SALUS SECURITY

MAY 2024

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 Range: See Appendix - 1

Project Dashboard

Application Summary

Name	Ink Finance - Incremental Audit
Version	v1
Type	Solidity
Dates	May 08 2024
Logs	May 08 2024

Vulnerability Summary

Total High-Severity issues	3
Total Medium-Severity issues	1
Total Low-Severity issues	0
Total informational issues	2
Total	6

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 6

1. Incorrect judgment of baseReqTokenType leads to DoS 6

2. Vote tally incorrect when there is no governance token 8

3. Lack of access control leads to arbitrary transfer of funds 10

4. Anyone can participate in voting 11

2.3 Informational Findings 12

5. Missing zero address checks 12

6. Missing two-step transfer ownership pattern 13

Appendix **14**

Appendix 1 - Files in Scope 14

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	Incorrect judgment of baseReqTokenType leads to DoS	High	Denial of Service	Pending
2	Vote tally incorrect when there is no governance token	High	Business Logic	Pending
3	Lack of access control leads to arbitrary transfer of funds	High	Access Control	Pending
4	Anyone can participate in voting	Medium	Access Control	Pending
5	Missing zero address checks	Informational	Data Validation	Pending
6	Missing two-step transfer ownership pattern	Informational	Business Logic	Pending

2.2 Notable Findings

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

1. Incorrect judgment of baseReqTokenType leads to DoS

Severity: High	Category: Denial of Service

Target:
- contracts/bases/BaseCommittee.sol

Description

contracts/bases/BaseCommittee.sol:L201-L242

```
function _getVoteRequirement(VoteIdentity memory identity) internal view
    returns (
        uint256 voteTokenType,
        uint256 baseReqTokenType,
        uint256 baseReqTokenAmt
    )
{
    bytes32 typeID;
    bytes memory intData;

    (typeID, intData) = IProposalHandler(getParentDAO())
        .getProposalMetadata(identity.proposalID, VOTE_TOKEN_TYPE);
    if (intData.length > 0) {
        voteTokenType = abi.decode(intData, (uint256));
    }
    ...

    (address govToken, address badgeAddress) = IDAO(getParentDAO())
        .getDAOTokenInfo();
    if (voteTokenType == 0) {
        if (govToken != address(0)) {
            // use economy token's pledge value
            voteTokenType = 2;
        } else {
            // use badge's pledge value
            voteTokenType = 1;
        }
    }

    if (voteTokenType == 1) {
        baseReqTokenType = 2;
    } else {
        baseReqTokenType = 1;
    }
}
```

By observing the above code, when baseReqTokenType == 2, the vote should be cast based on the badge's pledge value.

contracts/bases/BaseCommittee.sol:L244-L275

```
function _reachBaseLimitation(VoteIdentity memory identity, address user)
    internal
```

```
    view
    returns (bool)
{
    (
        uint256 voteTokenType,
        uint256 baseReqTokenType,
        uint256 baseReqTokenAmt
    ) = _getVoteRequirement(identity);
    if (baseReqTokenAmt > 0) {
        (address economyToken, address badgeAddress) = IDAO(getParentDAO())
            .getDAOTokenInfo();
        if (baseReqTokenType == 1) {
            // check badge is enough
            uint256 badgeBalance = IERC20(badgeAddress).balanceOf(user);
           if (badgeBalance < baseReqTokenAmt) {
                return false;
            }
        } else {
            // check governance token is enough
            uint256 tokenBalance = IERC20(economyToken).balanceOf(user);
            if (tokenBalance < baseReqTokenAmt) {
                return false;
            }
        }
        return true;
    } else {
        return true;
    }
}
```

However, in the _reachBaseLimitation() function, when baseReqTokenType == 2, there is incorrect usage of the economy token's pledge value for checking. This could result in unexpected behavior of the protocol, and worse, the economy token's address might be 0 at this time, potentially leading to denial-of-service for voting operations.

Recommendation

Consider checking governance token (not badge token) when baseReqTokenType == 1.

2. Vote tally incorrect when there is no governance token

Severity: High	Category: Business Logic

Target:
- contracts/bases/BaseCommittee.sol

Description

contracts/bases/BaseCommittee.sol:L201-L242

```
function _getVoteRequirement(VoteIdentity memory identity) internal view
    returns (
        uint256 voteTokenType,
        uint256 baseReqTokenType,
        uint256 baseReqTokenAmt
    )
{
    bytes32 typeID;
    bytes memory intData;

    (typeID, intData) = IProposalHandler(getParentDAO())
        .getProposalMetadata(identity.proposalID, VOTE_TOKEN_TYPE);
    if (intData.length > 0) {
        voteTokenType = abi.decode(intData, (uint256));
    }
    ...

    (address govToken, address badgeAddress) = IDAO(getParentDAO())
        .getDAOTokenInfo();
    if (voteTokenType == 0) {
        if (govToken != address(0)) {
            // use economy token's pledge value
            voteTokenType = 2;
        } else {
            // use badge's pledge value
            voteTokenType = 1;
        }
    }

    if (voteTokenType == 1) {
        baseReqTokenType = 2;
    } else {
        baseReqTokenType = 1;
    }
}
```

By observing the above code, the voteTokenType is first determined by the proposal metadata.

This means that a proposal should be able to specify which type of token can be used for voting.

contracts/bases/BaseCommittee.sol:L102-L130

```
function _calculatePledgeValue(
    VoteIdentity memory identity,
    address user,
    uint256 votes
)
    internal
```

```
    view
    returns (bool requirePledgeEngine, uint256 requirePledgeValue)
{
    (
        uint256 voteTokenType,
        uint256 baseReqTokenType,
        uint256 baseReqTokenAmt
    ) = _getVoteRequirement(identity);

    (address govToken, address badgeAddress) = IDAO(getParentDAO())
        .getDAOTokenInfo();

    if (govToken == address(0) || voteTokenType == 1) {
        // no govenance token, P(ledge)=1, B=badge in the wallets
        requirePledgeValue = IERC20(badgeAddress).balanceOf(user);
        requirePledgeEngine = false;
    } else {
        // there governance tokens, B(adge)=1, badge > 0
        requirePledgeValue = votes;
        requirePledgeEngine = true;
    }
}
```

When the proposal specifies that the vote must be cast using the governance token (voteTokenType == 2). If dao does not have a governance token, the above vote counting process will still follow the badge's pledge value.

This could cause the token used for the final vote tally to be different from what was originally requested in the proposal.

Recommendation

Consider using voteTokenType as the sole basis for calculating the pledge value in _calculatePledgeValue().

3. Lack of access control leads to arbitrary transfer of funds

Severity: High	Category: Access Control

Target:
- contracts/products/funds/FundManager.sol

Description

contracts/products/funds/FundManager.sol:L720-L727

```
function triggerApproveTrade(
    address escrowAddress,
    bytes32 fundID,
    bytes32 tradeID,
    IEscrowManager.EscrowTradeStatus state
) external {
    _triggerApproveTrade(escrowAddress, fundID, tradeID, state);
}

function _triggerApproveTrade(
    address escrowAddress,
    bytes32 fundID,
    bytes32 tradeID,
    IEscrowManager.EscrowTradeStatus state
) internal {
    if (_allRoleSigned(fundID, 2, tradeID)) {
        IEscrowManager(escrowAddress).approveTrade(tradeID, state);
        IEscrowManager.Trade memory trade = IEscrowManager(escrowAddress)
            .getTrade(tradeID);
        for (uint8 i = 0; i < trade.assetOfFund.length; i++) {
            Asset memory payForAsset = trade.assetOfFund[i];
            IFund(_funds[fundID]).unfrozenAsset(
                payForAsset.tokenAddress,
                payForAsset.amount,
                payForAsset.tokenType,
                payForAsset.tokenIdentity
            );

            IUCV(_funds[fundID]).transferTo(
                escrowAddress,
                payForAsset.tokenAddress,
                payForAsset.tokenType,
                payForAsset.tokenIdentity,
                payForAsset.amount,
                "for escrow trading"
            );
        }
    }
}
```

The triggerApproveTrade() function lacks access control, resulting in funds in the fund being able to be transferred out by anyone at will.

Recommendation

Consider adding appropriate access control to the triggerApproveTrade() function.

4. Anyone can participate in voting

Severity: Medium	Category: Access Control
Target: - contracts/bases/BaseCommittee.sol	

Description

contracts/bases/BaseCommittee.sol:L442-L458

```
function _checkAllowOperate(
    ProposalSummary memory proposal,
    VoteIdentity memory identity,
    address user
) internal view returns (bool) {
    if (!_checkProposalStatus(proposal, identity)) {
        console.log("no right proposal");
        return false;
    }

    // if (!_hasRequiredVoteBadge(proposal, user)) {
    //     console.log("no right badge");
    //     return false;
    // }

    return true;
}
```

The _checkAllowOperate() is used to check if the user is eligible to vote.

However, the above code does not check for user, which would result in anyone being able to vote on the proposal.

Recommendation

It is recommended to uncomment the above highlighted code and properly check users permissions.

2.3 Informational Findings

5. Missing zero address checks

Severity: Informational	Category: Data Validation
Target: - contracts/utils/SimpleFaucet.sol	

Description

It is considered a security best practice to verify addresses against the zero address during initialization or setting. However, this precautionary step is absent for address variables “faucetManager”.

contracts/utils/SimpleFaucet.sol:L45-L50

```
function addFaucetManager(address manager) external onlyOwner {
    if (_faucetManager.contains(manager)) {
        return;
    }
    _faucetManager.add(manager);
}
```

Recommendation

Consider adding zero address checks for address variables.

6. Missing two-step transfer ownership pattern

Severity: Informational	Category: Business logic
Target: - contracts/utils/SimpleFaucet.sol	

Description

The SimpleFaucet contract inherits from the Ownable contract. This contract does not implement a two-step process for transferring ownership. Thus, ownership of the contract can easily be lost when making a mistake in transferring ownership.

Recommendation

Consider using the Ownable2Step contract from OpenZeppelin instead.

Appendix

Appendix 1 - Files in Scope

We audited the commit 6003272 that introduced new features to the Ink-Finance-Inc/v2-governance-core repository.

File	SHA-1 hash
ContractUpgradeAgent.sol	5414043cab3bdc6da7c87e3a27d0e2db972def7c
TreasuryManagerAgent.sol	bcb74dcda0c3480fd78d7a7464145674243fc9cb
BaseCommittee.sol	4f2bf939ffa98ab8f3e5c1567a25ad60fd833a37
BaseDAO.sol	2b176560c9d97040bf5def2c7e673f6a8b0551f7
KYCVerifyManager.sol	fbf48fb5adcdb311c1ecf814bb5bb17c59b06721
InkMainDAO.sol	4b1c0ef5c1ce88cf6aa1d986c9b8d7625e5dc966
MasterDAO.sol	a31c121442d9b99f6e0b73ba7502217641db7b15
FundManager.sol	8bae542945a81d8b9321acdbe057b2c218984e21
InkFund.sol	ea67d3be7957b890e3b85c4b7cf05faf96c6e2a7
ProposalHandler.sol	0d33b49d72522ea520ba914f26b1d76bf56f591c
PayrollUCV.sol	944891677e091b85901fca26439e33d37dee2e9e
PayrollUCVManager.sol	e05dd2669bad146397c8a13a5e2112851e37bf3f
SignManager.sol	0e223e195adba4fdca4cbdb8b1d4b9833efed161
SimpleFaucet.sol	6ec60e724da9092e7c77be9efc884eaf3a5293dd