Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

2 Audit Methodology

3 Project Overview

3.1 Project Introduction

3.2 Vulnerability Information

4 Code Overview

4.1 Contracts Description

4.2 Visibility Description

4.3 Vulnerability Summary

5 Audit Result

6 Statement

1 Executive Summary

On 2022.05.17, the SlowMist security team received the Helio team's security audit application for Helio Ceros 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/helio-money/helio-smart-contracts

commit: d543f087d7adb685aebf960275520030baefdcad

Fixed Version:

https://github.com/helio-money/helio-smart-contracts

commit: 831b06cd2af3ccd2df9e8182744f83effe3030ad

Audit Scope:

- contracts/ceros/CeVault.sol
- contracts/ceros/CerosRouter.sol
- contracts/ceros/HelioProvider.sol

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Over-withdrawal issue	Design Logic Audit	Low	Fixed
N2	Potential overflow risk	Integer Overflow and Underflow Vulnerability	Low	Fixed
N3	Risk of excessive authority	Authority Control Vulnerability	Medium	Confirmed
N4	Risk of Fund Theft	Design Logic Audit	Critical	Fixed
N5	Risk of front-run withdraw	Reordering Vulnerability	Medium	Fixed

NO	Title	Category	Level	Status
N6	Sandwich Attack Risk	Reordering Vulnerability	Medium	Fixed
N7	Redundant code issue	Others	Suggestion	Fixed
N8	Compatibility issue	Design Logic Audit	Medium	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:

CerosRouter			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	initializer
deposit	External	Payable	nonReentrant
depositABNBc	External	Can Modify State	nonReentrant
claim	External	Can Modify State	nonReentrant
claimProfit	External	Can Modify State	nonReentrant

CerosRouter			
withdrawABNBc	External	Can Modify State	nonReentrant
withdraw	External	Can Modify State	nonReentrant
withdrawWithSlippage	External	Can Modify State	nonReentrant
getProfitFor	External	-	-
changeVault	External	Can Modify State	onlyOwner
changeDex	External	Can Modify State	onlyOwner
changePool	External	Can Modify State	onlyOwner

CeVault			
Function Name	Visibility	Mutability	Modifiers
initialize	External	Can Modify State	initializer
deposit	External	Can Modify State	nonReentrant
depositFor	External	Can Modify State	nonReentrant
claimYieldsFor	External	Can Modify State	onlyRouter nonReentrant
claimYields	External	Can Modify State	nonReentrant
_claimYields	Private	Can Modify State	-
withdraw	External	Can Modify State	nonReentrant
withdrawFor	External	Can Modify State	nonReentrant onlyRouter
getTotalAmountInVault	External	-	-
getPrincipalOf	External	-	-

CeVault			
getYieldFor	External	-	-
getCeTokenBalanceOf	External	-	-
getDepositOf	External	-	-
getClaimedOf	External	-	-
changeRouter	External	Can Modify State	onlyOwner
getName	External	-	-

HelioProvider			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	initializer
provide	External	Payable	nonReentrant
provideInABNBc	External	Can Modify State	nonReentrant
claimInABNBc	External	Can Modify State	nonReentrant onlyOperator
release	External	Can Modify State	nonReentrant
releaseInABNBc	External	Can Modify State	nonReentrant
liquidation	External	Can Modify State	onlyDao nonReentrant
daoBurn	External	Can Modify State	onlyDao nonReentrant
daoMint	External	Can Modify State	onlyDao nonReentrant
_provideCollateral	Internal	Can Modify State	-
_withdrawCollateral	Internal	Can Modify State	-

HelioProvider			
changeDao	External	Can Modify State	onlyOwner
changeCeToken	External	Can Modify State	onlyOwner
changeCollateralToken	External	Can Modify State	onlyOwner

4.3 Vulnerability Summary

[N1] [Low] Over-withdrawal issue

Category: Design Logic Audit

Content

In the CeVault contract, the user will calculate the number of ceTokens according to the ratio value of the aBNBc contract when performing the deposit operation. When the user performs the withdraw operation, the ratio is multiplied by the number of ceTokens to be withdrawn to calculate the number of aBNBc tokens to be withdrawn. If the value of the ratio during the withdraw operation is greater than the value of the user's deposit operation, then the number of ceTokens held by the user is theoretically More aBNBc tokens can be withdrawn, although the depositors record limits this over-withdrawal operation.

Code location: contracts/ceros/CeVault.sol

```
    function deposit(address recipient, uint256 amount)
        external
        override
        nonReentrant
        returns (uint256)
    {
        uint256 ratio = _aBNBc.ratio();
        _aBNBc.transferFrom(msg.sender, address(this), amount);
        uint256 toMint = (amount * 1e18) / ratio;
        // add profit as other part of yield(yield includes profit before the first
claim)
        _depositors[msg.sender] += amount;
```

```
        _ceTokenBalances[msg.sender] += toMint;
        //  mint ceToken to recipient
        ICertToken(_ceToken).mint(recipient, toMint);
        emit Deposited(msg.sender, recipient, toMint);
        return toMint;
    }

    function withdraw(address recipient, uint256 amount)
        external
        override
        nonReentrant
        returns (uint256)
    {
        uint256 ratio = _aBNBc.ratio();
        uint256 realAmount = (amount * ratio) / 1e18;
        require(
            _aBNBc.balanceOf(address(this)) >= realAmount,
            "not such amount in the vault"
        );
        uint256 balance = _ceTokenBalances[msg.sender];

        require(balance >= amount, "insufficient balance");
        _ceTokenBalances[msg.sender] -= amount;
        // burn ceToken from owner
        ICertToken(_ceToken).burn(msg.sender, amount);
        _depositors[msg.sender] -= realAmount;
        _aBNBc.transfer(recipient, realAmount);
        emit Withdrawn(msg.sender, recipient, realAmount);
        return realAmount;
    }
```

Solution

When the user has withdrawn all depositors, ceTokenBalances should also be cleared to 0.

Status

Fixed; After communicating with the project team, the project team stated that the ratio value of the aBNBc contract will decrease every day and will not increase.

[N2] [Low] Potential overflow risk

Category: Integer Overflow and Underflow Vulnerability

Content

In the CeVault contract, the getYieldFor function is used to calculate the amount of yield that the user can obtain. But if the ratio value of the aBNBc contract is greater than the value of the user deposit, then the result of getPrincipalOf will be greater than the user's `_depositors`. This will cause the calculation of totalYields to fail due to overflow.

Code location: contracts/ceros/CeVault.sol

```
function getYieldFor(address account)
    external
    view
    override
    returns (uint256)
{
    uint256 principal = this.getPrincipalOf(account);
    uint256 totalYields = _depositors[account] - principal;
    if (totalYields <= _claimed[account]) {
        return 0;
    }
    return totalYields - _claimed[account];
}
```

Solution

getYieldFor should return 0 when principal is greater than `_depositors[account]`.

Status

Fixed

[N3] [Medium] Risk of excessive authority

Category: Authority Control Vulnerability

Content

In the CeVault contract, the owner can modify the `_router` address through the changeRouter function, and the router role can withdraw aBNBc tokens for the user through the withdrawFor function, so this will lead to the risk of excessive owner permissions.

In the CerosRouter contract, the owner can modify the `_vault`, `_dex` and `_pool` addresses through the changeVault, changeDex and changePool functions. This will lead to the risk of excessive owner permissions.

In the CerosRouter contract, the owner can transfer the aBNBc tokens of users who have approved the contract into this contract through the depositABNBcFrom function, and mint the ceToken to the owner through the vault contract. This would lead to the risk that the owner could transfer the aBNBc tokens of any user who has approved the contract.

Code location:

contracts/ceros/CeVault.sol

```
function changeRouter(address router) external onlyOwner {
        _router = router;
        emit RouterChanged(router);
    }
```

contracts/ceros/CerosRouter.sol

```
    function changeVault(address vault) external onlyOwner {
        _vault = IVault(vault);
        emit ChangeVault(vault);
    }

    function changeDex(address dex) external onlyOwner {
        _dex = IDex(dex);
        emit ChangeDex(dex);
    }

    function changePool(address pool) external onlyOwner {
        _pool = IBinancePool(pool);
        emit ChangePool(pool);
    }
```

contracts/ceros/CerosRouter.sol

```
    modifier onlyProvider() {
        require(
            msg.sender == owner() || msg.sender == _provider,
            "Dao: not allowed"
        );
        _;
    }

    function depositABNBcFrom(address owner, uint256 amount)
    external
    override
    onlyProvider
    nonReentrant
    returns (uint256 value)
    {
        _certToken.transferFrom(owner, address(this), amount);
        value = _vault.depositFor(msg.sender, amount);
        emit Deposit(msg.sender, _wBnbAddress, value, 0);
        return value;
    }
```

Solution

It is recommended to transfer owner ownership to community governance.

Status

Confirmed

[N4] [Critical] Risk of Fund Theft

Category: Design Logic Audit

Content

In the CerosRouter contract, any user can transfer the specified user's aBNBc token to the vault and mint the ceToken to himself through the depositABNBc function. This will allow malicious users to steal aBNBc tokens from users who have approved this contract through this function.

Code location: contracts/ceros/CerosRouter.sol

```
function depositABNBc(address owner, uint256 amount)
    external
    override
    nonReentrant
    returns (uint256 value)
{
    // let's check balance of CeRouter in aBNBc
    _certToken.transferFrom(owner, address(this), amount);
    value = _vault.depositFor(msg.sender, amount);
    emit Deposit(msg.sender, _wBnbAddress, value, 0);
    return value;
}
```

Solution

It is recommended to restrict that this function can only be called by the HelioProvider contract.

Status

Fixed

[N5] [Medium] Risk of front-run withdraw

Category: Reordering Vulnerability

Content

In the CerosRouter contract, any user can transfer the ceToken into the contract, and use the withdrawWithSlippage function to first withdraw the aBNBc token from the vault contract and then swap the aBNBc token for BNB to the user. Since the user needs to transfer the ceToken into the CerosRouter contract first, a malicious user can call the withdrawWithSlippage function to take possession of the BNB token at a higher gas fee after the user transfers the ceToken.

Code location: contracts/ceros/CerosRouter.sol

```
function withdrawWithSlippage(address recipient, uint256 amount)
    external
```

```
        override
        nonReentrant
        returns (uint256)
    {
        uint256 realAmount = _vault.withdraw(address(this), amount);
        address[] memory path = new address[](2);
        path[0] = address(_certToken);
        path[1] = _wBnbAddress;
        uint256[] memory outAmounts = _dex.getAmountsOut(realAmount, path);
        uint256[] memory amounts = _dex.swapExactTokensForETH(
            amount,
            outAmounts[1],
            path,
            recipient,
            block.timestamp + 300
        );
        //  console.log(); TODO CHECK BALANCE OF ROUTER
        emit Withdrawal(msg.sender, recipient, _wBnbAddress, amounts[1]);
        return amounts[1];
    }
```

Solution

It is recommended to use the withdrawFor function to withdraw aBNBc tokens for the caller.

Status

Fixed

[N6] [Medium] Sandwich Attack Risk

Category: Reordering Vulnerability

Content

In the CerosRouter contract, any user can transfer the ceToken into the contract, and use the withdrawWithSlippage function to first withdraw the aBNBc token from the vault contract and then swap the aBNBc token for BNB to the user. In the swap operation, the incoming amountOutMin parameter is calculated through getAmountsOut, so the calculation result will be affected by the previous user's swap operation. So using the getAmountsOut calculation as a slippage check has no effect.

Code location: contracts/ceros/CerosRouter.sol

```
function withdrawWithSlippage(address recipient, uint256 amount)
    external
    override
    nonReentrant
    returns (uint256)
{
    uint256 realAmount = _vault.withdraw(address(this), amount);
    address[] memory path = new address[](2);
    path[0] = address(_certToken);
    path[1] = _wBnbAddress;
    uint256[] memory outAmounts = _dex.getAmountsOut(realAmount, path);
    uint256[] memory amounts = _dex.swapExactTokensForETH(
        amount,
        outAmounts[1],
        path,
        recipient,
        block.timestamp + 300
    );
    //  console.log(); TODO CHECK BALANCE OF ROUTER
    emit Withdrawal(msg.sender, recipient, _wBnbAddress, amounts[1]);
    return amounts[1];
}
```

Solution

If it is not designed as expected, it is recommended to pass the calculation result of getAmountsOut in the front end as the amountOutMin parameter to participate in the swap operation or use an oracle.

Status

Fixed

[N7] [Suggestion] Redundant code issue

Category: Others

Content

In the HelioProvider contract, the daoBurn and daoMint functions are called by the DAOInteraction contract to burn and mint the collateralToken. But there is no interface for calling daoBurn and daoMint functions in the DAOInteraction contract.

Code location: contracts/ceros/HelioProvider.sol

```
function daoBurn(address account, uint256 value)
    external
    override
    onlyDao
    nonReentrant
{
    _collateralToken.burn(account, value);
}

function daoMint(address account, uint256 value)
    external
    override
    onlyDao
    nonReentrant
{
    _collateralToken.mint(account, value);
}
```

Solution

It is recommended to clarify design expectations.

Status

Fixed

[N8] [Medium] Compatibility issue

Category: Design Logic Audit

Content

In the CerosRouter contract, users can first withdraw aBNBc tokens from the vault contract through the withdraw function, and then withdraw BNB from the BinancePool contract through the unstakeCerts function. However, through the analysis of BinancePool and aBNBb contracts, it does not transfer BNB tokens to users but records the corresponding status through `_pendingBurn[account]`. This is inconsistent with the functionality described in the comments for the withdraw function.

Code location: contracts/ceros/CerosRouter.sol

```
function withdraw(address recipient, uint256 amount)
    external
    override
    nonReentrant
    returns (uint256 realAmount)
{
    realAmount = _vault.withdrawFor(msg.sender, address(this), amount);
    _pool.unstakeCerts(recipient, realAmount);
    emit Withdrawal(msg.sender, recipient, _wBnbAddress, realAmount);
    return realAmount;
}
```

Solution

It is recommended to clarify design expectations.

Status

Fixed; After communicating with the project team, the project team added the getPendingWithdrawalOf interface so that users can observe the amount of BNB that can be withdrawn.

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002205240004	SlowMist Security Team	2022.05.17 - 2022.05.24	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 critical risk, 4 medium risks, 2 low risks, and 1 suggestion. And 1 medium risk vulnerability was confirmed and fixed; 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