// HALBORN

Aragon - aragonOS v1.3.0

Smart Contract Security Assessment

Prepared by: **Halborn**

Date of Engagement: **May 29th, 2023 - June 13th, 2023**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE
0.1	Document Creation	06/13/2023
0.2	Draft Review	06/13/2023
1.0	Remediation Plan	06/19/2023
1.1	Remediation Plan Review	06/19/2023

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Ferran Celades	Halborn	Ferran.Celades@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Aragon engaged Halborn to conduct a security assessment on their smart contracts beginning on May 29th, 2023 and ending on June 13th, 2023. The security assessment was scoped to the smart contracts provided to the Halborn team.

1.2 ASSESSMENT SUMMARY

The team at Halborn was provided four weeks for the engagement and assigned a full-time security engineer to verify the security of the smart contract. The security engineer is a blockchain and smart-contract security expert with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this assessment is to:

- Ensure that smart contract functions operate as intended
- Identify potential security issues with the smart contracts

In summary, Halborn identified some security risks that were successfully addressed by the Aragon team.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of this assessment. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of the bridge code and can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of the assessment:

- Research into architecture and purpose
- Smart contract manual code review and walk-through
- Manual assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes
- Manual testing by custom scripts
- Static Analysis of security for scoped contract, and imported functions. (Slither)
- Local deployment (Hardhat, Remix IDE, Brownie)

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. The quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that were used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.
3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

10 - CRITICAL
9 - 8 - HIGH
7 - 6 - MEDIUM
5 - 4 - LOW
3 - 1 - VERY LOW AND INFORMATIONAL

1.4 SCOPE

The security assessment was scoped to the following smart contracts:

- core/dao/DAO.sol
- core/dao/IDAO.sol
- core/dao/IEIP4824.sol
- core/permission/IPermissionCondition.sol
- core/permission/PermissionLib.sol
- core/permission/PermissionManager.sol
- core/plugin/IPlugin.sol
- core/plugin/Plugin.sol
- core/plugin/PluginCloneable.sol
- core/plugin/PluginUUPSUpgradeable.sol
- core/plugin/dao-authorizable/DaoAuthorizable.sol
- core/plugin/dao-authorizable/DaoAuthorizableUpgradeable.sol
- core/plugin/membership/IMembership.sol
- core/plugin/proposal/IProposal.sol
- core/plugin/proposal/Proposal.sol
- core/plugin/proposal/ProposalUpgradeable.sol
- core/utils/BitMap.sol
- core/utils/CallbackHandler.sol
- core/utils/auth.sol
- framework/utils/InterfaceBasedRegistry.sol
- framework/utils/RegistryUtils.sol

- framework/utils/TokenFactory.sol
- framework/utils/ens/ENSMigration.sol
- framework/utils/ens/ENSSubdomainRegistrar.sol
- plugins/token/IMerkleDistributor.sol
- plugins/token/IMerkleMinter.sol
- plugins/token/MerkleDistributor.sol
- plugins/token/MerkleMinter.sol
- framework/dao/DAOFactory.sol
- framework/dao/DAORegistry.sol
- framework/plugin/repo/IPluginRepo.sol
- framework/plugin/repo/PluginRepo.sol
- framework/plugin/repo/PluginRepoFactory.sol
- framework/plugin/repo/PluginRepoRegistry.sol
- framework/plugin/setup/IPluginSetup.sol
- framework/plugin/setup/PluginSetup.sol
- framework/plugin/setup/PluginSetupProcessor.sol
- framework/plugin/setup/PluginSetupProcessorHelpers.sol
- plugins/counter-example/MultiplyHelper.sol
- plugins/counter-example/v1/CounterV1.sol
- plugins/counter-example/v1/CounterV1PluginSetup.sol
- plugins/counter-example/v2/CounterV2.sol
- plugins/counter-example/v2/CounterV2PluginSetup.sol
- plugins/governance/admin/Admin.sol
- plugins/governance/admin/AdminSetup.sol

- plugins/governance/majority-voting/IMajorityVoting.sol
- plugins/governance/majority-voting/MajorityVotingBase.sol
- plugins/governance/majority-voting/addresslist/AddresslistVoting.sol
- plugins/governance/majority-voting/addresslist/AddresslistVotingSetup.sol
- plugins/governance/majority-voting/token/TokenVoting.sol
- plugins/governance/majority-voting/token/TokenVotingSetup.sol
- plugins/governance/multisig/IMultisig.sol
- plugins/governance/multisig/Multisig.sol
- plugins/governance/multisig/MultisigSetup.sol
- plugins/placeholder-version/PlaceholderSetup.sol
- plugins/utils/Addresslist.sol
- plugins/utils/Ratio.sol
- token/ERC20/IERC20MintableUpgradeable.sol
- token/ERC20/governance/GovernanceERC20.sol
- token/ERC20/governance/GovernanceWrappedERC20.sol
- token/ERC20/governance/IGovernanceWrappedERC20.sol
- utils/Proxy.sol
- utils/UncheckedMath.sol
- utils/protocol/IProtocolVersion.sol
- utils/protocol/ProtocolVersion.sol
- Deployment scripts were also reviewed

Branch: d49a55a78b41436219a377cf23542ad09a85609a
Pull request: 395

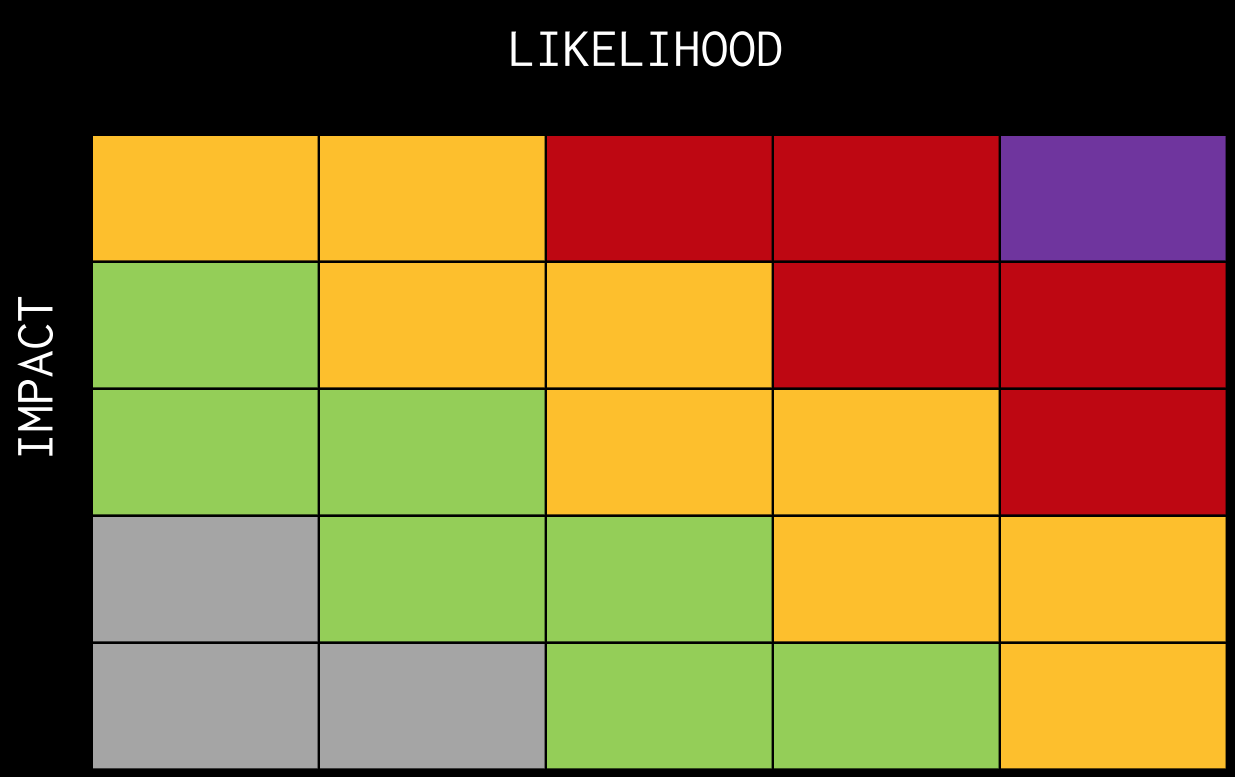
Remediation Plan:

Branch: 0ad8cad2bb661fbd53086d097d11228304d9b73e

OUT-OF-SCOPE:
Other smart contracts in the repository, external libraries and economical attacks.

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	0	2



EXECUTIVE OVERVIEW

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) MISSING CHECK	Informational (0.2)	SOLVED - 06/19/2023
(HAL-02) MISSING CONTRACT CHECK	Informational (0.0)	SOLVED - 06/19/2023

FINDINGS & TECH DETAILS

3.1 (HAL-01) MISSING CHECK - INFORMATIONAL (0.2)

Description:

The `applySingleTargetPermissions` function under the `core/permission/PermissionManager.sol` does not verify the `operation == PermissionLib.Operation.GrantWithCondition` case. Although the `SingleTargetPermission` struct does not support the `_condition` field, it is possible to set the operation to conditional and have `0` effect on the apply function call.

BVSS:

AO:S/AC:L/AX:L/C:N/I:L/A:N/D:N/Y:N/R:F/S:C (0.2)

Recommendation:

It is recommended to either add a revert statement if the operation is a `GrantWithCondition` or modify the `SingleTargetPermission.operation` type to use a different `enum` which contains only the `Grant` and `Revoke` states.

Remediation plan:

SOLVED: A condition is now checking for the stated and reverting accordingly with `GrantWithConditionNotSupported`.

3.2 (HAL-02) MISSING CONTRACT CHECK - INFORMATIONAL (0.0)

Description:

The _grantWithCondition under the core/permission/PermissionManager.sol contract, does not verify that the given _condition address is a valid contract neither it does verify the interface compatibility with supportsInterface. Although internally the _isGranted will catch any exception, it is advised to perform the check as soon as possible on the permission chain.

BVSS:

AO:S/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:P/S:C (0.0)

Recommendation:

It is recommended to add a check using the Address library to the isContract function.

Remediation plan:

SOLVED: The granting system has been split into two internal functions _grant and _grantWithCondition. The former does now allow ANY_ADDR and the latter does verify it against isPermissionRestrictedForAnyAddr. The isContract and supportsInterface checks for IPermissionCondition have been added.

MANUAL TESTING

4.1 /utils

UncheckedMath.sol:

Expected functionality and no issues found

Used in:
- plugins/utils/Addresslist.sol

Proxy.sol:

Standard ERC1967Proxy proxy creation.

Used in:
- plugins/counter-example/v2/CounterV2PluginSetup.sol
- plugins/counter-example/v1/CounterV1PluginSetup.sol
- plugins/governance/majority-voting/addresslist/AddresslistVotingSetup.sol
- plugins/governance/majority-voting/token/TokenVotingSetup.sol
- plugins/governance/multisig/MultisigSetup.sol
- framework/dao/DAOFactory.sol
- framework/plugin/repo/PluginRepoFactory.sol
- framework/plugin/setup/PluginSetup.sol
- plugins/token/MerkleMinter.sol

4.2 /utils/protocol

ProtocolVersion.sol:

It only contains a function returning the version of the protocol as an array of 3 values. It is stated on the contract that no storage should be added to this abstract, causing the storage layout to shift down.

Used in:

- framework/dao/DAOFactory.sol
- core/dao/DAO.sol

4.3 /core/utils

BitMap.sol:

Expected functionality and no issues found

Used in:
- core/dao/DAO.sol

CallbackHandler.sol:

Callback of 0x0000000 is allowed, referring to the fallback/receive function. Unregistering a callback is achieved by setting the magicNumber to 0.

Used in:
- core/dao/DAO.sol

auth.sol:

It exposes one function named _auth which does call the provided DAO address hasPermission function. If the hasPermission function does return false, the transaction will revert.

Used in:
- core/plugin/dao-authorizable/DaoAuthorizable.sol
- core/plugin/dao-authorizable/DaoAuthorizableUpgradeable.sol

4.4 /core/permission

PermissionLib.sol:

A library exposing several structs:

- Operation: It abstracts the operations that permission can have, Grant, Revoke and GrantWithCondition.
- SingleTargetPermission: It states what permission and operations a who user can have
- MultiTargetPermission: It states what permission and operations a who user can have on a where address. There is also a condition address that allows specifying a conditional contract.

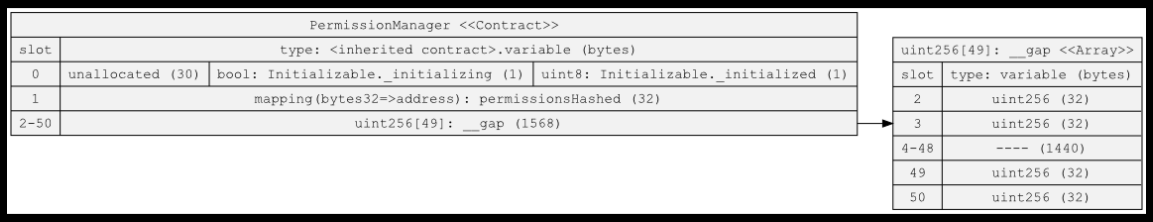
PermissionManager.sol:

Used to manage the permissions on any contract inheriting from it. It acts as an ACL for that contract.

- The ANY_ADDR variable which corresponds to the address 0xFFF..FFF is treated as the “any” address and does validate to any user.
- Since the contract allows permissions to be conditioned from another contract, the flag indicating that single permission is set corresponds to the address(2). The address(0) is used as the not set flag.
- In order to set permissions on the inheriting contract, the caller to grant and grantWithCondition must be granted the ROOT_PERMISSION_ID permission.
- The internal _isGranted will verify if a given packed where/who/permissionID is set on the internal permissionsHashed mapping. If the condition of the mapping is not address 0 (denied) or 2 (allowed) it will try to call the function isGranted on the stored address as an IPermissionCondition contract to verify the condition.

- The _auth does call the isGranted function by always using where=addres(this) and who=msg.sender.
- There are two apply functions which will iterate over a list of permissions and call the corresponding _grant or _grantWithCondition .
- The isPermissionRestrictedForAnyAddr does restrict all the permissions that ANY_ADDR is not allowed in either where or who. Contracts inheriting from this contract are expected to override this.
- The ROOT_PERMISSION_ID is not allowed to be granted to ANY_ADDR in neither where nor who. Only single addresses are allowed.
- ANY_ADDR can only be used if a _condition contract is specified. Granting to any address using an ALLOW_FLAG is not permitted by the system.

Storage layout:



PermissionManager <<Contract>>			
slot	type: <inherited contract>.variable (bytes)		
0	unallocated (30)	bool: Initializable._initializing (1)	uint8: Initializable._initialized (1)
1	mapping(bytes32=>address): permissionsHashed (32)		
2-50	uint256[49]: __gap (1568)		

uint256[49]: __gap <<Array>>	
slot	type: variable (bytes)
2	uint256 (32)
3	uint256 (32)
4-48	---- (1440)
49	uint256 (32)
50	uint256 (32)

Used in:

- core/dao/DAO.sol (Initialized correctly)
- framework/plugin/repo/PluginRepo.sol (Initialized correctly)

4.5 /core/dao

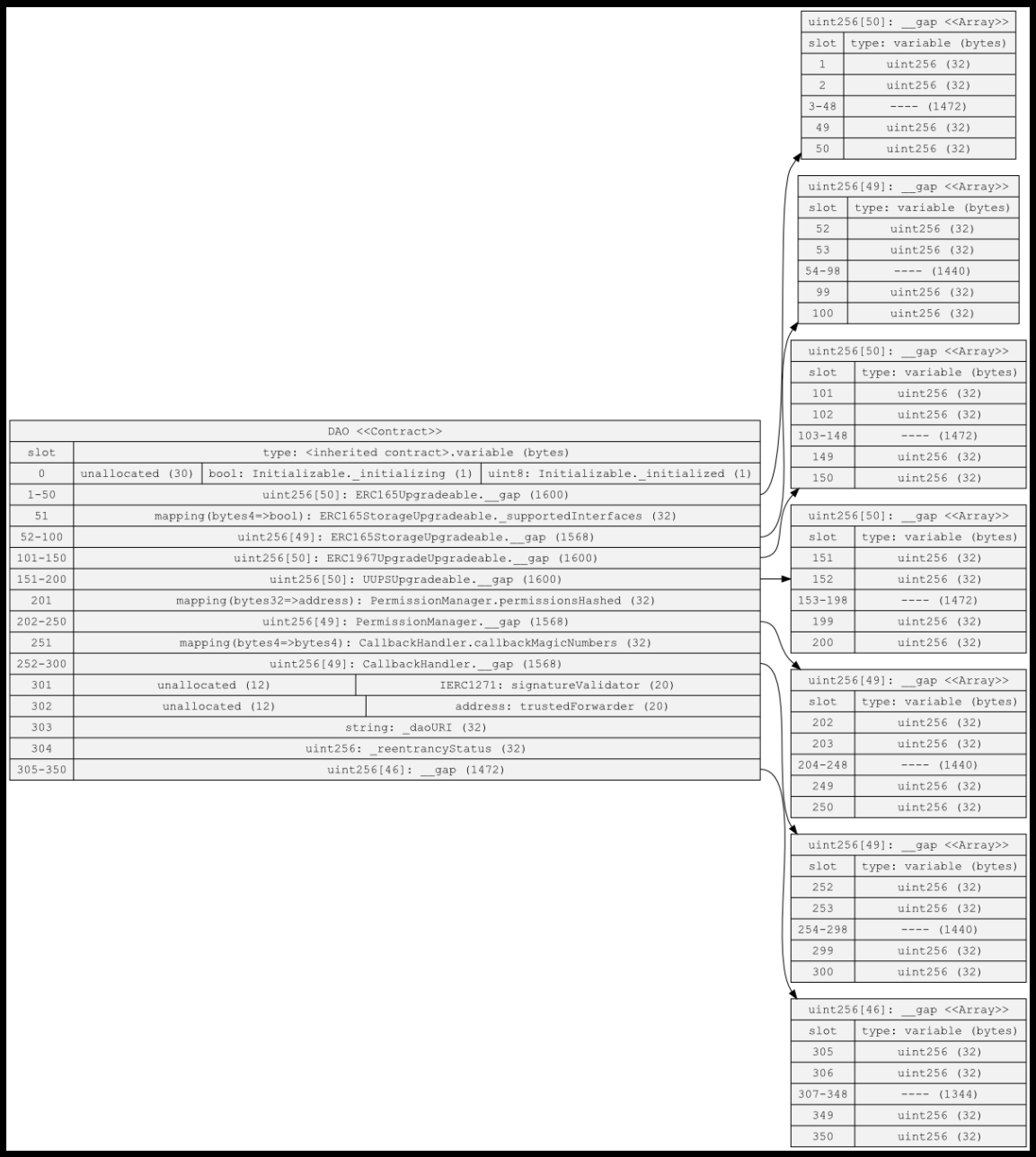
Dao.sol:

- Does delegate the signature verification to an external signatureValidator contract. The initial value with be address(0), which means that all given signatures will be treated as unverified/invalid. **Probably it will be a good idea to add it to the initialize function**
- The initializer does set all properties of the DAO using the _set functions, registers all the interfaces and callbacks for onERC functions and initializes the permission manager with the _initialOwner.
- Although the onERCxxxReceived functions are dynamically registered and will not revert, there is no pre-defined way to interact and transfer those received tokens.
- All permissions used on the DAO are registered under the isPermissionRestrictedForAnyAddr overriding function. This will deny ANY_ADDR on the permission system for the DAO and always have granularity on who has what permission.

Storage layout:

Correctly giving gaps between inherited contracts to solve storage shift down if adding new variables.

MANUAL TESTING



4.6 /plugin/

Plugin.sol:

Exposes a generic abstract plugin of type PluginType.Constructable and DaoAuthorizable.

This contract is not being used.

PluginCloneable.sol:

Exposes a generic abstract plugin of type PluginType.Cloneable and DaoAuthorizable but using Proxy compatible initializers.

Used in:

- plugins/governance/admin/Admin.sol

PluginUUPSUpgradeable:

Exposes a generic abstract plugin of type PluginType.UUPS and DaoAuthorizable but using Proxy compatible initializers. Furthermore, it does override the _authorizeUpgrade internal function with authorization for callers with permission UPGRADE_PLUGIN_PERMISSION_ID.

Used in:

- plugins/counter-example/MultiplyHelper.sol
- plugins/counter-example/v2/CounterV2.sol
- plugins/token/MerkleDistributor.sol
- plugins/counter-example/v1/CounterV1.sol
- plugins/token/MerkleMinter.sol

4.7 /plugin/dao-authorizable

DaoAuthorizable.sol:

It does use the auth from utils to call the immutable stored DAO hasPermission for the current address(this)(where) and using msg.sender as the who. It is only used on the none-upgradeable and none-uups version of a plugin and takes no storage slots.

Used in:

- core/plugin/Plugin.sol

DaoAuthorizableUpgradeable.sol:

Same as DaoAuthorizable.sol but this time it uses initializers instead of constructors, which does require a private dao_ variable and a corresponding __gap.

Used in:

- framework/utils/InterfaceBasedRegistry.sol (Initialized under __InterfaceBasedRegistry_init)
- framework/utils/ens/ENSSubdomainRegistrar.sol (Initialized under initialize)
- core/plugin/PluginCloneable.sol (Initialized under __PluginCloneable_init)

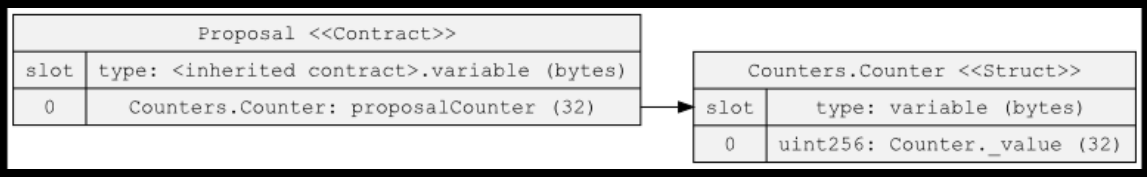
4.8 /core/plugin/proposal

Proposal.sol:

Abstract contract that implements the functionality required to create generic proposals on a DAO.

It does expose _createProposal and _executeProposal to be used in other plugins as a way to generate new proposal ids and to call the DAO execute method.

Storage Layout:



Proposal <<Contract>>	
slot	type: <inherited contract>.variable (bytes)
0	Counters.Counter: proposalCounter (32)

Counters.Counter <<Struct>>	
slot	type: variable (bytes)
0	uint256: Counter._value (32)

ProposalUpgradeable.sol:

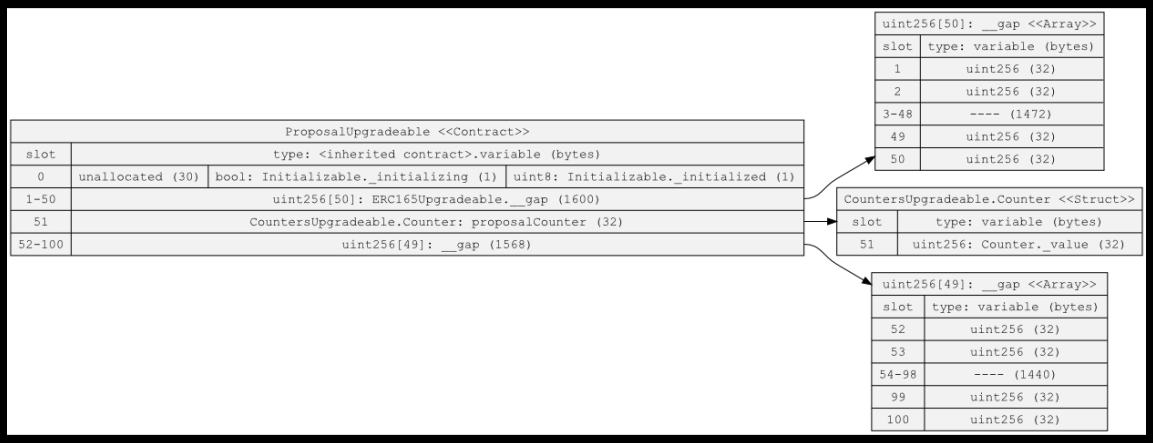
Same as the normal Proposal.so contract, but using CountersUpgradeable for the id counter instead. It has the correct gaps for later storage expansion.

Used in:

- plugins/governance/admin/Admin.sol

Storage Layout:

MANUAL TESTING



4.9 /framework/utils/

InterfaceBasedRegistry.sol:

- Upgradeable contract used to keep track if an address is registered and tracked. It does so via the _register function that verifies that the entry is not present.
- The contract is upgradeable via UUPS and requires the UPGRADE_REGISTRY_PERMISSION_ID permission. **However, this permission is not automatically given to any other scoped contract**
- It does contain gaps in case the storage is extended in the future.

Used in:

- framework/plugin/repo/PluginRepoRegistry.sol
- framework/dao/DAORegistry.sol

RegistryUtils.sol:

It does only expose a function named isSubdomainValid. This function will verify that a string does contain only printable characters, numbers and -. **The function will report as valid subdomain an empty string**. However, all contracts using this function do check the length before calling it.

Used in:

- framework/dao/DAORegistry.sol
- framework/plugin/repo/PluginRepoRegistry.sol

TokenFactory.sol:

It does allow creating a new governance ERC20 token and a MerkleMinter which has permissions to mint them. The merkleMint function under the MerkleMinter can be called to initialize a new distribution campaign using a Merkle tree.

- The setupBases function does create a new MerkleDistributor, a new GovernanceERC20 and GovernanceWrappedERC20 used to clone it during the token creation (The clonable token does initialize itself on the constructor, preventing from re-initalization).
- The createToken function does take a config struct containing token name, symbol and address for the token generation. If the address is not specified (address(0)) a new ERC20 governance token is deployed, otherwise a wrapped governance is deployed for the provided ERC20 token, turning the parent wrapper into a governance token. The wrap does add some functions such as depositFor, withdrawTo and auto delegation to itself of the voting power under _afterTokenTransfer.
- **It does allow wrapping the current governance ERC20 token.**
- It does correctly set the permissions required for MerkleMinter to operate over the token and sets the DAO permissions to directly allow minting of the newly created token.
- **However, the GovernanceWrappedERC20 token does not have any functionality or can be managed though the DAO instead of the GovernanceERC20.**

ens/ENSSubdomainRegistrar.sol:

It does allow registering subdomains into an ENS ENSSubdomainRegistrar via a DaoAuthorizableUpgradeable. The exposed registerSubnode function does register the given label as a subdomain.

Used in:

- framework/plugin/repo/PluginRepoRegistry.sol
- framework/dao/DAORegistry.sol

4.10 /framework/dao

DAOFactory.sol:

It does allow creating a new DAO with all the permissions required to manage it. Initially, the ROOT_PERMISSION_ID permission is given to the factory itself and revoked later after transferring it to the original initial owner.

- During creation, the factory will give pluginSetupProcessor the ROOT_PERMISSION_ID to be able to install plugin permissions. It does also grant APPLY_INSTALLATION_PERMISSION_ID permissions required to apply the plugins. The latter wouldn't be required if the application was called from the DAO itself.
- Each of the plugin is prepared and applied.
- **The DAO is given all the permissions that are listed under the isPermissionRestrictedForAnyAddr but EXECUTE_PERMISSION_ID.** This is expected as the EXECUTE_PERMISSION_ID should be given to the plugin implementing the proposal creation/execution to perform executions as the DAO.

DAORegistry.sol:

The managing DAO does require the REGISTER_DAO_PERMISSION_ID permissions.

- **It does allow registering a DAO without a subdomain** as the length is checked before calling the isSubdomainValid and the registerSubnode under the managed ENSSubdomainRegistrar.

4.11 /plugins/token

MerkleDistributor.sol:

It is using the standard MerkleDistributor.sol from Uniswap modified to include ERC165 and UUPS upgradability support.

Reference: https://github.com/Uniswap/merkle-distributor/blob/master/contracts/Merk

Used in:

- framework/utils/TokenFactory.sol
- plugins/token/MerkleDistributor.sol
- plugins/token/MerkleMinter.sol

MerkleMinter.sol:

UUPS plugin that does expose several functions:
- changeDistributorBase: Used to change the distributor template address. The caller does require CHANGE_DISTRIBUTOR_PERMISSION_ID. **This permission is by default not given to anyone, neither the DAO.**
- merkleMint: It does allow creating a new distributor proxy from the template base. This allows setting a new token or an entire new Merkle root for a token distribution. The Merkle minter does require mint permissions to the TOKEN and the caller requires MERKLE_MINT_PERMISSION_ID permissions, which by default the TokenFactory does grant to the managing DAO.

Used in:

- framework/utils/TokenFactory.sol (Used as a template and cloned, correctly initialized)

4.12 /framework/dao

DAORegistry.sol:

4.13 /framework/plugin/repo

PluginRepo.sol:

It does allow creating and managing repos by using release versions. The contract does have a permission manager itself to protect against anyone from creating new releases (MAINTAINER_PERMISSION_ID) or upgrading the contract (UPGRADE_REPO_PERMISSION_ID).

- A release is a number from 1 to 255, both included. This is similar to how OZ does use the reinitializer numbering. The top number indicates in both cases that no more versions do exist.
- From the initial look on createVersion it looks like it is possible to use the same _release as the latestRelease. This is possible to add new buildsPerRelease. It does verify that the same _pluginSetup address is not used on previous release.
- **The updateReleaseMetadata function does allow emitting the ReleaseMetadataUpdated event for a release version that is not yet tracked into the system.**

Used in:

- framework/dao/DAOFactory.sol
- framework/plugin/repo/PluginRepoRegistry.sol
- framework/plugin/repo/PluginRepo.sol
- framework/plugin/repo/PluginRepoFactory.sol
- framework/plugin/setup/PluginSetupProcessorHelpers.sol
- framework/plugin/setup/PluginSetupProcessor.sol

PluginRepoRegistry.sol:

An implementation of a InterfaceBasedRegistry contract that does allow tracking plugin addresses and registering them using ENS subdomains on the ENSSubdomainRegistrar.

- It does correctly implement extra gaps for the upgradability.

Used in:

- framework/plugin/repo/PluginRepoFactory.sol
- framework/plugin/setup/PluginSetupProcessor.sol

PluginRepoFactory.sol:

It does allow creating PluginRepo contracts and will register them under the public PluginRepoRegistry. Anyone can create a new repo connected to a plugin and have it deploy new releases.

- The createPluginRepo function does internally call _createPluginRepo which does instantiate the PluginRepo giving the _initialOwner permissions ROOT_PERMISSION_ID, MAINTAINER_PERMISSION_ID and UPGRADE_REPO_PERMISSION_ID.
- The createPluginRepoWithFirstVersion does initialize the PluginRepo with the owner to begin the PluginRepoFactory itself. This allows the contract to call createVersion on the newly created pluginRepo. The function then revokes all 3 permissions given by the creation of the PluginRepo and adds those to the original _maintainer from the caller.

4.14 /framework/plugin/setup

PluginSetup.sol:

It does define the bases for developers to use to set up a plugin.

Used in:

- plugins/placeholder-version/PlaceholderSetup.sol
- plugins/governance/multisig/MultisigSetup.sol
- plugins/governance/majority-voting/addresslist/AddresslistVotingSetup.sol
- plugins/governance/admin/AdminSetup.sol
- plugins/governance/majority-voting/token/TokenVotingSetup.sol
- plugins/counter-example/v2/CounterV2PluginSetup.sol
- plugins/counter-example/v1/CounterV1PluginSetup.sol
- framework/plugin/repo/PluginRepo.sol
- framework/plugin/setup/PluginSetupProcessorHelpers.sol
- framework/plugin/setup/PluginSetupProcessor.sol

PluginSetupProcessor.sol:

This single contract allows performing CRUD operations on DAO plugins.

- The prepareInstallation, does verify that the given pluginSetupRepo is registered on the repoRegistry so no unregistered plugins can be used. It also verifies that the versionTag is present.
- The caller on all the apply methods, which can only be the given DAO parameter address, do require the corresponding permission for that action being APPLY_INSTALLATION_PERMISSION_ID, APPLY_UPDATE_PERMISSION_ID or APPLY_UNINSTALLATION_PERMISSION_ID.
- The applyInstallation does take directly the helpersHash instead of using the same preparedSetupData there should be some frontend preprocess on that hash.

- `prepareUpdate` does verify that the currently installed plugin is indeed the one provided in the arguments. This is done by verifying that `currentAppliedSetupId` for the DAO `setupPayload.plugin` is equal to the `appliedSetupId` which is computed from the current tag of the function parameters and `currentHelpersHash`.
- The update can be applied only if the caller has `UPGRADE_PLUGIN_PERMISSION_ID`. This means that a proposal should be granted this permission to `PluginSetupProcessor` before actually applying the update.

Used in:

- `framework/dao/DAOFactory.sol`

4.15 /plugins/governance/admin

Admin.sol:

- It does contain a single function executeProposal which allows creating and executing a proposal without voting.
- Membership is defined as having the permission EXECUTE_PROPOSAL_PERMISSION_ID on the DAO.
- **It does not define a gap for future storage variables. However, currently no storage is being used.**

Used in:

- plugins/governance/admin/AdminSetup.sol

AdminSetup.sol:

The setup plugin for the Admin.sol contract.

- It does correctly give both, EXECUTE_PROPOSAL_PERMISSION_ID and EXECUTE_PERMISSION_ID permissions.
- prepareUninstallation does revoke only the EXECUTE_PERMISSION_ID for the plugin. However, all members will still hold the EXECUTE_PROPOSAL_PERMISSION_ID to the old plugin address. This shouldn't be a problem if the plugin is updated later on, as a new where address will be granted instead.

4.16 /plugins/governance/majority-voting

MajorityVotingBase.sol:

Inheriting contracts should implement _canExecute and _canVote.

Used in:

- plugins/governance/majority-voting/addresslist/AddresslistVotingSetup.sol
- plugins/governance/majority-voting/addresslist/AddresslistVoting.sol
- plugins/governance/majority-voting/token/TokenVotingSetup.sol
- plugins/governance/majority-voting/token/TokenVoting.sol

AddresslistVoting.sol:

It does use the Addresslist to track members of the DAO. Anyone on the DAO is allowed to create proposals.

AddresslistVotingSetup.sol:

It correctly gives all permissions required to update the voting settings and the addresses. Furthermore, it also gives the EXECUTE_PERMISSION_ID permission to allow DAO execution.

TokenVoting.sol:

Same as AddresslistVoting but using voting power instead of membership.

- The voting power is calculated based on the proposal snapshot block. This means that all balances for voting will be using that block as a reference. During voting, the getPastVotes is used to fetch the voter power.

TokenVotingSetup.sol:

It does allow specifying an already existing ERC20 for the voting phase. If the token cannot be used to vote, a wrapped version will be created.

- All permissions are correctly given. If a new token is created, the MINT_PERMISSION_ID is also given.

Multisig.sol:

Works really similar to AddresslistVoting but instead of voting, there is an approve function.

- The _execute is completely override to verify that the amount of approvals is bigger than minApprovals.

MultisigSetup.sol:

Same as the AddresslistVotingSetup permissions.

4.17 /token/ERC20/governance

GovernanceERC20.sol:

Standard ERC20 with minting, voting and DAO managed. Minting does require the caller to have MINT_PERMISSION_ID on the DAO.

- It does automatically turn delegation of voting power to itself for each transfer/mint if no checkpoints (aka first time using voting tokens) and no delegates have been already set.

Used in:

- framework/utils/TokenFactory.sol
- plugins/governance/majority-voting/token/build-metadata.json
- plugins/governance/majority-voting/token/TokenVotingSetup.sol

GovernanceWrappedERC20.sol:

Only the depositFor and withdrawTo functions can be used, which do transfer the underlying token to the wrapped version, causing the delegation to activate (for voting purposes) and automatically minting wrapped tokens.

Used in:

- plugins/governance/majority-voting/token/TokenVotingSetup.sol
- ‘framework/utils/TokenFactory.sol

4.18 /plugins/utils

Addresslist.sol:

It does use the CheckpointsUpgradeable to check whether an address is whitelisted or not currently and in past blocks. It does set the checkpoint to 1 or 0 accordingly.

Used in:

- plugins/governance/majority-voting/addresslist/AddresslistVoting.sol
- plugins/governance/multisig/Multisig.sol

Ratio.sol:

Does only expose a pure function named _applyRatioCeiled. This function does convert a given value with a given ratio (ratio always being less than 10**6) to the 0, 10**6 range. If max ratio is specified, the value returned is the same as the input. However, if less than 10**6 is given as the ratio, the value is treated as the percentage of the big 0, 10**6 range. For example, 10**4 would correspond to the 1 percent of the big range, so the returned value will be a 1% of the input value plus 1 if the remainder of the division is none zero. This means that 1.01 will be ceiled to 2.

4.19 Deploy scripts

- utils/abi.ts:
  - The getMergedABI function does not check if the duplicated events have the same number of arguments, it relies only on the event name.
- deploy/helpers.ts:
  - Valid DAO_PERMISSIONS.

    - uploadToIPFS has the X-API-KEY hardcoded, **consider using environment variables.**

- deploy/new/00_managing-dao/00_managing-dao.ts: It does deploy the managing DAO with default parameters.
    - **trustedForwarder is set to zero, same goes with metadata and daoURI**.
    - The deployment happens using a ERC1967Proxy on the DAO.sol contract.
        - This deployment is granting ROOT permissions on the deployer .
- deploy/new/00_managing-dao/01_managing-dao-permissions.ts: Does grant EXECUTE_PERMISSION permissions on the managingDAOAddress to the deployer.
- deploy/new/00_managing-dao/02_managing-dao_conclude.ts: Does store the implementation and proxy address under aragonToVerifyContracts.
- deploy/new/00_managing-dao/20_set-dao-permission.ts: Does grant to the managingDAO contract permissions to itself. The permissions are the same present on the DAOFactory.sol contract under the _setDAOPermissions function.
- deploy/new/00_managing-dao/99_verify_step.ts: Does verify that the deployer has root permissions and the managingDAO does have all DAO_PERMISSIONS permissions.
- deploy/new/10_framework/00_ens_registry.ts: Does deploy the ENSRegistry and PublicResolver. It will register the given array if the subdomain is not already registered.
- deploy/new/10_framework/01_ens_subdomains.ts: It will verify if the dao and plugin ENS nodes are present and register them otherwise. It will transfer the domain to the managing DAO.
- deploy/new/10_framework/02_ens_subdomain_registrars.ts: Does deploy the proxied ENSSubdomainRegistrar.
- deploy/new/10_framework/09_ens_conclude.ts: Does push the ens contracts to the aragonToVerifyContracts.
- deploy/new/10_framework/10_dao-registry.ts: Does deploy the proxied DAORegistry.
- deploy/new/10_framework/11_dao-registry_conclude.ts: Does push the registry to the aragonToVerifyContracts array.

- deploy/new/10_framework/20_plugin-repo-registry.ts: Does deploy the proxied PluginRepoRegistry.
- deploy/new/10_framework/21_plugin-repo-registry_conclude.ts: Does push the registry to the aragonToVerifyContracts array.
- deploy/new/10_framework/30_repo-factory.ts: Does deploy the proxied PluginRepoFactory.
- deploy/new/10_framework/31_repo-factory_conclude.ts: Does push the registry to the aragonToVerifyContracts array.
- deploy/new/10_framework/40_plugin_setup_processor.ts: Does deploy the proxied PluginSetupProcessor.
- deploy/new/10_framework/41_plugin_setup_processor_conclude.ts: Does push the registry to the aragonToVerifyContracts array.
- deploy/new/10_framework/50_dao-factory.ts: Does deploy the proxied DAOFactory.
- deploy/new/10_framework/51_dao-factory_conclude.ts: Does push the registry to the aragonToVerifyContracts array.
- deploy/new/10_framework/99_verifiy_step: Does verify that the previous single step contracts do contain a valid address. It also verifies that the ENS contract does contain the correct relationship address.
- deploy/new/20_permissions/00_ens-permissions: Does grant all required permissions to manage and work with ENS.
  - Grant REGISTER_ENS_SUBDOMAIN_PERMISSION of DAO_ENSSubdomainRegistrar to DAORegistry.
  - Grant REGISTER_ENS_SUBDOMAIN_PERMISSION of Plugin_ENSSubdomainRegistrar to PluginRepoRegistry.
  - Grant UPGRADE_REGISTRAR_PERMISSION of DAO_ENSSubdomainRegistrar to ManagingDAO.
  - Grant UPGRADE_REGISTRAR_PERMISSION of Plugin_ENSSubdomainRegistrar to ManagingDAO.
- deploy/new/20_permissions/10_dao-registry-permissions: Does grant all required permissions to manage and work with the DAO registry:
  - Grant REGISTER_DAO_PERMISSION of DAORegistry to DAOFactory.
  - Grant UPGRADE_REGISTRY_PERMISSION of DAORegistry to ManagingDAO.
- deploy/new/20_permissions/20_plugin-registry-permissions: Does

grant all required permissions to manage and work with the PluginRepoRegistry:

  - Grant REGISTER_PLUGIN_REPO_PERMISSION of PluginRepoRegistry to DAOFactory.
  - Grant UPGRADE_REGISTRY_PERMISSION of PluginRepoRegistry to ManagingDAO.

- deploy/new/20_permissions/99_verifiy: Does verify that the previous given permissions are set.
- deploy/new/30_plugins/00_plugin-setups/00_addresslist_voting_setup : Does deploy AddresslistVotingSetup.
- deploy/new/30_plugins/00_plugin-setups/01_addresslist_voting_setup_conclude : It does add the AddressListVotingSetup to aragonToVerifyContracts.
- deploy/new/30_plugins/00_plugin-setups/10_token_voting_setup: Does deploy TokenVotingSetup.
- deploy/new/30_plugins/00_plugin-setups/11_token_voting_setup_conclude : Does add the deployed TokenVotingSetup to aragonToVerifyContracts . It also adds the underlying governanceERC20Base and governanceWrappedERC20Base tokens.
- deploy/new/30_plugins/00_plugin-setups/20_admin_setup: Does deploy AdminSetup.
- deploy/new/30_plugins/00_plugin-setups/21_admin_setup_conclude Does add AdminSetup to aragonToVerifyContracts.
- deploy/new/30_plugins/00_plugin-setups/30_multisig_setup: Does deploy MultisigSetup.
- deploy/new/30_plugins/00_plugin-setups/31_multisig_setup_conclude: Does add MultisigSetup to aragonToVerifyContracts.
- deploy/new/30_plugins/00_plugin-setups/40_placeholder_setup: Does deploy PlaceholderSetup.
- deploy/new/30_plugins/00_plugin-setups/41_placeholder_setup_conclude : Does add PlaceholderSetup to aragonToVerifyContracts.
- deploy/new/30_plugins/10_plugin-repos/00_create_address_list_voting_repo : It does add the AddressListVotingSetup plugin to the repo with version 1.1.
- deploy/new/30_plugins/10_plugin-repos/01_create_address_list_voting_repo_concl : Add contract for verification.
- deploy/new/30_plugins/10_plugin-repos/10_create_token_voting_repo: It does add the TokenVotingSetup plugin to the repo with version

1.1.

- deploy/new/30_plugins/10_plugin-repos/11_create_token_voting_repo_conclude : Add contract for verification.
- deploy/new/30_plugins/10_plugin-repos/20_create_admin_repo: It does add the AdminSetup plugin to the repo with version 1.1.
- deploy/new/30_plugins/10_plugin-repos/21_create_admin_repo_conclude : Add contract for verification.
- deploy/new/30_plugins/10_plugin-repos/30_create_multisig_repo: It does add the MultisigSetup to the repo with version 1.2. **This means that a placeholder will be created for version 1.1**.
- deploy/new/30_plugins/10_plugin-repos/31_create_multisig_repo_conclude : Add contract for verification.
- deploy/new/40_finalize-managing-dao/00_grant-permissions: It does grant several permissions to the deployer and PluginSetupProcessor:
  - Grant REGISTER_DAO_PERMISSION to Deployer.
  - Grant ROOT_PERMISSION to PluginSetupProcessor.
  - Grant APPLY_INSTALLATION_PERMISSION to Deployer.
    It does also grant all permissions required (ROOT_PERMISSION, MAINTAINER_PERMISSION, UPGRADE_REPO_PERMISSION) on the PluginRepo on each deployed plugin to update and create new releases.
  - deploy/new/40_finalize-managing-dao/20_register-managing-dao-on-dao-registry: It does register the managing DAO on the DAO registry.
  - deploy/new/40_finalize-managing-dao/30_install-multisig-on-managing-dao: It does apply and install the Multisig plugin to the managing DAO.
  - deploy/new/40_finalize-managing-dao/40_revoke-permissions: It does revoke several permissions used to apply and install the plugins. It also removes ROOT from PluginSetupProcessor and ROOT_PERMISSION, MAINTAINER_PERMISSION and UPGRADE_REPO_PERMISSION from the deployer.
- deploy/new/40_finalize-managing-dao/99_verify_step: It does verify that previous permissions are revoked.
- deploy/update/to_v1.3.0/01_DAOFactory: Does deploy a new DAOFactory by removing the old factory permissions. The action is appended to applyMultiTargetPermissions for proposal creation on the last step.

- deploy/update/to_v1.3.0/02_DAOFactory_conclude: It does push the new DAOFactory to aragonToVerifyContracts.
- deploy/update/to_v1.3.0/10_Multisig_Plugin: It does deploy a new MultisigSetup release. **There is a check to verify if the deployer has MAINTAINER_PERMISSION_ID permissions and take a shortcut, as seen on the Permissions' table that should never be the case.** If everything else fails, fallbacks to creating an managingDAOActions.
- deploy/update/to_v1.3.0/11_Multisig_Plugin_conclude: Does add contract for verification.
- deploy/update/to_v1.3.0/20_TokenVoting_Plugin: Same as 10_Multisig_Plugin deployment.
- deploy/update/to_v1.3.0/21_TokenVoting_Plugin_conclude: Same as 20_TokenVoting_Plugin.
- deploy/update/to_v1.3.0/30_AddresslistVoting_Plugin.ts: Same as 10_Multisig_Plugin deployment.
- deploy/update/to_v1.3.0/31_AddresslistVoting_Plugin_conclude.ts: Same as 20_TokenVoting_Plugin.
- verification/99_conclude/00_save-contract-addresses: Does store all deployment addresses.
- verification/99_conclude/01_managing_dao_proposal: Does create a proposal using all the managigDAOActions actions set during update.
- verification/99_conclude/10_verify-contracts: Does verify all the contracts pushed to aragonToVerifyContracts.

Permissions:

After checking all the scripts and the permissions that are manually granted and revoked, the following table does show in green all the permissions that are still alive on the system and in red all the permissions that have been revoked at some point.

	A	B	C	D	E
1	**Script**	**Permission**	**Where**	**Who**	**Action**
2	00_managing-dao.ts	ROOT_PERMISSION	managingDAO	Deployer	Grant
3	00_managing-dao/01_managing-dao-permissions.ts	EXECUTE_PERMISSION	managingDAO	Deployer	Grant
4	00_managing-dao/20_set-dao-permission.ts	ROOT_PERMISSION	managingDAO	managingDAO	Grant
5	00_managing-dao/20_set-dao-permission.ts	UPGRADE_DAO_PERMISSION	managingDAO	managingDAO	Grant
6	00_managing-dao/20_set-dao-permission.ts	SET_SIGNATURE_VALIDATOR_PERMISSION	managingDAO	managingDAO	Grant
7	00_managing-dao/20_set-dao-permission.ts	SET_TRUSTED_FORWARDER_PERMISSION	managingDAO	managingDAO	Grant
8	00_managing-dao/20_set-dao-permission.ts	SET_METADATA_PERMISSION	managingDAO	managingDAO	Grant
9	00_managing-dao/20_set-dao-permission.ts	REGISTER_STANDARD_CALLBACK_PERMISSION	managingDAO	managingDAO	Grant
10	20_permissions/00_ens-permissions.ts	REGISTER_ENS_SUBDOMAIN_PERMISSION	DAO_ENSSubdomainRegistrar	DAORegistry	Grant
11	20_permissions/00_ens-permissions.ts	REGISTER_ENS_SUBDOMAIN_PERMISSION	Plugin_ENSSubdomainRegistrar	PluginRepoRegistry	Grant
12	20_permissions/00_ens-permissions.ts	UPGRADE_REGISTRAR_PERMISSION	DAO_ENSSubdomainRegistrar	ManagingDAO	Grant
13	20_permissions/00_ens-permissions.ts	UPGRADE_REGISTRAR_PERMISSION	Plugin_ENSSubdomainRegistrar	ManagingDAO	Grant
14	20_permissions/10_dao-registry-permissions.ts	REGISTER_DAO_PERMISSION	DAORegistry	DAOFactory	Grant
15	20_permissions/10_dao-registry-permissions.ts	UPGRADE_REGISTRY_PERMISSION	DAORegistry	ManagingDAO	Grant
16	20_permissions/20_plugin-registrty-permissions.ts	REGISTER_PLUGIN_REPO_PERMISSION	PluginRepoRegistry	PluginRepoFactory	Grant
17	20_permissions/20_plugin-registrty-permissions.ts	UPGRADE_REGISTRY_PERMISSION	PluginRepoRegistry	ManagingDAO	Grant
18	40_finalize-managing-dao/00_grant-permissions.ts	REGISTER_DAO_PERMISSION	DAORegistry	Deployer	Grant
19	40_finalize-managing-dao/00_grant-permissions.ts	ROOT_PERMISSION	managingDAO	PluginSetupProcessor	Grant
20	40_finalize-managing-dao/00_grant-permissions.ts	APPLY_INSTALLATION_PERMISSION	PluginSetupProcessor	Deployer	Grant
21	40_finalize-managing-dao/00_grant-permissions.ts	SET_METADATA_PERMISSION	managingDAO	Deployer	Grant
22	40_finalize-managing-dao/00_grant-permissions.ts	ROOT_PERMISSION	EACH PLUGIN	ManagingDAO	Grant
23	40_finalize-managing-dao/00_grant-permissions.ts	MAINTAINER_PERMISSION	EACH PLUGIN	ManagingDAO	Grant
24	40_finalize-managing-dao/00_grant-permissions.ts	UPGRADE_REPO_PERMISSION	EACH PLUGIN	ManagingDAO	Grant
25	40_finalize-managing-dao/40_revoke-permissions.ts	REGISTER_DAO_PERMISSION	DAORegistry	Deployer	Revoke
26	40_finalize-managing-dao/40_revoke-permissions.ts	ROOT_PERMISSION	managingDAO	PluginSetupProcessor	Revoke
27	40_finalize-managing-dao/40_revoke-permissions.ts	APPLY_INSTALLATION_PERMISSION	PluginSetupProcessor	Deployer	Revoke
28	40_finalize-managing-dao/40_revoke-permissions.ts	ROOT_PERMISSION	managingDAO	Deployer	Revoke
29	40_finalize-managing-dao/40_revoke-permissions.ts	SET_METADATA_PERMISSION	managingDAO	Deployer	Revoke
30	40_finalize-managing-dao/40_revoke-permissions.ts	EXECUTE_PERMISSION	managingDAO	Deployer	Revoke
31	40_finalize-managing-dao/40_revoke-permissions.ts	ROOT_PERMISSION	EACH PLUGIN	Deployer	Revoke
32	40_finalize-managing-dao/40_revoke-permissions.ts	MAINTAINER_PERMISSION	EACH PLUGIN	Deployer	Revoke
33	40_finalize-managing-dao/40_revoke-permissions.ts	UPGRADE_REPO_PERMISSION	EACH PLUGIN	Deployer	Revoke
34	to_v1.3.0/01_DAOFactory.ts	REGISTER_DAO_PERMISSION	DAORegistry	DAOFactory	Revoke
35	to_v1.3.0/01_DAOFactory.ts	REGISTER_DAO_PERMISSION	DAORegistry	DAOFactory 1.3.0	Grant
36					

Figure 1: Permissions still present on the system after deployment

AUTOMATED TESTING

5.1 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance the coverage of certain areas of the smart contract in scope. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified the smart contract in the repository and was able to compile it correctly into its ABI and binary format, Slither was run against the contract. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire code-base.

Several tools were executed, including Mythx and Slither. All the reported issues were verified and, if applicable, reported in previous sections. Slither reported several issues regarding uninitialized variables, which were false positives as inheritance was in place. Furthermore, an issue regarding the possibility to self-destruct the DAO contract due to not blocking the initializer was also a false positive as upgradeTo can only be called through a delegatecall.

THANK YOU FOR CHOOSING

// HALBORN