Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

On 2024.01.08, the SlowMist security team received the Doubler team's security audit application for Doubler, 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

Doubler is a defi protocol for investment type.

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Preemptive Initialization	Race Conditions Vulnerability	Suggestion	Fixed
N2	Risk of excessive authority	Authority Control Vulnerability Audit	Medium	Acknowledged

NO	Title	Category	Level	Status
N3	Pool depth and TWAP interval in uniswap V3 price queries	Others	Suggestion	Acknowledged
N4	Recommendations on the conditions of winner	Others	Suggestion	Acknowledged
N5	Suggestions for setTwapInterval	Others	Low	Fixed
N6	Redundant code	Others	Suggestion	Fixed
N7	Function permission control issues	Authority Control Vulnerability Audit	Low	Fixed
N8	Overlooking purchase price relative to target profit in pool ending logic	Others	High	Fixed

4 Code Overview

4.1 Contracts Description

https://github.com/doublerpro/doubler-view

46c416a8c787eafdb9039a255e0b20fea464aa58

Audit scope:

- contracts/Doubler.sol
- contracts/FastPriceFeed.sol
- contracts/FRNFT.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:

Doubler			
Function Name	Visibility	Mutability	Modifiers
initialize	External	Can Modify State	-
updateAssetConfig	External	Can Modify State	nonReentrant onlyRole
upgradeReceiver	External	Can Modify State	nonReentrant onlyRole
newPool	External	Can Modify State	nonReentrant
input	External	Can Modify State	nonReentrant
_getLastLayer	Internal	Can Modify State	-
getMaxMultiple	External	-	-
_getMaxMultiple	Internal	-	-
_input	Internal	Can Modify State	-
_addTvl	Internal	Can Modify State	-
_getPrice	Internal	-	-
_getProfitPrice	Internal	-	-
forceWithdraw	External	Can Modify State	nonReentrant
_subTvl	Internal	Can Modify State	-
endPool	External	Can Modify State	nonReentrant
getWinnerOffset	External	-	-
_getWinnerOffset	Internal	-	-
_updateEndPool	Internal	Can Modify State	-
_getPoolProfitAmount	Internal	-	-
_countWithdrawAmount	Internal	-	-

Doubler			
_getWithdrawAmount	Internal	-	-
getTokenProfit	External	-	-
_getTokenProfit	Internal	-	-
gains	External	Can Modify State	nonReentrant
_gain	Internal	Can Modify State	-
getWinnerRange	External	-	-
_getWinnerRange	Internal	-	-
_getHitSpan	Internal	-	-
_countWinner	Internal	-	-
getPool	Public	-	-
getLayerCount	External	-	-
getLayerData	External	-	-
getPrivateVar	Public	-	-

FRNFT			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	ERC721
initialize	External	Can Modify State	-
getTraits	External	-	-
getTraitsList	External	-	-
isTokenOwner	External	-	-
exists	External	-	-
mint	External	Can Modify State	onlyRole

FRNFT			
burnFrom	External	Can Modify State	onlyRole
transferFrom	Public	Can Modify State	-
safeTransferFrom	Public	Can Modify State	-
getAttributes	Internal	-	-
getNormalAttribute	Internal	-	-
getNumberAttribute	Internal	-	-
tokenURI	Public	-	-
setDefaultRoyaltyInfo	External	Can Modify State	onlyRole
deleteDefaultRoyalty	External	Can Modify State	onlyRole
setTokenRoyalty	External	Can Modify State	onlyRole
resetTokenRoyalty	External	Can Modify State	onlyRole
_burn	Internal	Can Modify State	-
supportsInterface	Public	-	-
tokenOfOwnerByIndex	Public	-	-
getTokenList	Public	-	-
totalSupply	Public	-	-
tokenByIndex	Public	-	-
_beforeTokenTransfer	Internal	Can Modify State	-
_addTokenToOwnerEnumeration	Private	Can Modify State	-
_addTokenToAllTokensEnumeration	Private	Can Modify State	-
_removeTokenFromOwnerEnumeration	Private	Can Modify State	-
_removeTokenFromAllTokensEnumeration	Private	Can Modify State	-

FastPriceFeed			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
closeAsset	External	Can Modify State	isSupportedToken onlyRole
batchSetAssetPriceLimit	External	Can Modify State	onlyRole
newAsset	External	Can Modify State	onlyRole
updatePriceAggregator	External	Can Modify State	isSupportedToken onlyRole
upgradePlan	External	Can Modify State	isSupportedToken onlyRole
initChainlinkPriceFeed	Internal	Can Modify State	-
initDexPriceFeed	Internal	Can Modify State	-
checkChainlinkAggregatorVallid	Internal	-	-
setTwapInterval	External	Can Modify State	onlyRole
getPriceFromDex	Internal	-	-
mockDexPrice	Internal	-	-
getLastedDataFromChainlink	Internal	-	-
getPrice	External	-	isSupportedToken
mockPrice	Internal	-	-
isSupported	External	-	-

4.3 Vulnerability Summary

[N1] [Suggestion] Preemptive Initialization

Category: Race Conditions Vulnerability

Content

- contracts/FRNFT.sol

This function has the problem of being preempted.

```
function initialize(address doublerPool) external {
    require(_initialized == false, "initialized err");
    _initialized = true;
    _doublerPool = doublerPool;
    _grantRole(DOUBLER_ROLE, doublerPool);
}
```

- contracts/Doubler.sol

This function has the problem of being preempted.

```
    function initialize(
        address _initTeam,
        address _initEco,
        address _initFastPriceFeed,
        address _initDoublerNFT,
        address _initDbrTokenAddress,
        address _initMultiSigWallet,
        uint16 _initProtectBlock
    ) external {
        if (_initialized == true) revert E_Initialized();
        _initialized = true;
        _team = _initTeam;
        _eco = _initEco;
        _fastPriceFeed = _initFastPriceFeed;
        _FRNFT = _initDoublerNFT;
        _ecoFeeRatio = 2000; // 20% * 100
        _feeRatio = 20; // 0.2% * 100
        _protectBlock = _initProtectBlock;
        _grantRole(DEFAULT_ADMIN_ROLE, _initMultiSigWallet);
        emit Initialize(_initTeam, _initFastPriceFeed, _initDoublerNFT,
_initDbrTokenAddress, _initMultiSigWallet);
    }
```

Solution

It is suggested that the initialize operation can be called in the same transaction immediately after the contract is created to avoid being maliciously called by the attacker.

Status

Fixed

[N2] [Medium] Risk of excessive authority

Category: Authority Control Vulnerability Audit

Content

- contracts/FastPriceFeed.sol

The admin role can be set to price related, if the private key leakage will cause the price anomaly caused by the pool function is impaired.

```
admin can newAsset
admin can updatePriceAggregator
admin can upgradePlan
admin can setTwapInterval
```

Solution

In the short term, in order to cope with the scenario that the protocol needs to frequently set parameters in the early stage, the Admin can be divided into two roles, one is an EOA address, which is used to manage the protocol's emergency pause permission, and the other is a multisign address, which is used to manage necessary parameter configuration and modification. This can solve the single-point risk without losing too much flexibility, but it cannot effectively mitigate the risk of excessive privileges. In the long run, it is more reasonable to entrust the protocol's parameter configuration and modification permissions to the timelock contract, and to entrust the timelock contract to community governance can effectively mitigate the risk of excessive privileges. This can also improve the trust of community users in the protocol.

Status

Acknowledged; Followed by the use of multiple signatures to manage admin.

[N3] [Suggestion] Pool depth and TWAP interval in uniswap V3 price queries

Category: Others

Content

- contracts/FastPriceFeed.sol

When getting the price of the token, you should pay attention to the depth of the corresponding pool, if the depth of the pool is too shallow and the price range is set too low, there is still a possibility of price manipulation.

```
function getPriceFromDex(address _asset) internal view returns (uint256 price) {
    require(_isSupported[_asset], 'UniV3: oracle in mainnet not initialized yet!');
    address uniswapV3Pool = _assetFeedMap[_asset];
    uint32 twapInterval = _twapIntervals[_asset];
    IUniswapV3Pool pool = IUniswapV3Pool(uniswapV3Pool);
    IUniswapV3Pool.Slot0 memory slot0;
    IUniswapV3Pool.Observation memory obs;
    slot0 = pool.slot0();
    obs = pool.observations((slot0.observationIndex + 1) %
slot0.observationCardinality);
    require(obs.initialized, "UNIV3: Pair did't initialized");
    uint32 delta = uint32(block.timestamp) - obs.blockTimestamp;
    require(delta >= twapInterval, 'UniV3: token pool does not have enough transaction
history in mainnet');
    uint32[] memory secondsAgos = new uint32[](2);
    secondsAgos[0] = twapInterval;
    secondsAgos[1] = 0;
    (int56[] memory tickCumulatives, ) = pool.observe(secondsAgos);
    uint160 sqrtPriceX96 = TickMath.getSqrtRatioAtTick(
        int24((tickCumulatives[1] - tickCumulatives[0]) /
int56(uint56(twapInterval)))
    );
    (uint256 price0, uint256 price1) = mockDexPrice(pool, sqrtPriceX96);
    return pool.token0() == _asset ? price0 : price1;
}
```

Solution

Check the depth of the pool and set the proper time intervals.

Status

Acknowledged

[N4] [Suggestion] Recommendations on the conditions of winner

Category: Others

Content

- contracts/Doubler.sol

In some extreme cases, it may be possible to control the final winner.

Solution

Can set how many blocks a user must purchase before becoming a winner.

Status

Acknowledged

[N5] [Low] Suggestions for setTwapInterval

Category: Others

Content

- contracts/FastPriceFeed.sol

It should be a supported token to set the interval.

```
function setTwapInterval(address _asset, uint32 _twapInterval) external
onlyRole(DEFAULT_ADMIN_ROLE) {
    require(!_isSupported[_asset], 'Oracle: do not support this token');
    require(_plans[_asset] == Plan.DEX, "setTwapInterval: Only dex _asset");
    require(
        MAX_INTERVA >= _twapIntervals[_asset] && _twapIntervals[_asset] >=
MIN_INTERVA,
        'setTwapInterval: Invalid twapInterval'
    );
    emit SetTwapInterval(_asset, _twapIntervals[_asset], _twapInterval);
    _twapIntervals[_asset] = _twapInterval;
}
```

Solution

Change to a supported token to set it.

Status

Fixed

[N6] [Suggestion] Redundant code

Category: Others

Content

- contracts/Doubler.sol

The hot in the structure is not used.

```
struct Pool {
        address asset;
        address creator;
        address terminator;
        uint16 fallRatio;
        uint16 profitRatio;
        uint16 rewardRatio;
        uint16 winnerRatio;
        uint32 double;
        uint32 lastLayer;
        uint256 tokenId;
        uint256 unitSize;
        uint256 maxRewardUnits;
        uint256 winnerOffset;
        uint256 endPrice;
        uint256 hot; //SLOWMIST// unused
        uint256 lastOpenPrice;
        uint256 tvl;
        uint256 amount;
        uint256 margin;
        uint256 joins;
        uint256 lastInputBlockNo;
        uint256 kTotal;
    }
```

Solution

If the code is redundant delete it.

Status

Fixed

[N7] [Low] Function permission control issues

Category: Authority Control Vulnerability Audit

Content

- contracts/FRNFT.sol

The roles used by these functions are not set and cannot be executed subsequently.

```
setTokenRoyalty
resetTokenRoyalty
```

```
setDefaultRoyaltyInfo
deleteDefaultRoyalty
```

Solution

Setting up corresponding roles for control.

Status

Fixed

[N8] [High] Overlooking purchase price relative to target profit in pool ending logic

Category: Others

Content

- contracts/Doubler.sol

The scenario where the purchase price is less than or equal to the target profit price has not been considered. When the price at the time of purchase is less than or equal to the target profit price, the purchase should not be allowed. Otherwise, users can manipulate the ending of the pool after purchasing to position themselves as the winner.

```
function _input(AddInput memory _addInput, uint8 _decimals) internal returns (uint256
tokenId) {
        Pool memory pool = _poolMap[_addInput.poolId];
        if (_addInput.margin == 0 || _addInput.margin > _addInput.amount ||
_addInput.margin.mod(pool.unitSize) != 0)
            revert E_Margin();
        if (IERC20(pool.asset).allowance(_msgSender(), address(this)) <
_addInput.margin) revert E_Approve();
        if (IERC20(pool.asset).balanceOf(_msgSender()) < _addInput.margin) revert
E_Balance();
        if (_addInput.multiple < 1 || _addInput.margin.mul(_addInput.multiple) !=
_addInput.amount) revert E_Multiple();
        if (_addInput.multiple > 1 && _addInput.multiple > _getMaxMultiple(pool,
_addInput.curPrice, _decimals))
            revert E_MultipleLimit();
        _addInput.layer = _getLastLayer(_addInput.poolId, _addInput.curPrice,
_addInput.amount);
        LayerData memory layer = _layerDataMap[_addInput.poolId][_addInput.layer];
        if (layer.amount >= layer.cap) revert E_LayerCap();
        if (layer.cap.sub(layer.amount) < _addInput.margin) {
            _addInput.margin = _addInput.amount = layer.cap.sub(layer.amount);
        } else {
            _addInput.amount = layer.cap.sub(layer.amount) < _addInput.amount
```

```
                ? layer.cap.sub(layer.amount)
                : _addInput.amount;
        }
        IERC20(pool.asset).safeTransferFrom(_msgSender(), address(this),
_addInput.margin);
        uint256 layerAmount = _addTvl(_addInput);
        uint256 layerRanking = layerAmount.div(pool.unitSize);
        tokenId = IFRNFT(_FRNFT).mint(
            _msgSender(),
            _addInput.poolId,
            _addInput.layer,
            _addInput.margin,
            _addInput.amount,
            _addInput.curPrice,
            layerRanking
        );
        emit NewInput(
            tokenId,
            _addInput.poolId,
            _msgSender(),
            _addInput.layer,
            _addInput.margin,
            _addInput.amount,
            _addInput.curPrice
        );
    }
```

Solution

Purchases below the profitprice are not allowed.

Status

Fixed

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002401120001	SlowMist Security Team	2024.01.08 - 2024.01.12	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 1 high risk, 1 medium risk, 2 low risk, 4 suggestion 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