// Security Assessment

02.19.2025 - 02.24.2025

Taiko DAO Contracts

Taiko

HALBORN

Taiko DAO Contracts - Taiko

Prepared by: HALBORN

Last Updated 03/24/2025

Date of Engagement: February 19th, 2025 - February 24th, 2025

Summary

100% ⓘ OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
13	0	0	0	1	12

TABLE OF CONTENTS

1. Introduction

Taiko Labs engaged Halborn to conduct a security assessment on their smart contracts beginning on February 19th, 2025 and ending on February 25th, 2025. The security assessment was scoped to the smart contracts provided to Halborn. Commit hashes and further details can be found in the Scope section of this report.

The Taiko Labs codebase in scope consists of a DAO protocol leveraged by Aragon.

2. Assessment Summary

Halborn was provided 5 days for the engagement and assigned 2 full-time security engineers to review the security of the smart contracts in scope. The engineers are blockchain and smart contract security experts with advanced penetration testing and smart contract hacking skills, and deep knowledge of multiple blockchain protocols.

The purpose of the assessment is to:

- Identify potential security issues within the smart contracts.
- Ensure that smart contract functionality operates as intended.

In summary, Halborn identified some improvements to reduce the likelihood and impact of risks, which were mostly addressed by the Taiko Labs team. The main ones are the following::

- Introduce require statements in the constructor or deployOnce to validate all parameters.
- Provide an upgrade path for long-lived DAOs.
- Capture the creator's owner/agent and store the agent used for encryption.

3. Test Approach And Methodology

Halborn performed a combination of manual review of the code 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 smart contracts 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, purpose and use of the platform.
- Smart contract manual code review and walkthrough to identify any logic issue.
- Thorough assessment of safety and usage of critical Solidity variables and functions in scope that could led to arithmetic related vulnerabilities.
- Local testing with custom scripts (Foundry).
- Fork testing against main networks (Foundry).
- Static analysis of security for scoped contract, and imported functions (Slither).

4. Static Analysis Report

4.1 Description

Halborn used automated testing techniques to enhance the coverage of certain areas of the smart contracts in scope. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified the smart contracts in the repository and was able to compile them correctly into their abis and binary format, Slither was run against the contracts. 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.

The security team assessed all findings identified by the Slither software, however, findings with related to external dependencies are not included in the below results for the sake of report readability.

4.2 Output

The findings obtained as a result of the Slither scan were reviewed, and many were not included in the report because they were determined as false positives.

```
INFO:Detectors:
OptimisticTokenVotingPlugin.createProposal(bytes,IDAO.Action[],uint256,uint64) (src/OptimisticTokenVotingPlugin.sol#357-421) uses a dangerous strict equality:
        - effectiveVotingPower(snapshotTimestamp,_enableL2) == 0 (src/OptimisticTokenVotingPlugin.sol#370)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
Reentrancy in TaikoDaoFactory.deployOnce() (src/factory/TaikoDaoFactory.sol#107-146):
        External calls:
        - dao = prepareDao() (src/factory/TaikoDaoFactory.sol#115)
                - address(new ERC1967Proxy(_logic,_data)) (lib/osx/packages/contracts/src/utils/Proxy.sol#13)
                - dao.applySingleTargetPermissions(address(dao),items) (src/factory/TaikoDaoFactory.sol#174)
        State variables written after the call(s):
        - deployment.dao = dao (src/factory/TaikoDaoFactory.sol#116)
        TaikoDaoFactory.deployment (src/factory/TaikoDaoFactory.sol#99) can be used in cross function reentrancies:
        - TaikoDaoFactory.deployOnce() (src/factory/TaikoDaoFactory.sol#107-146)
        - TaikoDaoFactory.getDeployment() (src/factory/TaikoDaoFactory.sol#354-356)
Reentrancy in TaikoDaoFactory.deployOnce() (src/factory/TaikoDaoFactory.sol#107-146):
        External calls:
        - dao = prepareDao() (src/factory/TaikoDaoFactory.sol#115)
                - address(new ERC1967Proxy(_logic,_data)) (lib/osx/packages/contracts/src/utils/Proxy.sol#13)
                - dao.applySingleTargetPermissions(address(dao),items) (src/factory/TaikoDaoFactory.sol#174)
        - (deployment.signerList,deployment.encryptionRegistry) = prepareSignerListAndEncryptionRegistry(dao) (src/factory/TaikoDaoFactory.sol#119)
                - address(new ERC1967Proxy(_logic,_data)) (lib/osx/packages/contracts/src/utils/Proxy.sol#13)
                - dao.grant(address(signerList),address(this),UPDATE_SIGNER_LIST_SETTINGS_PERMISSION_ID) (src/factory/TaikoDaoFactory.sol#184)
                - signerList.updateSettings(SignerList.Settings(encryptionRegistry,uint16(settings.multisigMembers.length))) (src/factory/TaikoDaoFactory.sol#186)
                - dao.revoke(address(signerList),address(this),UPDATE_SIGNER_LIST_SETTINGS_PERMISSION_ID) (src/factory/TaikoDaoFactory.sol#189)
                - dao.grant(address(signerList),address(dao),UPDATE_SIGNER_LIST_SETTINGS_PERMISSION_ID) (src/factory/TaikoDaoFactory.sol#193)
        State variables written after the call(s):
        - (deployment.signerList,deployment.encryptionRegistry) = prepareSignerListAndEncryptionRegistry(dao) (src/factory/TaikoDaoFactory.sol#119)
        TaikoDaoFactory.deployment (src/factory/TaikoDaoFactory.sol#99) can be used in cross function reentrancies:
        - TaikoDaoFactory.deployOnce() (src/factory/TaikoDaoFactory.sol#107-146)
        - TaikoDaoFactory.getDeployment() (src/factory/TaikoDaoFactory.sol#354-356)
Reentrancy in TaikoDaoFactory.deployOnce() (src/factory/TaikoDaoFactory.sol#107-146):
        External calls:
        - dao = prepareDao() (src/factory/TaikoDaoFactory.sol#115)
                - address(new ERC1967Proxy(_logic,_data)) (lib/osx/packages/contracts/src/utils/Proxy.sol#13)
                - dao.applySingleTargetPermissions(address(dao),items) (src/factory/TaikoDaoFactory.sol#174)
        - (deployment.signerList,deployment.encryptionRegistry) = prepareSignerListAndEncryptionRegistry(dao) (src/factory/TaikoDaoFactory.sol#119)
                - address(new ERC1967Proxy(_logic,_data)) (lib/osx/packages/contracts/src/utils/Proxy.sol#13)
                - dao.grant(address(signerList),address(this),UPDATE_SIGNER_LIST_SETTINGS_PERMISSION_ID) (src/factory/TaikoDaoFactory.sol#184)
                - signerList.updateSettings(SignerList.Settings(encryptionRegistry,uint16(settings.multisigMembers.length))) (src/factory/TaikoDaoFactory.sol#186)
                - dao.revoke(address(signerList),address(this),UPDATE_SIGNER_LIST_SETTINGS_PERMISSION_ID) (src/factory/TaikoDaoFactory.sol#189)
                - dao.grant(address(signerList),address(dao),UPDATE_SIGNER_LIST_SETTINGS_PERMISSION_ID) (src/factory/TaikoDaoFactory.sol#193)
        - (deployment.multisigPlugin,deployment.multisigPluginRepo,preparedMultisigSetupData) = prepareMultisig(dao,deployment.signerList) (src/factory/TaikoDaoFactory.sol#122)
                - pluginRepo = PluginRepoFactory(settings.pluginRepoFactory).createPluginRepoWithFirstVersion(settings.stdMultisigEnsDomain,address(settings.multisigPluginSetup),address(dao), , ) (src/factory/TaikoDaoFactory.sol#198-204)
                - dao.grant(address(pluginRepo),address(dao),pluginRepo.MAINTAINER_PERMISSION_ID()) (src/factory/TaikoDaoFactory.sol#205)
                - (plugin,preparedSetupData) = settings.pluginSetupProcessor.prepareInstallation(address(dao),PluginSetupProcessor.PrepareInstallationParams(PluginSetupRef(PluginRepo.Tag(1,1),PluginRepo(pluginRepo)),settingsData)) (src/factory/TaikoDaoFactory.sol#218-221)
        State variables written after the call(s):
        - (deployment.multisigPlugin,deployment.multisigPluginRepo,preparedMultisigSetupData) = prepareMultisig(dao,deployment.signerList) (src/factory/TaikoDaoFactory.sol#122)
        TaikoDaoFactory.deployment (src/factory/TaikoDaoFactory.sol#99) can be used in cross function reentrancies:
        - TaikoDaoFactory.deployOnce() (src/factory/TaikoDaoFactory.sol#107-146)
        - TaikoDaoFactory.getDeployment() (src/factory/TaikoDaoFactory.sol#354-356)
Reentrancy in TaikoDaoFactory.deployOnce() (src/factory/TaikoDaoFactory.sol#107-146):
        External calls:
        - dao = prepareDao() (src/factory/TaikoDaoFactory.sol#115)
                - address(new ERC1967Proxy(_logic,_data)) (lib/osx/packages/contracts/src/utils/Proxy.sol#13)
                - dao.applySingleTargetPermissions(address(dao),items) (src/factory/TaikoDaoFactory.sol#174)
        - (deployment.signerList,deployment.encryptionRegistry) = prepareSignerListAndEncryptionRegistry(dao) (src/factory/TaikoDaoFactory.sol#119)
                - address(new ERC1967Proxy(_logic,_data)) (lib/osx/packages/contracts/src/utils/Proxy.sol#13)
                - dao.grant(address(signerList),address(this),UPDATE_SIGNER_LIST_SETTINGS_PERMISSION_ID) (src/factory/TaikoDaoFactory.sol#184)
                - signerList.updateSettings(SignerList.Settings(encryptionRegistry,uint16(settings.multisigMembers.length))) (src/factory/TaikoDaoFactory.sol#186)
                - dao.revoke(address(signerList),address(this),UPDATE_SIGNER_LIST_SETTINGS_PERMISSION_ID) (src/factory/TaikoDaoFactory.sol#189)
                - dao.grant(address(signerList),address(dao),UPDATE_SIGNER_LIST_SETTINGS_PERMISSION_ID) (src/factory/TaikoDaoFactory.sol#193)
```

```
        - (deployment.multisigPlugin,deployment.multisigPluginRepo,preparedMultisigSetupData) = prepareMultisig(dao,deployment.signerList) (src/factory/TaikoDaoFactory.sol#122)
                - pluginRepo = PluginRepoFactory(settings.pluginRepoFactory).createPluginRepoWithFirstVersion(settings.stdMultisigEnsDomain,address(settings.multisigPluginSetup),address(dao), , ) (src/factory/TaikoDaoFactory.sol#198-204)
                - dao.grant(address(pluginRepo),address(dao),pluginRepo.MAINTAINER_PERMISSION_ID()) (src/factory/TaikoDaoFactory.sol#205)
                - (plugin,preparedSetupData) = settings.pluginSetupProcessor.prepareInstallation(address(dao),PluginSetupProcessor.PrepareInstallationParams(PluginSetupRef(PluginRepo.Tag(1,1),PluginRepo(pluginRepo)),settingsData)) (src/factory/TaikoDaoFactory.sol#218-221)
        - (deployment.emergencyMultisigPlugin,deployment.emergencyMultisigPluginRepo,preparedEmergencyMultisigSetupData) = prepareEmergencyMultisig(dao,deployment.signerList) (src/factory/TaikoDaoFactory.sol#124-127)
                - pluginRepo = PluginRepoFactory(settings.pluginRepoFactory).createPluginRepoWithFirstVersion(settings.emergencyMultisigEnsDomain,address(settings.emergencyMultisigPluginSetup),address(dao), , ) (src/factory/TaikoDaoFactory.sol#228-234)
                - dao.grant(address(pluginRepo),address(dao),pluginRepo.MAINTAINER_PERMISSION_ID()) (src/factory/TaikoDaoFactory.sol#235)
                - (plugin,preparedSetupData) = settings.pluginSetupProcessor.prepareInstallation(address(dao),PluginSetupProcessor.PrepareInstallationParams(PluginSetupRef(PluginRepo.Tag(1,1),PluginRepo(pluginRepo)),settingsData)) (src/factory/TaikoDaoFactory.sol#247-250)
        State variables written after the call(s):
        - (deployment.emergencyMultisigPlugin,deployment.emergencyMultisigPluginRepo,preparedEmergencyMultisigSetupData) = prepareEmergencyMultisig(dao,deployment.signerList) (src/factory/TaikoDaoFactory.sol#124-127)
        TaikoDaoFactory.deployment (src/factory/TaikoDaoFactory.sol#99) can be used in cross function reentrancies:
        - TaikoDaoFactory.deployOnce() (src/factory/TaikoDaoFactory.sol#107-146)
        - TaikoDaoFactory.getDeployment() (src/factory/TaikoDaoFactory.sol#354-356)
Reentrancy in TaikoDaoFactory.deployOnce() (src/factory/TaikoDaoFactory.sol#107-146):
        External calls:
        - dao = prepareDao() (src/factory/TaikoDaoFactory.sol#115)
                - address(new ERC1967Proxy(_logic,_data)) (lib/osx/packages/contracts/src/utils/Proxy.sol#13)
                - dao.applySingleTargetPermissions(address(dao),items) (src/factory/TaikoDaoFactory.sol#174)
        - (deployment.signerList,deployment.encryptionRegistry) = prepareSignerListAndEncryptionRegistry(dao) (src/factory/TaikoDaoFactory.sol#119)
                - address(new ERC1967Proxy(_logic,_data)) (lib/osx/packages/contracts/src/utils/Proxy.sol#13)
                - dao.grant(address(signerList),address(this),UPDATE_SIGNER_LIST_SETTINGS_PERMISSION_ID) (src/factory/TaikoDaoFactory.sol#184)
                - signerList.updateSettings(SignerList.Settings(encryptionRegistry,uint16(settings.multisigMembers.length))) (src/factory/TaikoDaoFactory.sol#186)
                - dao.revoke(address(signerList),address(this),UPDATE_SIGNER_LIST_SETTINGS_PERMISSION_ID) (src/factory/TaikoDaoFactory.sol#189)
                - dao.grant(address(signerList),address(dao),UPDATE_SIGNER_LIST_SETTINGS_PERMISSION_ID) (src/factory/TaikoDaoFactory.sol#193)
        - (deployment.multisigPlugin,deployment.multisigPluginRepo,preparedMultisigSetupData) = prepareMultisig(dao,deployment.signerList) (src/factory/TaikoDaoFactory.sol#122)
                - pluginRepo = PluginRepoFactory(settings.pluginRepoFactory).createPluginRepoWithFirstVersion(settings.stdMultisigEnsDomain,address(settings.multisigPluginSetup),address(dao), , ) (src/factory/TaikoDaoFactory.sol#198-204)
                - dao.grant(address(pluginRepo),address(dao),pluginRepo.MAINTAINER_PERMISSION_ID()) (src/factory/TaikoDaoFactory.sol#205)
                - (plugin,preparedSetupData) = settings.pluginSetupProcessor.prepareInstallation(address(dao),PluginSetupProcessor.PrepareInstallationParams(PluginSetupRef(PluginRepo.Tag(1,1),PluginRepo(pluginRepo)),settingsData)) (src/factory/TaikoDaoFactory.sol#218-221)
        - (deployment.emergencyMultisigPlugin,deployment.emergencyMultisigPluginRepo,preparedEmergencyMultisigSetupData) = prepareEmergencyMultisig(dao,deployment.signerList) (src/factory/TaikoDaoFactory.sol#124-127)
                - pluginRepo = PluginRepoFactory(settings.pluginRepoFactory).createPluginRepoWithFirstVersion(settings.emergencyMultisigEnsDomain,address(settings.emergencyMultisigPluginSetup),address(dao), , ) (src/factory/TaikoDaoFactory.sol#228-234)
                - dao.grant(address(pluginRepo),address(dao),pluginRepo.MAINTAINER_PERMISSION_ID()) (src/factory/TaikoDaoFactory.sol#235)
                - (plugin,preparedSetupData) = settings.pluginSetupProcessor.prepareInstallation(address(dao),PluginSetupProcessor.PrepareInstallationParams(PluginSetupRef(PluginRepo.Tag(1,1),PluginRepo(pluginRepo)),settingsData)) (src/factory/TaikoDaoFactory.sol#247-250)
        - (deployment.optimisticTokenVotingPlugin,deployment.optimisticTokenVotingPluginRepo,preparedOptimisticSetupData) = prepareOptimisticTokenVoting(dao,address(deployment.multisigPlugin),address(deployment.emergencyMultisigPlugin)) (src/factory/TaikoDaoFactory.sol#129-133)
                - pluginRepo = PluginRepoFactory(settings.pluginRepoFactory).createPluginRepoWithFirstVersion(settings.optimisticTokenVotingEnsDomain,address(settings.optimisticTokenVotingPluginSetup),address(dao), , ) (src/factory/TaikoDaoFactory.sol#261-267)
                - dao.grant(address(pluginRepo),address(dao),pluginRepo.MAINTAINER_PERMISSION_ID()) (src/factory/TaikoDaoFactory.sol#268)
                - (plugin,preparedSetupData) = settings.pluginSetupProcessor.prepareInstallation(address(dao),PluginSetupProcessor.PrepareInstallationParams(PluginSetupRef(PluginRepo.Tag(1,1),PluginRepo(pluginRepo)),settingsData)) (src/factory/TaikoDaoFactory.sol#304-307)
        State variables written after the call(s):
        - (deployment.optimisticTokenVotingPlugin,deployment.optimisticTokenVotingPluginRepo,preparedOptimisticSetupData) = prepareOptimisticTokenVoting(dao,address(deployment.multisigPlugin),address(deployment.emergencyMultisigPlugin)) (src/factory/TaikoDaoFactory.sol#129-133)
        TaikoDaoFactory.deployment (src/factory/TaikoDaoFactory.sol#99) can be used in cross function reentrancies:
        - TaikoDaoFactory.deployOnce() (src/factory/TaikoDaoFactory.sol#107-146)
        - TaikoDaoFactory.getDeployment() (src/factory/TaikoDaoFactory.sol#354-356)
Reentrancy in PluginSetupProcessor.prepareUninstallation(address,PluginSetupProcessor.PrepareUninstallationParams) (lib/osx/packages/contracts/src/framework/plugin/setup/PluginSetupProcessor.sol#552-605):
        External calls:
        - permissions = PluginSetup(version.pluginSetup).prepareUninstallation(_dao,_params.setupPayload) (lib/osx/packages/contracts/src/framework/plugin/setup/PluginSetupProcessor.sol#576-579)
        State variables written after the call(s):
        - pluginState.preparedSetupIdToBlockNumber[preparedSetupId] = block.number (lib/osx/packages/contracts/src/framework/plugin/setup/PluginSetupProcessor.sol#594)
        PluginSetupProcessor.states (lib/osx/packages/contracts/src/framework/plugin/setup/PluginSetupProcessor.sol#59) can be used in cross function reentrancies:
        - PluginSetupProcessor.applyInstallation(address,PluginSetupProcessor.ApplyInstallationParams) (lib/osx/packages/contracts/src/framework/plugin/setup/PluginSetupProcessor.sol#349-388)
        - PluginSetupProcessor.applyUninstallation(address,PluginSetupProcessor.ApplyUninstallationParams) (lib/osx/packages/contracts/src/framework/plugin/setup/PluginSetupProcessor.sol#612-644)
        - PluginSetupProcessor.applyUpdate(address,PluginSetupProcessor.ApplyUpdateParams) (lib/osx/packages/contracts/src/framework/plugin/setup/PluginSetupProcessor.sol#500-545)
        - PluginSetupProcessor.prepareInstallation(address,PluginSetupProcessor.PrepareInstallationParams) (lib/osx/packages/contracts/src/framework/plugin/setup/PluginSetupProcessor.sol#286-344)
        - PluginSetupProcessor.prepareUninstallation(address,PluginSetupProcessor.PrepareUninstallationParams) (lib/osx/packages/contracts/src/framework/plugin/setup/PluginSetupProcessor.sol#552-605)
        - PluginSetupProcessor.prepareUpdate(address,PluginSetupProcessor.PrepareUpdateParams) (lib/osx/packages/contracts/src/framework/plugin/setup/PluginSetupProcessor.sol#396-495)
        - PluginSetupProcessor.states (lib/osx/packages/contracts/src/framework/plugin/setup/PluginSetupProcessor.sol#59)
        - PluginSetupProcessor.validatePreparedSetupId(bytes32,bytes32) (lib/osx/packages/contracts/src/framework/plugin/setup/PluginSetupProcessor.sol#650-658)
Reentrancy in PluginSetupProcessor.prepareUpdate(address,PluginSetupProcessor.PrepareUpdateParams) (lib/osx/packages/contracts/src/framework/plugin/setup/PluginSetupProcessor.sol#396-495):
        External calls:
        - (initData,preparedSetupData) = PluginSetup(newVersion.pluginSetup).prepareUpdate(_dao,_params.currentVersionTag.build,_params.setupPayload) (lib/osx/packages/contracts/src/framework/plugin/setup/PluginSetupProcessor.sol#469-473)
        State variables written after the call(s):
        - pluginState.preparedSetupIdToBlockNumber[preparedSetupId] = block.number (lib/osx/packages/contracts/src/framework/plugin/setup/PluginSetupProcessor.sol#489)
        PluginSetupProcessor.states (lib/osx/packages/contracts/src/framework/plugin/setup/PluginSetupProcessor.sol#59) can be used in cross function reentrancies:
        - PluginSetupProcessor.applyInstallation(address,PluginSetupProcessor.ApplyInstallationParams) (lib/osx/packages/contracts/src/framework/plugin/setup/PluginSetupProcessor.sol#349-388)
```

```
        - PluginSetupProcessor.applyUninstallation(address,PluginSetupProcessor.ApplyUninstallationParams) (lib/osx/packages/contracts/src/framework/plugin/setup/PluginSetupProcessor.sol#612-644)
        - PluginSetupProcessor.applyUpdate(address,PluginSetupProcessor.ApplyUpdateParams) (lib/osx/packages/contracts/src/framework/plugin/setup/PluginSetupProcessor.sol#500-545)
        - PluginSetupProcessor.prepareInstallation(address,PluginSetupProcessor.PrepareInstallationParams) (lib/osx/packages/contracts/src/framework/plugin/setup/PluginSetupProcessor.sol#286-344)
        - PluginSetupProcessor.prepareUninstallation(address,PluginSetupProcessor.PrepareUninstallationParams) (lib/osx/packages/contracts/src/framework/plugin/setup/PluginSetupProcessor.sol#552-605)
        - PluginSetupProcessor.prepareUpdate(address,PluginSetupProcessor.PrepareUpdateParams) (lib/osx/packages/contracts/src/framework/plugin/setup/PluginSetupProcessor.sol#396-495)
        - PluginSetupProcessor.states (lib/osx/packages/contracts/src/framework/plugin/setup/PluginSetupProcessor.sol#59)
        - PluginSetupProcessor.validatePreparedSetupId(bytes32,bytes32) (lib/osx/packages/contracts/src/framework/plugin/setup/PluginSetupProcessor.sol#650-658)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
INFO:Detectors:
EncryptionRegistry.appointAgent(address).exists (src/EncryptionRegistry.sol#58) is a local variable never initialized
EncryptionRegistry._setPublicKey(address,bytes32).exists (src/EncryptionRegistry.sol#144) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables
INFO:Detectors:
EmergencyMultisig._execute(uint256,bytes,IDAO.Action[]) (src/EmergencyMultisig.sol#337-349) ignores return value by proposal_.destinationPlugin.createProposal(_metadataUri,_actions,0,0) (src/EmergencyMultisig.sol#343-348)
Multisig._execute(uint256) (src/Multisig.sol#322-334) ignores return value by proposal_.destinationPlugin.createProposal(proposal_.metadataURI,proposal_.destinationActions,0,multisigSettings.destinationProposalDuration) (src/Multisig.sol#328-333)
OptimisticTokenVotingPlugin.isL2Available() (src/OptimisticTokenVotingPlugin.sol#218-238) ignores return value by (verifiedAt) = taikoL1.getLastVerifiedBlock() (src/OptimisticTokenVotingPlugin.sol#230-236)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Detectors:
OptimisticTokenVotingPlugin.initialize(IDAO,OptimisticTokenVotingPlugin.OptimisticGovernanceSettings,IVotesUpgradeable,address,address)._taikoBridge (src/OptimisticTokenVotingPlugin.sol#171) lacks a zero-check on :
                - taikoBridge = _taikoBridge (src/OptimisticTokenVotingPlugin.sol#179)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
SignerList.getEncryptionAgents() (src/SignerList.sol#143-183) has external calls inside a loop: appointed = settings.encryptionRegistry.getAppointedAgent(_encryptionAccounts[accIdx]) (src/SignerList.sol#156)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#calls-inside-a-loop
INFO:Detectors:
EmergencyMultisig._isProposalOpen(EmergencyMultisig.Proposal) (src/EmergencyMultisig.sol#400-403) uses timestamp for comparisons
        Dangerous comparisons:
        - ! proposal_.executed && proposal_.parameters.expirationDate > currentTimestamp64 (src/EmergencyMultisig.sol#402)
Multisig._isProposalOpen(Multisig.Proposal) (src/Multisig.sol#385-388) uses timestamp for comparisons
        Dangerous comparisons:
        - ! proposal_.executed && proposal_.parameters.expirationDate > currentTimestamp64 (src/Multisig.sol#387)
OptimisticTokenVotingPlugin.isL2Available() (src/OptimisticTokenVotingPlugin.sol#218-238) uses timestamp for comparisons
        Dangerous comparisons:
        - (verifiedAt + governanceSettings.l2InactivityPeriod) < block.timestamp (src/OptimisticTokenVotingPlugin.sol#232)
OptimisticTokenVotingPlugin.createProposal(bytes,IDAO.Action[],uint256,uint64) (src/OptimisticTokenVotingPlugin.sol#357-421) uses timestamp for comparisons
        Dangerous comparisons:
        - _enableL2 = isL2Available() && votingToken.getPastVotes(taikoBridge,snapshotTimestamp) > 0 (src/OptimisticTokenVotingPlugin.sol#369)
        - effectiveVotingPower(snapshotTimestamp,_enableL2) == 0 (src/OptimisticTokenVotingPlugin.sol#370)
        - ! _enableL2 || _duration == 0 (src/OptimisticTokenVotingPlugin.sol#401)
OptimisticTokenVotingPlugin._isProposalOpen(OptimisticTokenVotingPlugin.Proposal) (src/OptimisticTokenVotingPlugin.sol#546-548) uses timestamp for comparisons
        Dangerous comparisons:
        - block.timestamp.toUint64() < proposal_.parameters.vetoEndDate && ! proposal_.executed (src/OptimisticTokenVotingPlugin.sol#547)
OptimisticTokenVotingPlugin._isProposalEnded(OptimisticTokenVotingPlugin.Proposal) (src/OptimisticTokenVotingPlugin.sol#553-557) uses timestamp for comparisons
        Dangerous comparisons:
        - currentTime ≥ proposal_.parameters.vetoEndDate (src/OptimisticTokenVotingPlugin.sol#556)
OptimisticTokenVotingPlugin._proposalL2VetoAggregationOpen(OptimisticTokenVotingPlugin.Proposal) (src/OptimisticTokenVotingPlugin.sol#574-581) uses timestamp for comparisons
        Dangerous comparisons:
        - block.timestamp.toUint64() < proposal_.parameters.vetoEndDate + governanceSettings.l2AggregationGracePeriod (src/OptimisticTokenVotingPlugin.sol#579-580)
OptimisticTokenVotingPlugin._proposalInExitWindow(OptimisticTokenVotingPlugin.Proposal) (src/OptimisticTokenVotingPlugin.sol#586-594) uses timestamp for comparisons
        Dangerous comparisons:
        - block.timestamp.toUint64() < exitWindowTimestamp (src/OptimisticTokenVotingPlugin.sol#593)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
SignerList.getEncryptionAgents() (src/SignerList.sol#143-183) uses assembly
        - INLINE ASM (src/SignerList.sol#179-181)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
EncryptionRegistry.appointAgent(address) (src/EncryptionRegistry.sol#41-91) has a high cyclomatic complexity (12).
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#cyclomatic-complexity
INFO:Detectors:
createProxyAndCall(address,bytes) (src/helpers/proxy.sol#6-8) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
```

```
INFO:Detectors:
Low level call in OptimisticTokenVotingPluginSetup._supportsErc20(address) (src/setup/OptimisticTokenVotingPluginSetup.sol#303-315):
        - (success,data) = token.staticcall(abi.encodeCall(IERC20Upgradeable.balanceOf,(address(this)))) (src/setup/OptimisticTokenVotingPluginSetup.sol#304-305)
        - (success,data) = token.staticcall(abi.encodeCall(IERC20Upgradeable.totalSupply,())) (src/setup/OptimisticTokenVotingPluginSetup.sol#308)
        - (success,data) = token.staticcall(abi.encodeCall(IERC20Upgradeable.allowance,(address(this),address(this)))) (src/setup/OptimisticTokenVotingPluginSetup.sol#311)
Low level call in OptimisticTokenVotingPluginSetup._supportsIVotes(address) (src/setup/OptimisticTokenVotingPluginSetup.sol#320-335):
        - (success,data) = token.staticcall(abi.encodeCall(IVotesUpgradeable.getVotes,(address(this)))) (src/setup/OptimisticTokenVotingPluginSetup.sol#321-322)
        - (success,data) = token.staticcall(abi.encodeCall(IVotesUpgradeable.getPastVotes,(address(this),0))) (src/setup/OptimisticTokenVotingPluginSetup.sol#325)
        - (success,data) = token.staticcall(abi.encodeCall(IVotesUpgradeable.getPastTotalSupply,(0))) (src/setup/OptimisticTokenVotingPluginSetup.sol#328)
        - (success,data) = token.staticcall(abi.encodeCall(IVotesUpgradeable.delegates,(address(this)))) (src/setup/OptimisticTokenVotingPluginSetup.sol#331)
Low level call in OptimisticTokenVotingPluginSetup._supportsIGovernanceWrappedERC20(address) (src/setup/OptimisticTokenVotingPluginSetup.sol#340-345):
        - (success,data) = token.staticcall(abi.encodeCall(IERC165.supportsInterface,(bytes4(0)))) (src/setup/OptimisticTokenVotingPluginSetup.sol#341)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
EmergencyMultisig.__gap (src/EmergencyMultisig.sol#434) is never used in EmergencyMultisig (src/EmergencyMultisig.sol#19-435)
Multisig.__gap (src/Multisig.sol#420) is never used in Multisig (src/Multisig.sol#21-421)
OptimisticTokenVotingPlugin.__gap (src/OptimisticTokenVotingPlugin.sol#597) is never used in OptimisticTokenVotingPlugin (src/OptimisticTokenVotingPlugin.sol#24-598)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variable
INFO:Detectors:
Loop condition i < accountList.length (src/EncryptionRegistry.sol#59) should use cached array length instead of referencing `length` member of the storage array.
 Loop condition i < accountList.length (src/EncryptionRegistry.sol#145) should use cached array length instead of referencing `length` member of the storage array.
 Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#cache-array-length
INFO:Detectors:
EncryptionRegistry.addresslist (src/EncryptionRegistry.sol#30) should be immutable
StandardProposalCondition.dao (src/conditions/StandardProposalCondition.sol#14) should be immutable
StandardProposalCondition.minDuration (src/conditions/StandardProposalCondition.sol#15) should be immutable
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-immutable
INFO:Detectors:
The function SignerList.resolveEncryptionAccountAtBlock(address,uint256) (src/SignerList.sol#127-140) reads this.isListedAtBlock(_appointer,_blockNumber) (src/SignerList.sol#134) with `this` which adds an extra STATICCALL.
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-variable-read-in-external-context
INFO:Slither:. analyzed (111 contracts with 93 detectors), 38 result(s) found
```

5. RISK METHODOLOGY

Every vulnerability and issue observed by Halborn is ranked based on **two sets** of **Metrics** and a **Severity Coefficient**. This system is inspired by the industry standard Common Vulnerability Scoring System.

The two **Metric sets** are: **Exploitability** and **Impact**. **Exploitability** captures the ease and technical means by which vulnerabilities can be exploited and **Impact** describes the consequences of a successful exploit.

The **Severity Coefficients** is designed to further refine the accuracy of the ranking with two factors: **Reversibility** and **Scope**. These capture the impact of the vulnerability on the environment as well as the number of users and smart contracts affected.

The final score is a value between 0-10 rounded up to 1 decimal place and 10 corresponding to the highest security risk. This provides an objective and accurate rating of the severity of security vulnerabilities in smart contracts.

The system is designed to assist in identifying and prioritizing vulnerabilities based on their level of risk to address the most critical issues in a timely manner.

5.1 EXPLOITABILITY

ATTACK ORIGIN (AO):

Captures whether the attack requires compromising a specific account.

ATTACK COST (AC):

Captures the cost of exploiting the vulnerability incurred by the attacker relative to sending a single transaction on the relevant blockchain. Includes but is not limited to financial and computational cost.

ATTACK COMPLEXITY (AX):

Describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Includes but is not limited to macro situation, available third-party liquidity and regulatory challenges.

METRICS:

EXPLOITABILITY METRIC (M_E)	METRIC VALUE	NUMERICAL VALUE
Attack Origin (AO)	Arbitrary (AO:A) Specific (AO:S)	1 0.2

EXPLOITABILITY METRIC (M_E)	METRIC VALUE	NUMERICAL VALUE
Attack Cost (AC)	Low (AC:L) Medium (AC:M) High (AC:H)	1 0.67 0.33
Attack Complexity (AX)	Low (AX:L) Medium (AX:M) High (AX:H)	1 0.67 0.33

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

5.2 IMPACT

CONFIDENTIALITY (C):

Measures the impact to the confidentiality of the information resources managed by the contract due to a successfully exploited vulnerability. Confidentiality refers to limiting access to authorized users only.

INTEGRITY (I):

Measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of data stored and/or processed on-chain. Integrity impact directly affecting Deposit or Yield records is excluded.

AVAILABILITY (A):

Measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. This metric refers to smart contract features and functionality, not state. Availability impact directly affecting Deposit or Yield is excluded.

DEPOSIT (D):

Measures the impact to the deposits made to the contract by either users or owners.

YIELD (Y):

Measures the impact to the yield generated by the contract for either users or owners.

METRICS:

IMPACT METRIC (M_I)	METRIC VALUE	NUMERICAL VALUE
Confidentiality (C)	None (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1
Integrity (I)	None (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1
Availability (A)	None (A:N) Low (A:L) Medium (A:M) High (A:H) Critical (A:C)	0 0.25 0.5 0.75 1
Deposit (D)	None (D:N) Low (D:L) Medium (D:M) High (D:H) Critical (D:C)	0 0.25 0.5 0.75 1
Yield (Y)	None (Y:N) Low (Y:L) Medium (Y:M) High (Y:H) Critical (Y:C)	0 0.25 0.5 0.75 1

Impact I is calculated using the following formula:

$$I = max(m_I) + \frac{\sum m_I - max(m_I)}{4}$$

5.3 SEVERITY COEFFICIENT

REVERSIBILITY (R):

Describes the share of the exploited vulnerability effects that can be reversed. For upgradeable contracts, assume the contract private key is available.

SCOPE (S):

Captures whether a vulnerability in one vulnerable contract impacts resources in other contracts.

METRICS:

SEVERITY COEFFICIENT (C)	COEFFICIENT VALUE	NUMERICAL VALUE
Reversibility (r)	None (R:N) Partial (R:P) Full (R:F)	1 0.5 0.25

SEVERITY COEFFICIENT (C)	COEFFICIENT VALUE	NUMERICAL VALUE
Scope (s)	Changed (S:C) Unchanged (S:U)	1.25 1

Severity Coefficient C is obtained by the following product:

$$C = rs$$

The Vulnerability Severity Score S is obtained by:

$$S = min(10, EIC * 10)$$

The score is rounded up to 1 decimal places.

SEVERITY	SCORE VALUE RANGE
Critical	9 - 10
High	7 - 8.9
Medium	4.5 - 6.9
Low	2 - 4.4
Informational	0 - 1.9

6. SCOPE

FILES AND REPOSITORY

(a) Repository:

(b) Assessed Commit ID: eed36d3

(c) Items in scope:

- src/adapted-dependencies/ITaikoL1.sol
- src/conditions/StandardProposalCondition.sol
- src/factory/TaikoDaoFactory.sol
- src/helpers/proxy.sol
- src/setup/EmergencyMultisigPluginSetup.sol
- src/setup/MultisigPluginSetup.sol
- src/setup/OptimisticTokenVotingPluginSetup.sol
- src/DelegationWall.sol
- src/EmergencyMultisig.sol
- src/EncryptionRegistry.sol
- src/Multisig.sol
- src/OptimisticTokenVotingPlugin.sol
- src/SignerList.sol

Out-of-Scope: Third party dependencies and economic attacks.

REMEDIATION COMMIT ID:

- https://github.com/aragon/taiko-contracts/pull/50/commits/f80fb99dea86277dfadcdcacdf869689f804c53b
- https://github.com/aragon/taiko-contracts/pull/50/commits/2628cad4b3af2bddefb69b5f2d02b04ad29f5eb4
- https://github.com/aragon/taiko-contracts/pull/50/commits/a8fa1e56c8a0ff1aaf5ff449cc67c1b2f7eea591
- https://github.com/aragon/taiko-contracts/pull/50/commits/62b17859716d3482800ae08f6e721fdd5d55c95f
- https://github.com/aragon/taiko-contracts/pull/50/commits/eb18176260c3e200ae79f74ddb233b81790dd601

Out-of-Scope: New features/implementations after the remediation commit IDs.

7. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW
0	0	0	1

INFORMATIONAL
12

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
INCONSISTENT SNAPSHOT FOR ENCRYPTION AGENTS	LOW	RISK ACCEPTED - 03/13/2025
UNUSED COMPONENTS	INFORMATIONAL	SOLVED - 03/13/2025
PUBLIC FUNCTIONS NOT INVOKED INTERNALLY	INFORMATIONAL	ACKNOWLEDGED - 03/20/2025
LACK OF ACCOUNT REMOVAL MECHANISM IN REGISTRY	INFORMATIONAL	SOLVED - 03/17/2025
MISSING INPUT VALIDATION	INFORMATIONAL	PARTIALLY SOLVED - 03/14/2025
MISSING VISIBILITY MODIFIER	INFORMATIONAL	SOLVED - 03/14/2025
EMPTY 'REVERT' STATEMENT	INFORMATIONAL	SOLVED - 03/14/2025
MISSING ADDRESS VALIDATION IN SIGNER MANAGEMENT	INFORMATIONAL	ACKNOWLEDGED - 03/20/2025

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
UNHANDLED RETURN VALUES	INFORMATIONAL	ACKNOWLEDGED - 03/20/2025
FLOATING PRAGMA	INFORMATIONAL	ACKNOWLEDGED - 03/20/2025
REDUNDANT USE OF `THIS` KEYWORD	INFORMATIONAL	SOLVED - 03/14/2025
MISSING EVENTS	INFORMATIONAL	ACKNOWLEDGED - 03/20/2025
TYPO IN ERROR NAME	INFORMATIONAL	SOLVED - 03/14/2025

8. FINDINGS & TECH DETAILS

8.1 INCONSISTENT SNAPSHOT FOR ENCRYPTION AGENTS

// LOW

Description

The system does not snapshot which encryption agent was tied to a signer at proposal creation, which can lead to a **voting integrity issue** if agents are changed during an active proposal. The `SignerList.resolveEncryptionAccountAtBlock()` function uses the current `EncryptionRegistry` mapping to resolve an approver's owner/agent relationship, but only the signer's listed status is checked at the historical block.

This means if an owner replaces their agent after the proposal was created, the **new** agent (who was not part of the original encrypted payload distribution) **is now allowed to approve the pending proposal**. Conversely, the original agent (who had the encrypted details) **would no longer be recognized after removal** (since `appointerOf(oldAgent)` becomes **0 (zero)** and thus cannot approve.

This dynamic can be problematic, as a newly appointed agent might cast a vote without actually knowing the proposal's content (if the encryption key changed or wasn't shared). It also lets a signer potentially **rotate agents to influence a vote** – for example, if an owner's original agent was uncooperative, the owner could appoint a new agent who will blindly approve, and the contract would count it. While this doesn't allow unauthorized entities to vote (the new agent is still appointed by a listed signer), it breaks the expectation that only those privy to the original proposal can vote on it. It slightly undermines the integrity of the emergency voting process by not locking in the "voting identity" (owner or their delegate) at proposal creation.

Code Location: `SignerList.sol` – `resolveEncryptionAccountAtBlock()` uses the current `encryptionRegistry.appointerOf(_address)` without a historical reference. There is no storage of the agent at snapshot time in the `Proposal` struct. The **EmergencyMultisig** `_canApprove()` then bases approval on this potentially updated information.

```
    function resolveEncryptionAccountAtBlock(address _address, uint256 _blockNumber)
        public
        view
        returns (address _owner, address _agent)
    {
        if (isListedAtBlock(_address, _blockNumber)) {
            // The owner + the agent
            return (_address, settings.encryptionRegistry.getAppointedAgent(_address));
        }

        address _appointer = settings.encryptionRegistry.appointerOf(_address);
        if (this.isListedAtBlock(_appointer, _blockNumber)) {
            // The appointed agent votes
            return (_appointer, _address);
        }

        // Not found, returning empty addresses
    }
```

BVSS

AO:S/AC:L/AX:L/R:N/S:U/C:H/A:M/I:H/D:N/Y:N (2.1)

Recommendation

In the `createProposal()` functions of the **EmergencyMultisig** and **Multisig** contracts, capture the creator's owner/agent and store the agent used for encryption (since all approvers would likely use the same mapping at creation time). Then, in `_canApprove()`, require that if an agent was set at creation for that owner, only that specific agent can approve.

Alternatively, treat an agent change as invalid for existing proposals: if `appointerOf(msg.sender)` at the snapshot block doesn't match the current appointer (meaning the agent changed), then reject the approval **or** require the original agent to vote. This ensures the individuals who actually have the decrypted proposal (the original agent or the owner with original key) are the ones voting.
Implementing a full snapshot of agent mappings might be complex; at minimum, document this behavior so the council knows not to change agents during active proposals. As a procedural mitigation, the Security Council should refrain from rotating encryption agents until all active emergency proposals are resolved.

Remediation Comment

RISK ACCEPTED: The **Taiko Labs team** has accepted the risk related to this finding.

References

aragon/taiko-contracts/src/SignerList.sol#L134-L151
aragon/taiko-contracts/src/EmergencyMultisig.sol#L366

8.2 UNUSED COMPONENTS

// INFORMATIONAL

Description

Throughout the files in scope, there are several instances where components are declared but never used. Instances of this issue include:

- In the `StandardProposalCondition` contract, the `dao` variable is declared but not used throughout the contract.
- In the `Multisig` contract the `InvalidAddressListSource` error is declared but never used.
- In the `OptimisticTokenVotingPlugin` contract the `ProposalCreationForbidden` error is declared but never used.

Additionally, it was identified that several imported contracts or interfaces are not used within the proposed scope.

– `src/SignerList.sol`

```
import {IERC165} from "@openzeppelin/contracts/utils/introspection/IERC165.sol";
```

– `src/factory/TaikoDaoFactory.sol`

```
import {Addresslist} from "@aragon/osx/plugins/utils/Addresslist.sol";

import {ERC1967Proxy} from "@openzeppelin/contracts/proxy/ERC1967/ERC1967Proxy.sol";
```

– `src/setup/EmergencyMultisigPluginSetup.sol`

```
import {DAO} from "@aragon/osx/core/dao/DAO.sol";
```

– `src/setup/MultisigPluginSetup.sol`

```
import {DAO} from "@aragon/osx/core/dao/DAO.sol";
```

– `src/setup/OptimisticTokenVotingPluginSetup.sol`

```
import {ITaikoL1} from "../adapted-dependencies/ITaikoL1.sol";
```

BVSS

AO:A/AC:L/AX:M/R:N/S:U/C:N/A:N/I:L/D:N/Y:N (1.7)

Recommendation

- Remove the unused `dao` variable from the `StandardProposalCondition` contract. Alternatively, if the variable is intended to be used, implement the necessary logic.

- Remove unused custom errors and ensure that any relevant error checks consistently reference active custom errors. This maintains a clean codebase, reduces confusion, and makes the contract's logic clearer for future readers and maintainers.
- Remove unused imports in order to increase code maintainability and avoid unnecessary bloat.

Remediation Comment

SOLVED: The **Taiko Labs team** solved this finding in commit `f80fb99` by following the mentioned recommendation.

Remediation Hash

https://github.com/aragon/taiko-contracts/pull/50/commits/f80fb99dea86277dfadcdcacdf869689f804c53b

References

aragon/taiko-contracts/src/conditions/StandardProposalCondition.sol#L14
aragon/taiko-contracts/src/SignerList.sol#L6
aragon/taiko-contracts/src/factory/TaikoDaoFactory.sol#L14-L15
aragon/taiko-contracts/src/factory/TaikoDaoFactory.sol#L20
aragon/taiko-contracts/src/interfaces/IEmergencyMultisig.sol#L5
aragon/taiko-contracts/src/interfaces/IMultisig.sol#L5
aragon/taiko-contracts/src/interfaces/IOptimisticTokenVoting.sol#L6
aragon/taiko-contracts/src/setup/EmergencyMultisigPluginSetup.sol#L6
aragon/taiko-contracts/src/setup/MultisigPluginSetup.sol#L6
aragon/taiko-contracts/src/setup/OptimisticTokenVotingPluginSetup.sol#L21

8.3 PUBLIC FUNCTIONS NOT INVOKED INTERNALLY

// INFORMATIONAL

Description

The smart contracts in-scope include several functions that are declared as `public` but are not invoked internally within the smart contracts. These functions are intended to be called only from external sources.

- src/DelegationWall.sol

```
function register(bytes memory _contentUrl) public {
function getCandidateAddresses() public view returns (address[] memory) {
function candidateCount() public view returns (uint256) {
```

- src/EmergencyMultisig.sol

```
function supportsInterface(bytes4 _interfaceId)
function getProposal(uint256 _proposalId)
function hasApproved(uint256 _proposalId, address _account) public view returns (bool) {
function execute(uint256 _proposalId, bytes memory _metadataUri, IDAO.Action[] calldata _actions) public {
```

- src/EncryptionRegistry.sol

```
function appointAgent(address _newAgent) public {
function setOwnPublicKey(bytes32 _publicKey) public {
function setPublicKey(address _accountOwner, bytes32 _publicKey) public {
function getRegisteredAccounts() public view returns (address[] memory) {
function getAppointedAgent(address _account) public view returns (address) {
function supportsInterface(bytes4 _interfaceId) public view virtual override returns (bool) {
```

- src/Multisig.sol

```
function supportsInterface(bytes4 _interfaceId)
function getProposal(uint256 _proposalId)
function hasApproved(uint256 _proposalId, address _account) public view returns (bool) {
function execute(uint256 _proposalId) public {
```

- src/OptimisticTokenVotingPlugin.sol

```
function supportsInterface(bytes4 _interfaceId)
function hasVetoed(uint256 _proposalId, address _voter) public view returns (bool) {
function getProposal(uint256 _proposalId)
function veto(uint256 _proposalId) public virtual {
function execute(uint256 _proposalId) public virtual {
```

```
function updateOptimisticGovernanceSettings(OptimisticGovernanceSettings calldata _governanceSettings)

function parseProposalId(uint256 _proposalId)
```

- src/SignerList.sol

```
function isListedOrAppointedByListed(address _address) public view returns (bool listedOrAppointedByListed) {

function getListedEncryptionOwnerAtBlock(address _address, uint256 _blockNumber)

function resolveEncryptionAccountAtBlock(address _address, uint256 _blockNumber)

function supportsInterface(bytes4 _interfaceId) public view virtual override returns (bool) {
```

- src/conditions/StandardProposalCondition.sol

```
function supportsInterface(bytes4 _interfaceId) public view virtual override returns (bool) {
```

- src/factory/TaikoDaoFactory.sol

```
function deployOnce() public {

function getSettings() public view returns (DeploymentSettings memory) {

function getDeployment() public view returns (Deployment memory) {
```

BVSS

AO:A/AC:M/AX:M/R:N/S:U/C:N/A:N/I:L/D:N/Y:N (1.1)

Recommendation

To improve code clarity and optimize gas usage, it is recommended to change the visibility of these functions from `public` to `external`.

The `external` keyword is specifically designed for functions that are meant to be called from outside the contract, and it can result in more efficient code execution.

By making this change, you can enhance the readability and performance of the smart contracts.

Remediation Comment

ACKNOWLEDGED: The **Taiko Labs team** made a business decision to acknowledge this finding and not alter the contracts.

References

aragon/taiko-contracts/src/DelegationWall.sol#L8
aragon/taiko-contracts/src/EmergencyMultisig.sol#L19
aragon/taiko-contracts/src/EncryptionRegistry.sol#L14
aragon/taiko-contracts/src/OptimisticTokenVotingPlugin.sol#L24
aragon/taiko-contracts/src/SignerList.sol#L23
aragon/taiko-contracts/src/conditions/StandardProposalCondition.sol#L13

aragon/taiko-contracts/src/factory/TaikoDaoFactory.sol#L26

8.4 LACK OF ACCOUNT REMOVAL MECHANISM IN REGISTRY

// INFORMATIONAL

Description

The `EncryptionRegistry` contract maintains an `accountList` array that stores all registered accounts, but lacks functionality to remove accounts that are no longer active or have been removed as appointed agents. Accounts are added to this `accountList` when they first appoint an agent or set a public key, but there is no corresponding mechanism to remove them when they become inactive or delisted.

As the `accountList` grows indefinitely, operations that iterate through this list (like checking for existing accounts in `appointAgent()` and `_setPublicKey()`) will consume increasingly more gas. This creates a potential DoS vector where the list becomes so large that operations become prohibitively expensive or hit block gas limits.

BVSS

AO:S/AC:L/AX:L/R:N/S:U/C:N/A:N/I:M/D:N/Y:N (1.0)

Recommendation

Implement a removal mechanism that allows accounts to be removed from the `accountList` when they are removed as appointed agents. This could be achieved through a cleanup during agent appointment or key setting operations.

Remediation Comment

SOLVED: The **Taiko Labs team** solved this finding in commit `2628cad` by following the mentioned recommendation.

Remediation Hash

https://github.com/aragon/taiko-contracts/pull/50/commits/2628cad4b3af2bddefb69b5f2d02b04ad29f5eb4

References

aragon/taiko-contracts/src/EncryptionRegistry.sol#L71
aragon/taiko-contracts/src/EncryptionRegistry.sol#L156

8.5 MISSING INPUT VALIDATION

// INFORMATIONAL

Description

Throughout the contracts in scope, there are several instances where input validation is missing. Instances if this issue include:

- The `_governanceERC20Base` and `_governanceWrappedERC20Base` parameters are not checked against the zero address in the `OptimisticTokenVotingPluginSetup` contract constructor.
- The `_minDuration` parameter is not validated to fall within a reasonable range in the `StandardProposalCondition` contract constructor. A long duration could potentially lock the contract for an excessive period.
- The `_dao` parameter in the `initialize()` function of the `SignerList` contract is not validated against the zero address.
- The `_setPublicKey()` function in the `EncryptionRegistry` contract does not validate the `_publicKey` parameter against a valid length.
- In the **OptimisticTokenVotingPlugin** contract, the `votingToken` and `taikoBridge` state variables are updated without checking whether the assigned address is the zero address (`address(0)`).

BVSS

AO:A/AC:H/AX:L/R:N/S:U/C:N/A:N/I:L/D:N/Y:N (0.8)

Recommendation

Implement input validation to ensure that the input parameters are valid and within the expected ranges.

Remediation Comment

PARTIALLY SOLVED: The **Taiko Labs team** partially solved this finding in commit `a8fa1e5` by verifying the `_governanceERC20Base` and `_governanceWrappedERC20Base` parameters against the zero address.

Remediation Hash

https://github.com/aragon/taiko-contracts/pull/50/commits/a8fa1e56c8a0ff1aaf5ff449cc67c1b2f7eea591

References

aragon/taiko-contracts/src/setup/OptimisticTokenVotingPluginSetup.sol#L78
aragon/taiko-contracts/src/conditions/StandardProposalCondition.sol#L30
aragon/taiko-contracts/src/SignerList.sol#L55
aragon/taiko-contracts/src/EncryptionRegistry.sol#L143

aragon/taiko-contracts/src/OptimisticTokenVotingPlugin.sol#L177-L179

8.6 MISSING VISIBILITY MODIFIER

// INFORMATIONAL

Description

In the `StandardProposalCondition` contract, the `dao` and `minDuration` variables are missing the visibility modifier.

By default, variables are set to `internal` visibility. However, It is considered best practice to explicitly specify visibility to enhance clarity and prevent ambiguity. Clearly labeling the visibility of all variables and functions will help in maintaining clear and understandable code.

BVSS

AO:A/AC:L/AX:H/R:N/S:U/C:N/A:N/I:L/D:N/Y:N (0.8)

Recommendation

Explicitly define the visibility of all variables in the contracts to enhance readability and reduce the potential for errors.

Remediation Comment

SOLVED: The **Taiko Labs team** solved this finding in commit `62b1785` by following the mentioned recommendation.

Remediation Hash

https://github.com/aragon/taiko-contracts/pull/50/commits/62b17859716d3482800ae08f6e721fdd5d55c95f

References

aragon/taiko-contracts/src/conditions/StandardProposalCondition.sol#L14-L15

8.7 EMPTY 'REVERT' STATEMENT

// INFORMATIONAL

Description

In the **OptimisticTokenVotingPlugin** contract, there is an empty `revert` statement, as follows:

```
if (_taikoL1 == address(0)) revert();
```

In case the condition is met, the function call to the `initialize()` function will revert without a descriptive error message.

BVSS

AO:A/AC:L/AX:H/R:N/S:U/C:N/A:N/I:L/D:N/Y:N (0.8)

Recommendation

Consider creating a custom error for this specific condition, or reverting with a string (reason) for clarity.

Remediation Comment

SOLVED: The **Taiko Labs team** solved this finding in commit `a8fa1e5` by following the mentioned recommendation.

Remediation Hash

https://github.com/aragon/taiko-contracts/pull/50/commits/a8fa1e56c8a0ff1aaf5ff449cc67c1b2f7eea591

References

aragon/taiko-contracts/src/OptimisticTokenVotingPlugin.sol#L175

8.8 MISSING ADDRESS VALIDATION IN SIGNER MANAGEMENT

// INFORMATIONAL

Description

The `SignerList` contract lacks validation against zero addresses when adding new signers through the `initialize()` and `addSigners()` functions. These functions rely on the internal `_addAddresses()` function inherited from the `Addresslist` contract, which does not perform zero address validation.

The addition of zero addresses could affect quorum calculations, by artificially inflating the number of available signers.

BVSS

AO:S/AC:L/AX:L/R:N/S:U/C:N/A:L/I:L/D:N/Y:N (0.6)

Recommendation

Implement zero address validation to prevent adding invalid addresses as signers.

Remediation Comment

ACKNOWLEDGED: The **Taiko Labs team** made a business decision to acknowledge this finding and not alter the contracts.

References

aragon/taiko-contracts/src/SignerList.sol#L55
aragon/taiko-contracts/src/SignerList.sol#L68

8.9 UNHANDLED RETURN VALUES

// INFORMATIONAL

Description

Several function calls throughout the contracts in scope ignore the return values of the functions they call. Ignoring return values can lead to unexpected behavior and may result in unanticipated outcomes.

Instances of unhandled return values include:

- In src/EmergencyMultisig.sol:

```
proposal_.destinationPlugin.createProposal(
```

- In src/Multisig.sol

```
proposal_.destinationPlugin.createProposal(
```

BVSS

AO:S/AC:L/AX:L/R:N/S:U/C:N/A:N/I:L/D:N/Y:N (0.5)

Recommendation

Ensure that the return values of external calls are handled appropriately throughout the contracts.

Remediation Comment

ACKNOWLEDGED: The **Taiko Labs team** made a business decision to acknowledge this finding and not alter the contracts.

References

aragon/taiko-contracts/src/EmergencyMultisig.sol#L343
aragon/taiko-contracts/src/Multisig.sol#L328

8.10 FLOATING PRAGMA

// INFORMATIONAL

Description

Smart contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

During the analysis of the proposed scope, it was identified that all smart contracts are using a floating pragma, as follows:

```
pragma solidity ^0.8.17;
```

BVSS

AO:A/AC:H/AX:H/R:N/S:U/C:N/A:N/I:L/D:N/Y:N (0.3)

Recommendation

Lock the pragma version and also consider known bugs (https://github.com/ethereum/solidity/releases) for the compiler version that is chosen.

Remediation Comment

ACKNOWLEDGED: The **Taiko Labs team** made a business decision to acknowledge this finding and not alter the contracts.

References

aragon/taiko-contracts/src/conditions/StandardProposalCondition.sol#L3
aragon/taiko-contracts/src/factory/TaikoDaoFactory.sol#L2
aragon/taiko-contracts/src/helpers/proxy.sol#L2
aragon/taiko-contracts/src/setup/OptimisticTokenVotingPluginSetup.sol#L3
aragon/taiko-contracts/src/DelegationWall.sol#L3
aragon/taiko-contracts/src/EmergencyMultisig.sol#L3
aragon/taiko-contracts/src/EncryptionRegistry.sol#L3
aragon/taiko-contracts/src/Multisig.sol#L3
aragon/taiko-contracts/src/OptimisticTokenVotingPlugin.sol#L3
aragon/taiko-contracts/src/SignerList.sol#L3

8.11 REDUNDANT USE OF `THIS` KEYWORD

// INFORMATIONAL

Description

In the `resolveEncryptionAccountAtBlock()` function of the `SignersList` contract, the `this` keyword is used redundantly when calling the `isListedAtBlock()` function.

```
if (this.isListedAtBlock(_appointer, _blockNumber)) {
    // The appointed agent votes
    return (_appointer, _address);
}
```

While using the `this` keyword is not incorrect, it is redundant and may create gas overhead in this context.

BVSS

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

Recommendation

Call the `isListedAtBlock()` function directly without using the `this` keyword.

Remediation Comment

SOLVED: The **Taiko Labs team** solved this finding in commit `eb18176` by following the mentioned recommendation.

Remediation Hash

https://github.com/aragon/taiko-contracts/pull/50/commits/eb18176260c3e200ae79f74ddb233b81790dd601

References

aragon/taiko-contracts/src/SignerList.sol#L145

8.12 MISSING EVENTS

// INFORMATIONAL

Description

In the `deployOnce()` function of the `TaikoFactory` contract, state is modified. However, these changes are not reflected in any event emission. Additionally, in the `execute()` function of the `OptimisticTokenVotingPlugin` no events are emitted as well.

BVSS

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

Recommendation

Emit events for all state changes that occur as a result of administrative functions to facilitate off-chain monitoring of the system.

Remediation Comment

ACKNOWLEDGED: The **Taiko Labs team** made a business decision to acknowledge this finding and not alter the contracts.

References

aragon/taiko-contracts/src/factory/TaikoDaoFactory.sol#L107

8.13 TYPO IN ERROR NAME

// INFORMATIONAL

Description

In the `SignerList` contract there is a typo in an error name, where the word `Registry` is misspelled as `Regitry`.

```
error InvalidEncryptionRegitry(address givenAddress);
```

While this typo does not affect the functionality of the code, it can make the codebase harder to read and understand.

BVSS

AO:S/AC:H/AX:H/R:N/S:U/C:N/A:N/I:L/D:N/Y:N (0.1)

Recommendation

Correct the typo in the error name to improve the readability of the codebase.

Remediation Comment

SOLVED: The **Taiko Labs team** solved this finding in commit `eb18176` by following the mentioned recommendation.

Remediation Hash

https://github.com/aragon/taiko-contracts/pull/50/commits/eb18176260c3e200ae79f74ddb233b81790dd601

References

aragon/taiko-contracts/src/SignerList.sol#L30
