Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

On 2023.02.16, the SlowMist security team received the Earning.Farm team's security audit application for Earning.Farm Phase6, 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

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

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, ENF_WBTC_Borrow_ETH and ENF_ETH_Leverage products. 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.

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Index conflict issue	Design Logic Audit	Low	Fixed
N2	Visibility issue with getPID function	Gas Optimization Audit	Suggestion	Fixed
N3	Read-only reentrancy checks subject to rounding errors	Gas Optimization Audit	Suggestion	Fixed
N4	Slippage check issue when Vault gets totalAssets	Design Logic Audit	Low	Acknowledged
N5	Logic optimization	Gas Optimization Audit	Suggestion	Acknowledged
N6	Swap optimization from ETH to FrxETH	Gas Optimization Audit	Suggestion	Fixed

4 Code Overview

4.1 Contracts Description

Codebase:

Audit Version:

https://github.com/Shata-Capital/ENF_ETH_Lowrisk (Include FRX section) commit: 8addc3f4484e61189ee285b5b3adcd75ee93b7c5

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

commit: 481eafc462ab53e1899c321a9f87fe6c3a349814

https://github.com/Shata-Capital/ENF_V3 (Include MIM section)

commit: 4df2ceb35085f57e412a0938aa33716b3de97155

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

commit: 65da088b4b85463ab6aa630ba6d6bc29b5aaafee

Fixed Version:

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

commit: e78cbadf996d8da7e7401d527300b80360d074ae

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

commit: 2d5c5ef2fb14392e3983df01fef374ff96df439e

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:

Whitelist			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	initializer
setCA	Public	Can Modify State	onlyOwner
listed	Public	-	-

ENF_WBTC_Borrow EFVault			
Function Name	Visibility	Mutability	Modifiers

ENF_WBTC_Borrow EFVault			
initialize	Public	Can Modify State	initializer
deposit	External	Can Modify State	nonReentrant unPaused onlyAllowed
getBalance	Internal	-	-
withdraw	Public	Can Modify State	nonReentrant unPaused onlyAllowed
redeem	Public	Can Modify State	nonReentrant unPaused onlyAllowed
_withdraw	Internal	Can Modify State	-
mint	External	Can Modify State	onlySS
totalAssets	Public	-	-
assetsPerShare	Internal	-	-
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
setWhitelist	Public	Can Modify State	onlyOwner
pause	Public	Can Modify State	onlyOwner
resume	Public	Can Modify State	onlyOwner
setSubStrategy	Public	Can Modify State	onlyOwner

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

WBTCBorrowETH			
<Receive Ether>	External	Payable	-
totalAssets	External	-	-
_totalAssets	Internal	-	-
_collateralInWBTC	Internal	-	-
_totalETH	Internal	-	-
deposit	External	Can Modify State	onlyController
_deposit	Internal	Can Modify State	-
withdraw	External	Can Modify State	onlyController
_withdraw	Internal	Can Modify State	-
_swapExactInput	Internal	Can Modify State	-
_swapExactOutput	Internal	Can Modify State	-
getBalance	Internal	-	-
harvest	External	Can Modify State	onlyOwner
raiseLTV	Public	Can Modify State	onlyOwner
reduceLTV	Public	Can Modify State	onlyOwner
emergencyWithdraw	Public	Can Modify State	onlyOwner
withdrawable	External	-	-
ownerDeposit	Public	Can Modify State	onlyOwner
getCollateral	Public	-	-
getDebt	Public	-	-
setController	Public	Can Modify State	onlyOwner
setVault	Public	Can Modify State	onlyOwner
setFeePool	Public	Can Modify State	onlyOwner

WBTCBorrowETH			
setFeeRatio	Public	Can Modify State	onlyOwner
setDepositSlippage	Public	Can Modify State	onlyOwner
setWithdrawSlippage	Public	Can Modify State	onlyOwner
setSwapSlippage	Public	Can Modify State	onlyOwner
setLeverageSlippage	Public	Can Modify State	onlyOwner
setSwapInfo	Public	Can Modify State	onlyOwner
setHarvestGap	Public	Can Modify State	onlyOwner
setMaxDeposit	Public	Can Modify State	onlyOwner
setMLR	Public	Can Modify State	onlyOwner

ENF_V3 EFVault			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	initializer
deposit	Public	Can Modify State	nonReentrant unPaused onlyAllowed
getBalance	Internal	-	-
withdraw	Public	Can Modify State	nonReentrant unPaused onlyAllowed
redeem	Public	Can Modify State	nonReentrant unPaused onlyAllowed
_withdraw	Internal	Can Modify State	-
assetsPerShare	Internal	-	-
totalAssets	Public	-	-
convertToShares	Public	-	-
convertToAssets	Public	-	-
setMaxDeposit	Public	Can Modify State	onlyOwner

ENF_V3 EFVault			
setMaxWithdraw	Public	Can Modify State	onlyOwner
setController	Public	Can Modify State	onlyOwner
setDepositApprover	Public	Can Modify State	onlyOwner
setWhitelist	Public	Can Modify State	onlyOwner
pause	Public	Can Modify State	onlyOwner
resume	Public	Can Modify State	onlyOwner

Mim			
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	-
getPID	Public	-	-
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

Mim			
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

ENF_ETH_Lowrisk EFVault			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	initializer
deposit	Public	Payable	nonReentrant unPaused onlyAllowed
withdraw	Public	Can Modify State	nonReentrant unPaused onlyAllowed
redeem	Public	Can Modify State	nonReentrant unPaused onlyAllowed
_withdraw	Internal	Can Modify State	-
totalAssets	Public	-	-
convertToShares	Public	-	-
assetsPerShare	Internal	-	-
convertToAssets	Public	-	-
setMaxDeposit	Public	Can Modify State	onlyOwner
setMaxWithdraw	Public	Can Modify State	onlyOwner
setController	Public	Can Modify State	onlyOwner

ENF_ETH_Lowrisk EFVault			
setDepositApprover	Public	Can Modify State	onlyOwner
setWhitelist	Public	Can Modify State	onlyOwner
pause	Public	Can Modify State	onlyOwner
resume	Public	Can Modify State	onlyOwner

FrxETH			
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	-
getBalance	Internal	-	-
_swap	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	Payable	onlyOwner
setController	Public	Can Modify State	onlyOwner
setDepositSlippage	Public	Can Modify State	onlyOwner

FrxETH			
setWithdrawSlippage	Public	Can Modify State	onlyOwner
setHarvestGap	Public	Can Modify State	onlyOwner
setMaxDeposit	Public	Can Modify State	onlyOwner
setExchange	Public	Can Modify State	onlyOwner
setSwapPath	Public	Can Modify State	onlyOwner

ENF_Leverage EFVault			
Function Name	Visibility	Mutability	Modifiers
<Receive Ether>	External	Payable	-
initialize	Public	Can Modify State	initializer
deposit	Public	Payable	nonReentrant unPaused onlyAllowed
mint	External	Can Modify State	onlySS
withdraw	Public	Can Modify State	nonReentrant unPaused onlyAllowed
redeem	Public	Can Modify State	nonReentrant unPaused onlyAllowed
_withdraw	Internal	Can Modify State	-
totalAssets	Public	-	-
convertToShares	Public	-	-
assetsPerShare	Internal	-	-
convertToAssets	Public	-	-
setMaxDeposit	Public	Can Modify State	onlyOwner
setMaxWithdraw	Public	Can Modify State	onlyOwner
setController	Public	Can Modify State	onlyOwner
setSubStrategy	Public	Can Modify State	onlyOwner

ENF_Leverage EFVault			
setWhitelist	Public	Can Modify State	onlyOwner
pause	Public	Can Modify State	onlyOwner
resume	Public	Can Modify State	onlyOwner

4.3 Vulnerability Summary

[N1] [Low] Index conflict issue

Category: Design Logic Audit

Content

In the MIM strategy of ENF_V3, the getPID function is used to obtain the pool id of the specified LP token in ConvexBooster. Returns the current index if the match is successful, otherwise returns the 0 index. However, there is a corresponding LP (Curve.fi cDAI/cUSDC) configuration for the 0 index in ConvexBooster, which will make it impossible for the caller to determine whether the return of the 0 index is due to a matching failure or LP tokens in the 0 pool of ConvexBooster.

Code location: ENF_V3/contracts/subStrategies/convex/Mim.sol

```
    function getPID(address _lpToken) public view returns (uint256) {
        for (uint256 i = 0; i < IConvexBooster(convex).poolLength(); i++) {
            (address lp_token, , , , , bool shutdown) =
IConvexBooster(convex).poolInfo(i);
            if (lp_token == _lpToken) return i;
        }
        return 0;
    }
```

Solution

It is recommended to return `type(uint256).max` in case of a match failure. Therefore, the length of the pool cannot reach uint256.

Status

Fixed

[N2] [Suggestion] Visibility issue with getPID function

Category: Gas Optimization Audit

Content

There is a getPID function with public visibility in the MIM strategy of ENF_V3, but this function is not called by other functions in this contract, so using public visibility will consume more gas than external visibility.

Code location: ENF_V3/contracts/subStrategies/convex/Mim.sol

```
    function getPID(address _lpToken) public view returns (uint256) {
        ...
    }
```

Solution

It is recommended to replace public visibility with external.

Status

Fixed

[N3] [Suggestion] Read-only reentrancy checks subject to rounding errors

Category: Gas Optimization Audit

Content

In the StETH contract in ENF_ETH_Lowrisk, the remove_liquidity_one_coin function will be called during the deposit and withdraw operations to avoid virtual price manipulation. However, the remove_liquidity_one_coin operation does not always succeed due to rounding errors in the calculation of `_get_y_D`.

Code location: ENF_ETH_Lowrisk/contracts/subStrategies/convex/StETH.sol

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

        ICurvePoolStETH(curvePool).remove_liquidity_one_coin(0, tokenId, 0);
        ...
    }

    function withdraw(uint256 _amount) external override onlyController returns
  (uint256) {
        ...
        // Do remove liquidity for reentrancy guard
        ICurvePoolStETH(curvePool).remove_liquidity_one_coin(0, tokenId, 0);
        ...
    }
```

Solution

It is recommended to use remove_liquidity instead of remove_liquidity_one_coin operation.

Status

Fixed

[N4] [Low] Slippage check issue when Vault gets totalAssets

Category: Design Logic Audit

Content

In the Vault contract of ENF_ETH_Lowrisk, the totalAssets function is used to obtain the total assets held by the protocol, which will be counted by calling the totalAssets function of each SS contract. In the FrxETH strategy, in order to ensure that the amount of totalAssets obtained has not been manipulated, a slippage check will be performed according to the fetch flag. Fetch is passed as true in the Vault contract, which will ignore the slippage check.

Code location: ENF_ETH_Lowrisk/contracts/core/Vault.sol

```
    function totalAssets() public view virtual returns (uint256) {
        return IController(controller).totalAssets(true);
    }
```

Solution

It is recommended to pass in a false fetch when fetching totalAssets in the Vault contract to ensure that the fetched amount is not manipulated.

Status

Acknowledged; After communicating with the project team, the project team stated that slippage check when calling totalAssets, we allow it to affect from slippage when fetching totalAssets at vault level, so we passed true.

This is especially for offchain fetch to be possible even in extreme condition. But this is only for view, when using totalAssets in deposit or other functions, we use totalAssets(false), so it will be checked by slippage.

[N5] [Suggestion] Logic optimization

Category: Gas Optimization Audit

Content

In the FrxETH contract of ENF_ETH_Lowrisk, there are some functions to convert between share and asset through the totalSupply and totalAssets values of sFrx. However, these interfaces have been provided in the sFrx contract, and the calculated decimal is more accurate. Here is some alternative logic:

The frxBal calculation in `_totalAssets` function can be done by `ISfrx(sFrx).convertToAssets(sFrxBal)`

The lastEarnPrice calculation in `_deposit` function can be done by `ISfrx(sFrx).pricePerShare()`

The currentPrice calculation in `harvest` function can be done by `ISfrx(sFrx).pricePerShare()`

Code location: ENF_ETH_Lowrisk/contracts/subStrategies/frx/FrxETH.sol

```
    function _totalAssets(bool fetch) internal view returns (uint256) {
        ...
        uint256 frxBal = (sFrxBal * sFrxTotal) / totalSupply;
        ...
    }

    function _deposit(uint256 _amount) internal returns (uint256) {
        ...

        if (lastEarnPrice == 0) lastEarnPrice = (ISfrx(sFrx).totalAssets() * 1e18) /
ISfrx(sFrx).totalSupply();
        ...
    }

    function harvest() external override onlyController {
        uint256 currentPrice = (ISfrx(sFrx).totalAssets() * 1e18) /
ISfrx(sFrx).totalSupply();
        ...
        currentPrice = (ISfrx(sFrx).totalAssets() * 1e18) /
ISfrx(sFrx).totalSupply();
        ...
    }
```

Solution

It is recommended to use the interface of the sFrx contract for calculation.

Status

Acknowledged

[N6] [Suggestion] Swap optimization from ETH to FrxETH

Category: Gas Optimization Audit

Content

In the FrxETH contract of ENF_ETH_Lowrisk, the `_deposit` function will select the optimal exchange path according to the price of CurvePool. When the amount exchanged by CurvePool is greater than or equal to `_amount` (curveExpect >= _amount), it will exchange tokens through CurvePool . If `curveExpect == _amount`, converting through CurvePool may consume more gas than minting through frxMinter.

Code location: ENF_ETH_Lowrisk/contracts/subStrategies/frx/FrxETH.sol

```
function _deposit(uint256 _amount) internal returns (uint256) {
    ...
    if (curveExpect < _amount) {
        IFrxMinter(frxMinter).submit{value: _amount}();
    } else {
        _swap(swapFromRouters, swapFromIndexes);
    }
    ...
}
```

Solution

It is recommended to modify the `curveExpect < _amount` check to `curveExpect <= _amount`.

Status

Fixed

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002302220001	SlowMist Security Team	2023.02.16 - 2023.02.22	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 low risks, 4 suggestions. All the findings were fixed. The code was not deployed to the mainnet.

6 Statement

Official Website

www.slowmist.com

E-mail

team@slowmist.com

Twitter

@SlowMist_Team

Github

https://github.com/slowmist