Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

On 2024.10.21, the SlowMist security team received the Avalon Finance team's security audit application for Avalon Finance, 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 focuses on the liquidity staking and token modules of Avalon Finance, with the primary scope being the SavingAccount, USDa, and USDaOFTAdapter contracts. Users can mint sUSDa tokens from the SavingAccount contract using USDa tokens, and holding sUSDa tokens automatically calculates the staking interest. When users redeem their USDa tokens, the accrued interest is also transferred to them. Interest distribution requires the manager role to manually call the distributeInterests function to transfer the interest into the contract for distribution.

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Improper use of logical symbols	Design Logic Audit	High	Fixed
N2	Inappropriate function logic	Design Logic Audit	Medium	Fixed
N3	Lack of amount check during minting	Design Logic Audit	Suggestion	Fixed
N4	Risk of excessive privilege	Authority Control Vulnerability Audit	Medium	Acknowledged
N5	Missing scope limit	Others	Suggestion	Fixed
N6	Potential loss of interest	Design Logic Audit	Low	Fixed
N7	Potential issue with claim failures	Design Logic Audit	Low	Acknowledged

4 Code Overview

4.1 Contracts Description

Audit Version:

https://github.com/avalonfinancexyz/Saving-Account

commit: 3806530473d4ec9775c2664577c6744f9109b734

https://github.com/avalonfinancexyz/USDa-oft

commit: 393349344296c03bf4d5512c8ede7daf2b83cf24

Fixed Version:

https://github.com/avalonfinancexyz/Saving-Account

commit: a5651e2e65f4c78384ee4133097f77f38abd166d

https://github.com/avalonfinancexyz/USDa-oft

commit: 7f04f2f458e9e9bac52ead700baf4fba41f36ce3

Audit scope:

- Saving-Account/contracts/SavingAccount.sol
- Saving-Account/contracts/interface/*.sol
- USDa-oft/contracts/USDaOFTAdapter.sol
- USDa-oft/contracts/USDa.sol

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:

SavingAccount			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
pause	External	Can Modify State	onlyRole
unpause	External	Can Modify State	onlyRole
setTargetAPR	External	Can Modify State	onlyRole realizeReward
setProcessPeriod	External	Can Modify State	onlyRole
getTotalUnderlying	Public	-	-
getSharesByAmount	Public	-	-
getAmountByShares	Public	-	-
getRPS	Public	-	-
mint	External	Can Modify State	-
mintFor	External	Can Modify State	-

SavingAccount			
_mintFor	Internal	Can Modify State	realizeReward nonReentrant whenNotPaused
redeem	External	Can Modify State	realizeReward nonReentrant whenNotPaused
claimRedeem	External	Can Modify State	realizeReward nonReentrant whenNotPaused
balanceOf	Public	-	-
distributeInterests	External	Can Modify State	realizeReward onlyRole

USDa			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	OFT Ownable
addBlackList	Public	Can Modify State	onlyRole
removeBlackList	Public	Can Modify State	onlyRole
pause	External	Can Modify State	onlyOwner
unpause	External	Can Modify State	onlyOwner
mint	Public	Can Modify State	onlyRole
burn	Public	Can Modify State	onlyRole
_update	Internal	Can Modify State	whenNotPaused

USDaOFTAdapter			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	OFTAdapter Ownable

4.3 Vulnerability Summary

[N1] [High] Improper use of logical symbols

Category: Design Logic Audit

Content

In the USDa contract, the _update function is called when performing token transfers, where it checks whether the from and to addresses are in the blacklist. However, the logical operator used for this check is || instead of &&, which means that if one address is in the blacklist and the other is not, the check will still pass.

Code Location:

USDa-oft/contracts/USDa.sol#L63

```
    function _update(address from, address to, uint256 value) internal override
whenNotPaused {
        require(!isBlackListed[from] || !isBlackListed[to], "isBlackListed");
        super._update(from, to, value);
    }
```

Solution

It is recommended to change the logical operator in the check to &&.

Status

Fixed

[N2] [Medium] Inappropriate function logic

Category: Design Logic Audit

Content

In the SavingAccount contract, the getTotalUnderlying function is used to calculate the total amount of USDa and is utilized in the getSharesByAmount and getAmountByShares functions to compute the conversion quantities between shares and USDa.

However, in the actual deposit and redemption processes, the totalUnderlying used does not directly include the interest; instead, the interest can only be added after the pool manager role calls the distributeInterests function. This would result in a situation where, if interest has been generated on the deposits within the contracta but the manager role has not yet called the distributeInterests function to add this interest to the totalUnderlying, the total amount of USDa obtained by calling the getTotalUnderlying function would not match the actual totalUnderlying. This would also affect the results of the getSharesByAmount and getAmountByShares functions.

Code Location:

Saving-Account/contracts/SavingAccount.sol#L122-126

```
    function getTotalUnderlying() public view returns (uint256) {
        // need include manager fee
        uint256 totalInterest = getRPS() * (block.timestamp - lastCheckpoint);
        return totalUnderlying + totalInterest;
    }
```

Solution

It is recommended that the getTotalUnderlying function directly return the value of totalUnderlying used during actual minting and redemption.

Status

Fixed; Update: The fixed version of totalUnderlying will be updated in real-time when calculating interest.

[N3] [Suggestion] Lack of amount check during minting

Category: Design Logic Audit

Content

In the SavingAccount contract, During minting, there is no check on the passed amount parameter. If the amount parameter is too small, it might be calculated as 0 in the getRPS function due to rounding issues when calculating interest per second.

Code Location:

Saving-Account/contracts/SavingAccount.sol#L196-213

```
    function _mintFor(
        uint256 amount,
        address receiver
    ) internal realizeReward nonReentrant whenNotPaused{
        usda.safeTransferFrom(msg.sender, address(this), amount);

        ...

        totalUnderlying = totalUnderlying + amount;
    }
```

Solution

It is recommended to add a minimum limit check for the amount parameter in the _mintFor function.

Status

Fixed

[N4] [Medium] Risk of excessive privilege

Category: Authority Control Vulnerability Audit

Content

In the USDa contract, the MINT_ROLE can call the mint function to mint any amount of USDa tokens to any address, and the BURN_ROLE can call the burn function to burn USDa tokens from any address. If the privilege is lost or misused, this may have an impact on the user's assets.

Code Location:

USDa-oft/contracts/USDa.sol#L54-60

```
function mint(address _user, uint256 _amount) public onlyRole(MINT_ROLE) {
    _mint(_user, _amount);
}

function burn(address _user, uint256 _amount) public onlyRole(BURN_ROLE) {
    _burn(_user, _amount);
}
```

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 and executing through a timelock can effectively mitigate the excessive privilege risk and increase the community users' trust in the protocol.

Status

Acknowledged; Response from the project team: the timelock and multi-signature contracts will be used to manage permissions.

[N5] [Suggestion] Missing scope limit

Category: Others

Content

In the SavingAccount contract, the pool manager role can set the interval for claiming tokens by calling the setProcessPeriod function. Since there is no upper limit set for the new processPeriod when it is being configured, setting it too high may result in users being unable to retrieve their asset tokens.

Code Location:

Saving-Account/contracts/SavingAccount.sol#L112-117

```
    function setProcessPeriod(
        uint256 _processPeriod
    ) external onlyRole(POOL_MANAGER_ROLE) {
        processPeriod = _processPeriod;
        emit ProcessPeriodChanged(_processPeriod);
    }
```

Solution

It is recommended to implement a maximum limit check for the _processPeriod parameter within the setProcessPeriod function.

Status

Fixed

[N6] [Low] Potential loss of interest

Category: Design Logic Audit

Content

In the SavingAccount contract, since the allocation of unpaid interest requires the manager role to manually call the distributeInterests function to add the interest to the totalUnderlying, the following scenarios may occur:

1.If the distributeInterests function is called after the user calls the redeem function, then the user will not receive the interest that has accrued from the time of deposit to the time of redemption.

2.If the distributeInterests function is called before the user calls the redeem function, then the user will not receive the interest that accrues from the time of the distributeInterests function call until the time of redemption.

3.A call to the getRPS function after the distributeInterests function cannot accumulate the newly generated interest during this period into the totalUnderlying for compound interest calculation.

Solution

It is recommended to update totalUnderlying in real time to avoid interest loss.

Status

Fixed

[N7] [Low] Potential issue with claim failures

Category: Design Logic Audit

Content

In the SavingAccount contract, after users perform a redemption operation, they need to wait for a certain period before they can claim back their tokens (principal + interest). If the processPeriod is relatively short, and assuming that before the user calls the claim function, the manager role has not yet called the distributeInterests function to transfer the interest portion of the tokens into the contract, this could result in users being unable to claim their tokens due to insufficient tokens in the contract.

Solution

It is recommended that the manager role timely distributes the interest after users redeem, to ensure there are sufficient tokens in the contract for users to claim.

Status

Acknowledged; The project team responds: They will transfer a sum of funds into the contract in advance and distribute the interest based on the situation.

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002410220001	SlowMist Security Team	2024.10.21 - 2024.10.22	Medium 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 risk, 2 low risk and 2 suggestion. All the findings were fixed or acknowledged. Since the project has not yet been deployed to the mainnet and the permissions of the core roles have not yet been transferred, the risk level reported is temporarily medium.

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