Smart Contract
Security Audit Report

Table Of Contents

1 Executive Summary

On 2022.10.31, the SlowMist security team received the Earning.Farm team's security audit application for Earning Farm - ETH Leverage, 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_Leverage

commit: ebc757f3d78c84800a4fb46285f5dfe43c1568f1

Fixed Version:

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

commit: 06a04c5d05ec19dfe3cf00303d52148e859746bd

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Lack of access control issue	Authority Control Vulnerability	Suggestion	Fixed
N2	Gas optimization	Gas Optimization Audit	Suggestion	Fixed
N3	Redundant logic issue	Others	Suggestion	Fixed
N4	The problem of checking the number of swaps	Design Logic Audit	Low	Fixed
N5	Risk of excessive authority	Authority Control Vulnerability	Medium	Confirmed
N6	Risk of exchange slippage	Design Logic Audit	Medium	Confirmed

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
<Receive Ether>	External	Payable	-
initialize	Public	Can Modify State	initializer
deposit	Public	Payable	nonReentrant unPaused
mint	External	Can Modify State	onlySS
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

EFVault			
setSubStrategy	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
<Receive Ether>	External	Payable	-
deposit	External	Can Modify State	onlyVault
_deposit	Internal	Can Modify State	-
withdraw	External	Can Modify State	onlyVault
withdrawable	Public	-	-
getBalance	Internal	-	-
moveFund	Public	Can Modify State	onlyOwner
totalAssets	External	-	-
_totalAssets	Internal	-	-
subStrategyLength	External	-	-
setVault	Public	Can Modify State	onlyOwner
setAPYSort	Public	Can Modify State	onlyOwner
setTreasury	Public	Can Modify State	onlyOwner

Controller			
setExchange	Public	Can Modify State	onlyOwner
setWithdrawFee	Public	Can Modify State	onlyOwner
setHarvestFee	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

ETHLeverExchange			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	initializer
<Receive Ether>	External	Payable	-
swapStETH	External	Can Modify State	onlyLeverSS
swapETH	External	Can Modify State	onlyLeverSS
swapExactETH	External	Can Modify State	onlyLeverSS

BalancerReceiver			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	initializer
<Receive Ether>	External	Payable	-
getFee	External	-	-

BalancerReceiver			
flashLoan	External	Can Modify State	loanProcess
receiveFlashLoan	Public	Payable	-

ETHLeverage			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	initializer
<Receive Ether>	External	Payable	-
loanFallback	External	Can Modify State	onlyReceiver
totalAssets	External	-	-
_totalAssets	Internal	-	-
deposit	External	Can Modify State	onlyController onDeposit
_deposit	Internal	Can Modify State	-
withdraw	External	Can Modify State	onlyController onWithdraw
_harvest	Internal	Can Modify State	-
raiseLTV	Public	Can Modify State	onlyOwner
reduceLTV	Public	Can Modify State	onlyOwner onWithdraw
emergencyWithdraw	Public	Can Modify State	onlyOwner onEmergencyWithdraw
withdrawable	External	-	-
ownerDeposit	Public	Payable	onlyOwner onDeposit
getCollateral	Public	-	-

ETHLeverage			
getDebt	Public	-	-
setController	Public	Can Modify State	onlyOwner
setVault	Public	Can Modify State	onlyOwner
setFeePool	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
setFlashLoanReceiver	Public	Can Modify State	onlyOwner
setExchange	Public	Can Modify State	onlyOwner
setBlockRate	Public	Can Modify State	onlyOwner
setMLR	Public	Can Modify State	onlyOwner

4.3 Vulnerability Summary

[N1] [Suggestion] Lack of access control issue

Category: Authority Control Vulnerability

Content

In the BalancerReceiver contract, the SS contract can initiate WETH flash loans through the flashLoan function, but the flashLoan function allows any user to call. Although the loanFallback function of the SS contract checks curState, it is undoubtedly more expected that the flashLoan function can only be called by the SS contract.

Code location: contracts/subStrategies/loanReceivers/BalancerReceiver.sol

```
    function flashLoan(address token, uint256 amount) external override loanProcess {
        ...
    }
```

Solution

It is recommended to restrict the flashLoan function to only be called by the SS contract.

Status

Fixed

[N2] [Suggestion] Gas optimization

Category: Gas Optimization Audit

Content

In the ETHLeverage contract, the `_harvest` function is used to collect fees, which will only be charged when `lastEarnBlock` and `block.number` are used. But the function does not check whether the difference between `lastEarnBlock` and `block.number` is 0. If multiple users in the same block trigger the `_harvest` function, it will cause unnecessary gas consumption.

Code location: contracts/subStrategies/ETH_Leverage.sol

```
    function _harvest() internal {
        if (_totalAssets() == 0) {
            lastEarnBlock = block.number;
            return;
        }

        uint256 collapsed = block.number - lastEarnBlock;
        ...
    }
```

Solution

It is recommended to return the function when the difference between `lastEarnBlock` and `block.number` is 0.

Status

Fixed

[N3] [Suggestion] Redundant logic issue

Category: Others

Content

In the Controller contract, the owner can set the exchange and harvestFee parameters respectively through the setExchange and setHarvestFee functions. But in this contract the exchange and harvestFee parameters are not used.

Code location: contracts/core/Controller.sol

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

        emit SetExchange(exchange);
    }

    function setHarvestFee(uint256 _harvestFee) public onlyOwner {
        require(_harvestFee < magnifier, "INVALID_Harvest_FEE");
        harvestFee = _harvestFee;

        emit SetHarvestFee(harvestFee);
    }
```

Solution

It is recommended to remove redundant logic.

Status

Fixed

[N4] [Low] The problem of checking the number of swaps

Category: Design Logic Audit

Content

In the ETHLeverExchange contract, the swapExactETH function is used to exchange stETH to ETH during emergency withdrawal. It will get the amount of ETH that can be exchanged through the get_dy function and check if the swap amount is larger than the expected required amount. But in theory it is acceptable for the number of swaps to be equal to what is expected to be required.

Code location: contracts/subStrategies/exchange/Exchange.sol

```
    function swapExactETH(uint256 input, uint256 output) external override
onlyLeverSS {
        ...
        uint256 ethOut = ICurve(curvePool).get_dy(1, 0, input);
        require(ethOut > output, "EXTREME_MARKET");

        ...
    }
```

Solution

It is recommended to change `ethOut > output` to `ethOut >= output`

Status

Fixed

[N5] [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

Confirmed

[N6] [Medium] Risk of exchange slippage

Category: Design Logic Audit

Content

When users make withdrawals in the protocol, they need to exchange stETH tokens for ETH tokens through CurvePool. However, the exchange slippage is not limited in the ETHLeverExchange contract, which will make users vulnerable to sandwich attacks when withdrawing.

Code location: contracts/subStrategies/exchange/Exchange.sol

```
    function swapETH(uint256 amount) external override onlyLeverSS {
        require(IERC20(stETH).balanceOf(address(this)) >= amount,
"INSUFFICIENT_STETH");

        // Approve STETH to curve
        IERC20(stETH).approve(curvePool, 0);
        IERC20(stETH).approve(curvePool, amount);
        ICurve(curvePool).exchange(1, 0, amount, 0);

        uint256 ethBal = address(this).balance;

        // Transfer STETH to LeveraSS
```

```
        TransferHelper.safeTransferETH(leverSS, ethBal);
    }
```

Solution

It is recommended to perform a slippage check on CurvePool when withdrawing.

Status

Confirmed

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002211080001	SlowMist Security Team	2022.10.31 - 2022.11.08	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 3 suggestion vulnerabilities. And 2 medium-risk vulnerabilities were confirmed; 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