expelee

Building the Futuristic **Blockchain Ecosystem**

Audit Report

FOR

Elon Vs Twitter

OVERVIEW

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 With High Risk
KYC Verification	Not Done
Audit Date	17 Sep 2022

Why Passed?

Other than some owner functoins there is no other issues that can disable trading.

- Team Expelee

PROJECT DESCRIPTION

Elon Vs Twitter BSC Contract

Elon Vs Twitter #EVT is setting up to make history in the crypto space. Our #EVT Reward system is unlike anything that has ever been seen before. We are unique, innovative and we are here to take over. The reach and attention this trial will bring us is beyond anything you can imagine.
We will be presenting Elon Vs Twitter infront of the global media for the whole world to see.

evt2022.com

ElonVsTwitter

ElonVsTwitt

It's always good to check the social profiles of the project, before making your investment.

- Team Expelee

CONTRACT DETAILS

Contract Name
DividendToken

Token Type
ERC20

Contract Address (Verified)
0x6BAFAea58B24266D6C5BDA39698155D47b2305e9

Network
BSC

Language
Solidity

Total Supply
44,000,000,000 EVT

Decimals
18

Compiler
v0.8.13+commit.abaa5c0e

License
Default license

Contract SHA-256 Chechsum:

ba5ec51d07a4ac0e951608704431d59a02b21a4e951acc10505a8dc407c501ee

What is checksum?
This is the hash signature of contract source code, if anything even a tiny word changes in the contract this signature would betotaly different, use it to know if the team is using the same contract that we audited or not.

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
- Complier
- Hardhat

FUNCTION OVERVIEW

Can Take Back Ownership	Not Detected
Owner Change Balance	Not Detected
Blacklist	Not Detected
Modify Fees	Not Detected
Proxy	Not Detected
Whitelisted	Not Detected
Anti Whale	Detected
Trading Cooldown	Not Detected
Transfer Pausable	Not Detected
Cannot Sell All	Not Detected
Hidden Owner	Not Detected
Creator Address	0x086855a8e128c64690F10ea519b1ff45915cD0D9
Creator Balance	42082.081355045666 EVT
Owner Address	0x086855a8e128c64690f10ea519b1ff45915cd0d9
Mint	Not Detected

VULNERABILITY CHECKLIST

Design Logic	Passed
Compiler warnings.	Passed
Private user data leaks	Passed
Timestamp 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 Zeppelin module	Passed
Fallback function security	Passed

RISK CLASSIFICATION

When performing smart contract audits, our specialists look for known vulnerabilities as well as logical and access 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

Information level is to offer suggestions for improvement of efficacy or security for features with a risk free factor.

AUDIT SUMMARY

Used Tools

Slither, Echidna, etc - we used automated static-analysis tools to check contract for common solidity vulnerability & mistakes.

Manual Review:
we spent most of the audit process time reading the whole contract line by line, we even checked standard libraries & contracts (ERC20, Safemath, etc).

Overview:
Taxes will be accumulated inside the contract, a portion will be sent to owner wallet & Liqudidity pool, other parts will be swapped to doge and then sent to divident tracker.

Ownership & Owner privileges:

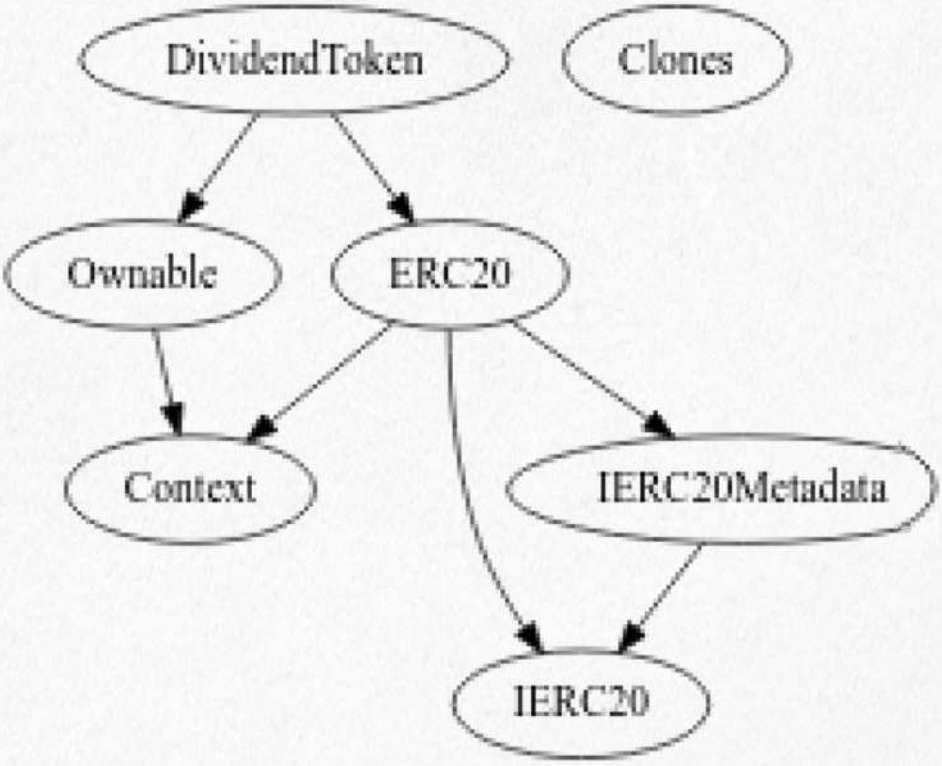
Current owner of the contract is:
0x086855a8e128c64690f10ea519b1ff45915cd0d9
owner has several privileges over contract, all them are discussed in this report

---

Contracts & Inheritance Tree:

all of below contracts are in this audit scope, most of this contracts are audited and their safety is proven

AddressUpgradeable.sol
Initializeable.sol
ContextUpgradeable.sol
IERC20MetadataUpgradeable.sol
IERC20Upgradeable.sol
IterableMapping.sol
SafeMathUint.sol
SafeMathInt.sol
SafeMathInt.sol
IUniswapV2Pair.sol
SafeMath.sol
OwnableUpgradeable.sol
ERC20Upgradeable.sol
Context.sol
IERC20Metadata.sol
IERC20.sol
DividendTokenDividendTracker.sol
IUniswapV2Router02.sol
IUniswapV2Factory.sol
Clones.sol
Ownable.sol
ERC20.sol
DividendToken.sol



MANUAL AUDIT

Severity Criteria

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

Vulnerabilities are divided into three primary risk categories: **high**, **medium**, and **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			

Findings Summary

- **High Risk Findings:** 0
- **Medium Risk Findings:** 0
- **Low Risk Findings:** 0
- **Suggestions & Discussion:** 2

High Risk Findings

Centralization - Owner is able to set **maxWallet** and **maxTransactionAmount** to zero, and hence disable trades for non-whitelisted wallets

Code:

```
function updateMaxWallet(uint256 _maxWallet) external onlyOwner {
    maxWallet = _maxWallet;
}
```

```
external
onlyOwner
{
    maxTransactionAmount = _maxTransactionAmount;
}
```

Medium - at **addLiquidity**, lpWallet is accumulating LP tokens, a Private Key leak can damageLiquidity Pool

Medium - owner is able to set up to 30% tax on buy and sell (60% in total buy + sell)

Suggestions & Discussion

- at this block of code:

```
pancakeCaller.swapExactTokensForTokensSupportingFeeOnTransferTokens(
    uniswapV2Router,
    tokenAmount,
    0, // accept any amount of BaseToken
    path,
    block.timestamp
);
```

swap is done using **pancakeCaller**, its not clear what is usage of this interface and why we are notusing **uniswapV2Router**

- it might be better to change **updateUniswapV2Pair** name to something like **addUniswapV2Pair**
- at **updateUniswapV2Router** function, new router may not match the prevoius one and pairs maynot be created using **create** method, however this is very unlikely.
- **emit** an event from this functions:
  updateMaxWallet - **updateMaxTransactionAmount**

Gas Optimizations:

- set this functions as external:
  updateDividendTracker
  updateUniswapV2Router
  excludeFromFees
  setAutomatedMarketMakerPair
- at **setMarketingWallet** emit **MarketingWalletUpdated** before changing **_marketingWalletAddress** to avoid decalring a new memory variable

- at **updateLPWallet** emit **LiquidityWalletUpdated** before changing **lpWallet** to avoid decalring anew memory variable
- at **updateLiquidityFee** and **updateMarketingFee** emit function arguments to avoid reading fromstorage
- at _transfer, second **if(!swapping)** is redundant
- mabe change name to **swapTokensForDoge**? cuz reward token is doge coin

```
if (!swapping) {
            if (!isExcludedFromMaxTransactionAmount[from]) {
                require(
```

- at **swapAndSendToFee** set **rewardToken** to a memory variable to avoid reading from storagemultiple times

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 guarantees against the sale of team tokens or the removal of liquidity by the project audited in this document.

Always Do your own research and protect 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. Always 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.