Smart Contract
Security Audit Report

Table Of Contents

1 Executive Summary

On 2022.09.05, the SlowMist security team received the Earning.Farm team's security audit application for Earning.Farm V3, 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.

Level	Description
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

Serial Number	Audit Class	Audit Subclass
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
		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

Audit Version:

https://github.com/Shata-Capital/ENF_V3

commit: 3686154870acbffc1a846781574abdd0b5295bd5

Fixed Version:

https://github.com/Shata-Capital/ENF_V3

commit: dc0dbd9b78483855c4cdb6e006f0d7b1e6e9d2f8

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Deposit defect issue	Design Logic Audit	Low	Confirmed
N2	Incorrect withdrawal amount check	Design Logic Audit	Low	Fixed
N3	Risk of overburning shares	Design Logic Audit	Critical	Fixed
N4	Small deposit issue	Design Logic Audit	Low	Ignored
N5	The deflationary token issue	Others	Suggestion	Fixed
N6	Risk of share manipulation	Design Logic Audit	Critical	Fixed
N7	Missing event records	Others	Suggestion	Fixed
N8	AllocPoint deposit issue	Design Logic Audit	Low	Fixed
N9	check withdrawal amount issue	Design Logic Audit	Low	Ignored

NO	Title	Category	Level	Status
N10	Risks of fake routers	Design Logic Audit	Medium	Fixed
N11	Loss of computational precision	Design Logic Audit	Low	Fixed
N12	Risks of strict equality checks	Design Logic Audit	Critical	Fixed
N13	Negative number check issue	Design Logic Audit	Medium	Fixed
N14	Risk of excessive authority	Authority Control Vulnerability	Medium	Confirmed
N15	Code redundancy issue	Others	Suggestion	Fixed
N16	Invalid minimum output calculation	Design Logic Audit	Medium	Fixed
N17	Risk of pid acquisition	Design Logic Audit	Low	Fixed
N18	Redundant approval issue	Others	Suggestion	Fixed
N19	Incorrect storage of temporary variables	Design Logic Audit	Medium	Fixed
N20	Compound interest slippage check issue	Design Logic Audit	Medium	Confirmed
N21	Lack of access control	Authority Control Vulnerability	Medium	Fixed

4 Code Overview

4.1 Contracts Description

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:

EFVault			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	initializer
deposit	Public	Can Modify State	nonReentrant unPaused
getBalance	Internal	-	-
withdraw	Public	Can Modify State	nonReentrant unPaused
totalAssets	Public	-	-
convertToShares	Public	-	-
convertToAssets	Public	-	-
setMaxDeposit	Public	Can Modify State	onlyOwner
setMaxWithdraw	Public	Can Modify State	onlyOwner
setController	Public	Can Modify State	onlyOwner
setDepositApprover	Public	Can Modify State	onlyOwner
pause	Public	Can Modify State	onlyOwner
resume	Public	Can Modify State	onlyOwner

Controller			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	initializer

Controller			
<Receive Ether>	External	Payable	-
deposit	External	Can Modify State	onlyVault
_deposit	Internal	Can Modify State	-
withdraw	External	Can Modify State	onlyVault
harvest	Public	Can Modify State	onlyOwner
getBalance	Internal	-	-
moveFund	Public	Can Modify State	onlyOwner
totalAssets	External	-	-
subStrategyLength	External	-	-
setVault	Public	Can Modify State	onlyOwner
setAPYSort	Public	Can Modify State	onlyOwner
setTreasury	Public	Can Modify State	onlyOwner
setExchange	Public	Can Modify State	onlyOwner
setWithdrawFee	Public	Can Modify State	onlyOwner
setAllocPoint	Public	Can Modify State	onlyOwner
registerSubStrategy	Public	Can Modify State	onlyOwner
setDefaultDepositSS	Public	Can Modify State	onlyOwner
setDefaultOption	Public	Can Modify State	onlyOwner

DepositApprover

DepositApprover			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
deposit	Public	Can Modify State	-
setVault	Public	Can Modify State	onlyOwner

Exchange			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
<Receive Ether>	External	Payable	-
swapExactTokenInput	External	Can Modify State	onlyController
swapExactETHInput	External	Payable	onlyController
getBalance	Internal	-	-

BalancerBatchV2			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
<Receive Ether>	External	Payable	-
setExchange	Public	Can Modify State	onlyOwner
addPath	Public	Can Modify State	onlyOwner
getPathIndex	Public	-	-
removePath	Public	Can Modify State	onlyOwner

BalancerBatchV2			
pathFrom	Public	-	-
pathTo	Public	-	-
swap	External	Can Modify State	onlyExchange
getBalance	Internal	-	-

BalancerV2			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
<Receive Ether>	External	Payable	-
setExchange	Public	Can Modify State	onlyOwner
addPath	Public	Can Modify State	onlyOwner
getPathIndex	Public	-	-
removePath	Public	Can Modify State	onlyOwner
pathFrom	Public	-	-
pathTo	Public	-	-
swap	External	Can Modify State	onlyExchange
getBalance	Internal	-	-

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

Curve			
<Receive Ether>	External	Payable	-
setExchange	Public	Can Modify State	onlyOwner
addCurvePool	Public	Can Modify State	onlyOwner
removeCurvePool	Public	Can Modify State	onlyOwner
getPathIndex	Public	-	-
pathFrom	Public	-	-
pathTo	Public	-	-
swap	External	Can Modify State	onlyExchange
getBalance	Internal	-	-

UniswapV2			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
<Receive Ether>	External	Payable	-
setExchange	Public	Can Modify State	onlyOwner
addPath	Public	Can Modify State	onlyOwner
getPathIndex	Public	-	-
removePath	Public	Can Modify State	onlyOwner
pathFrom	Public	-	-
pathTo	Public	-	-

UniswapV2			
swap	External	Can Modify State	onlyExchange
getBalance	Internal	-	-

UniswapV3			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
<Receive Ether>	External	Payable	-
setExchange	Public	Can Modify State	onlyOwner
addPath	Public	Can Modify State	onlyOwner
getPathIndex	Public	-	-
removePath	Public	Can Modify State	onlyOwner
pathFrom	Public	-	-
pathTo	Public	-	-
swap	External	Can Modify State	-
getBalance	Internal	-	-

Aave			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	initializer
totalAssets	External	-	-
getVirtualPrice	Public	-	-

Aave			
_totalAssets	Internal	-	-
deposit	External	Can Modify State	onlyController
_deposit	Internal	Can Modify State	-
withdraw	External	Can Modify State	onlyController
harvest	External	Can Modify State	onlyController
emergencyWithdraw	Public	Can Modify State	onlyOwner
ownerDeposit	Public	Can Modify State	onlyOwner
withdrawable	External	-	-
setController	Public	Can Modify State	onlyOwner
setDepositSlippage	Public	Can Modify State	onlyOwner
setWithdrawSlippage	Public	Can Modify State	onlyOwner
setPoolId	Public	Can Modify State	onlyOwner
setLPToken	Public	Can Modify State	onlyOwner
setCurvePool	Public	Can Modify State	onlyOwner
setHarvestGap	Public	Can Modify State	onlyOwner
setMaxDeposit	Public	Can Modify State	onlyOwner
addRewardToken	Public	Can Modify State	onlyOwner
removeRewardToken	Public	Can Modify State	onlyOwner

Alusd

Alusd			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	initializer
totalAssets	External	-	-
getVirtualPrice	Public	-	-
_totalAssets	Internal	-	-
deposit	External	Can Modify State	onlyController
_deposit	Internal	Can Modify State	-
withdraw	External	Can Modify State	onlyController
harvest	External	Can Modify State	onlyController
emergencyWithdraw	Public	Can Modify State	onlyOwner
ownerDeposit	Public	Can Modify State	onlyOwner
withdrawable	External	-	-
setController	Public	Can Modify State	onlyOwner
setDepositSlippage	Public	Can Modify State	onlyOwner
setWithdrawSlippage	Public	Can Modify State	onlyOwner
setPoolId	Public	Can Modify State	onlyOwner
setLPToken	Public	Can Modify State	onlyOwner
setCurvePool	Public	Can Modify State	onlyOwner
setHarvestGap	Public	Can Modify State	onlyOwner
setMaxDeposit	Public	Can Modify State	onlyOwner

Alusd			
addRewardToken	Public	Can Modify State	onlyOwner
removeRewardToken	Public	Can Modify State	onlyOwner
getPID	Public	-	-

CompoundV3			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	initializer
totalAssets	External	-	-
getVirtualPrice	Public	-	-
_totalAssets	Internal	-	-
deposit	External	Can Modify State	onlyController
_deposit	Internal	Can Modify State	-
withdraw	External	Can Modify State	onlyController
harvest	External	Can Modify State	onlyController
emergencyWithdraw	Public	Can Modify State	onlyOwner
ownerDeposit	Public	Can Modify State	onlyOwner
withdrawable	External	-	-
setController	Public	Can Modify State	onlyOwner
setDepositSlippage	Public	Can Modify State	onlyOwner
setWithdrawSlippage	Public	Can Modify State	onlyOwner

CompoundV3			
setPoolId	Public	Can Modify State	onlyOwner
setLPToken	Public	Can Modify State	onlyOwner
setCurvePool	Public	Can Modify State	onlyOwner
setHarvestGap	Public	Can Modify State	onlyOwner
setMaxDeposit	Public	Can Modify State	onlyOwner
addRewardToken	Public	Can Modify State	onlyOwner
removeRewardToken	Public	Can Modify State	onlyOwner

Lusd			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	initializer
totalAssets	External	-	-
getVirtualPrice	Public	-	-
_totalAssets	Internal	-	-
deposit	External	Can Modify State	onlyController
_deposit	Internal	Can Modify State	-
withdraw	External	Can Modify State	onlyController
harvest	External	Can Modify State	onlyController
emergencyWithdraw	Public	Can Modify State	onlyOwner
ownerDeposit	Public	Can Modify State	onlyOwner

Lusd			
withdrawable	External	-	-
setController	Public	Can Modify State	onlyOwner
setDepositSlippage	Public	Can Modify State	onlyOwner
setWithdrawSlippage	Public	Can Modify State	onlyOwner
setPoolId	Public	Can Modify State	onlyOwner
setLPToken	Public	Can Modify State	onlyOwner
setCurvePool	Public	Can Modify State	onlyOwner
setHarvestGap	Public	Can Modify State	onlyOwner
setMaxDeposit	Public	Can Modify State	onlyOwner
addRewardToken	Public	Can Modify State	onlyOwner
removeRewardToken	Public	Can Modify State	onlyOwner

Tri			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	initializer
totalAssets	External	-	-
getVirtualPrice	Public	-	-
_totalAssets	Internal	-	-
deposit	External	Can Modify State	onlyController
_deposit	Internal	Can Modify State	-

Tri			
withdraw	External	Can Modify State	onlyController
harvest	External	Can Modify State	onlyController
emergencyWithdraw	Public	Can Modify State	onlyOwner
ownerDeposit	Public	Can Modify State	onlyOwner
withdrawable	External	-	-
setController	Public	Can Modify State	onlyOwner
setDepositSlippage	Public	Can Modify State	onlyOwner
setWithdrawSlippage	Public	Can Modify State	onlyOwner
setPoolId	Public	Can Modify State	onlyOwner
setLPToken	Public	Can Modify State	onlyOwner
setCurvePool	Public	Can Modify State	onlyOwner
setHarvestGap	Public	Can Modify State	onlyOwner
setMaxDeposit	Public	Can Modify State	onlyOwner
addRewardToken	Public	Can Modify State	onlyOwner
removeRewardToken	Public	Can Modify State	onlyOwner

Cusdc			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	initializer
totalAssets	External	-	-

Cusdc			
_totalAssets	Internal	-	-
deposit	External	Can Modify State	onlyController
_deposit	Internal	Can Modify State	-
withdraw	External	Can Modify State	onlyController
_withdraw	Internal	Can Modify State	-
harvest	External	Can Modify State	onlyController
emergencyWithdraw	Public	Can Modify State	onlyOwner
withdrawable	External	-	-
ownerDeposit	Public	Can Modify State	onlyOwner
setController	Public	Can Modify State	onlyOwner
setDepositSlippage	Public	Can Modify State	onlyOwner
setWithdrawSlippage	Public	Can Modify State	onlyOwner
setHarvestGap	Public	Can Modify State	onlyOwner
setMaxDeposit	Public	Can Modify State	onlyOwner

4.3 Vulnerability Summary

[N1] [Low] Deposit defect issue

Category: Design Logic Audit

Content

In the EFVault contract, it is not restricted to call the deposit function only by the DepositApprover contract. If the user transfers funds to the EFVault contract by mistake, any user can call the deposit function to deposit for himself.

Code location: contracts/core/vault.sol

```
    function deposit(uint256 assets, address receiver) public virtual nonReentrant
unPaused returns (uint256 shares) {
        ...
    }
```

Solution

It is recommended that the deposit function can only be called by the DepositApprover contract.

Status

Confirmed; After communicating with the project team, the project team stated that this is the expected design.

[N2] [Low] Incorrect withdrawal amount check

Category: Design Logic Audit

Content

In the vault contract, users can withdraw funds through the withdraw function. It will check if the funds withdrawn by the user is less than the user's total deposit, but this will prevent the user from withdrawing all of their total deposit.

Code location: contracts/core/vault.sol

```
    function withdraw(uint256 assets, address receiver) public virtual nonReentrant
unPaused returns (uint256 shares) {
        ...
        uint256 totalDeposit = convertToAssets(balanceOf(msg.sender));
        console.log("Total Deposit: ", totalDeposit);

        require(assets < totalDeposit, "EXCEED_TOTAL_DEPOSIT");
        ...
    }
```

Solution

It is recommended to check whether the user's withdrawal amount is less than or equal to his total deposit.

Status

Fixed

[N3] [Critical] Risk of overburning shares

Category: Design Logic Audit

Content

In the vault contract, users can burn their shares to withdraw funds through the withdraw function. However, when calculating the required burning share, it incorrectly divides the user's total deposit. This will cause the number of shares to be burned to be much larger than expected.

Code location: contracts/core/vault.sol

```
    function withdraw(uint256 assets, address receiver) public virtual nonReentrant
unPaused returns (uint256 shares) {
        ...
        uint256 totalDeposit = convertToAssets(balanceOf(msg.sender));
        ...
        // Calculate share amount to be burnt
        shares = (totalSupply() * assets) / totalDeposit;
        ...
    }
```

Solution

TotalAssets should be divided when calculating the share required to burn.

Status

Fixed

[N4] [Low] Small deposit issue

Category: Design Logic Audit

Content

When a user makes a deposit, the vault contract will deposit the user's funds into the strategy pool and then mint the

corresponding share to the user. If the total deposit of the contract is very large at this time, when the user deposits a small amount of funds, the final result of the division operation will be 0 when the amount is too small when withdrawing. Causes the problem that small assets cannot be withdrawn.

Solution

It is recommended to limit the minimum deposit amount.

Status

Ignored

[N5] [Suggestion] The deflationary token issue

Category: Others

Content

In the DepositApprover contract, the amount of the deposit is the amount passed in by the user. If the tokens supported by the protocol become deflationary tokens in the future (for example, USDT enables the transfer fee function), this will cause the actual number of tokens received by the protocol to be inconsistent with the number of dedicated incoming tokens.

The same is true for Controller and SS contracts.

Code location: contracts/core/DepositApprover.sol

```
    function deposit(uint256 amount) public {
        require(IERC20(asset).balanceOf(msg.sender) >= amount,
"INSUFFICIENT_AMOUNT");
        require(IERC20(asset).allowance(msg.sender, address(this)) >= amount,
"INSUFFICIENT_ALLOWANCE");

        IERC20(asset).transferFrom(msg.sender, vault, amount);
        IVault(vault).deposit(amount, msg.sender);
    }
```

Solution

It is recommended to use the difference between the balance of the contract before and after the user's transfer as

the actual deposit amount.

Status

Fixed

[N6] [Critical] Risk of share manipulation

Category: Design Logic Audit

Content

When the user deposits in the agreement, the contract will mint the corresponding share to the user, and when the user withdraws, the corresponding share will be burned. The totalAssets function is used to participate in the calculation when calculating the share, and in the SS contract of the convex, the totalAssets are obtained through the calc_withdraw_one_coin function of the Curve Pool. However, the calc_withdraw_one_coin function is vulnerable to the balance in the Curve Pool, so malicious users can manipulate the calc_withdraw_one_coin function to affect the number of shares minted by the contract.

Code location: contracts/subStrategies/convex/*.sol

```
    function _totalAssets() internal view returns (uint256) {
        if (totalLP == 0) return 0;
        uint256 assets =
ICurvePoolCompound(curvePool).calc_withdraw_one_coin(totalLP, tokenId);
        return assets;
    }
```

Solution

It is recommended to use the get_virtual_price interface for total assets calculation.

Status

Fixed

[N7] [Suggestion] Missing event records

Category: Others

Content

In the vault contract, the owner can modify the maxDeposit, maxWithdraw, controller and depositApprover parameters through the setMaxDeposit, setMaxWithdraw, setController and setDepositApprover functions respectively. But event logging is not used.

In the Controller contract, the owner can modify the vault, apySort, treasury, exchange, withdrawFee, defaultDepositSS and isDefault parameters through the setVault, setAPYSort, setTreasury, setExchange, setWithdrawFee, setDefaultDepositSS and setDefaultOption functions. But event logging is not used.

In the contracts under the exchanges folder, the owner can set the exchange contract address through the setExchange function. But event logging is not used.

In the contracts under the subStrategies/convex folder, the owner can modify the controller, depositSlippage, pId, lpToken, curvePool, harvestGap, maxDeposit, rewardTokens parameters through the setController, setDepositSlippage, setWithdrawSlippage, setPoolId, setLPToken, setCurvePool, setHarvestGap, setMaxDeposit, addRewardToken and removeRewardToken functions. But event logging is not used.

In the cusdc contract, the owner can modify the controller, depositSlippage, withdrawSlippage, harvestGap and maxDeposit parameters through the setController, setDepositSlippage, setWithdrawSlippage, setHarvestGap and setMaxDeposit functions. But event logging is not used.

Code location:

contracts/core/vault.sol

```
    function setMaxDeposit(uint256 _maxDeposit) public onlyOwner {
        require(_maxDeposit > 0, "INVALID_MAX_DEPOSIT");
        maxDeposit = _maxDeposit;
    }

    function setMaxWithdraw(uint256 _maxWithdraw) public onlyOwner {
        require(_maxWithdraw > 0, "INVALID_MAX_WITHDRAW");
```

```
        maxWithdraw = _maxWithdraw;
    }

    function setController(address _controller) public onlyOwner {
        require(_controller != address(0), "INVALID_ZERO_ADDRESS");
        controller = _controller;
    }

    function setDepositApprover(address _approver) public onlyOwner {
        require(_approver != address(0), "INVALID_ZERO_ADDRESS");
        depositApprover = _approver;
    }
```

contracts/core/controller.sol

```
    function setVault(address _vault) public onlyOwner {
        require(_vault != address(0), "INVALID_ADDRESS");
        vault = _vault;
    }

    function setAPYSort(uint256[] memory _apySort) public onlyOwner {
        require(_apySort.length == subStrategies.length, "INVALID_APY_SORT");
        apySort = _apySort;
    }

    function setTreasury(address _treasury) public onlyOwner {
        require(_treasury != address(0), "ZERO_ADDRESS");
        treasury = _treasury;
    }

    function setExchange(address _exchange) public onlyOwner {
        require(_exchange != address(0), "ZERO_ADDRESS");
        exchange = _exchange;
    }

    function setWithdrawFee(uint256 _withdrawFee) public onlyOwner {
        require(_withdrawFee < magnifier, "INVALID_WITHDRAW_FEE");
        withdrawFee = _withdrawFee;
    }

    function setDefaultDepositSS(uint8 _ssId) public onlyOwner {
        require(_ssId < subStrategies.length, "INVALID_SS_ID");
```

```
        defaultDepositSS = _ssId;
    }

    function setDefaultOption(bool _isDefault) public onlyOwner {
        isDefault = _isDefault;
    }
```

contract/exchanges/*.sol (The same goes for all contracts in the exchanges folder)

```
    function setExchange(address _exchange) public onlyOwner {
        require(exchange != address(0), "ZERO_ADDRESS");
        exchange = _exchange;
    }
```

contracts/subStrategies/convex/*.sol (The same goes for all contracts in the convex folder)

```
    function setController(address _controller) public onlyOwner {
        require(_controller != address(0), "INVALID_LP_TOKEN");
        controller = _controller;
    }

    function setDepositSlippage(uint256 _slippage) public onlyOwner {
        require(_slippage < magnifier, "INVALID_SLIPPAGE");

        depositSlippage = _slippage;
    }

    function setWithdrawSlippage(uint256 _slippage) public onlyOwner {
        require(_slippage < magnifier, "INVALID_SLIPPAGE");

        withdrawSlippage = _slippage;
    }

    function setPoolId(uint256 _pId) public onlyOwner {
        require(_pId < IConvexBooster(convex).poolLength(), "INVALID_POOL_ID");
        pId = _pId;
    }

    function setLPToken(address _lpToken) public onlyOwner {
        require(_lpToken != address(0), "INVALID_LP_TOKEN");
        lpToken = _lpToken;
```

```
}

function setCurvePool(address _curvePool) public onlyOwner {
    require(_curvePool != address(0), "INVALID_LP_TOKEN");
    curvePool = _curvePool;
}

function setHarvestGap(uint256 _harvestGap) public onlyOwner {
    require(_harvestGap > 0, "INVALID_HARVEST_GAP");
    harvestGap = _harvestGap;
}

function setMaxDeposit(uint256 _maxDeposit) public onlyOwner {
    require(_maxDeposit > 0, "INVALID_MAX_DEPOSIT");
    maxDeposit = _maxDeposit;
}

function addRewardToken(address _token) public onlyOwner {
    require(_token != address(0), "ZERO_ADDRESS");

    for (uint256 i = 0; i < rewardTokens.length; i++) {
        require(rewardTokens[i] != _token, "DUPLICATE_REWARD_TOKEN");
    }
    rewardTokens.push(_token);
}

function removeRewardToken(address _token) public onlyOwner {
    require(_token != address(0), "ZERO_ADDRESS");

    bool succeed;

    for (uint256 i = 0; i < rewardTokens.length; i++) {
        if (rewardTokens[i] == _token) {
            rewardTokens[i] = rewardTokens[rewardTokens.length - 1];
            rewardTokens.pop();

            succeed = true;
            break;
        }
    }

    require(succeed, "REMOVE_REWARD_TOKEN_FAIL");
}
```

contracts/subStrategies/notional/cusdc.sol

```
    function setController(address _controller) public onlyOwner {
        require(_controller != address(0), "INVALID_LP_TOKEN");
        controller = _controller;
    }

    function setDepositSlippage(uint256 _slippage) public onlyOwner {
        require(_slippage < magnifier, "INVALID_SLIPPAGE");

        depositSlippage = _slippage;
    }

    function setWithdrawSlippage(uint256 _slippage) public onlyOwner {
        require(_slippage < magnifier, "INVALID_SLIPPAGE");

        withdrawSlippage = _slippage;
    }

    function setHarvestGap(uint256 _harvestGap) public onlyOwner {
        require(_harvestGap > 0, "INVALID_HARVEST_GAP");
        harvestGap = _harvestGap;
    }

    function setMaxDeposit(uint256 _maxDeposit) public onlyOwner {
        require(_maxDeposit > 0, "INVALID_MAX_DEPOSIT");
        maxDeposit = _maxDeposit;
    }
```

Solution

It is recommended to record events when sensitive parameters are modified for subsequent self-inspection or community review.

Status

Fixed

[N8] [Low] AllocPoint deposit issue

Category: Design Logic Audit

Content

In the Controller contract, deposits are made according to the allocPoint of each SS, which calculates the number of tokens transferred to each SS through the following algorithm

`amountForSS = (_amount * subStrategies[i].allocPoint) / totalAllocPoint;`

However, due to the loss of precision in the division calculation, a small amount of funds cannot be transferred into SS.

Code location: contracts/core/controller.sol

```
    function _deposit(uint256 _amount) internal returns (uint256 depositAmt) {
        if (isDefault) {
           ...
        } else {
            for (uint256 i = 0; i < subStrategies.length; i++) {
                // Calculate how much to deposit in one sub strategy
                uint256 amountForSS = (_amount * subStrategies[i].allocPoint) /
totalAllocPoint;

                if (amountForSS == 0) continue;

                // Transfer asset to substrategy
                TransferHelper.safeTransfer(address(asset),
subStrategies[i].subStrategy, amountForSS);

                // Calls deposit function on SubStrategy
                uint256 amount =
ISubStrategy(subStrategies[i].subStrategy).deposit(amountForSS);
                depositAmt += amount;
            }
        }
    }
```

Solution

It is recommended to record the remaining number of tokens when calculating the number of tokens transferred into

the SS contract as the funds required to be transferred into the last SS contract.

Status

Fixed

[N9] [Low] check withdrawal amount issue

Category: Design Logic Audit

Content

After the Controller contract withdraws from SS, it will check whether withdrawAmt is greater than 0. But since the protocol will havest periodically, theoretically withdrawAmt should be greater than or equal to the `_amount` parameter passed in by the user.

Code location: contracts/core/controller.sol

```
    function withdraw(uint256 _amount, address _receiver) external override onlyVault
returns (uint256 withdrawAmt) {
        ...

        if (withdrawAmt > 0) {
            require(asset.balanceOf(address(this)) >= withdrawAmt,
"INVALID_WITHDRAWN_AMOUNT");

            // Pay Withdraw Fee to treasury and send rest to user
            uint256 fee = (withdrawAmt * withdrawFee) / magnifier;
            TransferHelper.safeTransfer(address(asset), treasury, fee);

            // Transfer withdrawn token to receiver
            uint256 toReceive = withdrawAmt - fee;
            TransferHelper.safeTransfer(address(asset), _receiver, toReceive);
        }
    }
```

Solution

It is recommended to check whether withdrawAmt is greater than or equal to the `_amount` parameter passed in by the user.

Status

Ignored; After discussion with the project team, we believe that in some cases (such as the impact of slippage), withdrawAmt will be less than _amount, so this issue is ignored.

[N10] [Medium] Risks of fake routers

Category: Design Logic Audit

Content

In the Controller contract, the owner role can compound interest through the harvest function. However, it is not checked whether the router list passed in by owner is as expected. If an unexpected router is passed in, it may lead to failure to harvest normally or loss of funds.

Code location: contracts/core/controller.sol

```
function harvest(
    uint256[] memory _ssIds,
    bytes32[] memory _indexes,
    address[] memory _routers
) public onlyOwner returns (uint256) {
    ...
}
```

Solution

It is recommended to check whether the router is as expected through a whitelist.

Status

Fixed

[N11] [Low] Loss of computational precision

Category: Design Logic Audit

Content

In the vault contract, the convertToAssets function is used to convert shares to corresponding asset amounts.

However, it performs the calculation by performing the division operation first and then the multiplication operation, which will result in loss of calculation accuracy.

Code location: contracts/core/vualt.sol

```
    function convertToAssets(uint256 shares) public view virtual returns (uint256) {
        uint256 supply = totalSupply();

        return supply == 0 ? shares : (shares / supply) * totalAssets();
    }
```

Solution

It is recommended to perform the multiplication operation first and then the division operation for the calculation.

Status

Fixed

[N12] [Critical] Risks of strict equality checks

Category: Design Logic Audit

Content

In Convex's SS contract, when the user makes a withdrawal, it is checked whether the LP balance of the current contract is strictly equal to the LP amount required by the user. If a malicious user intentionally transfers any amount of LP tokens to the current contract, this will cause the SS contract to become unusable.

Code location: contracts/subStrategies/convex/*.sol

```
    function withdraw(uint256 _amount) external override onlyController returns
(uint256) {
        ...
        uint256 lpWithdrawn = IERC20(lpToken).balanceOf(address(this));
        require(lpWithdrawn == lpAmt, "LP_WITHDRAWN_NOT_MATCH");
        ...
    }
```

Solution

It is recommended to check whether the LP balance of the current contract is greater than or equal to the number of LPs required by the user.

Status

Fixed

[N13] [Medium] Negative number check issue

Category: Design Logic Audit

Content

In the Cusdc contract, the `_totalAssets` function is used to obtain the total collateralized assets. It is calculated by multiplying the number of nTokens held by the protocol by the price of nTokens and dividing the total supply of nTokens. The price of nToken is obtained through the getPresentValueUnderlyingDenominated function, but the return value of the getPresentValueUnderlyingDenominated function is int256, while the return value of the INusdc interface is defined as uint256. If it returns a negative number, it will overflow.

Code location:

contracts/subStrategies/notional/interfaces/INusdc.sol

```
interface INusdc {
    function getPresentValueUnderlyingDenominated() external view returns (uint256);
}
```

contracts/subStrategies/notional/cusdc.sol

```
    function _totalAssets() internal view returns (uint256) {
        uint256 nTokenBal = IERC20(nUSDC).balanceOf(address(this));

        uint256 nTokenTotal = IERC20(nUSDC).totalSupply();

        uint256 underlyingDenominated =
INusdc(nUSDC).getPresentValueUnderlyingDenominated();
```

```
        return ((nTokenBal * underlyingDenominated) * usdcDecimal) / noteDecimal /
nTokenTotal;
    }
```

Solution

It is recommended to keep the return value of the getPresentValueUnderlyingDenominated interface consistent with nToken. And check if its return value is greater than 0.

Status

Fixed

[N14] [Medium] Risk of excessive authority

Category: Authority Control Vulnerability

Content

In the Controller contract, the owner can set allocation point of a sub strategy, register the substrategies to the controller contract and withdraw the assets from one SS and deposit to other SS. This will have an impact on the user's deposit and withdrawal operations.

Code location: contracts/core/controller.sol

```
    function moveFund(
        uint256 _fromId,
        uint256 _toId,
        uint256 _amount
    ) public onlyOwner {
        ...
    }

    function setAllocPoint(uint256 _allocPoint, uint256 _ssId) public onlyOwner {
        ...
    }

    function registerSubStrategy(address _subStrategy, uint256 _allocPoint) public
onlyOwner {
```

```
    ...
}
```

Solution

It is recommended to transfer owner ownership to community governance.

Status

Confirmed

[N15] [Suggestion] Code redundancy issue

Category: Others

Content

- In the vault contract, the convertToShares function is defined, but it is not actually used in the contract.

Code location: contracts/core/vault.sol

```
function convertToShares(uint256 assets) public view virtual returns (uint256) {
    uint256 supply = totalSupply();

    return supply == 0 ? assets : (assets * supply) / totalAssets();
}
```

- In the contracts under the exchanges folder, the pathBytes parameter is defined, and it is not added during the addPath operation, but its index is removed in the removePath function.

Code location: contract/exchanges/*.sol (The same goes for all contracts in the exchanges folder)

```
function removePath(bytes32 index) public onlyOwner {
    ...
    // Remove index in the list
    for (uint256 i = 0; i < pathBytes.length; i++) {
        if (pathBytes[i] == index) {
            pathBytes[i] = pathBytes[pathBytes.length - 1];
            pathBytes.pop();
            break;
```

```
            }
        }
        ...
    }
```

Solution

- In case of unexpected design, it is recommended to remove redundant code.
- In case of unexpected design, it is recommended to remove the code related to changing pathBytes in the removePath function.

Status

Fixed

[N16] [Medium] Invalid minimum output calculation

Category: Design Logic Audit

Content

In the SS contract, the calc_token_amount function will be used to calculate the minimum amount of LP tokens received during the deposit operation; the minimum amount of staking tokens received will be calculated through the calc_withdraw_one_coin function during the withdrawal operation. However, the calc_token_amount function and the calc_withdraw_one_coin function are easily affected by the last transaction of CurvePool, so they cannot play the role of slippage protection.

Lusd and Tri contracts also have slippage issue, but the slippage check is annotated in the deposit function.

Code location: contracts/subStrategies/convex/*.sol

```
    function _deposit(uint256 _amount) internal returns (uint256) {
        …

        uint256 expectOutput = ICurvePoolAave(curvePool).calc_token_amount(amounts,
true);

        // Calculate Minimum output considering slippage
```

```
        uint256 minOutput = (expectOutput * (magnifier - depositSlippage)) /
magnifier;

        // Add liquidity to Curve pool
        ICurvePoolAave(curvePool).add_liquidity(amounts, minOutput, true);

        ...
    }

    function withdraw(uint256 _amount) external override onlyController returns
(uint256) {
        ...
        uint256 minAmt =
ICurvePoolAave(curvePool).calc_withdraw_one_coin(lpWithdrawn, tokenId);
        minAmt = (minAmt * (magnifier - withdrawSlippage)) / magnifier;

        IERC20(lpToken).approve(curvePool, lpWithdrawn);

        ICurvePoolAave(curvePool).remove_liquidity_one_coin(lpWithdrawn, tokenId,
minAmt, true);

        ...
    }
```

Solution

It is recommended to use the get_virtual_price function for indirect calculations.

Status

Fixed

[N17] [Low] Risk of pid acquisition

Category: Design Logic Audit

Content

In the Alusd contract, the getPID function user obtains the corresponding LP pool address in the ConvexBooster contract. It will return 0 if LpToken does not exist, but pid0 has a value in the ConvexBooster contract. So when getPID returns 0, it will be hard to tell if pid exists.

Code location: contracts/subStrategies/convex/Alusd.sol

```
    function getPID(address _lpToken) public view returns (uint256) {
        for (uint256 i = 0; i < IConvexBooster(convex).poolLength(); i++) {
            (address lpToken_, , , , , ) = IConvexBooster(convex).poolInfo(i);

            if (lpToken_ == _lpToken) return i;
        }
        return 0;
    }
```

Solution

When LpToken does not exist, it is recommended to return ~uint256(0)

Status

Fixed

[N18] [Suggestion] Redundant approval issue

Category: Others

Content

In the swapExactTokenInput function of the Exchange contract, it will first transfer the tokens that need to be swapped from the controller contract to the router contract. But the swapExactTokenInput function approves the router contract again, which is unnecessary.

Code location: contracts/core/Exchange.sol

```
    function swapExactTokenInput(
        address _from,
        address _to,
        address _router,
        bytes32 _index,
        uint256 _amount
    ) external override onlyController returns (uint256) {
        // Transfer token from controller
        TransferHelper.safeTransferFrom(_from, controller, address(_router),
_amount);
```

```
    // Approve token to router
    IERC20(_from).approve(_router, 0);
    IERC20(_from).approve(_router, _amount);
    ...
}
```

Solution

It is recommended to remove redundant logic.

Status

Fixed

[N19] [Medium] Incorrect storage of temporary variables

Category: Design Logic Audit

Content

In the router contract, the removePath function is used to remove the swap path recorded in the contract. It will first store the balancerBatchAssets variable through storage, then delete it, and then use this variable for event recording after deletion.

Code location: contracts/exchanges/*.sol

```
function removePath(bytes32 index) public onlyOwner {
    require(balancerBatchAssets[index].length != 0, "NON_EXIST_PATH");

    // TempSave for assets info
    IAsset[] storage assets = balancerBatchAssets[index];

    // Delete path record from mapping
    delete balancerBatchAssets[index];
    delete poolBatchIds[index];

    // Remove index in the list
    for (uint256 i = 0; i < pathBytes.length; i++) {
        if (pathBytes[i] == index) {
            pathBytes[i] = pathBytes[pathBytes.length - 1];
            pathBytes.pop();
            break;
```

```
            }
        }

        emit RemoveBalancerBatchSwap(index, assets);
    }
```

Solution

It is recommended to use memory for storage.

Status

Fixed

[N20] [Medium] Compound interest slippage check issue

Category: Design Logic Audit

Content

In the router contract, no slippage check is performed during the swap operation. If there are more funds with compound interest, there will be a risk of being attacked by sandwiches.

Code location: contracts/exchanges/*.sol

```
    function swap(
        address _from,
        address _to,
        bytes32 _index,
        uint256 _amount
    ) external override onlyExchange {
        ...

        limits[0] = int256(_amount);
        for (uint256 i = 1; i < length; i++) {
            limits[i] = int256(0);
        }

        ...
    }
```

```
    function swap(
        address _from,
        address _to,
        bytes32 _index,
        uint256 _amount
    ) external override onlyExchange {
        ...

        uint256 limit = 0;

        ...
    }

    function swap(
        address _from,
        address _to,
        bytes32 _index,
        uint256 _amount
    ) external override onlyExchange {
        ...

        if (_to == weth) ICurvePoolToETH(curve.pool).exchange(curve.i, curve.j,
_amount, 0, true);
        else ICurvePool(curve.pool).exchange_underlying(curve.i, curve.j, _amount,
0);

        ...
    }

    function swap(
        address _from,
        address _to,
        bytes32 _index,
        uint256 _amount
    ) external override onlyExchange {
        ...

        if (_to == weth) {
            // If target token is Weth

IUniswapV2Router(router).swapExactTokensForETHSupportingFeeOnTransferTokens(
                _amount,
                0,
                paths[_index].path,
```

```
                address(exchange),
                block.timestamp + 3600
            );
        } else if (_from == weth) {

IUniswapV2Router(router).swapExactETHForTokensSupportingFeeOnTransferTokens{value:
_amount}(
                0,
                paths[_index].path,
                address(exchange),
                block.timestamp + 3600
            );
        } else {

IUniswapV2Router(router).swapExactTokensForTokensSupportingFeeOnTransferTokens(
                _amount,
                0,
                paths[_index].path,
                address(exchange),
                block.timestamp + 3600
            );
        }
    }

    function swap(
        address _from,
        address _to,
        bytes32 _index,
        uint256 _amount
    ) external override {
        ...

        IUniswapV3Router.ExactInputSingleParams memory params =
IUniswapV3Router.ExactInputSingleParams({
            tokenIn: _from,
            tokenOut: _to,
            fee: _path.fee,
            recipient: address(this),
            amountIn: _amount,
            amountOutMinimum: 0,
            sqrtPriceLimitX96: 0
        });
```

```
    ...
}
```

Solution

If there are more funds with compound interest, it is recommended to perform a slippage check.

Status

Confirmed

[N21] [Medium] Lack of access control

Category: Authority Control Vulnerability

Content

The swap function in the UniswapV3 contract is not subject to permission control, which will allow any user to call it.

Code location: contracts/exchanges/UniswapV3.sol

```
function swap(
    address _from,
    address _to,
    bytes32 _index,
    uint256 _amount
) external override {
...
}
```

Solution

It is recommended to only allow calls from Exchange contracts.

Status

Fixed

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002209140001	SlowMist Security Team	2022.09.05 - 2022.09.14	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 3 critical risks, 7 medium risks, 7 low risks, 4 suggestions. And 2 medium risks, 1 low risk were confirmed; 2 low risk vulnerabilities were ignored; All other findings were fixed. The code was not deployed to the mainnet.

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