expelee

Building the Futuristic Blockchain Ecosystem

SECURITY AUDIT REPORT

PRESIDENT DONALD MUSK

HIGH RISK ANALYSIS

No High Risks found

Findings	Found
High Risk	0

TABLE OF CONTENTS

OVERVIEW

The Expelee team has performed a line-by-line manual analysis and automated review of the smart contract. The smart contract was analysed mainly for common smart contract vulnerabilities, exploits, and manipulation hacks. According to the smart contract audit:

Audit Result	**Passed**
KYC Verification	**-**
Audit Date	**2 June 2023**

CONTRACT DETAILS

Token Name: President Donald Musk

Symbol: PDM

Network: Binance Smart Chain

Language: Solidity

Contract Address:
0xC26b6F2E80180773716FD640e56C03eCFe7dC935

Total Supply: 1000000000000

Owner's Wallet:
0xd4C58AF81467154B46c834CF00A5E31C2e631497

Deployer's Wallet:
0xd4C58AF81467154B46c834CF00A5E31C2e631497

Testnet:
https://testnet.bscscan.com/address/0x6717077FBb340B2C802B46D500287629259568l9

OWNER PRIVILEGES

- Trading must be enabled by the owner
- The owner can set fees with limit up to 5%
- The owner can exclude accounts from fees
- The owner can change swap settings
- The owner can withdraw any token and stuck ETH (except native token) from the contract

AUDIT METHODOLOGY

Audit Details

Our comprehensive audit report provides a full overview of the audited system's architecture, smart contract codebase, and details on any vulnerabilities found within the system.

Audit Goals

The audit goal is to ensure that the project is built to protect investors and users, preventing potentially catastrophic vulnerabilities after launch , that lead to scams and rugpulls.

Code Quality

Our analysis includes both automatic tests and manual code analysis for the following aspects:

- Exploits
- Back-doors
- Vulnerability
- Accuracy
- Readability

Tools

- DE
- Open Zeppelin
- Code Analyzer
- Solidity Code
- Compiler
- Hardhat

VULNERABILITY CHECKS

Design Logic	Passed
Compiler warnings	Passed
Private user data leaks	Passed
Timestamps dependence	Passed
Integer overflow and underflow	Passed
Race conditions & reentrancy. Cross-function race conditions	Passed
Possible delays in data delivery	Passed
Oracle calls	Passed
Front Running	Passed
DoS with Revert	Passed
DoS with block gas limit	Passed
Methods execution permissions	Passed
Economy model	Passed
Impact of the exchange rate on the logic	Passed
Malicious event log	Passed
Scoping and declarations	Passed
Uninitialized storage pointers	Passed
Arithmetic accuracy	Passed
Cross-function race conditions	Passed
Safe Zepplin module	Passed

RISK CLASSIFICATION

When performing smart contract audits, our specialists look for known vulnerabilities as well as logical and acces control issues within the code. The exploitation of these issues by malicious actors may cause serious financial damage to projects that failed to get an audit in time. We categorize these vulnerabilities by the following levels:

High Risk

Issues on this level are critical to the smart contract's performance/functionality and should be fixed before moving to a live environment.

Medium Risk

Issues on this level are critical to the smart contract's performance/functionality and should be fixed before moving to a live environment.

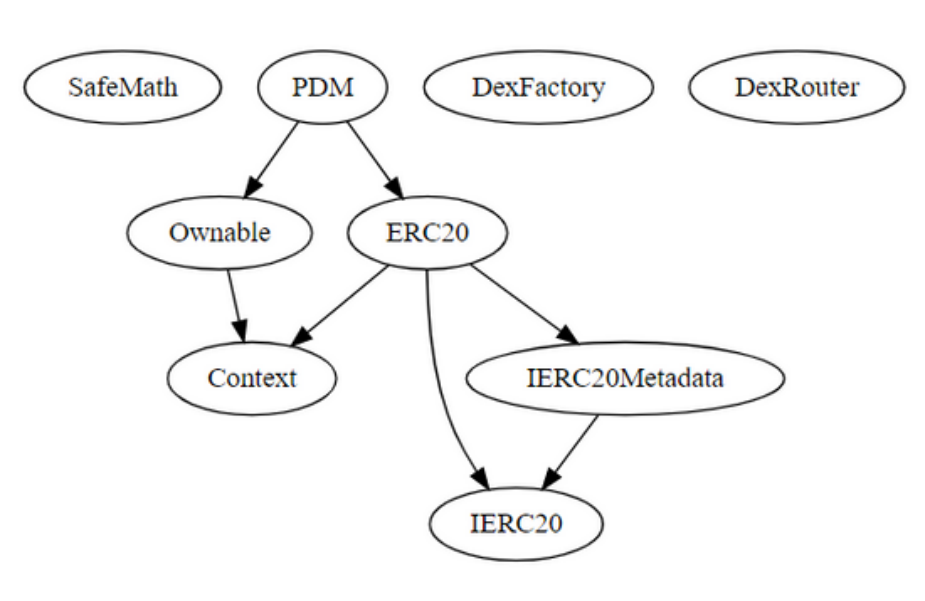
Low Risk

Issues on this level are minor details and warning that can remain unfixed.

Informational

Issues on this level are minor details and warning that can remain unfixed.

INHERITANCE TREES



expelee

FUNCTION DETAILS

PDM	Implementation	ERC20, Ownable			
└	<Constructor>	Public !	🔴	ERC20	
└	setmarketingWallet	External !	🔴	onlyOwner	
└	enableTrading	External !	🔴	onlyOwner	
└	setBuyTaxes	External !	🔴	onlyOwner	
└	setSellTaxes	External !	🔴	onlyOwner	
└	setTransferFees	External !	🔴	onlyOwner	
└	setSwapTokensAtAmount	External !	🔴	onlyOwner	
└	toggleSwapping	External !	🔴	onlyOwner	
└	setWhitelistStatus	External !	🔴	onlyOwner	
└	checkWhitelist	External !		NO !	
└	_takeTax	Internal 🔒	🔴		
└	_transfer	Internal 🔒	🔴		
└	internalSwap	Internal 🔒	🔴		
└	swapToETH	Internal 🔒	🔴		
└	withdrawStuckETH	External !	🔴	onlyOwner	
└	withdrawStuckTokens	External !	🔴	onlyOwner	
└	<Receive Ether>	External !	💶	NO !	

MANUAL REVIEW

Severity Criteria

Expelee assesses the severity of disclosed vulnerabilities according to methodology based on OWASP standarts.

Vulnerabilities are dividend into three primary risk categroies:
High
Medium
Low

High-level considerations for vulnerabilities span the following key areas when conducting assessments:

- Malicious input handling
- Escalation of privileges
- Arithmetic
- Gas use

Overall Risk Severity				
Impact	HIGH	Medium	High	Critical
	MEDIUM	Low	Medium	High
	LOW	Note	Low	Medium
		LOW	MEDIUM	HIGH
	Likelihood			

expelee

FINDINGS

Findings	Severity	Found
High Risk	High	0
Medium Risk	Medium	0
Low Risk	Low	5
Suggestion & discussion	Informational	2
Gas Optimizations	Gas Opt.	0

LOW RISK FINDING

Trading must be enabled by the owner

Severity : Low

Overview

The **enableTrading** function can be used to enable trading on the contract.

```
function enableTrading() external onlyOwner { //@audit Trade
    require(!tradingEnabled, "Trading is already enabled");
    tradingEnabled = true;
    startTradingBlock = block.number;
}
```

Recommendation

Recommended to include appropriate access control mechanisms to ensure that only authorized users can modify the smart contract's critical parameters. This can help prevent unauthorized changes to the smart contract that can potentially cause issues or put the business at risk.

LOW RISK FINDING

The owner can set fees with limit up to 5%

Severity : Low

Overview

Owner can update fees up to 5% with using **setBuyTaxes()**, **setSellTaxes()** and but **setTransferFees()**

```
function setBuyTaxes(uint256 _marketingTax, uint256 _burnTax) external onlyOwner {
    buyTaxes = Tax(_marketingTax, _burnTax);
    require(_marketingTax + _burnTax <= 5, "Can not set buy fees higher than 5%");
    emit BuyFeesUpdated(_marketingTax, _burnTax);
}

0 references | Control flow graph | a11a1682 | ftrace | funcSig
function setSellTaxes(uint256 _marketingTax, uint256 _burnTax) external onlyOwner {
    sellTaxes = Tax(_marketingTax, _burnTax);
    require(_marketingTax + _burnTax <= 5, "Can not set sell fees higher than 5%");
    emit SellFeesUpdated(_marketingTax, _burnTax);
}

0 references | Control flow graph | f639d534 | ftrace | funcSig
function setTransferFees(uint256 _marketingTax, uint256 _burnTax) external onlyOwner {
    transferTaxes = Tax(_marketingTax, _burnTax);
    require(_marketingTax + _burnTax <= 5, "Can not set transfer tax higher than 5%");
    emit TransferFeesUpdated(_marketingTax, _burnTax);
}
```

Recommendation

You should careffuly manage the private key of the owner's account. You should use powerful security mechanism that will prevent a single user from accessing the contract owner functions.

LOW RISK FINDING

The owner can exclude accounts from fees

Severity : Low

Overview

Authorizing privileged roles to exclude accounts from fees.After excluding the user from accounts, the user trades without paying a any fee and the other user sees it).

```
function setWhitelistStatus(address _wallet,bool _status) external onlyOwner {
    whitelisted[_wallet] = _status;
    emit Whitelist(_wallet, _status);
}
```

Recommendation

You should careffuly manage the private key of the owner's account. You should use powerful security mechanism that will prevent a single user from accessing the contract owner functions. That risk can be prevented by temporarily locking the contract or renouncing ownership

LOW RISK FINDING

The owner can change swap settings

Severity : Low

Overview

The owner can set new swap settings and allows the contract owner to enable or disable the automatic swapping.

```
function toggleSwapping() external onlyOwner { //@audit Owner can cl
    swapAndLiquifyEnabled = (swapAndLiquifyEnabled) ? false : true;
}
```

Recommendation

Recommended to include appropriate access control mechanisms to ensure that only authorized users can modify the smart contract's critical parameters.

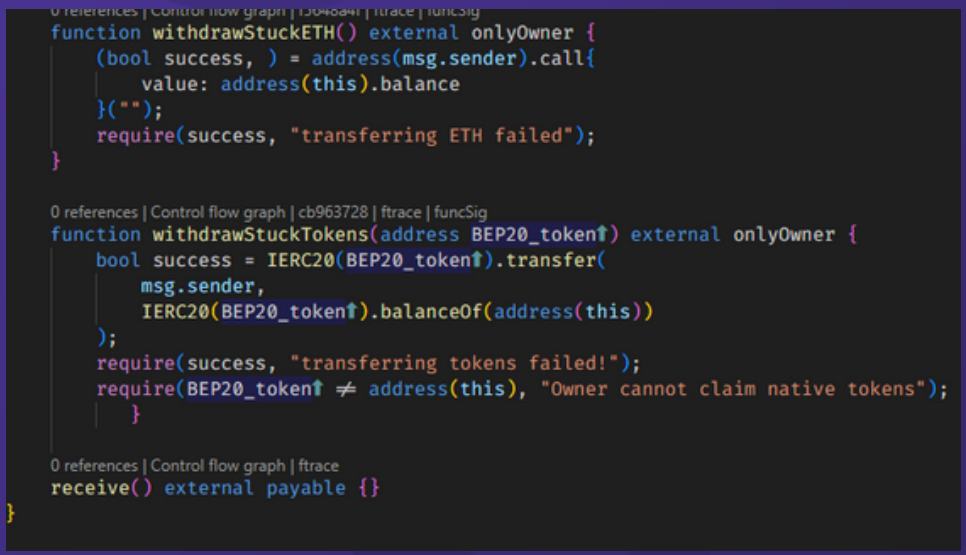
LOW RISK FINDING

The owner can withdraw any token and stuck ETH (except native token) from the contract

Severity : Low

Overview

The owner's ability to withdraw any token and any stuck ETH from the contract can be a potential security risk, especially if not implemented with appropriate safeguards. Even if there is an onlyOwner modifier. it is still important to consider the potential risks associated with allowing the owner to withdraw funds, especially if the contract holds a large amount of assets.



```
function withdrawStuckETH() external onlyOwner {
    (bool success, ) = address(msg.sender).call{
        value: address(this).balance
    }("");
    require(success, "transferring ETH failed");
}

0 references | Control flow graph | cb963728 | ftrace | funcSig
function withdrawStuckTokens(address BEP20_token) external onlyOwner {
    bool success = IERC20(BEP20_token).transfer(
        msg.sender,
        IERC20(BEP20_token).balanceOf(address(this))
    );
    require(success, "transferring tokens failed!");
    require(BEP20_token != address(this), "Owner cannot claim native tokens");
    }

0 references | Control flow graph | ftrace
receive() external payable {}
}
```

Recommendation:

Use multi-sig wallets or time-locks to ensure that the withdrawal function can only be executed after multiple owners have approved it or after a certain amount of time has elapsed. It is recommended that you still implement some safeguards to ensure that the owner cannot withdraw funds in a malicious or fraudulent manner.

INFORMATIONAL RISK FINDING

Too many digits

Severity : Informational

Overview

Literals with many digits are difficult to read and review.

```
3 references
uint256 private constant _totalSupply = 1_000_000_000_000 * 1e18;
```

Recommendation

While 1_ether looks like 1 ether, it is 10 ether. As a result, it's likely to be used incorrectly.

INFORMATIONAL RISK FINDING

Outdated versions of pragma;

Severity : Informational

Overview

Outdated versions were detected **pragma solidity 0.8.17;**

```
pragma solidity 0.8.17;
```

Recommendation

Consider using the latest version of Solidity for testing. Should lock pragmas to a specific compiler version. Besides, consider the known compiler bugs in the following references and check whether the contracts include those bugs.

ABOUT EXPELEE

Expelee is a product-based aspirational Web3 start-up. Coping up with numerous solutions for blockchain security and constructing a Web3 ecosystem from deal making platform to developer hosting open platform, while also developing our own commercial and sustainable blockchain.

www.expelee.com

expeleeofficial
expelee
Expelee
expelee
expelee_official
expelee-co

expelee

Building the Futuristic Blockchain Ecosystem

DISCLAIMER

All the content provided in this document is for general information only and should not be used as financial advice or a reason to buy any investment. Team provides no guarantess against the sale of team tokens or the removal of liquidity by the project audited in this document.

Always do your own research and project yourselves from being scammed. The Expelee team has audited this project for general information and only expresses their opinion based on similar projects and checks from popular diagnostic tools.

Under no circumstances did Expelee receive a payment to manipulate those results or change the awarding badge that we will be adding in our website. Alway do your own research and protect yourselves from scams.

This document should not be presented as a reason to buy or not buy any particular token. The Expelee team disclaims any liability for the resulting losses.