Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

On 2023.05.09, the SlowMist security team received the DeSyn Protocol team's security audit application for DeSyn LeverageStaking, 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:

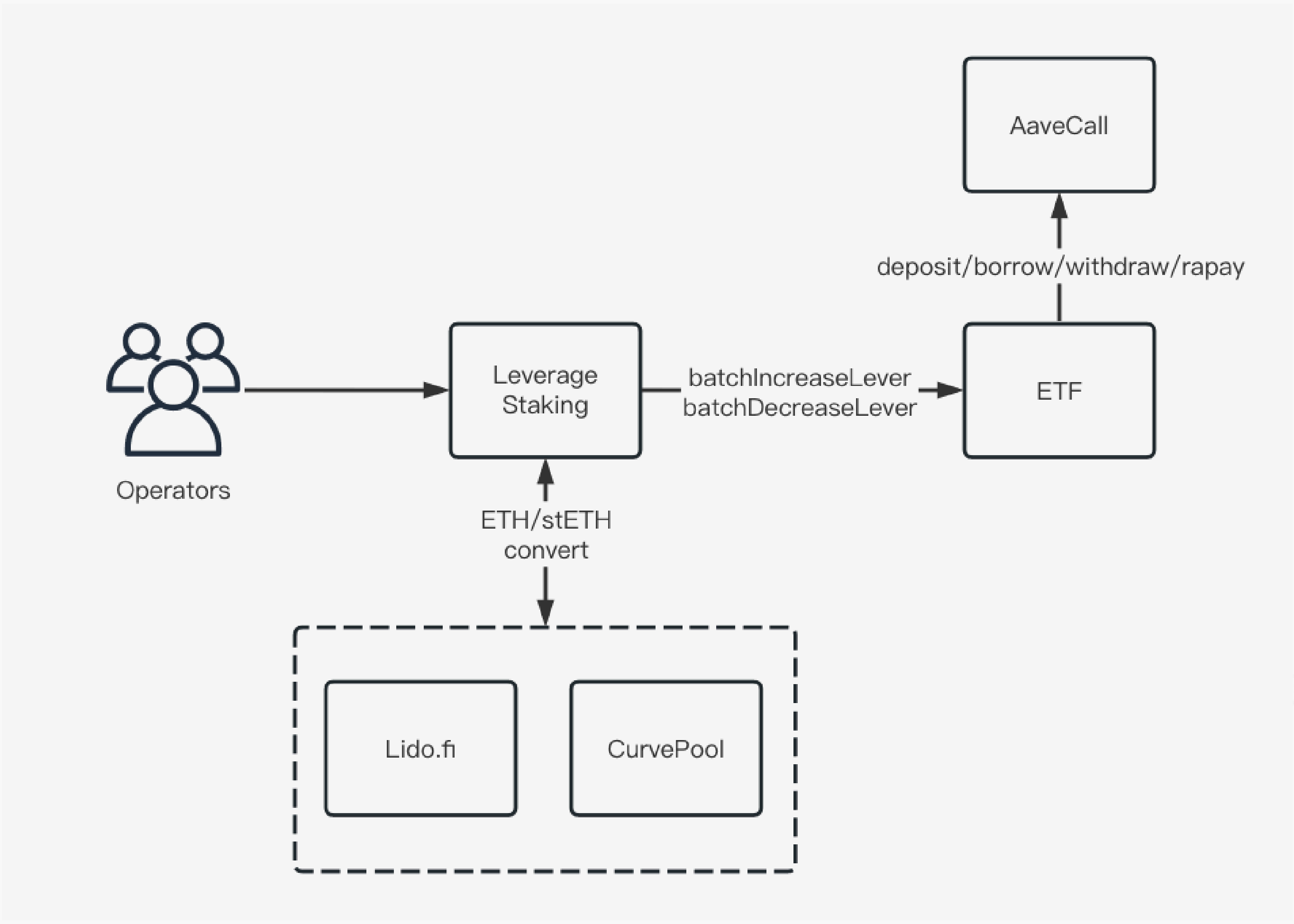
Serial Number	Audit Class	Audit Subclass
1	Overflow Audit	-
2	Reentrancy Attack Audit	-
3	Replay Attack Audit	-
4	Flashloan Attack Audit	-
5	Race Conditions Audit	Reordering Attack Audit
6	Permission Vulnerability Audit	Access Control Audit
		Excessive Authority Audit
7	Security Design Audit	External 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

Desyn is a web3 asset management platform that provides a decentralized asset management infrastructure for everyone around the world. This audit is the leverage staking module of the DeSyn protocol, which consists of two parts: AaveCall and LeverageStake. The operator converts WETH in the ETF into stETH through LeverageStake and deposits it in AAVE for revolving borrows to increase leverage. AaveCall library is used to provide interface support for operating ETF. The following is a brief architecture diagram:



3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Potential Unable to Borrow Issue	Design Logic Audit	Low	Fixed
N2	Library function visibility issue	Gas Optimization Audit	Suggestion	Fixed
N3	Function multiple logic mixes	Design Logic Audit	Low	Fixed
N4	Potential Malicious Liquidation ETF Issue	Design Logic Audit	High	Fixed
N5	min_dy without slippage and exchange fees	Design Logic Audit	Critical	Fixed

NO	Title	Category	Level	Status
N6	Potential Operator Arbitrage Risk	Design Logic Audit	Medium	Fixed
N7	Unreasonable defaultSlippage	Design Logic Audit	Critical	Fixed
N8	Redundant receive function	Others	Suggestion	Fixed
N9	Issues with not updating bound tokens	Design Logic Audit	Suggestion	Acknowledged
N10	Partial rebind issue	Design Logic Audit	Suggestion	Acknowledged

4 Code Overview

4.1 Contracts Description

Audit Version:

https://github.com/Meta-DesynLab/desyn-modules/tree/feature/leverageStaking/contracts/staking

commit: b2dbc563c8bb0f62b595ad55fb10dd6e8f98ff7a

Fixed Version:

https://github.com/Meta-DesynLab/desyn-modules/tree/feature/leverageStaking/contracts/staking

commit: 427e55c7dd682e95d2e50df43b8c72df87e2b21a

Audit Scope:

- contracts/staking/AaveCall.sol
- contracts/staking/LeverageStake.sol

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:

AaveCall			
Function Name	Visibility	Mutability	Modifiers
getDepositCalldata	Public	-	-
invokeDeposit	External	Can Modify State	-
getWithdrawCalldata	Public	-	-
invokeWithdraw	External	Can Modify State	-
getBorrowCalldata	Public	-	-
invokeBorrow	External	Can Modify State	-
getRepayCalldata	Public	-	-
invokeRepay	External	Can Modify State	-
getSetUserUseReserveAsCollateralCalldata	Public	-	-
invokeSetUserUseReserveAsCollateral	External	Can Modify State	-
getSwapBorrowRateModeCalldata	Public	-	-
invokeSwapBorrowRateMode	External	Can Modify State	-

LeverageStake			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
getAstETHBalance	Public	-	-
getBalanceSheet	Public	-	-
getStethBalance	Public	-	-
getLeverageInfo	Public	Can Modify State	-
deposit	Public	Can Modify State	-

LeverageStake			
borrow	Public	Can Modify State	-
withdraw	Public	Can Modify State	-
repayBorrow	Public	Can Modify State	-
batchIncreaseLever	External	Can Modify State	-
batchDecreaseLever	External	Can Modify State	-
increaseLever	Public	Can Modify State	-
decreaseLever	Public	Can Modify State	-
exchange	Public	Can Modify State	-
convertToAstEth	External	Can Modify State	-
convertToWeth	External	Can Modify State	-
setFactory	External	Can Modify State	onlyOwner
setBorrowRate	External	Can Modify State	onlyOwner
updateLendingPoolInfo	External	Can Modify State	onlyOwner
_updatePosition	Internal	Can Modify State	-
_checkTx	Internal	-	-
<Receive Ether>	External	Payable	-

4.3 Vulnerability Summary

[N1] [Low] Potential Unable to Borrow Issue

Category: Design Logic Audit

Content

In the LeverageStake contract, the batchDecreaseLever function reduces the leverage ratio of the ETF by withdrawing/repaying in AAVE, and may completely repay the debt in AAVE in the process. In this case, when the operator fully withdraws stETH from AAVE, the state of usingAsCollateral of the ETF in AAVE will be set to false. Theoretically, the aToken in the ETF will be 0 at this time, and usingAsCollateral will be automatically set to true when the ETF deposits in AAVE next time. But in fact, AAVE may still have 1wei of aToken left in the ETF due to arithmetic precision errors when calculating the number of aTokens that need to be burned by withdrawing the amount. Therefore, when the ETF deposits in AAVE next time, usingAsCollateral will not be set to true, and since there is no interface for calling the setUserUseReserveAsCollateral function in LeverageStake, this may cause the ETF to no longer be able to perform borrow operations.

Code location: contracts/staking/LeverageStake.sol

Solution

It is recommended to add an interface to call the invokeSetUserUseReserveAsCollateral function in the AaveCall library in the LeverageStake contract.

Status

Fixed

[N2] [Suggestion] Library function visibility issue

Category: Gas Optimization Audit

Content

In the protocol, the LeverageStake contract operates ETF through the interface of the AaveCall library, so the AaveCall library is stateless, and there is no need to set the visibility of the function to external/public.

Code location: contracts/staking/AaveCall.sol

```
function *(...) public (...) {...}
function *(...) external (...) {...}
```

Solution

It is recommended to modify the function visibility in the AaveCall library to internal.

Status

Fixed

[N3] [Low] Function multiple logic mixes

Category: Design Logic Audit

Content

In the getLeverageInfo function comment of the LeverageStake contract, it is explained that this function is used to obtain the fund status of the ETF in AAVE. But it not only gets it through the getUserAccountData function but also rebinds the ETF. These are two completely unrelated functions but used in the same function. And due to the openness of the getLeverageInfo function, any user can perform rebind operations through this function, which may be contrary to the design philosophy of ETF.

Code location: contracts/staking/LeverageStake.sol

```
function getLeverageInfo()
  public
  override
  returns (
    uint256 totalCollateralETH,
    uint256 totalDebtETH,
    uint256 availableBorrowsETH,
    uint256 currentLiquidationThreshold,
    uint256 ltv,
    uint256 healthFactor
  )
{
  if (IBpool(etf.bPool()).isBound(address(astETH))) {
    etf.invokeRebind(address(astETH), getAstETHBalance(), 50e18, true);
  }

  (
    totalCollateralETH,
    totalDebtETH,
    availableBorrowsETH,
    currentLiquidationThreshold,
    ltv,
    healthFactor
  ) = lendingPool.getUserAccountData(etf.bPool());
}
```

Solution

It is recommended to implement these two functions separately, and modify the getLeverageInfo function to a view function. Implement the function for the invokeRebind operation and perform the _checkTx check.

Status

Fixed

[N4] [High] Potential Malicious Liquidation ETF Issue

Category: Design Logic Audit

Content

In the LeverageStake contract, the operator can increase the leverage ratio of the ETF in AAVE through the borrow function. Since there is no maximum leverage limit, when the leverage ratio is too high and stETH is close to the liquidation line, the ETF may be liquidated due to the accumulation of loan interest. If the operator acts maliciously subjectively, the funds of users in the ETF will be at risk.

Code location: contracts/staking/LeverageStake.sol

```
function borrow(uint256 amount, uint16 referralCode) public override {
  _checkTx();

  etf.invokeBorrow(lendingPool, address(WETH), amount, 2, referralCode);

  etf.invokeUnwrapWETH(address(WETH), amount);
}
```

Solution

It is recommended to limit the maximum leverage, or ensure that the health coefficient of the ETF is maintained at a certain level.

Status

Fixed

[N5] [Critical] min_dy without slippage and exchange fees

Category: Design Logic Audit

Content

In the LeverageStake contract, the increaseLever and convertToAstEth functions all have the function of exchanging stETH and ETH tokens through the exchange function. However, the `min_dy` parameter passed in is consistent with the `dx` parameter. Due to the existence of slippage and exchange fees, a certain amount of stETH cannot be exchanged for exactly the same amount of ETH tokens. Therefore, the exchange function in these functions cannot be executed normally.

Code location: contracts/staking/LeverageStake.sol

```
function increaseLever(
  uint256 amount,
  uint16 referralCode,
  bool isTrade
) public override returns (uint256) {
  ...
  if (isTrade) {
    exchange(0, 1, amount, amount);
  } else {
    etf.invokeMint(address(stETH), address(0), amount);
  }
  ...
}

function convertToAstEth(bool isTrade) external override {
  ...
  if (isTrade) {
    receivedStEth = exchange(0, 1, convertedAmount, convertedAmount);
  } else {
    etf.invokeMint(address(stETH), address(0), convertedAmount);

    receivedStEth = stETH.balanceOf(etf.bPool());
  }
  ...
}
```

Solution

It is recommended to pass in a reasonable `min_dy` after considering the impact of slippage and exchange fees.

Status

Fixed

[N6] [Medium] Potential Operator Arbitrage Risk

Category: Design Logic Audit

Content

In the LeverageStake contract, the operator can exchange stETH and ETH tokens through the exchange function, but its min_dy parameter is also set by the operator. Therefore, if the caller does not pass in min_dy, the exchange operation of the ETF in the Curve Pool may suffer from a sandwich attack, resulting in the loss of ETF assets.

Code location: contracts/staking/LeverageStake.sol

```
function exchange(
  int128 i,
  int128 j,
  uint256 dx,
  uint256 min_dy
) public override returns (uint256) {
  ...

  bytes memory methodData = abi.encodeWithSignature(
    'exchange(int128,int128,uint256,uint256)',
    i,
    j,
    dx,
    min_dy
  );

  ...
}
```

Solution

If the exchange function only serves other functions in the contract, its visibility can be changed to internal. Or do not allow the caller to pass in min_dy, but calculate a reasonable min_dy through dx.

Status

Fixed

[N7] [Critical] Unreasonable defaultSlippage

Category: Design Logic Audit

Content

There is a defaultSlippage variable in the LeverageStake contract, which is used in the decreaseLever and convertToWeth functions to exchange the minimum received amount between stETH and ETH tokens. It defaults to 1 and cannot be modified, which will cause min_dy to be 1% of dx during the exchange operation, making the token exchange process vulnerable to sandwich attacks.

Code location: contracts/staking/LeverageStake.sol

```
  uint256 public defaultSlippage = 1;

  function decreaseLever(uint256 amount) public override returns (uint256, bool) {
      ...
      uint256 receivedETH = exchange(1, 0, amount,
 amount.mul(defaultSlippage).div(100));
      ...
  }

  function convertToWeth() external override {
    ...
    uint256 receivedETH = exchange(
      1,
      0,
      withdrawnAmount,
      withdrawnAmount.mul(defaultSlippage).div(100)
    );
    ...
  }
```

Solution

It is recommended to modify it to 99 or other reasonable values.

Status

Fixed

[N8] [Suggestion] Redundant receive function

Category: Others

Content

There is a receive function in the LeverageStake contract to enable the contract to receive native tokens. However, in actual business, the contract does not need to receive native tokens, so the receive function is redundant, which may also cause users to mistakenly transfer native tokens to this contract and then fail to withdraw them.

Code location: contracts/staking/LeverageStake.sol

```
  receive() external payable {}
```

Solution

It is recommended to remove the receive function.

Status

Fixed

[N9] [Suggestion] Issues with not updating bound tokens

Category: Design Logic Audit

Content

In the LeverageStake contract, the deposit, borrow, withdraw, and repayBorrow functions are used to operate the ETF to deposit, borrow, withdraw, and repay to AAVE, respectively. However, the `_records` of the bound tokens in the ETF are not updated in the above operations. This will cause the token `_records` in the ETF to be skewed after the operator operates through the above function.

Code location: contracts/staking/LeverageStake.sol

```
function deposit(uint256 amount) public override returns (uint256) {
  ...
}

function borrow(uint256 amount, uint16 referralCode) public override {
  ...
}

function withdraw(uint256 amount) public override {
  ...
}

function repayBorrow(uint256 amount) public override returns (uint256) {
  ...
}
```

Solution

If it is not the expected design, it is recommended to modify the token `_records` status in the ETF while performing the above operations.

Status

Acknowledged; After communicating with the project team, the project team stated that only the `_records` of

WETH and astETH tokens in the ETF will be updated, and the rest of the liabilities and other fragment tokens will not be updated.

[N10] [Suggestion] Partial rebind issue

Category: Design Logic Audit

Content

In the LeverageStake contract, the increaseLever and decreaseLever functions respectively increase/decrease the leverage ratio of the ETF in AAVE, and at the same time rebind the astETH tokens in the ETF, but do not update the `_records` status of ETH and stETH in the ETF, which will cause The `_records` state does not match the actual token balance in the ETF.

Code location: contracts/staking/LeverageStake.sol

```
function increaseLever(
  uint256 amount,
  uint16 referralCode,
  bool isTrade
) public override returns (uint256) {
  ...

  etf.invokeRebind(address(astETH), getAstETHBalance(), 50e18, true);

  ...
}

function decreaseLever(uint256 amount) public override returns (uint256, bool) {
    ...

    etf.invokeRebind(address(astETH), getAstETHBalance(), 50e18, true);

    ...
}
```

Solution

If the unexpected design suggestion is as mentioned in N9, directly update the `_records` state of the ETF in the deposit/borrow/withdraw/repayBorrow function.

Status

Acknowledged; After communicating with the project team, the project team stated that only the `_records` of

WETH and astETH tokens in the ETF will be updated, and the rest of the liabilities and other fragment tokens will not be updated.

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002305110002	SlowMist Security Team	2023.05.09 - 2023.05.11	Passed

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 2 critical risks, 1 high risk, 1 medium risk, 2 low risks, 4 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.

SLOWMIST

Official Website

www.slowmist.com

E-mail

team@slowmist.com

Twitter

@SlowMist_Team

Github

https://github.com/slowmist