



# Droggy, Contract Code Review and Security Analysis Report

---

Customer: Droggy  
Prepared on: 13th December 2023  
Platform: Binance Smart Chain  
Language: Solidity

**[rdauditors.com](https://rdauditors.com)**

## Table of Contents

<b>Disclaimer</b>	<b>3</b>
<b>Document</b>	<b>4</b>
<b>Introduction</b>	<b>5</b>
<b>Project Scope</b>	<b>6</b>
<b>Executive Summary</b>	<b>7</b>
<b>Code Quality</b>	<b>7</b>
<b>Documentation</b>	<b>9</b>
<b>Use of Dependencies</b>	<b>9</b>
<b>AS-IS Overview</b>	<b>10</b>
<b>Code Flow Diagram - Droggy</b>	<b>13</b>
<b>Interaction Diagram</b>	<b>14</b>
<b>Inheritance Diagram</b>	<b>15</b>
<b>Code Flow Diagram - Slither Results Log</b>	<b>16</b>
<b>Audit Findings</b>	<b>20</b>
<b>Conclusion</b>	<b>22</b>
<b>Note For Contract Users</b>	<b>22</b>
<b>Our Methodology</b>	<b>24</b>
<b>Disclaimers</b>	<b>27</b>

## Disclaimer

This document may contain confidential information about its systems and intellectual property of the customer as well as information about potential vulnerabilities and methods of their exploitation.

The report containing confidential information can be used internally by the customer or it can be disclosed publicly after all vulnerabilities are fixed - upon the decision of the customer.

---

## Document

Name	Smart Contract Code Review and Security Analysis Report of Droggy
Platform	Binance Smart Chain/ Solidity
File	Droggy.sol
MD5 hash	b32245b80a9fb6ce00b42d6cd1dc5b2b
SHA256 hash	e3e67d7bfc059685c0a1bf8a0b787be587c217a3609702db0af6612f22f4a71a
Date	13/12/2023

## Introduction

RD Auditors (Consultant) were contracted by Droggy (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report represents the findings of the security assessment of the customer's smart contract and its code review conducted between 12th - 14th December 2023.

This contract consists of one file.

## Project Scope





The scope of the project is a smart contract. We have scanned this smart contract for commonly known and more specific vulnerabilities, below are those considered (the full list includes but is not limited to):

- Reentrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with (Unexpected) Throw
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Byte array vulnerabilities
- Style guide violation
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Unchecked external call - Unchecked math
- Unsafe type inference
- Implicit visibility level

## Executive Summary






According to the assessment, the customer's solidity smart contract is now **Secured**.

You are Here

 Insecure  Poorly Secured  Secure  Well-Secured

Automated checks are with smartDec, Mythril, Slither and remix IDE. All issues were performed by our team, which included the analysis of code functionality, the manual audit found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the audit overview section. The general overview is presented in the AS-IS section and all issues found are located in the audit overview section.

We found the following;

Total Issues	0
 Critical	0
 High	0
 Medium	0
 Low	0
 Very Low	0

## Code Quality

The libraries within this smart contract are part of a logical algorithm. A library is a different type of smart contract that contains reusable code. Once deployed on the blockchain (only once), it is assigned to a specific address and its properties/methods can be reused many times by other contracts.

The Droggy team has not provided scenario and unit test scripts, which would help to determine the integrity of the code in an automated way.

Overall, the code is not commented. Commenting can provide rich documentation for functions, return variables and more. Use of the Ethereum Natural Language Specification Format (NatSpec) for commenting is recommended.



## Documentation

We were given the Droggy code as a link:

<https://bscscan.com/address/0x556ee4EaB4fbF6DDc4C05285966FB839F424c8a8#code>

The hash of that file is mentioned in the table. As mentioned above, it's recommended to write comments on smart contract code, so anyone can quickly understand the programming flow as well as complex code logic.

Comments are very helpful in understanding the overall architecture of the protocol. It also provides a clear overview of the system components, including helpful details, like the lifetime of the background script.

## Use of Dependencies

As per our observation, the libraries are used in this smart contract infrastructure. Those were based on well known industry standard open source projects and even core code blocks that are written well and systematically.

## AS-IS Overview

### Droggy.sol

#### File And Function Level Report

Contract: ERC20  
Inherit: Context, IERC20, IERC20Metadata  
Observation: Passed  
Test Report: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	name	read	Passed	All Passed	No Issue	Passed
2	symbol	read	Passed	All Passed	No Issue	Passed
3	decimals	read	Passed	All Passed	No Issue	Passed
4	totalSupply	read	Passed	All Passed	No Issue	Passed
5	balanceOf	read	Passed	All Passed	No Issue	Passed
6	transfer	write	Passed	All Passed	No Issue	Passed
7	allowance	read	Passed	All Passed	No Issue	Passed
8	approve	write	Passed	All Passed	No Issue	Passed
9	transferFrom	write	Passed	All Passed	No Issue	Passed
10	IncreaseAllowance	write	Passed	All Passed	No Issue	Passed
11	decreaseAllowance	write	Passed	All Passed	No Issue	Passed
12	_transfer	internal	Passed	All Passed	No Issue	Passed
13	_mint	internal	Passed	All Passed	No Issue	Passed
14	_burn	internal	Passed	All Passed	No Issue	Passed

15	_approve	internal	Passed	All Passed	No Issue	Passed
16	_spendAllowance	internal	Passed	All Passed	No Issue	Passed
17	_beforeTokenTransfer	internal	Passed	All Passed	No Issue	Passed
18	_afterTokenTransfer	internal	Passed	All Passed	No Issue	Passed

Contract: Ownable

Inherit: Context

Observation: Passed

Test Report: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	Owner	read	Passed	All Passed	No Issue	Passed
2	_CheckOwner	internal	Passed	All Passed	No Issue	Passed
3	renounceOwnership	onlyOwner	Passed	All Passed	No Issue	Passed
4	transferOwnership	onlyOwner	Passed	All Passed	No Issue	Passed

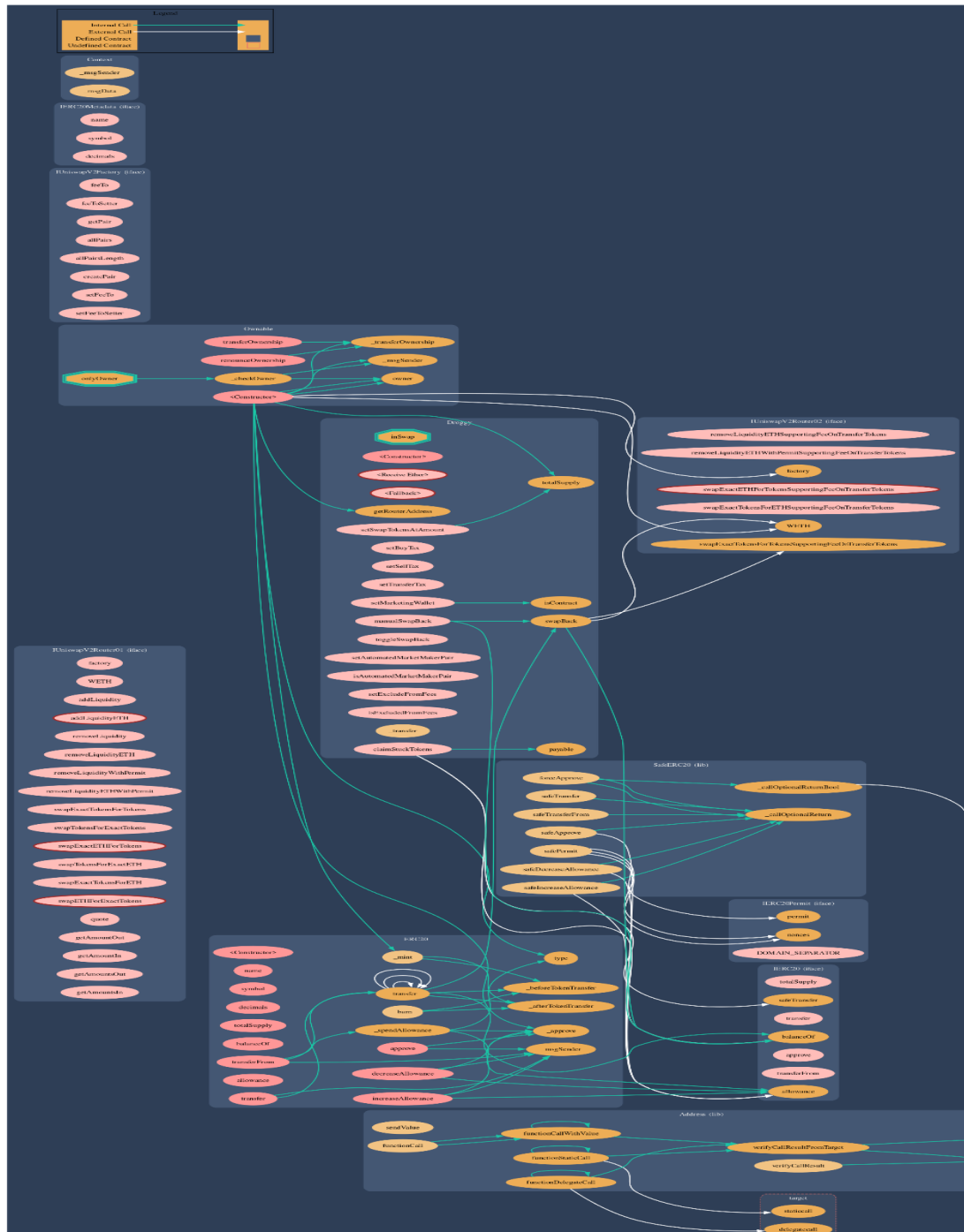
Contract: Droggy

Inherit: Ownable, ERC20

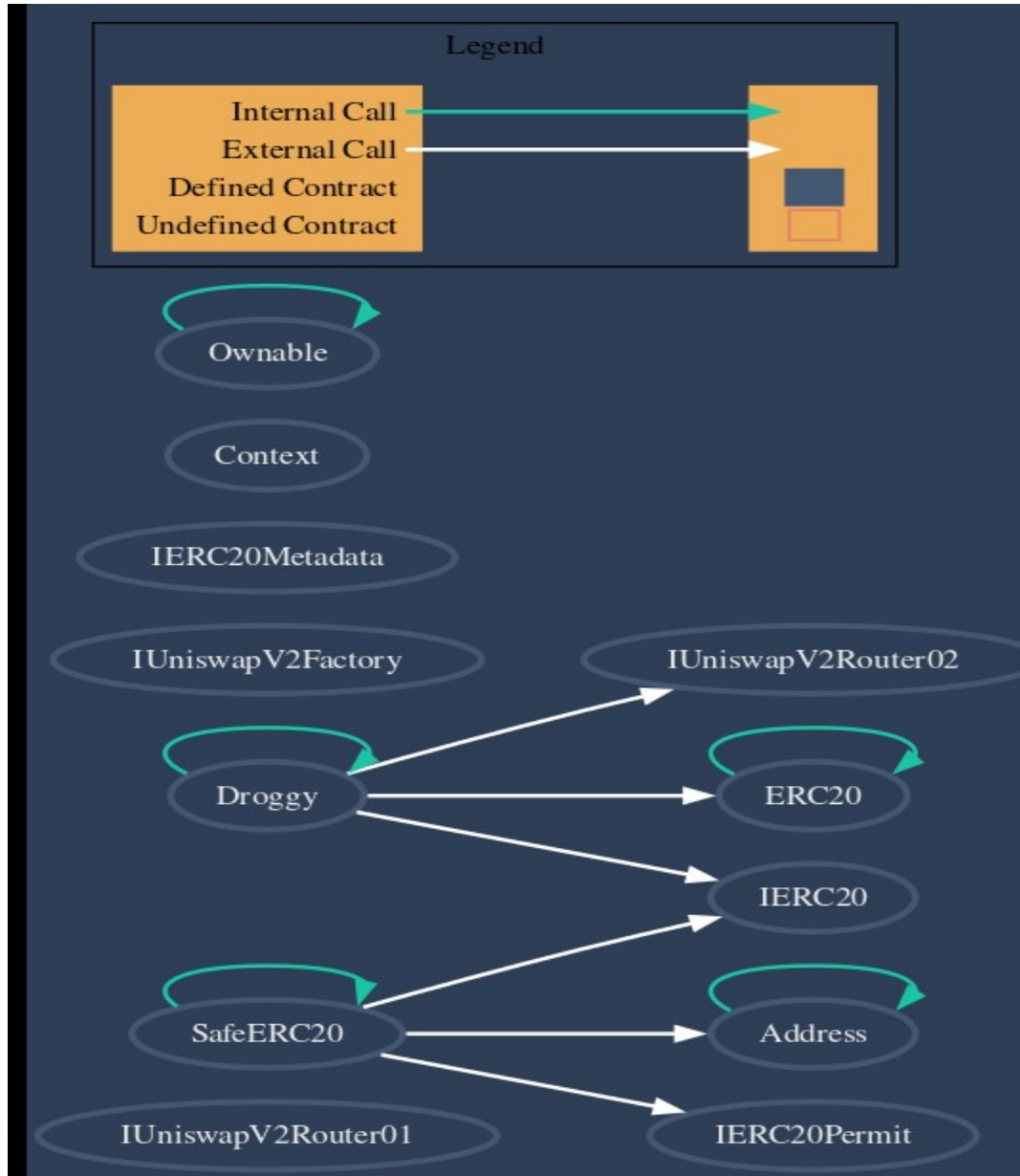
Observation: Passed

Test Report: Passed

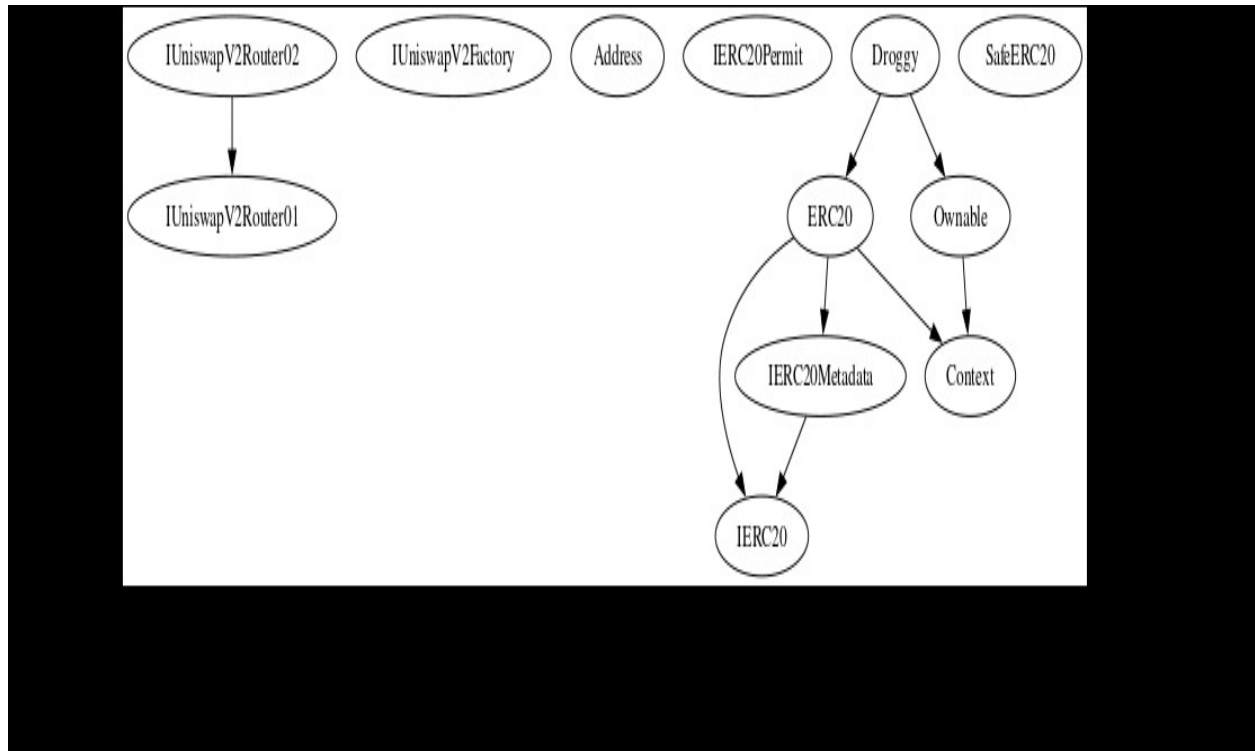
Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	isContract	read	Passed	All Passed	No Issue	Passed
2	getRouterAddress	read	Passed	All Passed	No Issue	Passed
3	ClaimStuckTokens	OnlyOwner	Passed	All Passed	No Issue	Passed
4	SetBuyTax	OnlyOwner	Passed	All Passed	No Issue	Passed
5	SetSellTax	OnlyOwner	Passed	All Passed	No Issue	Passed
6	SetTransferTax	OnlyOwner	Passed	All Passed	No Issue	Passed
7	SetMarketing Wallet	OnlyOwner	Passed	All Passed	No Issue	Passed
8	SetSwapTokensAtAmount	OnlyOwner	Passed	All Passed	No Issue	Passed
9	toggleSwapBack	OnlyOwner	Passed	All Passed	No Issue	Passed
10	SetAutomatedMarketMakerPair	OnlyOwner	Passed	All Passed	No Issue	Passed
11	isAutomatedMarketMakerPair	read	Passed	All Passed	No Issue	Passed
12	SetExcludeFromFees	OnlyOwner	Passed	All Passed	No Issue	Passed
13	isExcludeFromFees	read	Passed	All Passed	No Issue	Passed
14	_transfer	internal	Passed	All Passed	No Issue	Passed
15	SwapBack	internal	Passed	All Passed	No Issue	Passed
16	manualSwapBack	external	Passed	All Passed	No Issue	Passed



## Interaction Diagram



## Inheritance Diagram



## Code Flow Diagram - Slither Results Log

```
Reentrancy in Droggy._transfer(address,address,uint256) (droggy.sol#1086-1140):
  External calls:
    - swapBack() (droggy.sol#1109)
      - uniswapV2Router.swapExactTokensForTokensSupportingFeeOnTransferTokens(contractTokenBalance,0,path,address(marketingWallet),block.timestamp) (droggy.sol#1150-1161)
  State variables written after the call(s):
    - super._transfer(from,address(this),fees) (droggy.sol#1135)
      - _balances[from] = fromBalance - amount (droggy.sol#742)
      - _balances[to] += amount (droggy.sol#744)
  ERC20._balances (droggy.sol#628) can be used in cross function reentrancies:
    - ERC20._mint(address,uint256) (droggy.sol#752-764)
    - ERC20._transfer(address,address,uint256) (droggy.sol#726-750)
    - ERC20.balanceOf(address) (droggy.sol#658-662)
    - super._transfer(from,to,amount) (droggy.sol#1139)
      - _balances[from] = fromBalance - amount (droggy.sol#742)
      - _balances[to] += amount (droggy.sol#744)
  ERC20._balances (droggy.sol#628) can be used in cross function reentrancies:
    - ERC20._mint(address,uint256) (droggy.sol#752-764)
    - ERC20._transfer(address,address,uint256) (droggy.sol#726-750)
    - ERC20.balanceOf(address) (droggy.sol#658-662)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
```

```
Address.functionCall(address,bytes) (droggy.sol#271-282) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (droggy.sol#292-304) is never used and should be removed
Address.functionDelegateCall(address,bytes) (droggy.sol#355-365) is never used and should be removed
Address.functionDelegateCall(address,bytes,string) (droggy.sol#367-380) is never used and should be removed
Address.functionStaticCall(address,bytes) (droggy.sol#328-338) is never used and should be removed
Address.functionStaticCall(address,bytes,string) (droggy.sol#340-353) is never used and should be removed
Address.sendValue(address,uint256) (droggy.sol#258-269) is never used and should be removed
Address.verifyCallResult(bool,bytes,string) (droggy.sol#398-408) is never used and should be removed
Context._msgData() (droggy.sol#622-624) is never used and should be removed
ERC20._burn(address,uint256) (droggy.sol#766-782) is never used and should be removed
SafeERC20._callOptionalReturnBool(IERC20,bytes) (droggy.sol#597-606) is never used and should be removed
SafeERC20.forceApprove(IERC20,address,uint256) (droggy.sol#547-565) is never used and should be removed
SafeERC20.safeApprove(IERC20,address,uint256) (droggy.sol#494-507) is never used and should be removed
SafeERC20.safeDecreaseAllowance(IERC20,address,uint256) (droggy.sol#525-545) is never used and should be removed
SafeERC20.safeIncreaseAllowance(IERC20,address,uint256) (droggy.sol#509-523) is never used and should be removed
SafeERC20.safePermit(IERC20Permit,address,address,uint256,uint256,uint8,bytes32,bytes32) (droggy.sol#567-584) is never used and should be removed
```



---

# Solidity Static Analysis

## Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in `Address.functionCallWithValue(address,bytes,uint256,string)`: Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.

[more](#)

Pos: 326:5:

## Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in `SafeERC20.safeDecreaseAllowance(contract IERC20,address,uint256)`: Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.

[more](#)

Pos: 544:1:

## Inline assembly:

The Contract uses inline assembly, this is only advised in rare cases. Additionally static analysis modules do not parse inline Assembly, this can lead to wrong analysis results.

[more](#)

Pos: 440:39:

## Block timestamp:

Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.

[more](#)

Pos: 1165:277:

**Low level calls:**

Use of "call": should be avoided whenever possible. It can lead to unexpected behavior if return value is not handled properly. Please use Direct Calls via specifying the called contract's interface.

[more](#)

Pos: 288:8:

**Gas costs:**

Gas requirement of function ERC20.name is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Pos: 665:4:

**Constant/View/Pure functions:**

IUniswapV2Router01.removeLiquidityWithPermit(address,address,uint256,uint256,uint256,address,uint256,bool,uint8,bytes32,bytes32) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.

[more](#)

Pos: 97:4:

**Constant/View/Pure functions:**

IUniswapV2Router01.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.

[more](#)

Pos: 54:6:

**Constant/View/Pure functions:**

IUniswapV2Router01.removeLiquidityETHWithPermit(address,uint256,uint256,uint256,address,uint256,bool,uint8,bytes32,bytes32) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.

[more](#)

Pos: 109:14:

---

## Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to lost tokens etc.
High	High level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g. public access to crucial functions.
Medium	Medium level vulnerabilities are important to fix; however, they cannot lead to lost tokens.
Low	Low level vulnerabilities are most related to outdated, unused etc. These code snippets cannot have a significant impact on execution.
Lowest Code Style/ Best Practice	Lowest level vulnerabilities, code style violations and information statements cannot affect smart contract execution and can be ignored.

## Audit Findings

### Critical:

No critical severity vulnerabilities were found.

### High:

No high severity vulnerabilities were found.

### Medium:

No medium severity vulnerabilities were found.

### Low:

No low severity vulnerabilities were found.

### Very Low:

No very low severity vulnerabilities were found.

### Discussion

1. Overpowered owners: The contracts are using 11 functions that can only be called by the owners. Giving too many privileges to the owners via critical functions might put the user's funds at risk if the owners are compromised or if a rug-pulling attack takes place.

2. Users with token balance more than 5% - some addresses contain more than 5% of circulating token supply.

0x34533e98cae1276bf150e14f19acbf087a2e5672 13.02%

0xa0387adba7636722abe119cbf9220ce0b9938b0b 8.084%

Token distribution plays an important role when controlling the price of an asset.

### 3. ERC20 race condition

The contract is vulnerable to ERC-20 approve Race condition vulnerability. ERC-20 approve function is vulnerable to a frontrunning attack which can be exploited by the token receiver to withdraw more tokens than the allowance. Proper mitigation steps should be implemented to prevent such vulnerabilities.

4. Hardcoded addresses: The contract was hardcoding addresses in the code. This may represent that those parameters can never be changed or updated unless it's a proxy contract. It is recommended to go through the code to know more about these hardcoded values and its use.

## Conclusion

We were given a contract file and have used all possible tests based on the given object. so it is ready for mainnet deployment. We have used all the latest static tools and manual observations to cover maximum possible test cases to scan everything.

The security state of the reviewed contract is **"Secured"**.

## Note For Contract Users

**ClaimStuckTokens:** This function allows the owner to claim either native Ether or ERC-20 tokens that are accidentally sent to the contract. If the specified token address is the zero address, it transfers Ether; otherwise, it transfers the ERC-20 token balance to the owner. The owner can use this function to recover any funds that might be stuck in the contract.

**SetBuyTax:** This function allows the owner to update the buy tax for the contract. It performs checks to ensure the new value is different from the current one and within a specified limit. If the conditions are met, it updates the buy tax and emits an event to log the change.

**SetSellTax:** This function allows the owner to update the sell tax for the contract. It performs checks to ensure the new value is different from the current one and within a specified limit. If the conditions are met, it updates the sell tax and emits an event to log the change.

**SetTransferTax:** This function allows the owner to update the transfer tax for the contract. It performs checks to ensure the new value is different from the current one and within a specified limit. If the conditions are met, it updates the transfer tax and emits an event to log the change.

**SetMarketingWallet:** This function allows the owner to update the marketing wallet address for the contract. It performs checks to ensure the new address is different from the current one, not the zero address, and not a contract. If the conditions are met, it updates the marketing wallet address and emits an event to log the change.

**SetSwapTokensAtAmount:** This function allows the owner to update the threshold at which a token swap operation will be triggered. It performs checks to ensure the new value is different from the current one and meets a minimum threshold. If the conditions are met, it updates swapTokensAtAmount

**ToggleSwapBack:** function allows the owner to toggle the status of the swap back feature. It checks if the new status is different from the current one, and if so, it updates isSwapBackEnabled and emits an event to log the change. This can be useful for enabling or disabling certain functionalities in the contract, such as automatic token swaps.

**SetAutomatedMarketMakerPair:** This function allows the owner to set the status of an automated market maker pair. It performs checks to ensure that unnecessary state changes are avoided and prevents the owner from setting the status for the Uniswap V2 pair directly. If the conditions are met, it updates the status and emits an event to log the change

**SetExcludeFromFee:** This function allows the owner to set the exclusion status from fees for a specific account. It performs checks to ensure that unnecessary state changes are avoided. If the conditions are met, it updates the exclusion status and emits an event to log the change.



## Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

### Manual Code Review

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

### Vulnerability Analysis

Our audit techniques included manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's web site to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

## Documenting Results

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyse the feasibility of an attack in a live system.

## Suggested Solutions

We search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinised by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

---

## Disclaimers

### RD Auditors Disclaimer

The smart contracts given for audit have been analysed in accordance with the best industry practices at the date of this report, in relation to: cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment and functionality (performing the intended functions).

Because the total number of test cases are unlimited, the audit makes no statements or warranties on the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only - we recommend proceeding with several independent audits and a public bug bounty program to ensure security of smart contracts.

### Technical Disclaimer

Smart contracts are deployed and executed on the blockchain. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to hacks. Thus, the audit can't guarantee explicit security of the audited smart contracts.



Email: [info@rdauditors.com](mailto:info@rdauditors.com)

Website: [www.rdauditors.com](http://www.rdauditors.com)

