Smart Contract Security Audit Report

Contents

1. Executive Summary

On Feb. 18, 2021, the SlowMist security team received the earning.farm 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.

SlowMist Smart Contract DeFi project test method:

Black box testing	Conduct security tests from an attacker's perspective externally.
Grey box testing	Conduct security testing on code module 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.

SlowMist Smart Contract DeFi project risk level:

Critical vulnerabilities	Critical vulnerabilities will have a significant impact on the security of the DeFi project, and it is strongly recommended to fix the critical vulnerabilities.
High-risk vulnerabilities	High-risk vulnerabilities will affect the normal operation of DeFi project. It is strongly recommended to fix high-risk vulnerabilities.
Medium-risk vulnerabilities	Medium vulnerability will affect the operation of DeFi project. It is recommended to fix medium-risk vulnerabilities.

Low-risk vulnerabilities	Low-risk vulnerabilities may affect the operation of DeFi project in certain scenarios. It is suggested that the project party should evaluate and consider whether these vulnerabilities need to be fixed.
Weaknesses	There are safety risks theoretically, but it is extremely difficult to reproduce in engineering.
Enhancement Suggestions	There are better practices for coding or architecture.

2. Audit Methodology

Our security audit process for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using public and in-house 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:

- Reentrancy attack and other Race Conditions
- Replay attack
- Reordering attack
- Short address attack
- Denial of service attack
- Transaction Ordering Dependence attack
- Conditional Completion attack
- Authority Control attack
- Integer Overflow and Underflow attack

- TimeStamp Dependence attack
- Gas Usage, Gas Limit and Loops
- Redundant fallback function
- Unsafe type Inference
- Explicit visibility of functions state variables
- Logic Flaws
- Uninitialized Storage Pointers
- Floating Points and Numerical Precision
- tx.origin Authentication
- "False top-up" Vulnerability
- Scoping and Declarations

3. Project Background

3.1 Project Introduction

The earning.farm protocol is a yield seeking protocol that pools together users' assets and deposits them into other Defi protocols to seeking high yield, while mitigate potential loss by only invest asset in homogeneity pools like stablecoin pools, BTC ERC20 token pools and ETH pools.

Initial audit version:

File name

cff.zip:

SHA256:

6638ef7940660f01597da84c4748ba8845a4d6acb80311fe79b343d56e7f7ab1

Final audit version：

File name

cff.zip:

SHA256:

6638ef7940660f01597da84c4748ba8845a4d6acb80311fe79b343d56e7f7ab1

3.2 Project Structure

```
.
├── AddressList.sol
├── Migrations.sol
├── TrustList.sol
├── TrustListTools.sol
├── assets
│   └── TokenBank.sol
├── core
│   ├── CFController.sol
│   ├── CFETHController.sol
│   ├── CFETHVault.sol
│   ├── CFVault.sol
│   ├── CRVExchange.sol
│   ├── IPool.sol
│   ├── btcpool
│   │   ├── BbtcPool.sol
│   │   ├── HbtcPool.sol
│   │   ├── IWbtcPoolBase.sol
│   │   └── TbtcPool.sol
│   ├── ethpool
│   │   ├── IethPoolBase.sol
│   │   └── SethPool.sol
│   └── pool
│       ├── AavePool.sol
│       ├── BusdPool.sol
│       ├── CompoundPool.sol
│       ├── GusdPool.sol
│       ├── IPoolBase.sol
│       ├── TriPool.sol
```

```
|       └── YPool.sol
├── erc20
|   ├── ERC20DepositApprover.sol
|   ├── ERC20Impl.sol
|   ├── ERC20Token.sol
|   ├── IERC20.sol
|   ├── SafeERC20.sol
|   └── TokenInterface.sol
├── test
|   ├── CRV.sol
|   ├── DummyDex.sol
|   ├── STDERC20.sol
|   ├── TestAddressList.sol
|   ├── TestERC20.sol
|   ├── TestTokenClaimer.sol
|   ├── TestUSDCPool.sol
|   ├── TestUSDCPool2.sol
|   ├── USDC.sol
|   ├── USDT.sol
|   ├── WBTC.sol
|   ├── hack.sol
|   └── hacketh.sol
└── utils
    ├── Address.sol
    ├── AddressArray.sol
    ├── Ownable.sol
    ├── ReentrancyGuard.sol
    ├── SafeMath.sol
    └── TokenClaimer.sol
```

3.3 Contract Address

Contract Name	Contract Address
CRVExchange	0x65ff67d6a61ae6bab8821c584f5f394d35e9b50e
TrustList（USDC）	0x246e50a0161b005AC9DD0AE80661C5ae843B64c3
CFToken（USDC）	0x44B439f6624e84c29f2ee5fd3C2CCf29C8DC7572

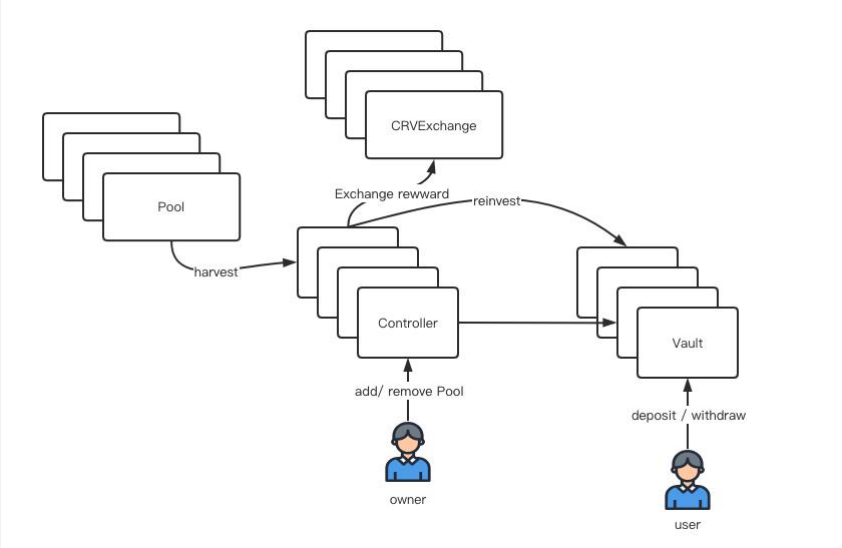
ERC20DepositApprover（USDC）	0xe5afC078684683dc232E053c2c9D86015Aa00Ec6
CFVault（USDC）	0xa1E263225E24333CA4d26083C94092D6bfEe1DDd
CFController（USDC）	0x6c57eA7e3B81b8a166ED5b8E396a401C7a9c890b
CompoundPool	0x4484F01080e8F596d82407926e99895A947EC87b
BUSDPool	0x01964ca263624b105DDE4486E6d1130A207D9117
TriPool	0x35fD3579956808e68D4DF7b3efD63575230Df26F
YPool	0xE673562C20bC3898b91d7700e81Fcb30abDb398C
AavePool	0x6E2549D909ED2D461b0594CC755eE6dD927e0640
GUSDPool	0xb7768f0672a64953e30C4F0c0a2Bd901172fB020
TrustList（WBTC）	0xb9a5263108F14EF5021047a82852f473c2Dc400f
CFToken（WBTC）	0x0319180cA78edB186fBFC7786E0E5f51FDF21263
CFVault（WBTC）	0x1C62D47Aae452877D4E4ff6D7ECDaE5A50ef7326
CFController（WBTC）	0xeb792a00F482DD2f75702dE9eFbC1C2C72e54a2E
HbtcPool	0x8C224F75433EaF4e1B07E0FCa9Ba79148498384D
BbtcPool	0x8E8482f207AeEF576bfe4D5526a53894eE9aAc2b
TbtcPool	0xbab88A555c865744Ef3d207649A215D56576D663
TrustList (ETH)	0x745d97Eb8c714ac839d3dD505C996A3dFA27B476
CFETHToken	0xA709eCF2253B18A757214D64F42026Be8F008bD8
CFETHVault	0xa5eafc384f22d743EDfFa1aE5375bac95495fE2e

CFETHController	0x56Ac11ac8801c1929D19bb84B5f06a9D7c551B1e
SethPool	0x7478064432745612B7944C3229ff7ece51bf3e3E

4. Code Overview

4.1 Contract Structure

The earning.farm system is mainly composed of 5 parts, namely Controller contract, Pool contract, ERC20 Token contract, CRVExchange contract, and Vault contract. The Pool contract is used to deposit the corresponding funds to the Curve, the Controller is responsible for adding/removing the corresponding Pool and reinvesting into the Curve, the ERC20 Token contract is used to generate the user's corresponding deposit share, and the CRVExchange contract is used to convert the revenue Reinvest into other tokens, and the Vault contract is used to process user deposits and withdrawals. The overall architecture diagram is as follows:



4.2 Contract Visibility Analysis

BUSDPool			
Function Name	Visibility	Mutability	Modifiers
constructor	Public	can modify state	-
deposit_usdc	Internal	can modify state	-
withdraw_from_curve	Internal	can modify state	-
get_virtual_price	Public	-	-

TokenBank			
Function Name	Visibility	Mutability	Modifiers
fallback	External	payable	-
constructor	Public	can modify state	TrustListTools
claimStdTokens	Public	can modify state	onlyOwner

balance	Public	can modify state	-
token	Public	-	-
transfer	Public	can modify state	onlyOwner
issue	Public	can modify state	is_trusted

TokenBankFactory			
Function Name	Visibility	Mutability	Modifiers
newTokenBank	Public	can modify state	-

TrustListInterface			
Function Name	Visibility	Mutability	Modifiers
is_trusted	Public	can modify state	-

TrustListTools			
Function Name	Visibility	Mutability	Modifiers
constructor	Public	can modify state	-

TokenClaimer			
Function Name	Visibility	Mutability	Modifiers
_claimStdTokens	Internal	can modify state	-

CurveInterface			
Function Name	Visibility	Mutability	Modifiers
add_liquidity	Public	can modify state	-

remove_liquidity_one_coin	Public	can modify state	-

HbtcPool			
Function Name	Visibility	Mutability	Modifiers
constructor	Public	can modify state	-
deposit_wbtc	Internal	can modify state	-
withdraw_from_curve	Internal	can modify state	-
get_virtual_price	Public	can modify state	-

PriceInterface			
Function Name	Visibility	Mutability	Modifiers
get_virtual_price	Public	-	-

CRVGaugeInterface			
Function Name	Visibility	Mutability	Modifiers
deposit	Public	can modify state	-
withdraw	Public	can modify state	-
claim_rewards	Public	can modify state	-

MinterInterface			
Function Name	Visibility	Mutability	Modifiers
mint	Public	can modify state	-

IWbtcPoolBase

Function Name	Visibility	Mutability	Modifiers
constructor	Public	can modify state	-
deposit_wbtc	Internal	can modify state	-
deposit	Public	can modify state	onlyController
deposit_to_gauge	Internal	can modify state	-
withdraw_from_curve	Internal	can modify state	-
withdraw	Public	can modify state	onlyController
withdraw_from_gauge	Internal	can modify state	-
setController	Public	can modify state	onlyOwner
claimStdToken	Public	can modify state	onlyOwner
earn_crv	Public	can modify state	onlyController
get_lp_token_balance	Public	-	-
get_lp_token_addr	Public	-	-

IethPoolBase			
Function Name	Visibility	Mutability	Modifiers
constructor	Public	can modify state	-
deposit_eth	Internal	can modify state	-
deposit	Public	payable	onlyController
deposit_to_gauge	Internal	can modify state	-
withdraw_from_curve	Internal	can modify state	-
withdraw	Public	can modify state	onlyController
withdraw_from_gauge	Internal	can modify state	-
setController	Public	can modify state	onlyOwner

claimStdToken	Public	can modify state	onlyOwner
earn_crv	Public	can modify state	onlyController
get_lp_token_balance	Public	-	-
get_lp_token_addr	Public	-	-

CompoundPool			
Function Name	Visibility	Mutability	Modifiers
constructor	Public	can modify state	-
deposit_usdc	Internal	can modify state	-
withdraw_from_curve	Internal	can modify state	-
get_virtual_price	Public	-	-

TriPool			
Function Name	Visibility	Mutability	Modifiers
constructor	Public	can modify state	-
deposit_usdc	Internal	can modify state	-
withdraw_from_curve	Internal	can modify state	-
get_virtual_price	Public	-	-

YPool			
Function Name	Visibility	Mutability	Modifiers
constructor	Public	can modify state	-
deposit_usdc	Internal	can modify state	-
withdraw_from_curve	Internal	can modify state	-

get_virtual_price	Public	-	-

CFController			
Function Name	Visibility	Mutability	Modifiers
constructor	Public	can modify state	-
get_current_pool	Public	-	-
add_pool	Public	can modify state	onlyOwner
remove_pool	Public	can modify state	onlyOwner
change_current_pool	Public	can modify state	onlyOwner
earnCRV	Public	can modify state	-
refundTarget	Public	can modify state	-
pauseAndTransferTo	Public	can modify state	onlyOwner
changeExtraToken	Public	can modify state	onlyOwner
changeCRVHandler	Public	can modify state	onlyOwner
changeFeePool	Public	can modify state	onlyOwner
changeHarvestFee	Public	can modify state	onlyOwner

CFControllerFactory			
Function Name	Visibility	Mutability	Modifiers
createCFController	Public	can modify state	-

CFETHController			
Function Name	Visibility	Mutability	Modifiers

constructor	Public	can modify state	-
get_current_pool	Public	-	-
add_pool	Public	can modify state	onlyOwner
remove_pool	Public	can modify state	onlyOwner
change_current_pool	Public	can modify state	onlyOwner
earnCRV	Public	can modify state	-
refundTarget	Public	payable	-
pauseAndTransferTo	Public	can modify state	onlyOwner
changeExtraToken	Public	can modify state	onlyOwner
changeCRVHandler	Public	can modify state	onlyOwner
changeFeePool	Public	can modify state	onlyOwner
changeHarvestFee	Public	can modify state	onlyOwner

CFETHControllerFactory			
Function Name	Visibility	Mutability	Modifiers
createCFETHController	Public	can modify state	-

CFETHVault			
Function Name	Visibility	Mutability	Modifiers
constructor	Public	can modify state	-
deposit	Public	payable	nonReentrant
withdraw	Public	can modify state	nonReentrant
changeWithdrawFee	Public	can modify state	onlyOwner
changeDepositFee	Public	can modify state	onlyOwner

changeController	Public	can modify state	onlyOwner
changeFeePool	Public	can modify state	onlyOwner
get_virtual_price	Public	-	-

CFETHVaultFactory			
Function Name	Visibility	Mutability	Modifiers
createCFETHVault	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	-	-
_generateTokens	Internal	can modify state	-
_destroyTokens	Internal	can modify state	-
_enableTransfers	Internal	can modify state	-

getValueAt	Internal	-	-
updateValueAtNow	Internal	can modify state	-
onTransferDone	Internal	can modify state	-
_addTransferListener	Internal	can modify state	-
_removeTransferListener	Internal	can modify state	-
min	Internal	-	-

CFVault			
Function Name	Visibility	Mutability	Modifiers
constructor	Public	can modify state	-
deposit	Public	can modify state	-
withdraw	Public	can modify state	-
changeWithdrawFee	Public	can modify state	onlyOwner
changeDepositFee	Public	can modify state	onlyOwner
changeController	Public	can modify state	onlyOwner
changeFeePool	Public	-	onlyOwner
get_virtual_price	Public		-

CRVExchange			
Function Name	Visibility	Mutability	Modifiers
constructor	Public	can modify state	ERC20Base TrustListTools
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
enableTransfers	Public	can modify state	onlyOwner

4.3 Code Audit

4.3.1 High-risk vulnerabilities

4.3.1.1 Function is not available

a. The type of the i variable in the parameter list of the remove_liquidity_one_coin interface is wrong, which results in the failure of normal withdrawal. The corresponding type should be modified to the int256 type.

```
contract CurveInterface{
  function add_liquidity(uint256[2] memory uamounts, uint256 min_mint_amount) public;
  function remove_liquidity_one_coin(uint256 _token_amount, uint128 i, uint256 min_mint_amount) public;
  address public curve;}
```

b. The PriceInterface in the Aave Pool uses the curve variable of the CurveInterface when obtaining the price, but the actual Curve Aave pool does not have this variable. As a result, the interface cannot be used normally, and the Aave Pool cannot be deposit and withdrawn.

```
function get_virtual_price() public view returns(uint256){
    return PriceInterface(pool_deposit.curve()).get_virtual_price();
  }
```

Fix status: fixed.

4.3.2 Medium-risk vulnerabilities

4.3.2.1 Failure to check slippage

a. The IPoolBase / IethPoolBase/ IWbtcPoolBase contract directly transfers the corresponding funds to the Curve fund pool when processing the user's fund recharge, without corresponding slippage control, which may cause the user to be rushed away during the deposit, resulting in asset loss.

```
function deposit(uint256 _amount) public{
    deposit_usdc_amount = deposit_usdc_amount + _amount;
    deposit_usdc(_amount);
    uint256 cur = IERC20(lp_token_addr).balanceOf(address(this));
    lp_balance = lp_balance + cur;
    deposit_to_gauge();
}
```

```
function deposit(uint256 _amount) public onlyController{
    deposit_wbtc_amount = deposit_wbtc_amount + _amount;
    deposit_wbtc(_amount);
    uint256 cur = IERC20(lp_token_addr).balanceOf(address(this));
    lp_balance = lp_balance + cur;
    deposit_to_gauge();
}
```

```
function deposit() public payable onlyController{
    deposit_wbtc_amount = deposit_wbtc_amount + _amount;
    deposit_eth();
    uint256 cur = IERC20(lp_token_addr).balanceOf(address(this));
    lp_balance = lp_balance + cur;
    deposit_to_gauge();
}
```

b. During the CRV exchange operation, the contract did not check the exchange slippage, resulting in possible losses due to the slippage.

```
function handleExtraToken(address from, address target_token, uint256 amount) public{
    uint256 maxOut = 0;
    uint256 fdi = 0;
    uint256 fpi = 0;

    for(uint di = 0; di < dexs.length; di ++){
      for(uint pi = 0; pi < path_indexes.length; pi ++){
        if(path_from_addr(pi) != from || path_to_addr(pi) != target_token){
          continue;
        }
        uint256 t = get_out_for_dex_path(di, pi, amount);
        if( t > maxOut ){
          fdi = di;
          fpi = pi;
          maxOut = t;
        }
      }
    }
    IERC20(from).transferFrom(msg.sender, address(this), amount);
    IERC20(from).approve(dexs[fdi], amount);
    SushiUniInterface(dexs[fdi]).swapExactTokensForTokens(amount, 0, paths[path_indexes[fpi]], address(this),
block.timestamp + 10800);

    uint256 target_amount = IERC20(target_token).balanceOf(address(this));
    IERC20(target_token).approve(address(msg.sender), target_amount);
    CFControllerInterface(msg.sender).refundTarget(target_amount);
    }
```

Fix status: fixed.

4.3.2.2 Risk of Excessive Authority

The owner authority of controller / Vault / Exchgange / TrustList / Pool has not been transferred to the corresponding Timelock contract or community governance at present. The project party can arbitrarily modify sensitive parameters or transfer user funds through owner authority. There is a risk of excessive authority.

Fix status: not fix.

4.3.3 Enhancement Suggestion

4.3.3.1 The verification recharge amount has not been verified.

```
function deposit(uint256 _amount) public{
    require(controller != CFControllerInterface(0x0) && controller.get_current_pool() != ICurvePool(0x0), "paused");
    require(IERC20(target_token).allowance(msg.sender, address(this)) >= _amount, "CFVault: not enough allowance");

    //SlowMist// there has not restriction on _amount, which may cause gas wasting

    uint tt_before = IERC20(target_token).balanceOf(address(this));
    IERC20(target_token).safeTransferFrom(msg.sender, address(this), _amount);

    if(deposit_fee_ratio != 0 && fee_pool != address(0x0)){
      uint256 f = _amount.safeMul(deposit_fee_ratio).safeDiv(ratio_base);
      emit CFFDepositFee(msg.sender, _amount, f);
      _amount = _amount.safeSub(f);
      if(f != 0){
        IERC20(target_token).safeTransfer(fee_pool, f);
      }
    }

    uint tt_after = IERC20(target_token).balanceOf(address(this));
    _amount = tt_after.safeSub(tt_before);

    IERC20(target_token).safeApprove(address(controller.get_current_pool()), 0);
    IERC20(target_token).safeApprove(address(controller.get_current_pool()),_amount);
```

Fix stauts: ignored.

4.3.3.2 Hard coded address

The address is hard-coded in many places in the contract. Once the relevant external address is changed, the business logic will become invalid.

Fix stauts: ignored.

4.3.3.3 Simplified operation leads to possible calculation errors

The CFValut.sol / CFETHVault.sol contract uses precision merging to expand the recharge amount when processing the accuracy of the recharge, instead of first expanding and then reducing it, which may cause a certain degree of error.

```
function deposit(uint256 _amount) public{
    require(controller != CFControllerInterface(0x0) && controller.get_current_pool() != ICurvePool(0x0), "paused");
    require(IERC20(target_token).allowance(msg.sender, address(this)) >= _amount, "CFVault: not enough allowance");
    require(_amount <= max_amount, "too large amount");
    require(slip != 0, "Slippage not set");

    uint tt_before = IERC20(target_token).balanceOf(address(this));
    IERC20(target_token).safeTransferFrom(msg.sender, address(this), _amount);

    if(deposit_fee_ratio != 0 && fee_pool != address(0x0)){
      uint256 f = _amount.safeMul(deposit_fee_ratio).safeDiv(ratio_base);
      emit CFFDepositFee(msg.sender, _amount, f);
      if(f != 0){
        IERC20(target_token).safeTransfer(fee_pool, f);
      }
    }

    uint tt_after = IERC20(target_token).balanceOf(address(this));
    _amount = tt_after.safeSub(tt_before);

    uint dec = uint(10)**(ERC20Base(target_token).decimals());
    uint vir = controller.get_current_pool().get_virtual_price();
    uint min_amount = _amount.safeMul(uint(1e32)).safeMul(slip).safeDiv(dec).safeDiv(vir);

    IERC20(target_token).safeApprove(address(controller.get_current_pool()), 0);
    IERC20(target_token).safeApprove(address(controller.get_current_pool()),_amount);
```

```
function deposit() public payable nonReentrant{
    require(controller != CFETHControllerInterface(0x0) && controller.get_current_pool() != ICurvePoolForETH(0x0),
"paused");
```

```
    require(msg.value > 0, "CFVault: zero amount");
    require(slip != 0, "Slippage not set");

    uint _amount = msg.value;

    require(_amount <= max_amount, "too large amount");

    if(deposit_fee_ratio != 0 && fee_pool != address(0x0)){
      uint256 f = _amount.safeMul(deposit_fee_ratio).safeDiv(ratio_base);
      emit CFFDepositFee(msg.sender, _amount, f);
      _amount = _amount.safeSub(f);
      if(f != 0){
        fee_pool.transfer(f);
      }
    }
    uint eth_before = address(this).balance;

    uint vir = controller.get_current_pool().get_virtual_price();
    uint min_amount = _amount.safeMul(uint(1e14)).safeMul(slip).safeDiv(vir);

    uint lp_before = controller.get_current_pool().get_lp_token_balance();
```

Fix stauts: ignored.

4.3.3.4 There is no restriction on the refundTarget function, which may result in the loss of user funds

The CFController.sol / CFETHController.sol contract does not limit the calling source of the refundTarget function, and the user may misuse the function to call this function and cause the loss of funds.

```
function refundTarget() public payable{
    //IERC20(target_token).safeTransferFrom(msg.sender, address(this), _amount);
    uint _amount = msg.value;
    if(harvest_fee_ratio != 0 && fee_pool != address(0x0)){
      uint256 f = _amount.safeMul(harvest_fee_ratio).safeDiv(ratio_base);
      emit CFFRefund(_amount, f);
      _amount = _amount.safeSub(f);
      if(f != 0){
```

```
            fee_pool.transfer(_amount);
        }
    }else{
        emit CFFRefund(_amount, 0);
    }
    ICurvePoolForETH(current_pool).deposit.value(_amount)();
}
```

Fix stauts: ignored.

5. Audit Result

5.1 Conclusion

Audit Result : Medium Risk

Audit Number : 0X002103180003

Audit Date : Mar. 18, 2021

Audit Team : SlowMist Security Team

Summary conclusion: The SlowMist security team use a manual and SlowMist Team in-house analysis tool audit of the codes for security issues. There are nine security issues found during the audit. There are two high-risk vulnerabilities, three low-risk vulnerabilities and four enhancement suggestion. We also provide five enhancement suggestions. Since the current contract authority of the project has not been transferred to the community governance or TimeLock contract, the project still has the risk of excessive authority. Comprehensive assessment is medium risk.

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 base 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 this 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