Aragon Protocol
Smart Contract Audit

coinspect

Smart Contract Audit
Aragon Protocol

Prepared for Aragon • November 2020

v201113

1. Executive Summary

In October 2020, Aragon engaged Coinspect to perform a source code review of changes to Aragon Protocol. The objective of the audit was to evaluate the security of the smart contracts.

The assessment covered the repository at https://github.com/aragon/protocol as of commit `f1b3361a160da92b9bb449c0a05dee0c30e41594` tag `audit`, with focus on recent changes on top of existing Aragon Court code.

No security issues were identified during the assessment.

2. Scope

The audit started on October 26th and included the repository at https://github.com/aragon/protocol as of commit `f1b3361a160da92b9bb449c0a05dee0c30e41594` tag `audit`. The focus of the audit was a set of changes on top of the existing Aragon Court 1.2 code, as detailed at https://github.com/aragon/protocol/tree/diff/packages/evm/diffs.

The scope of the audit was limited to the following Solidity source files, shown here with their sha256sum hash:

```
6c4fcf6ce7ae2d30402733c6bfd2b7c6bd25eef5de2607f639388dd1b57b2046  AragonProtocol.sol
d80c306819d39c9e7fe467a584aff55254d0ae30d822d2d6c000ef74656e8173  arbitration/IArbitrable.sol
53e99cc8eb0f34c377cfa94588a4e79fc507f38c99ecbf8679dc34fe8b94039e  arbitration/IArbitrator.sol
989531e0db8b17d1f4cd037b657eea5aae400ffab166787d2d0ab5c1c694e3f2  disputes/DisputeManager.sol
edad7367bc1a92bc7e42111f18b557e3c12420a31af2c0f21a1d3d6756468f33  disputes/IDisputeManager.sol
69e051a6d70bf7e69cee4c25dd142475890f82a608bdb9c54448d511fdf87c22  payments/IPaymentsBook.sol
fa7c8c6380a24501366c510920b53dd71c7dd2000160f5b3231d956d362712e5  payments/PaymentsBook.sol
bc5d77ef323e92497444bbfe812f966f158740731338421ddc8786bf34868238  registry/GuardiansRegistry.sol
05136412ba98d04c81e360d7c7c4cb05a228028d8d0b14c012cd6313d2bce8e4  registry/IGuardiansRegistry.sol
b3de5e76b631650b0d0feb0a98ccb0461f67a714a625bfface4d8ebd5bc5f2ee  registry/ILockManager.sol
c89cb78a5057cad48a541f47c26292b0f66a5039e83a493efb3c9f1983222b55  test/AragonProtocolMock.sol
678c1624d0e14d4f2bafb8c2f246bc426453c759d228a358d94475cc66bddc80  treasury/ITreasury.sol
0d659856aea98acfc83a1e0643d78543bd8f20fb698f026fbac46f23b7d66ed1  treasury/ProtocolTreasury.sol
1b9069dc7484b77378a213ac048cce6b8dd9efb0e5c10c13073b138b0fe160e1  voting/CRVoting.sol
48ec486bf3c8560eb11886ada698f3d393709bfeae539091ebc627b0b8e66752  voting/ICRVotingOwner.sol
c24da0a8f5be087ef5986af3cc04547da1476d2cf0287234f6eb48eb41d88931  voting/ICRVoting.sol
475ef8ef4a967861004b41fd72f48a55bbaef88f876dafaf96741320fbcf5ffb  core/clock/IClock.sol
75c8c9a937c4e75c3c67fc378a032032d77049bd2d7fe15758227ce29e0dc67c  core/clock/ProtocolClock.sol
977f78de665f60a5e7a34b011bd3c143d80ca251f93edaa53601a25b4d573a39  core/config/ConfigConsumer.sol
10bb4f4e59668a964adcf6b346642df20c57de596c50ec7b670abd062c0ad3d0  core/config/IConfig.sol
df560cdef7c43b631749ba16044a6e22cb75cb9ecf8bf583972741e369a6cfe9
core/config/ProtocolConfigData.sol
7408f7cf58315b7bfbecd6e981a8470b2db16d63c81abdb1e95172af75e19bef  core/config/ProtocolConfig.sol
e78e41ad24597529617cd4128c0de64288adcd790ae207f592fe3156efd4b0b9  core/modules/ACL.sol
5cfb1a6e3f4f205bcd6397a6f2ebaa94077bb0f1e49bd0ae2b9c0b88931a2a1a
core/modules/ControlledRecoverable.sol
b84f72ada51a99a456bc3b358d07386abf14e89780145f8a69f9042c47f6e230  core/modules/Controlled.sol
95a3d52c7dd236012b7b1545603bb7f457748ec39f1a71947406be81f6ea9691  core/modules/Controller.sol
35312fd1bc9e294d9049efd06a87d4416c75b0f8ac8897e7024e8b111ea1f595  core/modules/IModulesLinker.sol
aef1b9743807029abd3e652b23612ee30bed31907e8411e88a09fc5691bffa7c  core/modules/ModuleIds.sol
ee311b191e060a405617fdfc82d4c13e3ff0a5070938ab9a6171fb9268ded1d2  lib/math/SafeMath64.sol
de10c2fb8e844e251e1104090668bcfc1d79b13b64d68c59b2ba94b0b01a473a  lib/math/SafeMath.sol
07ef87a5ccb08ae65d3a3bc8f50456773d8c7801458679fc8080e557a838714a  lib/standards/IERC20.sol
2a90bf3e9fff1d9c57f9c6e7b592dbbefb2f111a899a05d0f71f1a0178dcbb03  lib/tree/Checkpointing.sol
767f054474be48dc6a652f92833fb6d464add15d47c862c3d67dba884132b49e
lib/tree/GuardiansTreeSortition.sol
7923e18489087eb0f21074bde0b7e1199a4439b21f5d8cc6d83eeea60208ec5d  lib/tree/HexSumTree.sol
264e3a7ddc63816fff55066f2c0cab3ed7b225006fc0296fb169995cccf48adb  lib/utils/IsContract.sol
eafd36c3c68d6620a4835224f5d1d5304cd243c22f640efc7c6c972398257b2e  lib/utils/PctHelpers.sol
30146a909041b86dfded49bbc62f4a4613fc733236f9cf5985009519ddffad2b  lib/utils/SafeERC20.sol
88e13cf826300c62dcf4009705436fba93ba812a85cef4a8512d18da7890f789  lib/utils/TimeHelpers.sol
8ebb921d740fa807b96cb8f51638cc852590280ac03c1657541216f192ce48a1  lib/utils/Uint256Helpers.sol
```

3. Assessment

Introduction

Aragon Protocol is a dispute resolution protocol. It handles subjective disputes that cannot be solved purely by smart contracts. Aragon Protocol relies on *guardians* that need to stake tokens to the Protocol in order to be drafted for voting on disputes, and this allows them to earn a share of the collected payments.

The `AragonProtocol` contract is the entry point and controller for the following modules:

- `DisputeManager`
- `GuardiansRegistry`
- `CRVoting`
- `PaymentsBook`
- `ProtocolTreasury`

These contracts are based on existing contracts from Aragon Court, with modifications including:

- Changes in terminology.
- Support for hot-swapping of modules.
- A new `PaymentsBook` module that replaces the `Subscriptions` module.
- Support custom functions that can be added to the `AragonProtocol` contract after deployment.
- Role-delegation for several functions in `GuardiansRegistry`, `CRVoting` and `ProtocolTreasury`.
- Functions for evidence submission in the `DisputeManager` module.
- Changes to `ControlledRecoverable` to allow recovery not only of tokens but also Ether.

The contracts are specified to be compiled with Solidity 0.5.17. This is the latest maintenance release of the 0.5.x series.

The contracts compile without any errors or warnings. Linting does not show any problems. The code is very clean and well commented.

The repository contains a very comprehensive set of 2054 unit tests for the smart contracts. After taking care of some minor hiccups (‘out of memory’ errors and timeouts), all tests passed.

Modules and custom functions

Among the changes introduced relative to Aragon Court 1.2, Aragon Protocol was made more modular. The new mechanism of “module linking” is defined by the interface `IModulesLinker` and implemented in the `Controlled` contract, and it allows replacing individual modules while on-going disputes are still being resolved. Only the *modules governor* can replace modules after deployment, or enable and disable modules.

The *module's governor* can also extend the `Controller` (`AragonProtocol`) with “custom functions”. A custom function is just a pair (`function signature, address`), and if the fallback function ends up being called for a function with the given signature, the call is forwarded to the corresponding address:

```
function () external payable {
    address target = customFunctions[msg.sig];
    require(target != address(0), ERROR_CUSTOM_FUNCTION_NOT_SET);

    (bool success,) = address(target).call.value(msg.value)(msg.data);
    assembly {
        let size := returndatasize
        let ptr := mload(0x40)
        returndatacopy(ptr, 0, size)

        let result := success
        switch result case 0 { revert(ptr, size) }
        default { return(ptr, size) }
    }
}
```

Note that when a custom function is called `msg.sender` is the address of the `Controller` itself, *not* the original sender. And once a custom function is added, a call from the Controller to the custom function can be triggered by anyone without any sort of access control mechanism.

Role-based authorization

The Protocol also includes a role-based system for authorizing a particular address to call a function in a module. This mechanism enables future extensions, for example a contract could implement its own verification scheme (e.g. off-chain signatures) and act on behalf of guardians.

The role-based system is implemented in `Controlled` and `Controller`. A function in a `Controlled` (i.e., a module) can use the `authenticateSender(user)` modifier, and this allows the function call to proceed only if 1) `msg.sender == user`, or 2) `msg.sender` has been granted the role to call this specific function:

```
modifier authenticateSender(address _user) {
    _authenticateSender(_user);
    _;
}

function _authenticateSender(address _user) internal view {
    require(_isSenderAllowed(_user), ERROR_SENDER_NOT_ALLOWED);
}

function _isSenderAllowed(address _user) internal view returns (bool) {
    return msg.sender == _user || _hasRole(msg.sender);
}

function _hasRole(address _addr) internal view returns (bool) {
    bytes32 roleId = keccak256(abi.encodePacked(address(this), msg.sig));
```

```
        return controller.hasRole(_addr, roleId);
    }
```

The `Controller` provides functions for setting and managing roles, and only the *config governor* can call these functions.

Note that a role corresponds to a function in a given contract, regardless of the `user` parameter passed to the modifier `authenticateSender`; this means that an address (a contract or an external account) with permission to call a function can call this function on behalf of *any* user. If no roles are set, the modifier `authenticateSender(user)` is equivalent to `require(msg.sender == user)`.

Role-based authorization is currently supported for the following functions: `withdraw` in `ProtocolTreasury`; `claimGuardianShare` in `PaymentsBook`; `commit` and `delegate` in `CRVoting`; and `unstake`, `activate`, `deactivate`, `stakeAndActivate`, `lockActivation` and `unlockActivation` in `GuardiansRegistry`.

Registry

Guardians register in the `GuardianRegistry` and stake funds. A given amount of tokens can be “activated” allowing the Protocol to lock them if the guardian is drafted for a dispute resolution period. The guardians have three kinds of balances in the registry:

- Active: tokens activated for the Protocol that can be locked if the guardian is drafted.
- Locked: amount of active tokens that are locked for a draft.
- Available: tokens that are not activated for the Protocol and can be withdrawn by the guardian at any time.

Locked tokens cannot be withdrawn until they are unlocked after the period for which the guardian was drafted is over.

Voting

The `CRVoting` contract implements a commit/reveal voting scheme that the guardians use for voting on disputes. The contract also allows for a guardian (or an authorized address) to delegate voting responsibilities to another party (a *delegate*).

The guardian, or a delegate designated by the guardian, or an authorized address can call the `commit` function to vote.

Also on a side note, the internal function `_chainId` in `CRVoting` is no longer used and can be removed.

Payments and shares

The new `PaymentsBook` module implements functions for paying fees to the Protocol, and for splitting the income between the *config governor* and active *guardians*. After a period has ended, a guardian can call the `claimGuardianShare` function to receive their share of tokens or Ether for that period, and any address can call `claimGovernorShare` in order to make the governor receive his share.

The share each guardian gets is proportional to the guardian's active balance in the `GuardiansRegistry` (at certain checkpoint within the period in question).

Any address with the required role can call `claimGuardianShare` on behalf of a guardian, and the share of tokens or Ether is sent to the guardian. It is worth mentioning that the *config governor* has permission to call `setGovernorSharePct` to change his own share of the collected payments.

5. Disclaimer

The information presented in this document is provided as is and without warranty. Vulnerability assessments are a “point in time” analysis and as such it is possible that something in the environment could have changed since the tests reflected in this report were run. This report should not be considered a perfect representation of the risks threatening the analysed system, networks and applications.