BLOCKSTARS

Smart Contract Security Audit

PROJECT:	SPEEDY SMART CONTRACT
ANALYSIS TYPE:	BASIC AUDIT
VERSION:	1
START DATE:	21 NOVEMBER 2024
AUDIT RESULT:	SECURE

REPORT CONTENTS

DECLARATION

This is a **BLOCKSTARS TECHNOLOGY PTY LTD** smart contract security audit created for the Speedy smart contract.

Results and documentation within this report contain confidential information regarding the clients' smart contracts.

Analysis results of the smart contract is supplied containing lists of vulnerabilities and malicious code which could be used to malform the project.

The Speedy smart contract is public.

For sake of professionalism, confidentiality of risks, and abiding to information security best practices, BLOCKSTARS TECHNOLOGY will treat this report with highest confidentiality priority level.

1. Introduction

This security audit report contains valuable information regarding the Speedy smart contract. Security vulnerabilities, smart contract best practices, and possible attack vectors are identified using standardised tests with static, dynamic, and symbolic analysis tools. We outlined our systematic approach to evaluate potential security issues within the core smart contract of this project and provide an audit summary with remedies for mitigating the vulnerability findings.

2. Executive Summary

The BLOCKSTARS TECHNOLOGY smart contact audit team had identified vulnerabilities in the SPEEDY contract related to potential attack vectors. These vulnerabilities were mitigated by distributing all tokens to external wallets and renouncing contract ownership, thereby eliminating the risks associated with owner-controlled functions.

As a result, the contract is now **SECURE**, with no risk of token lock-up or misuse through ownership privileges and external calls.

3. Project Context

ITEM	DESCRIPTION
Website	https://speedytheturtle.xyz/
Source	Smart contract address: 0xc8F69A9B46B235DE8d0b77c355FFf7994F1B090f
Language	Solidity
Blockchain	Ethereum (ETH)
Project Location	https://etherscan.io/address/0xc8f69a9b46b235de8d0b77c355fff7994f1b090f#code
Audit type	**BASIC**
Analysis Methods	Static, Dynamic, and symbolic analysis, and report with recommendations.
Audit Team	Pura, Faizin, and Marcus
Approved By	Nilanga Saluwadana
Timeline	**From:** 21st NOV 2024 **To:** 26th NOV 2024
Change logs	V1

4. Audit Scope

Scope of this project is to identify any identifiable vulnerabilities to improve the coding practices required for the given smart contract.

The following security audit steps were conducted:

- Basic audit using smart contract security analysis tools which include static, dynamic, and Symbolic testing methods.

Audit method:	BASIC	STANDARD	DEEP

ITEM	DESCRIPTION
Repository	https://etherscan.io/address/0xc8f69a9b46b235de8d0b77c355fff7994f1b090f#code
Address	0xc8F69A9B46B235DE8d0b77c355FFf7994F1B090f
Technical Documentation	Business logics provided: **NO**

5. Definitions

5.1. Security Severities

SEVERITY	DESCRIPTION
High	High level vulnerabilities may or may not be difficult to exploit, however they contain significant impact potential on the smart contract's execution due to lack of security access control (Example: Public access to crucial data and manipulation of functions).
Medium	Medium vulnerabilities may not lead to loss of assets or data, but it is important to fix these issues which may be used to exploit the project.
Low	Low level vulnerabilities are related to out-dated, unused code snippets. They may not have a significant impact on contract execution but is recommended to execute any fixes to them.
Informational	Does not contain vulnerabilities, but requires best practices, code standards and documentary code. It is recommended to execute a solution for these findings.

6. Analytic Statistics

Security Issues	#	Description	Type	Contract
Programming Issues	1	Unchecked return value from low-level external calls	SWC-104	P
	2	Floating pragma is set	SWC-103	P
	3	Error Handling and logging are implemented	Custom	F
	4	State variable should not be used without being initialized	Custom	P
	5	Is inheritance used properly	SWC-125	P
	6	External components used insecurely	Custom	P
	7	Functions that loop over unbounded data structures	Custom	F
	8	Msg.value should not be used in a loop	Custom	P
Code Specifications, Best Practices	9	Use of the "constant" state mutability modifier is deprecated.	SWC-111	P
	10	Use of the "throw" state mutability modifier is deprecated.	SWC-111	P
	11	Strict equalities should not render the function to be unusable	Custom	P
	12	Use of best Practices	Custom	F
	13	Business logic is implemented as per the documents provided	Custom	
Optimise Gas	14	Message call with hardcoded gas amount	SWC-134	P
	15	Check for gas usage and minimize gas consumption.	Custom	F
Risk to Attacks	16	Code contains suicidal, greedy, and prodigal instructions	SWC-106	P
	17	Contract is Haltable	Custom	P
	18	Adopt checks-effects-interactions patterns for any transactions of value	Custom	F
	19	Reduce and remove unnecessary code to reduce attack surface area.	Custom	P
	20	Timestamps should not be used to execute critical functions.	SWC-116	P
	21	Sensitive data in normal form should not be stored on-chain	Custom	P
	22	Vulnerable to Integer overflow and under-flow	SWC-101	P

F Fail P Pass

7. Functional Audit Overview

#	Function	Type	Observation	Status
1	**Constructor()**	Constructor, Inherits ERC20	Initializes ERC20 token with name, symbol, and other parameters.	P
2	**receive()**	External onlyOwner	Allows contract to receive Ether directly.	P
3	**openTrading()**	External onlyOwner	Enables trading, sets fees, and creates Uniswap pair.	P
4	**setMkt()**	External onlyOwner	Updates marketing wallet address. No Validation implemented for marketing wallet address	F
5	**setTx()**	External onlyOwner	Updates maximum transaction limit.	P
6	**setWallet()**	External onlyOwner	Updates maximum wallet holding limit.	P
7	**setSwapback()**	External onlyOwner	Updates swapback thresholds.	P
8	**rmvLimits()**	External onlyOwner	Removes transaction and wallet holding limits.	P
9	**setTax()**	External onlyOwner	Updates buy and sell tax percentages.	P
10	**removeETH()**	External	Allows tax wallet to withdraw Ether from contract.	P
11	**removeErrorToken()**	External	Allows tax wallet to withdraw tokens sent to the contract.	P
12	**addAddress()**	External onlyOwner	Adds or removes addresses from _canTx mapping. For loop will cause out of gas risk if a large size of array is passed	F
13	**exemptFee()**	External onlyOwner	Excludes an account from fees or re-enables fees for it. No input address validation.	F
14	**viewInfo()**	External View	Returns buy fee, sell fee, and limit parameters.	P
15	**_transfer()**	Internal	Handles transfers with tax deduction and swapping logic.	P
16	**min()**	Private Pure	Utility function to return the minimum of two values.	P
17	**isContract()**	Private View	Checks if an address is a contract.	P
18	**triggerSwap()**	External onlyOwner taxWallet	Triggers token swap manually for contract balance.	P
19	**swapTokensForEth()**	Private lockTheSwap	Converts tokens to Ether using Uniswap router.	P
20	**sendETHToFee()**	Private	Sends Ether from contract to the tax wallet.	F

F Fail P Pass

8.1. HIGH

1. **Issue:** Sending ETH to Arbitrary User
   Function name: sendETHToFee

```
function sendETHToFee() private {
bool success;
(success, ) = address(_taxWallet).call{value: address(this).balance}(
""
);
}
```

Description:
The function sends ETH to an arbitrary user without proper safeguards, potentially exposing funds to misuse or loss due to re-entrancy attack.
Using low-level calls without error handling can lead to silent failures or unexpected behavior.

Resolution:
Validate the _taxWallet address and ensure proper error handling for the call operation.
It is recommended to use reentrancy guard from openzeppelin nonReentrant modifier

Reference:
https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/utils/ReentrancyGuard.sol

2. **Issue:** Reentrancy in _transfer

Function name: _transfer

```
swapTokensForEth(min(contractTokenBalance,_swapbackMax));
uint256 contractETHBalance = address(this).balance;
if (contractETHBalance > 0) {
sendETHToFee();
}
```

Description:
External calls during the _transfer function can lead to reentrancy attacks, as malicious contracts could exploit these calls to manipulate state.

Resolution:
It is recommended to use reentrancy guard from openzeppelin nonReentrant modifier

Reference:
https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/utils/ReentrancyGuard.sol

3. **Issue:** Reentrancy in openTrading
   Function name: openTrading

```
 uniswapV2Router.addLiquidityETH{value: address(this).balance}(
address(this),
balanceOf(address(this)),
0,
0,
owner(),
block.timestamp
);
IERC20(uniswapV2Pair).approve(address(uniswapV2Router),
type(uint).max);
```

Description:
External calls to Uniswap during openTrading can lead to reentrancy attacks, especially if the state is not finalized beforehand.

Resolution:
It is recommended to use reentrancy guard from openzeppelin nonReentrant modifier, and apply check effect interaction pattern.

Reference:
https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/utils/ReentrancyGuard.sol

8.2. MEDIUM

1. **Issue:** Ignored Return Values in openTrading
   Function name: openTrading

```
 uniswapV2Router.addLiquidityETH{value: address(this).balance}(
address(this),
balanceOf(address(this)),
0,
0,
owner(),
block.timestamp
);
IERC20(uniswapV2Pair).approve(address(uniswapV2Router),
type(uint).max);
```

Description:
The function does not check the return values of addLiquidityETH and approve. Ignoring these can result in undetected failures, potentially causing contract functionality to break unexpectedly.

Resolution:
Check the return values of critical function calls and handle failures appropriately.

2. **Issue:** Unchecked Token Transfer

Function name: removeErrorToken

```
require(msg.sender == _taxWallet, "Only fee receiver can trigger");
IERC20(_address).transfer(_taxWallet,IERC20(_address).balanceOf(address(this)));
}
```

Description:
The function does not verify the return value of transfer, which can fail silently and leave tokens stuck.

Resolution:
Check the return value of the transfer function to ensure successful execution.

8.3. LOW

1. **Issue:** Missing Input Validation

Function name: setMkt

```
function setMkt(address payable marketingWallet) external onlyOwner {
_taxWallet = marketingWallet;

emit FeeReceiverUpdated(marketingWallet);
}
```

Description:
The function lacks validation to ensure that the provided marketingWallet address is not zero. Setting a zero address for _taxWallet could lead to funds being irretrievably lost.

Resolution:
Add a check to ensure the marketingWallet is not the zero address.

2. **Issue:** Assembly Usage
   Function name: isContract

```
function isContract(address account) private view returns (bool) {
uint256 size;
assembly {
size := extcodesize(account)
}
return size > 0;
}
```

Description:
Inline assembly can introduce vulnerabilities and is generally harder to audit.

Resolution:
Utilise OpenZeppelins 'address' library.

3. Issue: Missing Input Validation
Line: 959, 963

```
function setSwapback(
uint256 taxSwapThreshold,
uint256 maxTaxSwap
) external onlyOwner {
_swapbackMin = (totalSupply() * taxSwapThreshold) / 10000;
_swapbackMax = (totalSupply() * maxTaxSwap) / 10000;
emit SwapbackUpdated(taxSwapThreshold, maxTaxSwap);
}
```

Description:
The function does not validate that taxSwapThreshold and maxTaxSwap are within reasonable bounds. Improper values can result in unexpected or undesired behavior, such as too frequent or infrequent swaps.

Resolution:
Add checks to validate the input values are within acceptable ranges

4. **Issue:** Gas Consumption in Loop
   Lines: 992, 999

```
function addAddress(address[] calldata amount, bool status)
external
onlyOwner
{
for (uint256 i = 0; i < amount.length; i++) {
_canTx[amount[i]] = status;
}
}
```

Description:
The loop inside the function could consume excessive gas if the amount array is too large, potentially causing the transaction to fail.

Resolution:
Add a limit to the length of the amount array

5. **Issue:** Missing Input Validation
   Line: 1001, 1004

```
function exemptFee(address account, bool status) external onlyOwner {

_isExcludedFromFee[account] = status;
emit ExcludedFromFee(account, status);
}
```

Description:
The function does not validate the account address, which could allow setting a zero address or another invalid address as exempt from fees.

Resolution:
Add a check to ensure the account is valid

8.4. INFORMATIONAL

1. **Issue:** Variable Shadowing
   Function name: constructor

```
constructor() ERC20(unicode"SPEEDY", "$SPEEDY") {
uint256 _totalSupply = 1_000_000_000 * 10 ** 18;
------
}
```

Description:
The _totalSupply variable in SPEEDY shadows the inherited ERC20._totalSupply. This can cause confusion and unintended behavior during contract development or audits.

Resolution:
Rename the local _totalSupply variable to avoid shadowing.

2. **Issue:** Constant Declaration
   Line: 859 **informational**

```
uint256 private _preventSwapBefore = 23;
```

Description:
The variable _preventSwapBefore is immutable and should be declared as a constant for better readability and gas efficiency.

Resolution:
Declare the variable as a constant.

9. Conclusion

The auditing team at BLOCKSTARS TECHNOLOGY was tasked with 1 smart contract from SPEEDY for a **BASIC** audit evaluation. The team has completed audit on the SPEEDY smart contract, using static, dynamic, and symbolic analysis tools for reviewing each function within the contract.

Based on the analysis using audit tools, the SPEEDY contract has **10 vulnerabilities,** and **2 informational** findings to note.

RESULTS:	**3 HIGH**	**2 MEDIUM**	**5 LOW**	**2 INFORMATIONAL**

Risk Analysis:

Initial assessment using the standardised audit tools, had identified the contract as INSECURE due to potential attack vectors as described in the report. These vulnerabilities have been effectively mitigated through:

- Complete token distribution to distributed wallets
- Renouncement of contract ownership

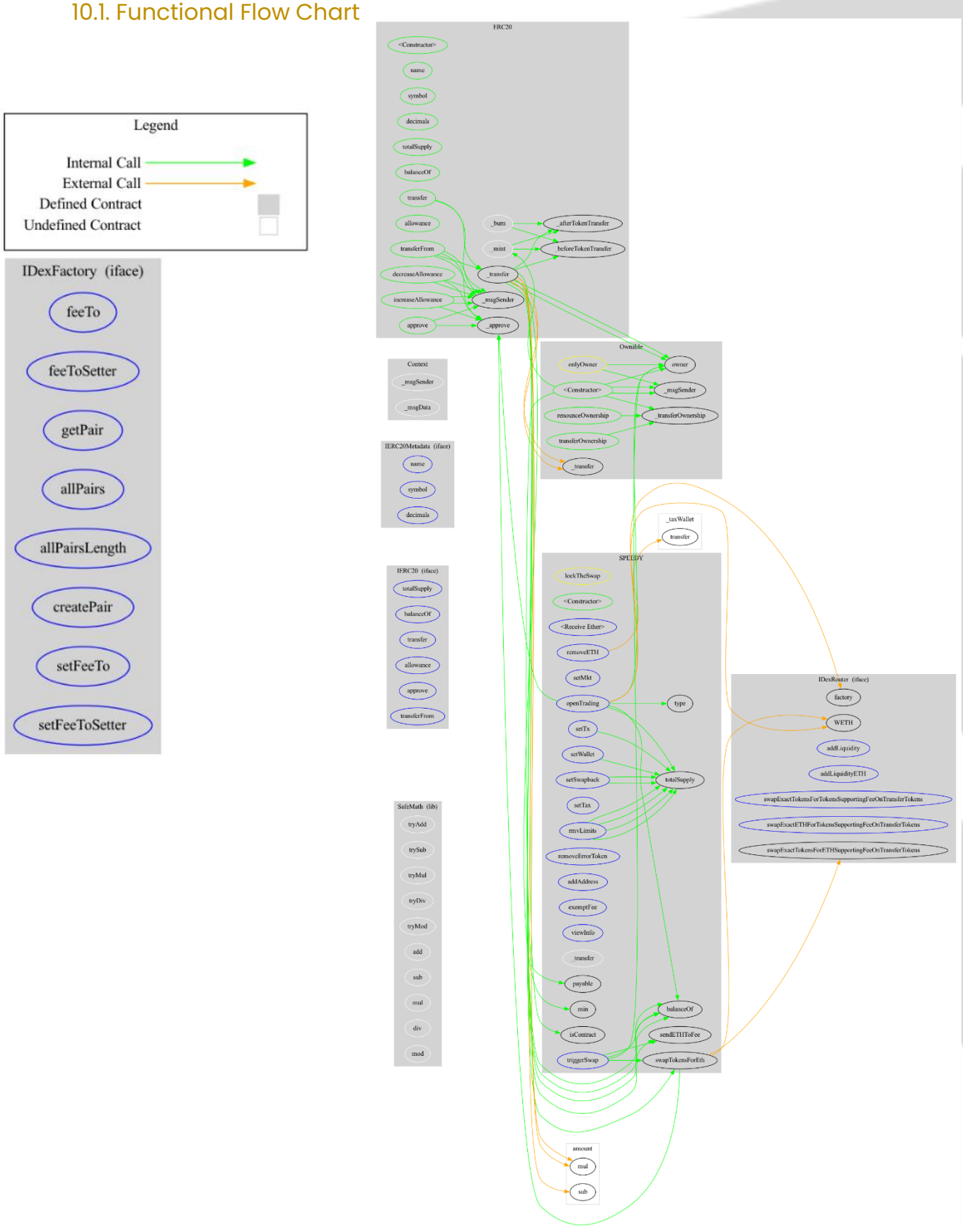
The SPEEDY contract had previously distributed all the SPEEDY tokens to other wallets before the ownership was renounced. Ownership of the contract has now been successfully renounced and as a result:

- Functions requiring owner privileges are no longer accessible.
- The potential for misuse or exploitation through owner-controlled functions has been mitigated.

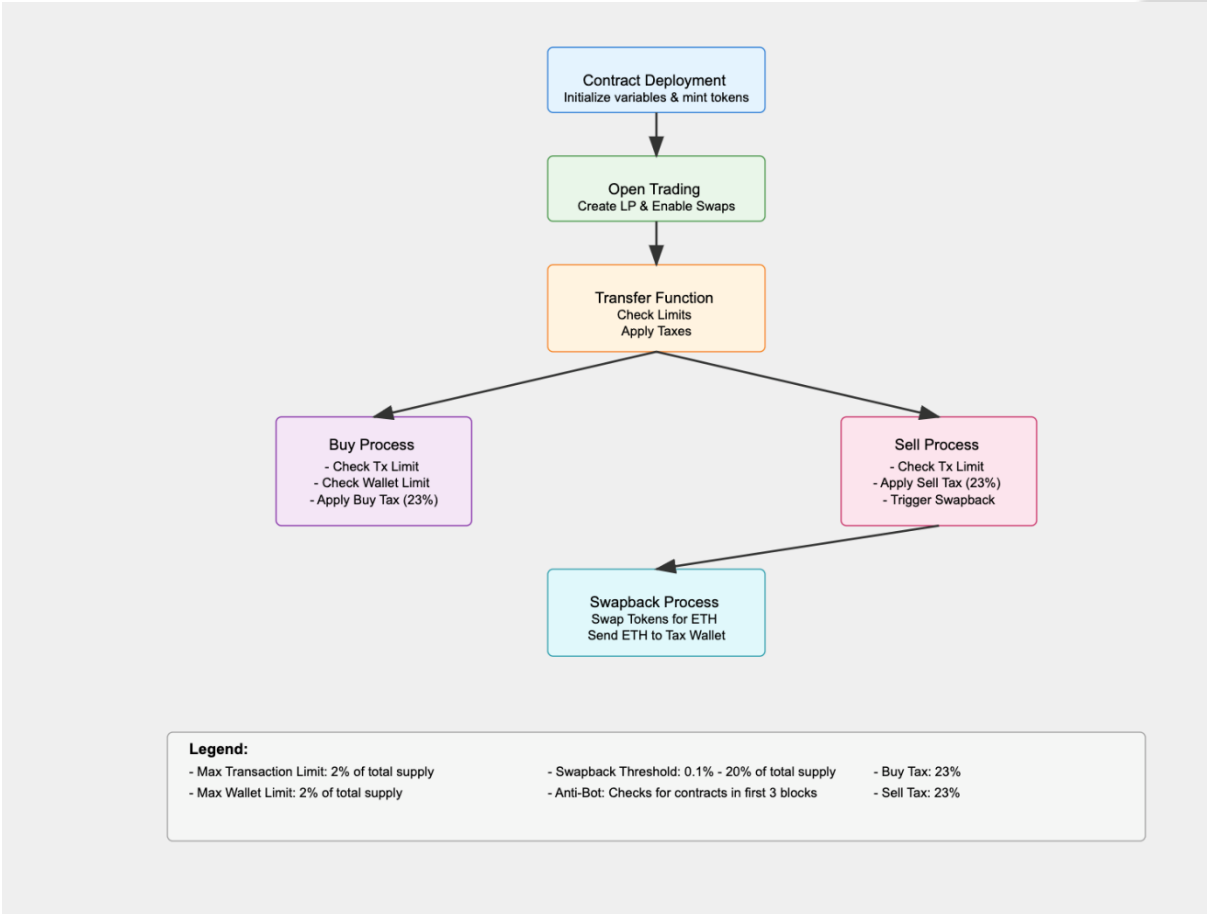
In its current state, the contract is **SECURE** from vulnerabilities that may arise due to ownership control or external calls. However, prior to renouncement, the distribution of tokens to external wallets ensured there would be no risk of tokens being locked in the contract.

10. Appendix

10.1. Functional Flow Chart



10.2. Contract Flow Diagram



10.3. Transaction hash

Creation Hash:

0xd1d187d26f2808d8a844e291b291362dda3abd3cb2273ac076f8eb8d41fda900 (0x60806040)