Aragon Protocol contest Findings & Analysis Report

2023-12-12

Table of contents

Overview

About C4

Code4rena (C4) is an open organization consisting of security researchers, auditors, developers, and individuals with domain expertise in smart contracts.

A C4 audit is an event in which community participants, referred to as Wardens, review, audit, or analyze smart contract logic in exchange for a bounty provided by sponsoring projects.

During the audit outlined in this document, C4 conducted an analysis of the Aragon Protocol smart contract system written in Solidity. The audit took place between March 3—March 10 2023.

Wardens

43 Wardens contributed reports to Aragon Protocol:

1. 0x52
2. 0x6980
3. 0xAgro
4. 0xSmartContract
5. 0xWeiss
6. 0xmichalis
7. 0xnev
8. 0xsomeone
9. AkshaySrivastav
10. BRONZEDISC
11. DevABDee
12. IceBear
13. JCN
14. Madalad
15. Phantasmagoria
16. Rageur
17. RaymondFam
18. ReyAdmirado
19. Rolezn
20. SaeedAlipoor01988
21. Sathish9098

22. V_B (Barichek and vlad_bochok)
23. adriro
24. arialblack14
25. atharvasama
26. banky
27. brgltd
28. carlitox477
29. chrisdior4
30. codeislight
31. descharre
32. hunter_w3b
33. imare
34. lukris02
35. luxartvinsec
36. matrix_0wl
37. rbserver
38. sakshamguruji
39. saneryee
40. tnevler
41. volodya
42. yongskiws

This audit was judged by 0xean.

Final report assembled by liveactionllama.

Summary

The C4 analysis yielded an aggregated total of 4 unique vulnerabilities. Of these vulnerabilities, 0 received a risk rating in the category of HIGH severity and 4 received a risk rating in the category of MEDIUM severity.

Additionally, C4 analysis included 29 reports detailing issues with a risk rating of LOW severity or non-critical. There were also 18 reports recommending gas optimizations.

All of the issues presented here are linked back to their original finding.

Scope

The code under review can be found within the C4 Aragon Protocol repository, and is composed of 62 smart contracts written in the Solidity programming language and includes 7,152 lines of Solidity code.

Severity Criteria

C4 assesses the severity of disclosed vulnerabilities based on three primary risk categories: high, medium, and low/non-critical.

High-level considerations for vulnerabilities span the following key areas when conducting assessments:

- Malicious Input Handling
- Escalation of privileges
- Arithmetic
- Gas use

For more information regarding the severity criteria referenced throughout the submission review process, please refer to the documentation provided on the C4 website, specifically our section on Severity Categorization.

Medium Risk Findings (4)

[M-01] User may force fail the action from the `DAO:execute`

Submitted by V_B, also found by 0xWeiss

DAO.sol#L186
MajorityVotingBase.sol#L286
MajorityVotingBase.sol#L459

The `execute` function from the `DAO.sol` contract allow to execution of any call to any address if the caller has appropriate permission. Some calls are expected to be always successfully executed, and some may revert and `execute` will continue the execution.

The following code may call and handle call status.

```
    address to = _actions[i].to;
    (bool success, bytes memory response) = to.call{value: _a
        _actions[i].data
    );

    if (!success) {
        // If the call failed and wasn't allowed in allowFai
        if (!hasBit(_allowFailureMap, uint8(i))) {
            revert ActionFailed(i);
        }

        // If the call failed, but was allowed in allowFailu
        // this specific action has actually failed.
        failureMap = flipBit(failureMap, uint8(i));
    }
```

Also, the function is expected to be used in a different scenario, where the caller may be a user, voter, etc. (See `MajorityVotingBase`). So the caller is not a trusted entity and that means any manipulation of the DAO call should be avoided.

The problem is that caller may choose the gas with which the code is executed. If the child call execution spends enough gas then the user may choose that amount of gas, that child call frame fails, but the left gas is enough to successfully finish `DAO:execute` function.

Please note, even though the `execute` pass all gas to the child call, actually only 63/64 gas is passed and 1/64 of gas is left on the parent call (EIP-150).

- https://medium.com/iovlabs-innovation-stories/the-dark-side-of-ethereum-1-64th-call-gas-reduction-ba661778568c

Attack scenario

The DAO starts majority voting, and users who have DAO tokens may vote for the proposal. The proposal is to call one `target` protocol, which may fail in case of an inner reason. So the DAO set that the call may fail. The approximate gas that is needed to finish the call to the `target` contract is `700k`. A malicious voter call `execute` function with `711.1k` of gas. Since `63/64 * 711.1 < 700`, the requested call will fail. And the remaining gas is still sufficient to end the `execute` function logic.

Impact

The user may forcefully fail the inner call from the `execute` function. Also, anyone who will use the usual `eth_estimateGas` for the gas estimation for the `execute` function will accidentally calculate the amount of gas that will fail the call.

Since majority voting is hard to process with many users involved, creating another proposal may create a lot of pain.

Recommended Mitigation Steps

Add the require that gas after the call is bigger than gas before / 64.

```
    uint256 gasBefore;
    // Do call...
    require(gasleft() > gasBefore/64);
```

novaknole20 (Aragon) confirmed

[M-02] `MerkleMinter` created through `TokenFactory` cannot be upgraded

Submitted by adriro

During the token creation process in the `TokenFactory` contract, the function creates a `MerkleMinter` contract to setup and handle token initial token distribution.

https://github.com/code-423n4/2023-03-aragon/blob/main/packages/contracts/src/framework/utils/TokenFactory.sol#L119-L125

```
    ...

    // Clone and initialize a `MerkleMinter`
    address merkleMinter = merkleMinterBase.clone();
    MerkleMinter(merkleMinter).initialize(
        _managingDao,
        IERC20MintableUpgradeable(token),
        distributorBase
    );

    ...
```

The `MerkleMinter` contract is an upgradeable contract, as it inherits from `PluginUUPSUpgradeable`:

https://github.com/code-423n4/2023-03-aragon/blob/main/packages/contracts/src/plugins/token/MerkleMinter.sol#L20

```
contract MerkleMinter is IMerkleMinter, PluginUUPSUpgrad
```

However, as we can see in the first code snippet, the `MerkleMinter` instance created in `createToken` is a **cloned** instance (using OpenZeppelin `Clones` library). This is incompatible with upgradeable contracts, which require the use of a proxy.

This issue will cause the `MerkleMinter` instance created through `TokenFactory` to fail to be upgraded. The `MerkleMinter` contract will contain all the required logic to be upgraded, but the action will fail as there is no proxy to change to a new potential implementation.

Proof of Concept

The following test illustrates the issue. We call `createToken` to get an instance of `MerkleMinter`. We then simulate a new version of the contract to upgrade to (`merkleMinterV2Impl`) and try to upgrade the `MerkleMinter` instance to this new implementation. The call fails with a "Function must be called through active proxy" error (error is defined in OpenZeppelin base `UUPSUpgradeable` contract).

Note: the snippet shows only the relevant code for the test. Full test file can be found here.

```
function test_TokenFactory_createToken_MerkleMinterNotUp
    DAO dao = createDao();
    TokenFactory tokenFactory = new TokenFactory();
    grantRootPermission(dao, address(tokenFactory));

    TokenFactory.TokenConfig memory tokenConfig = TokenF
```

```
            addr: address(0),
            name: "DAO Token",
            symbol: "DAOT"
        });

        address[] memory receivers = new address[](0);
        uint256[] memory amounts = new uint256[](0);
        GovernanceERC20.MintSettings memory mintSettings = Go
            receivers: receivers,
            amounts: amounts
        });

        (, MerkleMinter merkleMinter) = tokenFactory.createTo

        // Assume we have a new V2 implementation...
        MerkleMinter merkleMinterV2Impl = new MerkleMinter();

        // The following will fail when the UUPS checks if th
        vm.expectRevert("Function must be called through acti
        PluginUUPSUpgradeable(merkleMinter).upgradeTo(address
    }
```

Recommendation

The `MerkleMinter` instance should be created using a proxy over the base implementation (`createERC1967Proxy`) instead of cloning the implementation:

```
diff --git a/src/framework/utils/TokenFactory.sol b/src/
index 381e745..91441e5 100644
--- a/src/framework/utils/TokenFactory.sol
+++ b/src/framework/utils/TokenFactory.sol
@@ -15,6 +15,7 @@ import {GovernanceWrappedERC20} from ".
 import {IERC20MintableUpgradeable} from "../../token/ERC
 import {DAO} from "../../core/dao/DAO.sol";
 import {IDAO} from "../../core/dao/IDAO.sol";
+import {createERC1967Proxy} from "../../utils/Proxy.sol'

 /// @title TokenFactory
 /// @author Aragon Association - 2022-2023
```

```
@@ -116,12 +117,15 @@ contract TokenFactory {
                 _mintSettings
             );

-            // Clone and initialize a `MerkleMinter`
-            address merkleMinter = merkleMinterBase.clone();
-            MerkleMinter(merkleMinter).initialize(
-                _managingDao,
-                IERC20MintableUpgradeable(token),
-                distributorBase
+            // Create proxy and initialize a `MerkleMinter`
+            address merkleMinter = createERC1967Proxy(
+                merkleMinterBase,
+                abi.encodeWithSelector(
+                    MerkleMinter.initialize.selector,
+                    _managingDao,
+                    token,
+                    distributorBase
+                )
             );

             // Emit the event
```

Oxean (judge) commented:

> I don’t believe a MerkleMinter instance is intended to be upgradeable.

> “The **Clones** library provides a way to deploy minimal non-upgradeable proxies for cheap. This can be useful for applications that require deploying many instances of the same contract (for example one per user, or one per task). These instances are designed to be both cheap to deploy, and cheap to call. The drawback being that they are not

> Will leave open for sponsor confirmation. But most likely this is invalid.

novaknole20 (Aragon) confirmed

Oxean (judge) commented:

> It appears the sponsor does intend for this to be upgradeable since they confirmed the issue. Awarding as Medium.

Please note: the following additional discussion took place after judging and awarding were finalized.

novaknole20 (Aragon) commented:

> Sorry we don't know why this got confirmed from our side... As seen in the code the MerkleMinter is only used by the `TokenFactory` and there we use clones (minimal non-upgradable proxies).
> Therefore we need to extend from `PluginUUPSUpgradeable` to disable the initializer in the base during construction.
> We'd like to note our mistake in accepting it and that we never intend to use it in an upgradeable pattern.

novaknole20 (Aragon) commented:

> `MerkleMinter` is `UUPSUpgradeable` and the upgradeability is part of its feature set. However, in the context of the `TokenFactory` we opted to deploy it via the minimal proxy pattern to save gas. This is because its upgradeability feature is not needed here and the cloning can be done as well. We are fully aware that the `MerkleMinter` proxies created through the `TokenFactory` are not upgradeable.

> Although we accidentally accepted this medium risk finding initially, we don't think that intentional absent of upgradeability qualifies as medium risk/finding. Accordingly, we would suggest to not rate this as a medium risk finding in the final audit report.

> To make our intentions more clear, we made the following change to the documentation: https://github.com/aragon/osx/pull/362/files

> It is also worth noting that we are currently using neither `TokenFactory` nor `MerkleMinter` / `MerkleDistributor` in our framework and that we will move the code out of the aragon/osx repository in the future.

[M-03] `createProposal` snapshot block can temporarily desync with `minApproval` / `minVotingPower`

Submitted by 0x52, also found by AkshaySrivastav

minApproval and member list will be temporarily out of sync, potentially causing approval issues.

Proof of Concept

Multisig.sol#L214-L245

```
        uint64 snapshotBlock = block.number.toUint64() - 1;

        ...

        // Create the proposal
        Proposal storage proposal_ = proposals[proposalId];


        proposal_.parameters.snapshotBlock = snapshotBlock;
        proposal_.parameters.startDate = _startDate;
        proposal_.parameters.endDate = _endDate;
        proposal_.parameters.minApprovals = multisigSettings
```

When creating a proposal all the voting contracts (multisig, token, addresslist) use a snapshot block that is one block in the past. The problem is that they use the current settings for determining min voting are but uses a snapshot block that is 1 block behind. This causes a desync between the members who can vote and the approval needed to pass a proposal.

Example

Imagine the following scenario. There is a multisig with 10 members and a min approval of 6. There is some kind of leak that exposes the private keys of 4 of the members and their addresses are hijacked. The multisig creates a proposal that removes the 4 members and lowers the min approval to 4. On the block that the proposal is executed, one of the malicious members creates a proposal removing the other six members from the multisig. Now when that proposal is created it will use voting eligibility from the block before (which includes the 4 compromised accounts) but it will apply the newly changed min approval of 3. Now the 4 compromised accounts can approve their proposal and hijack the multisig.

Similar scenarios can occur in token and addresslist voting anytime the approval threshold and supply/members are changed in a single proposal.

Recommended Mitigation Steps

Threshold parameters should be snapshot the same way that eligibility or token balances are.

Oxean (judge) decreased severity to Medium and commented:

> Hard to see how this qualifies as High severity. The warden's premise is built off of:

> "There is some kind of leak that exposes the private keys of 4 of the members and their addresses are hijacked."

> Which by definition already means that there are security assumptions broken across the entire DAO. I think the point is however valid about the synchronization of these settings.

novaknole20 (Aragon) confirmed and commented:

> Very good finding. Thank you

[M-04] `DAO.execute(bytes32, Action[], uint256)` is vulnerable to re-entrancy attacks

Submitted by carlitox477

The present implementation permits the execution of a predetermined sequence of instructions, where the order of execution is at times crucial, as described in the documentation:

> Imagine a DAO is currently governed by a multisig (it has the Multisig plugin installed) and wants to transition the DAO governance to token voting. To achieve this, one of the Multisig members creates a proposal in the plugin proposing to

1. install the TokenVoting plugin
2. uninstall the Multisig plugin

> If enough multisig signers approve and the proposals passes, the action array can be executed and the transition happens.

> **Here, it is important that the two actions happen in the specified order and both are required to succeed.** Otherwise, if the first action would fail or the second one would happen beforehand, the DAO could end up in a state without having a governance plugin enabled.

In the event that any of the actions performs a callback to the permitted msg.sender, that particular address would gain the ability to dispatch the same set of actions.

Impact

Considering that the `msg.sender` could be a contract, the importance of the order of execution (at least in certain scenarios), and the potential for reentrancy, it can be concluded that the intended behavior is at risk of compromise

Proof of Concept

In the following hypothetical scenario:

1. A contract named `DAOsWilling` possesses the necessary authorization to invoke `DAO.execute`.
2. `DAOsWilling` has established a predetermined sequence of five ordered actions that, according to its code, are guaranteed to succeed. However, it is imperative that these actions be executed in a specific order.
3. `DAOsWilling` permits anyone to invoke a function - let's call it `executeDaosWilling` - in order to execute this pre-established sequence of ordered actions.
4. Furthermore, `DAOsWilling` has a contract that contains a callback function for one of the aforementioned actions, which subsequently calls back to the original caller of `executeDaosWilling`."

Then, next action could happen:

1. A DAO has established a set of 5 actions in `DAOsWilling` contract, the 3° action do a callback in name of `executeDaosWilling` caller
2. Bob calls `executeDaosWilling` through a smart contracts he has designed, after the 3° action a callback is done to `DAOsWilling`, which calls Bob's contract, and Bob's contracts calls `executeDaosWilling` again
3. Then 1 to 5 action is executed in a row (let's suppose that in this other callback done by the 3° action Bob's contract do nothing), and then action 4 and 5 are executed

This POC shows that there scenarios where we cannot guarantee that actions are execute in order.

Recommended Mitigation Steps

Use `ReentrancyGuard` contract from openzeppelin and add `nonReentrant` modifier to `execute(bytes32, Action[], uint256)` function or assume the risk informing them in documentation.

```
    function execute(
        bytes32 _callId,
        Action[] calldata _actions,
        uint256 _allowFailureMap
    )
+       nonReentrant
        external
        override
        auth(EXECUTE_PERMISSION_ID)
        returns (bytes[] memory execResults, uint256 fai
    {
```

Oxean (judge) commented:

> I think this is worth leaving open for sponsor comment, but would assume that the validation of the proposal payload is meant to happen by voters and would include validating that the payload doesn't re-enter.

novaknole20 (Aragon) acknowledged and commented:

> Hey, we're aware of this. This would be only possible if action calls the caller which again calls `dao.execute`.

> `cA => dao.execute([action1, action2])` => action2 calls back `cA` => which calls `dao.execute`. Thats the only possibility as `dao.execute` is protected by `EXECUTE_PERMISSION`.

> This all means that action shouldn't be calling back the original caller, which must be the members' responsibility to review.

Oxean (judge) decreased severity to Low/Non-Critical

carlitox477 (warden) commented:

> I would like to respectfully express my disagreement with the judge's decision to downgrade this issue from Medium to QA. My disagreement is based on the following facts:

- **Sponsors acknowledgment and lack of information about it in the documentation:** Although the sponsor acknowledged the bug, they did not make it explicitly clear in the documentation or the code. In the appropriate scenario, unless developers who built on top of Aragon take notice, this bug would remain undiscovered, leaving them exposed to potential risks.
- **Precedent:** Olympus DAO M-04 exposed a similar vulnerability which was awarded as medium: The `executeProposal` function in Olympus DAO was vulnerable to the same bug, this function can be considered similar to `executeProposal()` in Aragon protocol. Taking into account that Olympus DAO protocol was a concrete implementation of a DAO and not a protocol to build on top, it can be perfectly argued that Olympus DAO finding vulnerability impact is less than Aragon DAO current bug reported.
- **The protocol is meant to be used to build on top of it:** The Aragon protocol is designed to be used as a foundation for building DAOs using the offered contracts. Developers should be provided with all the necessary information that can improve their development experience and help them avoid exposing their users to potential attack vectors that can compromise the products they offer. In this specific case, if this bug had not been reported, and since the sponsor did not explicitly mention it in the documentation, new DAOs built on top of the Aragon protocol would have been exposed to this vulnerability. For example, if OlympusDAO had chosen to use the Aragon protocol as the foundation for building their DAO, they too would have been exposed to the same bug that they themselves identified as a Medium severity issue in their own audit.
- > Current medium criteria:

- Asset are not at direct risk: Met
- > Requirement of at least one of next cases:

    1. **function of the protocol:** Sponsors stated `This all means that action shouldn't be calling back the original caller`
    2. **its availability could be impacted:** Not the case
    3. **leak value with a hypothetical attack path with stated assumptions, but external requirements:** The hypothetical attack path with stated assumption was exposed and acknowledged by the sponsors. Another attack path is presented above, more simpler than the one exposed in the presented issue, but based in the same vector attack allowed by this bug: reentrancy.

> **About particular sponsor response:** The sponsor argue that * `cA => dao.execute([action1, action2]) => action2` calls back `cA` => which calls `dao.execute`. Thats the only possibility as `dao.execute` is protected by `EXECUTE_PERMISSION`. This all means that **action** shouldn't be calling back the original caller, which must be onto the members' responsibility to review*. This argument can be illustrated like:

> The sponsor is suggesting that an **action** should intentionally produce a callback (by DAO builders decision), also that `EXECUTE_PERMISSION` will be enough to forbid a malicious user to exploit this vector attack.

> This line of thought seems reasonable at first, however it ignores 2 crucial factors: action have consequences:

1. `EXECUTE_PERMISSION` can mean nothing if builders decides so. They just would have to create a contract with an execute function which calls `dao.execute(lastVotedActions)`. Why would they be willing to do this? To show how decentralized they are (as Olympus DAO decided with their executeProposal function)

2. Actions has consequences: The sponsor supposes that action 2 should necessarily imply a super specific call back, but if action 2 do an ERC-777 transfer or an NFT safeMinting or transfer (maybe as a participation reward) to the one who has trigger the execution of voted action, then an indirect consequence of this would be opening to the re-entrancy attack vector.

> Here the use case I think the sponsor is suggesting:
> Sponsor suggested case

> Here the use case in which an executer (considering it is a smart contract deployed by anyone) abuse of this bug to mint 2 NFTs instead of 1:
> NFT minting case

> Given the sponsor acknowledgment, the lack of explicit statement of the bugs in the documentation, the consideration this protocol is meant to be used to build on top of it, current rules for medium requirement are met, and OlympusDAO precedent I would like to respectfully disagree with current judge's decision to downgrade the issue from medium to QA, and beg them to reconsider their decision based on the exposed arguments.

> I would also like to encourage the Aragon protocol to implement my recommendation, as it would demonstrate that the protocol is taking all necessary steps to protect developers and builders from possible misunderstandings about how to correctly build and develop using the protocol. By adopting this recommendation, the Aragon protocol would provide greater clarity and transparency, enabling developers to build with confidence and reducing the likelihood of vulnerabilities being introduced into the system.

Oxean (judge) increased severity to Medium and commented:

> @carlitox477 - thanks for stating your objections in a productive way.

> The external requirement here is that a DAO fails to properly review a proposal, if that assumption is broken essentially *ANY* action can be taken on behalf of the DAO. Tools like tenderly should also provide some

> really good visibility into the transactions results before approval / execution.

> All of that being said, I don't think it is that wild to imagine a DAO missing this, and approving a malicious transaction that has some subtle re-entrant behavior. Given that this is supposed to be generic tooling for DAOs, I am going to re-open this back to Medium.

Low Risk and Non-Critical Issues

For this audit, 24 reports were submitted by wardens detailing low risk and non-critical issues. The report highlighted below by **rbserver** received the top score from the judge.

The following wardens also submitted reports: yongskiws, chrisdior4, 0x6980, brgltd, DevABDee, imare, RaymondFam, 0xAgro, codeislight, IceBear, lukris02, 0xnev, descharre, matrix_0wl, luxartvinsec, 0xSmartContract, Rolezn, arialblack14, 0xmichalis, Sathish9098, tnevler, SaeedAlipoor01988, *and* BRONZEDISC.

Summary

	Issue
[01]	`DAO.execute` FUNCTION DOES NOT CONSIDER TOKEN'S `transfer` OR `transferFrom` FUNCTION CALL THAT DOES NOT REVERT BUT RETURNS `false` AS A FAILURE
[02]	`DAO.execute` FUNCTION DOES NOT CHECK IF `_actions[i].to` HAS ANY CONTRACT CODE WHEN `_actions[i].data` IS NOT EMPTY
[03]	WHEN `_actions[i].data` IS NOT EMPTY, `DAO.execute` FUNCTION DOES NOT CHECK IF SUCH `_actions[i].data` 'S FUNCTION EXISTS IN `_actions[i].to` CONTRACT
[04]	`PermissionManager.revoke` TRANSACTION CAN BE FRONTRUN
[05]	WHETHER `PermissionManager._grantWithCondition` FUNCTION SHOULD REVERT WHEN `_where == ANY_ADDR` OR `_who == ANY_ADDR` IS TRUE NEEDS TO BE RESOLVED

	Issue
[06]	`DAO` CONTRACT'S `receive()` CAN BE UPDATED TO CALL `DAO.deposit` FUNCTION WITH `_token` INPUT BEING `address(0)`
[07]	UNLIKE `DAO.deposit`, `DAO` CONTRACT HAS NO FUNCTIONS FOR DEPOSITING ERC721 AND ERC1155 TO DAO
[08]	`PermissionManager.applyMultiTargetPermissions` FUNCTION ALREADY COVERS USE CASES OF `PermissionManager.applySingleTargetPermissions` FUNCTION
[09]	MISSING `address(0)` CHECKS FOR CRITICAL ADDRESS INPUTS
[10]	REDUNDANT RETURN STATEMENTS FOR FUNCTIONS WITH NAMED RETURNS CAN BE REMOVED
[11]	VULNERABILITIES IN VERSION 4.8.1 OF `@openzeppelin/contracts` AND `@openzeppelin/contracts-upgradeable`
[12]	SOLIDITY VERSION `0.8.19` CAN BE USED
[13]	DEFINITIONS OF `UNSET_FLAG` AND `ALLOW_FLAG` ARE INCORRECT IN DOCUMENTATION
[14]	`bytes4(0)` CAN BE REPLACED WITH A CONSTANT
[15]	`NewURI` EVENT CAN BE MOVED TO `IDAO` INTERFACE
[16]	INPUT VARIABLE CAN BE NAMED WITH LEADING UNDERSCORE
[17]	WORD TYPING TYPOS
[18]	INCOMPLETE NATSPEC COMMENTS

[01] `DAO.execute` FUNCTION DOES NOT CONSIDER TOKEN'S `transfer` OR `transferFrom` FUNCTION CALL THAT DOES NOT REVERT BUT RETURNS `false` AS A FAILURE

Some tokens do not revert but return `false` when calling their `transfer` or `transferFrom` functions fail; to cover this scenario, OpenZeppelin's `SafeERC20` library would ensure that the corresponding return data must not be false by executing this require statement. However, for the same scenario, when calling the following `DAO.execute` function, if `_actions[i].to` corresponds to such token and `_actions[i].data`'s function is `transfer` or `transferFrom`, `success` would be set to true after executing `(bool success, bytes memory response) = to.call{value: _actions[i].value}(_actions[i].data)`, and this call would not be considered as a failure. As a result, the DAO can result in an unexpected state; for example, because transferring such tokens to the DAO fail silently, the DAO could falsely think that it has received the corresponding funds and update its accounting system incorrectly.

As a mitigation, the `DAO.execute` function can be updated to check if `response` returned by executing `to.call` is false when `_actions[i].data`'s function is `transfer` or `transferFrom`. If it is false, the corresponding call should be considered as a failure; whether such failure can be allowed or not can then be determined by the `_allowFailureMap` input.

https://github.com/code-423n4/2023-03-aragon/blob/main/packages/contracts/src/core/dao/DAO.sol#L168-L215

```
    function execute(
        bytes32 _callId,
        Action[] calldata _actions,
        uint256 _allowFailureMap
    )
        external
        override
        auth(EXECUTE_PERMISSION_ID)
        returns (bytes[] memory execResults, uint256 fai
    {
        ...
        for (uint256 i = 0; i < _actions.length; ) {
            address to = _actions[i].to;
            (bool success, bytes memory response) = to.ca
                _actions[i].data
            );

            if (!success) {
                // If the call failed and wasn't allowed
                if (!hasBit(_allowFailureMap, uint8(i)))
                    revert ActionFailed(i);
                }

                // If the call failed, but was allowed in
                // this specific action has actually fai
                failureMap = flipBit(failureMap, uint8(i
            }
            ...
        }
        ...
    }
```

novaknole20 (Aragon) commented:

> Because it is a generic executor we don't check for this and it is up to the user.

0xean (judge) commented:

> Low Risk

[02] `DAO.execute` FUNCTION DOES NOT CHECK IF `_actions[i].to` HAS ANY CONTRACT CODE WHEN `_actions[i].data` IS NOT EMPTY

It is a popular practice for a protocol to deploy its token using the same deployer contract address and the same nonce so the addresses of such token are the same on different chains. Thus, the DAO could expect that the addresses of an external protocol's token are the same on different chains.

Yet, it is possible that the external protocol has not deployed its token on one of these chains but the DAO does not notice this and calls the following `DAO.execute` function to interact with such token address on such chain, such as for transferring certain amount of such token to the DAO. When the `_actions[i].to` address corresponding to such token does not have any contract code, executing `(bool success, bytes memory response) = to.call{value: _actions[i].value}(_actions[i].data)` in the `DAO.execute` function would return a true `success`. In this case, this action for interacting with such token on such chain is considered as a success even though such interaction did fail silently, which can cause the DAO to end up in an unexpected state, such as that the DAO expects to receive an amount of such token but does not in reality. Because of this, disputes can occur among the DAO, the DAO's community, and this protocol.

As a mitigation, the `DAO.execute` function can be updated to check the `_actions[i].to` address's contract code size if `_actions[i].data` is not empty for `_actions[i]`. When `_actions[i].data` is not empty but `_actions[i].to` address does not have any contract code, the corresponding call should be considered as a failure and can be processed according to the provided `_allowFailureMap`.

https://github.com/code-423n4/2023-03-aragon/blob/main/packages/contracts/src/core/dao/DAO.sol#L168-L215

```
    function execute(
        bytes32 _callId,
        Action[] calldata _actions,
        uint256 _allowFailureMap
    )
        external
        override
        auth(EXECUTE_PERMISSION_ID)
        returns (bytes[] memory execResults, uint256 fai
    {
        ...
```

```
            for (uint256 i = 0; i < _actions.length; ) {
                address to = _actions[i].to;
                (bool success, bytes memory response) = to.ca
                    _actions[i].data
                );

                if (!success) {
                    // If the call failed and wasn't allowed
                    if (!hasBit(_allowFailureMap, uint8(i)))
                        revert ActionFailed(i);
                    }

                    // If the call failed, but was allowed i
                    // this specific action has actually fai
                    failureMap = flipBit(failureMap, uint8(i
                }
                ...
            }
            ...
        }
```

novaknole20 (Aragon) commented:

> It is ok to send data in a tx even when it gets send to an EOA wallet. See this TX from euler to their recent attacker https://etherscan.io/tx/0x8f2b61a0c70012df1e3d918e7ac6486a1c332a9e530f3d4061735c6620f960d9

0xean (judge) commented:

> Low Risk

[03] WHEN `_actions[i].data` IS NOT EMPTY, `DAO.execute` FUNCTION DOES NOT CHECK IF SUCH `_actions[i].data` 'S FUNCTION EXISTS IN `_actions[i].to` CONTRACT

After contracts for the same usage are deployed on different chains by a protocol, it is possible that these contracts are somewhat different, such as due to an upgrade. When a contract has a function but the contract for the same usage on the other chain does not have that function while still having a fallback function, the DAO might not notice this and could expect that both contracts are the same, such as because of the lack of clear documentation by the external protocol that owns these contracts. In this situation, when the DAO calls the following `DAO.execute` function with `_actions[i].to` being the contract on the other chain, the corresponding `_actions[i].data`'s function is not found in the `_actions[i].to` contract but the `_actions[i].to` contract's fallback function is triggered. This can result in an unexpected state for the DAO; for example, the `_actions[i].to` contract's fallback function can transfer out the DAO's funds and execute some logics but calling `_actions[i].data`'s function should transfer out the DAO's funds for executing different logics.

As a mitigation, when `_actions[i].data` is not empty, the `DAO.execute` function can be updated to check if such `_actions[i].data`'s function exists in the `_actions[i].to` contract. If such `_actions[i].data`'s function does not exist, the corresponding call should be considered as a failure and can be processed based on the specified `_allowFailureMap`.

https://github.com/code-423n4/2023-03-aragon/blob/main/packages/contracts/src/core/dao/DAO.sol#L168-L215

```
    function execute(
        bytes32 _callId,
        Action[] calldata _actions,
        uint256 _allowFailureMap
    )
        external
        override
        auth(EXECUTE_PERMISSION_ID)
        returns (bytes[] memory execResults, uint256 fai
    {
        ...
```

```
            for (uint256 i = 0; i < _actions.length; ) {
                address to = _actions[i].to;
                (bool success, bytes memory response) = to.ca
                    _actions[i].data
                );

                if (!success) {
                    // If the call failed and wasn't allowed
                    if (!hasBit(_allowFailureMap, uint8(i)))
                        revert ActionFailed(i);
                    }

                    // If the call failed, but was allowed i
                    // this specific action has actually fai
                    failureMap = flipBit(failureMap, uint8(i
                }
                ...
            }
            ...
        }
```

novaknole20 (Aragon) commented:

> [03] is the same as [02]

0xean (judge) commented:

> Low Risk

[04] `PermissionManager.revoke` TRANSACTION CAN BE FRONTRUN

Calling the following `PermissionManager.revoke` function, which further calls the `PermissionManager._revoke` function, would revoke the permission from an address for calling the corresponding functions on a target contract, such as when such address has become untrusted. However, such address can monitor the mempool and frontruns the relevant `PermissionManager.revoke` transaction so it can call the corresponding functions on the target contract in a malicious manner before losing the permission. For example, when the address, who has the permission associated with `REGISTER_STANDARD_CALLBACK_PERMISSION_ID`, becomes compromised or malicious, this address can frontrun the DAO's `PermissionManager.revoke` transaction to call the `DAO.registerStandardCallback` function with the `_callbackSelector` input being `IERC721ReceiverUpgradeable.onERC721Received.selector` and the `_magicNumber` input being a `bytes4` that is not `IERC721ReceiverUpgradeable.onERC721Received.selector`; then, the safe-transfers of any ERC721 tokens to the DAO can fail.

As a mitigation, flashbots can be used to keep the `PermissionManager.revoke` transactions away from the mempool for counteracting frontrunning.

https://github.com/code-423n4/2023-03-aragon/blob/main/packages/contracts/src/core/permission/PermissionManager.sol#L135-L141

```
    function revoke(
        address _where,
        address _who,
        bytes32 _permissionId
    ) external virtual auth(ROOT_PERMISSION_ID) {
        _revoke(_where, _who, _permissionId);
    }
```

https://github.com/code-423n4/2023-03-aragon/blob/main/packages/contracts/src/core/permission/PermissionManager.sol#L282-L289

```
    function _revoke(address _where, address _who, bytes
        bytes32 permHash = permissionHash(_where, _who, _
        if (permissionsHashed[permHash] != UNSET_FLAG) {
            permissionsHashed[permHash] = UNSET_FLAG;

            emit Revoked(_permissionId, msg.sender, _whe
        }
    }
```

https://github.com/code-423n4/2023-03-aragon/blob/main/packages/contracts/src/core/dao/DAO.sol#L309-L317

```
        function registerStandardCallback(
            bytes4 _interfaceId,
            bytes4 _callbackSelector,
            bytes4 _magicNumber
        ) external override auth(REGISTER_STANDARD_CALLBACK_I
            _registerInterface(_interfaceId);
            _registerCallback(_callbackSelector, _magicNumbe
            emit StandardCallbackRegistered(_interfaceId, _ca
        }
```

novaknole20 (Aragon) commented:

> Frontrunning this function call implies that the frontrunner has permission to call this function which is already a security risk in itself.

0xean (judge) commented:

> Non-Critical

[05] WHETHER `PermissionManager._grantWithCondition` FUNCTION SHOULD REVERT WHEN `_where == ANY_ADDR` OR `_who == ANY_ADDR` IS TRUE NEEDS TO BE RESOLVED

https://devs.aragon.org/docs/osx/how-it-works/core/permissions/#granting-permission-to-any_addr states that "by granting the `USE_PERMISSION_ID` to `_who: ANY_ADDR` on the contract `_where: serviceAddr` you allow everyone to call the `use()` function and you can add more conditions to it"; yet, this documentation does not indicate that condition(s) must be used in this case. However, https://github.com/code-423n4/2023-03-aragon#grant-with-condition-with-any_addr states that "we only allow setting `who` / `where` to `ANY_ADDR` if `oracle` is present", where `oracle` seems to mean condition(s); this is matched by the code in which calling the following `PermissionManager._grantWithCondition` function would execute `revert ConditionNotPresentForAnyAddress()` if `_where == ANY_ADDR || _who == ANY_ADDR` is true.

If https://github.com/code-423n4/2023-03-aragon#grant-with-condition-with-any_addr and the `PermissionManager._grantWithCondition` function are accurate, then please update https://devs.aragon.org/docs/osx/how-it-works/core/permissions/#granting-permission-to-any_addr to explicitly indicate that condition(s) must be used in this situation to avoid confusions for users and prevent future disputes. Otherwise, the `PermissionManager._grantWithCondition` function needs to be updated to not revert if `_where == ANY_ADDR || _who == ANY_ADDR` is true because granting permission without condition(s) in this case would be allowed.

https://github.com/code-423n4/2023-03-aragon/blob/main/packages/contracts/src/core/permission/PermissionManager.sol#L230-L275

```
    function _grantWithCondition(
        address _where,
        address _who,
        bytes32 _permissionId,
        IPermissionCondition _condition
    ) internal virtual {
        if (_where == ANY_ADDR && _who == ANY_ADDR) {
            revert AnyAddressDisallowedForWhoAndWhere();
        }

        if (_where == ANY_ADDR || _who == ANY_ADDR) {
            bool isRestricted = isPermissionRestrictedFo
            if (_permissionId == ROOT_PERMISSION_ID || is
                revert PermissionsForAnyAddressDisallowe
            }

            if (address(_condition) == ALLOW_FLAG) {
                revert ConditionNotPresentForAnyAddress(
            }
        }

        ...
    }
```

novaknole20 (Aragon) commented:

> That is not fully right. `_grantWithConditiion` also gets called from `_grant` with the ALLOW_FLAG as the condition. So the check is there and everything is as described in the documentation.

0xean (judge) commented:

> Non-Critical

[06] `DAO` CONTRACT'S `receive()` CAN BE UPDATED TO CALL `DAO.deposit` FUNCTION WITH `_token` INPUT BEING `address(0)`

The `DAO` contract's `receive()` currently emits the `NativeTokenDeposited` event. Yet, the `DAO.deposit` function covers the same purpose of depositing ETH into the DAO, which emits another event that is `Deposited`. Similar to WETH's fallback function, the `DAO` contract's `receive()` can be updated to call the `DAO.deposit` function with the `_token` input being `address(0)`. In this way, only the `Deposited` event is needed, and the off-chain monitoring can become more efficient.

https://github.com/code-423n4/2023-03-aragon/blob/main/packages/contracts/src/core/dao/DAO.sol#L263-L265

```
    receive() external payable {
        emit NativeTokenDeposited(msg.sender, msg.value)
    }
```

https://github.com/code-423n4/2023-03-aragon/blob/main/packages/contracts/src/core/dao/DAO.sol#L218-L236

```
    function deposit(
        address _token,
        uint256 _amount,
        string calldata _reference
    ) external payable override {
        if (_amount == 0) revert ZeroAmount();

        if (_token == address(0)) {
            if (msg.value != _amount)
                revert NativeTokenDepositAmountMismatch(
        } else {
            ...
        }

        emit Deposited(msg.sender, _token, _amount, _refe
    }
```

novaknole20 (Aragon) commented:

> We didn't do it because it increases the gas cost for no beneficial reason.

Oxean (judge) commented:

> Non-Critical

[07] UNLIKE `DAO.deposit`, `DAO` CONTRACT HAS NO FUNCTIONS FOR DEPOSITING ERC721 AND ERC1155 TO DAO

Calling the following `DAO.deposit` function with the `_token` input being not `address(0)` can conveniently deposit an amount of the corresponding ERC20 token to the DAO. Yet, the `DAO` contract has no functions that allow users to directly interact with the DAO for depositing ERC721 and ERC1155 tokens. To provide more convenience to users, please consider adding functions in the `DAO` contract for depositing ERC721 and ERC1155 tokens to the DAO.

https://github.com/code-423n4/2023-03-aragon/blob/main/packages/contracts/src/core/dao/DAO.sol#L218-L236

```
    function deposit(
        address _token,
        uint256 _amount,
        string calldata _reference
    ) external payable override {
        if (_amount == 0) revert ZeroAmount();

        if (_token == address(0)) {
            ...
        } else {
            if (msg.value != 0)
                revert NativeTokenDepositAmountMismatch(

            IERC20Upgradeable(_token).safeTransferFrom(m
        }

        emit Deposited(msg.sender, _token, _amount, _ref
    }
```

novaknole20 (Aragon) commented:

> We don't need the deposit functions for these token standards because they have a callback when the user uses the `safe...` functions and thus we get the events we need for our subgraph.

Oxean (judge) commented:

> Non-Critical

[08] `PermissionManager.applyMultiTargetPermissions` FUNCTION ALREADY COVERS USE CASES OF `PermissionManager.applySingleTargetPermissions` FUNCTION

Calling the following `PermissionManager.applySingleTargetPermissions` function can grant or revoke the permission for `item.who` on a single `_where` target contract; yet, calling this function cannot grant the permission for `item.who` on a single `_where` target contract with condition. However, calling the `PermissionManager.applyMultiTargetPermissions` function can also grant or revoke the permission for `item.who` on a single `item.where` target contract; besides that, calling it can also grant the permission for `item.who` on a single `item.where` target contract with condition, which cannot be achieved by calling the `PermissionManager.applySingleTargetPermissions` function. Since the `PermissionManager.applyMultiTargetPermissions` function covers the use cases of the `PermissionManager.applySingleTargetPermissions` function, the `PermissionManager.applySingleTargetPermissions` function can be considered as redundant. If there is no need to keep the `PermissionManager.applySingleTargetPermissions` function, please consider removing it.

https://github.com/code-423n4/2023-03-aragon/blob/main/packages/contracts/src/core/permission/PermissionManager.sol#L146-L163

```
    function applySingleTargetPermissions(
        address _where,
        PermissionLib.SingleTargetPermission[] calldata i
    ) external virtual auth(ROOT_PERMISSION_ID) {
        for (uint256 i; i < items.length; ) {
            PermissionLib.SingleTargetPermission memory i

            if (item.operation == PermissionLib.Operatio
                _grant(_where, item.who, item.permissionI
            } else if (item.operation == PermissionLib.Op
                _revoke(_where, item.who, item.permissio
            }

            unchecked {
                ++i;
            }
        }
    }
```

https://github.com/code-423n4/2023-03-aragon/blob/main/packages/contracts/src/core/permission/PermissionManager.sol#L167-L190

```
    function applyMultiTargetPermissions(
        PermissionLib.MultiTargetPermission[] calldata _i
    ) external virtual auth(ROOT_PERMISSION_ID) {
        for (uint256 i; i < _items.length; ) {
            PermissionLib.MultiTargetPermission memory it

            if (item.operation == PermissionLib.Operatio
                _grant(item.where, item.who, item.permis
            } else if (item.operation == PermissionLib.Op
                _revoke(item.where, item.who, item.permi
            } else if (item.operation == PermissionLib.Op
                _grantWithCondition(
                    item.where,
                    item.who,
                    item.permissionId,
                    IPermissionCondition(item.condition)
                );
            }

            unchecked {
                ++i;
            }
        }
    }
```

novaknole20 (Aragon) commented:

> Think of `applySingleTragetPermissions` to be the simpler form to grant permissions and also the more cost effective (less gas used).

Oxean (judge) commented:

> Non-Critical

[09] MISSING `address(0)` CHECKS FOR CRITICAL ADDRESS INPUTS

(Please note that the following instances are not found in the known automated findings.)

To prevent unintended behaviors, critical address inputs should be checked against `address(0)`.

`address(0)` check is missing for `_initialOwner` in the following constructor. Please consider checking it. https://github.com/code-423n4/2023-03-aragon/blob/main/packages/contracts/src/core/dao/DAO.sol#L104-L119

```
function initialize(
    bytes calldata _metadata,
    address _initialOwner,
    address _trustedForwarder,
    string calldata daoURI_
) external initializer {
    ...
    __PermissionManager_init(_initialOwner);
}
```

`address(0)` check is missing for `_newTrustedForwarder` in the following function. Please consider checking it. https://github.com/code-423n4/2023-03-aragon/blob/main/packages/contracts/src/core/dao/DAO.sol#L139-L143

```
function setTrustedForwarder(
    address _newTrustedForwarder
) external override auth(SET_TRUSTED_FORWARDER_PERMI
    _setTrustedForwarder(_newTrustedForwarder);
}
```

`address(0)` check is missing for `_signatureValidator` in the following function. Please consider checking it. https://github.com/code-423n4/2023-03-aragon/blob/main/packages/contracts/src/core/dao/DAO.sol#L239-L245

```
function setSignatureValidator(
    address _signatureValidator
) external override auth(SET_SIGNATURE_VALIDATOR_PERM
    signatureValidator = IERC1271(_signatureValidato
    ...
}
```

novaknole20 (Aragon) commented:

> Signature validator and trusted forwarder can be set to 0 to disable the functionatliy. DAO `initialize` only gets called from DAOFactory and there we use the DAOFactory as the initial owner. So no need for the check.

0xean (judge) commented:

> Non-Critical - already declared in **known issues from C4udit**.

[10] REDUNDANT RETURN STATEMENTS FOR FUNCTIONS WITH NAMED RETURNS CAN BE REMOVED

When a function has named returns and a return statement, this return statement becomes redundant. To improve readability and maintainability, the return statements of the following functions can be removed.

https://github.com/code-423n4/2023-03-aragon/blob/main/packages/contracts/src/framework/plugin/setup/PluginSetupProcessor.sol#L286-L344

```
        function prepareInstallation(
            address _dao,
            PrepareInstallationParams calldata _params
        ) external returns (address plugin, IPluginSetup.Pre
            ...
            return (plugin, preparedSetupData);
        }
```

https://github.com/code-423n4/2023-03-aragon/blob/main/packages/contracts/src/framework/plugin/setup/PluginSetupProcessor.sol#L396-L495

```
        function prepareUpdate(
            address _dao,
            PrepareUpdateParams calldata _params
        )
            external
            returns (bytes memory initData, IPluginSetup.Pre
        {
            ...
            return (initData, preparedSetupData);
        }
```

Oxean (judge) commented:

> Non-Critical

[11] VULNERABILITIES IN VERSION 4.8.1 OF `@openzeppelin/contracts` AND `@openzeppelin/contracts-upgradeable`

As shown in the following code in `package.json`, version 4.8.1 of `@openzeppelin/contracts` and `@openzeppelin/contracts-upgradeable` can be used. As described in https://security.snyk.io/package/npm/@openzeppelin%2Fcontracts/4.8.1 and https://security.snyk.io/package/npm/@openzeppelin%2Fcontracts-upgradeable/4.8.1, these versions are vulnerable to incorrect calculation for minting NFTs in batches. To reduce the potential attack surface and be more future-proofed, please consider upgrading this package to at least version 4.8.2.

https://github.com/code-423n4/2023-03-aragon/blob/main/packages/contracts/package.json#L81-L82

```
        "@openzeppelin/contracts": "4.8.1",
        "@openzeppelin/contracts-upgradeable": "4.8.1"
```

novaknole20 (Aragon) commented:

> We don't mint NFTs in batches and don't have the functionality to do so.

Oxean (judge) commented:

> This is more of a general suggestion, should be Non-Critical.

[12] SOLIDITY VERSION `0.8.19` CAN BE USED

As described in https://github.com/ethereum/solidity/releases, Version `0.8.19` is the latest version of Solidity, which "contains a fix for a long-standing bug that can result in code that is only used in creation code to also be included in runtime bytecode". To be more secured and more future-proofed, please consider using Version `0.8.19`. Some files that use Version `0.8.17` currently are shown below.

```
packages\contracts\src\core\dao\DAO.sol
  3: pragma solidity 0.8.17;

packages\contracts\src\core\permission\PermissionLib.sol
  3: pragma solidity 0.8.17;

packages\contracts\src\core\permission\PermissionManager
  3: pragma solidity 0.8.17;

packages\contracts\src\core\utils\CallbackHandler.sol
  3: pragma solidity 0.8.17;
```

novaknole20 (Aragon) commented:

> We decided to used `0.8.17`. No need to upgrade and retest everything for no benefit.

Oxean (judge) commented:

> Non-Critical

[13] DEFINITIONS OF `UNSET_FLAG` AND `ALLOW_FLAG` ARE INCORRECT IN DOCUMENTATION

https://devs.aragon.org/docs/osx/how-it-works/core/permissions/#permissions shows that for `permissionsHashed`, "the `bytes32` keys are the permission hashes and the address values are either zero-address flags, such as `ALLOW_FLAG = address(0)` and `UNSET_FLAG = address(2)` indicating if the permission is set, or an actual address pointing to a PermissionCondition contract". However, `UNSET_FLAG` is set to `address(0)` and `ALLOW_FLAG` is set to `address(2)` in the `PermissionManager` contract. To avoid misinformation, please consider updating the documentation to match the code.

https://github.com/code-423n4/2023-03-aragon/blob/main/packages/contracts/src/core/permission/PermissionManager.sol#L20-L24

```
    /// @notice A special address encoding if a permissi
    address internal constant UNSET_FLAG = address(0);

    /// @notice A special address encoding if a permissi
    address internal constant ALLOW_FLAG = address(2);
```

Oxean (judge) commented:

> Low Risk

[14] `bytes4(0)` CAN BE REPLACED WITH A CONSTANT

To improve readability and maintainability, the following `bytes4(0)` in the `DAO` contract can be replaced with a constant like how the `CallbackHandler` contract does.

https://github.com/code-423n4/2023-03-aragon/blob/main/packages/contracts/src/core/dao/DAO.sol#L248-L258

```
    function isValidSignature(
        bytes32 _hash,
        bytes memory _signature
    ) external view override(IDAO, IERC1271) returns (by
        if (address(signatureValidator) == address(0)) {
            // Return the invalid magic number
            return bytes4(0);
        }
        ...
    }
```

https://github.com/code-423n4/2023-03-aragon/blob/main/packages/contracts/src/core/utils/CallbackHandler.sol#L14

```
    bytes4 internal constant UNREGISTERED_CALLBACK = byte
```

novaknole20 (Aragon) commented:

> No need to use a constant for a value that is only used once.

0xean (judge) commented:

> Non-Critical

[15] `NewURI` EVENT CAN BE MOVED TO `IDAO` INTERFACE

Because all other events of the `DAO` contract are included in the `IDAO` interface, the following `NewURI` event can be moved from the `DAO` contract to the `IDAO` interface for better code organization.

https://github.com/code-423n4/2023-03-aragon/blob/main/packages/contracts/src/core/dao/DAO.sol#L89

```
    event NewURI(string daoURI);
```

Oxean (judge) commented:

> Non-Critical

[16] INPUT VARIABLE CAN BE NAMED WITH LEADING UNDERSCORE

The best practice recommends that a function's input variable can be named with a leading underscore. Please consider renaming `items` to `_items` in the following function.

https://github.com/code-423n4/2023-03-aragon/blob/main/packages/contracts/src/core/permission/PermissionManager.sol#L146-L163

```
    function applySingleTargetPermissions(
        address _where,
        PermissionLib.SingleTargetPermission[] calldata :
    ) external virtual auth(ROOT_PERMISSION_ID) {
        ...
    }
```

Oxean (judge) commented:

> Non-Critical

[17] WORD TYPING TYPOS

(Please note that the following instances are not found in https://gist.github.com/Picodes/16984274f6ad7b83b7a59f8b33cee6a6#nc-5-typos.)

`recieves` can be changed to `receives` in the following comments.

```
packages\contracts\src\core\permission\PermissionManager.
  217: /// @param _where The address of the target contra
  225: /// @param _where The address of the target contra
  278: /// @param _where The address of the target contra
```

`implecitly` can be changed to `implicitly` in the following comment.

```
packages\contracts\src\framework\dao\DAOFactory.sol
  134: // Revoke Temporarly `ROOT_PERMISSION_ID` from `p
```

Oxean (judge) commented:

> Non-Critical

[18] INCOMPLETE NATSPEC COMMENTS

NatSpec comments provide rich code documentation. The following functions miss the `@param` or `@return` comments. Please consider completing the NatSpec comments for these functions.

```
packages\contracts\src\core\dao\DAO.sol
  104: function initialize(

packages\contracts\src\core\dao\IEIP4824.sol
  10: function daoURI() external view returns (string mem

packages\contracts\src\core\utils\CallbackHandler.sol
```

```
31: function _handleCallback(
```

Oxean (judge) commented:

> Non-Critical

Gas Optimizations

For this audit, 18 reports were submitted by wardens detailing gas optimizations. The report highlighted below by **JCN** received the top score from the judge.

The following wardens also submitted reports: 0xSmartContract, Rageur, 0x6980, hunter_w3b, atharvasama, RaymondFam, volodya, yongskiws, ReyAdmirado, Phantasmagoria, matrix_0wl, descharre, 0xnev, saneryee, Rolezn, Madalad, and Sathish9098.

Summary

Certain optimizations were benchmarked to use as baselines for issues + illustrate average savings. All benchmarks were done via the protocol's tests. Instances are illustrated with diffs.

Functions that are not covered in the protocol's tests and/or not benchmarked: gas savings are explained via opcodes & EVM gas costs. Instances are also illustrated with diffs.

Note: Some code snippets may be truncated to save space. Code snippets may also be accompanied by `@audit` tags in comments to aid in explaining the issue.

Gas Optimizations

Number	Issue	Instances	Total Gas Saved
G-01	State variables only set in the constructor should be declared immutable	11	23100
G-02	State variables can be cached instead of re-reading them from storage	16	1600
G-03	Multiple accesses of a mapping/array should use a storage pointer	4	227
G-04	Avoid calling the same internal function multiple times	-	-

[G-01] State variables only set in the constructor should be declared immutable

The solidity compiler will directly embed the values of immutable variables into your contract bytecode and therefore will save you from incurring a `Gsset (20000 gas)` when you set storage variables in the constructor, a `Gcoldsload (2100 gas)` when you access storage variables for the first time in a transaction, and a `Gwarmaccess (100 gas)` for each subsequent access to that storage slot.

Total Instances: 11

Gas Savings: `11 * 2100 = 23100`

https://github.com/code-423n4/2023-03-aragon/blob/main/packages/contracts/src/framework/plugin/repo/PluginRepoFactory.sol#L15-L18

Gas Savings for `createPluginRepo()`, obtained via protocol's tests: Avg 4206 gas | Set `pluginRepoRegistry` and `pluginRepoBase` to `immutable` to save ~4200 gas.

	Min	Max	Avg	# calls
Before	-	-	531312	4

	Min	Max	Avg	# calls
After	-	-	527106	4

```
File: src/framework/plugin/repo/PluginRepoFactory.sol
15:   PluginRepoRegistry public pluginRepoRegistry;
16:
17:    /// @notice The address of the `PluginRepo` base
18:    address public pluginRepoBase;
```

```
diff --git a/src/framework/plugin/repo/PluginRepoFactory
index 9bb5590..88df6de 100644
--- a/src/framework/plugin/repo/PluginRepoFactory.sol
+++ b/src/framework/plugin/repo/PluginRepoFactory.sol
@@ -12,10 +12,10 @@ import {PluginRepo} from "./PluginRe
 /// @notice This contract creates `PluginRepo` proxies
 contract PluginRepoFactory {
     /// @notice The Aragon plugin registry contract.
-    PluginRepoRegistry public pluginRepoRegistry;
+    PluginRepoRegistry public immutable pluginRepoRegis

     /// @notice The address of the `PluginRepo` base co
-    address public pluginRepoBase;
+    address public immutable pluginRepoBase;
```

https://github.com/code-423n4/2023-03-aragon/blob/main/packages/contracts/src/framework/plugin/setup/PluginSetupProcessor.sol#L128

Set `repoRegistry` as `immutable` to save ~2100 gas

```
File: src/framework/plugin/setup/PluginSetupProcessor.so
127:    /// @notice The plugin repo registry listing the
128:    PluginRepoRegistry public repoRegistry;
```

```
diff --git a/src/framework/plugin/setup/PluginSetupProces
index 41f68e5..4278c4b 100644
--- a/src/framework/plugin/setup/PluginSetupProcessor.so
+++ b/src/framework/plugin/setup/PluginSetupProcessor.so
@@ -125,7 +125,7 @@ contract PluginSetupProcessor {
     }

     /// @notice The plugin repo registry listing the `P
-    PluginRepoRegistry public repoRegistry;
+    PluginRepoRegistry public immutable repoRegistry;
```

https://github.com/code-423n4/2023-03-aragon/blob/main/packages/contracts/src/framework/utils/TokenFactory.sol#L26-L36

Move `setupBases()` logic into the `constructor` and set `governanceERC20Base`, `governanceWrappedERC20Base`, `merkleMinterBase`, and `distributorBase` to `immutable` to save ~8400 gas.

Note that `setupBases()` *is a private function only called in the constructor.*

```
File: src/framework/utils/TokenFactory.sol
26:     /// @notice The address of the `GovernanceERC20` 
27:     address public governanceERC20Base;
28:
29:     /// @notice The address of the `GovernanceWrappedE
30:     address public governanceWrappedERC20Base;
31:
32:     /// @notice The address of the `MerkleMinter` base
33:     address public merkleMinterBase;
34:
35:     /// @notice The `MerkleDistributor` base contract
36:     MerkleDistributor public distributorBase;
```

```
diff --git a/src/framework/utils/TokenFactory.sol b/src/
```

```diff
index 381e745..c24b133 100644
--- a/src/framework/utils/TokenFactory.sol
+++ b/src/framework/utils/TokenFactory.sol
@@ -24,16 +24,16 @@ contract TokenFactory {
     using Clones for address;

     /// @notice The address of the `GovernanceERC20` ba
-    address public governanceERC20Base;
+    address public immutable governanceERC20Base;

     /// @notice The address of the `GovernanceWrappedER
-    address public governanceWrappedERC20Base;
+    address public immutable governanceWrappedERC20Base

     /// @notice The address of the `MerkleMinter` base 
-    address public merkleMinterBase;
+    address public immutable merkleMinterBase;

     /// @notice The `MerkleDistributor` base contract u
-    MerkleDistributor public distributorBase;
+    MerkleDistributor public immutable distributorBase;

     /// @notice Emitted when a new token is created.
     /// @param token [ERC-20](https://eips.ethereum.org
@@ -66,7 +66,19 @@ contract TokenFactory {

     /// @notice Initializes the different base contract
     constructor() {
-        setupBases();
+        distributorBase = new MerkleDistributor();
+        governanceERC20Base = address(
+            new GovernanceERC20(
+                IDAO(address(0)),
+                "baseName",
+                "baseSymbol",
+                GovernanceERC20.MintSettings(new addres
+            )
+        );
+        governanceWrappedERC20Base = address(
+            new GovernanceWrappedERC20(IERC20Upgradeabl
+        );
+        merkleMinterBase = address(new MerkleMinter());
     }
```

https://github.com/code-423n4/2023-03-aragon/blob/main/packages/contracts/src/plugins/counter-example/v1/CounterV1PluginSetup.sol#L19-L21

Set `multiplyHelperBase` and `counterBase` to `immutable` to save ~4200 gas.

```
File: src/plugins/counter-example/v1/CounterV1PluginSetu
19:    // For testing purposes, the below are public...
20:    MultiplyHelper public multiplyHelperBase;
21:    CounterV1 public counterBase;
```

```
diff --git a/src/plugins/counter-example/v1/CounterV1Plug
index 7af94ab..a7ce824 100644
--- a/src/plugins/counter-example/v1/CounterV1PluginSetu
+++ b/src/plugins/counter-example/v1/CounterV1PluginSetu
@@ -17,8 +17,8 @@ contract CounterV1PluginSetup is Plugi
     using Clones for address;

     // For testing purposes, the below are public...
-    MultiplyHelper public multiplyHelperBase;
-    CounterV1 public counterBase;
+    MultiplyHelper public immutable multiplyHelperBase;
+    CounterV1 public immutable counterBase;
```

https://github.com/code-423n4/2023-03-aragon/blob/main/packages/contracts/src/plugins/counter-example/v2/CounterV2PluginSetup.sol#L19-L21

Set `multiplyHelperBase` and `counterBase` to `immutable` to save ~4200 gas.

```
File: src/plugins/counter-example/v2/CounterV2PluginSetu
19:     // For testing purposes, the contracts below are 
20:     MultiplyHelper public multiplyHelperBase;
21:     CounterV2 public counterBase;
```

```
diff --git a/src/plugins/counter-example/v2/CounterV2Plu
index 527074b..8d73f02 100644
--- a/src/plugins/counter-example/v2/CounterV2PluginSetu
+++ b/src/plugins/counter-example/v2/CounterV2PluginSetu
@@ -17,8 +17,8 @@ contract CounterV2PluginSetup is Plugi
     using Clones for address;

     // For testing purposes, the contracts below are pu
-    MultiplyHelper public multiplyHelperBase;
-    CounterV2 public counterBase;
+    MultiplyHelper public immutable multiplyHelperBase;
+    CounterV2 public immutable counterBase;
```

[G-02] State variables can be cached instead of re-reading them from storage

Caching of a state variable replaces each `Gwarmaccess (100 gas)` with a much cheaper stack read.

Note these are instances that the c4udit tool missed.

Total Instances: 16

Gas savings: `11 * 100 + 501 (5 benchmarked instances) = 1601`

https://github.com/code-423n4/2023-03-aragon/blob/main/packages/contracts/src/framework/plugin/repo/PluginRepo.sol#L128-L153

Gas Savings for `createVersion()`, obtained via protocol's tests:

Avg 107 gas | Cache `latestRelease` as a stack variable to save 1 SLOAD

	Min	Max	Avg	# calls
Before	121713	178154	161829	40
After	121606	178047	161722	40

```
File: src/framework/plugin/repo/PluginRepo.sol
128:    function createVersion(
129:        uint8 _release,
130:        address _pluginSetup,
131:        bytes calldata _buildMetadata,
132:        bytes calldata _releaseMetadata
133:    ) external auth(MAINTAINER_PERMISSION_ID) {
134:        if (!_pluginSetup.supportsInterface(type(IPl
135:            revert InvalidPluginSetupInterface();
136:        }
137:
138:        if (_release == 0) {
139:            revert ReleaseZeroNotAllowed();
140:        }
141:
142:        // Check that the release number is not incr
143:        if (_release - latestRelease > 1) { // @audi
144:            revert InvalidReleaseIncrement({latestRe
145:        }
146:
147:        if (_release > latestRelease) { // @audit: 2
148:            latestRelease = _release;
149:
150:            if (_releaseMetadata.length == 0) {
151:                revert EmptyReleaseMetadata();
152:            }
153:        }
```

```
diff --git a/src/framework/plugin/repo/PluginRepo.sol b/
index 6dc2c8b..dfde528 100644
--- a/src/framework/plugin/repo/PluginRepo.sol
+++ b/src/framework/plugin/repo/PluginRepo.sol
```

```diff
@@ -140,11 +140,12 @@ contract PluginRepo is
         }

         // Check that the release number is not incremen
-        if (_release - latestRelease > 1) {
-            revert InvalidReleaseIncrement({latestRelea
+        uint8 _latestRelease = latestRelease;
+        if (_release - _latestRelease > 1) {
+            revert InvalidReleaseIncrement({latestRelea
         }

-        if (_release > latestRelease) {
+        if (_release > _latestRelease) {
             latestRelease = _release;
```

https://github.com/code-423n4/2023-03-aragon/blob/main/packages/contracts/src/framework/utils/ens/ENSSubdomainRegistrar.sol#L82-L96

Gas Savings for `createVersion()`, obtained via protocol's tests: Avg 394 gas | Cache `node`, `ens`, and `resolver` to save 4 SLOADs

	Min	Max	Avg	# calls
Before	140546	142827	141691	14
After	140152	142433	141297	14

```
File: src/framework/utils/ens/ENSSubdomainRegistrar.sol
82:     function registerSubnode(
83:         bytes32 _label,
84:         address _targetAddress
85:     ) external auth(REGISTER_ENS_SUBDOMAIN_PERMISSION_
86:         bytes32 subnode = keccak256(abi.encodePacked(
87:         address currentOwner = ens.owner(subnode); //
88:
89:         if (currentOwner != address(0)) {
90:             revert AlreadyRegistered(subnode, current
91:         }
92:
        ens.setSubnodeOwner(node, _label, address(thi
        ens.setResolver(subnode, resolver); // @audit
        Resolver(resolver).setAddr(subnode, _targetAd
    }
```

```
diff --git a/src/framework/utils/ens/ENSSubdomainRegistra
index 0d5ea5e..3ed14bd 100644
--- a/src/framework/utils/ens/ENSSubdomainRegistrar.sol
+++ b/src/framework/utils/ens/ENSSubdomainRegistrar.sol
@@ -83,16 +83,19 @@ contract ENSSubdomainRegistrar is UUI
         bytes32 _label,
         address _targetAddress
     ) external auth(REGISTER_ENS_SUBDOMAIN_PERMISSION_II
-        bytes32 subnode = keccak256(abi.encodePacked(noc
-        address currentOwner = ens.owner(subnode);
+        bytes32 _node = node;
+        ENS _ens = ens;
+        address _resolver = resolver;
+        bytes32 subnode = keccak256(abi.encodePacked(_nc
+        address currentOwner = _ens.owner(subnode);

         if (currentOwner != address(0)) {
             revert AlreadyRegistered(subnode, currentOwn
         }

-        ens.setSubnodeOwner(node, _label, address(this)
-        ens.setResolver(subnode, resolver);
-        Resolver(resolver).setAddr(subnode, _targetAddre
+        _ens.setSubnodeOwner(_node, _label, address(thi
+        _ens.setResolver(subnode, _resolver);
+        Resolver(_resolver).setAddr(subnode, _targetAdd
     }
```

https://github.com/code-423n4/2023-03-aragon/blob/main/packages/contracts/src/plugins/token/MerkleMinter.sol#L74-L95

Cache `token` as a stack variable to save 1 SLOAD

```
File: src/plugins/token/MerkleMinter.sol
    function merkleMint(
        bytes32 _merkleRoot,
        uint256 _totalAmount,
        bytes calldata _tree,
        bytes calldata _context
    ) external override auth(MERKLE_MINT_PERMISSION_II
        address distributorAddr = createERC1967Proxy(
            address(distributorBase),
            abi.encodeWithSelector(
                MerkleDistributor.initialize.selector,
                dao(),
                IERC20Upgradeable(address(token)), //
                _merkleRoot
            )
        );

        token.mint(distributorAddr, _totalAmount); //

        emit MerkleMinted(distributorAddr, _merkleRoo

        return IMerkleDistributor(distributorAddr);
    }
```

```
diff --git a/src/plugins/token/MerkleMinter.sol b/src/plu
index fab8959..a372746 100644
--- a/src/plugins/token/MerkleMinter.sol
+++ b/src/plugins/token/MerkleMinter.sol
@@ -77,17 +77,18 @@ contract MerkleMinter is IMerkleMinte
         bytes calldata _tree,
         bytes calldata _context
     ) external override auth(MERKLE_MINT_PERMISSION_ID)
+        IERC20MintableUpgradeable _token = token;
         address distributorAddr = createERC1967Proxy(
             address(distributorBase),
             abi.encodeWithSelector(
                 MerkleDistributor.initialize.selector,
                 dao(),
-                IERC20Upgradeable(address(token)),
+                IERC20Upgradeable(address(_token)),
                 _merkleRoot
             )
         );

-           token.mint(distributorAddr, _totalAmount);
+           _token.mint(distributorAddr, _totalAmount);

         emit MerkleMinted(distributorAddr, _merkleRoot,
```

https://github.com/code-423n4/2023-03-aragon/blob/main/packages/contracts/src/core/dao/DAO.sol#L248-L258

Cache `signatureValidator` as a stack variable to save 1 SLOAD

```
File: src/core/dao/DAO.sol
248:    function isValidSignature(
249:        bytes32 _hash,
250:        bytes memory _signature
251:    ) external view override(IDAO, IERC1271) returns
252:        if (address(signatureValidator) == address(0
253:            // Return the invalid magic number
254:            return bytes4(0);
255:        }
256:        // Forward the call to the set signature val
257:        return signatureValidator.isValidSignature(_
258:    }
```

```
diff --git a/src/core/dao/DAO.sol b/src/core/dao/DAO.sol
index d7c912d..9b3379f 100644
--- a/src/core/dao/DAO.sol
+++ b/src/core/dao/DAO.sol
@@ -249,12 +249,13 @@ contract DAO is
          bytes32 _hash,
          bytes memory _signature
      ) external view override(IDAO, IERC1271) returns (by
-         if (address(signatureValidator) == address(0))
+         IERC1271 _signatureValidator = signatureValidat
+        if (address(_signatureValidator) == address(0))
             // Return the invalid magic number
             return bytes4(0);
         }
         // Forward the call to the set signature valida
-        return signatureValidator.isValidSignature(_hasl
+        return _signatureValidator.isValidSignature(_has
     }
```

https://github.com/code-423n4/2023-03-aragon/blob/main/packages/contracts/src/framework/plugin/repo/PluginRepo.sol#L155-L163

Cache `version.tag.release` as a stack variable to save 2 SLOADs

```
File: src/framework/plugin/repo/PluginRepo.sol
155:        // Make sure the same plugin setup wasn't use
156:        Version storage version = versions[latestTagH
157:        if (version.tag.release != 0 && version.tag.
158:            revert PluginSetupAlreadyInPreviousReleas
159:                version.tag.release, // @audit: 3rd s
160:                version.tag.build,
161:                _pluginSetup
162:            );
163:        }
```

```
diff --git a/src/framework/plugin/repo/PluginRepo.sol b/s
index 6dc2c8b..4c18f74 100644
--- a/src/framework/plugin/repo/PluginRepo.sol
+++ b/src/framework/plugin/repo/PluginRepo.sol
@@ -154,9 +154,10 @@ contract PluginRepo is

         // Make sure the same plugin setup wasn't used i
         Version storage version = versions[latestTagHasl
-        if (version.tag.release != 0 && version.tag.rele
+        uint8 prevRelease = version.tag.release;
+        if (prevRelease != 0 && prevRelease != _release)
```

```
                revert PluginSetupAlreadyInPreviousRelease(
-                   version.tag.release,
+                   prevRelease,
                    version.tag.build,
                    _pluginSetup
                );
```

https://github.com/code-423n4/2023-03-aragon/blob/main/packages/contracts/src/plugins/counter-example/v1/CounterV1PluginSetup.sol#L43-L88

Cache `counterBase` and `multiplyHelperBase` as stack variables to save 2 SLOADs

```
File: src/plugins/counter-example/v1/CounterV1PluginSetu
43:         if (_multiplyHelper == address(0)) {
44:             multiplyHelper = createERC1967Proxy(addre
45:         }
46:
47:         bytes memory initData = abi.encodeWithSelecto
48:             bytes4(keccak256("initialize(address,addr
49:             _dao,
50:             multiplyHelper,
51:             _num
52:         );
53:
54:         PermissionLib.MultiTargetPermission[]
55:             memory permissions = new PermissionLib.Mu
56:                 _multiplyHelper == address(0) ? 3 : 2
57:             );
58:         address[] memory helpers = new address[](1);
59:
60:         // deploy
61:         plugin = createERC1967Proxy(address(counterBa
62:
63:         // set permissions
64:         permissions[0] = PermissionLib.MultiTargetPer
65:             PermissionLib.Operation.Grant,
66:             _dao,
            plugin,
            PermissionLib.NO_CONDITION,
            keccak256("EXECUTE_PERMISSION")
        );

        permissions[1] = PermissionLib.MultiTargetPerı
            PermissionLib.Operation.Grant,
            plugin,
            _dao,
            PermissionLib.NO_CONDITION,
            counterBase.MULTIPLY_PERMISSION_ID() // @ɑ
        );

        if (_multiplyHelper == address(0)) {
            permissions[2] = PermissionLib.MultiTarge
                PermissionLib.Operation.Grant,
                multiplyHelper,
                plugin,
                PermissionLib.NO_CONDITION,
                multiplyHelperBase.MULTIPLY_PERMISSIOI
            );
        }
```

```
diff --git a/src/plugins/counter-example/v1/CounterV1Plu
index 7af94ab..85d2dc6 100644
--- a/src/plugins/counter-example/v1/CounterV1PluginSetu
+++ b/src/plugins/counter-example/v1/CounterV1PluginSetu
@@ -39,9 +39,9 @@ contract CounterV1PluginSetup is Plugi
         (address _multiplyHelper, uint256 _num) = abi.de

         address multiplyHelper = _multiplyHelper;
-
+        MultiplyHelper _multiplyHelperBase = multiplyHe
         if (_multiplyHelper == address(0)) {
-            multiplyHelper = createERC1967Proxy(address
+            multiplyHelper = createERC1967Proxy(address
         }

         bytes memory initData = abi.encodeWithSelector(
@@ -58,7 +58,8 @@ contract CounterV1PluginSetup is Plugi
         address[] memory helpers = new address[](1);
         // deploy
-        plugin = createERC1967Proxy(address(counterBase
+        CounterV1 _counterBase = counterBase;
+        plugin = createERC1967Proxy(address(_counterBase

         // set permissions
         permissions[0] = PermissionLib.MultiTargetPermis
@@ -74,7 +75,7 @@ contract CounterV1PluginSetup is Plugi
             plugin,
             _dao,
             PermissionLib.NO_CONDITION,
-            counterBase.MULTIPLY_PERMISSION_ID()
+            _counterBase.MULTIPLY_PERMISSION_ID()
         );

         if (_multiplyHelper == address(0)) {
@@ -83,7 +84,7 @@ contract CounterV1PluginSetup is Plugi
                 multiplyHelper,
                 plugin,
                 PermissionLib.NO_CONDITION,
-                multiplyHelperBase.MULTIPLY_PERMISSION_
+                _multiplyHelperBase.MULTIPLY_PERMISSION_
             );
         }
```

https://github.com/code-423n4/2023-03-aragon/blob/main/packages/contracts/src/plugins/counter-example/v2/CounterV2PluginSetup.sol#L44-L89

Cache `counterBase` and `multiplyHelperBase` as stack variables to save 2 SLOADs (Identical to instance above)

https://github.com/code-423n4/2023-03-aragon/blob/main/packages/contracts/src/plugins/governance/majority-voting/MajorityVotingBase.sol#L313-L321

Cache `proposal_.parameters.supportThreshold` as a stack variable to save 1 SLOAD

```
File: src/plugins/governance/majority-voting/MajorityVot
    function isSupportThresholdReached(uint256 _prop
        Proposal storage proposal_ = proposals[_prop

        // The code below implements the formula of
        // `(1 - supportThreshold) * N_yes > support
        return
            (RATIO_BASE - proposal_.parameters.suppo
            proposal_.parameters.supportThreshold * 
    }
```

```
diff --git a/src/plugins/governance/majority-voting/Majo
index 58ba2fe..d6ce8b0 100644
--- a/src/plugins/governance/majority-voting/MajorityVot
+++ b/src/plugins/governance/majority-voting/MajorityVot
@@ -315,9 +315,10 @@ abstract contract MajorityVotingBas

         // The code below implements the formula of the
         // `(1 - supportThreshold) * N_yes > supportThre
+        uint32 _supportThreshold = proposal_.parameters
         return
-            (RATIO_BASE - proposal_.parameters.supportT
-            proposal_.parameters.supportThreshold * pro
+            (RATIO_BASE - _supportThreshold) * proposal_
+            _supportThreshold * proposal_.tally.no;
     }
```

https://github.com/code-423n4/2023-03-aragon/blob/main/packages/contracts/src/plugins/governance/majority-voting/MajorityVotingBase.sol#L324-L338

Cache `proposal_.tally.yes` and `proposal_.parameters.supportThreshold` as stack variables to save 2 SLOADs

```
File: src/plugins/governance/majority-voting/MajorityVot
```

```
    function isSupportThresholdReachedEarly(
        uint256 _proposalId
    ) public view virtual returns (bool) {
       Proposal storage proposal_ = proposals[_propos
        
        uint256 noVotesWorstCase = totalVotingPower(
            proposal_.tally.yes - // @audit: 1st slo
            proposal_.tally.abstain;
        
        // The code below implements the formula of 
        // `(1 - supportThreshold) * N_yes > support
        return
            (RATIO_BASE - proposal_.parameters.suppo
            proposal_.parameters.supportThreshold * 
    }
```

```
diff --git a/src/plugins/governance/majority-voting/Majo
index 58ba2fe..e6cf684 100644
--- a/src/plugins/governance/majority-voting/MajorityVot
+++ b/src/plugins/governance/majority-voting/MajorityVot
@@ -325,16 +325,18 @@ abstract contract MajorityVotingBa
         uint256 _proposalId
     ) public view virtual returns (bool) {
         Proposal storage proposal_ = proposals[_proposa
+        uint256 _yes = proposal_.tally.yes;
+        uint32 _supportThreshold = proposal_.parameters

         uint256 noVotesWorstCase = totalVotingPower(prop
-            proposal_.tally.yes -
+            _yes -
             proposal_.tally.abstain;

         // The code below implements the formula of the
         // `(1 - supportThreshold) * N_yes > supportThre
         return
-            (RATIO_BASE - proposal_.parameters.supportTh
-            proposal_.parameters.supportThreshold * noVo
+            (RATIO_BASE - _supportThreshold) * _yes >
+            _supportThreshold * noVotesWorstCase;
     }
```

[G-03] Multiple accesses of a mapping/array should use a storage pointer

Caching a mapping's value in a storage pointer when the value is accessed multiple times saves ~40 gas per access due to not having to perform the same offset calculation every time. Help the Optimizer by saving a storage variable's reference instead of repeatedly fetching it.

To achieve this, declare a storage pointer for the variable and use it instead of repeatedly fetching the reference in a map or an array. As an example, instead of repeatedly calling `proposals[_proposalId]`, save its reference via a storage pointer: `Proposal storage proposal_ = proposals[_proposalId]` and use the pointer instead.

Total instances: 4

Gas savings: `2 * 40 + 147 (from 2 benchmarked instances) = 227`

https://github.com/code-423n4/2023-03-aragon/blob/main/packages/contracts/src/plugins/governance/multisig/Multisig.sol#L348-L359

Gas Savings for `execute()`, obtained via protocol's tests: Avg 147 gas | Use storage pointer `proposal_` instead of re-accessing the mapping.

	Min	Max	Avg	# calls
Before	70489	72989	71991	10
After	70342	72842	71844	10

```
File: src/plugins/governance/multisig/Multisig.sol
348:    function _execute(uint256 _proposalId) internal 
349:        Proposal storage proposal_ = proposals[_prop
350:
351:        proposal_.executed = true;
352:
353:        _executeProposal(
354:            dao(),
355:            _proposalId,
356:            proposals[_proposalId].actions, // @audi
357:            proposals[_proposalId].allowFailureMap /
358:        );
359:    }
```

```diff
diff --git a/src/plugins/governance/multisig/Multisig.so
index d2dd072..eba5457 100644
--- a/src/plugins/governance/multisig/Multisig.sol
+++ b/src/plugins/governance/multisig/Multisig.sol
@@ -353,8 +353,8 @@ contract Multisig is
         _executeProposal(
             dao(),
             _proposalId,
-            proposals[_proposalId].actions,
-            proposals[_proposalId].allowFailureMap
+            proposal_.actions,
+            proposal_.allowFailureMap
         );
     }
```

https://github.com/code-423n4/2023-03-aragon/blob/main/packages/contracts/src/plugins/governance/majority-voting/MajorityVotingBase.sol#L457-L466

`proposals[_proposalId]` can be cached as a storage pointer

```
File: src/plugins/governance/majority-voting/MajorityVot
457:    function _execute(uint256 _proposalId) internal
458:        proposals[_proposalId].executed = true;
459:
460:        _executeProposal(
461:            dao(),
462:            _proposalId,
                proposals[_proposalId].actions,
                proposals[_proposalId].allowFailureMap
            );
        }
```

```
diff --git a/src/plugins/governance/majority-voting/Majo
index 58ba2fe..3985b7b 100644
--- a/src/plugins/governance/majority-voting/MajorityVot
+++ b/src/plugins/governance/majority-voting/MajorityVot
@@ -455,13 +455,14 @@ abstract contract MajorityVotingBa
     /// @notice Internal function to execute a vote. It
     /// @param _proposalId The ID of the proposal.
     function _execute(uint256 _proposalId) internal vir
-        proposals[_proposalId].executed = true;
+        Proposal storage proposal_ = proposals[_proposa
+        proposal_.executed = true;

         _executeProposal(
             dao(),
             _proposalId,
-            proposals[_proposalId].actions,
-            proposals[_proposalId].allowFailureMap
+            proposal_.actions,
+            proposal_.allowFailureMap
         );
     }
```

[G-04] Avoid calling the same internal function multiple times

Calling an internal function will cost at least ~16-20 gas (JUMP to internal function instructions and JUMP back to original instructions). If you call the same internal function multiple times you can save gas by caching the return value of the internal function.

https://github.com/code-423n4/2023-03-aragon/blob/main/packages/contracts/src/plugins/governance/majority-voting/token/TokenVoting.sol#L91-L102

Cache value from `_msgSender()`

```
File: src/plugins/governance/majority-voting/token/Token\
91:         if (votingToken.getPastVotes(_msgSender(), sna
92:             revert ProposalCreationForbidden(_msgSende
93:         }
94:
95:         proposalId = _createProposal({
96:             _creator: _msgSender(),
97:             _metadata: _metadata,
98:             _startDate: _startDate,
99:             _endDate: _endDate,
100:             _actions: _actions,
101:             _allowFailureMap: _allowFailureMap
102:         });
```

```
diff --git a/src/plugins/governance/majority-voting/toke
index a3e26c3..820d3bc 100644
--- a/src/plugins/governance/majority-voting/token/Token\
+++ b/src/plugins/governance/majority-voting/token/Token\
@@ -88,12 +88,13 @@ contract TokenVoting is IMembership,
             revert NoVotingPower();
         }
 
-        if (votingToken.getPastVotes(_msgSender(), snaps
-            revert ProposalCreationForbidden(_msgSender
+        address sender = _msgSender();
+        if (votingToken.getPastVotes(sender, snapshotBl
+            revert ProposalCreationForbidden(sender);
         }
 
         proposalId = _createProposal({
-            _creator: _msgSender(),
+            _creator: sender,
             _metadata: _metadata,
             _startDate: _startDate,
                _endDate: _endDate,
```

https://github.com/code-423n4/2023-03-aragon/blob/main/packages/contracts/src/plugins/governance/majority-voting/addresslist/AddresslistVoting.sol#L97-L108

```
File: src/plugins/governance/majority-voting/addresslist,
97:         if (minProposerVotingPower() != 0 && !isListe
98:             revert ProposalCreationForbidden(_msgSend
99:         }
100:
101:         proposalId = _createProposal({
102:             _creator: _msgSender(),
103:             _metadata: _metadata,
104:             _startDate: _startDate,
105:             _endDate: _endDate,
106:             _actions: _actions,
107:             _allowFailureMap: _allowFailureMap
108:         });
```

```
diff --git a/src/plugins/governance/majority-voting/addre
index 0ccd8a3..df9912d 100644
--- a/src/plugins/governance/majority-voting/addresslist,
+++ b/src/plugins/governance/majority-voting/addresslist,
@@ -93,13 +93,14 @@ contract AddresslistVoting is IMembe
         unchecked {
             snapshotBlock = block.number.toUint64() - 1
         }
-
-        if (minProposerVotingPower() != 0 && !isListedA
-            revert ProposalCreationForbidden(_msgSender
+
+        address sender = _msgSender();
+        if (minProposerVotingPower() != 0 && !isListedA
+            revert ProposalCreationForbidden(sender);
         }

         proposalId = _createProposal({
```

```
-            _creator: _msgSender(),
+            _creator: sender,
             _metadata: _metadata,
             _startDate: _startDate,
             _endDate: _endDate,
```

https://github.com/code-423n4/2023-03-aragon/blob/main/packages/contracts/src/plugins/governance/multisig/Multisig.sol#L216-L237

```
File: src/plugins/governance/multisig/Multisig.sol
216:        if (multisigSettings.onlyListed && !isListed
217:            revert ProposalCreationForbidden(_msgSen
218:        }
219:
220:        if (_startDate == 0) {
221:            _startDate = block.timestamp.toUint64();
222:        } else if (_startDate < block.timestamp.toUi
223:            revert DateOutOfBounds({limit: block.tim
224:        }
225:
226:        if (_endDate < _startDate) {
227:            revert DateOutOfBounds({limit: _startDat
228:        }
229:
230:        proposalId = _createProposal({
231:            _creator: _msgSender(),
232:            _metadata: _metadata,
233:            _startDate: _startDate,
234:            _endDate: _endDate,
235:            _actions: _actions,
236:            _allowFailureMap: _allowFailureMap
237:        });
```

```
diff --git a/src/plugins/governance/multisig/Multisig.so
index d2dd072..f64e8ec 100644
--- a/src/plugins/governance/multisig/Multisig.sol
+++ b/src/plugins/governance/multisig/Multisig.sol
```

```diff
@@ -212,9 +212,10 @@ contract Multisig is
         uint64 _endDate
     ) external returns (uint256 proposalId) {
         uint64 snapshotBlock = block.number.toUint64() -
-
-        if (multisigSettings.onlyListed && !isListedAtB
-            revert ProposalCreationForbidden(_msgSender
+
+        address sender = _msgSender();
+        if (multisigSettings.onlyListed && !isListedAtB
+            revert ProposalCreationForbidden(sender);
         }

         if (_startDate == 0) {
@@ -228,7 +229,7 @@ contract Multisig is
         }

         proposalId = _createProposal({
-            _creator: _msgSender(),
+            _creator: sender,
             _metadata: _metadata,
             _startDate: _startDate,
             _endDate: _endDate,
```

Oxean (judge) commented:

> While some of these have been identified in the **known issues** - the warden does a good job of going well beyond the known issues and showing the actual saving.

Oxean (judge) commented:

> This report may have less findings than some other Grade-A reports, but they are validated in code, which I think makes them significantly more valuable. Awarding as Best.

novaknole20 (Aragon) acknowledged

Disclosures

C4 is an open organization governed by participants in the community.

C4 Audits incentivize the discovery of exploits, vulnerabilities, and bugs in smart contracts. Security researchers are rewarded at an increasing rate for finding higher-risk issues. Audit submissions are judged by a knowledgeable security researcher and solidity developer and disclosed to sponsoring developers. C4 does not conduct formal verification regarding the provided code but instead provides final verification.

C4 does not provide any guarantee or warranty regarding the security of this project. All smart contract software should be used at the sole risk and responsibility of users.

Top