MixBytes()

AragonApp Smart Contract Review

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: Aragon One

Reviewed commit: AragonApp.sol version 6c7da96.

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. 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. Call-specific values, block parameters, oracles are among them.

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` (example).

Acknowledged

COMMENTS

1. ReentrancyGuard.sol#L25

The reentrancy guard can be optimized using an incrementing value (e.g. this way). 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.