SLOWMIST

Smart Contract

Security Audit Report

Table Of Contents

1 Executive Summary

2 Audit Methodology

3 Project Overview

3.1 Project Introduction

3.2 Vulnerability Information

4 Code Overview

4.1 Contracts Description

4.2 Visibility Description

4.3 Vulnerability Summary

5 Audit Result

6 Statement

1 Executive Summary

On 2022.12.13, the SlowMist security team received the Earning.Farm team's security audit application for ENF_ETH_Lowrisk, 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_ETH_Lowrisk

commit: f050de5cc4096502e588d6befa2aeedd3d1115b8

Fixed Version:

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

commit: c3f5e5ef17595228e1fdd5074f789593fa759a34

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Defects in the defaultDepositSS check	Design Logic Audit	Medium	Acknowledged
N2	ownerDeposit remaining deposit issue	Design Logic Audit	Suggestion	Fixed
N3	Redundant variable	Others	Suggestion	Fixed
N4	Risk of excessive authority	Authority Control Vulnerability	Medium	Acknowledged
N5	Compound interest slippage check issue	Design Logic Audit	Low	Acknowledged

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:

StETH			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	initializer
<Receive Ether>	External	Payable	-
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	Payable	onlyOwner
withdrawable	External	-	-
setController	Public	Can Modify State	onlyOwner

StETH			
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

CEth			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	initializer
<Receive Ether>	External	Payable	-
totalAssets	External	-	-
_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

CEth			
emergencyWithdraw	Public	Can Modify State	onlyOwner
withdrawable	External	-	-
ownerDeposit	Public	Payable	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] [Medium] Defects in the defaultDepositSS check

Category: Design Logic Audit

Content

In the harvest function of the Controller contract, before re-depositing the protocol income into the strategy, it will check whether the default SS exists through `subStrategies.length > defaultDepositSS`. But actually, defaultDepositSS will be 0 when the default SS does not exist, so the `subStrategies.length > defaultDepositSS` check will always pass. Eventually the protocol will fail to re-deposit.

Code location: contracts/core/Controller.sol

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

        // Check Such default SS exists in current pool
        require(subStrategies.length > defaultDepositSS, "INVALID_POOL_LENGTH");

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

        // Calls deposit function on SubStrategy
        ISubStrategy(subStrategies[defaultDepositSS].subStrategy).deposit(toDeposit);

       ...
    }
```

Solution

It is recommended to check whether the default SS is enabled through the isDefault parameter.

Status

Acknowledged; After communicating with the project team, the project team stated that it is what they expected, which means, they set defaultDepositSS as the first one for default, so that even though they don't set it manually, it deposits to first SS automatically.

[N2] [Suggestion] ownerDeposit remaining deposit issue

Category: Design Logic Audit

Content

In the ownerDeposit function of the StETH contract, the owner role will directly deposit ETH into the strategy. It checks that `msg.value` must be greater than or equal to the amount to be deposited through `_amount <= msg.value`. But when the owner's `msg.value` is greater than `_amount`, the ownerDeposit function does not implement the refund of excess ETH. This will result in funds being locked.

The same is true for the ownerDeposit function of the CEth contract.

Code location: contracts/subStrtegies/convex/StETH.sol

```
    function ownerDeposit(uint256 _amount) public payable onlyOwner {
        require(_amount <= msg.value, "INSUFFICIENT_ETH");

        // Call deposit
        _deposit(_amount);

        emit OwnerDeposit(_amount);
    }
```

Solution

It is recommended to check the deposit amount with `_amount == msg.value`.

Status

Fixed

[N3] [Suggestion] Redundant variable

Category: Others

Content

There is a weth global variable in the CEth contract, but this variable is not used in the contract.

Code location: contracts/subStrtegies/notional/CEth.sol

```
    address public weth;
```

Solution

If the design is not intended, it is recommended to remove redundant variables.

Status

Fixed

[N4] [Medium] Risk of excessive authority

Category: Authority Control Vulnerability

Content

In the protocol, the owner role has many permissions, such as: the owner can set sensitive parameters, can suspend the contract, can make emergency withdrawals, can migrate the funds of the SS contract, etc. It is obviously inappropriate to give all the permissions of the protocol to the owner, which will greatly increase the single point of risk.

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. And 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.

Status

Acknowledged

[N5] [Low] 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 {
    // Check Path from and to
    require(pathFrom(_index) == _from, "INVALID_FROM_ADDRESS");
    require(pathTo(_index) == _to, "INVALID_TO_ADDRESS");
```

```
        uint256 balance = getBalance(_from, address(this));
        require(balance >= _amount, "INSUFFICIENT_TOKEN_TRANSFERED");

        // Get Curve Pool address
        CurvePool memory curve = pools[_index];

        address initial = _from == weth ? NULL_ADDR : _from;
        address to = _to == weth ? NULL_ADDR : _to;
        (address[6] memory _route, uint256[8] memory _indices, uint256 _min_received)
= ICurve3Pool(curve.pool)
            .get_exchange_routing(initial, to, _amount);

        if (_from != weth) {
            // Approve token
            IERC20(_from).approve(curve.pool, 0);
            IERC20(_from).approve(curve.pool, _amount);

            // Call Exchange
            ICurve3Pool(curve.pool).exchange(_amount, _route, _indices, 0,
address(this));
        } else {
            // Call Exchange
            ICurve3Pool(curve.pool).exchange{value: _amount}(_amount, _route,
_indices, 0, address(this));
        }

        uint256 out = getBalance(_to, address(this));

        // If toTOken is weth, withdraw ETH from it
        if (_to == weth) {
            TransferHelper.safeTransferETH(exchange, out);
        } else {
            // Transfer output token to exchnage
            TransferHelper.safeTransfer(_to, exchange, out);
        }
    }
```

Solution

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

Status

Acknowledged

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002212160003	SlowMist Security Team	2022.12.13 - 2022.12.16	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 2 medium risks, 1 low-risk, and 2 suggestions. All the findings were fixed or acknowledged. 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