SLOWMIST

Smart Contract
Security Audit Report

Table Of Contents

1 Executive Summary

On 2025.11.24, the SlowMist security team received the team's security audit application for Monster(MON), 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

MonsterToken is an enhanced ERC20 token contract that integrates DEX trading functionality. The project implements a trading control mechanism, including: a trading tax system, a whitelist mechanism, trading cooldown times, trading size limits, and integration with the PancakeSwap DEX. The contract is mintable and destroyable, and access control is implemented through role-based access management.

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Swap Imbalance and Profit via "Target = Pair" Misconfiguration	Design Logic Audit	Medium	Fixed
N2	Risk of excessive privilege	Authority Control Vulnerability Audit	Medium	Fixed
N3	Denial of Service via Unbounded Tax Settings	Denial of Service Vulnerability	Low	Acknowledged
N4	Burn Function Reverts if Pair Not Set	Design Logic Audit	Low	Acknowledged
N5	Missing Parameter Validation	Others	Low	Acknowledged
N6	Lifetime Mint Cap Logic Issue	Design Logic Audit	Suggestion	Acknowledged
N7	Strict Sell Limits in Low Liquidity State	Others	Information	Acknowledged

4 Code Overview

4.1 Contracts Description

Audit Version:

https://github.com/monmondev/monster-token-contract/blob/audit/MonsterToken.sol

commit: 5d4ebcb49c11e29a20273f9185c3e1087ad44a22

The main network address of the contract is as follows:

https://bscscan.com/token/0x7d2ba25b1c208c8489d0a99b72a1060a8fef834e

4.2 Visibility Description

The SlowMist Security team analyzed the visibility of major contracts during the audit, the result as follows:

MonsterToken			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	ERC20
setPair	Public	Can Modify State	onlyRole
toggleBuyWhitelistedUser	Public	Can Modify State	onlyRole
toggleColdTimeWhitelistedUser	Public	Can Modify State	onlyRole
toggleTradeSizeCapWhitelistedUser	Public	Can Modify State	onlyRole
toggleTaxExemption	Public	Can Modify State	onlyRole
addTaxSetting	Public	Can Modify State	onlyRole
updateTaxSetting	Public	Can Modify State	onlyRole
setEnableBuyTimestamp	Public	Can Modify State	onlyRole
setBuySizeCapBps	Public	Can Modify State	onlyRole
setSellSizeCapBps	Public	Can Modify State	onlyRole
setColdTime	Public	Can Modify State	onlyRole
mintInitialSupply	Public	Can Modify State	onlyRole
mint	Public	Can Modify State	onlyRole
burn	External	Can Modify State	-
_update	Internal	Can Modify State	-
_getReservesOrdered	Internal	-	-

4.3 Vulnerability Summary

[N1] [Medium] Swap Imbalance and Profit via "Target = Pair" Misconfiguration

Category: Design Logic Audit

Content

In the MonsterToken contract, the taxSettings allow the DEFAULT_ADMIN_ROLE to set any address as the tax

recipient (target). A issue arises if the target is configured as the PancakeSwap Pair address. When a user sells

tokens (swaps for USDT), the contract splits the transfer into two parts:

Tax Transfer: taxAmount is sent to the target.

Swap Transfer: value - taxAmount is sent to the pair.

If `target == pair`, both transfers go to the Pair contract. The Pair contract sums up all received tokens to

determine the swap input amount. Consequently, the Pair sees the total input as `taxAmount + (value -`

`taxAmount) = value`.

Impact:

Tax Evasion: The user pays zero effective tax.

Pool Imbalance & LP Dilution: The pool receives tokens that were intended to be removed from circulation (or sent to

a treasury). Instead of the user "paying" the tax, the Liquidity Providers (LPs) are forced to "buy" these tax tokens

from the user with USDT from the pool.

Value Extraction: This extracts more USDT from the pool than intended. Over time, this systematically drains the

pool's USDT reserves and leaves LPs holding more MonsterTokens than they should, effectively devaluing their LP

positions.

Code location:

monster-token-contract/MonsterToken.sol#L123-L142, L195-L251

```solidity
    function addTaxSetting(
        ...
    ) public onlyRole(DEFAULT_ADMIN_ROLE) {
        taxSettings.push(TaxSetting({target: target, bps: bps}));
        ...
    }

    function updateTaxSetting(
        ...
    ) public onlyRole(DEFAULT_ADMIN_ROLE) {
        require(index < taxSettings.length, "invalid index");
        taxSettings[index].target = target;
        taxSettings[index].bps = bps;
        ...
    }
```

```
    function _update(
        ...
    ) internal override(ERC20) {
        uint256 totalTaxAmount = 0;
        // selling or add LP, tax either way
        if (to == pair) {
            if (!_taxExemptions.contains(from)) {
                for (uint256 i = 0; i < taxSettings.length; i++) {
                    uint256 taxAmount = (value * taxSettings[i].bps) / 10000;
                    super._update(from, taxSettings[i].target, taxAmount);
                    totalTaxAmount += taxAmount;
                }
            }
            ...
        }
        ...
        super._update(from, to, value - totalTaxAmount);
    }
```

Solution

It's recommended to add a validation check in addTaxSetting and updateTaxSetting to prevent setting the target to

the pair address.

Status

Fixed; After communicating with the project team, they stated that their deploy scripts will ensure this won't happen.

Furthermore, after deployment confirmation, only three contracts exist in the contract's TaxSetting: two are multi-

signature addresses and one is a Rancoin address; none are set as a pair address. Also, the contract is renounced

DEFAULT_ADMIN_ROLE, meaning TaxSetting can no longer be changed using the addTaxSetting or

updateTaxSetting functions.

[N2] [Medium] Risk of excessive privilege

Category: Authority Control Vulnerability Audit

Content

The DEFAULT_ADMIN_ROLE possesses extensive privileges that can be abused to manipulate the contract state,

potentially leading to fund loss or denial of service for users.

1.The admin can use addTaxSetting and updateTaxSetting to set the total tax rate to 100% (10000 bps). While the

contract prevents exceeding 100%, setting it to exactly 100% means users receive 0 tokens/USDT from trades,

effectively confiscating their funds.

2.The admin can use setPair to set the pair address to any arbitrary address (including malicious contracts or address(0)), enabling the "Target = Pair" attack or breaking the burn function.

3.The admin can use setColdTime to set an arbitrarily long duration, effectively freezing user assets by preventing them from selling or buying for extended periods.

4.The admin can use setBuySizeCapBps and setSellSizeCapBps to set the caps to 0. This would cause the require(value <= 0) check to fail for any non-zero trade, completely halting trading.

Code location:

monster-token-contract/MonsterToken.sol#L74-L77, L123-L170

```
    function setPair
    function addTaxSetting
    function updateTaxSetting
    function setEnableBuyTimestamp
    function setBuySizeCapBps
    function setSellSizeCapBps
    function setColdTime
```

Solution

In the short term, assigning privileged roles to a multisig wallet can effectively mitigate the single point of failure risk, while still allowing configuration flexibility during the early stages of the project. In the long term, once the project operates stably, transferring these privileged roles to DAO governance can effectively address the risk of excessive centralization of authority. During the transition period, using a multisig setup in combination with a timelock mechanism for delayed transaction execution can further mitigate the risks associated with over-privileged control.

Status

Fixed; After communicating with the project team, they stated that they will renounce the admin role in the near future.

Multiple roles in the contract renounced their rights in the following transaction, and the DEFAULT_ADMIN_ROLE address was fully renounced.

https://bscscan.com/tx/0xf9e63cf3426cf247f5e3cd3be964fc597635d0ec16524ec016b518320a66a6ed

[N3] [Low] Denial of Service via Unbounded Tax Settings

Category: Denial of Service Vulnerability

Content

The taxSettings array is unbounded, and the addTaxSetting function allows the admin to push an unlimited number of tax settings. In the _update function, the contract iterates over the entire taxSettings array for every sell transaction. If the array becomes too large, the gas cost of the loop (combined with multiple external calls/events in super._update) will exceed the block gas limit, causing all sell transactions to revert.

Code location:

monster-token-contract/MonsterToken.sol#L195-L251

```solidity
function _update(
    address from,
    address to,
    uint256 value
) internal override(ERC20) {
    uint256 totalTaxAmount = 0;
    // selling or add LP, tax either way
    if (to == pair) {
        if (!_taxExemptions.contains(from)) {
            for (uint256 i = 0; i < taxSettings.length; i++) {
                uint256 taxAmount = (value * taxSettings[i].bps) / 10000;
                super._update(from, taxSettings[i].target, taxAmount);
                totalTaxAmount += taxAmount;
            }
        }
        ...
    }
    ...
}
```

Solution

It's recommended to limit the maximum number of tax settings in addTaxSetting.

Status

Acknowledged

[N4] [Low] Burn Function Reverts if Pair Not Set

Category: Design Logic Audit

Content

The burn function calls _burn, which internally calls _update. In _update, there is a check `if (to == pair)`. By default, pair is address(0). When burning, to is address(0), so the condition address(0) == address(0) evaluates to true. The code then attempts to call _getReservesOrdered() which calls IPancakePair(pair).getReserves(). Calling this on address(0) causes a revert. Consequently, users cannot burn tokens until the admin calls setPair with a non-zero address.

Code location:

monster-token-contract/MonsterToken.sol#L195-L251

```solidity
    function _update(
        address from,
        address to,
        uint256 value
    ) internal override(ERC20) {
        uint256 totalTaxAmount = 0;
        // selling or add LP, tax either way
        if (to == pair) {
            ...
        }
        ...
    }
```

Solution

It's recommended to modify the condition in _update to ensure the pair is set before executing pair-specific logic. As `if (pair != address(0) && to == pair)`.

Status

Acknowledged; After communicating with the project team, they stated that they will make sure this won't happen in their deploy script.

[N5] [Low] Missing Parameter Validation

Category: Others

Content

Multiple functions in the contract lack appropriate validation for input parameters, such as checks for zero addresses, zero values, or reasonable minimum/maximum bounds. This can lead to misconfiguration or unexpected behavior.

Affected Functions:

Constructor: No checks for _maxSupply > 0, _initialSupply > 0, _buySizeCapBps > 0, etc.

setPair: No check for _pair != address(0).

toggle...WhitelistedUser / toggleTaxExemption: No check for account != address(0).

addTaxSetting / updateTaxSetting: No check for target != address(0) (or target != pair as mentioned in N1).

setEnableBuyTimestamp: No check if timestamp is in the past or too far in the future.

setBuySizeCapBps / setSellSizeCapBps: No check for _sizeCapBps > 0 (preventing DoS) or _sizeCapBps <= 10000.

setColdTime: No check for reasonable maximum cold time (preventing DoS).

Code location:

monster-token-contract/MonsterToken.sol#L54-L170

```
constructor(...) ERC20(...) {...}
function setPair
function toggleBuyWhitelistedUser
function toggleColdTimeWhitelistedUser
function toggleTradeSizeCapWhitelistedUser
function toggleTaxExemption
function addTaxSetting
function updateTaxSetting
function setEnableBuyTimestamp
function setBuySizeCapBps
function setSellSizeCapBps
function setColdTime
```

Solution

It's recommended to add require statements to validate all input parameters.

Like:

```
require(_pair != address(0), "invalid pair"); require(_sizeCapBps > 0 && _sizeCapBps <=
10000, "invalid bps"); require(_coldTime <= MAX_COLD_TIME, "cold time too long"); etc.
```

Status

Acknowledged; After communicating with the project team, they stated that they will make sure this won't happened, and if did, they will correct the configs.

[N6] [Suggestion] Lifetime Mint Cap Logic Issue

Category: Design Logic Audit

Content

The MonsterToken contract tracks `totalMintedAmount` which strictly increases upon minting and never decreases upon burning. The mint function checks `totalMintedAmount + amount <= maxSupply`. This means maxSupply acts as a lifetime cap on the total number of tokens ever minted, rather than a cap on the current circulating supply. Once the cumulative minted amount reaches maxSupply, no more tokens can be minted, even if the current totalSupply is zero (due to burning).

Code location:

MonsterToken.sol#L180-L193

```
    function mint(
        address to,
        uint256 amount
    ) public onlyRole(WHITELISTED_CONTRACT_ROLE) {
        // have to track total supply on our own since when `to` is zero address, the
  built-in total supply will be decreased,
        // which is not what we want
        require(totalMintedAmount + amount <= maxSupply, "max supply exceeded");
        _mint(to, amount);
        totalMintedAmount += amount;
    }

    function burn(uint256 amount) external {
        _burn(msg.sender, amount);
    }
```

Solution

If the intention is to cap the current supply, use totalSupply() + amount <= maxSupply. If the intention is a lifetime cap, no change is needed, but this limitation should be clearly documented.

Status

Acknowledged; After communicating with the project team, they stated that this is by design.

[N7] [Information] Strict Sell Limits in Low Liquidity State

Category: Others

Content

The sellSizeCap is calculated as a percentage of the current pool reserves (reserveToken * sellSizeCapBps / 10000).

In the early stages when liquidity is low, this calculated cap can be extremely small (e.g., smaller than a typical user's balance), making it impossible for users to sell reasonable amounts of tokens.

Code location:

monster-token-contract/MonsterToken.sol#L222

```
uint256 sellSizeCap = (reserveToken * sellSizeCapBps) / 10000;
```

Solution

It's recommended to implement a minimum floor for the sell limit to ensure basic usability even with low liquidity.

Status

Acknowledged; After communicating with the project team, they stated that this is intended behavior by design.

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002511260001	SlowMist Security Team	2025.11.24 - 2025.11.26	Low Risk

Summary conclusion: The SlowMist security team uses a manual and the SlowMist team's analysis tool to audit the project. During the audit work, we identified 2 medium risks, 3 low risks, 1 suggestion, and 1 information. All the findings were acknowledged. After deployment, multiple roles in the contract renounced their rights in the following transaction, and the DEFAULT_ADMIN_ROLE address was fully renounced. The project is now fully decentralized.

Tx hash:

https://bscscan.com/tx/0xf9e63cf3426cf247f5e3cd3be964fc597635d0ec16524ec016b518320a66a6ed

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