEDULED_PAYOUTS_PER_TX` adjustable within 1 .. 100 range to be ready for possible changes in the block gas limit and gas consumption by token transfers in the future.

Acknowledged

15. Allocations.sol#L74

`Account.balance` is not used.

Fixed at 26e6d37

16. Allocations.sol#L60

`metadata` is not used.

Fixed at 26e6d37

17. Allocations.sol#L88

Allocations.sol#L89

`firstTransactionId` and `lastTransactionId` are not used at all.

Fixed at 26e6d37

18. Allocations.sol#L57

`candidateKeys` is not used at all.

Fixed at 26e6d37

19. Allocations.sol#L82

`income` is not used at all.

Fixed at 26e6d37

20. Allocations.sol#L81

There is no need for mapping, as the account has only one token. In total, `AccountStatement` is reducible to `uint256 expenses`.

Acknowledged

21. Allocations.sol#L392

This and similar checks for sufficient funds are purely informative and do not give any guarantees. On the one hand, it is impossible to pay out more funds than there are in the vault. On the other hand, there are many reasons why funding may not be sufficient, despite the initial checklist. For example, another account may pay out all vault funds, or access to vault may be subsequently limited by access rights. There is no mechanism for reserving tokens/ether for a certain payment.

Acknowledged

22. Allocations.sol#L392

This check should also involve `_recurrences`, as at the moment it does not reflect the full amount of future payments.

Fixed at 26e6d37

23. Allocations.sol#L76

If it is not planned to set the budget for the account equal to 0, then `hasBudget` can be omitted, as the `budget != 0` comparison will be equivalent to `hasBudget`.

Acknowledged

24. Allocations.sol#L153

Allocations.sol#L167

Allocations.sol#L182

Allocations.sol#L191

Allocations.sol#L203

Allocations.sol#L220

We recommend adding the `isInitialized` modifier.

Fixed at 26e6d37

25. Allocations.sol#L128

Allocations.sol#L246

Incorrect description of functions. We recommend updating the comments.

Fixed at 26e6d37

26. Allocations.sol#L428

There is no need to emit the `Time` event here, because it will be emitted inside `_nextPaymentTime`.

Fixed at 26e6d37

27. Allocations.sol#L376

We recommend adding extra checks:

- that the length of `_candidateAddresses` is equal to the length of `_supports`
- `_amount > 0`

Acknowledged

28. Allocations.sol#L406

Zero initialization is unnecessary.

Fixed at 26e6d37

CONCLUSION

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