

SALUS SECURITY

DEC 2023



# CODE SECURITY ASSESSMENT

INK FINANCE

# Overview

## Project Summary

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

## Project Dashboard

### Application Summary

Name	Ink Finance - Incremental Audit
Version	v2
Type	Solidity
Dates	Dec 12 2023
Logs	Dec 12 2023; Dec 14 2023

### Vulnerability Summary

Total High-Severity issues	2
Total Medium-Severity issues	2
Total Low-Severity issues	0
Total informational issues	0
Total	4

## Contact

E-mail: [support@salusec.io](mailto:support@salusec.io)

## Risk Level Description

<b>High Risk</b>	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.
<b>Medium Risk</b>	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.
<b>Low Risk</b>	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.
<b>Informational</b>	The issue does not pose an immediate risk, but is relevant to security best practices or defense in depth.

# Content

<b>Introduction</b>	<b>4</b>
1.1 About SALUS	4
1.2 Audit Breakdown	4
1.3 Disclaimer	4
<b>Findings</b>	<b>5</b>
2.1 Summary of Findings	5
2.2 Notable Findings	6
1. Unable to unpledge after the proposal has been denied	6
2. InkBadge token being transferable can lead to a repeat voting attack	8
3. Proposer can skip the public vote process	10
4. User can vote on non-existent proposals	12
<b>Appendix</b>	<b>13</b>
Appendix 1 - Files in Scope	13

# 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](https://twitter.com/salus_sec)), or Email ([support@salusec.io](mailto: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	Unable to unpledge after the proposal has been denied	High	Business Logic	Pending
2	InkBadge token being transferable can lead to a repeat voting attack	High	Business Logic	Acknowledged
3	Proposer can skip the public vote process	Medium	Business Logic	Acknowledged
4	User can vote on non-existent proposals	Medium	Business Logic	Pending

## 2.2 Notable Findings

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

### 1. Unable to unpledge after the proposal has been denied

Severity: High

Category: Business Logic

Target:

- v3-economy-core/contracts/engines/pledge/PledgeEngine.sol

### Description

If the DAO sets up a govToken, the user will use the pledged tokens to vote. Tokens already used for voting will be locked and cannot be unstaked.

v3-economy-core/contracts/engines/pledge/PledgeEngine.sol:L171-L182

```
function unpledge(address daoUser, bytes32 proposalId)
    external
    override
    onlyDaoContractPledgedByUser(daoUser)
    returns (uint256)
{
    if (IDaoGovernance(msg.sender).getProposalStatus(proposalId) != 1) {
        revert INK_ERROR(2049); /// PE: DAO didn't allow to unpledge
    }

    return _unpledge(msg.sender, daoUser, proposalId);
}

// v2-governance-core\contracts\proposal\ProposalHandler.sol
function getProposalStatus(bytes32 proposalID)
    external
    view
    override
    returns (uint256 status)
{
    ProposalSummary memory summary = _getProposalSummary(proposalID);
    if (summary.status == ProposalStatus.PENDING) {
        return 0;
    }
    if (summary.status == ProposalStatus.AGREE) {
        return 1;
    }
    if (summary.status == ProposalStatus.DENY) {
        return 2;
    }
}
```

However, in the highlighted code above, if the proposal status is not 1 (meaning the proposal is not agreed upon), the unpledge() function will revert. Consequently, when a proposal is denied, users are unable to unpledge their stakes and therefore cannot unstake.

### Recommendation

It is recommended to use the correct proposal status check.

For example, the if condition can be changed to the following:

```
if (IDaoGovernance(msg.sender).getProposalStatus(proposalId) == 0) {  
    revert INK_ERROR(2049); /// PE: DAO didn't allow to unpledge  
}
```



## 2. InkBadge token being transferable can lead to a repeat voting attack

Severity: High

Category: Business Logic

Target:

- v2-governance-core/contracts/bases/BaseCommittee.sol
- v2-governance-core/contracts/tokens/InkBadgeERC20.sol

### Description

During the voting process, if DAO does not have a govToken set, the balance of the InkBadge token will be used as the maximum number of votes the user can cast.

v2-governance-core/contracts/bases/BaseCommittee.sol:L191-233,L101-119

```
function _vote(
    VoteIdentity memory identity,
    bool agree,
    uint256 count,
    bool requestPledge,
    string memory feedback,
    bytes memory data
) internal {
    // pledge
    (
        bool requirePledgeEngine,
        uint256 requirePledgeValue
    ) = _calculatePledgeValue(_msgSender(), count);
    if (requestPledge) {
        if (requirePledgeEngine) {
            // address stakingEngine = IDAO(getParentDAO()).getStakingEngine();
            IDAO(getParentDAO()).pledge(
                _msgSender(),
                identity.proposalID,
                requirePledgeValue
            );
        } else {
            (, address badgeAddress) = IDAO(getParentDAO())
                .getDAOTokenInfo();

            uint256 decimal = IERC20Metadata(badgeAddress).decimals();

            // require badge
            // require(
            //     count * 10**decimal <= requirePledgeValue,
            //     "badge is not enough"
            // );
            if (count * 10**decimal > requirePledgeValue) {
                revert INK_ERROR(1017);
            }
        }
    }
}
```

```

    }
  }
  ...
}
function _calculatePledgeValue(address user, uint256 votes)
  internal
  view
  returns (bool requirePledgeEngine, uint256 requirePledgeValue)
{
  ...
  if (govToken == address(0)) {
    // no govenance token, P(ledge)=1, B=badge in the wallets
    requirePledgeValue = IERC20(badgeAddress).balanceOf(user);
    requirePledgeEngine = false;
  } else {
    ...
  }
}

```

However, the InkBadge token supports transfer, which will result in the user being able to vote multiple times by vote=>transfer to another account=>vote.

If an attacker creates enough accounts, they can control the voting result of any proposal in this way.

## Recommendation

Consider disabling transfers for badge tokens

## Status

This issue has been acknowledged by the team.

### 3. Proposer can skip the public vote process

Severity: Medium

Category: Business Logic

Target:

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

## Description

v2-governance-core/contracts/proposal/ProposalHandler.sol:L504-538,L843-L911,L271-279

```
function _newProposal(
    NewProposalInfo memory proposal,
    bool commit,
    address proposer,
    bytes memory data
) internal returns (bytes32 proposalID) {
    ...
    for (uint256 i = 0; i < proposal.metadata.length; i++) {
        ItemValue memory itemValue;
        itemValue.typeID = proposal.metadata[i].typeID;
        itemValue.data = proposal.metadata[i].data;
        p.metadata[proposal.metadata[i].key] = itemValue;
    }
    ...
}
```

Proposer can set absoluteWinVotes when creating the proposal.

v2-governance-core/contracts/bases/BaseCommittee.sol:L401-L451

```
function _calculateVoteResults(VoteIdentity memory identity) internal returns (bool
    _passedOrNot) {
    (
        ...
        uint256 absoluteWinVotes,
        ...
    ) = IProposalHandler(getParentDAO()).getVoteRules(
        identity.proposalID
    );
    VoteInfo storage voteInfo = _voteInfos[identity._getIdentityID()];
    bool agree;
    if (absoluteWinVotes!=0 && voteInfo.agreeVotes > absoluteWinVotes &&
        voteInfo.denyVotes <= absoluteWinVotes ) {
        agree = true;
    }
    ...
    _passedOrNot = agree;
}
```

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

```
function getVoteRules(bytes32 proposalID) external view override returns (
    uint256 absoluteWinVotes,
```

```

    ...
)
{
    ...
    absoluteWinVotes = 0;
    (typeID, intData) = getProposalMetadata(
        proposalID,
        ABSOLUTE_WIN_VOTES
    );

    if (typeID == TypeID.UINT256) {
        uint256 value = abi.decode(intData, (uint256));
        if (value > 0) {
            absoluteWinVotes = value;
        }
    }
    ...
}

function getProposalMetadata(bytes32 proposalID, string memory key) public view override
returns (bytes32 typeID, bytes memory data) {
    Proposal storage proposal = _proposals[proposalID];
    return (proposal.metadata[key].typeID, proposal.metadata[key].data);
}

```

BaseCommittee.\_calculateVoteResults() uses *absoluteWinVotes* as a key variable to determine the result.

If a proposer sets the *absoluteWinVotes* to 1 (which allows the proposal to complete settlement with a very low number of votes) and then use `vote()` and `tallyVote()` to complete the voting process, the public vote will be skipped, public voters will not have any opportunity to participate in the vote. As a result, by setting *absoluteWinVotes* to 1, the proposer can pass any proposal.

## Recommendation

It is recommended to add a check for the minimum number of votes, for example:

```

if (
    voteInfo.agreeVoterNum + voteInfo.denyVoterNum <=
    minEffectiveWallets
) {
    agree = false;
}

```

## Status

This issue has been acknowledged by the team.

## 4. User can vote on non-existent proposals

Severity: Medium

Category: Business Logic

Target:

- v2-governance-core/contracts/committee/TheBoard.sol

### Description

TheBoard allows users to vote on non-existent proposals, which means that board members can vote on non-existent proposals before they are removed.

v2-governance-core/contracts/proposal/ProposalHandler.sol:L540-L543

```
function _generateProposalID() internal returns (bytes32 proposalID) {  
    totalProposal++;  
    proposalID = keccak256(abi.encode(_msgSender(), totalProposal));  
}
```

The code above shows that the proposalId is determined by the proposer's address and the total number of proposals. Since the number of proposers is finite, the evil boardMember can predict the proposalId of the next proposal to be created, and thus vote on it before being removed.

### Attach Scenario

1. The board creates a proposalA to remove a particular boardMember(Alice).
2. Alice casts a malicious early vote by predicting the proposalId of a proposal about to be created by another member, before the proposalA has been executed.

Alice's malicious vote could influence the Board's decision-making on upcoming proposals.

### Recommendation

It is recommended to add a check in TheBoard.vote() for the existence of the proposal, for example:

```
require(allowOperate(identity, voteUser));
```

# Appendix

## Appendix 1 - Files in Scope

We audited the commit [9a9fc3f](#), [1a4b6f4](#), [57b3e54](#) that introduced new features to the [Ink-Finance-Inc/v2-governance-core](#) repository.

File	SHA-1 hash
BaseCommittee.sol	fe87e1cd7e7ee37e1945a7b6cb262e78851dd994
BaseDAO.sol	64a6f9d5172b1fa02c792d6fe38290d1814018fa
ThePublic.sol	1dbaebee7e8272d23b3bcfd5bd1025cd6cf53cb2
ProposalHandler.sol	1a77dc8067e3e84f12e94f719bffd7c85dffd91