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 2023.02.01, the SlowMist security team received the Earning.Farm team's security audit application for Earning.Farm Phase5, 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

Serial Number	Audit Class	Audit Subclass
		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

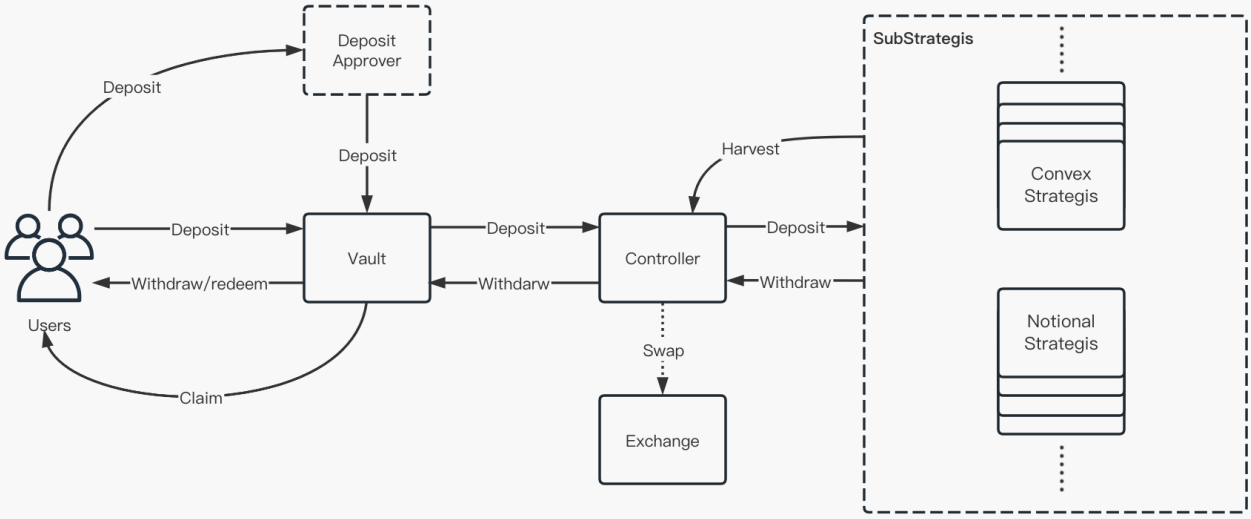
3.1 Project Introduction

Vision of Earning.Farm is to provide user-friendly investment tools for mass population to enjoy the innovation of DEFI.

This time it is an iterative audit of Earning.Farm's ENFv3, ENF_lowrisk_farm, and ENF_lowrisk_ETH_farm products. These three products have a similar architecture, as shown in the figure below, the protocol architecture is mainly divided into four parts: Vault, Controller, Exchange, and Strategies.

The Vault part is used to interact with users. Users can directly deposit, withdraw, and claim in Vault, or

indirectly deposit through DepositApprover. The Vault contract will transfer the user's deposit to the Controller contract, and the Controller contract will deposit funds into each strategy according to the configuration. The Owner role will regularly perform harvest operations on each strategy through the Controller contract to perform compound interest or directly issue rewards to users. The Exchange module is used to assist the token swap operation necessary for the operation of the protocol. In ENF_lowrisk_farm and ENF_lowrisk_ETH_farm, users can claim reward tokens through the Vault module.



3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Redundant Code Usage	Others	Suggestion	Acknowledged
N2	Potential Precision Calculation Issue	Others	Suggestion	Acknowledged
N3	Business logic flaws in reward distribution	Design Logic Audit	Medium	Fixed
N4	Token Transfer Missing Rewards Update	Design Logic Audit	Critical	Fixed
N5	Issue with checking on fromToken	Design Logic Audit	Suggestion	Acknowledged

NO	Title	Category	Level	Status
N6	Incorrect reward receiving address	Design Logic Audit	Low	Fixed
N7	Direct distribution of rewards is not available	Design Logic Audit	Suggestion	Fixed
N8	Risk of price manipulation	Design Logic Audit	Critical	Fixed

4 Code Overview

4.1 Contracts Description

Codebase:

Audit Version:

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

commit: 500611872e25f55e11ce3ef098979ae79956a37e

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

commit: e9a6abd9a2daa1a4e2dc5d990c340bc63fdb5bf6

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

commit: 4a347d89c2ba0cd00e5f400e86d5ed25ab083ebb

Fixed Version:

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

commit: 52ef94ea680448c78a40bbd5772d5ec51055ceb4

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

commit: 4a845bfd4c4b9bd3086ff8a828bf5b141631ec87

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:

ENFv3 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
redeem	Public	Can Modify State	nonReentrant unPaused
_withdraw	Internal	Can Modify State	-
assetsPerShare	Internal	-	-
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

ENF_lowrisk_ETH_farm EFVault			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	initializer

ENF_lowrisk_ETH_farm EFVault			
<Receive Ether>	External	Payable	-
deposit	Public	Payable	nonReentrant unPaused
getBalance	Internal	-	-
withdraw	Public	Can Modify State	nonReentrant unPaused
redeem	Public	Can Modify State	nonReentrant unPaused
_withdraw	Internal	Can Modify State	-
_updateUserData	Internal	Can Modify State	-
pendingReward	Public	-	-
claim	Public	Can Modify State	-
_claim	Internal	Can Modify State	-
_swap	Internal	Can Modify State	-
assetsPerShare	Internal	-	-
_getTokenIndex	Internal	-	-
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
setExchange	Public	Can Modify State	onlyOwner
setSwapPath	Public	Can Modify State	onlyOwner

ENF_lowrisk_ETH_farm EFVault			
pause	Public	Can Modify State	onlyOwner
resume	Public	Can Modify State	onlyOwner

ENF_lowrisk_ETH_farm 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	-	-
harvest	Public	Can Modify State	onlyOwner
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
setExchange	Public	Can Modify State	onlyOwner
setWithdrawFee	Public	Can Modify State	onlyOwner
setHarvestFee	Public	Can Modify State	onlyOwner

ENF_lowrisk_ETH_farm Controller			
setAllocPoint	Public	Can Modify State	onlyOwner
registerSubStrategy	Public	Can Modify State	onlyOwner
setDefaultDepositSS	Public	Can Modify State	onlyOwner
setDefaultOption	Public	Can Modify State	onlyOwner
addRewardToken	Public	Can Modify State	onlyOwner
removeRewardToken	Public	Can Modify State	onlyOwner
getRewardInfo	Public	-	-

ENF_lowrisk_farm EFVault			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	initializer
<Receive Ether>	External	Payable	-
deposit	Public	Can Modify State	nonReentrant unPaused
getBalance	Internal	-	-
withdraw	Public	Can Modify State	nonReentrant unPaused
redeem	Public	Can Modify State	nonReentrant unPaused
_withdraw	Internal	Can Modify State	-
_updateUserData	Internal	Can Modify State	-
pendingReward	Public	-	-
claim	Public	Can Modify State	-
_claim	Internal	Can Modify State	-
_swap	Internal	Can Modify State	-
assetsPerShare	Internal	-	-

ENF_lowrisk_farm EFVault			
_getTokenIndex	Internal	-	-
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
setExchange	Public	Can Modify State	onlyOwner
setSwapPath	Public	Can Modify State	onlyOwner
pause	Public	Can Modify State	onlyOwner
resume	Public	Can Modify State	onlyOwner

ENF_lowrisk_farm 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	-	-
harvest	Public	Can Modify State	onlyOwner
getBalance	Internal	-	-

ENF_lowrisk_farm Controller			
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
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
addRewardToken	Public	Can Modify State	onlyOwner
removeRewardToken	Public	Can Modify State	onlyOwner
getRewardInfo	Public	-	-

4.3 Vulnerability Summary

[N1] [Suggestion] Redundant Code Usage

Category: Others

Content

In the Vault contract of ENFv3/ENF_lowrisk_farm/ENF_lowrisk_ETH_farm, the user will first calculate the share

when performing the withdraw operation. But it is calculated in the same way as the convertToShares function, so it is not necessary to use duplicate code for the calculation without using the convertToShares function.

Code location:

ENFv3/contracts/core/vault.sol

ENF_lowrisk_farm/contracts/core/vault.sol

ENF_lowrish_ETH_farm/contracts/core/vault.sol

```
    function withdraw(uint256 assets, address receiver) public virtual nonReentrant
unPaused returns (uint256 shares) {
        require(assets != 0, "ZERO_ASSETS");
        require(assets <= maxWithdraw, "EXCEED_ONE_TIME_MAX_WITHDRAW");

        // Calculate share amount to be burnt
        shares = (totalSupply() * assets) /
IController(controller).totalAssets(false);

        require(balanceOf(msg.sender) >= shares, "EXCEED_TOTAL_DEPOSIT");

        // Withdraw asset
        _withdraw(assets, shares, receiver);
    }

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

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

Solution

It is recommended to use the convertToShares function for share calculation.

Status

Acknowledged

[N2] [Suggestion] Potential Precision Calculation Issue

Category: Others

Content

In the Vault contract of ENFv3/ENF_lowrisk_farm/ENF_lowrisk_ETH_farm, users can burn shares through the

redeem function to get back staked assets. It uses `(shares * assetsPerShare()) / 1e24` to calculate the number of assets corresponding to the share, and the assetsPerShare function will multiply `(assetDecimal * 1e18)` when performing calculations. If assetDecimal is not equal to 6, dividing 1e24 when performing assets calculation will cause the decimal of the result to deviate.

Code location:

ENFv3/contracts/core/vault.sol

ENF_lowrisk_farm/contracts/core/vault.sol

ENF_lowrish_ETH_farm/contracts/core/vault.sol

```
    function redeem(uint256 shares, address receiver) public virtual nonReentrant
unPaused returns (uint256 assets) {
        require(shares > 0, "ZERO_SHARES");
        require(shares <= balanceOf(msg.sender), "EXCEED_TOTAL_BALANCE");

        assets = (shares * assetsPerShare()) / 1e24;

        require(assets <= maxWithdraw, "EXCEED_ONE_TIME_MAX_WITHDRAW");

        // Withdraw asset
        _withdraw(assets, shares, receiver);
    }
    function assetsPerShare() internal view returns (uint256) {
        return (IController(controller).totalAssets(false) * assetDecimal * 1e18) /
totalSupply();
    }
```

Solution

It is recommended to remove `assetDecimal * 1e18` when calculating assets, that is, modify the algorithm to `(shares * assetsPerShare()) / (assetDecimal * 1e18)` to avoid incorrect assetDecimal settings.

Status

Acknowledged

[N3] [Medium] Business logic flaws in reward distribution

Category: Design Logic Audit

Content

In the Vault contract of ENF_lowrisk_farm/ENF_lowrisk_ETH_farm, users can obtain shares through deposits, and receive harvest dividends according to the amount of shares held. The key to reward calculation is the accRewardPerTokens and prevBalace parameters. The owner role will increase the accRewardPerTokens parameter every time the harvest operation is performed, and prevBalace represents the user's share balance before reward settlement.

But there will be a way to collect rewards by front-run deposits to improve the efficiency of capital utilization: When the owner role performs the harvest operation, the user deposits at a higher gas fee. At this point the accRewardPerShares of the protocol has not been updated, and the user will get a portion of the shares. Then the owner performs the harvest operation, and the accRewardPerShares of the protocol will increase. Finally the user makes withdrawal and gets reward.

Malicious users can use this method to obtain rewards in the blocks before and after the harvest operation, or even in the same block, without worrying about the problem of liquidity being locked in the protocol, which improves the utilization rate of funds.

Code location:

ENF_lowrisk_farm/contracts/core/vault.sol

ENF_lowrish_ETH_farm/contracts/core/vault.sol

```
    function harvest() public onlyOwner {
        ...
        for (uint256 i = 0; i < rewardTokens.length; i++) {
            uint256 increase = IERC20(rewardTokens[i]).balanceOf(vault) -
rewardBalances[i];
            accRewardPerShares[i] += increase * (1e12) / IVault(vault).totalSupply();
        }

    }

    function _claim(
        bool toAsset,
        address receiver,
        uint256 prevBal
    ) internal {
        ...
        for (uint256 i = 0; i < rewardTokens.length; i++) {
            UserInfo storage user = userInfo[rewardTokens[i]][msg.sender];
            // Calculate user's current pending
```

```
            uint256 pending = (accRewardPerTokens[i] * prevBal) / (1e12) +
user.pendingReward - user.rewardDebt;
            pendings[i] = pending;

            // Update user's reward related info
            user.rewardDebt = (accRewardPerTokens[i] * balanceOf(msg.sender)) /
(1e12);
            user.pendingReward = 0;
        }
        ...
    }
```

Solution

It is recommended to design the distribution of rewards from the time dimension. For example, users are required to stake tokens in order to receive rewards, and the calculation of rewards is positively related to the stake time. This will largely avoid the risk of receiving multiple rewards for one share.

Status

Fixed; After communicating with the project team, the project team stated that it will mitigate this risk by setting a minimum deposit time.

[N4] [Critical] Token Transfer Missing Rewards Update

Category: Design Logic Audit

Content

In the Vault contract of ENF_lowrisk_farm/ENF_lowrisk_ETH_farm, the `_updateUserData` function is used to update the user's reward, but it is not updated when the user's share is transferred. This will result in accounting errors during share token transfers. Users can steal rewards by continuously transferring share tokens to new addresses.

Code location:

ENF_lowrisk_farm/contracts/core/vault.sol

ENF_lowrish_ETH_farm/contracts/core/vault.sol

```
    function _transfer(
        address from,
        address to,
        uint256 amount
```

```
    ) internal virtual {
        require(from != address(0), "ERC20: transfer from the zero address");
        require(to != address(0), "ERC20: transfer to the zero address");

        _beforeTokenTransfer(from, to, amount);

        uint256 fromBalance = _balances[from];
        require(fromBalance >= amount, "ERC20: transfer amount exceeds balance");
        unchecked {
            _balances[from] = fromBalance - amount;
            _balances[to] += amount;
        }

        emit Transfer(from, to, amount);

        _afterTokenTransfer(from, to, amount);
    }

    function _updateUserData(address account, uint256 prevAmount) internal {
        (address[] memory rewardTokens, uint256[] memory accRewardPerTokens) =
IController(controller).getRewardInfo();

        for (uint8 i = 0; i < rewardTokens.length; i++) {
            UserInfo storage user = userInfo[rewardTokens[i]][account];
            // Calculate user's current pending
            uint256 pending = (accRewardPerTokens[i] * prevAmount) / (1e12) +
user.pendingReward - user.rewardDebt;
            user.pendingReward = pending;
            user.rewardDebt = (accRewardPerTokens[i] * balanceOf(account)) / (1e12);
        }
    }
```

Solution

It is recommended to perform the `_updateUserData` operation on the from and to addresses before the share token transfer.

Status

Fixed

[N5] [Suggestion] Issue with checking on fromToken

Category: Design Logic Audit

Content

In the vault contract of ENF_lowrisk_farm/ENF_lowrisk_ETH_farm, the `_swap` function is used to exchange

reward tokens for the specified toToken. It will check whether fromToken is WETH, if the check is true, it will be exchanged through the swapExactETHInput function, if the check is false, it will be exchanged through swapExactTokenInput. However, when fromToken is address(0), the token exchange will also be performed through the swapExactTokenInput function, which may cause the `_swap` function to fail to perform as expected.

Code location:

ENF_lowrisk_farm/contracts/core/vault.sol

ENF_lowrish_ETH_farm/contracts/core/vault.sol

```
    function _swap(address[] memory rewardTokens, uint256[] memory pendings) internal
{
        ...

            if (fromToken == weth) {
                IExchange(exchange).swapExactETHInput{value: amount}(toToken,
swapRouters[i], swapIndexes[i], amount);
            } else {
                // Approve fromToken to Exchange
                IERC20Upgradeable(fromToken).approve(exchange, 0);
                IERC20Upgradeable(fromToken).approve(exchange, amount);

                // Call Swap on exchange
                IExchange(exchange).swapExactTokenInput(fromToken, toToken,
swapRouters[i], swapIndexes[i], amount);
            }
        }
    }
```

Solution

It is recommended to check not only whether the fromToken is WETH, but also whether it is a 0 address.

Status

Acknowledged; After communicating with the project team, the project team stated that the 0 address will not be used in the protocol.

[N6] [Low] Incorrect reward receiving address

Category: Design Logic Audit

Content

In the Vault contract of ENF_lowrisk_farm/ENF_lowrisk_ETH_farm, users can claim rewards through the claim function and specify the receiving address of the rewards. When toAsset is false, the protocol will issue the reward directly to the user, but the destination address of the reward is not the receiver address specified by the user but msg.sender. This is not as expected.

Code location:

ENF_lowrisk_farm/contracts/core/vault.sol

ENF_lowrish_ETH_farm/contracts/core/vault.sol

```
    function _claim(
        bool toAsset,
        address receiver,
        uint256 prevBal
    ) internal {
        ...

        if (!toAsset) {
            for (uint256 i = 0; i < rewardTokens.length; i++) {
                TransferHelper.safeTransferFrom(rewardTokens[i], address(this),
msg.sender, pendings[i]);
                emit Claim(rewardTokens[i], receiver, msg.sender, pendings[i]);
            }
        } else {
            // Swap Reward token to principal asset
            _swap(rewardTokens, pendings);
            uint256 assetOut = getBalance(address(asset), address(this));
            TransferHelper.safeTransfer(address(asset), receiver, assetOut);
        }
    }
```

Solution

It is recommended to change the receiving address of the reward to the receiver address.

Status

Fixed

[N7] [Suggestion] Direct distribution of rewards is not available

Category: Design Logic Audit

Content

In the Vault contract of ENF_lowrisk_farm/ENF_lowrisk_ETH_farm, users can claim rewards through the claim function. When toAsset is false, the protocol will directly issue rewards to users. The safeTransferFrom function is used to transfer tokens when issuing rewards, but the contract has not been approved before. This will cause the contract to be unable to successfully execute the safeTransferFrom operation due to insufficient allowances, and ultimately result in failure to issue rewards.

Code location:

ENF_lowrisk_farm/contracts/core/vault.sol

ENF_lowrish_ETH_farm/contracts/core/vault.sol

```
    function _claim(
        bool toAsset,
        address receiver,
        uint256 prevBal
    ) internal {
        ...

        if (!toAsset) {
            for (uint256 i = 0; i < rewardTokens.length; i++) {
                TransferHelper.safeTransferFrom(rewardTokens[i], address(this),
msg.sender, pendings[i]);
                emit Claim(rewardTokens[i], receiver, msg.sender, pendings[i]);
            }
        } else {
            // Swap Reward token to principal asset
            _swap(rewardTokens, pendings);
            uint256 assetOut = getBalance(address(asset), address(this));
            TransferHelper.safeTransfer(address(asset), receiver, assetOut);
        }
    }
```

Solution

It is recommended to use the safeTransfer interface to issue reward tokens.

Status

Fixed

[N8] [Critical] Risk of price manipulation

Category: Design Logic Audit

Content

In the stETH contract of ENF_lowrisk_ETH_farm, the slippage check of SS depends on the virtual price (get_virtual_price) of Curve Pool, which will be affected by the reentrancy vulnerability of ETH/stETH Pool (please check to Ref[1][2]). Failure to check for slippage will result in malicious theft of funds from the strategy.

Ref:

[1] https://chainsecurity.com/curve-lp-oracle-manipulation-post-mortem/

[2] https://chainsecurity.com/heartbreaks-curve-lp-oracles/

Solution

You can refer to the solution proposed in Ref[1], it is recommended to call `remove_liquidity(0,)` again when obtaining the virtual price to avoid obtaining the wrong virtual price.

Fix reference: https://github.com/makerdao/curve-lp-oracle/commit/302f5e6966fdbfebe0f7063c9d6f6bc1f6470f28

Status

Fixed

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002302080001	SlowMist Security Team	2023.02.01 - 2023.02.08	Passed

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 critical risk, 1 medium risk, 1 low risk, 4 suggestion vulnerabilities. All the 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