HALBORN

Aragon - aragonOS

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **February 7th, 2023 - February 24th, 2023**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY 4
CONTACTS 4
1 EXECUTIVE OVERVIEW 5
1.1 INTRODUCTION 6
1.2 AUDIT SUMMARY 6
1.3 TEST APPROACH & METHODOLOGY 6
RISK METHODOLOGY 7
1.4 SCOPE 9
2 ASSESSMENT SUMMARY & FINDINGS OVERVIEW 12
3 FINDINGS & TECH DETAILS 13
3.1 (HAL-01) SELF PERMISSIONS DO CASCADE TO EXTERNAL CONTRACTS - MEDIUM 15
Description 15
Code Location 15
Risk Level 16
Recommendation 16
Remediation Plan 16
3.2 (HAL-02) UNTRUSTED PLUGIN USAGE CAN CAUSE DOS - MEDIUM 17
Description 17
Risk Level 17
Recommendation 17
Remediation Plan 17
3.3 (HAL-03) MERKLE ROOT RE-USAGE - LOW 18
Description 18

Risk Level 18
Recommendation 18
Remediation Plan 18
3.4 (HAL-04) EMPTY SUBDOMAIN ALLOWED - LOW 19
Description 19
Risk Level 19
Recommendation 19
Remediation Plan 19
3.5 (HAL-05) MISSING CALLBACK SELECTOR ZERO CASE - INFORMATIONAL 20
Description 20
Risk Level 20
Recommendation 20
Remediation Plan 20
3.6 (HAL-06) UNUSED CODE - INFORMATIONAL 21
Description 21
Risk Level 21
Recommendation 21
Remediation Plan 21
4 MANUAL TESTING 22
4.1 DAO 23
4.2 PermissionManager 23
4.3 DAOFactory 23
4.4 InterfaceBasedRegistry 24

4.5 PluginSetupProcessor 24
4.6 ENSSubdomainRegistrar 24
4.7 AddresslistVoting 25
4.8 TokenVoting 25
5 AUTOMATED TESTING 25
5.1 STATIC ANALYSIS REPORT 27
Description 27

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	02/24/2023	Ferran Celades
0.2	Draft Review	02/24/2023	Gabi Urrutia
1.0	Remediation Plan	02/27/2023	Ferran Celades
1.1	Remediation Plan Review	02/27/2023	Gabi Urrutia

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 audit on their smart contracts beginning on February 7th, 2023 and ending on February 24th, 2023. The security assessment was scoped to the smart contracts provided to the Halborn team.

1.2 AUDIT SUMMARY

The team at Halborn was provided four weeks for the engagement and assigned a full-time security engineer to audit 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 audit 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 mostly 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 audit. 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 audit:

- 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.

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL

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/membership/IMembershipContract.sol
- core/plugin/proposal/Proposal.sol
- core/plugin/proposal/ProposalBase.sol
- core/plugin/proposal/ProposalUpgradeable.sol
- core/utils/BitMap.sol
- core/utils/CallbackHandler.sol
- core/utils/auth.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
- framework/utils/InterfaceBasedRegistry.sol
- framework/utils/RegistryUtils.sol
- framework/utils/TokenFactory.sol

- framework/utils/ens/ENSMigration.sol
- framework/utils/ens/ENSSubdomainRegistrar.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/Multisig.sol
- plugins/governance/multisig/MultisigSetup.sol
- plugins/token/IMerkleDistributor.sol
- plugins/token/IMerkleMinter.sol
- plugins/token/MerkleDistributor.sol
- plugins/token/MerkleMinter.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

Branch: 85f8b31e1b6dd012542cf3430b6985030ab3b44b
Pull request: 257
Pull request: 266

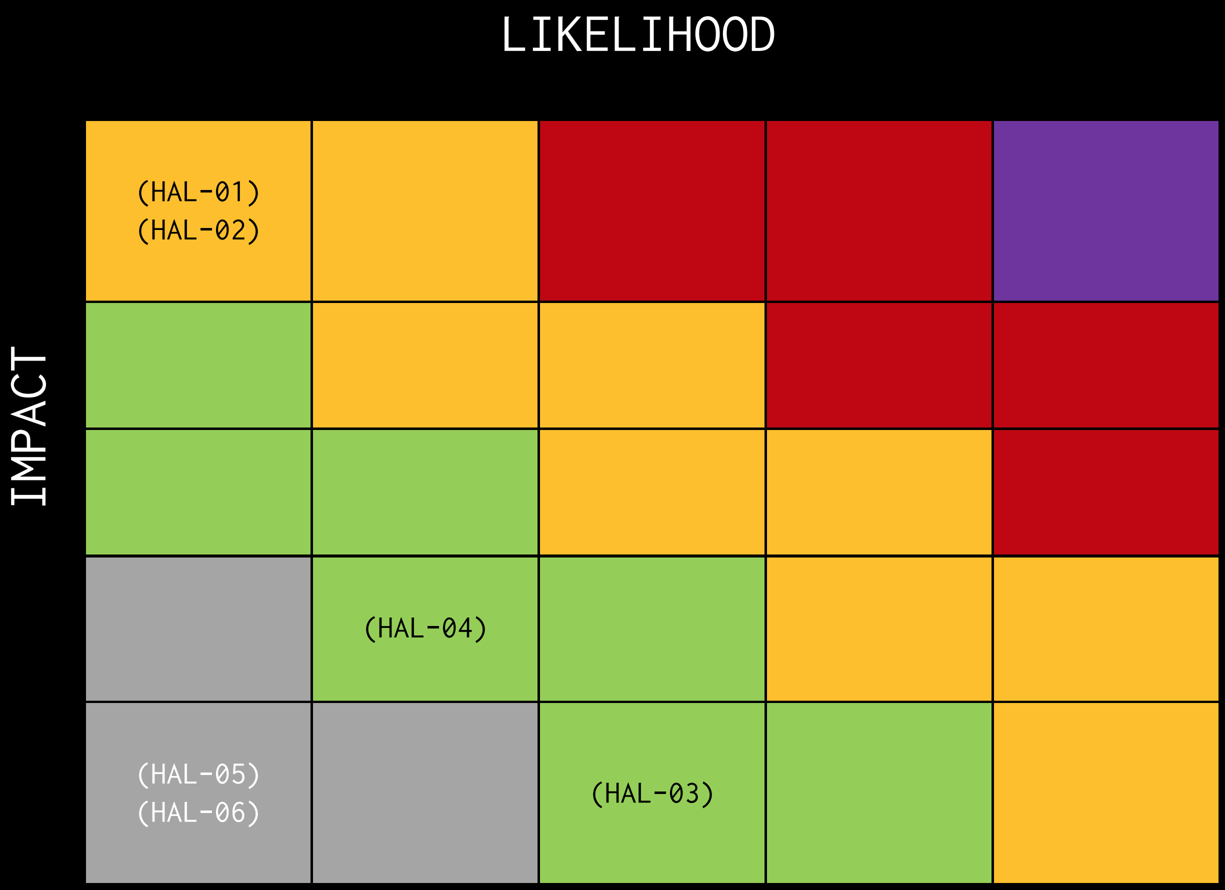
Remediation Plan:

Branch: cb0621dc5185a73240a6ca33fccc7698f059fdf5

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	2	2	2



SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
SELF PERMISSIONS DO CASCADE TO EXTERNAL CONTRACTS	Medium	SOLVED - 02/27/2023
UNTRUSTED PLUGIN USAGE CAN CAUSE DOS	Medium	SOLVED - 02/27/2023
MERKLE ROOT RE-USAGE	Low	NOT APPLICABLE
EMPTY SUBDOMAIN ALLOWED	Low	SOLVED - 02/27/2023
MISSING CALLBACK SELECTOR ZERO CASE	Informational	ACKNOWLEDGED
UNUSED CODE	Informational	SOLVED - 02/27/2023

FINDINGS & TECH DETAILS

3.1 (HAL-01) SELF PERMISSIONS DO CASCADE TO EXTERNAL CONTRACTS - MEDIUM

Description:

The _auth function on the PermissionManager is being used to verify if a given permission (permission id) is set to a contract (where) from a specific origin (from). The check does verify in two different scenarios:

- The where contract does have the permission from the current sender.
- The current contract (address(this)) does hold the permission from the current sender.

This means that if a given permission (PERMx) is set to the self contract, any call to any contract (anywhere) that may use the permission (PERMX) will succeed due to the second check. It has been seen that the entire code base is always using a where of address(this) which means that the where check is redundant and unused and adds a possible critical scenario described below.

Code Location:

Listing 1: src/core/permission/PermissionManager.sol (Lines 336,337)

```
334 function _auth(address _where, bytes32 _permissionId) private view
 ↳  {
335     if (
336         !isGranted(address(this), msg.sender, _permissionId, msg.
 ↳ data) &&
337         !isGranted(_where, msg.sender, _permissionId, msg.data)
338     ) {
339         revert Unauthorized({
340             here: address(this),
341             where: _where,
342             who: msg.sender,
343             permissionId: _permissionId
```

```
        });
    }
}
```

Risk Level:

Likelihood - 1
Impact - 5

Recommendation:

It is recommended that the where check is removed from the _auth function on the PermissionManager. If that's not possible due to architectural designs on future releases, the documentation should clearly state that providing a different where clause rather than address(this) can lead to bad behaviors and possible privilege escalation issues.

- If the contract x inherits from permission manager, it should only be using the modifier such as auth(address(this, permission). If not, and it has auth(where, permission), this is dangerous because if user A is granted permission Y on address(this)= x, it automatically means it has permission Y anywhere.
- Plugins or other contracts that are part of the DAO shouldn't use the modifier of Permission Manager. If they do, the same situation as the previous point could happen. This is not possible unless a custom contract is written.

Remediation Plan:

SOLVED: The Aragon team did change all auth(where, permission) modifiers to use auth(permission) which explicitly uses the DAO authorization system and does not rely on the Permissionmanager function.

3.2 (HAL-02) UNTRUSTED PLUGIN USAGE CAN CAUSE DOS - MEDIUM

Description:

Installing or upgrading a plugin without prior validation can lead to the contract and DAO being nonfunctional. This is possible if a plugin does self-destruct during the installation or upgrade. Furthermore, not only caution should be taken during the installation/upgradability but also on the deployed implementation contract as it would lead all proxied contracts to be unusable.

Risk Level:

Likelihood - 1
Impact - 5

Recommendation:

All contracts using UUPSUpgradeable should make sure that the contract implementation is initialized and access controls mechanism are in place so no one can perform any upgrade and self-destruct the implementation contract itself. A good suggestion is to use _disableInitializers on those contract constructors. Doing so could cause all proxies to be invalid and nonfunctional. Furthermore, DAO's should make sure that used plugins do not contain untrusted calls and executions.

Remediation Plan:

SOLVED: The Aragon team added _disableInitializers and stated that no action could happen due to a bad implementation initialization or take-over.

3.3 (HAL-03) MERKLE ROOT RE-USAGE - LOW

Description:

The merkleMint function on the MerkleMinter contract does allow specifying a previous used _merkleRoot. The merkle proofs do not consider the token being used as part of the proof. This allows the same proof, with the same merkle root, to be used for any token. This means that the created MerkleDistributor will successfully mint the tokens independently of the internal token used for a valid proof. This scenario is unlikely to happen, but bots could be constantly scanning the blockchain for such scenario and profit from it.

Risk Level:

Likelihood - 3
Impact - 1

Recommendation:

The code should check that the _merkleRoot is not reused, this will prevent proof re-usage to mint another token from a previous used proof of another token. With the current code implementation, it is hard to prevent this scenario on multiple MerkleMinter being generated with the TokenFactory as it would require the factory to keep track of the roots. However, it is possible to protect against this on a MerkleMinter bases and assume the other scenario by checking the re-usage of the Merkle Root on the MerkleMinter contract itself.

Remediation Plan:

NOT APPLICABLE: The Aragon team stated that this is considered a feature rather than a potential issue, as this would allow multiple tokens to be generated for the same address list without having to modify the root.

3.4 (HAL-04) EMPTY SUBDOMAIN ALLOWED - LOW

Description:

The registerPluginRepo function on the PluginRepoRegistry contract does allow specifying an empty string as a subdomain, which will result in the c5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470 hash node. This is allowed due to the registering function and the isSubdomainValid internal function considering an empty string as a valid subdomain.

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

It is recommended that the isSubdomainValid function does verify that the string is not empty to cover all possible invalid subdomains.

Remediation Plan:

SOLVED: The code now reverts if the subdomain string is empty before the isSubdomainValid check.

3.5 (HAL-05) MISSING CALLBACK SELECTOR ZERO CASE - INFORMATIONAL

Description:

The _registerCallback function on the CallbackHandler contract does allow registering the _callbackSelector of 0x00000000 which refers to the Solidity EVM fallback or receive functions. The callBackSelector is expected to be equal to _magicNumber in most of the cases. However, it is possible that _magicNumber is different from callBackSelector. For the former, when they are equal, registering the _magicNumber/callBackSelector of 0x00000000 will cause any call not handled by the EVM, such as data=0x to revert even after register. However, the _handleCallback, does treat the 0x0 (_magicNubmer) as UNREGISTERED_CALLBACK which will cause the transaction to revert.

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended that if the DAO wants to register the callBackSelector=0x00000000 to use _magicNumber different from 0x0. Furthermore, it will be a nice option to enforce that the _registerCallback function _magicNumber is always different from 0x and add an unregisterCallback function which sets the _magicNumber for a given callBackSelector to 0x0.

Remediation Plan:

ACKNOWLEDGED: The Aragon team and Halborn agreed on the low severity of this issue and stated that reverting unhandled selectors to default is a good option, as it would force the developers to explicitly add the selector if it is required.

3.6 (HAL-06) UNUSED CODE - INFORMATIONAL

Description:

The _applyRatioFloored is never being used in the code. Furthermore, as the function name states, the result will be floored and the decimal precision lost. It is recommended to either remove the code or take extra precaution when using the function, as unexpected results may occur.

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to remove any unused code or provide valid recommendations and suggestions on the documentation on how to use those.

Remediation Plan:

SOLVED: The Aragon team did remove the unused code.

MANUAL TESTING

4.1 DAO

- Permissions are correctly described.
- No missing arguments and bad interface implementations.
- All inherited contracts are upgradeable safe.
- Initializer does have all initializers for inherited contracts.
- All calls do check for valid return value and revert otherwise (if not prevented by the failure mapping).
- Since the contract does use funds, it has both fallback and receive functions implemented.
- Setters do have ACL permissions.

4.2 PermissionManager

- All inherited contracts are upgradeable safe.
- Permissions are correctly described.
- No missing arguments and bad interface implementations.
- Granting permissions are implemented as expected
- Validating given permissions is implemented as expected, one issue described in the report as a vulnerability.
- Granting ROOT permissions or protected permission to ANY_ADDR is restricted correctly.
- Revoking does revoke correctly and works for ANY_ADDR. However, it should be stated that revoking a strictly specified where/from permission with ANY_ADDR will not revoke the permission. ANY_ADDR on the revoke function cannot be treated as a “clean all” permissions.

4.3 DAOFactory

- Internal permissions are correctly used.
- DAO permissions are the ones necessary for the operation and all registered in the isPermissionRestrictedForAnyAddr.
- All temporary permissions given to the factory are successfully

removed afterward.
- The newly created dao is registered on the DAORegistry.

4.4 InterfaceBasedRegistry

- It does not allow registering twice the same address
- All inherited contracts are upgradeable safe.
- Permissions are correctly described.
- No missing arguments and bad interface implementations.

4.5 PluginSetupProcessor

- Permissions are correctly described.
- No missing arguments and bad interface implementations.
- All the prepare statements are using preparedSetupIdToBlockNumber internally to store the current state. All the states are differentiated against them by the action, such as install, upgrade and uninstall. This is important; otherwise, a prepare statement will be applicable in any action independently of the prepare action. Meaning that a prepareUninstall could be used on the applyUpdate state, causing bad behaviours and unexpected results.
- The installation preparation does verify that the plugin repo is registered.
- The prepareUpdate does not verify that the repo is registered, as it is assumed to be if installed by checking currentAppliedSetupId.

4.6 ENSSubdomainRegistrar

- It does correctly create the ENS node by packing the base node and label, in this order.
- Label is being passed as the keccak256 of the subdomain as specified in the ENS documentation.

- Internal permissions are correctly used and delegated to the DAO.

4.7 AddresslistVoting

- Proposal creation is based on the Addresslist implementation, which uses the CheckpointsUpgradeable contract from OZ.
- The ability to create a proposal requires the member to have the voting power in the block before the proposal creation. This does prevent possible front-runs.
- If a user gets removed from the list, it will still be able to vote for proposals that were created before its removal.

4.8 TokenVoting

- Proposal creation is based on the ERC20Voting implementation, which uses snapshot mechanisms to store the balance for a given address back in time.
- The ability to create a proposal requires the member to have the voting power in the block before the proposal creation. This does prevent possible front-runs.
- New voting power (either increment or decrement) will not be taken into consideration for past proposals, which means that a user could still vote if it had enough power before the proposal was created.

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