MixBytes()

AUTARK

SMART CONTRACT
AUDIT REPORT

NOVEMBER 5
2019

FOREWORD TO REPORT

A small bug can cost you millions. **MixBytes** is a team of experienced blockchain engineers that reviews your codebase and helps you avoid potential heavy losses. More than 10 years of expertise in information security and high-load services and 15 000+ lines of audited code speak for themselves. This document outlines our methodology, scope of work, and results. We would like to thank **Autark** for their trust and opportunity to audit their smart contracts.

CONTENT DISCLAIMER

TABLE OF CONTENTS

01 INTRODUCTION TO THE AUDIT

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

The scope of the audit included:

1. **The AddressBook contract**
2. **The AragonApp contract**
3. **The Allocations contract**
4. **The RewardsApp contract**
5. **The StandardBounties contract**
6. **The Projects contract**
7. **The DotVoting contract**

02 SECURITY ASSESSMENT PRINCIPLES

CLASSIFICATION OF ISSUES

CRITICAL

Bugs leading to Ether or token theft, fund access locking or any other loss of Ether/tokens to be transferred to any party (for example, dividends).

MAJOR

Bugs that can trigger a contract failure. Further recovery is possible only by manual modification of the contract state or replacement.

WARNINGS

Bugs that can break the intended contract logic or expose it to DoS attacks.

COMMENTS

Other issues and recommendations reported to/acknowledged by the team.

SECURITY ASSESSMENT METHODOLOGY

The audit was performed by 2 auditors. Stages of the audit were as follows:

1. “Blind” manual check of the code and its model
2. “Guided” manual code review
3. Checking the code compliance with customer requirements
4. Automated security analysis using the internal solidity security checker
5. Automated security analysis using public analyzers
6. Manual checklist system inspection
7. Discussion of independent audit results
8. Report preparation

03 ADDRESS BOOK AUDIT REPORT

Code written by: **Autark**

Audited commit: **AddressBook.sol**

DETECTED ISSUES

CRITICAL

Not found.

MAJOR

Not found.

WARNINGS

Not found.

COMMENTS

1. AddrtessBook.sol#L103

We recommend adding the explicit check `isInitialized`.

2. AddressBook.sol#L91

There is a constant for this kind of error message - `ERROR_CID_MALFORMED`. We recommend factoring out the entire check as a modifier.

3. AddressBook.sol#L33

There is no way to get the entire list of addresses stored in the address book. An array can be added to keep track of all present addresses. When an entry is deleted from the address book, the last array element can replace the deleted element to prevent array fragmentation.

4. AddressBook.sol#L33

Explicit positions of the storage data are not used, which can complicate migration of the current contract instance to a new one. A simple example of explicit storage data positions can be seen **here**.

5. AddressBook.sol#L62

It is expected that structured content objects for the entries will be stored in IPFS. Users and developers should keep in mind that IPFS does not guarantee data availability. After some time unused data is removed from IPFS unless explicitly pinned by some node.

6. AddressBook.sol#L64

IPFS addresses have the form of `<encoding>.encode(multihash(<digest>, <function>))`, which makes the check of line 64 valid only for base58 encoding of sha256 hashes.

7. AddressBook.sol#L86

Similarly to the `_cid` argument of the `removeEntry` function, an additional argument `oldCid` and a content check can be introduced to prevent race conditions and ensure that the updated entry was to be updated.

8. AddressBook.sol#L62
AddressBook.sol#L75
AddressBook.sol#L86

The functions can be marked as `external` to **save some gas**.

9. AddressBook.sol#L33

IPFS-address can be stored as an array of bytes instead of a string. A more “smart” check may be introduced for the adding/updating code (to dynamically identify the hash function used and the necessary input size).

Status of comments:

The comments were acknowledged and partially fixed by the client’s team.

CONCLUSION

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

04 ARAGON APP AUDIT REPORT

Code written by: **Aragon One**

Audited commit: **AragonApp.sol**

HIGH-LEVEL OVERVIEW

AragonApp is a base contract for DApp development. It is linked to a so-called kernel. The kernel stores the addresses of current app implementations which are accessed via proxies. Also, the kernel provides access to the ACL subsystem. The kernel is the coordination center of an app system.

AragonApp provides `auth` and `authP` modifiers which are thin wrappers over `IKernel.hasPermission` function. These modifiers are used to check permissions when accessing app `functions`.Besides, AragonApp provides `RecoveryVault` functionality to recover tokens/ether sent to the app.

AragonApp uses a proxy mechanism. This approach has several consequences.

Firstly, proxies have to be initialized (you can't use a constructor in case of a proxy). The code of proxy implementation is usually made uninitializable (petrified) to prevent issues as the one occurred with Parity wallets.

Secondly, code implementation versions must be consistent while accessing the storage. This is achieved with the help of `UnstructuredStorage` via direct access to storage slots, the addresses of which are calculated based on fully qualified field name hashes.

Thirdly, the addresses of current implementations must be kept in the kernel.

AragonApp can run EVMScripts. A script executor is typically determined by the first bytes of the script. The addresses of available executors are stored in the script registry app. In the most straightforward case, the ACL subsystem tells about “the doer” (who), the role (what he can do), and applications (where he can perform a role). You can go further and add rules to the permission. Rules are expression trees encoded in arrays. A number of variables are exposed to rules at invocation time (for instance, call-specific values, block parameters, oracles, etc.).

DETECTED ISSUES

CRITICAL

Not found.

MAJOR

Not found.

WARNINGS

1. AragonApp.sol#L56

The function `canPerform` calls `dangerouslyCastUintArrayToBytes` that rewrites its argument. So, the argument `_params` of `canPerform` is also rewritten. All the examples in the documentation use helpers `arr` with `canPerform` and `authP`. However, somebody may avoid using this helper (for example, if he already has an array of params).

We recommend returning the parameter to its original state by calling `dangerouslyCastBytesToUintArray`.

https://gist.github.com/quantum13/968399047d768dde554d7ae1379e6452

Status:

ACKNOWLEDGED

COMMENTS

1. ReentrancyGuard.sol#L25

The reentrancy guard can be optimized using an incrementing value (**example**). This will yield 2-3 times gas savings in some cases.

2. ACL.sol#L245

We recommend at least adding the information about function side effect (rewriting argument `_how`) to the function documentation. At most, return the parameter to its original state.

CONCLUSION

Overall code quality is very high. There was only one issue identified that might lead to errors on rare occasions.

05 ALLOCATIONS AUDIT REPORT

Code written by: **Autark**

Audited commit: **Allocations.sol**

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.

Status:

FIXED at **26e6d3766393ed2d12fc57471b56d42c4a680fef**

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.

Status:

FIXED at **f85f07565cb6d57161da0252ca7df2c473c3fcfd**

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.

Status:

FIXED at 26e6d3766393ed2d12fc57471b56d42c4a680fef

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.

Status:

FIXED at 26e6d3766393ed2d12fc57471b56d42c4a680fef

5. Allocations.sol#L358

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

Status:

FIXED at 26e6d3766393ed2d12fc57471b56d42c4a680fef

6. Allocations.sol#L528

Consider the situation:

1. `_candidateIndex` candidate takes away full payment from some `_payoutId` allocated to him at the moment.
2. 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.

Status:

FIXED at **5be80e35f6e8e2c58f2b1b0f95f43baf40886507**

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.

Status:

FIXED at **a0a5c6cfb739395994dc4710172a8b107997e4bf**

COMMENTS

1. Allocations.sol#L40-L45

Constants can be calculated in advance.

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.

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.

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`.

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).

6. **Allocations.sol#L118**

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

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.

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.

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.

10. **Allocations.sol#L427**

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

11. **Allocations.sol#L427**

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

12. **Allocations.sol#L496**

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

13. **Allocations.sol#L207**

We recommend adding a check for `_idx` index.

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.

15. **Allocations.sol#L74**

`Account.balance` is not used.

16. **Allocations.sol#L60**

`metadata` is not used.

17. **Allocations.sol#L88**
    Allocations.sol#L89

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

18. Allocations.sol#L57

`candidateKeys` is not used at all.

19. Allocations.sol#L82

`income` is not used at all.

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`.

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.

22. Allocations.sol#L392

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

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`.

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.

25. Allocations.sol#L128
Allocations.sol#L246

Incorrect description of functions. We recommend updating the comments.

26. Allocations.sol#L428

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

27. Allocations.sol#L376

We recommend adding extra checks:

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

Status of comments:

The comments were acknowledged and partially fixed by the client's team.

CONCLUSION

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

06 REWARDS APP AUDIT REPORT

Code written by: Autark

Audited commit: Rewards.sol

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.

Status:

FIXED at 7dab770e5bcf758268e51429e3702eb8305ce242

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.

Status:

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.

Status:

FIXED at **7dab770e5bcf758268e51429e3702eb8305ce242**

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.

Status:

ACKNOWLEDGED

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`).

Status:

FIXED at **7dab770e5bcf758268e51429e3702eb8305ce242**

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 `>=`.

Status:

FIXED at 7dab770e5bcf758268e51429e3702eb8305ce242

6. Rewards.sol#L256

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

Status:

FIXED at 7dab770e5bcf758268e51429e3702eb8305ce242

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.

2. Rewards.sol#L45

Typo in a word `occurrences`.

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

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

4. **Rewards.sol#L125**
   Rewards.sol#L186

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

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.

6. **Rewards.sol#L231**

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

7. **Rewards.sol#L189**

Check that `_duration > 0` could be added.

8. **Rewards.sol#L109**
   Rewards.sol#L131

We recommend adding the explicit check `isInitialized`.

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 **this** or navigate to #11.

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**.

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.

12. **Rewards.sol#L85**

We recommend adding the explicit check that the reward exists.

13. **Rewards.sol#L50**

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

14. **Rewards.sol#L237**
    Rewards.sol#L247

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

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.

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.

Status of comments:

The comments were acknowledged and partially fixed by the client’s team.

COMMENTS ON THE DEPENDENCIES

Written by: **Aragon One**

MiniMeToken.sol

1. 1.0.1 MiniMeToken.sol:463

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

The same warning also applies to **MiniMeToken.sol:463**.

2. 1.0.1 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. 1.0.1 MiniMeToken.sol

A lot of deprecation warnings during compilation.

Status of comments:

The comments were acknowledged and partially fixed by the client's team.

CONCLUSION

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

07 STANDARD BOUNTIES AUDIT REPORT

Code written by: **The Bounties Network**

Audited commit: **StandardBounties.sol**

DETECTED ISSUES

CRITICAL

1. StandardBounties.sol#L381

Contributions are not tagged as `refunded`. Funds can be re-withdrawn from the bounty balance by the `refundContribution` requests from contributors who have already received funds during the `refundContributions` call.

An example of the attack vector:

1. A contributor `C1` makes a contribution of 10 Ether with contribution `id = 0`.
2. A contributor `C2` makes a contribution of 10 Ether with contribution `id = 1`.
3. An issuer `I1` that does not relate in any way to `C1` or `C2` issues a `refundContributions` call with the `_contributionIds` parameter being equal `[1]`.
4. As a result of this call, `C2` receives 10 Ether back. After the call, the balance of the bounty is 10 Ether.
5. `C2` issues a `refundContribution` call with the `_contributionId` parameter being equal to `1`. As a result of this call, `C2` receives the remaining 10 Ether.
6. Ultimately, `C2` received a double refund, and the contribution of `C1` was in fact transferred to `C2`. The balance of the bounty is 0, despite the fact that the issuer intended to refund only `C2`.

Status:

FIXED at **7c7dfc604a981a6be8cef64e06c5814c939dc2c1**

MAJOR

Not found.

WARNINGS

1. StandardBounties.sol#L237

The function returns nothing, although the function signature indicates that the function should return a `uint`.

Status:

FIXED at 7c7dfc604a981a6be8cef64e06c5814c939dc2c1

2. StandardBounties.sol#L407

If an issuer does not withdraw all funds from the bounty, this may cause inequality between contributors. Some of them may manage to withdraw the remaining funds, and others do not.

Status:

ACKNOWLEDGED

3. StandardBounties.sol#L376

The `_contributionIds [i]` is allowed to go beyond the bounties `[_bountyId] .contributions` array bounds. We recommend replacing comparison with the strict one.

Status:

FIXED at e79d8443097cab2472c91cfa6d91c23bee6f9869

4. **StandardBounties.sol#L372**
   StandardBounties.sol#L402
   StandardBounties.sol#L506
   StandardBounties.sol#L639

As going out of array bounds in these functions is not controlled, `metaTxRelayer` can pass the `0` address as `sender` and successfully pass the access checks. Thus, an attacker who gained access to `metaTxRelayer` can bypass access control in some cases.

As transaction relayer is out of the audit scope, it is not possible to assess the security risk in this case. However, we recommend checking the digital signature of the account involving the relay transaction either in the contract code or in the Transaction relayer code. Function selector and all function parameters must be signed with a digital signature.

Status:

ACKNOWLEDGED

COMMENTS

1. **StandardBounties.sol#L19**

Gas consumption can be optimized

* `deadline` - `uint64` may be used for timestamp
* `tokenVersion` - `uint8` may be used (with `0` constant for Ether, `1` for ERC-20, `2` - for ERC-721).
* `deadline`, `hasPaidOut` and `tokenVersion` should be placed after `token` to use one storage slot for these fields.

2. **StandardBounties.sol#L60**

Gas consumption can be reduced by using **ReentrancyGuard.sol**

Explanation can be found here:

* **https://github.com/OpenZeppelin/openzeppelin-solidity/issues/1056**
* **https://github.com/OpenZeppelin/openzeppelin-solidity/pull/1155**

3. **StandardBounties.sol#L303**
   StandardBounties.sol#L816

Due to the check at the stage of bounty creation, `tokenVersion` can have only `0`, `20`, `721` values. If there are no logical errors in the contract, control never reaches the given code lines. We recommend using `assert` instead of `revert` for checking code consistency.

4. **StandardBounties.sol#L275**

We recommend adding a check that the deadline has not passed. Otherwise, contribution is meaningless as such.

5. **StandardBounties.sol#L198**

We recommend adding a check that the deadline has not passed.

6. **StandardBounties.sol#L111**
   StandardBounties.sol#L120
   StandardBounties.sol#L130
   StandardBounties.sol#L140

We recommend adding checks that the specified bounty exists and the array bounds are not exceeded in this modifier and alike. Otherwise, access control modifiers will be satisfied when passing `_sender` equal to `0`.

In some functions, array bound excess is checked separately, in others it is not checked at all. In any case, we believe that these checks should be in access control modifiers.

CONCLUSION

Overall code quality is rather high. However, there are some flaws, including the critical one, which was successfully fixed by the original contract authors, the Bounties Network, with a new contract version deployed.

08 PROJECTS AUDIT REPORT

Code written by: **Autark**

Audited commit: **Projects.sol**

DETECTED ISSUES

MAJOR

1. Projects.sol#L433

`_tokenAmounts` - number of tokens the user will receive as a reward through `StandardBounties` - is passed in the code of `reviewSubmission`. If the number of tokens is less than stated in `issue.bountySize`, the remaining tokens cannot be withdrawn:

* the rest of the reward **cannot be issued**
* or **removed altogether**, as withdrawal of the initial funds amount will fail.

Before calling `acceptFulfillment`, make sure that the sum of all the values in the `_tokenAmounts` array equals `issue.bountySize`.

Status:

FIXED at **59cfcc6e0df2b014db432a6ba67ec394376c223b**

WARNINGS

1. Projects.sol#L340

When a repository is deleted, funds in open bounties related to repository issues become (at least temporarily) blocked.

We recommend you keep score of open bounty repositories and prohibit deleting them if this entails a loss of funds.

Status:

FIXED at **59cfcc6e0df2b014db432a6ba67ec394376c223b**

2. Projects.sol#L447

If this function is called for a non-existing issue, this will in fact be the `acceptFulfillment` function call for `_bountyId = 0`.

We suggest checking that the issue passed in call parameters really exists.

Status:

FIXED at **59cfcc6e0df2b014db432a6ba67ec394376c223b**

3. Projects.sol#L485

If this function is called for a non-existing issue, this will in fact be the `changeData` and `changeDeadline` function calls for `_bountyId = 0`.

We suggest checking that the issue passed in call parameters really exists.

Status:

FIXED at **59cfcc6e0df2b014db432a6ba67ec394376c223b**

4. Projects.sol#L870

Overwriting an arbitrary `Issue` is allowed if `_repoId` and `_issueNumber` point to an existing issue. In particular, that may lead to blocking of funds associated with the rewritten issue.

It's highly recommended to verify that the `_repoId` repository exists, and the issue `_issueNumber` does not exist yet. We suggest assigning a number to a new issue automatically.

Status:

FIXED at **59cfcc6e0df2b014db432a6ba67ec394376c223b**

5. Projects.sol#L953

There is no check that input data lookups do not go out of the string boundaries. Taking into account a lack of validation in `addBounties` and the fact that `addBounties` is `public` (input parameters are stored in memory), a description fragment or some other fragment of memory may return as a hash.

We recommend adding a check for exceeding the boundaries of the input string.

Status:

FIXED **at 59cfcc6e0df2b014db432a6ba67ec394376c223b**

6. Projects.sol#L408

The current value of `issue.assignee` is always replaced, regardless of `_approved`.

Make sure that this is the desired scenario.

Status:

FIXED at **59cfcc6e0df2b014db432a6ba67ec394376c223b**

7. Projects.sol#L421

The `AssignmentApproved` event is always emitted, regardless of the `_approved`.

Make sure that this is the desired scenario.

Status:

FIXED at **59cfcc6e0df2b014db432a6ba67ec394376c223b**

8. Projects.sol#L194

In fact, the `Bounties` contract address used by `Projects` is immutable. `settings.bountyAllocator` is mutable, but is not used by the contract.

At least, the given code comment is incorrect. Current behavior may differ from the planned one.

Status:

FIXED at **59cfcc6e0df2b014db432a6ba67ec394376c223b**

COMMENTS

1. Projects.sol#L368
Projects.sol#L549
Projects.sol#L693

We recommend checking that the passed `_repoId` and `_issueNumber` are present before proceeding. This will help prevent errors, including the user ones, at an early stage.

2. Projects.sol#L397
Projects.sol#L433

`issue.assignee` and `issue.assignmentRequests` are in no way associated with the `Bounties` contract and the reward payment. Make sure that this is the desired behavior.

3. Projects.sol#L99

`BountySettings` are not used in the contract but for read-write functions in the data structure. Make sure that this is the desired behavior.

4. Projects.sol#L156

The field value changes during contract operation but it is not used later on. Make sure that this is the desired behavior.

5. Projects.sol#L102

Hashes can be calculated in advance and values can be recorded as it is done in AragonApp.

6. Projects.sol#L203
Projects.sol#L209

Re-calling `isContract(_bountiesAddr)` is not beneficial and gas-consuming. We recommend removing an extra call.

7. Projects.sol#L269

We recommend using the current `bountySize` directly from `Bounties`. This step may be skipped if, after fixing major issue #1, `bountySize` will always be equal to the corresponding bounty balance in `Bounties`.

8. Projects.sol#L572
Projects.sol#L622

Input array length validation is missing. We recommend adding a check that all input array lengths are equal.

9. Projects.sol#L747

Division of the `_bountyRegistry` code into segments is unnecessary. In addition, last bytes (3) of **the last segment will not be captured due to truncation in the course of division**.

We recommend calculating the `keccak256` value of the entire `_bountyRegistry` code.

10. Projects.sol#L815

We suggest adding the `require(_tokenType == 20);` check.

11. Projects.sol#L880

Since `assignee` is set here, using `ETH` instead of `address(0)` is misleading. We recommend writing `address(0)` explicitly, or declaring the constant `NO_ASSIGNEE = address(0)`.

12. Impossibility to withdraw or use contribution

There is no way to withdraw or use contributions made directly through the `Bounties` contract.

One of the possible solutions is multiple `drainBounty` function calls.

13. Impossibility to get issuer for a repository

There is no way to get all issues for a given repository. Problems may arise while creating the issue list.

One of the possible solutions is using a repository issue counter.

14. Projects.sol#L705

This comment is not accurate because the function does not return the id of the added repository, but the boolean flag of the repository presence in the index.

15. Projects.sol#L150-L151
Projects.sol#L157
Projects.sol#L162

The specified structure fields are not used. However, we assume this does not lead to excessive gas consumption.

We recommend removing the unused fields.

16. Projects.sol#L679
Projects.sol#L509

The specified formal function parameters are not used. If any interface requires them, we recommend leaving only their type in the function declaration to emphasize that they are input on purpose and are not currently used.

17. Projects.sol#L572
Projects.sol#L622

The common code (comprising 90% of the given functions) may be moved to a separate function to avoid mistakes in the future.

18. Projects.sol#L205
Projects.sol#L302
Projects.sol#L312
Projects.sol#L584
Projects.sol#L752
Projects.sol#L882

We recommend eliminating the commented code fragments.

19. Projects.sol#L244
Projects.sol#L286
Projects.sol#L328

and so on

For the sake of uniformity, strings may be put into constants. Note that the use of constants slightly increases gas consumption.

20. Projects.sol#L412
Projects.sol#L414

It is allowed to overwrite the `AssignmentRequest` status, for which `reviewApplication` has already been performed. Make sure that this is the desired scenario.

Status of comments:

The comments were acknowledged and partially fixed by the client's team.

CONCLUSION

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

09 DOTVOTING AUDIT REPORT

Code written by: Autark

Audited commit: DotVoting.sol

DETECTED ISSUES

CRITICAL

Not found.

MAJOR

Not found.

WARNINGS

1. ADynamicForwarder.sol#L344
   ADynamicForwarder.sol#L474
   ADynamicForwarder.sol#L328
   ADynamicForwarder.sol#L454
   ADynamicForwarder.sol#L460

During the `copy` **function execution** extra 32 bytes are being copied. The copy function is probably being used incorrectly or contains an error. We recommend checking this behavior.

Status:

FIXED at 549073274f690f65aef2e01dd68dc25703cbffce

2. **DotVoting.sol#L157**
   DotVoting.sol#L277
   DotVoting.sol#L265
   DotVoting.sol#L278
   DotVoting.sol#L310
   DotVoting.sol#L332
   DotVoting.sol#L342
   DotVoting.sol#L352
   DotVoting.sol#L419
   DotVoting.sol#L465
   DotVoting.sol#L486

Access to a non-existent voting is allowed. We recommend adding a check that the voting id passed in the parameters exists.

Status:

FIXED at **b5dd6c3c879c7e4123b1e10108359bcccee57d8a**

3. **DotVoting.sol#L442**
   DotVoting.sol#L452

It is allowed to go beyond the boundaries of the `cKeys` array. We recommend adding a check that `i` does not go beyond the boundaries of the array.

Status:

FIXED at **b5dd6c3c879c7e4123b1e10108359bcccee57d8a**

4. **ScriptHelpers.sol#L74**
   ScriptHelpers.sol#L80
   ScriptHelpers.sol#L86
   ScriptHelpers.sol#L95
   ScriptHelpers.sol#L50
   ADynamicForwarder.sol#L456

We suggest controlling and preventing memory references from going beyond array boundaries for functions that deal directly with array memory. This will prevent errors at an early stage and reduce the risk of hard-to-diagnose memory corruption errors.

Status:

ACKNOWLEDGED

COMMENTS

1. DotVoting.sol#L213

The account with the ADD_CANDIDATES_ROLE rights is able to block the vote. It can either add a large number of candidates or a long _metadata, so that further processing (_executeVote in particular) will be impossible due to block gas restrictions.

2. DotVoting.sol#L134
DotVoting.sol#L143
DotVoting.sol#L154
DotVoting.sol#L264
DotVoting.sol#L277
DotVoting.sol#L296
DotVoting.sol#L330
DotVoting.sol#L341
DotVoting.sol#L351

We recommend adding the isInitialized modifier.

3. DotVoting.sol#L159-L160

Access to a non-existent option is allowed. We recommend adding a check that _candidateIndex is valid.

4. DotVoting.sol#L443
DotVoting.sol#L446
DotVoting.sol#L454

During the function execution, the value of voteInstance.totalParticipation can be maintained in a local variable and then written to storage at the end of the method to save gas.

5. DotVoting.sol#L504

There is no need to copy Action into memory and waste gas on reading the entire structure and allocating memory. We recommend replacing the memory qualifier with storage.

6. DotVoting.sol#L77-L80

In fact, there's no truncation of the data specified in the comment. We recommend updating the comment.

7. DotVoting.sol#L204

The data type is not a `string` but an `address`. Moreover, this field is used to generate internal keys in the code, and options with the same `_description` are not allowed. We suggest verifying that this behavior is appropriate and update the comment accordingly.

8. DotVoting.sol#L53
ADynamicForwarder.sol#L57-L59

Since explicit positions of the storage data are not used, migration of the current contract instance to a new one may be complicated. A simple example of explicit storage data positions can be seen **here**.

9. DotVoting.sol#L501

The incorrect comment was most likely copied from the function below. We recommend updating the comment.

10. DotVoting.sol#L212

A check proving that the vote is still open can be added. Otherwise, it makes no sense to write data to the blockchain.

11. DotVoting.sol#L376

The `Description` parameter in the comment is missing. We suggest adding it.

12. DotVoting.sol#L415

We recommend checking that the length of the `_supports` array does not exceed the number of voting options.

13. DotVoting.sol#L372

At the moment, the first parameter can only be an address. We recommend correcting the comment.

14. DotVoting.sol#L474
ADynamicForwarder.sol#L165
ADynamicForwarder.sol#L177
ADynamicForwarder.sol#L401
ADynamicForwarder.sol#L460
ADynamicForwarder.sol#L499

We recommend using the `SafeMath` library for performing subtraction.

15. **ADynamicForwarder.sol#L115**

During processing of the specified expression, a value truncation may occur. Voting options must not exceed 256, which is not controlled. However, the `keyArrayIndex` field is not used. We recommend either deleting the field or adding the according preliminary overflow check.

16. **ADynamicForwarder.sol#L44**
    ADynamicForwarder.sol#L45
    ADynamicForwarder.sol#L57
    ADynamicForwarder.sol#L120

Intermediate hashing of options can be omitted. In `Action.optionKeys` you can immediately write the addresses. Then, `optionAddresses` can be omitted. `Action.options` can have the `mapping (address => OptionState)` type.

17. **ADynamicForwarder.sol#L75**
    ADynamicForwarder.sol#L91
    ADynamicForwarder.sol#L107

Access to a non-existent `Action` is allowed. We recommend adding a check that `_actionId` is valid.

18. **ADynamicForwarder.sol#L76**

Access to a non-existent `OptionState` is allowed. We recommend adding a check that `_optionIndex` is valid.

19. **ADynamicForwarder.sol#L61**

We recommend adding the parameters for description and additional identifiers to the event to make sure there were no errors in the script when creating the vote.

20. **ADynamicForwarder.sol#L122**

This operation does not change the `actionInstance.optionKeys` value and may consume gas. We recommend removing the assignment.

21. **ADynamicForwarder.sol#L41-L42**

These fields are assigned but are not used further on.

22. **ADynamicForwarder.sol#L414**

There is no need to allocate 32 memory bytes as the value will be replaced in the next line.

23. ADynamicForwarder.sol#L363 ADynamicForwarder.sol#L380

The incorrect comment was most likely copied from the function below. We recommend updating the comment.

24. ADynamicForwarder.sol#L460

In case numerical values (for instance, `288` and `256`) are calculated in a sophisticated way, we do not recommend writing them into the code in a pre-calculated form. This greatly complicates the code maintainability and readability. Structural parameter changes will entail verification and/or recalculation of such values. Some of these recalculations may be missed by mistake.

We recommend calculating them in the code explicitly, based on the number and nature of the calldata parameters. This will lead to higher gas consumption, but reduce the likelihood of errors.

25. ADynamicForwarder.sol#L222 ADynamicForwarder.sol#L184

We recommend adding additional checks:

* the size of all arrays is the same
* no exceeding the boundaries of `infoString`

Status of comments:

The comments were acknowledged and partially fixed by the client’s team.

CONCLUSION

The level of contract security is high, and a rather difficult task was solved by limited Solidity means.

Please note that `ADynamicForwarder` contains several different incompatible offset types: calldata offset, bytes array offset, EVM memory offset. They are often used together and it is difficult to distinguish the offset type of a variable, and which function receives and returns this or that offset type. Unfortunately, Solidity does not allow for the derived types (in other languages, you can introduce derivatives of `uint` types and make implicit casts impossible, thereby preventing confusion and errors). As an alternative, we suggest at least explicitly describing in the documentation, variable names and parameters which offset types they should contain.

Also, we recommend controlling and preventing accessing the memory outside of array boundaries and accessing non-existent mapping elements.The **fixed contract** doesn’t have any vulnerabilities according to our analysis.

ABOUT MIXBYTES

MixBytes is a team of blockchain developers, auditors and analysts keen on decentralized systems. We build open-source solutions, smart contracts and blockchain protocols, perform security audits, work on benchmarking and software testing solutions, consult universities and enterprises, do research, publish articles and documentation.

Stack	Blockchains

JOIN US