

SALUS SECURITY

JULY 2023



CODE SECURITY ASSESSMENT

INK FINANCE

Overview

Project Summary

- Name: Ink Finance
- Platform: EVM-compatible Chains
- Language: Solidity
- Repository:
 - <https://github.com/Ink-Finance-Inc/v2-governance-core>
 - <https://github.com/Ink-Finance-Inc/v3-economy-core>
- Audit Scope: See [Appendix - 1](#)

Project Dashboard

Application Summary

| | |
|---------|--------------|
| Name | Ink Finance |
| Version | v1 |
| Type | Solidity |
| Dates | July 17 2023 |
| Logs | July 17 2023 |

Vulnerability Summary

| | |
|------------------------------|----|
| Total High-Severity issues | 11 |
| Total Medium-Severity issues | 2 |
| Total Low-Severity issues | 9 |
| Total informational issues | 5 |
| Total | 27 |

Contact

E-mail: support@salusec.io

Risk Level Description

| | |
|----------------------|---|
| High Risk | The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for clients' reputations or serious financial implications for clients and users. |
| Medium Risk | The issue puts a subset of users' sensitive information at risk, would be detrimental to the client's reputation if exploited, or is reasonably likely to lead to a moderate financial impact. |
| Low Risk | The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances. |
| Informational | The issue does not pose an immediate risk, but is relevant to security best practices or defense in depth. |

Content

| | |
|---|-----------|
| Introduction | 4 |
| 1.1 About SALUS | 4 |
| 1.2 Audit Breakdown | 4 |
| 1.3 Disclaimer | 4 |
| Findings | 5 |
| 2.1 Summary of Findings | 5 |
| 2.2 Notable Findings | 7 |
| 1. Malicious contracts could become agents | 7 |
| 2. Incorrect logic in the enableToExecute() modifier | 9 |
| 3. Anyone can unstake someone else's stake | 10 |
| 4. No way to set _ucvManagerEnable in BaseUCV | 11 |
| 5. Malicious users can use InkFund.init() to reset critical states | 12 |
| 6. FactoryManager.upgradeImplement() is not working | 13 |
| 7. Malicious users can use ProposalHandler.init() to reset critical states | 14 |
| 8. Malicious users can use InkFund.startFund() to gain profit | 15 |
| 9. BaseDAO.deployByKey() can be used to deploy a contract with unexpected initData in advance | 16 |
| 10. Anyone can mint tokens | 17 |
| 11. Several critical configurations can be modified by anyone | 18 |
| 12. Any committee can advance the processing flow of the proposal | 19 |
| 13. Conditional statement error | 20 |
| 14. BaseDao allows users who are not Proposers to create proposals | 22 |
| 15. The get functions return a fixed value | 23 |
| 16. Inconsistency between code and documentation for keyId calculation | 24 |
| 17. Use safeTransfer()/safeTransferFrom() instead of transfer()/transferFrom() | 25 |
| 18. Implementation contract could be initialized by everyone | 26 |
| 19. Dead code could lead to the same flowID | 27 |
| 20. The return value is not set correctly for createFund() | 28 |
| 21. Missing validations for input parameters | 29 |
| 22. Incorrect check condition | 31 |
| 2.3 Informational Findings | 33 |
| 23. Inconsistency between codes and comment | 33 |
| 24. Thoughts on domain management | 34 |
| 25. Can use immutable to save gas | 35 |
| 26. Can use local variables to save gas | 36 |
| 27. Redundant code | 37 |
| Appendix | 39 |
| Appendix 1 - Files in Scope | 39 |

Introduction

1.1 About SALUS

At Salus Security, we are in the business of trust.

We are dedicated to tackling the toughest security challenges facing the industry today. By building foundational trust in technology and infrastructure through security, we help clients to lead their respective industries and unlock their full Web3 potential.

Our team of security experts employ industry-leading proof-of-concept (PoC) methodology for demonstrating smart contract vulnerabilities, coupled with advanced red teaming capabilities and a stereoscopic vulnerability detection service, to deliver comprehensive security assessments that allow clients to stay ahead of the curve.

In addition to smart contract audits and red teaming, our Rapid Detection Service for smart contracts aims to make security accessible to all. This high calibre, yet cost-efficient, security tool has been designed to support a wide range of business needs including investment due diligence, security and code quality assessments, and code optimisation.

We are reachable on Telegram (<https://t.me/salusec>), Twitter (https://twitter.com/salus_sec), or Email (support@salusec.io).

1.2 Audit Breakdown

The objective was to evaluate the repository for security-related issues, code quality, and adherence to specifications and best practices. Possible issues we looked for included (but are not limited to):

- Risky external calls
- Integer overflow/underflow
- Transaction-ordering dependence
- Timestamp dependence
- Access control
- Call stack limits and mishandled exceptions
- Number rounding errors
- Centralization of power
- Logical oversights and denial of service
- Business logic specification
- Code clones, functionality duplication

1.3 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release and does not give any warranties on finding all possible security issues with the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues.

Findings

2.1 Summary of Findings

| ID | Title | Severity | Category | Status |
|----|--|----------|----------------------|---------|
| 1 | Malicious contracts could become agents | High | Business Logic | Pending |
| 2 | Incorrect logic in the enableToExecute() modifier | High | Business Logic | Pending |
| 3 | Anyone can unstake someone else's stake | High | Access Control | Pending |
| 4 | No way to set _ucvManagerEnable in BaseUCV | High | Business Logic | Pending |
| 5 | Malicious users can use InkFund.init() to reset critical states | High | Access Control | Pending |
| 6 | FactoryManager.upgradeImplement() is not working | High | Business Logic | Pending |
| 7 | Malicious users can use ProposalHandler.init() to reset critical states | High | Access Control | Pending |
| 8 | Malicious users can use InkFund.startFund() to gain profit | High | Access Control | Pending |
| 9 | BaseDAO.deployByKey() can be used to deploy a contract with unexpected initData in advance | High | Access Control | Pending |
| 10 | Anyone can mint tokens | High | Access Control | Pending |
| 11 | Several critical configurations can be modified by anyone | High | Access Control | Pending |
| 12 | Any committee can advance the processing flow of the proposal | Medium | Data Validation | Pending |
| 13 | Conditional statement error | Medium | Business Logic | Pending |
| 14 | BaseDao allows users who are not Proposers to create proposals | Low | Access Control | Pending |
| 15 | The get functions return a fixed value | Low | Business Logic | Pending |
| 16 | Inconsistency between code and documentation for keyId calculation | Low | Business Logic | Pending |
| 17 | Use safeTransfer()/safeTransferFrom() instead of transfer()/transferFrom() | Low | Risky external calls | Pending |

| | | | | |
|----|--|---------------|------------------|---------|
| 18 | Implementation contract could be initialized by everyone | Low | Business Logic | Pending |
| 19 | Dead code could lead to the same flowID | Low | Business Logic | Pending |
| 20 | The return value is not set correctly for createFund() | Low | Business Logic | Pending |
| 21 | Missing validations for input parameters | Low | Data Validation | Pending |
| 22 | Incorrect check condition | Low | Data Validation | Pending |
| 23 | Inconsistency between codes and comment | Informational | Code Quality | Pending |
| 24 | Thoughts on domain management | Informational | Business Logic | Pending |
| 25 | Can use immutable to save gas | Informational | Gas Optimization | Pending |
| 26 | Can use local variables to save gas | Informational | Gas Optimization | Pending |
| 27 | Redundant code | Informational | Code Quality | Pending |

2.2 Notable Findings

Significant flaws that impact system confidentiality, integrity, or availability are listed below.

1. Malicious contracts could become agents

Severity: High

Category: Business Logic

Target:

- v2-governance-core\contracts\bases\BaseDAO.sol

Description

Agents can be set via the newProposal() function, which can result in malicious contracts becoming agents.

- Proof of Concept

Let's assume that Alice has deployed a malicious contract and wants it to be an agent for the masterDAO.

Alice first calls batchSetKV(aliceAddress, info) in ConfigManager. (The 'info.data' variable should contain the address of the malicious contract)

[v2-governance-core\contracts\utils\ConfigManager.sol:L85-L112](#)

```
function batchSetKV(
    address domain,
    ConfigHelper.KVInfo[] memory keyValueInfos
) external override {
    for (uint256 i = 0; i < keyValueInfos.length; i++) {
        if (
            hasRightToSet(
                domain,
                keyValueInfos[i].keyPrefix,
                keyValueInfos[i].keyName
            )
        ) {
            _domainKeyValues.addKeyValue(domain, keyValueInfos[i]);
        }
        ...
    }
}
```

Alice then gets the configKey for the malicious contract by using [buildConfigKey\(\)](#).

Next, If alice is a proposer, alice can call the newProposal() function and set the proposal.agents parameter to include the configKey for the malicious contract.

[v2-governance-core\contracts\bases\BaseDAO.sol:L261-L292](#)


```
function newProposal(  
    NewProposalInfo calldata proposal,  
    bool commit,  
    address proposer,  
    bytes calldata data  
) public override returns (bytes32 proposalID) {  
    ...  
    _setupAgents(proposalID, proposal.agents, data);  
    ...  
}
```

As a result, the malicious contract deployed by Alice will become the implementation for an agent contract, allowing Alice to exploit critical functions that are modified by `onlyAgent()` in `masterDAO` (e.g. `updateContract()`, `addDuty()`, `removeDuty()`). This could undermine the project.

Recommendation

We recommend setting agents after the proposal has been resolved, not when it was proposed. Therefore, the logic to set agents should be placed within the `decidedProposal()` function.

2. Incorrect logic in the enableToExecute() modifier

Severity: High

Category: Business Logic

Target:

- v2-governance-core\contracts\ucv\PayrollUCV.sol
- v2-governance-core\contracts\bases\BaseUCV.sol

Description

[v2-governance-core\contracts\ucv\PayrollUCV.sol:L44-L50](#)

```
/// @dev make sure msgSender is controller or manager, and manager has to be allow to do
all the operation
modifier enableToExecute() {
    if (
        _msgSender() != _ucvController &&
        (_msgSender() == _ucvManager && _ucvManagerEnable == true)
    ) revert OperateIsNowAllowed();
    _;
}
```

[v2-governance-core\contracts\bases\BaseUCV.sol:L45-L51](#)

```
modifier enableToExecute() {
    if (
        _msgSender() != _ucvController &&
        (_msgSender() == _ucvManager && _ucvManagerEnable == true)
    ) revert OperateIsNowAllowed();
    _;
}
```

In the above codes, the '(_msgSender() == _ucvManager && _ucvManagerEnable == true)' condition should be '!(_msgSender() == _ucvManager && _ucvManagerEnable == true)'.

As a result of this incorrect logic, users that are not _ucvManager can bypass the enableToExecute() modifier and can call [TransferTo\(\)](#) to drain the assets in UCV.

Recommendation

Consider changing the modifier to the following:

```
modifier enableToExecute() {
    if (
        _msgSender() != _ucvController &&
        !(_msgSender() == _ucvManager && _ucvManagerEnable == true)
    ) revert OperateIsNowAllowed();
    _;
}
```

3. Anyone can unstake someone else's stake

Severity: High

Category: Access Control

Target:

- v3-economy-core\contracts\engines\staking\StakingEngine.sol

Description

The unstake() function does not validate the caller, allowing anyone to unstake a stake from any staker.

[v3-economy-core\contracts\engines\staking\StakingEngine.sol:L405-L408](#)

```
function unstake(address basket, bytes32 itemId) external override validBasket(basket) {  
    _unstake(basket, itemId, address(0));  
    emit BasketUnstaked(msg.sender, basket, itemId);  
}
```

Recommendation

Consider validating the msg.sender in unstake().

4. No way to set `_ucvManagerEnable` in BaseUCV

Severity: High

Category: Business Logic

Target:

- v2-governance-core\contracts\bases\BaseUCV.sol

Description

The `_ucvManagerEnable` state variable can only be set by the `_ucvController` when it calls the `enableUCVManager()` function. The `_ucvController` is a private state variable in the BaseUCV contract and can be set only by the `init()` function.

Therefore, if a UCV inherited from BaseUCV does not correctly set the `_ucvController`, then the `_ucvManagerEnable` can not be enabled, resulting in malfunctioning for the `enableToExecute()` modifier.

This is the case for the InkFund contract:

[v2-governance-core/contracts/products/InkFund.sol:L97-L115](#)

```
function init(
    address fundManager,
    address config_,
    bytes calldata data_
) external override returns (bytes memory callbackEvent) {
    ...

    _init(dao_, config_, fundManager, address(0));

    ...
}
```

The same issue also applies to the PayrollUCV contract.

Recommendation

Consider setting `_ucvManagerEnable` to true in the `_init()` function if the controller is set to `address(0)`.

5. Malicious users can use InkFund.init() to reset critical states

Severity: High

Category: Access Control

Target:

- v2-governance-core\contracts\products\InkFund.sol
- v2-governance-core\contracts\ucv\PayrollUCV.sol

Description

The [init\(\)](#) function in the InkFund contract shouldn't be able to be called more than once. However, there is nothing preventing the user from calling the init() function again.

As a result, attackers can call the init() function again to override critical states such as _ucvManager, _fundID, _fund.

The same issue also applies to the [init\(\)](#) function in the PayrollUCV contract.

Recommendation

You should prevent an upgradeable contract from being initialized multiple times.

We recommend you to follow all of the rules for OpenZeppelin's [Writing Upgradeable Contracts](#) article. Specifically, the initializer in the child contract should use the initializer modifier, and the one in the parent contracts should use the onlyInitializing modifier.

6. FactoryManager.upgradeImplement() is not working

Severity: High

Category: Business Logic

Target:

- v2-governance-core\contracts\upgrade\FactoryManager.sol

Description

[v2-governance-core\contracts\upgrade\FactoryManager.sol:L133-L158](#)

```
function upgradeImplement(address instanceProxy, address newImplementation)
    external
{
    ...
    for (uint256 i = 0; i < deployedProxies.length(); i++) {
        address proxyAddress = deployedProxies.at(i);
        InkBeaconProxy(payable(proxyAddress)).upgradeTo(
            address(newBeaconAddress)
        );
        ...
    }
}
```

The upgradeImplement() function in the FactoryManager contract makes external calls to InkBeaconProxy's upgradeTo() function.

[v2-governance-core\contracts\upgrade\InkBeaconProxy.sol:L21-L25](#)

```
contract InkBeaconProxy is BeaconProxy {
    address public immutable _owner;

    ...

    function upgradeTo(address newBeacon) external {
        require(msg.sender == _owner, "Only MetaDAO can upgrade the beacon");
        _setBeacon(newBeacon, "");
    }
}
```

However, the upgradeTo() function requires the caller to be the address of the MasterDAO, which causes the external call from the FactoryManager to fail.

Recommendation

Consider redesigning the logic in upgradeImplement().

7. Malicious users can use ProposalHandler.init() to reset critical states

Severity: High

Category: Access Control

Target:

- v2-governance-core\contracts\proposal\ProposalHandler.sol

Description

The init() function in the ProposalHandler contract shouldn't be able to be called more than once. However, there is nothing preventing the user from calling the init() function again. As a result, attackers can call the init() function again to override the _defaultFlowIDIndex and the _dao state variable.

[v2-governance-core\contracts\proposal\ProposalHandler.sol:L80-L105](#)

```
function init(  
    address dao_,  
    address config_,  
    bytes calldata data_  
) external override returns (bytes memory callbackEvent) {  
    // super.init(config_);  
  
    /// board vote  
    _defaultFlows.push(  
        0x0000000000000000000000000000000000000000000000000000000000000000  
    );  
    /// public vote and board vote  
    _defaultFlows.push(  
        0x0000000000000000000000000000000000000000000000000000000000000001  
    );  
    /// public vote  
    _defaultFlows.push(  
        0x0000000000000000000000000000000000000000000000000000000000000002  
    );  
  
    _defaultFlowIDIndex = abi.decode(data_, (uint256));  
  
    _dao = dao_;  
  
    return callbackEvent;  
}
```

(Notice the super.init() line is commented out)

Recommendation

You should prevent an upgradeable contract from being initialized multiple times.

We recommend you to follow all of the rules for OpenZeppelin's [Writing Upgradeable Contracts](#) article. Specifically, the initializer in the child contract should use the initializer modifier, and the one in the parent contracts should use the onlyInitializing modifier.

8. Malicious users can use InkFund.startFund() to gain profit

Severity: High

Category: Access Control

Target:

- v2-governance-core\contracts\products\funds\FundManager.sol

Description

Although the startFund() function in the FundManager contract requires that the msg.sender is a fund admin, the startFund() function in the InkFund contract lacks proper access control, meaning anyone can call it.

Since the treasury parameter in InkFund.startFund() is a user input address and startFund() sends a fee to this treasury address, attackers can call InkFund.startFund() function with their own address as the treasury input to gain profit that not belong to them.

[v2-governance-core\contracts\products\funds\FundManager.sol:L962-L983](#)

```
function startFund(bytes32 fundID) external override {  
    // authorized  
    require(  
        _isCommitteeOperator(0, msg.sender),  
        "The user is not authorized"  
    );  
  
    ...  
    IFund(_funds[fundID]).startFund(treasuryUCV);  
    ...  
}
```

[v2-governance-core\contracts\products\InkFund.sol:L616-L670](#)

```
function startFund(address treasury) external override {  
    ...  
    takeFixedFee(treasury);  
    ...  
}
```

Recommendation

Consider adding a check for msg.sender in the startFund() function of the InkFund contract.

9. BaseDAO.deployByKey() can be used to deploy a contract with unexpected initData in advance

Severity: High

Category: Access Control

Target:

- v2-governance-core\contracts\bases\BaseDAO.sol

Description

[v2-governance-core\contracts\bases\BaseDAO.sol:L1013-L1046](#)

```
function _deployByFactoryKey(
    bool randomSalt,
    bytes32 typeId,
    bytes32 contractKey,
    bytes memory initData
) internal returns (address deployedAddress) {
    if (
        randomSalt == false &&
        _deployedContractdByKey[contractKey] != address(0)
    ) {
        // deploy only once
        return _deployedContractdByKey[contractKey];
    }
    ...
}
```

If the randomSalt is false, the _deployByFactoryKey() function returns the deployed address for contractKey when it exists.

[v2-governance-core\contracts\bases\BaseDAO.sol:L1048-L1054](#)

```
function deployByKey(
    bytes32 typeId,
    bytes32 contractKey,
    bytes memory initData
) external override returns (address deployedAddress) {
    return _deployByFactoryKey(false, typeId, contractKey, initData);
}
```

However, the deployByKey() function lacks access control. Attackers can call deployByKey() to deploy the contract for contractKey in advance, with unexpected initData, so that the users would reuse this deployed contract. Since the initData is set by the attacker, the interaction with this contract is risky.

Recommendation

Consider adding proper access control to deployByKey() function. Additionally, consider deploying a new contract instead of returning the deployed one if the initData is different in _deployByFactoryKey().

10. Anyone can mint tokens

Severity: High

Category: Access Control

Target:

- v2-governance-core\contracts\tokens\InkERC721.sol
- v2-governance-core\contracts\tokens\InkERC20.sol

Description

1.Anyone can call the mintTo() function to mint InkERC20 token.

[v2-governance-core\contracts\tokens\InkERC20.sol:L37-L39](#)

```
function mintTo(address target, uint256 amount) public virtual {  
    _mint(target, amount);  
}
```

2.Anyone can call the mint() function to mint InkERC721 token.

[v2-governance-core\contracts\tokens\InkERC721.sol:L19-L21](#)

```
function mint(address account, uint256 tokenId) public {  
    _mint(account, tokenId);  
}
```

Recommendation

Consider adding proper access control to public mint functions.

11. Several critical configurations can be modified by anyone

Severity: High

Category: Access Control

Target:

- v2-governance-core\contracts\bases\BaseDAO.sol
- v2-governance-core\contracts\cores\KYCVerifyManager.sol

Description

1.The setFlowStep() function lacks access control, this function can be used to set the processing step for any type of flowId.

[v2-governance-core\contracts\bases\BaseDAO.sol:L1060-L1062](#)

```
function setFlowStep(FlowInfo memory flow) external override {  
    _setFlowStep(flow);  
}
```

2.Anyone can update _identityManager by calling updateIdentityManager().

[v2-governance-core\contracts\cores\KYCVerifyManager.sol:L47-L49](#)

```
function updateIdentityManager(address identityManager_) external {  
    _identityManager = identityManager_;  
}
```

Recommendation

Consider adding proper access control to setter functions for critical states.

12. Any committee can advance the processing flow of the proposal

Severity: Medium

Category: Access Control

Target:

- v2-governance-core\contracts\bases\BaseDAO.sol

Description

In the `decideProposal()` function, any committee can advance a proposal process one step according to the defined flow, but there is no verification of whether the committee is the one that should be handling the proposal at the current stage.

This could potentially lead to the Public committee bypassing the Board's resolution process by making multiple calls to the `tallyVotes()` function.

[v2-governance-core\contracts\bases\BaseDAO.sol:L776-L787](#)

```
function decideProposal(
    bytes32 proposalID,
    bool agree,
    bytes calldata data
) external override onlyCommittee {
    IProposalHandler(_proposalHandlerAddress).decideProposal(
        proposalID,
        agree,
        data
    );
    // _decideProposal(proposalID, msg.sender, agree, data);
}
```

Recommendation

Consider verifying the committee.

For example, pass the caller committee to `ProposalHandler._decideProposal()` and check whether it's the intended committee.

```
function _decideProposal(
    bytes32 proposalID,
    address committee,
    bool agree,
    bytes memory data
) internal {
    ProposalProgress storage info = _proposalInfo[proposalID];
    require(info.proposalID == proposalID, "proposal err");
    require(committee == info.nextCommittee, "Committee that is not under processing");
    ...
}
```

13. Conditional statement error

Severity: Medium

Category: Business Logic

Target:

- v3-economy-core\contracts\engines\staking\StakingBasket.sol

Description

The claimRewards() function is not intended to claim before item.endDate.

However, if the current time is between a day before the endDate, the function will allow users to claim because the remainDays will be zero due to loss of precision when calculating the remainDays.

[v3-economy-core\contracts\engines\staking\StakingBasket.sol:L182-L219](#)

```
function claimRewards(
    bytes32 itemId,
    uint256 accruingPeriod,
    bool penaltyFlag,
    uint256 rewardRatio
)
    external
    override
    allowedStakingEngine
    validStakingItem(itemId)
    returns (
        address staker,
        uint256 rewards,
        uint256 remainDays
    )
{
    ...
    if (item.endDate <= block.timestamp) {
        // after staking item matures,
        if (accruingPeriod > 0) {
            uint256 passDays = (block.timestamp - item.beginDate) / _oneDayInSeconds;
            require(passDays >= accruingPeriod, "[STBasket]: Failed by accruingPeriod");
        }
    } else {
        remainDays = (item.endDate - block.timestamp) / _oneDayInSeconds;
    }
    if (!penaltyFlag || remainDays == 0) {
        uint256 rewardsForItem = item.effectiveStakingValue * (rewardRatio -
item.rewardRatio);
        if (rewardsForItem > item.settledRewards) {
            rewards = rewardsForItem - item.settledRewards;
            item.settledRewards = rewardsForItem;
        }
    }
}
```

Recommendation

Consider adding a boolean variable instead of remainDays to check if the current time has passed the endDate.

14. BaseDao allows users who are not Proposers to create proposals

Severity: Low

Category: Access Control

Target:

- v2-governance-core\contracts\bases\BaseDAO.sol

Description

[v2-governance-core\contracts\committee\TheBoard.sol:L38-L57](#)

```
function newProposal(  
    NewProposalInfo calldata proposal,  
    bool commit,  
    bytes calldata data  
) external override returns (bytes32 proposalID) {  
    // valid have dutyID to create the proposal  
    if (!_hasDutyToOperate(DutyID.PROPOSER, _msgSender())) {  
        revert YouDoNotHaveDutyToOperate();  
    }  
    ...  
}
```

[v2-governance-core\contracts\bases\BaseDAO.sol:L261-L292](#)

```
function newProposal(  
    NewProposalInfo calldata proposal,  
    bool commit,  
    address proposer,  
    bytes calldata data  
) public override returns (bytes32 proposalID) {  
    ...  
}
```

While the newProposal() function in the TheBoard contract requires the caller to be a PROPOSER, the newProposal() in BaseDAO lacks proper access control. As a result, non-proposer can use BaseDAO.newProposal() to create a proposal.

The complete invocation chain for creating a new proposal is: committee => dao => proposalHandler. Therefore, the newProposal function in dao should require the caller to be the committee.

Recommendation

Consider adding the onlyCommittee modifier to the newProposal() function in BaseDao.

15. The get functions return a fixed value

Severity: Low

Category: Business Logic

Target:

- v3-economy-core\contracts\dao\DaoGovernance.sol

Description

1. `getProposalStatus()` always returns `uint256(ProposalStatus.AGREE)`.

[v3-economy-core\contracts\dao\DaoGovernance.sol:L50-L53](#)

```
function getProposalStatus(bytes32 proposalId) external pure override returns
(uint256) {
    require(proposalId != bytes32(0), "Invalid Id");
    return uint256(ProposalStatus.AGREE);
}
```

2. This function always returns true even if `daoUser` doesn't have any duty.

[v3-economy-core\contracts\dao\DaoGovernance.sol:L56-L58](#)

```
function hasAnyDuty(address daoUser) external pure override returns (bool) {
    return true;
}
```

3. This function always returns true for non-zero `proposalIds` that are not in `deletedIds`.

[v3-economy-core\contracts\dao\DaoGovernance.sol:L96-L100](#)

```
function isValidProposalId(bytes32 proposalId) external view override returns
(bool) {
    if (proposalId == bytes32(0) || _deletedIds[proposalId]) return false;

    return true;
}
```

4. This function always returns true for non-zero `dutyIds` that are not in `deletedIds`.

[v3-economy-core\contracts\dao\DaoGovernance.sol:L102-L106](#)

```
function isValidDutyId(bytes32 dutyId) external view override returns (bool) {
    if (dutyId == bytes32(0) || _deletedIds[dutyId]) return false;

    return true;
}
```

Recommendation

Consider returning the correct values.

16. Inconsistency between code and documentation for keyId calculation

Severity: Low

Category: Business Logic

Target:

- v2-governance-core\contracts\utils\ConfigManager.sol

Description

According to the comment below, the keyId should be `keccak256(keccak256(<prefix>) + keccak256(keyName))`.

[v2-governance-core\contracts\utils\ConfigManager.sol:L13](#)

```
/// 3. keyID = keccak256(keccak256(<prefix>) + keccak256(keyName))
```

However, it is implemented as `keyID = keccak256(prefix + keccak256(keyName))`.

[v2-governance-core\contracts\utils\ConfigHelper.sol:L57-L69](#)

```
function packKeyID(string memory prefix, string memory keyName)
    internal
    pure
    returns (bytes32 keyID)
{
    string memory actualPrefix = getPrefix(prefix);
    keyID = keccak256(
        abi.encodePacked(
            abi.encodePacked(actualPrefix),
            keccak256(abi.encodePacked(keyName))
        )
    );
}
```

Recommendation

Consider updating the code to ensure it is consistent with the documentation.

17. Use `safeTransfer()/safeTransferFrom()` instead of `transfer()/transferFrom()`

Severity: Low

Category: Risky external calls

Target:

- v3-economy-core\contracts\engines\EmissionPool.sol
- v2-governance-core\contracts\bases\BaseUCV.sol

Description

Tokens not compliant with the ERC20 specification could return false from the transfer function call to indicate the transfer fails, while the calling contract would not notice the failure if the return value is not checked. Checking the return value is a requirement, as written in the [EIP-20](#) specification:

Callers MUST handle false from returns (bool success). Callers MUST NOT assume that false is never returned!

[v3-economy-core\contracts\engines\EmissionPool.sol:L117](#)

```
IERC20(token).transfer(to, amount);
```

[v2-governance-core\contracts\bases\BaseUCV.sol:L110](#)

```
IERC20(token).transfer(to, _value);
```

[v2-governance-core\contracts\bases\BaseUCV.sol:L167](#)

```
IERC20(token).transferFrom(msg.sender, address(this), amount);
```

Recommendation

Consider using the [SafeERC20](#) library implementation from OpenZeppelin and call `safeTransfer` or `safeTransferFrom` when transferring ERC20 tokens.

18. Implementation contract could be initialized by everyone

Severity: Low

Category: Business Logic

Target:

- All upgradeable contracts

Description

According to [OpenZeppelin](#), the implementation contract should not be left uninitialized.

An uninitialized implementation contract can be taken over by an attacker, which may impact the proxy. For the upgradeable contracts in the Ink codebase, there is nothing preventing the attacker from calling the initializers in the implementation contracts.

Recommendation

To prevent the implementation contract from being used, consider invoking the `_disableInitializers` function in the constructor of the implementation contract to automatically lock it when it is deployed.

19. Dead code could lead to the same flowID

Severity: Low

Category: Business Logic

Target:

- v2-governance-core\contracts\products\proposal\ProposalHandler.sol

Description

[v2-governance-core\contracts\products\proposal\ProposalHandler.sol:L136-L185](#)

```
function _getProposalFlow(bytes32 proposalID)
    internal
    view
    returns (bytes32 flowID)
{
    bytes32 proposalFlowID =
    0x0000000000000000000000000000000000000000000000000000000000000004;

    ...

    if (
        proposalFlowID ==
        0x0000000000000000000000000000000000000000000000000000000000000004
    ) {
        flowID = _defaultFlows[_defaultFlowIDIndex];
    } else {
        bool support = false;
        for (
            uint256 i = _defaultFlowIDIndex;
            i < _defaultFlows.length;
            i++
        ) {
            if (_defaultFlows[i] == proposalFlowID) {
                support = true;
                flowID = proposalFlowID;
                break;
            }
        }
    }
}
```

The value of proposalFlowID is not changed before the if clause, hence the code in the else branch is dead code.

Recommendation

Consider rewriting the logic of conditional branches.

20. The return value is not set correctly for createFund()

Severity: Low

Category: Business Logic

Target:

- v2-governance-core\contracts\products\funds\FundManager.sol

Description

The ucvAddress is defined as the returned value for createFund() but not correctly set by the function.

[v2-governance-core\contracts\products\funds\FundManager.sol:L902-937](#)

```
function createFund(NewFundInfo memory fundInfo)
    external
    override
    returns (address ucvAddress)
{
    ...
    // valid treasury exist
    address fundAddress = _deployByFactoryKey(
        FactoryKeyTypeID.UCV_TYPE_ID,
        fundInfo.fundDeployKey,
        initData
    );
    ...
}
```

Recommendation

Consider returning the fundAddress correctly.

21. Missing validations for input parameters

Severity: Low

Category: Data Validation

Target:

- v2-governance-core\contracts\products\proposal\ProposalHandler.sol

Description

The following functions lack proper validations for input parameters.

1. [v2-governance-core\contracts\products\proposal\ProposalHandler.sol:L80-L105](#)

```
function init(  
    address dao_,  
    address config_,  
    bytes calldata data_  
) external override returns (bytes memory callbackEvent) {  
    // super.init(config_);  
  
    /// board vote  
    _defaultFlows.push(  
        0x0000000000000000000000000000000000000000000000000000000000000000  
    );  
    /// public vote and board vote  
    _defaultFlows.push(  
        0x0000000000000000000000000000000000000000000000000000000000000001  
    );  
    /// public vote  
    _defaultFlows.push(  
        0x0000000000000000000000000000000000000000000000000000000000000002  
    );  
  
    _defaultFlowIDIndex = abi.decode(data_, (uint256));  
  
    _dao = dao_;  
  
    return callbackEvent;  
}
```

The init() function should add the following require statement.

```
require(_defaultFlowIDIndex < _defaultFlows.length && _dao != address(0));
```

2. [v2-governance-core\contracts\products\proposal\ProposalHandler.sol:L529](#)

When agents.length == 0, the function should revert but not.

```
function _newProposal(  
    NewProposalInfo memory proposal,  
    bool commit,  
    address proposer,  
    bytes memory data  
) internal returns (bytes32 proposalID) {  
    bytes32[] memory agents = proposal.agents;  
    if (agents.length == 0) {  
        // error  
    }  
}
```

```
}  
...  
}
```

Recommendation

Consider adding proper validation for input parameters.

22. Incorrect check condition

Severity: Low

Category: Data Validation

Target:

- v2-governance-core\contracts\bases\BaseDAO.sol

Description

The conditional statements and error messages are inconsistent. It is recommended to separate the multiple conditional checks to precisely identify the issues and facilitate accurate troubleshooting.

Additionally, it seems the condition 'IAgent(existAgents).isExecuted() == true' should be 'IAgent(existAgents).isExecuted() == false'.

[v2-governance-core\contracts\bases\BaseDAO.sol:L938-L944](#)

```
function _setupAgents(
    bytes32 proposalID,
    bytes32[] memory agents,
    bytes memory initData
) internal {
    for (uint256 i = 0; i < agents.length; i++) {
        if (
            agents[i] !=
            0x0000000000000000000000000000000000000000000000000000000000000000
        ) {
            address existAgents = _agents[agents[i]];
            if (
                existAgents != address(0) &&
                IAgent(existAgents).isExecuted() == true &&
                IAgent(existAgents).isUniqueInDAO() == true
            ) {
                revert AgentCanBeCreatedOnlyOnceInDAO(agents[i]);
            }
            ...
        }
    }
}
```

Recommendation

Consider writing condition checks separately, and use appropriate custom errors for each one.

Example pattern:

```
if (failCondition1) {
    revert failReason1();
}
if (failCondition2) {
```



```
    revert failReason2();  
}  
...
```

2.3 Informational Findings

23. Inconsistency between codes and comment

Severity: Informational

Category: Code Quality

Target:

- v2-governance-core\contracts\tokens\InkFundCertificateToken.sol

Description

mintTo function is expected to revert but it's not reverted.

[v2-governance-core\contracts\tokens\InkFundCertificateToken.sol:L23-L25](#)

```
function mintTo(address target, uint256 amount) public override {  
    // revert NotAllowed  
}
```

Recommendation

Consider adding a revert statement.

24. Thoughts on domain management

Severity: Informational

Category: Business Logic

Target:

- v2-governance-core\contracts\utils\ConfigManager.sol

Description

In the ConfigManager contract, for keys that belong to a domain, only users that have [rights](#) for the domain can set the values for the keys by using [batchSetKV\(\)](#).

While the setting part has the above design for domain management, the getting part lacks such design. If a function needs to check if a user has the rights to get or use the value from a key, or if the key is valid for the user, it can't do so since we can't get the domain from the key, and we need that domain information to verify the user.

Recommendation

Consider adding a design that verifies whether a key is valid for a user or if a user has the right to use the key.

25. Can use immutable to save gas

Severity: Informational

Category: Gas Optimization

Target:

- v2-governance-core\contracts\tokens\InkERC721.sol
- v3-economy-core\contracts\dao\DaoGovernance.sol

Description

The following variables could be set immutable.

[v2-governance-core\contracts\tokens\InkERC721.sol:L9](#)

```
address public creator;
```

v3-economy-core\contracts\dao\DaoGovernance.sol:[L12](#), [L14](#), [L15](#), [L17](#)

```
address private _daoCreator;  
address private _treasury;  
bool private _penalty;  
address public engineV1Factory;
```

Recommendation

Consider defining variables set in the constructor and not changed after deployment as immutable.

26. Can use local variables to save gas

Severity: Informational

Category: Gas Optimization

Target:

- v2-governance-core\contracts\products\funds\FundManager.sol
- v3-economy-core\contracts\dao\DaoGovernance.sol
- v3-economy-core\contracts\engines\staking\StakingEngine.sol

Description

1. `_dao` could be replaced with the local variable `dao_`.

[v2-governance-core\contracts\products\funds\FundManager.sol:L168](#)

```
_factoryManager = IDAO(dao).getDAODeployFactory();
```

2. `stakingEngine` could be replaced with the local variable `_stakingEngine`.

[v3-economy-core\contracts\dao\DaoGovernance.sol:L34-L36](#)

```
if (stakingEngine != address(0)) {  
    pledgeEngine = IStakingEngine(stakingEngine).getPledgeEngine();  
}
```

3. `_baskets.length` is read every time during iteration. Could have a local variable for the length.

[v3-economy-core\contracts\engines\staking\StakingEngine.sol:L675, L726](#)

```
for (uint256 i = 0; i < _baskets.length; i++) {
```

4. `_principalTokens.length` is read every time during iteration. Could have a local variable for the length.

[v3-economy-core\contracts\engines\staking\StakingEngine.sol:L627](#)

```
for (uint256 i = 0; i < _principalTokens.length; i++) {
```

Recommendation

Consider reading the same value from a local variable instead of storage to save gas.

27. Redundant code

Severity: Informational

Category: Code Quality

Target:

- v2-governance-core\contracts\committee\TheBoard.sol
- v2-governance-core\contracts\committee\TreasuryCommittee.sol
- v2-governance-core\contracts\base\BaseVerify.sol
- v2-governance-core\contracts\base\BaseUCV.sol
- v2-governance-core\contracts\base\BaseDAO.sol
- v2-governance-core\contracts\base\BaseCommittee.sol
- v2-governance-core\contracts\agents\TreasuryManagerAgent.sol
- v2-governance-core\contracts\cores\KYCVerifyManager.sol
- v2-governance-core\contracts\products\funds\FundManager.sol
- v2-governance-core\contracts\products\EscrowManager.sol
- v2-governance-core\contracts\products\InkFund.sol
- v2-governance-core\contracts\proposal\ProposalHandler.sol
- v2-governance-core\contracts\ucv\InvestmentUCV.sol
- v2-governance-core\contracts\agents\InvestmentManagementSetupAgent.sol
- v3-economy-core\contracts\engines\pools\BasketPool.sol
- v3-economy-core\contracts\engines\staking\StakingBasket.sol
- v3-economy-core\contracts\engines\staking\StakingEngine.sol

Description

Commented-out code, code for testing purposes (e.g. import "hardhat/console.sol", console.log...) and unused code (e.g. unused custom errors, unused structs) should be removed before deploying the contract to mainnet.

We have identified the following redundant codes.

- v2-governance-core\contracts\committee\TheBoard.sol:[L8](#), [L30-L33](#)
- v2-governance-core\contracts\committee\TreasuryCommittee.sol:[L22-L26](#), [L38-L62](#), [L72](#)
- v2-governance-core\contracts\base\BaseVerify.sol:[L25-L28](#)
- v2-governance-core\contracts\base\BaseUCV.sol:[L56](#)
- v2-governance-core\contracts\base\BaseDAO.sol:[L164-L170](#), [L225-L227](#), [L279](#), [L462](#), [L476-L477](#), [L484](#), [L491-L492](#), [L503](#), [L605-L611](#), [L786](#), [L913-L922](#), [L955 - L956](#)
- v2-governance-core\contracts\base\BaseCommittee.sol:[L264-L265](#), [L269-L278](#), [L372](#), [L461-L468](#)
- v2-governance-core\contracts\agents\TreasuryManagerAgent.sol:[L90](#), [L100](#), [L103](#), [L108-L110](#), [L115-L116](#), [L180-L254](#)
- v2-governance-core\contracts\cores\KYCVerifyManager.sol:[L37](#), [L39](#), [L59](#), [L68-L69](#), [L99-L100](#), [L104-L107](#)
- v2-governance-core\contracts\products\funds\FundsManager.sol:[L25-L26](#), [L64-L70](#), [L406-L418](#), [L483-L503](#), [L542-L550](#), [L598-L604](#), [L764-L771](#), [L784-L795](#), [L819](#), [L926](#),

- [L939-L947](#), [L982](#), [L995-L1032](#), [L1079-L1089](#), [L1129-L1137](#), [L1148-L1151](#), [L1192](#), [L1194](#)
- v2-governance-core\contracts\products\EscrowManager.sol:[L101-L103](#), [L220-L223](#), [L289](#)
 - v2-governance-core\contracts\products\InkFund.sol:[L18](#), [L106](#), [L133](#), [L281](#), [L304](#), [L306](#), [L341](#), [L351](#), [L448](#), [L520](#), [L565-L567](#), [L619](#), [L710](#), [L783-L786](#), [L808](#), [L820-L821](#), [L888](#)
 - v2-governance-core\contracts\proposal\ProposalHandler.sol:[L418-L432](#), [L453-L455](#), [L463](#)
 - [v2-governance-core\contracts\lucv\InvestmentUCV.sol:all](#)
 - v2-governance-core\contracts\agents\InvestmentManagementSetupAgent.sol:[L9](#), [L95-L97](#), [L106](#)
 - v3-economy-core\contracts\engines\pools\BasketPool.sol:[L29-L30](#), [L101](#)
 - v3-economy-core\contracts\engines\staking\StakingBasket.sol:[L13](#), [L26-L27](#)
 - v3-economy-core\contracts\engines\staking\StakingEngine.sol:[L16](#), [L104](#), [L1116](#)

Recommendation

Consider removing the redundant codes.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit [9534fc9](#) of the v2-governance-core repo:

| File | SHA-1 hash |
|---|--|
| contracts\agents\InvestmentManagementSetupAgent.sol | f9cec6af31f3a45e1f537f1b752b79c63f03c806 |
| contracts/agents/TreasuryManagerAgent.sol | 6b9b7d2ceb1306c6a9eff51f3c8a91503d228341 |
| contracts/bases/BaseAgent.sol | ae75ceab0d0fb93cfc2ee019c74f118c777d3535 |
| contracts/bases/BaseCommittee.sol | c38940eb21dac4c98d55ab107333675e3cb71930 |
| contracts/bases/BaseDAO.sol | 145a0223c1416d5e48d282787197d5c16ab8101e |
| contracts/bases/BaseUCV.sol | 91ded64bb375efd9a4fada4a5bd47bddf5eb3bba |
| contracts/bases/BaseUCVManager.sol | 033ff1ac7d3fff889605aa7c18a1f3e1ba8d6a87 |
| contracts/bases/BaseVerify.sol | b15f0e0ffb6ac5bc4a126554f0682f3b2770cff1 |
| contracts/committee/InvestmentCommittee.sol | c93b45489adc3961a69a0e1cac388449518a98cc |
| contracts/committee/TheBoard.sol | 7b0eec6c1f676c1785c93502dddcefaebcd1b853 |
| contracts/committee/ThePublic.sol | 9a3bd90ca772df8c1972de4c22971f3860481e20 |
| contracts/committee/TreasuryCommittee.sol | 25753f88fdd483d8fcdc5fa35a4b48c146b8f243 |
| contracts/cores/IdentityManager.sol | d4fff05872d6357c4e82d8cfb5441a8f6c4cc853 |
| contracts/cores/KYCVerifyManager.sol | b2fccaabef29ee1fd4f3c010a0ab0f35fbc0f485 |
| contracts/daos/MasterDAO.sol | 806d60ff76dc878a5acd5e529b4c8cb916384a65 |
| contracts/products/EscrowManager.sol | 31ed96af336a0c3929645e030ee0be285eae1745 |
| contracts/products/InkEnvelopeFactory.sol | 7160ae6e94c5889583519e08666f9f643162698f |
| contracts/products/InkFund.sol | 60043cc1eb6e9513cd29eb491ba61b010a5e6687 |
| contracts/products/funds/FundManager.sol | 1ee1bc038411789ff44a09d317417b45c54528f7 |
| contracts/products/tokens/WrappedERC1155.sol | c78c14b3cce501162c25731b24ddf24944591875 |
| contracts/products/tokens/WrappedERC20.sol | fe4f4f8447187cf70d8425a643e9dad42fc5f5f0 |
| contracts/products/tokens/WrappedERC721.sol | ea6240989cecba3a36484e566799e4bee963dfd2 |
| contracts/products/tokens/WrappedToken.sol | efeaaea7b72cf8d38ce2fc061ea80dceeea782b5 |

| | |
|--|--|
| contracts/proposal/ProposalHandler.sol | 652197abd602694a66a81a54afad14966cd98d9c |
| contracts/tokens/InkBadgeERC20.sol | 4c1d60328de5a271c42694db59e126a95ce48b78 |
| contracts/tokens/InkERC20.sol | 1cb45fa76e8ca6e1aca6dac2d5a3d7c3058f7d3b |
| contracts/tokens/InkERC721.sol | eda9ac18b3c00a74d96ceaa7cb9092711118542d |
| contracts/tokens/InkFundCertificateToken.sol | 7fcc4d0ea9f1ca0ce8140306f50b310245b6c1c6 |
| contracts/ucv/InvestmentUCV.sol | 0de59a513be7bcbcac542cfe3216f3bd6f9ed964 |
| contracts/ucv/InvestmentUCVManager.sol | efabf30d9f877bffe5f69b5fab3c46d58e88f447 |
| contracts/ucv/PayrollUCV.sol | 902a8696cfb0477157afb236f307a16a11e01512 |
| contracts/ucv/PayrollUCVManager.sol | f00804554d6d3bcca155a79a694182e0b739a6bd |
| contracts/ucv/TreasuryIncomeManager.sol | 13fe1db781679f4c140822b8752fd4fc64fc1ed5 |
| contracts/upgrade/FactoryManager.sol | 40ed55b6e3abc785fcea72c1d68360942c5c7dc0 |
| contracts/upgrade/InkBeaconProxy.sol | 21ebd9292e3345705111c6d9e1b14f7122d4c312 |
| contracts/utils/BytesUtils.sol | ce23b72d9db0830ef5dea3c66b6d26b6c6da2ad4 |
| contracts/utils/ConfigHelper.sol | 4e85352fa7f40df4ab26a35808de75d9f06adb0c |
| contracts/utils/ConfigManager.sol | 9460ac592d25c05c163334976f204404410c18f7 |
| contracts/utils/TransferHelper.sol | 4f5612cf785dc930888d8fdb18d458cdae1dd2de |

and the following files in commit [a779da4](#) of the v3-economy-core repo:

| File | SHA-1 hash |
|--|--|
| contracts/config/EconomyAddressesProvider.sol | 7502191c54570605f03448f101e263abe1cc65b1 |
| contracts/dao/DaoGovernance.sol | c0154eb1980012900fd39496bcee4913f67593dd |
| contracts/dao/IDaoGovernance.sol | 99bb51605387f6974032ad5bf2cd3d531d7d8e63 |
| contracts/interfaces/IBasketPool.sol | 07bd02f5f22d0604587e07bff4b0ca3f3740b166 |
| contracts/interfaces/IBufferPool.sol | 4bfdab74917b0ff1b828d4f16cb4bc9985a38510 |
| contracts/interfaces/IEconomyAddressesProvider.sol | a8a3fe3e131d24e4944d302a26295b9fa20a0497 |
| contracts/interfaces/IEconomyEngineV1Factory.sol | 10d1c3078ba7c4dbd719911fc4c7fa82329ab929 |
| contracts/interfaces/IEmissionPool.sol | 8d01b1a78623719f38c1a7b27f24afeec13bd682 |
| contracts/interfaces/IPledgeEngine.sol | c1a4203375a7b6139501bf6fae7d08674cd21016 |

| | |
|---|--|
| contracts/interfaces/IReclaimPool.sol | 27a4476c209f5cdb0dafdb07511b00d79a20d86e |
| contracts/interfaces/ISponsorEngine.sol | eac3bc1bfc170f68df1e4785f6db1e945d61b52b |
| contracts/interfaces/IStakingBasket.sol | 7e67162363bc610eb1eea0616b4f1308a4ec85b6 |
| contracts/interfaces/IStakingEngine.sol | 830388bfe29a92367fb8c90ad25ebb0e3745c715 |
| contracts/libraries/LDatetime.sol | ba70f707c033388419491e1adeea58ad7fe68b2d |
| contracts/libraries/LPledgeEngine.sol | 46e9d271852f9e08e258514b7538ca6d43d1bf70 |
| contracts/libraries/LStakingBasket.sol | 2ce1af2809bdb1f5afd81c367b379214ebb2a8d7 |
| contracts/libraries/LStakingEngine.sol | 7afb695d61e8c328c94c06d0684c2379ce1498f |
| contracts/libraries/LStakingItem.sol | 3e039a1efa9ae0d0b4b89a572dbc8ac518032222 |
| contracts/libraries/PRBMath.sol | 4d5e61cba5a6750063e765dc1806e3a766a93c5f |
| contracts/libraries/PRBMathUD60x18.sol | 8bf9d6b6f975dc9cc776bd7108ecacdb0c634b41 |
| contracts/engines/pledge/PledgeEngine.sol | 2d57e3332cad337bb9d6a7dd48c35275c9ae9f37 |
| contracts/engines/pools/BasketPool.sol | 7eebc02e605e9d836fa93295a0821d32bd92d68b |
| contracts/engines/pools/EmissionPool.sol | f8914abfd012ebb597c519f07cf530319de30d6e |
| contracts/engines/sponsor/SponsorEngine.sol | 8a90a9263078b6e4cf6eedeebef6b1b38a466059 |
| contracts/engines/staking/StakingBasket.sol | c75b44510c1ecdd727ee5ef9e189ae161d12cc77 |
| contracts/engines/staking/StakingEngine.sol | 6f86a18d2eb17a2d2572e281f6d44181dc5fc572 |
| contracts/engines/trust/TrustEngine.sol | b98e90a96ec719db93b11b2f2a2189fdaeec1deb |