Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

On 2022.12.02, the SlowMist security team received the Earning.Farm team's security audit application for ENF_WBTC_Borrow_ETH, 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_WBTC_Borrow_ETH

commit: 31edacafb1e3b59ead110184d1e25b333235c5d2

Fixed Version:

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

commit: 4eafa3c008b10816740967d301a4b67f5197f33d

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Redundant parameter issue	Others	Suggestion	Fixed
N2	Token swap defect when withdrawing	Design Logic Audit	Low	Fixed
N3	Swap balance has not been processed	Design Logic Audit	Medium	Fixed
N4	Slippage check issue with `_swapExactInput` operation	Design Logic Audit	Low	Acknowledged
N5	Defects in LTV operation	Design Logic Audit	Medium	Fixed
N6	Reduced availability for LTV operations	Design Logic Audit	High	Fixed
N7	The withdraw function will not work when the market is extreme	Design Logic Audit	Medium	Fixed

NO	Title	Category	Level	Status
N8	Risk of multiple leverages in unilateral market conditions	Design Logic Audit	Suggestion	Acknowledged
N9	Risk of excessive authority	Design Logic Audit	Medium	Acknowledged

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:

PriceOracle			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
getAssetPrice	External	-	-

WBTCBorrowETH			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	initializer
<Receive Ether>	External	Payable	-

WBTCBorrowETH			
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

WBTCBorrowETH			
setVault	Public	Can Modify State	onlyOwner
setFeePool	Public	Can Modify State	onlyOwner
setDepositSlippage	Public	Can Modify State	onlyOwner
setWithdrawSlippage	Public	Can Modify State	onlyOwner
setSwapSlippage	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

4.3 Vulnerability Summary

[N1] [Suggestion] Redundant parameter issue

Category: Others

Content

In the PriceOracle contract, the getAssetPrice function is used to obtain the relative price of WBTC and ETH. But the `_asset` parameter it receives is not used.

Code location: contracts/subStrategies/PriceOracle.sol

```
    function getAssetPrice(address _asset) external view override returns (uint256) {
        (, int256 ethPrice, , , ) = IChainlink(ethOracle).latestRoundData();
        (, int256 wbtcPrice, , , ) = IChainlink(wbtcOracle).latestRoundData();

        return (uint256(wbtcPrice) * 1e18) / uint256(ethPrice);
    }
```

Solution

It is recommended to remove this parameter if it is not the intended design.

Status

Fixed

[N2] [Low] Token swap defect when withdrawing

Category: Design Logic Audit

Content

In the WBTCBorrowETH contract, the withdraw function is used to withdraw WBTC tokens. When the repayable amount of the contract is less than the required loan amount (ethWithdrawn < ethDebt), the contract will withdraw wbtcToSwap amount of WBTC from AAVE to swap it into WETH, and use the wbtcAmt value as the amountInMaximum in the Swap exchange. However, since the wbtcAmt value is indirectly calculated through the ChainLink price, there may be a deviation from the price in Uniswap v3, so using the wbtcAmt value as the amountInMaximum parameter may not be successfully swapped due to the price deviation.

Code location: contracts/subStrategies/WBTC_Borrow_ETH.sol

```
    function _withdraw(uint256 _amount) internal returns (uint256) {
            ...
            uint256 price = IAavePriceOracle(priceOracle).getAssetPrice(wbtc);
            uint256 wbtcToSwap = ((((ethDebt - ethWithdrawn) * 1e8) / price) *
(magnifier + swapSlippage)) / magnifier;
            require((wbtcToSwap * magnifier) / _amount < withdrawSlippage,
"WITHDRAW_SLIPPAGE_EXCEED");

            // Withdraw WBTC
            uint256 amtBefore = IERC20(wbtc).balanceOf(address(this));
            IAave(aave).withdraw(wbtc, wbtcToSwap, address(this));
            uint256 wbtcAmt = IERC20(wbtc).balanceOf(address(this)) - amtBefore;

            _swapExactOutput(wbtc, weth, ethDebt - ethWithdrawn, wbtcAmt);
            ...
    }
```

Solution

Best practice should use the Uniswap v3 pool's token count for calculations and slippage checks. Pass in the calculation result as the amountInMaximum parameter.

Status

Fixed; After discussing with the project team, the project team fixed the failure problem of small swap, but did not do anything for large swap to avoid the risk of Pool being manipulated.

[N3] [Medium] Swap balance has not been processed

Category: Design Logic Audit

Content

In the _withdraw function of the WBTCBorrowETH contract, when `ethWithdrawn < ethDebt`, the contract will withdraw WBTC tokens from AAVE and swap them into ETH to repay the loan. If the amount of ETH is greater than the amount of debt required to be repaid (ethBal > ethDebt), the contract will swap the excess part into WBTC, but these excess WBTC tokens have not been sent to the user, nor have they been re-staked into AAVE. It was left in the SS contract. When the next user deposit, it will be billed as part of the user's deposit.

And when `ethWithdrawn >= ethDebt`, the contract will convert the excess ETH to WBTC, but the contract has not yet processed these WBTC.

Code location: contracts/subStrategies/WBTC_Borrow_ETH.sol

```
function _withdraw(uint256 _amount) internal returns (uint256) {
    ...
    if (ethWithdrawn >= ethDebt) {
         wbtcToWithdraw = _amount;
         _swapExactInput(weth, wbtc, ethWithdrawn - ethDebt);
     } else {
        ...
         uint256 amtBefore = IERC20(wbtc).balanceOf(address(this));
         IAave(aave).withdraw(wbtc, wbtcToSwap, address(this));
         uint256 wbtcAmt = IERC20(wbtc).balanceOf(address(this)) - amtBefore;

         _swapExactOutput(wbtc, weth, ethDebt - ethWithdrawn, wbtcAmt);
```

```
            wbtcToWithdraw = _amount - wbtcToSwap;

            // Check ETH is enough
            uint256 ethBal = address(this).balance;
            require(ethBal >= ethDebt, "INSUFFICIENT_ETH_SWAPPED");

            if (ethBal > ethDebt) _swapExactInput(weth, wbtc, ethBal - ethDebt);
        }

        ...

        uint256 wbtcBefore = IERC20(wbtc).balanceOf(address(this));

        IAave(aave).withdraw(wbtc, wbtcToWithdraw, address(this));

        uint256 withdrawn = IERC20(wbtc).balanceOf(address(this)) - wbtcBefore;

        uint256 minOutput = (_amount * (magnifier - withdrawSlippage)) / magnifier;

        require(withdrawn >= minOutput, "WITHDRAW_SLIPPAGE_TOO_BIG");

        TransferHelper.safeTransferToken(wbtc, controller, withdrawn);

        return withdrawn;
    }
```

Solution

It is recommended to send excess WBTC directly to users.

Status

Fixed

[N4] [Low] Slippage check issue with `_swapExactInput` operation

Category: Design Logic Audit

Content

In the WBTCBorrowETH contract, when performing _withdraw, and emergencyWithdraw operations, the `_swapExactInput` function will be used to exchange tokens. However, the `_swapExactInput` function does not perform a slippage check, which will result in a high probability of being subjected to a sandwich attack by MEV Bot when performing the above operations. This will result in far fewer exchanges than expected, or even very few left.

Code location: contracts/subStrategies/WBTC_Borrow_ETH.sol

```
    function _swapExactInput(
        address _from,
        address _to,
        uint256 _amount
    ) internal {
        require(univ3Router != address(0), "ROUTER_NOT_SET");

        IUniswapV3Router.ExactInputSingleParams memory params =
IUniswapV3Router.ExactInputSingleParams({
            tokenIn: _from,
            tokenOut: _to,
            fee: univ3Fee,
            recipient: address(this),
            amountIn: _amount,
            amountOutMinimum: 0,
            sqrtPriceLimitX96: 0
        });

        ...
    }
```

Solution

It is recommended to calculate the amountOutMinimum through the `PriceOracle::getAssetPrice` function.

Status

Acknowledged

[N5] [Medium] Defects in LTV operation

Category: Design Logic Audit

Content

In the reduceLTV operation, the contract will first extract x amount of WBTC from AAVE and exchange it into WETH for repayment. In this operation, although the liabilities of the contract are reduced, the amount of collateral of the contract is also reduced. At the same time, due to the impact of the slippage of the swap operation, the reduceLTV operation may not be able to effectively control the risk as expected.

Code location: contracts/subStrategies/WBTC_Borrow_ETH.sol

```
    function reduceLTV() public onlyOwner {
        uint256 e = getDebt();
        uint256 st = getCollateral();

        require(e * magnifier > st * mlr, "NO_NEED_TO_REDUCE");

        uint256 x = (e * magnifier - st * mlr) / (magnifier - mlr);

        IAave(aave).withdraw(wbtc, x, address(this));

        uint256 wbtcAmt = IERC20(wbtc).balanceOf(address(this));
        _swapExactInput(wbtc, weth, wbtcAmt);

        uint256 toSend = address(this).balance;
        TransferHelper.safeTransferETH(weth, toSend);

        uint256 wethBal = IERC20(weth).balanceOf(address(this));
        // Approve WETH to AAVE
        IERC20(weth).approve(aave, 0);
        IERC20(weth).approve(aave, wethBal);

        // Repay WETH to aave
        IAave(aave).repay(weth, wethBal, 2, address(this));
    }
```

Solution

It is recommended should do the `e*magnifier <= st*mlr` check at the end of the reduceLTV operation.

Status

Fixed

[N6] [High] Reduced availability for LTV operations

Category: Design Logic Audit

Content

In the protocol, the raiseLTV and reduceLTV functions are important means to improve capital utilization and prevent bad debts, but in these two functions, the token exchange is performed through the `_swapExactInput` function. The `_swapExactInput` function does not check for slippage, which will reduce the availability of raiseLTV and reduceLTV for the protocol.

Code location: contracts/subStrategies/WBTC_Borrow_ETH.sol

```
function raiseLTV(uint256 lt) public onlyOwner {
    ...
    _swapExactInput(weth, wbtc, wethAmt);
    ...
}


function reduceLTV() public onlyOwner {
    ...
    _swapExactInput(wbtc, weth, wbtcAmt);
    ...
}
```

Solution

It is recommended to perform slippage check or not perform token swap when operating LTV.

Status

Fixed

[N7] [Medium] The withdraw function will not work when the market is extreme

Category: Design Logic Audit

Content

In the protocol, when extreme market conditions occur (such as a sharp unilateral drop of BTC) and the owner has no time to adjust the protocol LTV through the reduceLTV function, the protocol's WBTC position will be liquidated. If the protocol's liabilities are fully liquidated (getDebt will become 0), ethDebt will be 0. This will cause the `_withdraw` function to fail to perform the repay operation, and the emergencyWithdraw operation will also not work. Users' funds will be locked in the protocol.

In the repay operation of AAVE, if the repayment amount is 0, it will fail the validateRepay check.

Solution:

Code location: contracts/subStrategies/WBTC_Borrow_ETH.sol

```
    function _withdraw(uint256 _amount) internal returns (uint256) {
        ...
        uint256 ethDebt = (getDebt() * _amount) / _collateralInWBTC();
        ...
        IERC20(weth).approve(aave, 0);
        IERC20(weth).approve(aave, ethDebt);

        // Repay ETH to AAVE
        IAave(aave).repay(weth, ethDebt, 2, address(this));

        ...
    }

    function emergencyWithdraw() public onlyOwner {
        ...
        // Repay ETH
        uint256 totalDebt = getDebt();

        uint256 ethToRepay;
        if (ethWithdrawn >= totalDebt) {
            ethToRepay = totalDebt;
            _swapExactInput(weth, wbtc, ethWithdrawn - ethToRepay);
        } else {
            ethToRepay = ethWithdrawn;
        }

        ...

        // Repay ETH to AAVE
        IAave(aave).repay(weth, ethToRepay, 2, address(this));
        ...
    }
```

Solution

It is recommended that the protocol add emergency measures in extreme market conditions.

Status

Fixed; After communicating with the project team, the project team will withdraw positions from the ENF_ETH_Leverage protocol to the owner through the emergencyWithdraw function in extreme market conditions. And it is expected that users cannot withdraw funds through the withdraw function.

[N8] [Suggestion] Risk of multiple leverages in unilateral market conditions

Category: Design Logic Audit

Content

The protocol deposits WBTC tokens deposited by users into AAVE and lends ETH, and then deposits the loaned ETH into the ENF_ETH_Leverage protocol. The ENF_ETH_Leverage protocol also creates positions in AAVE via ETH/stETH. This makes the ENF_WBTC_Borrow_ETH protocol have multiple leverages, which means it is extremely sensitive to market stability. Once the agreement does not manage LTV properly, it will lead to risks such as bad debts of the agreement.

Code location: contracts/subStrategies/WBTC_Borrow_ETH.sol

Solution

It is recommended to ensure the integrity and timeliness of the protocol risk control. For example: it should be ensured that the timeliness of LTV operations can be ensured to the greatest extent in any market environment.

Status

Acknowledged

[N9] [Medium] Risk of excessive authority

Category: Design Logic Audit

Content

In the protocol, the owner role has many permissions, such as the owner can set sensitive parameters, can suspend the contract, can make emergency withdrawals, etc. It is obviously inappropriate to give all the permissions of the protocol to the owner, which will greatly increase the single point of risk.

Solution

In the short term, transferring owner ownership to multisig contracts is an effective solution to avoid single-point risk. But in the long run, it is a more reasonable solution to implement a privilege separation strategy and set up multiple privileged roles to manage each privileged function separately. And the authority involving user funds should be managed by the community, and the authority involving emergency contract suspension can be managed by the EOA address. This ensures both a quick response to threats and the safety of user funds.

Status

Acknowledged

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002212070002	SlowMist Security Team	2022.12.02 - 2022.12.07	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 1 high risk, 4 medium risks, 2 low risks, and 2 suggestions. All the findings were fixed or acknowledged. 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.