Smart Contract Security Audit Report

[2021]

Table Of Contents

1 Executive Summary

On 2021.03.31, the SlowMist security team received the Layer2 Finance team's security audit application for Layer2 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 party 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 an alysis tools.

Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problem s.

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

- Reentrancy Vulnerability
- Replay Vulnerability
- Reordering Vulnerability
- Short Address Vulnerability
- Denial of Service Vulnerability
- Transaction Ordering Dependence Vulnerability
- Race Conditions Vulnerability
- Authority Control Vulnerability
- Integer Overflow and Underflow Vulnerability
- TimeStamp Dependence Vulnerability
- Uninitialized Storage Pointers Vulnerability
- Arithmetic Accuracy Deviation Vulnerability
- tx.origin Authentication Vulnerability
- "False top-up" Vulnerability
- Variable Coverage Vulnerability
- Gas Optimization Audit
- Malicious Event Log Audit
- Redundant Fallback Function Audit
- Unsafe External Call Audit
- Explicit Visibility of Functions State Variables Aduit

- Design Logic Audit
- Scoping and Declarations Audit

3 Project Overview

3.1 Project Introduction

Layer2.finance is a Celer Network project that aims to bring mass audiences to the existing Decentralized Finance (DeFi) ecosystem with layer-2 scaling technology. It acts as a low-cost and trust-free gateway for the “early majority” users to explore and benefit from the existing DeFi ecosystem without the concerns of high transaction (gas) costs offsetting their gains. It enables quadratic scaling of the existing layer-1 DeFi ecosystem “in-place” with no protocol migration needed and therefore, does not cause liquidity fragmentation or break composability. It achieves this using an optimized layer-2 rollup construct to aggregate N small-fish users’ fund allocation transactions on layer-2 into a single one on layer-1 in a trust-free fashion.

Initial audit files:

https://github.com/celer-network/layer2-finance-contracts

commit: adb3711bf526322bdf437c5db47e3b17446cd398

Final audit files:

https://github.com/celer-network/layer2-finance-contracts

commit: bdaf438c90871ea9f47bc6f2f13cf210baadf95d

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Slippage check is not performed when compounding interest	Design Logic Audit	Medium	Confirmed

NO	Title	Category	Level	Status
N2	Slippage check issue	Design Logic Audit	Suggestion	Confirming
N3	Event missing issue	Others	Suggestion	Confirming
N4	Risk of excessive authority	Authority Control Vulnerability	Medium	Confirming

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:

Registry			
Function Name	Visibility	Mutability	Modifiers
registerAsset	External	Can Modify State	onlyOwner
registerStrategy	External	Can Modify State	onlyOwner

RollupChain			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
<Fallback>	External	Payable	-

RollupChain			
<Receive Ether>	External	Payable	-
deposit	External	Can Modify State	whenNotPaused
depositETH	External	Payable	whenNotPaused
withdraw	External	Can Modify State	whenNotPaused
withdrawETH	External	Can Modify State	whenNotPaused
commitBlock	External	Can Modify State	whenNotPaused onlyOperator
executeBlock	External	Can Modify State	whenNotPaused
syncBalance	External	Can Modify State	whenNotPaused onlyOperator
disputeTransition	External	Can Modify State	-
disputePriorityTxDelay	External	Can Modify State	-
pause	External	Can Modify State	onlyOwner
unpause	External	Can Modify State	onlyOwner
drainToken	External	Can Modify State	whenPaused onlyOwner
drainETH	External	Can Modify State	whenPaused onlyOwner
setBlockChallengePeriod	External	Can Modify State	onlyOwner
setMaxPriorityTxDelay	External	Can Modify State	onlyOwner
setOperator	External	Can Modify State	onlyOwner
setNetDepositLimit	External	Can Modify State	onlyOwner
getCurrentBlockId	Public	-	-
_deposit	Private	Can Modify State	-

RollupChain

_withdraw	Private	Can Modify State	-
_revertBlock	Private	Can Modify State	-

TransitionDisputer

Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
disputeTransition	External	Can Modify State	-
_evaluateInvalidTransition	Private	Can Modify State	-
_getStateRootsAndIds	Private	Can Modify State	-
_invalidInitTransition	Private	Can Modify State	-
_getAccountInfoBytes	Private	-	-
_getStrategyInfoBytes	Private	-	-
_verifySequentialTransitions	Private	-	-
_checkTransitionInclusion	Private	-	-
_checkTwoTreeStateRoot	Private	-	-
_verifyProofInclusion	Private	-	-
_updateAndVerify	Private	-	-

TransitionEvaluator

Function Name	Visibility	Mutability	Modifiers
evaluateTransition	External	-	-

TransitionEvaluator

getTransitionStateRootAndAccessIds	External	-	-
_applyDepositTransition	Private	-	-
_applyWithdrawTransition	Private	-	-
_applyCommitTransition	Private	-	-
_applyUncommitTransition	Private	-	-
_applyCommitmentSyncTransition	Private	-	-
_applyBalanceSyncTransition	Private	-	-
_getAccountInfoHash	Private	-	-
_getStrategyInfoHash	Private	-	-

StrategyCompoundErc20LendingPool

Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
getAssetAddress	External	-	-
syncBalance	External	Can Modify State	-
harvest	External	Can Modify State	-
aggregateCommit	External	Can Modify State	-
aggregateUncommit	External	Can Modify State	-
setController	External	Can Modify State	onlyOwner

StrategyCompoundEthLendingPool

StrategyCompoundEthLendingPool			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
getAssetAddress	External	-	-
syncBalance	External	Can Modify State	-
harvest	External	Can Modify State	-
aggregateCommit	External	Can Modify State	-
aggregateUncommit	External	Can Modify State	-
setController	External	Can Modify State	onlyOwner
<Receive Ether>	External	Payable	-
<Fallback>	External	Payable	-

StrategyAaveLendingPool			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
getAssetAddress	External	-	-
syncBalance	External	-	-
harvest	External	Can Modify State	-
aggregateCommit	External	Can Modify State	-
aggregateUncommit	External	Can Modify State	-
setController	External	Can Modify State	onlyOwner

StrategyCurve3PoolDAI			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
getAssetAddress	External	-	-
syncBalance	External	-	-
harvest	External	Can Modify State	-
aggregateCommit	External	Can Modify State	-
aggregateUncommit	External	Can Modify State	-
setController	External	Can Modify State	onlyOwner

4.3 Vulnerability Summary

[N1] [Medium] Slippage check is not performed when compounding interest

Category: Design Logic Audit

Content

In a strategy contract, any user can use the harvest function to make the strategy pool perform compound interest operations. The strategy pool will sell the rewarded tokens through DEX, but the slippage check is not performed at the time of sale, which will cause malicious users to first increase the slippage of the DEX liquidity pool through flashloan, and then call the harvest function of the strategy pool. And finally restore the liquidity pool to normal, causing the profit of the strategy pool to be maliciously evacuated. Sandwich attacks are equally effective.

Code location: Strategy* contract

```
function harvest() external override {
    // Claim COMP token.
    IComptroller(comptroller).claimComp(address(this));
```

```
        uint256 compBalance = IERC20(comp).balanceOf(address(this));
        if (compBalance > 0) {
            // Sell COMP token for obtain more ETH
            IERC20(comp).safeIncreaseAllowance(uniswap, compBalance);

            address[] memory paths = new address[](2);
            paths[0] = comp;
            paths[1] = weth;

            IUniswapV2(uniswap).swapExactTokensForETH(
                compBalance,
                uint256(0),
                paths,
                address(this),
                block.timestamp.add(1800)
            );

            // Deposit ETH to Compound ETH Lending Pool and mint cETH.
            uint256 obtainedEthAmount = address(this).balance;
            ICEth(cEth).mint{value: obtainedEthAmount}();
        }
    }
```

Solution

It is recommended that the harvest function of the restricted strategy pool can only be called by the upper-level contract. And check slippage during swap operation.

Status

Confirmed; After communicating with the project team, the project team stated that it would limit the harvest function to be called only by the EOA account to avoid the risk of flashloan, but the risk of sandwich attacks still exists.

[N2] [Suggestion] Slippage check issue

Category: Design Logic Audit

Content

In the StrategyCurve3PoolDAI contract, it will transfer funds to Curve's 3Pool, and perform slippage checks when adding liquidity and removing liquidity. The slippage check is fixed at 1%. Because the strategy pool has a large amount of funds and the strategic target is a stablecoin pool, there is still room for arbitrage at 1% slippage, and there is a risk of being attacked by a sandwich.

Code location: StrategyCurve3PoolDAI.sol

```
    function aggregateCommit(uint256 _daiAmount) external override {
        require(msg.sender == controller, "Not controller");
        require(_daiAmount > 0, "Nothing to commit");

        // Pull DAI from Controller
        IERC20(dai).safeTransferFrom(msg.sender, address(this), _daiAmount);

        // Deposit DAI to 3Pool
        IERC20(dai).safeIncreaseAllowance(triPool, _daiAmount);
        uint256 minMintAmount = _daiAmount.mul(1e18).div(ICurveFi(triPool).get_virtual_price());
        ICurveFi(triPool).add_liquidity(
            [_daiAmount, 0, 0],
            minMintAmount.mul(DENOMINATOR.sub(slippage)).div(DENOMINATOR)
        );

        // Stake 3CRV in Gauge to farm CRV
        uint256 triCrvBalance = IERC20(triCrv).balanceOf(address(this));
        IERC20(triCrv).safeIncreaseAllowance(gauge, triCrvBalance);
        IGauge(gauge).deposit(triCrvBalance);

        emit Committed(_daiAmount);
    }
```

Solution

It is recommended to refer to the yDAI strategy pool and use 0.05% slippage check.

Status

Confirming

[N3] [Suggestion] Event missing issue

Category: Others

Content

In the strategy pool contract, the owner can set the controller address through the setController function, but the event is not recorded.

Code location:

```
    function setController(address _controller) external onlyOwner {
        controller = _controller;
    }
```

Solution

It is recommended to record incidents when modifying sensitive parameters for follow-up review.

Status

Confirming

[N4] [Medium] Risk of excessive authority

Category: Authority Control Vulnerability

Content

In the strategy pool contract, the owner can set the controller address through the setController function. Since the controller involves the destination of funds when withdrawing, this will lead to the problem of excessive owner authority

Code location:

```
    function setController(address _controller) external onlyOwner {
        controller = _controller;
    }
```

Solution

It is recommended to transfer the owner to community governance.

Status

Confirming

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0x002104160003	SlowMist Security Team	2021.03.31 - 2021.04.15	Medium Risk

Summary conclusion: The SlowMist security team use a manual and SlowMist team's analysis tool to audit the project, during the audit work we found 2 medium risk, 2 suggestion vulnerabilities. And 1 medium risk vulnerabilities were confirmed and being fixed; 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.

SLOWMIST

Official Website

www.slowmist.com

✉

E-mail

team@slowmist.com

Twitter

@SlowMist_Team

Github

https://github.com/slowmist