SLOWMIST

Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

On 2025.01.09, the SlowMist security team received the Shield Layer team's security audit application for Shield Layer, 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

Shield Layer is a DeFi protocol enabling users to mint stablecoins (USLT) against asset deposits and earn staking rewards through stUSLT. The protocol consists of multiple key contracts including ShieldLayer (main contract), USLT (stablecoin), stUSLT (staking vault), ShieldLayerSilo (cooldown management), and RewardProxy (rewards distribution).

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Asset Token Redemption Risk Due to Insufficient Contract Balance	Design Logic Audit	High	Acknowledged
N2	Risk of excessive authority	Authority Control Vulnerability Audit	Medium	Acknowledged
N3	The mintAndStake can bypass the belowMaxMintPerBlock check	Design Logic Audit	Medium	Fixed
N4	Receive can lock users’ native tokens	Others	Low	Fixed
N5	Token Lock Risk Due to Minimum Shares Requirement	Design Logic Audit	Low	Fixed
N6	Redundant code logic design	Design Logic Audit	Suggestion	Fixed
N7	Asset Type Mismatch in Redemption Process	Design Logic Audit	Suggestion	Acknowledged

4 Code Overview

4.1 Contracts Description

Audit Version:

https://github.com/Shield-Layer-2024/shield-layer

commit: ae76d5fb0919d5db5075ca598a94cf4feb79d474

Audit Scope:

./contracts

Fixed Version:

https://github.com/Shield-Layer-2024/shield-layer

commit: 2310749c289cdd3046de23f537e16638b49f3e3f

The main network address of the contract is as follows:

The code was not deployed to the mainnet.

4.2 Visibility Description

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

RewardProxy			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
transferInRewards	External	Can Modify State	onlyRole
rescueTokens	External	Can Modify State	onlyRole

SingleAdminAccessControl			
Function Name	Visibility	Mutability	Modifiers
transferAdmin	External	Can Modify State	onlyRole
acceptAdmin	External	Can Modify State	-
grantRole	Public	Can Modify State	onlyRole notAdmin
revokeRole	Public	Can Modify State	onlyRole notAdmin
renounceRole	Public	Can Modify State	notAdmin
owner	Public	-	-
_grantRole	Internal	Can Modify State	-

stUSLT			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	ERC20 ERC4626 ERC20Permit
transferInRewards	External	Can Modify State	nonReentrant onlyRole notZero

stUSLT

rescueTokens	External	Can Modify State	onlyRole
deposit	Public	Can Modify State	-
totalAssets	Public	-	-
getUnvestedAmount	Public	-	-
decimals	Public	-	-
_checkMinShares	Internal	-	-
_deposit	Internal	Can Modify State	nonReentrant notZero notZero onlyRole
_withdraw	Internal	Can Modify State	nonReentrant notZero notZero onlyRole
renounceRole	Public	Can Modify State	-

stUSLTv2

Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	stUSLT
withdraw	Public	Can Modify State	ensureCooldownOff
redeem	Public	Can Modify State	ensureCooldownOff
unstake	External	Can Modify State	onlyRole
cooldownAssets	External	Can Modify State	ensureCooldownOn onlyRole
cooldownShares	External	Can Modify State	ensureCooldownOn onlyRole
setCooldownDuration	External	Can Modify State	onlyRole

ShieldLayerSilo

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

ShieldLayerSilo			
withdraw	External	Can Modify State	onlyRole

USLT			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	ERC20 ERC20Permit
mint	External	Can Modify State	onlyRole
rescueTokens	External	Can Modify State	onlyRole

ShieldLayer			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
<Receive Ether>	External	Payable	-
mint	External	Can Modify State	nonReentrant belowMaxMintPerBlock
stake	External	Can Modify State	-
mintAndStake	External	Can Modify State	-
redeem	External	Can Modify State	nonReentrant belowMaxBurnPerBlock
unstake	External	Can Modify State	-
cooldownShares	External	Can Modify State	-
rescueTokens	External	Can Modify State	onlyRole
setMaxMintPerBlock	External	Can Modify State	onlyRole
setMaxBurnPerBlock	External	Can Modify State	onlyRole

ShieldLayer			
disableMintBurn	External	Can Modify State	onlyRole
isSupportedAsset	Public	-	-
getAssetRatio	Public	-	ensureAssetSupported
previewMint	Public	-	ensureAssetSupported
addSupportedAsset	Public	Can Modify State	onlyRole
removeSupportedAsset	Public	Can Modify State	onlyRole
setCustodianAddress	Public	Can Modify State	onlyRole
getDomainSeparator	Public	-	-
_mint	Internal	Can Modify State	-
_transferToBeneficiary	Internal	Can Modify State	-
_transferCollateralToCustodian	Internal	Can Modify State	-
_setMaxMintPerBlock	Internal	Can Modify State	-
_setMaxBurnPerBlock	Internal	Can Modify State	-
_computeDomainSeparator	Internal	-	-

4.3 Vulnerability Summary

[N1] [High] Asset Token Redemption Risk Due to Insufficient Contract Balance

Category: Design Logic Audit

Content

In the ShieldLayer contract, users can mint, stake, and mintAndStake to deposit asset tokens in exchange for USLT tokens, which can then be staked in the stUSLT contract for shares. However, the deposited asset tokens are directly transferred to the `custodianAddress` instead of being retained in the current contract. When users want to redeem their asset tokens by calling the redeem function, which burns their USLT tokens and transfers back the corresponding ratio of asset tokens, the transaction might fail due to insufficient asset token balance in the contract since it never directly holds the tokens.

Code location:

contracts/ShieldLayer.sol#L127-L129, L142-L154, L243-L281

```
  function redeem(address asset, uint256 amount) external nonReentrant
belowMaxBurnPerBlock(amount) {
    …
    _transferToBeneficiary(msg.sender, asset, assetAmount);
    emit Redeem(msg.sender, asset, assetAmount, amount);
  }

  function _mint(address asset, uint256 amount) internal returns (uint256) {
    mintedPerBlock[block.number] += amount;
    _transferCollateralToCustodian(amount, asset, msg.sender);
    …
    return usltAmount;
  }

  function _transferToBeneficiary(address beneficiary, address asset, uint256 amount)
internal {
    …
    IERC20(asset).safeTransfer(beneficiary, amount);
  }

  function _transferCollateralToCustodian(uint256 amount, address asset, address
from) internal {
    …
    token.safeTransferFrom(from, custodianAddress, amount);
  }
```

Solution

It's recommended the project team ensure sufficient funds are available in the contract when users attempt to redeem their tokens.

Status

Acknowledged

[N2] [Medium] Risk of excessive authority

Category: Authority Control Vulnerability Audit

Content

1.In the USLT contract, the DEFAULT_ADMIN_ROLE can grant the CONTROLLER_ROLE, and the CONTROLLER_ROLE can call the mint function to mint token arbitrarily without amount limitation.

Code location:

contracts/USLT.sol#L26-L28

```
  function mint(address to, uint256 amount) external onlyRole(CONTROLLER_ROLE) {
    _mint(to, amount);
  }
```

2.In the stUSLTv2 contract, the DEFAULT_ADMIN_ROLE can modify the `cooldownDuration` through the setCooldownDuration function.

Code location:

contracts/stUSLTv2.sol#L136-L144

```
  function setCooldownDuration(uint24 duration) external onlyRole(DEFAULT_ADMIN_ROLE)
{
    if (duration > MAX_COOLDOWN_DURATION) {
      revert InvalidCooldown();
    }

    uint24 previousDuration = cooldownDuration;
    cooldownDuration = duration;
    emit CooldownDurationUpdated(previousDuration, cooldownDuration);
  }
```

3.In the ShieldLayer contract, the DEFAULT_ADMIN_ROLE can modify some sensitive parameters through the setMaxMintPerBlock, setMaxBurnPerBlock, disableMintBurn, addSupportedAsset, removeSupportedAsset, and setCustodianAddress functions. These functions can affect the exchange of contract tokens and the operation of the agreement quota.

Code location:

contracts/ShieldLayer.sol#L167-L182, L202-L227, L284-L295

```
  function setMaxMintPerBlock(uint256 _maxMintPerBlock) external
onlyRole(DEFAULT_ADMIN_ROLE) {
    ...
  }

  function setMaxBurnPerBlock(uint256 _maxBurnPerBlock) external
onlyRole(DEFAULT_ADMIN_ROLE) {
    ...
  }

  function disableMintBurn() external onlyRole(DEFAULT_ADMIN_ROLE) {
    ...
  }
  function addSupportedAsset(address asset, uint256 ratio) public
onlyRole(DEFAULT_ADMIN_ROLE) {
    ...
  }

  function removeSupportedAsset(address asset) public onlyRole(DEFAULT_ADMIN_ROLE) {
    ...
  }

  function setCustodianAddress(address custodian) public onlyRole(DEFAULT_ADMIN_ROLE)
{
    ..
  }

  function _setMaxMintPerBlock(uint256 _maxMintPerBlock) internal {
    ...
  }

  function _setMaxBurnPerBlock(uint256 _maxBurnPerBlock) internal {
    ...
  }
```

4.In the ShieldLayer contract, the DEFAULT_ADMIN_ROLE can transfer specific ERC20 tokens from this contract to any address through the rescueTokens function. But the ERC20 tokens should not be the asset token of this contract, in case there are not enough asset tokens in the contract for the user to perform the redemption operation.

Code location:

contracts/ShieldLayer.sol#L164-L166

```
  function rescueTokens(address token, uint256 amount, address to) external
onlyRole(DEFAULT_ADMIN_ROLE) {
    IERC20(token).safeTransfer(to, amount);
  }
```

Solution

In the short term, transferring owner ownership to multisig contracts is an effective solution to avoid single-point risk. But in the long run, it is a more reasonable solution to implement a privilege separation strategy and set up multiple privileged roles to manage each privileged function separately. The authority involving user funds should be managed by the community, and the authority involving emergency contract suspension can be managed by the EOA address. This ensures both a quick response to threats and the safety of user funds.

5. It’s recommended to check the asset token is not the asset token.

Status

Acknowledged; After communicating with the project team, they stated that they will transfer admin role to a multi-sig wallet to avoid single-point risk.

[N3] [Medium] The mintAndStake can bypass the belowMaxMintPerBlock check

Category: Design Logic Audit

Content

In the ShieldLayer contract, there exists a belowMaxMintPerBlock modifier that checks if the current block's token minting amount does not exceed maxMintPerBlock. While the mint function includes this check, the mintAndStake function lacks this protection. This inconsistency allows users to bypass the block minting limit by calling mintAndStake to mint unlimited USLT tokens and stake them in the stUSLT contract.

Code Location:

contracts/ShieldLayer.sol#L135-L139

```
  function mintAndStake(address asset, uint256 amount) external {
    uint256 usltAmount = _mint(asset, amount);
    uslt.approve(address(stuslt), usltAmount);
    stuslt.deposit(usltAmount, msg.sender);
  }
```

Solution

It is recommended to add the belowMaxMintPerBlock modifier to the mintAndStake function to maintain consistency in minting limits across all minting functions.

Status

Fixed

[N4] [Low] Receive can lock users’ native tokens

Category: Others

Content

There is a receive function in the ShieldLayer contract so that the contract can receive native tokens. However, the receive function can lock users’ native tokens when users transfer the native token in these contracts by mistake.

Code location:

contracts/ShieldLayer.sol#L122-L124

```
receive() external payable {
  emit Received(msg.sender, msg.value);
}
```

Solution

It’s recommended to remove the receive() function if it is redundant.

Status

Fixed

[N5] [Low] Token Lock Risk Due to Minimum Shares Requirement

Category: Design Logic Audit

Content

In the stUSLT contract, the _checkMinShares function is called after both _deposit and _withdraw operations to ensure that when stUSLT share tokens still have a balance, the total supply cannot be lower than MINSHARES. This means a certain amount of USLT tokens will always be locked in the contract and cannot be fully withdrawn. This implementation creates a risk where user funds below the MINSHARES threshold could become permanently locked in the contract if they represent the last remaining shares. This will also result in a small portion of the remaining deposits being held by other users.

Code location:

contracts/stUSLT.sol#L136-L176

```
  function _checkMinShares() internal view {
    uint256 _totalSupply = totalSupply();
    if (_totalSupply > 0 && _totalSupply < MIN_SHARES) revert MinSharesViolation();
  }

  function _deposit(address caller, uint256 assets, uint256 shares)
    internal
    nonReentrant
    notZero(assets)
    notZero(shares)
    onlyRole(CONTROLLER_ROLE)
  {
    super._deposit(caller, caller, assets, shares);
    _checkMinShares();
  }

  function _withdraw(address caller, address receiver, address _owner, uint256
assets, uint256 shares)
    internal
    override
    nonReentrant
    notZero(assets)
    notZero(shares)
    onlyRole(CONTROLLER_ROLE)
  {
    super._withdraw(caller, receiver, _owner, assets, shares);
    _checkMinShares();
  }
```

Solution

Implement a minimum deposit requirement for shares to prevent deposits that would result in share amounts below MINSHARES from being locked in the contract. This would ensure users are aware of the minimum threshold before depositing and prevent potential fund lockup.

Status

Fixed

[N6] [Suggestion] Redundant code logic design

Category: Design Logic Audit

Content

In the stUSLT contract, getUnvestedAmount will record the unvested stUSLT tokens in the contract. When REWARDER_ROLE calls the transferInRewards function to transfer asset tokens to the contract, it will first check if (getUnvestedAmount() > 0) revert StillVesting(); whether getUnvestedAmount is greater than 0. If so, it will revert. However, in the subsequent code, uint256 newVestingAmount = amount + getUnvestedAmount(); will still add the result of getUnvestedAmount, which makes the code logic redundant.

Code location:

contracts/stUSLT.sol#L71-L81

```
  function transferInRewards(uint256 amount) external nonReentrant
onlyRole(REWARDER_ROLE) notZero(amount) {
    if (getUnvestedAmount() > 0) revert StillVesting();
    uint256 newVestingAmount = amount + getUnvestedAmount();

    vestingAmount = newVestingAmount;
    lastDistributionTimestamp = block.timestamp;
    // transfer assets from rewarder to this contract
    IERC20(asset()).safeTransferFrom(msg.sender, address(this), amount);

    emit RewardsReceived(amount, newVestingAmount);
  }

  function getUnvestedAmount() public view returns (uint256) {
    uint256 timeSinceLastDistribution = block.timestamp - lastDistributionTimestamp;

    if (timeSinceLastDistribution >= VESTING_PERIOD) {
      return 0;
    }

    return ((VESTING_PERIOD - timeSinceLastDistribution) * vestingAmount) /
VESTING_PERIOD;
  }
```

Solution

It is recommended to delete the code of redundant logic.

Status

Fixed

[N7] [Suggestion] Asset Type Mismatch in Redemption Process

Category: Design Logic Audit

Content

In the ShieldLayer contract, the DEFAULT_ADMIN_ROLE role can use the addSupportedAsset function to add any asset and the corresponding exchange ratio to the contract. Users can use the mint and mintAndStake functions to deposit contract-specific asset tokens, and finally redeem the specified asset tokens through the redeem function. If different asset tokens have different values and there is a setting deviation in the ratio, users may withdraw tokens with higher values deposited by other users during redemption. Similarly, if the removeSupportedAsset function removes an asset token previously added by a user, the user can no longer retrieve the previously deposited asset. Instead, only the currently supported asset tokens can be withdrawn.

Code location:

contracts/ShieldLayer.sol#L127-L129, L135-L139, L142-L154

```
  function mint(address asset, uint256 amount) external nonReentrant
belowMaxMintPerBlock(amount) {
    _mint(asset, amount);
  }

  function mintAndStake(address asset, uint256 amount) external {
    uint256 usltAmount = _mint(asset, amount);
    …
  }

  function redeem(address asset, uint256 amount) external nonReentrant
belowMaxBurnPerBlock(amount) {
    uint256 assetRatio = getAssetRatio(asset);
    uint256 assetAmount = amount / assetRatio;
    if (IERC20(asset).balanceOf(address(this)) < assetAmount) revert
InsufficientAsset();
    …
    _transferToBeneficiary(msg.sender, asset, assetAmount);
    emit Redeem(msg.sender, asset, assetAmount, amount);
  }
```

Solution

If this is not intended behavior, implement validation to ensure users can only redeem the same asset type they originally deposited.

Status

Acknowledged; Afte communicating with the project team, they stated that they will only support USDT for now, asset type mismatch will not happen.

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002501100001	SlowMist Security Team	2025.01.09 - 2025.01.10	High 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, 2 medium risks, 2 low risks, 1 suggestion, and 1 information. All the findings were fixed or acknowledged. The code was not deployed to the mainnet. After communicating with the project team, they stated that the funds come from exchange profit that is manually transferred in, and "insufficient funds" may often occur. So the audit result is high-risk

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.

SLOWMIST

Official Website
www.slowmist.com

E-mail
team@slowmist.com

Twitter
@SlowMist_Team

Github
https://github.com/slowmist