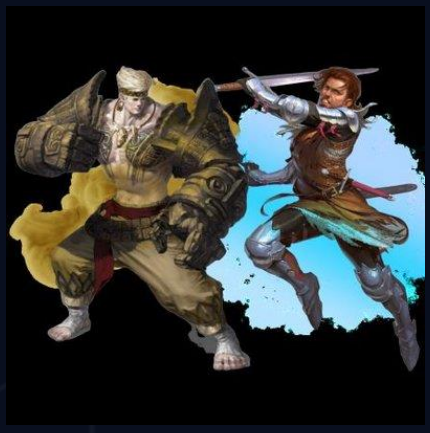
expelee

A Secure Place For Web3

SMART CONTRACT AUDIT OF

WORMSGAMES



Contract Address

0x72a2B17B1994Eb4693743a36F484304FCaD7e3d1

Audit Summary

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 MEDIUM RISK)**

Ownership: **NOT RENOUNCED**

KYC Verification: Not done till date of audit

Audit Date: 28/06/2022

Audit Team: **EXPELEE**

Be aware that smart contracts deployed on the blockchain aren't resistant to internal exploit, external vulnerability, or hack. For a detailed understanding of risk severity, source code vulnerability, functional hack, and audit disclaimer, kindly refer to the audit.

DISCLAMER

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.

Contract Review

Contract Name	**WormsGame**
Compiler Version	**v0.8.4+commit.c7e474f2**
Optimization	**Yes with 200 runs**
License	**MIT license**
Explorer	**https://bscscan.com/address/0x72a2B17B1994Eb4693743a36F484304FCaD7e3d1#code**
Symbol	**WRG**
Decimals	**9**
Total Supply	**1000,000,000,000**
Domain	**https://brutalworms.com/**

Project Review

Token Name: WORMS GAME

Web Site: https://brutalworms.com/

Twitter: https://twitter.com/madwormsrace

Telegram: https://t.me/BrutalWorms

Contract Address:
0x72a2B17B1994Eb4693743a36F484304FCaD7e3d1

Platform: Binance Smart Chain

Token Type: BEP 20

Language: SOLIDITY

Audit Methodology

The scope of this report is to audit the smart contract source code. We have scanned the contract and reviewed the project for common vulnerabilities, exploits, hacks, and back-doors. Below is the list of commonly known smart contract vulnerabilities, exploits, and hacks:

Category

Smart Contract Vulnerabilities	- Unhandled Exceptions - Transaction Order Dependency - Integer Overflow - Unrestricted Action - Incorrect Inheritance Order - Typographical Errors - Requirement Violation
Source Code Review	- Gas Limit and Loops - Deployment Consistency - Repository Consistency - Data Consistency - Token Supply Manipulation
Functional Assessment	- Operations Trail & Event Generation - Assets Manipulation - Liquidity Access

Vulnerability Checklist

№	Description.	Result
1	Compiler warnings.	Passed
2	Race conditions and Re-entrancy. Cross-function raceconditions.	Passed
3	Possible delays in data delivery.	Passed
4	Oracle calls.	Passed
5	Front running.	Passed
6	Timestamp dependence.	Passed
7	Integer Overflow and Underflow.	Passed
8	DoS with Revert.	Passed
9	DoS with block gas limit.	Passed
10	Methods execution permissions.	Passed
11	Economy model.	Passed
12	The impact of the exchange rate on the logic.	Passed
13	Private user data leaks.	Passed
14	Malicious Event log.	Passed
15	Scoping and Declarations.	Passed
16	Uninitialized storage pointers.	Passed
17	Arithmetic accuracy.	Passed
18	Design Logic.	Passed
19	Cross-function race conditions.	Passed
20	Safe Zeppelin module.	Passed
21	Fallback function security.	Passed

Manual Audit

Low-Risk

4 low-risk code issues found

Medium-Risk

0 medium-risk code issues found

High-Risk

0 high-risk code issues found

Low-Risk

1) Functions that send Ether to arbitrary destinations

```
function addLiquidity(uint256 tokenAmount, uint256 ethAmount) private {
        // approve token transfer to cover all possible scenarios
        _approve(address(this), address(uniswapV2Router), tokenAmount);

        // add the liquidity
        uniswapV2Router.addLiquidityETH{value: ethAmount}(
            address(this),
            tokenAmount,
            0, // slippage is unavoidable
            0, // slippage is unavoidable
            owner(),
            block.timestamp
        );
    }
```

Recommendation

Ensure that an arbitrary user cannot withdraw unauthorized funds.

2)Reentrancy vulnerabilities

Detection of the reentrancy bug. Do not report reentrancies that don't involve Ether (see reentrancy-no-eth)

```
function _transfer(
    address from,
    address to,
    uint256 amount
) private {
    require(from != address(0), "ERC20: transfer from the zero address");
    require(to != address(0), "ERC20: transfer to the zero address");
    require(amount > 0, "Transfer amount must be greater than zero");
    uint256 contractTokenBalance = balanceOf(address(this));

    bool overMinTokenBalance = contractTokenBalance >=
        numTokensSellToAddToLiquidity;
    if (
        overMinTokenBalance &&
        !inSwapAndLiquify &&
        from != uniswapV2Pair &&
        swapAndLiquifyEnabled
    ) {
        contractTokenBalance = numTokensSellToAddToLiquidity;
        //add liquidity
        swapAndLiquify(contractTokenBalance);
    }
    bool takeFee = true;
    if (_isExcludedFromFee[from] || _isExcludedFromFee[to]) {
        takeFee = false;
    }
    _tokenTransfer(from, to, amount, takeFee);
}
```

Recommendation

Apply the check-effects-interactions pattern.

3) Tautology or contradiction

```
constructor(
        string memory name_,
        string memory symbol_,
        uint256 totalSupply_,
        address router_,
        address charityAddress_,
        uint16 taxFeeBps_,
        uint16 liquidityFeeBps_,
        uint16 charityFeeBps_,
        address serviceFeeReceiver_,
        uint256 serviceFee_
    ) payable {
        require(taxFeeBps_ >= 0, "Invalid tax fee");
        require(liquidityFeeBps_ >= 0, "Invalid liquidity fee");
        require(charityFeeBps_ >= 0, "Invalid charity fee");
        if (charityAddress_ == address(0)) {
            require(
                charityFeeBps_ == 0,
                "Cant set both charity address to address 0 and charity percent more than 0"
            );
        }
        require(
            taxFeeBps_ + liquidityFeeBps_ + charityFeeBps_ <= 10**4 / 4,
            "Total fee is over 25%"
        );
```

Recommendation

Fix the incorrect comparison by changing the value type or the comparison.

4)Local variable shadowing

Detection of shadowing using local variables.

```
function allowance(address owner, address spender)
        public
        view
        override
        returns (uint256)
    {
        return _allowances[owner][spender];
    }
```

Recommendation

Rename the local variables that shadow another component.

Audit Summary

```
Compiled with solc
Number of lines: 1642 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 10 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 21
Number of informational issues: 85
Number of low issues: 4
Number of medium issues: 0
Number of high issues: 0
ERCs: ERC20
```

Name	# functions	ERCS	ERC20 info	Complex code	Features
SafeMath	13			No	
Address	11			No	Send ETH Delegatecall Assembly
IUniswapV2Router02	24			No	Receive ETH
IUniswapV2Factory	8			No	
LiquidityGeneratorToken	64	ERC20	No Minting Approve Race Cond.	No	Receive ETH Send ETH

Manual Audit (Contract Function)

```
contract LiquidityGeneratorToken is IERC20, Ownable, BaseToken {
    using SafeMath for uint256;
    using Address for address;

    uint256 public constant VERSION = 1;

    mapping(address => uint256) private _rOwned;
    mapping(address => uint256) private _tOwned;
    mapping(address => mapping(address => uint256)) private _allowances;

    mapping(address => bool) private _isExcludedFromFee;
    mapping(address => bool) private _isExcluded;
    address[] private _excluded;

    uint256 private constant MAX = ~uint256(0);
    uint256 private _tTotal;
    uint256 private _rTotal;
    uint256 private _tFeeTotal;

    string private _name;
    string private _symbol;
    uint8 private _decimals;

    uint256 public _taxFee;
    uint256 private _previousTaxFee = _taxFee;

    uint256 public _liquidityFee;
    uint256 private _previousLiquidityFee = _liquidityFee;

    uint256 public _charityFee;
    uint256 private _previousCharityFee = _charityFee;

    IUniswapV2Router02 public uniswapV2Router;
    address public uniswapV2Pair;
    address public _charityAddress;

    bool inSwapAndLiquify;
    bool public swapAndLiquifyEnabled;

    uint256 private numTokensSellToAddToLiquidity;

    event MinTokensBeforeSwapUpdated(uint256 minTokensBeforeSwap);
    event SwapAndLiquifyEnabledUpdated(bool enabled);
```

Important Points To Consider

✓ The owner cannot mint tokens after Initial

✓ The owner cannot stop Trading.

✓ Verified contract source

✗ Token is not sellable (not a honeypot) at this time

✗ Ownership NOT renounced or source does contain an ownership functionality

✗ Source does contain a fee modifier

✗ Owner/creator wallet contains less than 5% of circulating token supply (6.73%)

✓ All other holders possess less than 5% of circulating token supply

About Expelee

Expelee is a community driven organisation dedicated to fostering an anti-rug movement. We're here to keep investment safe from fraudsters. We've encountered several rug pulls and know how it feels to be duped, which is why we don't want anybody else to go through the same experience. We are here to raise awareness through our services so that the future of cryptocurrency can be rug-free.

The auditing process focuses to the following considerations with collaboration of an expert team:

- Functionality test of the Smart Contract to determine if proper logic has been followed throughout the whole process.
- Manually detailed examination of the code line by line by experts.
- Live test by multiple clients using Test net.
- Analysing failure preparations to check how the Smart
- Contract performs in case of any bugs and vulnerabilities.
- Checking whether all the libraries used in the code are on the latest version.
- Analysing the security of the on-chain data.

Social Media

/Expelee

/ExpeleeOfficial

/expelee-co