Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

On 2024.04.28, the SlowMist security team received the DeSyn Protocol team's security audit application for DeSyn - Restaking, 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.
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
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

Serial Number	Audit Class	Audit Subclass
7	Security Design 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

This is an audit of the Restaking module of the DeSyn protocol. It allows users to deposit the tokens obtained after depositing funds into a designated restaking protocol into an ETF.

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Enhanced `msg.value` check	Others	Suggestion	Fixed

NO	Title	Category	Level	Status
N2	Potential risks of approving denial of service	Denial of Service Vulnerability	High	Fixed
N3	Accidentally transferred funds that can be stolen	Others	Information	Acknowledged
N4	Risk of user assets being stolen due to arbitrary execution of data	Design Logic Audit	Critical	Fixed
N5	Decimal problems in share calculation	Arithmetic Accuracy Deviation Vulnerability	Suggestion	Acknowledged
N6	Issues where issueFee maybe 0	Design Logic Audit	Low	Acknowledged

4 Code Overview

4.1 Contracts Description

Audit Version:

https://github.com/Meta-DesynLab/desyn-modules-forge/blob/restaking/src/restaking/Restaking.sol

commit: 530af45ce53ea0a5e5c145f99bd6a58ff2a2b086

Fixed Version:

https://github.com/Meta-DesynLab/desyn-modules-forge/blob/restaking/src/restaking/Restaking.sol

commit: be394c78e88b035d9cb4a7aa102195fb31ecd2f7

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:

Restaking			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
invest	External	Payable	-
setDepositor	Public	Can Modify State	onlyOwner
batchSetDepositor	External	Can Modify State	onlyOwner
_calculateShare	Internal	-	-

4.3 Vulnerability Summary

[N1] [Suggestion] Enhanced `msg.value` check

Category: Others

Content

In the Restaking contract, users can deposit a specified token for restaking through the invest function. When the user passes in NATIVE_TOKEN as the rawToken, the contract checks whether the user's `msg.value` matches the expected value. However, when the user passes in a rawToken that is not NATIVE_TOKEN, there is no check to ensure that `msg.value` is 0. This may cause users to inadvertently transfer unnecessary native tokens.

Code location: src/restaking/Restaking.sol#L66

```
    function invest(RestakingParams memory restakingParams) external payable returns
(uint256) {
        ...
        if (restakingParams.rawToken != NATIVE_TOKEN) {
            ...
        } else {
            if (msg.value != restakingParams.depositValue) revert Mismatch();
            ...
        }
        ...
    }
```

Solution

It is recommended to check that `msg.value` is not 0 when the rawToken is not NATIVE_TOKEN.

Status

Fixed

[N2] [High] Potential risks of approving denial of service

Category: Denial of Service Vulnerability

Content

In the invest function, when the user passes in a rawToken that is neither NATIVE_TOKEN nor _WST_ETH token, the contract will call the user-provided depositor to execute custom user data. Prior to this, the contract approves the user's transferred funds to the depositor using the safeApprove function. It is important to note that the safeApprove function requires either the current approved amount to be 0 or the current allowance of the contract to the depositor to be 0 in order to safely approve; otherwise, it will revert. If a malicious user intentionally does not use up all the allowance during the depositor call, when the protocol attempts to execute safeApprove for this rawToken again, it will revert. This will cause a denial of service for some of the contract's functionality.

Code location: src/restaking/Restaking.sol#L73-L78

```
    function invest(RestakingParams memory restakingParams) external payable returns
(uint256) {
        ...
        if (restakingParams.rawToken != NATIVE_TOKEN) {
            if (restakingParams.rawToken == address(_WST_ETH)) {
                ...
                IERC20(address(_ST_ETH)).safeApprove(restakingParams.depositor,
stETHAmount);
                ...
            } else {
                IERC20(restakingParams.rawToken).safeTransferFrom(msg.sender,
address(this), restakingParams.depositValue);


IERC20(restakingParams.rawToken).safeApprove(restakingParams.depositor,
restakingParams.depositValue);
                restakingParams.depositor.functionCallWithValue(restakingParams.data,
0);
            }
        } else {
```

```
            ...
            if (restakingParams.target == address(_PUFFER_VAULT)) {
                ...
                IERC20(address(_ST_ETH)).safeApprove(restakingParams.depositor,
restakingParams.depositValue);
                ...
            } else {
                ...
            }
        }
        ...
        IERC20(restakingParams.target).safeApprove(restakingParams.etf, tokenRev);
        ...
    }
```

Solution

It is recommended to use the forceApprove function from a newer version of the SafeERC20 library instead of the safeApprove function to mitigate this risk.

Status

Fixed

[N3] [Information] Accidentally transferred funds that can be stolen

Category: Others

Content

In the invest function, when the user-specified rawToken is neither NATIVE_TOKEN nor _WST_ETH, the contract directly calls the depositor contract to execute user-specified data. When other users accidentally transfer rawTokens into the Restaking contract and the Restaking contract still has a remaining allowance for the depositor, these other users can deposit the accidentally transferred tokens into the depositor as their own profit.

It is obvious that the project team can also transfer out the erroneously transferred assets in the contract by approving a specific depositor.

Code location: restaking/src/restaking/Restaking.sol#L77,L87

```
    function invest(RestakingParams memory restakingParams) external payable returns
(uint256) {
        ...
        if (restakingParams.rawToken != NATIVE_TOKEN) {
            if (restakingParams.rawToken == address(_WST_ETH)) {
```

```
                ...
            } else {
                IERC20(restakingParams.rawToken).safeTransferFrom(msg.sender,
address(this), restakingParams.depositValue);


IERC20(restakingParams.rawToken).safeApprove(restakingParams.depositor,
restakingParams.depositValue);
                restakingParams.depositor.functionCallWithValue(restakingParams.data,
0);
            }
        } else {
            ...
            } else {
                restakingParams.depositor.functionCallWithValue(restakingParams.data,
restakingParams.depositValue);
            }
        }
        ...
    }
```

Solution

N/A

Status

Acknowledged

[N4] [Critical] Risk of user assets being stolen due to arbitrary execution of data

Category: Design Logic Audit

Content

In the invest function, when the user-specified rawToken is neither NATIVE_TOKEN nor _WST_ETH, the contract directly calls the depositor contract to execute user-specified data. It is important to note that in some restaking protocols (such as stETH and pufETH), the deposit, borrowing, and token transfer interfaces are in the same contract. Taking pufETH as an example, users can call the pufETH contract to make deposits and obtain pufETH tokens, and they can also call the pufETH contract to transfer pufETH tokens. Therefore, although the legality of the depositor is checked in the invest function, the `restakingParams.data` passed in by the user is not checked. This allows users to call the transferFrom function of the LST/LRT contract via

`restakingParams.depositor.functionCallWithValue` to transfer tokens of any user who has already approved this contract.

Code location: restaking/src/restaking/Restaking.sol#L77,L87

```
    function invest(RestakingParams memory restakingParams) external payable returns
(uint256) {
        ...
        if (restakingParams.rawToken != NATIVE_TOKEN) {
            if (restakingParams.rawToken == address(_WST_ETH)) {
                ...
            } else {
                IERC20(restakingParams.rawToken).safeTransferFrom(msg.sender,
address(this), restakingParams.depositValue);


IERC20(restakingParams.rawToken).safeApprove(restakingParams.depositor,
restakingParams.depositValue);
                restakingParams.depositor.functionCallWithValue(restakingParams.data,
0);
            }
        } else {
            ...
            } else {
                restakingParams.depositor.functionCallWithValue(restakingParams.data,
restakingParams.depositValue);
            }
        }
        ...
    }
```

Solution

It is strongly recommended not to use an open-ended calling approach to support other restaking protocols. If support for other protocols is needed, it is advisable to list the interfaces separately for calling to strictly check the data passed in by users.

Status

Fixed

[N5] [Suggestion] Decimal problems in share calculation

Category: Arithmetic Accuracy Deviation Vulnerability

Content

In the invest function, when a user completes a restaking operation, the contract deposits the restaking tokens into the user-specified ETF and transfers the ETF LP to the user. Prior to this, the contract calculates the amount of ETF LP the user can obtain when joining the pool through `_calculateShare`. It is important to note that if the user's deposit amount is relatively small, due to decimal errors during calculation, the amount of LP obtained during the joinPool operation may be 1 wei more than the amount calculated by `_calculateShare`. However, this does not affect the normal business logic of the protocol.

Code location: src/restaking/Restaking.sol#L140

```
    function _calculateShare(address etf, address token, uint256 amountIn) internal
view returns(uint256) {
        uint256 totalPoolShares = IETF(etf).totalSupply();

        IBpool bPool = IBpool(IETF(etf).bPool());
        uint256 totalTokenBalance = bPool.getBalance(token);

        (,,,,,,,,uint256 issueFeeRate,,) = IETF(etf).etfStatus();
        uint256 issueFee = issueFeeRate * 1000 / 1e18;

        uint256 share = (totalPoolShares - 1) * amountIn * (1000 - issueFee) / (1000
* (totalTokenBalance + 1));
        return share;
    }
```

Solution

In fact, this does not impact the normal business logic. If mitigation of this situation is needed, it is recommended to limit the minimum deposit amount for users.

Status

Acknowledged

[N6] [Low] Issues where issueFee maybe 0

Category: Design Logic Audit

Content

In the `_calculateShare` function, it calculates the amount of LP that can be obtained based on the amount of

tokens that need to be deposited into the ETF. In this process, it considers the minting fee charged when depositing into the ETF. However, it is important to note that if the user-specified ETF is closed-ended and the current `isCompletedCollect` state is false, the ETF will waive the minting fee for the depositor. This will cause the share calculated by the `_calculateShare` function to be smaller than expected, ultimately resulting in some funds remaining in the Restaking contract and not being deposited into the ETF.

Code location: src/restaking/Restaking.sol#L146

```
    function _calculateShare(address etf, address token, uint256 amountIn) internal
view returns(uint256) {
        uint256 totalPoolShares = IETF(etf).totalSupply();

        IBpool bPool = IBpool(IETF(etf).bPool());
        uint256 totalTokenBalance = bPool.getBalance(token);

        (,,,,,,,,uint256 issueFeeRate,,) = IETF(etf).etfStatus();
        uint256 issueFee = issueFeeRate * 1000 / 1e18;

        uint256 share = (totalPoolShares - 1) * amountIn * (1000 - issueFee) / (1000
* (totalTokenBalance + 1));
        return share;
    }
```

Solution

It is recommended to check the type of the user-specified ETF and the `isCompletedCollect` state to better calculate the ETF shares.

Status

Acknowledged

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002404290004	SlowMist Security Team	2024.04.28 - 2024.04.29	Low 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, 1 high risk, 1 low risk, 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.

Official Website

www.slowmist.com

E-mail

team@slowmist.com

Twitter

@SlowMist_Team

Github

https://github.com/slowmist