MonkyToken

Smart Contract Audit Report

DATE: 26th NOVEMBER 2024 | **PREPARED BY:** Entersoft Pty Ltd

Contents

Revision History & Version Control

Version	Date	Author(s)	Description
1.0	26-Nov-2024	Gurkirat singh	Final Report

Entersoft was commissioned by MonkyToken to perform a smart contract code review on their product. The review was conducted from **25 November 2024 to 26 November 2024**, with the aim of ensuring overall code quality, security, correctness, and to ensure that the code will work as intended.

The report is structured into two main sections:

- Executive Summary: which provides a high-level overview of the audit findings.
- Technical Analysis: which offers a detailed analysis of the smart contract code.
- Tested for Scenarios: this section outlines the test cases that have been executed for the smart contract.

Please note that the analysis is static and entirely limited to the smart contract code. The information provided in this report should be used to understand the security and quality of the code, as well as its expected behavior.

Scope included:

- MonkyToken.sol

Standards followed include:

- ARC-69 for NFT tokens
- OWASP (partially, for instance role validations, input validations etc.)
- NIST SP 800 (for encryptions, signatures)
- Smart Contract Security Verification Standard (SCSVS)

1. Disclaimer

This is a limited audit report on our findings based on our analysis, in accordance with good industry practice as at the date of this report, in relation to: (i) smart contract best coding practices and issues in the framework and algorithms based on white paper, code, the details of which are set out in this report, (Smart Contract audit). To get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us based on what it says or does not say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on the disclaimer below – please make sure to read it in full.

By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy any copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and Entersoft Australia and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers, and other representatives) (Entersoft) owe no duty of care towards you or any other person, nor does Entersoft make any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties, or other terms of any kind except as set out in this disclaimer, and Entersoft hereby excludes all representations, warranties, conditions, and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose, and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, Entersoft hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against Entersoft, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of the use of this report, and any reliance on this report. The analysis of the Smart contract is purely based on the smart contract code shared with us alone.

2. Executive Summary

Entersoft has conducted a static and dynamic smart contract audit of the MonkyToken project through a comprehensive smart contract audit. The primary objective was to identify potential vulnerabilities and risks within the codebase, ensuring adherence to industry-leading standards while prioritizing security, reliability, and performance. Our focus was on prompt and efficient identification and resolution of vulnerabilities to enhance the overall robustness of the solidity smart contract.

Testing Methodology:

Our testing methodology in Solidity adhered to industry standards and best practices, integrating partially implemented OWASP and NIST SP 800 standards for encryption and signatures.

We have performed a detailed manual analysis, adherence to industry standards, and the use of a comprehensive toolset. Our approach ensured a thorough evaluation within the designated Solidity code files.

Findings and Security Posture:

Our primary focus was on Access Control Policies, Transaction Signature Validations, Reentrancy, Time Manipulation, Default Visibility, Outdated Compiler Version, Input Validation, Deprecated Solidity Functions, Shadowing State Variables, Presence of Unused Variables, Overflow and Underflow Conditions, Assets Integrity, Errors and Exception.

Importantly, our audit process intentionally avoided reliance solely on automated tools, emphasizing a more in-depth and nuanced approach to security analysis. Conducted from November 25, 2024 to November 26, 2024 , our team diligently assessed and validated the security posture of the solidity smart contract, ultimately classifying it as "Secure," reflecting the absence of identified vulnerabilities and the robustness of the codebase against potential threats.

Assumptions Due to Missing Documentation:

As the relevant documentation was unavailable during the audit, the following assumptions were made regarding the token transfer behavior during the initial launch phase:

1. Only the owner has the ability to transfer tokens to whitelisted addresses.
2. Whitelisted addresses are permitted to transfer tokens exclusively to other whitelisted addresses.
3. Non-whitelisted and blacklisted addresses are restricted from transferring tokens to any address, including their own wallet.

3. Overview

2.1 Project Overview

Entersoft has meticulously audited the smart contract project from November 25, 2024 to November 26, 2024 with a primary focus on Solidity code files integral to blockchain functionality, emphasizing vulnerabilities in associated gas claiming. The working of basic functionalities was also tested during the review.

2.2. Scope

The audit scope covers the MonkyToken smart contract available in the GitHub repository:

Commit ID	c61d0663649849937875b7bdb2a914f86c0c7f19
Scope	MonkyToken.sol

OUT-OF-SCOPE: External contracts, other imported smart contracts.

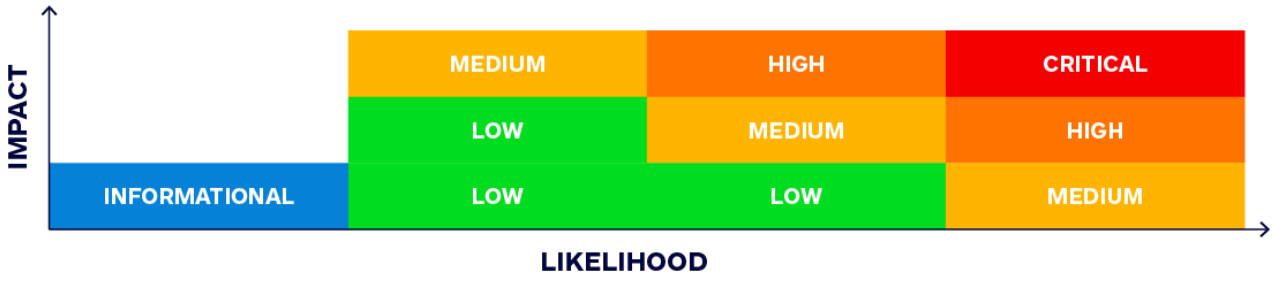
2.3 Summary of Findings

The following table is the summary of findings, which summarizes the overall risks identified during the audit.

Critical	High	Medium	Low	Informational
0	1	3	0	1

2.4 Security Level References

Every vulnerability in this report was assigned a severity level from the following classification table:



4. Comprehensive Analysis findings: A Dual Approach through Static and Manual Examination

Phase 1: Static Analysis

In our rigorous smart contract audit process, we employed a multifaceted approach combining both static and dynamic analysis methodologies to comprehensively evaluate the security posture of the protocol. Our static analysis phase involved the utilization of static analyzing tools such as Slither, Aderyn. These tools enabled us to conduct automated code analysis to identify potential vulnerabilities within the smart contracts.

Tools and Efforts:

Slither, Aderyn were instrumental in performing static analysis, allowing us to efficiently scan the codebase for common vulnerabilities such as reentrancy, denial-of-service (DOS) attacks, front-running vulnerabilities, time dependencies, token approval issues, and arithmetic errors. Through meticulous examination of the code, we meticulously identified and categorized potential risks, laying the groundwork for further in-depth analysis.

Phase 2: Dynamic Analysis

After the static analysis phase, we transitioned to dynamic analysis, which involved a more hands-on approach to scrutinizing the intended functionality and security of the smart contracts. Further, we devised a comprehensive suite of unit tests to validate the expected behavior of the smart contracts under various scenarios.

Processes and Test Cases:

Our dynamic analysis encomPASS a systematic exploration of the smart contracts' functionalities, focusing on critical areas prone to vulnerabilities. We meticulously crafted test cases to assess the resilience of the contracts against potential attack vectors, including but not limited to reentrancy attacks, DOS vulnerabilities, front-running exploits, time-sensitive vulnerabilities, token approval vulnerabilities, and arithmetic errors.

Throughout both the static and dynamic analysis phases, our team dedicated substantial efforts to meticulously review the codebase, identify vulnerabilities, and develop robust test cases to assess the security posture of the protocol comprehensively. By combining automated analysis with manual examination and testing, we ensured a thorough evaluation of the smart contracts, ultimately enhancing the security and reliability of the protocol.

5. Technical Analysis

Note:
The following values for “Severity” mean:

- Critical: This vulnerability poses a direct and severe threat to the funds or the main functionality of the protocol.
- High: Direct impact on the funds or the main functionality of the protocol.
- Medium: Indirect impact on the funds or the protocol's functionality.
- Low: Minimal to no impact on the funds or the protocol's main functionality.

The following values for “Result” mean:

- PASS: indicates that there is no security risk.
- FAIL : indicates that there is a security risk that needs to be remediated.
- Informational : Suggestions related to good coding practices and gas-efficient code.
- **Not Applicable**: means the attack vector is Not applicable or Not available

5.1 Transfer allowed by blacklisted users when trading is disabled

Result	**PASS**
Severity	High
Description	The _update function does not enforce the restriction on blacklisted (bannedAddresses) accounts. Blacklisted users are still able to initiate and receive token transfers, which defeats the purpose of the banning mechanism.
Location / Source File	MonkyToken.sol 51
Observation	This flaw allows blacklisted accounts to: • Continue participating in token transfers, undermining protocol restrictions. • Potentially bypass other safeguards in the system, depending on the implementation. This compromises the security of the contract and allows banned accounts to disrupt the ecosystem.

```
function test_transferWithBlacklistAddressWithLaunchDisabled() public transferTokensToUser(200){
    // Arrange
    uint256 transferValue = 100;
    console.log("Initial Bal of user1",monkyToken.balanceOf(user1));
    console.log("Initial Bal of user2",monkyToken.balanceOf(user2));
    console.log("The user1 become blacklisted BEFORE transfer" ,monkyToken.bannedAddresses(user1));
    console.log("The user2 become blacklisted BEFORE transfer" ,monkyToken.bannedAddresses(user2));

    vm.startPrank(user1);
    monkyToken.approve(user2, transferValue);
    monkyToken.transfer(user2, transferValue);
    vm.stopPrank();

    assertEq(monkyToken.bannedAddresses(user1),true);
    assertEq(monkyToken.bannedAddresses(user2),true);
    console.log("The user1 become blacklisted AFTER first transfer" ,monkyToken.bannedAddresses(user1));
    console.log("The user2 become blacklisted AFTER first transfer" ,monkyToken.bannedAddresses(user2));
    console.log("Final Bal of user1 after first transfer",monkyToken.balanceOf(user1));
    console.log("Final Bal of user2 after first transfer",monkyToken.balanceOf(user2));

    // Act
    vm.startPrank(user1);
    monkyToken.approve(user2, transferValue);
    monkyToken.transfer(user2, transferValue);
    vm.stopPrank();

    // Assert
    assertEq(monkyToken.balanceOf(user1),0,"Final Bal of user 1");
    assertEq(monkyToken.balanceOf(user2),200,"Final Bal of user 2");

    console.log("User1 was still able to do a transfer to User2 with disbaled trading");
    console.log("The user1 become blacklisted AFTER second transfer" ,monkyToken.bannedAddresses(user1));
    console.log("The user2 become blacklisted AFTER second transfer" ,monkyToken.bannedAddresses(user2));
    console.log("Final Bal of user1 after second transfer",monkyToken.balanceOf(user1));
    console.log("Final Bal of user2 after second transfer",monkyToken.balanceOf(user2));
}
```

```
gurkiratsingh@Gurkirats-MacBook-Air monky-contract-main % forge test --mt test_transferWithBlacklistAddressWithLaunchDisabled -vv
[⠊] Compiling...
[⠊] Compiling 1 files with Solc 0.8.25
[⠊] Solc 0.8.25 finished in 2.75s
Compiler run successful!

Ran 1 test for test/Monky.t.sol:MonkyTokenTest
[PASS] test_transferWithBlacklistAddressWithLaunchDisabled() (gas: 194977)
Logs:
  Initial Bal of user1 200
  Initial Bal of user2 0
  The user1 become blacklisted BEFORE transfer false
  The user2 become blacklisted BEFORE transfer false
  The user1 become blacklisted AFTER first transfer true
  The user2 become blacklisted AFTER first transfer true
  Final Bal of user1 after first transfer 100
  Final Bal of user2 after first transfer 100
  User1 was still able to do a transfer to User2 with disbaled trading
  The user1 become blacklisted AFTER second transfer true
  The user2 become blacklisted AFTER second transfer true
  Final Bal of user1 after second transfer 0
  Final Bal of user2 after second transfer 200

Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 2.61s (189.65ms CPU time)

Ran 1 test suite in 2.62s (2.61s CPU time): 1 tests passed, 0 failed, 0 skipped (1 total tests)
```

Remediation	Incorporate a check in the `_update` function to revert any transaction involving blacklisted accounts when trading disabled as it is being implemented while trading is enabled. Which ensures that any transaction involving blacklisted users is blocked, maintaining the integrity of the banning mechanism.

5.2 Centralization Risk For Trusted Owners

Result	**FAIL**
Severity	High
Description	Contracts have owners with privileged rights to perform admin tasks and need to be trusted to not perform malicious updates or drain funds.
Location / Source File	MonkyToken.sol 26,35,40
Observation	The decentralization ideals of the protocol may be compromised by the centralization threats posed by the existence of trusted owners with broad powers. A compromised or malicious owner's malicious activities could result in money loss or detrimental protocol changes, undermining trust and discouraging participation in the ecosystem.
Remediation	Examine and modify functions that use the `onlyOwner` modification to reduce the powers granted to owners in order to mitigate the risk of centralization. In order to decentralize decision-making, implement multi-signature procedures for important administrative activities and investigate community governance models. The protocol can reduce the risks associated with centralization and improve resistance against possible hostile activities by dispersing control and implementing safeguards.
Reference	https://www.certik.com/resources/blog/What-is-centralization-risk
Comments	The protocol has confirmed their intent to integrate multi-signature wallet functionality in future updates to address this limitation.

5.3 Transfer Allowed When Trading Is Disabled And User Is Not WhiteListed

Result	**PASS**
Severity	Medium
Description	The _update function does not prevent token transfers when trading is disabled and the involved addresses are not whitelisted. While the function bans the sender (_from) and receiver (_to) after the transfer, it does not revert the transaction. This allows users to transfer tokens even when trading is intended to be restricted.
Location / Source File	MonkyToken.sol 51
Observation	Malicious users could exploit this vulnerability to bypass the intended trading restrictions. For instance: • Unwhitelisted users could transfer tokens to other accounts before being banned. • This compromises the launch control mechanism and the protocol's ability to enforce pre-trading conditions.

```
function test_transferWithLaunchDisabledWithNotWhiteListedUsers() public transferTokensToUser(100){
    // Arrange
    uint256 transferValue = 100;
    uint256 intialBalUser1 = monkyToken.balanceOf(user1);

    console.log("Initial Bal of user1",monkyToken.balanceOf(user1));
    console.log("Initial Bal of user2",monkyToken.balanceOf(user2));
    console.log("The user1 become blacklisted BEFORE transfer" ,monkyToken.bannedAddresses(user1));
    console.log("The user2 become blacklisted BEFORE transfer" ,monkyToken.bannedAddresses(user2));
    console.log("The user1 whitelisted status BEFORE transfer" ,monkyToken.whitelistedAddresses(user1));
    console.log("The user2 whitelisted status BEFORE transfer" ,monkyToken.whitelistedAddresses(user2));

    assertEq(monkyToken.balanceOf(user1),transferValue,"Initial Bal of user 1");
    assertEq(monkyToken.balanceOf(user2),0,"Initial Bal of user 2");
    assertEq(monkyToken.bannedAddresses(user1),false);
    assertEq(monkyToken.bannedAddresses(user2),false);
    assertEq(monkyToken.whitelistedAddresses(user1),false,"The user1 whitelisted status BEFORE transfer");
    assertEq(monkyToken.whitelistedAddresses(user2),false,"The user2 whitelisted status BEFORE transfer");

    // Act
    vm.startPrank(user1);
    monkyToken.approve(user2, transferValue);
    monkyToken.transfer(user2, transferValue);
    vm.stopPrank();

    console.log("Final Bal of user1",monkyToken.balanceOf(user1));
    console.log("Final Bal of user2",monkyToken.balanceOf(user2));
    console.log("The user1 become blacklisted AFTER transfer" ,monkyToken.bannedAddresses(user1));
    console.log("The user2 become blacklisted AFTER transfer" ,monkyToken.bannedAddresses(user2));
    console.log("The user1 whitelisted status AFTER transfer" ,monkyToken.whitelistedAddresses(user1));
    console.log("The user2 whitelisted status AFTER transfer" ,monkyToken.whitelistedAddresses(user2));

    // Assert
    assertEq(monkyToken.balanceOf(user1),intialBalUser1 - transferValue,"Final Bal of user 1");
    assertEq(monkyToken.balanceOf(user2),transferValue,"Final Bal of user 2");
    assertEq(monkyToken.bannedAddresses(user1),true);
    assertEq(monkyToken.bannedAddresses(user2),true);
    assertEq(monkyToken.whitelistedAddresses(user1),false,"The user1 whitelisted status AFTER transfer");
    assertEq(monkyToken.whitelistedAddresses(user2),false,"The user2 whitelisted status AFTER transfer");
}
```

```
gurkiratsingh@Gurkirats-MacBook-Air monky-contract-main % forge test --mt test_transferWithLaunchDisabledWithNotWhiteListed
Users -vv
[⠒] Compiling...
No files changed, compilation skipped

Ran 1 test for test/Monky.t.sol:MonkyTokenTest
[PASS] test_transferWithLaunchDisabledWithNotWhiteListedUsers() (gas: 204916)
Logs:
  Initial Bal of user1 100
  Initial Bal of user2 0
  The user1 become blacklisted BEFORE transfer false
  The user2 become blacklisted BEFORE transfer false
  The user1 whitelisted status BEFORE transfer false
  The user2 whitelisted status BEFORE transfer false
  Final Bal of user1 0
  Final Bal of user2 100
  The user1 become blacklisted AFTER transfer true
  The user2 become blacklisted AFTER transfer true
  The user1 whitelisted status AFTER transfer false
  The user2 whitelisted status AFTER transfer false

Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 2.77s (212.06ms CPU time)
```

Remediation

Update the _update function to immediately revert any transaction involving unwhitelisted addresses when trading is disabled. This could be achieved by adding a check and revert statement after banning addresses, such as:

```
if (!tradingEnabled) {
bannedAddresses[_from] = true; bannedAddresses[_to] = true; revert
TradingNotEnabled();
}
```

This will ensure that any transaction attempted while trading is disabled is blocked outright and blocking the users involved as well.

5.4 Non-Whitelisted Sender Can Transfer Tokens During Restricted Launch Period

Result	**PASS**
Severity	Medium
Description	The `_update` function does not enforce a check for the sender (_from) address being whitelisted during the restricted launch phase. Currently, the logic only ensures that the recipient (_to) address is whitelisted if the launch is enabled and the current timestamp is within the restricted period (launchTime + 20 minutes). This allows non-whitelisted users to transfer tokens to whitelisted addresses during the restricted phase, bypassing the intended whitelist mechanism.
Location / Source File	MonkyToken.sol 51
Observation	Malicious users could exploit this vulnerability to bypass the intended trading restrictions. For instance: • A key purpose of implementing a whitelist during the launch phase is to control token transfers and limit them to approved participants. Allowing non-whitelisted senders to transfer undermines this restriction, creating a potential loophole. • Non-whitelisted users can use this loophole to engage in token transfers, potentially gaining an unfair advantage in token distribution or trading during the launch phase.
Remediation	Update the `_update` function to enforce a whitelist check for the sender (_from) address in addition to the recipient (_to) address during the restricted launch phase:

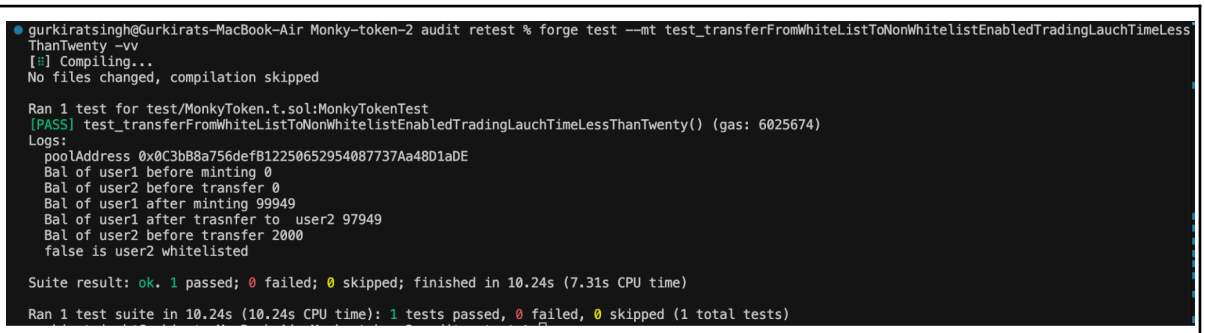
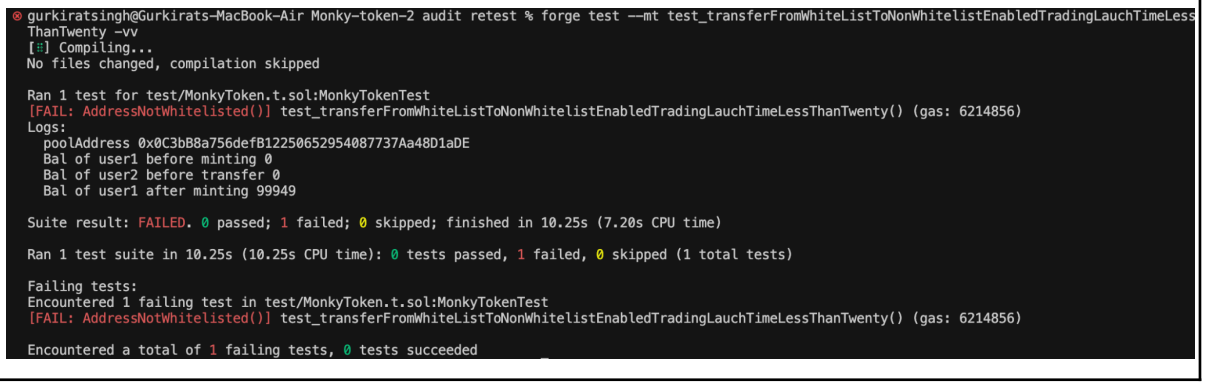
```
address[] listAddressToWhiteList;

function whitelistUsers(address[] memory users) public {
    vm.startPrank(owner);
    monkyToken.batchWhitelistAddresses(users);
    vm.stopPrank();
}

function test_transferFromNonWhitelistedToWhitelistedUser() public enableLaunch transferTokensToUser(200) {
    // Arrange
    uint256 transferValue = 200;
    uint256 intialBalUser1 = monkyToken.balanceOf(user1);
    listAddressToWhiteList.push(user2); // Add only user2 to the whitelist
    whitelistUsers(listAddressToWhiteList);
    console.log("Intial Bal of user1",monkyToken.balanceOf(user1));
    console.log("Intial Bal of user2",monkyToken.b                    ser2));
    console.log("The user1 whitelisted status BEFO                   r" ,monkyToken.whitelistedAddresses(user1));
    console.log("The user2 whitelisted status BEFORE transfer" ,monkyToken.whitelistedAddresses(user2));

    // Act
    vm.startPrank(user1);
    monkyToken.approve(user2, transferValue);
    monkyToken.transfer(user2, transferValue); // Perform the transfer
    vm.stopPrank();

    // Assert
    console.log("Final Bal of user1",monkyToken.balanceOf(user1));
    console.log("Final Bal of user2",monkyToken.balanceOf(user2));
    console.log("The user1 whitelisted status AFTER transfer" ,monkyToken.whitelistedAddresses(user1));
    console.log("The user2 whitelisted status AFTER transfer" ,monkyToken.whitelistedAddresses(user2));
    assertEq(monkyToken.balanceOf(user1), intialBalUser1 - transferValue, "Balance of user1 after transfer should be reduced by transferValue");
    assertEq(monkyToken.balanceOf(user2), transferValue, "Balance of user2 after transfer should be equal to transferValue");
}
```

```
gurkiratsingh@Gurkirats-MacBook-Air monky-contract-main % forge test --mt test_transferFromNonWhitelistedToWhitelistedUser -vv

[⠒] Compiling...
No files changed, compilation skipped

Ran 1 test for test/Monky.t.sol:MonkyTokenTest
[PASS] test_transferFromNonWhitelistedToWhitelistedUser() (gas: 236278)
Logs:
  Intial Bal of user1 200
  Intial Bal of user2 0
  The user1 whitelisted status BEFORE transfer false
  The user2 whitelisted status BEFORE transfer true
  Final Bal of user1 0
  Final Bal of user2 200
  The user1 whitelisted status AFTER transfer false
  The user2 whitelisted status AFTER transfer true

Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 2.77s (190.38ms CPU time)

Ran 1 test suite in 2.78s (2.77s CPU time): 1 tests passed, 0 failed, 0 skipped (1 total tests)
```

	`if (!whitelistedAddresses[_from] || !whitelistedAddresses[_to]) { revert MinFlokiBalanceNotMet(); }` This ensures that both the sender and receiver must be whitelisted during the restricted launch phase.

5.5 Unauthorized Transfers by Whitelisted Users To Non Whitelisted user During Launch Phase

Result	**FAIL**
Severity	Medium
Description	The smart contract fails to enforce proper restrictions on token transfers during the first 20 minutes after trading is enabled. Specifically, a whitelisted user is able to transfer tokens to an unauthorized (non-whitelisted) account.
Location / Source File	MonkyToken.sol 95
Observation	This flaw undermines the anti-bot measures during the critical launch phase, leading to the following issues: • Unauthorized Accumulation of Tokens: Non-whitelisted accounts can receive tokens from whitelisted accounts, bypassing intended restrictions and potentially enabling malicious actors to accumulate tokens. • Token Lock-in: While unauthorized recipients cannot further transfer tokens unless they are whitelisted, the tokens would remain locked in their wallets, disrupting the token distribution process and reducing overall liquidity during the launch phase. `function test_transferFromWhiteListToNonWhitelistEnabledTradingLauchTimeLessThanTwenty() public {` `addLiquidity();` `enableTrading();` `listAddressToWhiteList.push(user1);` `whitelistUsers(listAddressToWhiteList);` `deal(WBNB, address(user1), 3 * 1e18);` `console.log("Bal of user1 before minting" , monkyToken.balanceOf(user1));` `console.log("Bal of user2 before transfer" , monkyToken.balanceOf(user2));` `vm.startPrank(user1);` `IERC20(WBNB).approve(address(router), type(uint256).max);` `buySomeMonky(user1);` `console.log("Bal of user1 after minting" , monkyToken.balanceOf(user1));` `monkyToken.transfer(user2, 2000);` `vm.stopPrank();` `console.log("Bal of user1 after trasnfer to user2" , monkyToken.balanceOf(user1));` `console.log("Bal of user2 before transfer" , monkyToken.balanceOf(user2));` `console.log(monkyToken.whitelistedAddresses(user2), "is user2 whitelisted");` `}`

	
Remediation	To address this issue, modify the condition in the _update function to ensure that both the sender (_from) and the receiver (_to) must be whitelisted during the launch phase. Replace the `||` operator with `&&` as follows: Remove: `if (isTradingEnabled && block.timestamp < launchTime + 20 minutes) {` `bool isWhitelisted = isToWhitelisted || isFromWhitelisted;` Add: `if (isTradingEnabled && block.timestamp < launchTime + 20 minutes) {` `bool isWhitelisted = isToWhitelisted && isFromWhitelisted;` In provided POC we could see now a non whitelisted user is not able accept the token as smart contract throws an error 

5.6 Whitelist Mechanism ByPASS by Owner Transfers

Result	**FAIL**
Severity	Medium
Description	The current implementation of the MonkyToken contract allows the owner to transfer tokens to addresses that are not whitelisted. This undermines the whitelist mechanism, making it effectively obsolete, as owner-initiated transfers bypass the restrictions intended to enforce anti-bot and anti-whale protections during the early stages of trading. Specifically, the owner can transfer tokens to non-whitelisted addresses. This behavior contradicts the purpose of the whitelist mechanism, which is designed to control who can participate in token transfers during critical periods.
Location / Source File	MonkyToken.sol

Observation	• Security Risk: This loophole could be exploited to indirectly grant access to tokens for non-whitelisted or malicious actors. • Devaluation of Whitelist Functionality: The batchWhitelist functionality is rendered ineffective, as owner transfers override its purpose. `function test_transferFromOwnerToNonWhiteListUserEnabledLaunch() public {` `  addLiquidity();` `  enableTrading();` `  assertEq(monkyToken.whitelistedAddresses(user1),false);` `  assertEq(monkyToken.balanceOf(user1),0);` `  console.log("Balance of user 1 before transfer",monkyToken.balanceOf(user1));` `  console.log("User 1 is not whiteListed",monkyToken.whitelistedAddresses(user1));` `  vm.startPrank(owner);` `  monkyToken.approve(user1, 500);` `  monkyToken.transfer(user1, 500);` `  vm.stopPrank();` `  assertEq(monkyToken.whitelistedAddresses(user1),false);` `  assertEq(monkyToken.balanceOf(user1),500);` `  console.log("Balance of user 1 after transfer",monkyToken.balanceOf(user1));` `  console.log("User 1 is still not whiteListed",monkyToken.whitelistedAddresses(user1));` `}` `gurkiratsingh@Gurkirats-MacBook-Air Monky-token-2 audit retest % forge test --mt test_transferFromOwnerToNonWhiteListUserEnabledLaunch -vv` `[⠒] Compiling...` `No files changed, compilation skipped` `Ran 1 test for test/MonkyToken.t.sol:MonkyTokenTest` `[PASS] test_transferFromOwnerToNonWhiteListUserEnabledLaunch() (gas: 5719513)` `Logs:` `  poolAddress 0x0C3bB8a756defB12250652954087737Aa48D1aDE` `  Balance of user 1 before transfer 0` `  User 1 is not whiteListed false` `  Balance of user 1 after transfer 500` `  User 1 is still not whiteListed false` `Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 9.74s (6.81s CPU time)` `Ran 1 test suite in 9.75s (9.74s CPU time): 1 tests passed, 0 failed, 0 skipped (1 total tests)`
Remediation	To ensure the whitelist mechanism is respected for all transfers during the restricted period, including those initiated by the owner, modify the _update function to include additional checks for owner transfers.

5.7 Blacklisted Address Able to Transfer Funds To Its Wallet Before Launch

Result	**FAIL**
Severity	Medium
Description	When trading is disabled and an address is blacklisted after making a transfer, the first transaction goes through, transferring funds to the blacklisted wallet. However, the blacklisted address still manages to hold the funds.
Location / Source File	MonkyToken.sol 51
Observation	• The blacklisted address is able to transfer tokens to its wallet before being banned.

- Once the address is blacklisted, they cannot make further transfers, but the funds they obtained in the first transaction remain locked in their wallet.

Here is the test script below, which shows that while the blacklisted address receives tokens on the initial transfer to its wallet, it is unable to transfer tokens to any other address thereafter due to the ban

```
function test_transferFromBlacklistedAddressDisabledTrading() public {

    addLiquidity();
    // Trading not enabled so the bad user shouldn't be able to transfer right
    console.log("Initial balance of bad user",monkyToken.balanceOf(bad_user));
    console.log("Before tansfering status is user blackisted",monkyToken.bannedAddresses(bad_user));

    assertEq(monkyToken.balanceOf(bad_user), 0);
    assertEq(monkyToken.bannedAddresses(bad_user), false);

    deal(WBNB, address(bad_user), 3 * 1e18);
    vm.startPrank(bad_user);
    IERC20(WBNB).approve(address(router), type(uint256).max);
    uint256 amountOut = buySomeMonky(bad_user);
    vm.stopPrank();

    console.log("Final balance of bad user after 1st transfer",monkyToken.balanceOf(bad_user));
    console.log("After first tansfer status of user blackisted",monkyToken.bannedAddresses(bad_user));

    assertEq(monkyToken.balanceOf(bad_user), amountOut);
    assertEq(monkyToken.bannedAddresses(bad_user), true);

    vm.startPrank(bad_user);
    IERC20(WBNB).approve(address(router), type(uint256).max);
    vm.expectRevert();
    buySomeMonky(bad_user);
    vm.stopPrank();

    console.log("Final balance of bad user after 2nd transfer",monkyToken.balanceOf(bad_user));
    console.log("After first tansfer status of user blackisted",monkyToken.bannedAddresses(bad_user));

    assertEq(monkyToken.balanceOf(bad_user), amountOut);
    assertEq(monkyToken.bannedAddresses(bad_user), true);
}
```

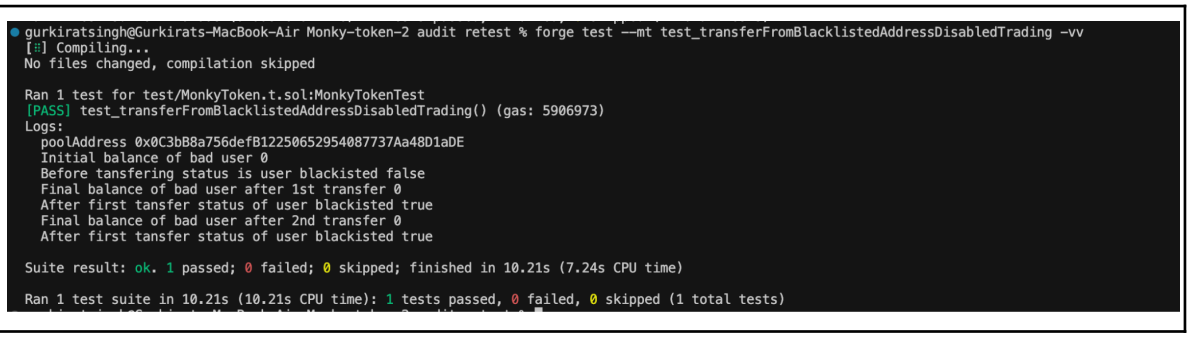
```
gurkiratsingh@Gurkirats-MacBook-Air Monky-token-2 audit retest % forge test --mt test_transferFromBlacklistedAddressDisabledTrading -vv
[⠊] Compiling...
No files changed, compilation skipped

Ran 1 test for test/MonkyToken.t.sol:MonkyTokenTest
[PASS] test_transferFromBlacklistedAddressDisabledTrading() (gas: 5929593)
Logs:
  poolAddress 0x0C3bB8a756defB12250652954087737Aa48D1aDE
  Initial balance of bad user 0
  Before tansfering status is user blackisted false
  Final balance of bad user after 1st transfer 99949
  After first tansfer status of user blackisted true
  Final balance of bad user after 2nd transfer 99949
  After first tansfer status of user blackisted true

Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 10.28s (7.35s CPU time)

Ran 1 test suite in 10.29s (10.28s CPU time): 1 tests passed, 0 failed, 0 skipped (1 total tests)
```

Remediation

To address this issue, update the bannedAddresses mechanism in the MonkyToken.sol contract to ensure that tokens are not transferred to blacklisted wallets during the initial transfer and state variable `bannedAddresses` is also updated. Specifically, modify the logic within the transfer function as follows:

```
// Sniper trap
if (!isTradingEnabled && isFromLiquidityPair && !isOwner) {
    bannedAddresses[_to] = true;
    return;
 }
```

You could see in the provided POC that now the user is not able to transfer the tokens to its wallet and is also banned.



5.8 State Variable Changes But No Event Is Emitted

Result	**PASS**
Severity	Low
Description	The `setLiquidityPair` and `enableTrading` functions in the MonkyToken contract modify the state variables `liquidityPair`, `tradingEnabled` and `launchTime`. However, they do not emit any events to notify external listeners about these changes, which can make it difficult for off-chain systems to track the changes in these values.
Location / Source File	MonkyToken.sol 26 35
Observation	The functions `setLiquidityPair()` and `enableTrading()` modify the state variables `liquidityPair, tradingEnabled` and `launchTime`, but they do not emit any events to signal these modifications.
Remediation	Add events to emit the new values of `liquidityPair`, `tradingEnabled` and `launchTime` after they are updated in the functions.
Reference	https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic

5.9 Solidity Pragma Should Be Specific, Not Wide

Result	**PASS**
Severity	Informational
Description	Using a wide version in the `Solidity pragma statement >=0.8.25` is discouraged. It's recommended to specify a particular version to ensure compatibility and avoid unexpected behavior due to potential breaking changes in future compiler versions.
Location / Source File	MonkyToken.sol 2
Observation	Failure to specify a specific version may lead to compatibility issues or unexpected behavior in future compiler versions. It's important to follow best practices to ensure the stability and security of the contracts.
Remediation	Update the pragma statements in the contracts to specify a particular version of Solidity. For example, replace `pragma solidity >=0.8.20;` with `pragma solidity 0.8.25;`.

Reference	https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used

5.10 Lack of Function to Unban Accidentally Banned Addresses

Result	FAIL
Severity	Informational
Description	The contract includes a mechanism to ban addresses, which is essential for mitigating malicious activities. However, there is no corresponding function to unban an address if it was accidentally banned. This limitation could create unnecessary friction during operations, particularly if a legitimate user is mistakenly banned and has no mechanism to be reinstated.
Location / Source File	MonkyToken.sol
Observation	Operational Inconvenience: Once an address is banned, there is no way to reverse the action without modifying the contract. This could lead to unnecessary complications if the ban was accidental or the situation warranting the ban has been resolved.
Remediation	Add a function to unban previously banned addresses. The function should be restricted to the contract owner to maintain control over the banning and unbanning process. Example implementation: `function unbanAddress(address _address) external onlyOwner {` `  if (!bannedAddresses[_address]) {` `  revert("Address is not banned");` `  }` `  bannedAddresses[_address] = false;` `}`

6. Tested for Scenarios

6.1 Verifying Launch Time Initialization Via enableTrading

Result	PASS
Function	enableTrading
Objective	To ensure that the enableTrading function is invoked by the contract owner only.
Test Case	`function test_startLaunchTime() public {` `  vm.startPrank(owner);` `  monkyToken.enableTrading();` `  vm.stopPrank();` `  assertEq(monkyToken.tradingEnabled(),true);` `  assertEq(monkyToken.launchTime(), block.timestamp);` `}`

Expected Behavior	• Upon calling enableTrading() by the contract owner, the launchTime should be updated to the value of block.timestamp. • Trading is enabled.

6.2 Ensuring Non Owner Can't Enable Trading Via enableTrading Function

Result	**PASS**
Function	enableTrading
Objective	To verify that only the contract owner can call the enableTrading function, ensuring unauthorized accounts cannot modify the launchTime or enable trading.
Test Case	`function test_nonOwnerStartLaunchTime() public {` `  vm.startPrank(user1);` `  vm.expectRevert(abi.encodeWithSelector(bytes4(keccak256("OwnableUnauthorizedAccount(address)")), user1));` `  monkyToken.enableTrading();` `  vm.stopPrank();` `}`
Expected Behavior	• If a non-owner account attempts to call the enableTrading function, the transaction should revert with the error OwnableUnauthorizedAccount(address) and the address of the unauthorized caller. • The launchTime should remain uninitialized, and trading should not be enabled.

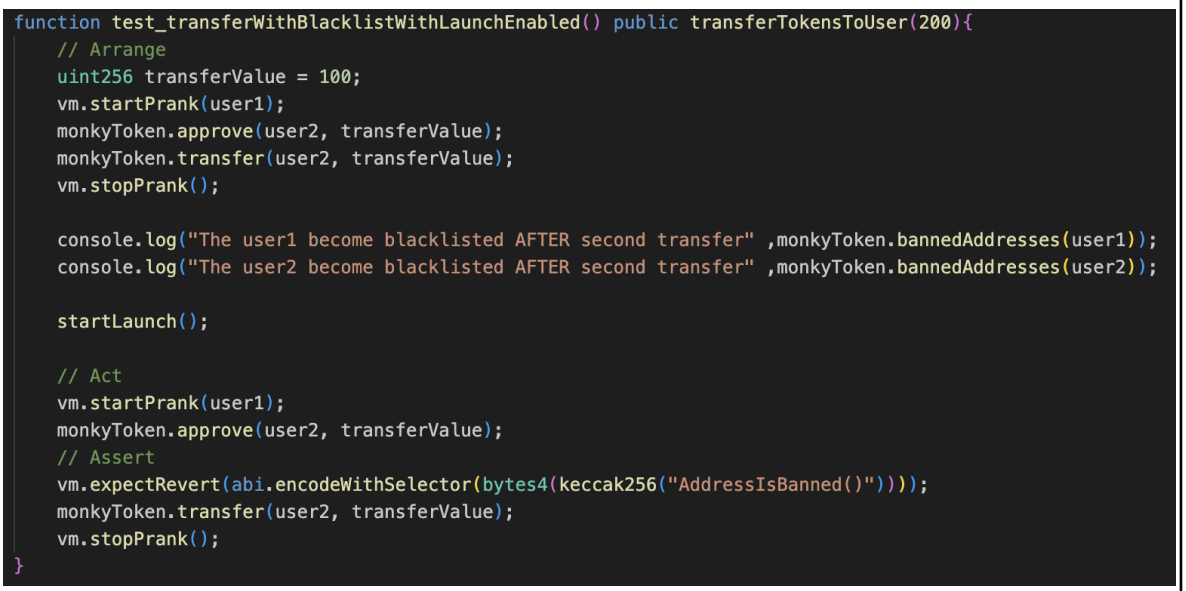
6.3 Verifying Transfers Initiated By The Owner

Result	**PASS**
Function	transfer
Objective	To validate that the contract owner is able to transfer tokens to other users without any restrictions, ensuring that owner privileges are upheld.

Test Case	
Expected Behavior	• The owner of the contract should be able to transfer tokens to any address. • The recipient's balance (user1) should increase by the transfer amount (transferValue). • No reverts or restrictions should occur during the transfer initiated by the owner.

```
function test_transferFromOwner() public {
    uint256 transferValue = 100;

    vm.startPrank(owner);
    monkyToken.approve(user1, transferValue);
    monkyToken.transfer(user1, transferValue);
    vm.stopPrank();

    uint256 balOfUserAfterTransfer = monkyToken.balanceOf(user1);
    assertEq(balOfUserAfterTransfer, transferValue);
}
```

6.4 Validating Transfers By Blacklisted Users After Launch

Result	**PASS**
Function	_update
Objective	To confirm that blacklisted users cannot transfer tokens even after the launch is enabled, ensuring that the blacklist functionality remains effective post-launch.
Test Case	

```
function test_transferWithBlacklistWithLaunchEnabled() public transferTokensToUser(200){
    // Arrange
    uint256 transferValue = 100;
    vm.startPrank(user1);
    monkyToken.approve(user2, transferValue);
    monkyToken.transfer(user2, transferValue);
    vm.stopPrank();

    console.log("The user1 become blacklisted AFTER second transfer" ,monkyToken.bannedAddresses(user1));
    console.log("The user2 become blacklisted AFTER second transfer" ,monkyToken.bannedAddresses(user2));

    startLaunch();

    // Act
    vm.startPrank(user1);
    monkyToken.approve(user2, transferValue);
    // Assert
    vm.expectRevert(abi.encodeWithSelector(bytes4(keccak256("AddressIsBanned()"))));
    monkyToken.transfer(user2, transferValue);
    vm.stopPrank();
}
```

Expected Behavior	• If a user (user1) is blacklisted before the launch, they should not be able to transfer tokens even after the launch is enabled. • An attempt by a blacklisted user to transfer tokens should result in a revert with the error AddressIsBanned(). • Blacklisted addresses should not be byPASS when trading is enabled.

6.5 Transfer From Whitelisted User To Non-Whitelisted User

Result	**PASS**
Function	_update
Objective	To ensure that transfers from a whitelisted user (user1) to a non-whitelisted user (user2) are not allowed during the launch phase if the recipient (user2) does not meet the parameters requirements.
Test Case	
Expected Behavior	• Transfers initiated by a whitelisted user (user1) should be validated against the recipient's whitelist status . • If the recipient (user2) is not whitelisted, the transfer should revert with the error MinFlokiBalanceNotMet().

```
function test_transferFromWhiteListedUserToNonWhitelistedUser() public enableLaunch transferTokensToUser(200) {
    // Arrange
    uint256 transferValue = 100;

    listAddressToWhiteList.push(user1);
    whitelistUsers(listAddressToWhiteList);

    // Act
    vm.startPrank(user1);
    monkyToken.approve(user2, transferValue);

    // Assert
    vm.expectRevert(abi.encodeWithSelector(bytes4(keccak256("MinFlokiBalanceNotMet()"))));
    monkyToken.transfer(user2, transferValue);
    vm.stopPrank();
}
```

6.6 Transfer After Launch Window Has Ended

Result	**PASS**
Function	_update
Objective	To verify that token transfers between whitelisted users function correctly after the 20-minute launch window has ended. During this time, the restrictions specific to the launch phase should no longer apply.

Test Case	

```
function test_transferAfterLaunchWindowHasEnded() public enableLaunch transferTokensToUser(200) {
    // Arrange
    uint256 transferValue = 200;
    uint256 initialBalUser1 = monkyToken.balanceOf(user1);
    uint256 initialBalUser2 = monkyToken.balanceOf(user2);

    // Whitelist user2
    listAddressToWhiteList.push(user1);
    listAddressToWhiteList.push(user2);
    whitelistUsers(listAddressToWhiteList);

    console.log("Initial Bal of user1", monkyToken.balanceOf(user1));
    console.log("Initial Bal of user2", monkyToken.balanceOf(user2));
    console.log("User1 whitelisted status BEFORE transfer", monkyToken.whitelistedAddresses(user1));
    console.log("User2 whitelisted status BEFORE transfer", monkyToken.whitelistedAddresses(user2));

    // Simulating time passing 21 minutes after launch
    vm.warp(block.timestamp + 21 minutes); // Warp time past the 20-minute window
    console.log("Launch Time", monkyToken.launchTime());
    console.log("Current Time Stamp", block.timestamp);

    // Act
    vm.startPrank(user1);
    monkyToken.approve(user2, transferValue);
    monkyToken.transfer(user2, transferValue);
    vm.stopPrank();

    // Assert
    console.log("Final Bal of user1", monkyToken.balanceOf(user1));
    console.log("Final Bal of user2", monkyToken.balanceOf(user2));
    console.log("User1 whitelisted status AFTER transfer", monkyToken.whitelistedAddresses(user1));
    console.log("User2 whitelisted status AFTER transfer", monkyToken.whitelistedAddresses(user2));

    // Assert the balances
    assertEq(monkyToken.balanceOf(user1), initialBalUser1 - transferValue, "Balance of user1 after transfer should be reduced by transferValue");
    assertEq(monkyToken.balanceOf(user2), initialBalUser2 + transferValue, "Balance of user2 after transfer should be increased by transferValue");
}
```

Expected Behavior	• After the 20-minute launch window has elapsed, transfers between whitelisted users should proceed without restrictions. • The sender's balance should be reduced by the transfer value, and the recipient's balance should increase by the same amount.

6.7 Transfer Exceeds Maximum Wallet Cap

Result	**PASS**
Function	_update
Objective	To verify that the contract enforces the MaxWalletCapExceeded restriction, preventing transfers that would result in a recipient holding a balance exceeding the maximum wallet cap.

Test Case	``` function test_transferExceedsMaxWalletCap() public enableLaunch transferTokensToUser(1500){ // Arrange uint256 transferValue = 1100; listAddressToWhiteList.push(user1); listAddressToWhiteList.push(user2); whitelistUsers(listAddressToWhiteList); // Act vm.startPrank(user1); monkyToken.approve(user2, transferValue); // Assert vm.expectRevert(abi.encodeWithSelector(bytes4(keccak256("MaxWalletCapExceeded()")))); monkyToken.transfer(user2, transferValue); vm.stopPrank(); } ```
Expected Behavior	• If a transfer would cause the recipient's balance to exceed the maximum wallet cap, the transfer should be reverted. • The contract should emit the MaxWalletCapExceeded() error, and no token balances should be modified.

6.8 Passing Checks

Here is the POC attached that depicts the execution of the test scripts and the corresponding results.

```
gurkiratsingh@Gurkirats-MacBook-Air monky-contract-main % forge test
[⠒] Compiling...
[⠒] Compiling 1 files with Solc 0.8.25
[⠒] Solc 0.8.25 finished in 3.08s
Compiler run successful!

Ran 12 tests for test/Monky.t.sol:MonkyTokenTest
[PASS] test_batchWhitelistAddresses_OwnerCanWhitelistAddresses() (gas: 132380421)
[PASS] test_getMaxWalletAmount_ReturnsCorrectValue() (gas: 11779)
[PASS] test_nonOwnerStartLaunchTime() (gas: 13956)
[PASS] test_startLaunchTime() (gas: 40752)
[PASS] test_transferAfterLaunchWindowHasEnded() (gas: 280826)
[PASS] test_transferExceedsMaxWalletCap() (gas: 255120)
[PASS] test_transferFromNonWhitelistedToWhitelistedUser() (gas: 236278)
[PASS] test_transferFromOwner() (gas: 73985)
[PASS] test_transferFromWhiteListedUserToNonWhitelistedUser() (gas: 207342)
[PASS] test_transferWithBlacklistAddressWithLaunchDisabled() (gas: 194977)
[PASS] test_transferWithBlacklistWithLaunchEnabled() (gas: 212837)
[PASS] test_transferWithLaunchDisabledWithNotWhiteListedUsers() (gas: 204916)
Suite result: ok. 12 passed; 0 failed; 0 skipped; finished in 3.02s (1.50s CPU time)

Ran 1 test suite in 3.03s (3.02s CPU time): 12 tests passed, 0 failed, 0 skipped (12 total tests)
```

7. Auditing Approach and Methodologies Applied

The solidity smart contract was audited in a comprehensive approach to ensure the highest level of security and reliability. Careful attention was given to the following key areas to ensure the overall quality of code:

- **Code quality and structure**: We conducted a detailed review of the codebase to identify any potential issues related to code structure, readability, and maintainability. This included analyzing the overall architecture of the solidity smart contract and reviewing the code to ensure it follows best practices and coding standards.
- **Security vulnerabilities**: Our team used manual techniques to identify any potential security vulnerabilities that could be exploited by attackers. This involved a thorough analysis of the code to identify any potential weaknesses, such as buffer overflows, injection vulnerabilities, Signatures, and deprecated functions.
- **Documentation and comments**: Our team reviewed the code documentation and comments to ensure they accurately describe the code's intended behavior and logic. This helps developers to better understand the codebase and make modifications without introducing new issues.
- **Compliance with best practices**: We checked that the code follows best practices and coding standards that are recommended by the solidity community and industry experts. This ensures that the solidity smart contract is secure, reliable, and efficient.

Our audit team followed OWASP and Ethereum(Solidity) community security guidelines for this audit. As a result, we were able to identify potential issues and provide recommendations to improve the smart contract security and performance.

Throughout the audit of the smart contract, our team placed great emphasis on ensuring the overall quality of the code and the use of industry best practices. We meticulously reviewed the codebase to ensure that it was thoroughly documented and that all comments and logic aligned with the intended behavior. Our approach to the audit was comprehensive, methodical, and aimed at ensuring that the smart contract was secure, reliable, and optimized for performance.

8. Limitations on Disclosure and Use of this Report

This report contains information concerning potential details of the MonkyToken Project and methods for exploiting them. Entersoft recommends that special precautions be taken to protect the confidentiality of both this document and the information contained herein. Security Assessment is an uncertain process, based on past experiences, currently available information, and known threats. All information security systems, which by their nature are dependent on human beings, are vulnerable to some degree. Therefore, while Entersoft considers the major security vulnerabilities of the analyzed systems to have been identified, there can be no assurance that any exercise of this nature will identify all possible vulnerabilities or propose exhaustive and operationally viable recommendations to mitigate those exposures. In addition, the analysis set forth herein is based on the technologies and known threats as of the date of this report. As technologies and risks change over time, the vulnerabilities associated with the operation of the MonkyToken solidity smart contract described in this report, as well as the actions necessary to reduce the exposure to such vulnerabilities will also change. Entersoft makes no undertaking to supplement or update this report based on changed circumstances or facts of which Entersoft becomes aware after the date here of, absent a specific written agreement to perform the supplemental or updated analysis. This report may recommend that Entersoft use certain software or hardware products manufactured or maintained by other vendors. Entersoft bases these recommendations upon its prior experience with the capabilities of those products. Nonetheless, Entersoft does not and cannot warrant that a particular product will work as advertised by the vendor, nor that it will operate in the manner intended. This report was prepared by Entersoft for the exclusive benefit of MonkyToken and is proprietary information. The Non-Disclosure Agreement (NDA) in effect between Entersoft and MonkyToken governs the disclosure of this report to all other parties including product vendors and suppliers.