# Smart Contract Security Audit Report

# Table Of Contents

# 1 Executive Summary

On 2024.12.04, the SlowMist security team received the SoSoValueLabs team's security audit application for SSI Protocol , developed the audit plan according to the agreement of both parties and the characteristics of the project, and finally issued the security audit report.

The SlowMist security team adopts the strategy of "white box lead, black, grey box assists" to conduct a complete security test on the project in the way closest to the real attack.

The test method information:

| Test method | Description |
| --- | --- |
| Black box testing | Conduct security tests from an attacker's perspective externally. |
| Grey box testing | Conduct security testing on code modules through the scripting tool, observing the internal running status, mining weaknesses. |
| White box testing | Based on the open source code, non-open source code, to detect whether there are vulnerabilities in programs such as nodes, SDK, etc. |

The vulnerability severity level information:

| Level | Description |
| --- | --- |
| Critical | Critical severity vulnerabilities will have a significant impact on the security of the DeFi project, and it is strongly recommended to fix the critical vulnerabilities. |
| High | High severity vulnerabilities will affect the normal operation of the DeFi project. It is strongly recommended to fix high-risk vulnerabilities. |
| Medium | Medium severity vulnerability will affect the operation of the DeFi project. It is recommended to fix medium-risk vulnerabilities. |
| Low | Low severity vulnerabilities may affect the operation of the DeFi project in certain scenarios. It is suggested that the project team should evaluate and consider whether these vulnerabilities need to be fixed. |
| Weakness | There are safety risks theoretically, but it is extremely difficult to reproduce in engineering. |
| Suggestion | There are better practices for coding or architecture. |

# 2 Audit Methodology

The security audit process of SlowMist security team for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using automated analysis tools.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that was considered during the audit of the smart contract:

| Serial Number | Audit Class | Audit Subclass |
| --- | --- | --- |
| 1 | Overflow Audit | - |
| 2 | Reentrancy Attack Audit | - |
| 3 | Replay Attack Audit | - |
| 4 | Flashloan Attack Audit | - |
| 5 | Race Conditions Audit | Reordering Attack Audit |
| 6 | Permission Vulnerability Audit | Access Control Audit |
| | | Excessive Authority Audit |
| 7 | Security Design Audit | External Module Safe Use Audit |
| | | Compiler Version Security Audit |
| | | Hard-coded Address Security Audit |
| | | Fallback Function Safe Use Audit |
| | | Show Coding Security Audit |
| | | Function Return Value Security Audit |
| | | External Call Function Security Audit |

| Serial Number | Audit Class | Audit Subclass |
| --- | --- | --- |
| 7 | Security Design Audit | Block data Dependence Security Audit |
| | | tx.origin Authentication Security Audit |
| 8 | Denial of Service Audit | - |
| 9 | Gas Optimization Audit | - |
| 10 | Design Logic Audit | - |
| 11 | Variable Coverage Vulnerability Audit | - |
| 12 | "False Top-up" Vulnerability Audit | - |
| 13 | Scoping and Declarations Audit | - |
| 14 | Malicious Event Log Audit | - |
| 15 | Arithmetic Accuracy Deviation Audit | - |
| 16 | Uninitialized Storage Pointer Audit | - |

# 3 Project Overview

## 3.1 Project Introduction

This audit primarily focuses on the differences between the SSI Protocol and the previous audit, which includes the new token staking module. The StakeFactory contract is used to create StakeToken for staking asset tokens. The AssetLocking contract allows users to lock up permitted tokens by depositing them. The USSI contract enables users to deposit asset tokens to mint USSI tokens, and they can later redeem their USSI tokens for the predefined redeemToken set in the contract.

## 3.2 Vulnerability Information

The following is the status of the vulnerabilities found in this audit:

| NO | Title | Category | Level | Status |
| --- | --- | --- | --- | --- |
| N1 | Missing updates to mapped values | Design Logic Audit | High | Fixed |
| N2 | Potential risk of token compatibility | Design Logic Audit | Suggestion | Fixed |
| N3 | Missing ERC20 return value check | Others | Suggestion | Fixed |
| N4 | Risk of excessive authority | Authority Control Vulnerability Audit | Medium | Acknowledged |
| N5 | Lack of checking of order expiration date | Design Logic Audit | Suggestion | Acknowledged |

# 4 Code Overview

## 4.1 Contracts Description

**Audit Version:**

https://github.com/SoSoValueLabs/ssi-protocol

commit: 584f29cda02369b1483f7c5eefb2eb5634abb565

Audit Scope:

the iterative changes compared to the last audit commit 6058696c4e0dee930e12260144698d1a7c3a4b94.

**Fixed Version:**

https://github.com/SoSoValueLabs/ssi-protocol

commit: 4ff5f0db5951905f277d5e5a71025f0968102c06

The main network address of the contract is as follows:

Swap: https://basescan.org/address/0x640cB7201810BC920835A598248c4fe4898Bb5e0

AssetFactory: https://basescan.org/address/0xb04eB6b64137d1673D46731C8f84718092c50B0D

AssetIssuer: https://basescan.org/address/0xdc74D8C5D9a900fdF8a6D03Ad419b236c9A1AD1d

AssetRebalancer: https://basescan.org/address/0x242626E1eCe44601A69d9BC3f72a755Eb393f4b1

AssetFeeManager: https://basescan.org/address/0x996c93827Ab4C55B1044aDd903D2bDb0dcd546BA

StakeFactory: https://basescan.org/address/0x585834242BB31427B1dC7486DD4BDe7c724e35c1

AssetLocking: https://basescan.org/address/0x935A4B1F6F3E891a226b2522ac22d45Ce5839383

USSI: https://basescan.org/address/0x3a46ed8FCeb6eF1ADA2E4600A522AE7e24D2Ed18

StakeToken: https://basescan.org/address/0x7f811E881693af12D84976D59fF3Fb0Eaf135524

## 4.2 Visibility Description

The SlowMist Security team analyzed the visibility of major contracts during the audit, the result as follows:

| StakeToken | | | |
|---|---|---|---|
| Function Name | Visibility | Mutability | Modifiers |
| <Constructor> | Public | Can Modify State | - |
| initialize | Public | Can Modify State | initializer |
| _authorizeUpgrade | Internal | Can Modify State | onlyOwner |
| pause | External | Can Modify State | onlyOwner |
| unpause | External | Can Modify State | onlyOwner |
| decimals | Public | - | - |
| stake | External | Can Modify State | whenNotPaused |
| unstake | External | Can Modify State | whenNotPaused |
| withdraw | External | Can Modify State | whenNotPaused |

| AssetLocking | | | |
|---|---|---|---|
| Function Name | Visibility | Mutability | Modifiers |
| <Constructor> | Public | Can Modify State | - |
| initialize | Public | Can Modify State | initializer |
| _authorizeUpgrade | Internal | Can Modify State | onlyOwner |
| pause | External | Can Modify State | onlyOwner |

| AssetLocking | | | |
|---|---|---|---|
| unpause | External | Can Modify State | onlyOwner |
| setEpoch | External | Can Modify State | onlyOwner |
| updateLockConfig | External | Can Modify State | onlyOwner |
| getActiveTokens | External | - | - |
| lock | External | Can Modify State | whenNotPaused |
| unlock | External | Can Modify State | whenNotPaused |
| withdraw | External | Can Modify State | whenNotPaused |

| StakeFactory | | | |
|---|---|---|---|
| Function Name | Visibility | Mutability | Modifiers |
| <Constructor> | Public | Can Modify State | - |
| initialize | Public | Can Modify State | initializer |
| _authorizeUpgrade | Internal | Can Modify State | onlyOwner |
| setSTImpl | External | Can Modify State | onlyOwner |
| _setSTImpl | Internal | Can Modify State | - |
| createStakeToken | External | Can Modify State | onlyOwner |
| getStakeTokens | External | - | - |
| pauseStakeToken | External | Can Modify State | onlyOwner |
| unpauseStakeToken | External | Can Modify State | onlyOwner |

| USSI | | | |
|---|---|---|---|
| Function Name | Visibility | Mutability | Modifiers |
| <Constructor> | Public | Can Modify State | - |

| USSI | | | |
|---|---|---|---|
| initialize | Public | Can Modify State | initializer |
| _authorizeUpgrade | Internal | Can Modify State | onlyOwner |
| decimals | Public | - | - |
| getSupportAssetIDs | External | - | - |
| addSupportAsset | External | Can Modify State | onlyOwner |
| removeSupportAsset | External | Can Modify State | onlyOwner |
| updateOrderSigner | External | Can Modify State | onlyOwner |
| updateRedeemToken | External | Can Modify State | onlyOwner |
| checkHedgeOrder | Public | - | - |
| applyMint | External | Can Modify State | onlyRole whenNotPaused |
| rejectMint | External | Can Modify State | onlyOwner |
| confirmMint | External | Can Modify State | onlyOwner |
| applyRedeem | External | Can Modify State | onlyRole whenNotPaused |
| rejectRedeem | External | Can Modify State | onlyOwner |
| confirmRedeem | External | Can Modify State | onlyOwner |
| getOrderHashs | External | - | - |
| getOrderHashLength | External | - | - |
| getOrderHash | External | - | - |

## 4.3 Vulnerability Summary

**[N1] [High] Missing updates to mapped values**

**Category: Design Logic Audit**

**Content**

In the AssetFactory contract, the owner role can update the issuer, rebalancer, and fee manager roles of the assetToken by calling the setIssuer, setRebalancer, and setFeeManager functions.However, these functions do not modify the corresponding contract addresses in the mappings issuers, rebalancers, and feeManagers. This will lead to other project contracts (such as USSI.sol) being unable to correctly obtain the updated contract addresses through these mappings.

Code Location:

src/AssetFactory.sol#L105-135

```
    function setIssuer(uint256 assetID, address issuer) external onlyOwner {
        ...
    }

    function setRebalancer(uint256 assetID, address rebalancer) external onlyOwner {
        ...
    }

    function setFeeManager(uint256 assetID, address feeManager) external onlyOwner {
        ...
    }
```

**Solution**

It is suggested to add the code implementation for updating the corresponding mapping values in these three functions.

**Status**

Fixed

### [N2] [Suggestion] Potential risk of token compatibility

**Category: Design Logic Audit**

**Content**

In the AssetLocking contract, users can deposit tokens into the contract for staking by calling the lock function. However, the function does not check whether the difference in the contract's token balance before and after the transfer equals the value of the `amount` parameter. If the token is a deflationary token, it might result in the actual amount transferred being less than the value of the amount, ultimately leading to unexpected errors.

Code Location:

src/AssetLocking.sol#L100-111

```
function lock(address token, uint256 amount) external whenNotPaused {
    ...

    lockData.amount += amount;
    lockConfig.totalLock += amount;
    IERC20(token).transferFrom(msg.sender, address(this), amount);
    emit Lock(msg.sender, token, amount);
}
```

**Solution**

It is recommended to use the difference in the token balance in the contract before and after the user's transfer as the actual amount, instead of directly using the `amount` parameter passed in.

**Status**

Fixed

### [N3] [Suggestion] Missing ERC20 return value check

**Category: Others**

**Content**

The ERC20.transferFrom() function returns a boolean value indicating success. This parameter needs to be checked for success. Some tokens do not revert if the transfer failed but return false instead.

In the AssetLocking contract, users can deposit tokens into the contract by calling the lock function. However, it does not check the return value.

Code Location:

src/AssetLocking.sol#L109

```
function lock(address token, uint256 amount) external whenNotPaused {
    ...

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

**Solution**

It is recommended to use SafeERC20 library or add a check of the return value.

**Status**

Fixed

### [N4] [Medium] Risk of excessive authority

**Category: Authority Control Vulnerability Audit**

**Content**

1.In the AssetFactory contract, the owner role can update the underlying implementation contract of the asset token contract by calling the setTokenImpl function. If the privileges are lost or misused, this could lead to the underlying logic contract of the asset token being upgraded to a malicious contract, posing a risk to users' assets.

Code Location:

src/AssetFactory.sol#L72-82

```
function setTokenImpl(address tokenImpl_) external onlyOwner {
    ...
}
```

2.In the StakeFactory contract, the owner role can update the underlying implementation contract of the stake token contract by calling the setTokenImpl function. If the privileges are lost or misused, this could lead to the underlying logic contract of the stake token being upgraded to a malicious contract, posing a risk to users' assets.

Code Location:

src/StakeFactory.sol#L40-42

```
function setSTImpl(address stImpl_) external onlyOwner {
    _setSTImpl(stImpl_);
}
```

3.In the AssetLocking contract, the owner role can call the updateLockConfig function to set the configuration items for token staking, including the staking epoch, limit, and cooldown period. If the privileges are lost or misused, this may have an impact on the user’s assets.

Code Location:

src/AssetLocking.sol#L72-82

```
    function updateLockConfig(address token, uint8 epoch, uint256 lockLimit, uint48
cooldown) external onlyOwner {
        ...
    }
```

4.In the USSI contract, the owner is responsible for confirming the minting and redemption operations of USSI tokens. Additionally, the owner can also call the updateOrderSigner function to set the signer for order data. If the privileges are lost or misused, this may have an impact on the user's assets.

Code Location:

src/USSI.sol

```
    function updateOrderSigner(address orderSigner_) external onlyOwner {
        ...
    }

    ...

    function rejectMint(bytes32 orderHash) external onlyOwner {
        ...
    }

    function confirmMint(bytes32 orderHash) external onlyOwner {
        ...
    }

    function rejectRedeem(bytes32 orderHash) external onlyOwner {
        ...
    }

    function confirmRedeem(bytes32 orderHash, bytes32 txHash) external onlyOwner {
        ...
    }
```

**Solution**

In the short term, during the early stages of the project, the protocol may need to frequently set various parameters to ensure the stable operation of the protocol. Therefore, transferring the ownership of core roles to a multisig

management can effectively solve the single-point risk, but it cannot mitigate the excessive privilege risk. In the long run, after the protocol stabilizes, transferring the owner ownership to community governance or executing through a timelock can effectively mitigate the excessive privilege risk and increase the community users' trust in the protocol.

**Status**

Acknowledged; The project team's response: we will use cobo mpc custody wallets to manage the ownership of the contracts.

Update: The contract was created using Cobo's MPC wallet, so the permissions for the core roles have been transferred to the MPC wallet for management. This has also been confirmed via official email by Cobo. The following are the transactions made during the setting of permissions:

https://basescan.org/tx/0xcf3f55f38e960f67920bc6ad64c5d427e1195620870812d3b584ce3b4332a63c

https://basescan.org/tx/0xf799ed692feec633522184fb0a64b87ab5bd27c10585b293442fd43382a09970

https://basescan.org/tx/0xd1cca9aa47312f906afd4954e74c60341bd0d682c829dbc14c9fcc0d3e75b45d

https://basescan.org/tx/0x31ca36dccafd358788655d277261d0a2d8d914bd106ada8aae9af05e101fd706

https://basescan.org/tx/0x7ba0be5e9655dd5b3d15edfec8a7853e1d38acaea78ab2d8bb95d216aeac4ffc

https://basescan.org/tx/0xd717eaa5e62c12014ce49312a67d7d6ed0a596cedff4f46b60b83b919fda2895

https://basescan.org/tx/0xab2877f317d48dec5f8c9ca4cf22e7b179a65dd2cc2ed94dc2120f74c7db598d

https://basescan.org/tx/0x59ed850c606e0c3ded162c240bb9f381fe58df1b229141a80dc69ff7029908d7

https://basescan.org/tx/0xbfe81349d9cc0285fabd29a2dc5d12c6345852249890597395ad393ac810138

The taker role for the Swap contract has been set to the relevant contracts; the following are the transactions for this setting:

https://basescan.org/tx/0x852cddcd5d7585b165849cb964d52c31183696f6b2f4c645e6d680adcb6815d9

https://basescan.org/tx/0xac38e9f56dc5e5554b28b8e056043b2bbf45213dca4e8dc428dcd7c6e2a39a53

https://basescan.org/tx/0x8ca6a324cfe6464c4d04fd69fd4a301bafea9fa6bb33fb41827f88d3133663eb

When the maker role confirms the swap in the Swap contract, it's just for bookkeeping purposes; only when the taker role confirms will the user's funds be transferred to the maker. If the maker unexpectedly confirms, the taker role will roll back the confirm status of the maker role. Additionally, before the owner role confirms the transaction, they will check if the outTxHash meets expectations:

- Whether the amount transferred to the vault is as expected.
- Whether the orderHash is included in the data field of the transfer transaction.

### [N5] [Suggestion] Lack of checking of order expiration date

**Category: Design Logic Audit**

**Content**

In the USSI contract, when performing a mint or redemption, the data submitted will set a deadline for the order. However, when the owner confirms the mint or redemption request, they do not check this deadline parameter. This means that even if the submitted order has exceeded the expected expiration time, it can still be successfully confirmed and executed.

Code Location:

src/USSI.sol

```
    function applyMint(HedgeOrder calldata hedgeOrder, bytes calldata orderSignature)
external onlyRole(PARTICIPANT_ROLE) whenNotPaused {
        ...

        hedgeOrder_.deadline = hedgeOrder.deadline;

        ...
    }

    function confirmMint(bytes32 orderHash) external onlyOwner {
        ...
    }

    function applyRedeem(HedgeOrder calldata hedgeOrder, bytes calldata
orderSignature) external onlyRole(PARTICIPANT_ROLE) whenNotPaused {
        ...

        hedgeOrder_.deadline = hedgeOrder.deadline;

        ...
    }

    function confirmRedeem(bytes32 orderHash, bytes32 txHash) external onlyOwner {
        ...
    }
```

**Solution**

It is recommended to add a check for the deadline when confirming mint and redemption requests.

**Status**

Acknowledged; Project team response: As expected, we will check the deadline off-chain.

# 5 Audit Result

| Audit Number | Audit Team | Audit Date | Audit Result |
| --- | --- | --- | --- |
| 0X002412060007 | SlowMist Security Team | 2024.12.04 - 2024.12.06 | Low Risk |

Summary conclusion: The SlowMist security team uses a manual and SlowMist team's analysis tool to audit the project, during the audit work we found 1 high risk, 1 medium risk and 3 suggestion. All the findings were fixed or acknowledged. The project has been deployed on the mainnet, and the permissions of the core roles have been transferred to be managed by the Cobo's MPC wallet and the contracts.

# 6 Statement

SlowMist issues this report with reference to the facts that have occurred or existed before the issuance of this report, and only assumes corresponding responsibility based on these.

For the facts that occurred or existed after the issuance, SlowMist is not able to judge the security status of this project, and is not responsible for them. The security audit analysis and other contents of this report are based on the documents and materials provided to SlowMist by the information provider till the date of the insurance report (referred to as "provided information"). SlowMist assumes: The information provided is not missing, tampered with, deleted or concealed. If the information provided is missing, tampered with, deleted, concealed, or inconsistent with the actual situation, the SlowMist shall not be liable for any loss or adverse effect resulting therefrom. SlowMist only conducts the agreed security audit on the security situation of the project and issues this report. SlowMist is not responsible for the background and other conditions of the project.

**Official Website**

www.slowmist.com

**E-mail**

team@slowmist.com

**Twitter**

@SlowMist_Team

**Github**

https://github.com/slowmist