

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 2026.08.19, the SlowMist security team received the PrimeUSD team's security audit application for Boring Vault Incremental 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
7	Security Design Audit	External Module Safe Use Audit
		Compiler Version Security Audit

Serial Number	Audit Class	Audit Subclass
7	Security Design 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

The audited project is the Boring Vault contract of the PrimeUSD protocol. This incremental audit focuses on the decoder and sanitizer contracts for the protocol's Pendle Router V4, Circle CCTP, Morpho Market, and Nest integrations.

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Unchecked `destinationCaller` May Permanently Lock Cross-Chain Funds	Design Logic Audit	Low	Fixed
N2	Hardcoded CCTP Fee Limits May Cause Cross-Chain Transfers to Fail	Design Logic Audit	Low	Fixed
N3	Unbound `lltv` Allows Interaction with Unintended Morpho Markets	Design Logic Audit	Low	Fixed
N4	Pendle Limit-Order Comments Do Not Match the Implementation	Others	Suggestion	Fixed
N5	External Aggregation Router Addresses Are Not Restricted	Design Logic Audit	Medium	Fixed
N6	Nested Aggregator Recipient Addresses Are Not Validated	Design Logic Audit	Medium	Fixed
N7	Strategist Can Benefit from Self-Trading Limit Orders Against the Vault	Authority Control Vulnerability Audit	Information	Acknowledged

NO	Title	Category	Level	Status
N8	Strategist Can Redirect Vault Assets Through Kyber Fee Parameters	Authority Control Vulnerability Audit	Information	Acknowledged
N9	Risk of Excessive Privilege	Authority Control Vulnerability Audit	Medium	Acknowledged

4 Code Overview

4.1 Contracts Description

https://github.com/SwellNetwork/boring-vault/

90aa954615e8d7a2104af785e57bb09e6c807591

Initial Audit Version: 90aa954615e8d7a2104af785e57bb09e6c807591

Final Audit Version: 3209585e78aa69d4326740a6538fda5476e27244

Audit scope:

```
src/base/DecodersAndSanitizers/hyperwave/common/PendleRouterV4DecoderAndSanitizer.sol
src/base/DecodersAndSanitizers/hyperwave/common/CCTPDecoderAndSanitizer.sol
src/base/DecodersAndSanitizers/hyperwave/common/MorphoMarketDecoderAndSanitizer.sol
src/base/DecodersAndSanitizers/hyperwave/aprimeusd/NestDecoderAndSanitizer.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:

NestDecoderAndSanitizer			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
deposit	External	-	-
requestRedeem	External	-	-
redeem	External	-	-

CCTPDecoderAndSanitizer			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
depositForBurn	External	-	-
bytes32ToAddress	Internal	-	-

MorphoMarketDecoderAndSanitizer			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
supplyCollateral	External	-	-
withdrawCollateral	External	-	-
borrow	External	-	-
repay	External	-	-

PendleRouterV4DecoderAndSanitizer			
Function Name	Visibility	Mutability	Modifiers

PendleRouterV4DecoderAndSanitizer			
<Constructor>	Public	Can Modify State	-
swapExactTokenForPt	External	-	-
swapExactPtForToken	External	-	-
addLiquiditySingleToken	External	-	-
addLiquiditySinglePt	External	-	-
addLiquidityDualTokenAndPt	External	-	-
removeLiquidityDualTokenAndPt	External	-	-
removeLiquiditySingleToken	External	-	-
removeLiquiditySinglePt	External	-	-
redeemPyToToken	External	-	-
redeemDueInterestAndRewards	External	-	-
swapTokensToTokens	External	-	-

4.3 Vulnerability Summary

[N1] [Low] Unchecked `destinationCaller` May Permanently Lock Cross-Chain Funds

Category: Design Logic Audit

Content

In `CCTPDecoderAndSanitizer`, `destinationCaller` restricts which address may call `receiveMessage()` on the destination chain, but the current Decoder neither validates this parameter nor includes it in Merkle verification. Consequently, a Strategist can set it to an incorrect or uncontrollable address, preventing other Relayers from submitting the message. Because the source-chain tokens have already been burned and CCTP V2 does not

support replacing `destinationCaller`, the destination-chain tokens may never be minted, resulting in a loss of funds.

- src/base/DecodersAndSanitizers/hyperwave/common/CCTPDecoderAndSanitizer.sol#L13-L20

```
function depositForBurn(
    uint256 amount,
    uint32 destinationDomain,
    bytes32 mintRecipient,
    address burnToken,
    bytes32 /* destinationCaller */,
    uint256 maxFee,
    uint32 minFinalityThreshold
) external view returns (bytes memory) {
```

Solution

Require `destinationCaller` to equal `bytes32(0)`, or include the permitted `destinationCaller` values in Merkle verification.

Status

Fixed

[N2] [Low] Hardcoded CCTP Fee Limits May Cause Cross-Chain Transfers to Fail

Category: Design Logic Audit

Content

`CCTPDecoderAndSanitizer` hardcodes the Fast Transfer `maxFee` cap at approximately 1 bps and requires a zero `maxFee` for Standard Transfers, while Circle CCTP fees adjust dynamically according to the network and route. If fees increase, the Strategist cannot set a sufficient `maxFee`, causing `depositForBurn()` to revert continuously and making the cross-chain functionality unavailable.

- src/base/DecodersAndSanitizers/hyperwave/common/CCTPDecoderAndSanitizer.sol#L29-L40

```
// Standard transfer requires minFinalityThreshold of 2000 and no maxFee
if (minFinalityThreshold == 2000 && maxFee > 0) {
    revert CCTPDecoderAndSanitizer_InvalidMaxFee();
}

// Fast transfer requires minFinalityThreshold of 1000 and 1 bps fee
if (minFinalityThreshold == 1000) {
    uint256 requiredMaxFee = (amount * 1) / 10_000 + 1;
    if (maxFee > requiredMaxFee) {
        revert CCTPDecoderAndSanitizer_InvalidMaxFee();
    }
}
```

Solution

Define a reasonable and adjustable `maxFee` range based on current Circle CCTP fees to accommodate fee changes while limiting abnormally high fees.

Status

Fixed

[N3] [Low] Unbound `lltv` Allows Interaction with Unintended Morpho Markets

Category: Design Logic Audit

Content

In `MorphoMarketDecoderAndSanitizer`, `lltv` forms part of both a Morpho market's unique identifier and its risk configuration, but the current Decoder does not bind it to the Merkle leaf. An authorized Strategist can therefore use the same loan asset, collateral asset, Oracle, and IRM to interact with unintended markets that use other enabled LLTV values, bypassing the market scope predefined by governance.

- src/base/DecodersAndSanitizers/hyperwave/common/MorphoMarketDecoderAndSanitizer.sol#L13-L31

```
function supplyCollateral(
    DecoderCustomTypes.MarketParams memory marketParams,
```

```
    uint256 /* assets */,
    address onBehalf,
    bytes calldata data
) external view returns (bytes memory addressesFound) {
    if (onBehalf != boringVault) {
        revert MorphoDecoderAndSanitizer_OnlyBoringVault();
    }
    if (data.length != 0) {
        revert MorphoDecoderAndSanitizer_InvalidData();
    }

    addressesFound = abi.encodePacked(
        marketParams.loanToken,
        marketParams.collateralToken,
        marketParams.oracle,
        marketParams.irm
    );
}
```

Solution

Include `lltv` in the Decoder output and Merkle leaf validation to ensure that the Strategist can interact only with explicitly authorized Morpho markets.

Status

Fixed

[N4] [Suggestion] Pendle Limit-Order Comments Do Not Match the Implementation

Category: Others

Content

In `PendleRouterV4DecoderAndSanitizer`, the function comments state that limit orders are unsupported, while the current implementation and Merkle configuration allow limit orders to be executed through `LimitRouter`.

- src/base/DecodersAndSanitizers/hyperwave/common/PendleRouterV4DecoderAndSanitizer.sol#L17-L35, L38-L55

```
// @desc Function to swap exact token for Pendle Pt, will revert if using limit orders
function swapExactTokenForPt(
    address receiver,
    address market,
    uint256 /* minPtOut */,
    DecoderCustomTypes.ApproxParams calldata /* guessPtOut */,
    DecoderCustomTypes.TokenInput calldata input,
    DecoderCustomTypes.LimitOrderData calldata limit
) external view virtual returns (bytes memory addressFound) {
    //...
}

// @desc Function to withdraw from Pendle PT tokens, does not support limit orders
function swapExactPtForToken(
    address receiver,
    address market,
    uint256 /* exactPtIn */,
    DecoderCustomTypes.TokenOutput calldata output,
    DecoderCustomTypes.LimitOrderData calldata limit
) external view virtual returns (bytes memory addressesFound) {
    //...
}
```

Solution

Update or remove the relevant comments so that they accurately reflect the implementation's current support for limit orders.

Status

Fixed

[N5] [Medium] External Aggregation Router Addresses Are Not Restricted

Category: Design Logic Audit

Content

When `PendleRouterV4DecoderAndSanitizer` performs aggregated swaps using `TokenInput` or

`TokenOutput`, `swapData.extRouter` is not included in Merkle verification. A malicious or compromised Strategist can set it to an attacker-controlled address, causing `PendleSwap` to approve swap assets to that address and execute arbitrary calldata. The malicious router can transfer all or most of the assets while returning few or none of the expected tokens; if the relevant minimum-output restriction is zero or too low, the transaction may still succeed and cause a loss of Vault funds.

- src/base/DecodersAndSanitizers/hyperwave/common/PendleRouterV4DecoderAndSanitizer.sol#L18-L36, L39-L56, L60-L77, L93-L108, L110-L125, L127-L144, L161-L176, L209-L230

```
function swapExactTokenForPt(
    address receiver,
    address market,
    uint256 /* minPtOut */,
    DecoderCustomTypes.ApproxParams calldata /* guessPtOut */,
    DecoderCustomTypes.TokenInput calldata input,
    DecoderCustomTypes.LimitOrderData calldata limit
) external view virtual returns (bytes memory addressFound) {
    //...
}

function swapExactPtForToken(
    address receiver,
    address market,
    uint256 /* exactPtIn */,
    DecoderCustomTypes.TokenOutput calldata output,
    DecoderCustomTypes.LimitOrderData calldata limit
) external view virtual returns (bytes memory addressesFound) {
    //...
}

function addLiquiditySingleToken(
    address receiver,
    address market,
    uint256 /* minLpOut */,
    DecoderCustomTypes.ApproxParams calldata /* guessPtReceivedFromSy */,
```

```
    DecoderCustomTypes.TokenInput calldata input,
    DecoderCustomTypes.LimitOrderData calldata limit
) external view returns (bytes memory addressesFound) {
    //...
}

function addLiquidityDualTokenAndPt(
    address receiver,
    address market,
    DecoderCustomTypes.TokenInput calldata input,
    uint256 /* netPtDesired */,
    uint256 /* minLpOut */
) external view returns (bytes memory addressesFound) {
    //...
}

function removeLiquidityDualTokenAndPt(
    address receiver,
    address market,
    uint256 /* netLpToRemove */,
    DecoderCustomTypes.TokenOutput calldata output,
    uint256 /* minPtOut */
) external view returns (bytes memory addressesFound) {
    //...
}

function removeLiquiditySingleToken(
    address receiver,
    address market,
    uint256 /* netLpToRemove */,
    DecoderCustomTypes.TokenOutput calldata output,
    DecoderCustomTypes.LimitOrderData calldata limit
) external view returns (bytes memory addressesFound) {
    //...
}

function redeemPyToToken(
    address receiver,
```

```
    address YT,
    uint256 /* netPyIn */,
    DecoderCustomTypes.TokenOutput calldata output
) external view returns (bytes memory addressesFound) {
    //...
}

function swapTokensToTokens(
    address pendleSwap,
    DecoderCustomTypes.SwapDataExtra[] calldata swaps,
    uint256[] calldata /* netSwaps */
) external pure returns (bytes memory addressesFound) {
    //...
}
```

Solution

Include `swapData.extRouter` from each affected function in `addressFound` so that it is verified by the Merkle tree against preapproved external router addresses.

Status

Fixed

[N6] [Medium] Nested Aggregator Recipient Addresses Are Not Validated

Category: Design Logic Audit

Content

Although `PendleRouterV4DecoderAndSanitizer` restricts the outer recipient address, it does not include the aggregator recipient encoded in `TokenInput.swapData.extCalldata` or `TokenOutput.swapData.extCalldata` in Merkle verification. A malicious or compromised Strategist can reuse a valid Merkle proof while redirecting the swapped assets to an arbitrary address; on call paths that permit a zero minimum output, this may result in a loss of Vault funds.

- src/base/DecodersAndSanitizers/hyperwave/common/PendleRouterV4DecoderAndSanitizer.sol#L18-L36, L39-L56, L60-L77, L93-L108, L110-L125, L127-L144, L161-L176

```
function swapExactTokenForPt(
    address receiver,
    address market,
    uint256 /* minPtOut */,
    DecoderCustomTypes.ApproxParams calldata /* guessPtOut */,
    DecoderCustomTypes.TokenInput calldata input,
    DecoderCustomTypes.LimitOrderData calldata limit
) external view virtual returns (bytes memory addressFound) {
    //...
}

function swapExactPtForToken(
    address receiver,
    address market,
    uint256 /* exactPtIn */,
    DecoderCustomTypes.TokenOutput calldata output,
    DecoderCustomTypes.LimitOrderData calldata limit
) external view virtual returns (bytes memory addressesFound) {
    //...
}

function addLiquiditySingleToken(
    address receiver,
    address market,
    uint256 /* minLpOut */,
    DecoderCustomTypes.ApproxParams calldata /* guessPtReceivedFromSy */,
    DecoderCustomTypes.TokenInput calldata input,
    DecoderCustomTypes.LimitOrderData calldata limit
) external view returns (bytes memory addressesFound) {
    //...
}

function addLiquidityDualTokenAndPt(
    address receiver,
```

```
    address market,
    DecoderCustomTypes.TokenInput calldata input,
    uint256 /* netPtDesired */,
    uint256 /* minLpOut */
) external view returns (bytes memory addressesFound) {
    //...
}

function removeLiquidityDualTokenAndPt(
    address receiver,
    address market,
    uint256 /* netLpToRemove */,
    DecoderCustomTypes.TokenOutput calldata output,
    uint256 /* minPtOut */
) external view returns (bytes memory addressesFound) {
    //...
}

function removeLiquiditySingleToken(
    address receiver,
    address market,
    uint256 /* netLpToRemove */,
    DecoderCustomTypes.TokenOutput calldata output,
    DecoderCustomTypes.LimitOrderData calldata limit
) external view returns (bytes memory addressesFound) {
    //...
}

function redeemPyToToken(
    address receiver,
    address YT,
    uint256 /* netPyIn */,
    DecoderCustomTypes.TokenOutput calldata output
) external view returns (bytes memory addressesFound) {
    //...
}

function swapTokensToTokens(
```

```
    address pendleSwap,
    DecoderCustomTypes.SwapDataExtra[] calldata swaps,
    uint256[] calldata /* netSwaps */
) external pure returns (bytes memory addressesFound) {
    //...
}
```

Solution

Parse `swapData.extCalldata` in `TokenInput` and `TokenOutput`, extract the external aggregator's final recipient address, and require it to be Pendle Router V4. Revert on unrecognized aggregator functions or calldata.

Status

Fixed

[N7] [Information] Strategist Can Benefit from Self-Trading Limit Orders Against the Vault

Category: Authority Control Vulnerability Audit

Content

The Strategist controls the content and execution price of limit orders, and the Decoder does not restrict them. A malicious or compromised Strategist can place a materially unfavorable order through an address under its control and then direct the Vault to fill that order as the taker, transferring value from the Vault to the controlled maker address through the price difference and causing a loss of Vault funds. This risk depends on the trust placed in the Strategist.

- src/base/DecodersAndSanitizers/hyperwave/common/PendleRouterV4DecoderAndSanitizer.sol#L18-L36, L39-L56, L60-L77, L127-L144

```
function swapExactTokenForPt(
    address receiver,
    address market,
    uint256 /* minPtOut */,
    DecoderCustomTypes.ApproxParams calldata /* guessPtOut */,
    DecoderCustomTypes.TokenInput calldata input,
```

```
    DecoderCustomTypes.LimitOrderData calldata limit
) external view virtual returns (bytes memory addressFound) {
    //...
}

function swapExactPtForToken(
    address receiver,
    address market,
    uint256 /* exactPtIn */,
    DecoderCustomTypes.TokenOutput calldata output,
    DecoderCustomTypes.LimitOrderData calldata limit
) external view virtual returns (bytes memory addressesFound) {
    //...
}

function addLiquiditySingleToken(
    address receiver,
    address market,
    uint256 /* minLpOut */,
    DecoderCustomTypes.ApproxParams calldata /* guessPtReceivedFromSy */,
    DecoderCustomTypes.TokenInput calldata input,
    DecoderCustomTypes.LimitOrderData calldata limit
) external view returns (bytes memory addressesFound) {
    //...
}

function removeLiquiditySingleToken(
    address receiver,
    address market,
    uint256 /* netLpToRemove */,
    DecoderCustomTypes.TokenOutput calldata output,
    DecoderCustomTypes.LimitOrderData calldata limit
) external view returns (bytes memory addressesFound) {
    //...
}
```

Solution

Treat the Strategist as a highly trusted role and mitigate the risk through multisignature approval, privilege separation, and monitoring for anomalous executions.

Status

Acknowledged

[N8] [Information] Strategist Can Redirect Vault Assets Through Kyber Fee Parameters

Category: Authority Control Vulnerability Audit

Content

The Strategist can control Kyber's `minReturnAmount`, `feeReceivers`, and `feeAmounts`. A malicious or compromised Strategist could specify an attacker-controlled fee recipient, configure excessive fees, and set a low minimum output, allowing the transaction to succeed while most Vault assets are transferred as fees.

This risk depends on a malicious or compromised Strategist and is therefore considered a Strategist trust risk.

- src/base/DecodersAndSanitizers/hyperwave/common/PendleRouterV4DecoderAndSanitizer.sol#L18-L36, L39-L56, L60-L77, L127-L144

```
function swapExactTokenForPt(
    address receiver,
    address market,
    uint256 /* minPtOut */,
    DecoderCustomTypes.ApproxParams calldata /* guessPtOut */,
    DecoderCustomTypes.TokenInput calldata input,
    DecoderCustomTypes.LimitOrderData calldata limit
) external view virtual returns (bytes memory addressFound) {
    //...
}

function swapExactPtForToken(
    address receiver,
    address market,
    uint256 /* exactPtIn */,
```

```
    DecoderCustomTypes.TokenOutput calldata output,
    DecoderCustomTypes.LimitOrderData calldata limit
) external view virtual returns (bytes memory addressesFound) {
    //...
}

function addLiquiditySingleToken(
    address receiver,
    address market,
    uint256 /* minLpOut */,
    DecoderCustomTypes.ApproxParams calldata /* guessPtReceivedFromSy */,
    DecoderCustomTypes.TokenInput calldata input,
    DecoderCustomTypes.LimitOrderData calldata limit
) external view returns (bytes memory addressesFound) {
    //...
}

function removeLiquiditySingleToken(
    address receiver,
    address market,
    uint256 /* netLpToRemove */,
    DecoderCustomTypes.TokenOutput calldata output,
    DecoderCustomTypes.LimitOrderData calldata limit
) external view returns (bytes memory addressesFound) {
    //...
}
```

Solution

The project may either trust the Strategist and leave these parameters unrestricted while monitoring for abnormal fees and swap transactions, or impose restrictions on fee recipients, fee rates, and minimum outputs at the cost of reduced Kyber routing flexibility and increased operational overhead.

Status

Acknowledged

[N9] [Medium] Risk of Excessive Privilege

Category: Authority Control Vulnerability Audit

Content

In the `PendleRouterV4DecoderAndSanitizer` contract, `GOVERNANCE_ROLE` can add or remove approved Kyber `callTarget` addresses and manage accounts holding the same role.

- src/base/DecodersAndSanitizers/hyperwave/common/PendleRouterV4DecoderAndSanitizer.sol#L68-L73

```
function setKyberswapCallTarget(
    address callTarget,
    bool allowed
) external onlyRole(GOVERNANCE_ROLE) {
    _setKyberswapCallTarget(callTarget, allowed);
}
```

Solution

In the short term, transferring ownership and privileged roles to a multi-signature wallet or Timelock contract is recommended to mitigate single-point-of-failure risks. In the long term, privilege separation should assign upgrades, trust-anchor changes, compliance actions, fee policy, and emergency controls to distinct constrained roles. Critical verifier and implementation changes should use pending-and-accept activation with an on-chain delay that gives users and operators time to react.

Status

Acknowledged

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002608200002	SlowMist Security Team	2026.08.19 - 2026.08.20	Medium Risk

Summary conclusion: The SlowMist security team use a manual and SlowMist team's analysis tool to audit the project, during the audit work we found 3 medium-risk vulnerabilities, 3 low-risk vulnerabilities, 1 suggestion, and 2 informational vulnerabilities.

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