# Smart Contract
# Security Audit Report

# Table Of Contents

# 1 Executive Summary

On 2022.10.19, the SlowMist security team received the Earning.Farm team's security audit application for EarningFarm v3 Iterative Audit, 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_V3

commit: ba8e800b5482725e9b5daf3e6e74a4953d6e4451

**Fixed Version:**

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

commit: 16f22c92acdddae13fe2ca2ededaab009ecf7e50

**Audit Scope:**

contracts/subStrategies/notional/CDai.sol

contracts/exchange/Curve3Pool.sol

contracts/core/Exchange.sol

## 3.2 Vulnerability Information

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

| NO | Title | Category | Level | Status |
| --- | --- | --- | --- | --- |
| N1 | Risk of excessive authority | Authority Control Vulnerability | Medium | Confirmed |
| N2 | Variable storage issue | Gas Optimization Audit | Suggestion | Fixed |
| N3 | Slippage issue | Design Logic Audit | Medium | Confirmed |
| N4 | get_dy index issue | Others | Suggestion | Fixed |
| N5 | Risk of breaching contract integrity | Design Logic Audit | High | Fixed |

| NO | Title | Category | Level | Status |
| --- | --- | --- | --- | --- |
| N6 | Risk of slippage checks being bypassed | Design Logic Audit | High | Fixed |

# 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:

| CDai | | | |
| --- | --- | --- | --- |
| Function Name | Visibility | Mutability | Modifiers |
| initialize | Public | Can Modify State | initializer |
| 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 | - |

| CDai | | | |
|---|---|---|---|
| harvest | External | Can Modify State | onlyController |
| emergencyWithdraw | Public | Can Modify State | onlyOwner |
| withdrawable | External | - | - |
| ownerDeposit | Public | Can Modify State | 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 |
| setSwapPath | Public | Can Modify State | onlyOwner |

| Curve3Pool | | | |
|---|---|---|---|
| Function Name | Visibility | Mutability | Modifiers |
| <Constructor> | Public | Can Modify State | - |
| <Receive Ether> | External | Payable | - |
| setExchange | Public | Can Modify State | onlyOwner |
| addCurvePool | Public | Can Modify State | onlyOwner |
| removeCurvePool | Public | Can Modify State | onlyOwner |
| getPathIndex | Public | - | - |
| pathFrom | Public | - | - |

| Curve3Pool | | | |
|---|---|---|---|
| pathTo | Public | - | - |
| swap | External | Can Modify State | onlyExchange |
| getBalance | Internal | - | - |

## 4.3 Vulnerability Summary

### [N1] [Medium] Risk of excessive authority

**Category: Authority Control Vulnerability**

**Content**

In the Exchange contract, the owner role can set swapCaller and router through the setSwapCaller and listRouter functions. If it is set to a malicious address, funds will be lost. In the CDai contract, the owner role can set sensitive parameters through the setSwapPath function. This will lead to the risk of excessive owner permissions.

Code location: contracts/core/Exchange.sol

```
    function setSwapCaller(address _caller, bool _status) public onlyOwner {
        require(_caller != address(0), "INVALID_ADDRESS");

        swapCaller[_caller] = _status;

        emit SetSwapCaller(_caller, _status);
    }
    function listRouter(address router) public onlyOwner {
        routerListed[router] = true;

        emit RouterListed(router);
    }
```

**Solution**

In the short term, it is recommended to transfer owner ownership to multi-signature wallets, and in the long term, it is

recommended to transfer owner ownership to community governance.

**Status**

Confirmed

### [N2] [Suggestion] Variable storage issue

**Category: Gas Optimization Audit**

**Content**

In the Curve3Pool contract, the swap function is used for token exchange. When fetching pools[_index], it uses storage to store the curve variable, but in this function there is no need to modify pools[_index], so this will consume more gas.

Code location: contracts/exchange/Curve3Pool.sol

```
    function swap(
        address _from,
        address _to,
        bytes32 _index,
        uint256 _amount
    ) external override onlyExchange {
        ...

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

        ...
    }
```

**Solution**

It is recommended to use memory to store curve variables.

**Status**

Fixed

### [N3] [Medium] Slippage issue

**Category: Design Logic Audit**

**Content**

In the Curve3Pool contract, the swap function is used to exchange tokens in 3pool, but the `_min_received` passed in during the exchange is 0, which will cause the exchange process to be subject to a sandwich attack.

Code location: contracts/exchange/Curve3Pool.sol

```
    function swap(
        address _from,
        address _to,
        bytes32 _index,
        uint256 _amount
    ) external override onlyExchange {
        ...

        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));
        }

        ...
    }
```

**Solution**

If the amount of funds exchanged is large, it is recommended to perform a slippage check.

**Status**

Confirmed

### [N4] [Suggestion] get_dy index issue

**Category: Others**

**Content**

In the CDai contract, the _totalAssets function converts DAI to USDC as total assets via CurvePool's get_dy. But `Calculate withdraw amout of usdc - from Dai (j) to USDC(i)` is stated in the comments, while according to the get_dy function description (https://curve.readthedocs.io/factory-pools.html#StableSwap.get_dy ), the i index should be DAI, and the j index should be USDC.

Code location: contracts/subStrategies/notional/CDai.sol

```
    function _totalAssets() internal view returns (uint256) {
        ...
            // Calculate withdraw amout of usdc - from Dai (j) to USDC(i)
            uint256 usdcBal = ICurvePool(poolAddr).get_dy(int128(uint128(i)),
int128(uint128(j)), daiBal);

            return usdcBal;
        }
    }
```

**Solution**

Please double check that the design is as expected.

**Status**

Fixed

### [N5] [High] Risk of breaching contract integrity

**Category: Design Logic Audit**

**Content**

When the user makes a withdrawal, the protocol will withdraw from the SS contract through the controller contract, and then burn the user's share.

In the withdraw function of the SS contract, it will first calculate the number of LPs that the user can withdraw (lpAmt), then extract the LP tokens from the convex, and then use balanceOf(address(this)) to obtain the LP balance of this contract as lpWithdrawn. TotalLP will then subtract lpWithdrawn and remove liquidity from CurvePool. The amount to remove liquidity is also lpWithdrawn. Then transfer all USDC tokens in the SS contract to the controller. Finally, the controller contract transfers the USDC token to the user.

In the withdraw function of the vault contract, after the controller completes the withdrawal, the number of burned shares is calculated based on the assets passed in by the user. This will lead to the destruction of the totalLP value if a malicious user transfers a large amount of LP tokens to the SS contract and withdraws them after depositing.

Code location: contracts/subStrategies/convex/*.sol

```
function withdraw(uint256 _amount) external override onlyController returns (uint256)
{
    ...

    totalLP -= lpWithdrawn;

    ...
}
```

**Solution**

It is suggested that lpAmt should be subtracted from totalLP when SS contract is withdrawn.

**Status**

Fixed

### [N6] [High] Risk of slippage checks being bypassed

**Category: Design Logic Audit**

**Content**

In the CDai contract, the totalAssets function calculates the total collateral amount of the SS contract in the strategy through the get_dy function of CurvePool. When the user withdraws, the contract will participate in the calculation of

the nDAI value to be withdrawn through totalAssets. Unfortunately, a malicious user can manipulate the CurvePool with large sums of money so that the value obtained by the get_dy function is much smaller than expected, which will cause the nDAI value to be much larger than expected when withdrawing. Malicious users can deplete the liquidity in CDai by stealing collateral that does not belong to them.

Code location: contracts/subStrategies/notional/CDai.sol

```
    function _totalAssets() internal view returns (uint256) {
        uint256 nTokenBal = IERC20(nDAI).balanceOf(address(this));

        uint256 nTokenTotal = IERC20(nDAI).totalSupply();

        int256 underlyingDenominated =
INusdc(nDAI).getPresentValueUnderlyingDenominated();

        if (underlyingDenominated < 0) return 0;
        else {
            uint256 daiBal = ((nTokenBal * uint256(underlyingDenominated)) *
daiDecimal) / noteDecimal / nTokenTotal;

            if (daiBal == 0) return 0;

            // Get Curve Pool Info - pool address, token i, j index
            CurvePool memory curvePool = ICurveRouter(router).pools(daiUSDCIndex);
            address poolAddr = curvePool.pool;
            uint256 i = curvePool.i;
            uint256 j = curvePool.j;

            // Calculate withdraw amout of usdc - from Dai (i) to USDC(j)
            uint256 usdcBal = ICurvePool(poolAddr).get_dy(int128(uint128(i)),
int128(uint128(j)), daiBal);

            return usdcBal;
        }
    }
```

**Solution**

It is recommended to implement a price oracle to perform slippage checking on CurvePool.

**Status**

Fixed

# 5 Audit Result

| Audit Number | Audit Team | Audit Date | Audit Result |
| --- | --- | --- | --- |
| 0X002210190002 | SlowMist Security Team | 2022.10.19 - 2022.10.19 | 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 high risks, 2 medium risks, and 2 suggestions. 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