



# Smart Contract Security Audit Report



# Table Of Contents

<b>1 Executive Summary</b>	_____
<b>2 Audit Methodology</b>	_____
<b>3 Project Overview</b>	_____
3.1 Project Introduction	_____
3.2 Vulnerability Information	_____
<b>4 Code Overview</b>	_____
4.1 Contracts Description	_____
4.2 Visibility Description	_____
4.3 Vulnerability Summary	_____
<b>5 Audit Result</b>	_____
<b>6 Statement</b>	_____

# 1 Executive Summary

On 2024.06.28, the SlowMist security team received the Doubler team's security audit application for Doubler Lite, 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

Double Lite is an asset investment contract.

### 3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Unchecked asset type in inputEth function allows potential asset mismatch	Others	Low	Fixed

NO	Title	Category	Level	Status
N2	Issue with fee allocation in the _limitMint function	Others	Low	Fixed
N3	Risk of excessive authority	Authority Control Vulnerability Audit	Low	Acknowledged
N4	Potential bypass Issue with onlyOncePerBlock	Others	Medium	Fixed
N5	Abnormal implementation logic in getPooledByShares	Others	Low	Fixed
N6	Potential risk due to the unique nature of 10xBToken	Others	Information	Acknowledged
N7	Impact of inflationary or deflationary tokens on the doubler lite economic model	Others	Information	Acknowledged

## 4 Code Overview

### 4.1 Contracts Description

<https://github.com/doublerpro/doubler-lite>

Commit: 60b877bd1bc6118879b24e1a971f01f29ec8667a

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:

Doubler			
Function Name	Visibility	Mutability	Modifiers

Doublor			
<Constructor>	Public	Can Modify State	Ownable
initializeDoublor	External	Can Modify State	onlyOwner
getPrivateVar	External	-	-
updateLowerOfInputMaximum	External	Can Modify State	onlyOwner
_concatenate	Private	-	-
newPool	External	Can Modify State	onlyOwner
_checkPoolParam	Internal	-	-
updatePool	External	Can Modify State	onlyOwner
_getAssetBalance	Private	-	-
_getAssetAvg	Private	-	-
_getCurPrice	Internal	-	-
_getUnitSize	Internal	-	-
_getTokenPoolTotal	Internal	-	-
inputEth	External	Payable	nonReentrant onlyOncePerBlock onlyAsset
input	External	Can Modify State	nonReentrant onlyOncePerBlock onlyAsset
_checkInputQAmount	Internal	-	-
_input	Internal	Can Modify State	-
_getSrvFeeAddr	Internal	-	-
_mintABCToken	Internal	Can Modify State	-
_getWithdrawAssetAmount	Internal	-	-

Doublor			
_getSpendBAmount	Internal	-	-
withdraw	External	Can Modify State	nonReentrant onlyOncePerBlock onlyAsset
_checkWithdrawLimit	Internal	-	-
_checkAmountSlip	Internal	-	-
redeem	External	Can Modify State	nonReentrant onlyOncePerBlock onlyAsset
getBTokenPoolTotal	External	-	-
_rebaseBToken	Internal	Can Modify State	-
rebaseCToken	External	Can Modify State	nonReentrant
_rebaseCToken	Internal	Can Modify State	-
getPool	External	-	-

RBToken			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	Ownable
name	Public	-	-
symbol	Public	-	-
decimals	Public	-	-
totalSupply	External	-	-
totalShare	External	-	-
balanceOf	External	-	-
getUserShare	External	-	-
transfer	External	Can Modify State	-



RBTOKEN			
allowance	External	-	-
approve	External	Can Modify State	-
transferFrom	External	Can Modify State	-
_getSharesByPooledToken	Private	-	-
_getPooledTokenByShares	Private	-	-
_transfer	Internal	Can Modify State	-
_approve	Internal	Can Modify State	-
_spendAllowance	Internal	Can Modify State	-
_transferShares	Internal	Can Modify State	-
mint	External	Can Modify State	onlyOwner
limitMint	External	Can Modify State	onlyOwner
_limitMint	Internal	Can Modify State	-
burnFrom	External	Can Modify State	onlyOwner
_emitTransferEvents	Internal	Can Modify State	-
rebase	External	Can Modify State	onlyOwner
getPooledByShares	Public	-	-
transferShares	External	Can Modify State	-
transferSharesFrom	External	Can Modify State	-

FastPriceFeed			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
isSupported	External	-	-

FastPriceFeed			
getPriceFeeds	External	-	onlyAsset
getAssetPlan	External	-	-
setAssetPriceLimit	External	Can Modify State	onlyAsset onlyRole
setPriceFeedTimeLimit	External	Can Modify State	onlyAsset onlyRole
_checkTimeLimit	Internal	-	-
_checkPriceRange	Internal	-	-
_checkAddrZero	Internal	-	-
_checkAssetPlanExist	Internal	-	-
newAsset	External	Can Modify State	onlyRole
switchPriceFeed	External	Can Modify State	onlyAsset onlyRole
_initChainLinkFeedCheck	Internal	-	-
_getPriceFromDex	Internal	-	onlyAsset
_mockDexPrice	Internal	-	-
_getLastDataFromChainLink	Internal	-	-
getPrice	External	-	onlyAsset
_mockPrice	Internal	-	-

## 4.3 Vulnerability Summary

**[N1] [Low] Unchecked asset type in inputEth function allows potential asset mismatch**

**Category: Others**

**Content**

The input function lacks a crucial check to verify if the `_asset` parameter corresponds to an ETH pool. This oversight creates a potential vulnerability. In a scenario where both ETH and BTC pools exist, a user could potentially input

ETH but have it processed as BTC. This mismatch between the intended and actual asset type could lead to unexpected behavior and potential exploitation of the system.

- contracts/Doubler.sol

```
function inputEth(
    address _asset,
    uint256 _qAmount,
    address _to
) external payable nonReentrant onlyOncePerBlock onlyAsset(_asset) {
    if (msg.value != _qAmount) revert E_Balance();
    _input(_asset, _qAmount, _to, true);
}
```

### Solution

Verify that asset is eth.

### Status

Fixed

## [N2] [Low] Issue with fee allocation in the \_limitMint function

### Category: Others

### Content

In the `_limitMint` function, the distribution of minting fees should be based on the length of `_srvFeeAddr`.

Currently, the number of `_srvFeeAddr` entries called from the Doubler contract is only 2. However, if other contracts call this function with a different length of `_srvFeeAddr` in the future, it will result in allocating more fees than intended.

- contracts/RBToken.sol

```
function _limitMint(
    address _recipient,
    uint256 _tokenAmount,
    uint256 _poolTotalLimit,
    address[] memory _srvFeeAddr,
    uint16 _srvFeeRatio
) internal returns (uint256 recipientTokenAmount) {
    ...
}
```

```

    _totalShare = _totalShare + newShares;
    uint256 recipientNewShare = newShares;
    recipientTokenAmount = _tokenAmount;
    if (_srvFeeRatio > 0) {
        uint256 srvFee = (newShares * _srvFeeRatio) / _perMil;
        recipientTokenAmount = recipientTokenAmount - (_tokenAmount * _srvFeeRatio) /
        _perMil;
        for (uint8 i = 0; i < _srvFeeAddr.length; i++) {
            _shares[_srvFeeAddr[i]] = _shares[_srvFeeAddr[i]] + srvFee / 2;
            recipientNewShare = recipientNewShare - srvFee / 2;
            _emitTransferEvents(address(0x0), _recipient, (_tokenAmount *
            _srvFeeRatio) / _perMil, srvFee);
        }
    }
    _shares[_recipient] = _shares[_recipient] + recipientNewShare;
    _emitTransferEvents(address(0x0), _recipient, recipientTokenAmount,
    recipientNewShare);
}

```

## Solution

The handling fee is distributed equally according to the length of the actual incoming \_srvFeeAddr.

## Status

Fixed

## [N3] [Low] Risk of excessive authority

### Category: Authority Control Vulnerability Audit

## Content

owner can modify the fees and fee recipients in the pool, which can cause damage to the project's funds if the private key is compromised.

- contracts/Doubler.sol

```

owner can initializeDoubler
owner can updateLowerOfInputMaximum
owner can newPool
owner can updatePool

```

ADMIN can set the upper and lower price limits of the prediction machine, which will affect the functionality of the contract if ADMIN's private key is compromised.

- contracts/FastPriceFeed.sol

```
ADMIN can setAssetPriceLimit
ADMIN can setPriceFeedTimeLimit
ADMIN can newAsset
ADMIN can switchPriceFeed
```

### Solution

In the short term, transferring ownership of the core role like the Operation and Signer role 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. Like the authority involving user funds should be managed by the community.

### Status

Acknowledged; The owner of Doubler contracts and FastPriceFeed contracts have been managed using multi-signature contract.

FastPriceFeed Contract Address: 0x63083a606562c90098219CAa8D59732ad37DB4f7

FastPriceFeed Owner Multi-signature Contract: 0x035cc80577fDA35E4e47dB042315c030Df065Ad9

Doubler Contract Address: 0x56386f04111057a5D8DF8d719827038B716333F0

Doubler Owner Multi-signature Contract: 0x035cc80577fDA35E4e47dB042315c030Df065Ad9

## [N4] [Medium] Potential bypass Issue with onlyOncePerBlock

### Category: Others

### Content

In the design logic of doubler lite, atoken, btoken, and ctoken are all transferable. The onlyOncePerBlock mechanism is intended to restrict a user to a single function operation within one block. However, this restriction only applies to msg.sender, allowing users to bypass the limitation by making calls through multiple contracts.

- contracts/Doubler.sol

```
modifier onlyOncePerBlock() {
    if (_lastBlockCalled[msg.sender] >= block.number) revert E_BlockOnce();
    _;
```

```
_lastBlockCalled[msg.sender] = block.number;
}
```

### Solution

To address this, consider using tx.origin instead of msg.sender to limit the number of operations a user can perform.

### Status

Fixed

## [N5] [Low] Abnormal implementation logic in getPooledByShares

### Category: Others

### Content

In the getPooledByShares function, it should retrieve the corresponding token amount based on sharesAmount.

However, the actual interface called retrieves sharesAmount based on the token amount.

- contracts/RBToken.sol

```
function getPooledByShares(uint256 _sharesAmount) public view returns (uint256) {
    return _getSharesByPooledToken(_sharesAmount);
}
```

### Solution

Use \_getPooledTokenByShares instead of \_getSharesByPooledToken.

### Status

Fixed

## [N6] [Information] Potential risk due to the unique nature of 10xBToken

### Category: Others

### Content

Since 10xBToken is a special type of token where users hold shares instead of actual quantities, there may be an extreme risk when users add liquidity providers. In an extreme scenario, subsequent investors can obtain a large number of shares, causing the price of BToken in the pool to rise sharply (because the actual token quantity obtainable by the pool's shares decreases). This could result in liquidity providers incurring losses as a small amount of tokens might be used to exchange for a large amount of corresponding assets.

**Status**

Acknowledged

**[N7] [Information] Impact of inflationary or deflationary tokens on the doubler lite economic model****Category: Others****Content**

In the economic model of Doubler Lite, the use of inflationary (e.g., stETH) or deflationary tokens does not affect the overall economic model. This is because all calculations are based on shares, and the inflation or deflation impacts only the temporary average price of the tokens, which aligns with the design expectations.

**Status**

Acknowledged

## 5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002407030001	SlowMist Security Team	2024.06.28 - 2024.07.03	Low 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 1 medium risk, 4 low risk 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](http://www.slowmist.com)



**E-mail**  
[team@slowmist.com](mailto:team@slowmist.com)



**Twitter**  
[@SlowMist\\_Team](https://twitter.com/SlowMist_Team)



**Github**  
<https://github.com/slowmist>