Smart Contract
Security Audit Report

Table Of Contents

1 Executive Summary

On 2022.05.13, the SlowMist security team received the ENF team's security audit application for earning.farm, 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

Project:

earning.farm

Module:

farm-Core + List + Token

Commit:

91798f6cddaf55803dc21be6afee11d9e3646a71

Review Version:

f3b81e4e6692260eef33ee5ddb03a12c61d22d98

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Event log missing	Malicious Event Log Audit	Suggestion	Fixed
N2	Risk of excessive authority	Authority Control Vulnerability	Medium	Ignored
N3	Potential Sandwich Attack Risk	Design Logic Audit	Medium	Fixed
N4	Potential Sandwich Attack Risk	Design Logic Audit	Medium	Fixed
N5	Redundant code	Others	Suggestion	Ignored
N6	Redundant code	Others	Suggestion	Fixed
N7	Risk of excessive authority	Authority Control Vulnerability	Low	Ignored

NO	Title	Category	Level	Status
N8	Risk of excessive authority	Authority Control Vulnerability	Critical	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:

EFLeverVault			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initAddresses	Public	Can Modify State	onlyOwner
receiveFlashLoan	Public	Payable	-
getFeeParam	Public	-	-
getCollecteral	Public	-	-
getDebt	Public	-	-
getVolume	Public	-	-
getVirtualPrice	Public	-	-

EFLeverVault			
deposit	Public	Payable	nonReentrant
_deposit	Internal	Can Modify State	-
withdraw	Public	Can Modify State	nonReentrant
_withdraw	Internal	Can Modify State	-
pause	Public	Can Modify State	onlyOwner
restart	Public	Can Modify State	onlyOwner
reduceActualLTV	Public	Can Modify State	onlyOwner
raiseActualLTV	Public	Can Modify State	onlyOwner
earnReward	Public	Can Modify State	onlyOwner
changeMaxLoanRate	Public	Can Modify State	onlyOwner
changeBlockRate	Public	Can Modify State	onlyOwner
changeFeePool	Public	Can Modify State	onlyOwner
callWithData	Public	Payable	onlyOwner
<Fallback>	External	Payable	-

EFCRVVault			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initAddresses	Public	Can Modify State	onlyOwner
deposit	Public	Payable	nonReentrant

EFCRVVault			
depositStable	Public	Payable	nonReentrant
_deposit	Internal	Can Modify State	-
_stake	Internal	Can Modify State	-
withdraw	Public	Can Modify State	nonReentrant
_withdraw	Internal	Can Modify State	-
earnReward	Public	Can Modify State	onlyOwner
_handleExtraToken	Internal	Can Modify State	-
_exchange_weth	Internal	Can Modify State	-
getLPTokenBalance	Public	-	-
changeWithdrawFee	Public	Can Modify State	onlyOwner
changeHarvestFee	Public	Can Modify State	onlyOwner
changeFeePool	Public	Can Modify State	onlyOwner
changePause	Public	Can Modify State	onlyOwner
getVirtualPrice	Public	-	-
getTotalVolume	Public	-	-
getTotalVolumeInStable	Public	-	-
getUserVolume	Public	-	-
getUserVolumeInStable	Public	-	-
addExtraToken	Public	Can Modify State	onlyOwner
removeExtraToken	Public	Can Modify State	onlyOwner

EFCRVVault			
callWithData	Public	Payable	onlyOwner
<Fallback>	External	Payable	-

ERC20DepositApprover			
Function Name	Visibility	Mutability	Modifiers
allowance	Public	-	-
deposit	Public	Can Modify State	-

ERC20Base			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
transfer	Public	Can Modify State	-
transferFrom	Public	Can Modify State	-
doTransfer	Internal	Can Modify State	-
balanceOf	Public	-	-
approve	Public	Can Modify State	-
allowance	Public	-	-
approveAndCall	Public	Can Modify State	-
totalSupply	Public	-	-
balanceOfAt	Public	-	-
totalSupplyAt	Public	-	-

ERC20Base			
_generateTokens	Internal	Can Modify State	-
_destroyTokens	Internal	Can Modify State	-
_enableTransfers	Internal	Can Modify State	-
getValueAt	Internal	-	-
getCheckPointAt	Internal	-	-
updateValueAtNow	Internal	Can Modify State	-
onTransferDone	Internal	Can Modify State	-
_addTransferListener	Internal	Can Modify State	-
_removeTransferListener	Internal	Can Modify State	-
min	Internal	-	-

ERC20Token			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	ERC20Base
claimStdTokens	Public	Can Modify State	onlyOwner
createCloneToken	Public	Can Modify State	-
addTransferListener	Public	Can Modify State	onlyOwner
removeTransferListener	Public	Can Modify State	onlyOwner
generateTokens	Public	Can Modify State	is_trusted
destroyTokens	Public	Can Modify State	is_trusted

ERC20Token			
enableTransfers	Public	Can Modify State	onlyOwner

SafeERC20			
Function Name	Visibility	Mutability	Modifiers
safeTransfer	Internal	Can Modify State	-
safeTransferFrom	Internal	Can Modify State	-
safeApprove	Internal	Can Modify State	-
safeIncreaseAllowance	Internal	Can Modify State	-
safeDecreaseAllowance	Internal	Can Modify State	-
callOptionalReturn	Private	Can Modify State	-

AddressList			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
get_all_addresses	Public	-	-
get_address	Public	-	-
get_address_num	Public	-	-
is_address_exist	Public	-	-
_add_address	Internal	Can Modify State	-
_remove_address	Internal	Can Modify State	-
_reset	Internal	Can Modify State	-

TrustListTools			
Function Name	Visibility	Mutability	Modifiers
changeTrustList	Public	Can Modify State	onlyOwner

TrustList			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
is_trusted	Public	-	-
get_trusted	Public	-	-
get_trusted_num	Public	-	-
add_trusted	Public	Can Modify State	onlyOwner
remove_trusted	Public	Can Modify State	onlyOwner

Migrations			
Function Name	Visibility	Mutability	Modifiers
setCompleted	Public	Can Modify State	restricted

4.3 Vulnerability Summary

[N1] [Suggestion] Event log missing

Category: Malicious Event Log Audit

Content

- contracts/core/EFLeverVault.sol

Modifying important variables in the contract requires corresponding event records.

```
function initAddresses(address[7] memory addr) public onlyOwner{
  aave = addr[0];
  balancer = addr[1];
  balancer_fee = addr[2];
  lido = addr[3];
  asteth = addr[4];
  curve_pool = addr[5];
  weth = addr[6];
}
```

- contracts/core/EFCRVVault.sol

Modifying important variables in the contract requires corresponding event records.

```
function initAddresses(address[11] memory addr) public onlyOwner{
  crv = addr[0];
  usdc = addr[1];
  eth_usdc_router = addr[2];
  weth = addr[3];
  cvxcrv = addr[4];
  eth_crv_router = addr[5];
  crv_cvxcrv_router = addr[6];
  eth_usdt_router = addr[7];
  usdt = addr[8];
  oracle = addr[9];
  staker = addr[10];
}
```

Solution

Record key events.

Status

Fixed

[N2] [Medium] Risk of excessive authority

Category: Authority Control Vulnerability

Content

- contracts/core/EFLeverVault.sol

The owner's authority is too large. If the private key is lost, the attacker can use the pause function to `transfer` the funds in the contract through `callWithData` ,or directly transfer astheth.

```
  function callWithData(address payable to, bytes memory data, uint256 amount)public payable onlyOwner{
    (bool status, ) = to.call.value(amount)(data);
    require(status, "call failed");
  }

  function delegateCallWithData(address payable to, bytes memory data)public payable onlyOwner{
    (bool status, ) = to.delegatecall(data);
    require(status, "call failed");
  }
```

Solution

It is recommended to set `owner` address to timelock contract, governance contract, or multi-sign contract to reduce the risk of private key loss.

Status

Ignored; owner is a multi-signature and multi-party managed address.

[N3] [Medium] Potential Sandwich Attack Risk

Category: Design Logic Audit

Content

- contracts/core/EFCRVVault.sol

it will perform the token swap operation through the `swapExactTokensForTokens` function. But the incoming `minAmountOut` is `0`, which will not do slippage checking,There is a risk of being attacked by sandwiches.

```
  function depositStable(uint256 _amount) public payable nonReentrant{
    require(!is_paused, "paused");

    require(IERC20(usdc).allowance(msg.sender, address(this)) >= _amount, "CFVault:
not enough allowance");
    IERC20(usdc).safeTransferFrom(msg.sender, address(this), _amount);

    if (IERC20(usdc).allowance(address(this), eth_usdc_router) != 0){
      IERC20(usdc).approve(eth_usdc_router, 0);
    }
    IERC20(usdc).approve(eth_usdc_router, _amount);

    uint256 weth_before = IERC20(weth).balanceOf(address(this));
    address[] memory t = new address[](2);
    t[0] = usdc;
    t[1] = weth;

    UniswapV3Interface(eth_usdc_router).swapExactTokensForTokens(_amount, 0, t,
address(this));
    uint256 weth_amount = IERC20(weth).balanceOf(address(this)).safeSub(weth_before);
    if (IERC20(weth).allowance(address(this), eth_crv_router) != 0){
      IERC20(weth).approve(eth_crv_router, 0);
    }
    IERC20(weth).approve(eth_crv_router, weth_amount);

    uint256 tt_before = IERC20(crv).balanceOf(address(this));
    CurveInterface256(eth_crv_router).exchange(0, 1, weth_amount, 0);
    uint256 tt_amount = IERC20(crv).balanceOf(address(this)).safeSub(tt_before);

    _deposit(_amount, tt_amount);
  }
```

Solution

It is recommended to perform a slippage check when performing a swap operation.

Status

Fixed

[N4] [Medium] Potential Sandwich Attack Risk

Category: Design Logic Audit

Content

- contracts/core/EFCRVVault.sol

it will perform the token swap operation through the `swapExactTokensForTokens` function. But the incoming `minAmountOut` is `0`, which will not do slippage checking,There is a risk of being attacked by sandwiches.

```
 function withdraw(uint256 _amount, bool _use_stable) public nonReentrant{
    require(!is_paused, "paused");

    {
      uint256 total_balance = IERC20(ef_token).balanceOf(msg.sender);
      require(total_balance >= _amount, "not enough LP tokens");
    }
    uint256 target_amount;
    {
      //if (IERC20(ef_token).totalSupply() == 0) require(false, "000");
      uint256 lp_amount =
_amount.safeMul(lp_balance).safeDiv(IERC20(ef_token).totalSupply());

      uint256 target_before = IERC20(crv).balanceOf(address(this));
      _withdraw(lp_amount);

      target_amount = IERC20(crv).balanceOf(address(this)).safeSub(target_before);
    }
    uint256 f = 0;
    if(withdraw_fee_ratio != 0 && fee_pool != address(0x0)){
      f = target_amount.safeMul(withdraw_fee_ratio).safeDiv(ratio_base);
      target_amount = target_amount.safeSub(f);
      IERC20(crv).transfer(fee_pool, f);
      TokenInterfaceERC20(ef_token).destroyTokens(msg.sender, _amount);
    }else{
      TokenInterfaceERC20(ef_token).destroyTokens(msg.sender, _amount);
    }
    if (!_use_stable){
      IERC20(crv).transfer(msg.sender, target_amount);
      emit CFFWithdraw(msg.sender, target_amount,
target_amount.safeMul(uint256(ChainlinkInterface(oracle).latestAnswer())).safeDiv(1e2
```

```
0), _amount, f, getVirtualPrice());
    }
    else{
      if (IERC20(crv).allowance(address(this), eth_crv_router) != 0){
        IERC20(crv).approve(eth_crv_router, 0);
      }
      IERC20(crv).approve(eth_crv_router, target_amount);

      uint256 weth_amount;
      {
        uint256 weth_before = IERC20(weth).balanceOf(address(this));
        CurveInterface256(eth_crv_router).exchange(1, 0, target_amount, 0);
        weth_amount = IERC20(weth).balanceOf(address(this)).safeSub(weth_before);
      }

      if (IERC20(weth).allowance(address(this), eth_usdc_router) != 0){
        IERC20(weth).approve(eth_usdc_router, 0);
      }
      IERC20(weth).approve(eth_usdc_router, weth_amount);

      uint256 usdc_amount;
      {
        address[] memory t = new address[](2);
        t[0] = weth;
        t[1] = usdc;
        uint256 usdc_before = IERC20(usdc).balanceOf(address(this));
        UniswapV3Interface(eth_usdc_router).swapExactTokensForTokens(weth_amount, 0,
t, address(this));
        usdc_amount = IERC20(usdc).balanceOf(address(this)).safeSub(usdc_before);
      }
      IERC20(usdc).transfer(msg.sender, usdc_amount);
      emit CFFWithdraw(msg.sender, target_amount, usdc_amount, _amount, f,
getVirtualPrice());
    }
  }
```

Solution

It is recommended to perform a slippage check when performing a swap operation.

Status

Fixed

[N5] [Suggestion] Redundant code

Category: Others

Content

- contracts/erc20/ERC20Impl.sol

`onTransferDone` function not being called

```
    function onTransferDone(address _from, address _to, uint256 _amount) internal {
      for(uint i = 0; i < transferListeners.length; i++){
        TransferEventCallBack t = TransferEventCallBack(transferListeners[i]);
        t.onTransfer(_from, _to, _amount);
      }
    }
```

Solution

If you confirm that you do not need this function, you can delete this function.

Status

Ignored

[N6] [Suggestion] Redundant code

Category: Others

Content

- contracts/core/EFLeverVault.sol

`IERC20(weth).balanceOf(address(this))` return result unused.

```
function raiseActualLTV(uint256 lt) public onlyOwner{//take lt = 7500
    uint256 e = getDebt();
    uint256 st = getCollecteral();
    require(e.safeMul(10000) < st.safeMul(mlr), "no need to raise");
    uint256 x =
st.safeMul(mlr).safeSub(e.safeMul(10000)).safeDiv(uint256(10000).safeSub(mlr));//x =
```

```
(mST-E)/(1-m)
    uint256 y = st.safeMul(lt).safeDiv(10000).safeSub(e).safeSub(1);
    if (x > y) {x = y;}
    IAAVE(aave).borrow(weth, x, 2, 0, address(this));
    IWETH(weth).withdraw(IERC20(weth).balanceOf(address(this)));
    ILido(lido).submit.value(address(this).balance)(address(this));

    IERC20(weth).balanceOf(address(this));//SlowMist//return result unused

    if (IERC20(lido).allowance(address(this), aave) != 0)
{IERC20(lido).safeApprove(aave, 0);}
    IERC20(lido).safeApprove(aave, IERC20(lido).balanceOf(address(this)));
    IAAVE(aave).deposit(lido, IERC20(lido).balanceOf(address(this)), address(this),
0);

    emit ActualLTVChanged(e, st, getDebt(), getCollecteral());
  }
```

Solution

If you are sure you don't need to use it, you can delete it.

Status

Fixed

[N7] [Low] Risk of excessive authority

Category: Authority Control Vulnerability

Content

- contracts/core/EFLeverVault.sol

If the owner permission is lost, the attacker can achieve free recharge by changing the address of the token, thereby taking away the funds in the contract.

```
function initAddresses(address[7] memory addr) public onlyOwner{
    aave = addr[0];
    balancer = addr[1];
    balancer_fee = addr[2];
    lido = addr[3];
```

```
    asteth = addr[4];
    curve_pool = addr[5];
    weth = addr[6];
    emit CFFNewAddress(addr);
  }
```

- contracts/core/EFCRVVault.sol

```
function initAddresses(address[11] memory addr) public onlyOwner{
    crv = addr[0];
    usdc = addr[1];
    eth_usdc_router = addr[2];
    weth = addr[3];
    cvxcrv = addr[4];
    eth_crv_router = addr[5];
    crv_cvxcrv_router = addr[6];
    eth_usdt_router = addr[7];
    usdt = addr[8];
    oracle = addr[9];
    staker = addr[10];
    emit CFFNewAddress(addr);
  }
```

Solution

It is recommended to set `owner` address to timelock contract, governance contract, or multi-sign contract to reduce the risk of private key loss.

Status

Ignored; This function will not have this function when it is officially deployed.

[N8] [Critical] Risk of excessive authority

Category: Authority Control Vulnerability

Content

- contracts/core/EFCRVVault.sol

`delegateCallWithData` is an arbitrary external call, if the private key is lost the attacker can unstake and transfer the funds And for users who have previously authorized the current contract, the attacker can transfer funds that are not operated by the user himself by constructing a malicious contract.

```
  function delegateCallWithData(address payable to, bytes memory data)public payable
onlyOwner{
    (bool status, ) = to.delegatecall(data);
    require(status, "call failed");
  }
```

Solution

It is recommended to delete or modify this function.

Status

Confirmed

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002205230003	SlowMist Security Team	2022.05.13 - 2022.05.23	High Risk

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 1 critical risk, 3 medium risk, 1 low risk,3 suggestion vulnerabilities. And 1 medium risk, 1 low risk and 1 suggestion vulnerabilities were ignored.

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