FOR

SEEDCOURT

WEBSITE

https://seedcourt.com

TELEGRAM

t.me/SeedCourt_SEC

SMART CONTRACT AUDIT

Twitter:
twitter.com/vital_block

Telegram:
t.me/vital_Block

Disclaimer

Vital Block Solidity reports are not, nor should be considered, an "endorsement" or "disapproval" of any project or team. These reports are not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Vital Block to perform a security review.

Vital Black Solidity Reports do not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analysed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

Vital Block Solidity Reports should not be used in any way to make decisions around investment or involvement with any project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort. Vital Block Solidity Reports represent an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. Vital Block Solidity's position is that each company and individual are responsible for their own due diligence and continuous security. Vital Block Solidity's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyse

What is a Vital Block Audit report?

•A document describing in detail an in-depth analysis of a particular piece(s) of source code provided to Vital Block Solidity by a Client.

•An organized collection of testing results, analysis and inferences made about the structure, implementation, and overall best practices of a particular piece of source code.

•Representation that a Client of Vital Block Solidity has indeed completed a round of auditing with the intention to increase the quality of the company/ product's IT infrastructure and or source code.

Overview

Project Summary

Project Name	**SEEDCOURT**
Description	**It takes a community to build a project.**
Platform	**Binance Mainnet**
Mainnet Contracts:	0x32217Eb6414382c420c6908f5b31D3c2cb2d6531 ***SEEDCOURT (SEC)***
Files:	**SEEDCOURT.sol**

Audit Summary

Delivery Date	**June 11 2022**
Method of Audit	**Static Analysis**
Timeline	**Story Points 100**

Vulnerability Summary

Total Issues Found	**3**
Total Issues Resolved	**3**
Total Critical	**0**
Total High	**1**
Total Medium	**2**
Total Low	**0**
Total Informational	**2**

Our Audit Methodology

- **STEP 1**
  A manual line-by-line code review to ensure the logic behind each function is safe and secured against common attack vectors.

- **STEP 2**
  Simulation of hundreds of thousands of Smart Contract Interactions on a test and Mainnet blockchain using a combination of automated test tools and manual testing to determine if any security vulnerabilities exist.

- **STEP 3**
  Consultation with the project team on the audit report pre-publication to implement recommendations and resolve any outstanding issues.

Grading

The following grading structure is used to assess the level of vulnerability found within all Smart Contracts:

THREAT LEVEL	DEFINITION
Critical	Severe vulnerabilities which compromise the entire protocol and could result in immediate data manipulation or asset loss.
High	Significant vulnerabilities which compromise the functioning of the smart contracts leading to possible data manipulation or asset loss.
Medium	Vulnerabilities which if not fixed within in a set timescale could compromise the functioning of the smart contracts leading to possible data manipulation or asset loss.
Low	Low level vulnerabilities which may or may not have an impact on the optimal performance of the Smart contract.
Informational	Issues related to coding best practice which do not have any impact on the functionality of the Smart Contracts

Description

SEEDCOURT is A community of like-minded members who use the power of a decentralized compatible ecosystem to fund Blockchain projects, out of the sinking dip.

Trading fees are 8% on buys and a 8% on sells. The fees are distributed as follows:

Buy Trading Fees 8.0% - LP | 5% - Marketing & Development | 1% - Liquidity | 2% - Team

Sell Trading Fees 8.0% - LP | 5% - Marketing & Development | 1% - Liquidity | 2% - Team

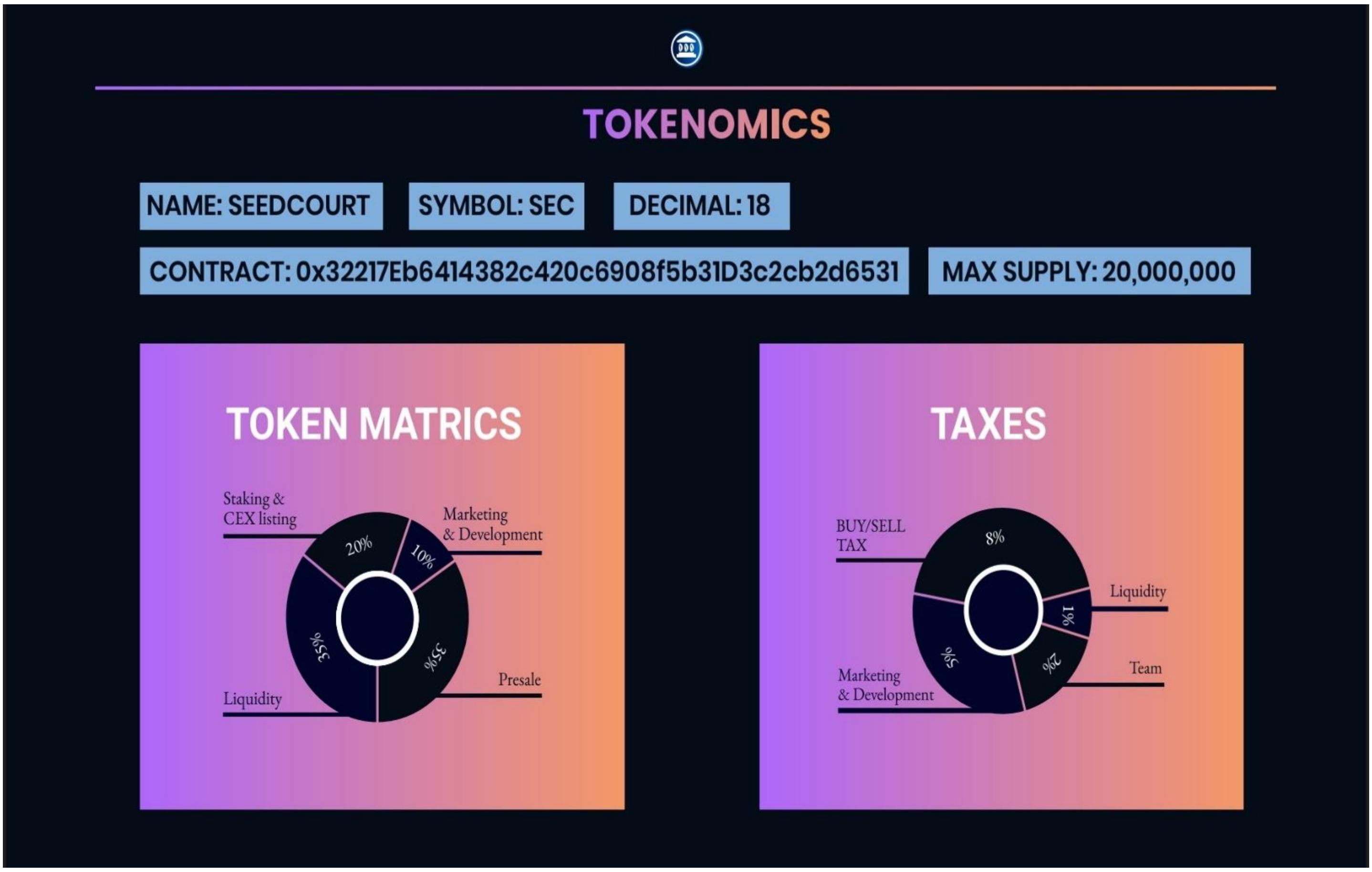
Initial supply is 20million tokens, distributed as follows:

7 Million - Presale

7 Million - Initial Liquidity

4 Million – Staking & CEX Listing

2 Million – Marketing & Development



File: Seedcourt.sol
Contract Address:
0x32217Eb6414382c420c6908f5b31D3c2cb2d6531

Vulnerability 1: Owner can change fees up to 50%

Threat level: High

Description:

The owner has the permission to change 'autoLiquidityReceiver' to any account. Now on transferring, if
'shouldAddLiquidity' returns true, all the balance of 'autoLiquidityReceiver' is converted to add liquidity.

Recommendation:

Manage a separate variable to store LP amount. Instead of sending autoLP tax token to 'autoLiquidityReceiver', send to contract address and use the contract balance only to add liquidity.

```
function setIsTxLimitExempt(address holder, bool exempt) external authorized {
    isTxLimitExempt[holder] = exempt;
}

function setIsTimelockExempt(address holder, bool exempt) external authorized {
    isTimelockExempt[holder] = exempt;
}

function setFees(uint256 _liquidityFee, uint256 _reflectionFee, uint256 _marketingFee, ui
    liquidityFee = _liquidityFee;
    reflectionFee = _reflectionFee;
    marketingFee = _marketingFee;
    teamfee = _teamfee;
    burnFee = _burnFee;
    totalFee = _liquidityFee.add(_reflectionFee).add(_marketingFee).add(_teamfee).add(_bu
    feeDenominator = _feeDenominator;
    require(totalFee < feeDenominator/2, "Fees cannot be more than 50%");
}

function setFeeReceivers(address _autoLiquidityReceiver, address _marketingFeeReceiver, a
    autoLiquidityReceiver = _autoLiquidityReceiver;
    marketingFeeReceiver = _marketingFeeReceiver;
    teamfeeReceiver = _teamfeeReceiver;
    burnFeeReceiver = _burnFeeReceiver;
}
```

Resolution status: Fully resolved before deployment to main net.

Vulnerability 1: Total Supply cant exceed MAX_SUPPLY

Threat level: Medium

Description:

The contract checks whether total Supply is Equal To MAX_SUPPLY. And there is A check to return Supply function when total supply Tends to exceeds MAX_SUPPLY.

```
    uint256 _totalSupply = 20 * 10**6 * 10**_decimals;

    uint256 public _maxTxAmount = _totalSupply.mul(2).div(100);
    uint256 public _maxWalletToken = _totalSupply.mul(4).div(100);

    mapping (address => uint256) _balances;
    mapping (address => mapping (address => uint256)) _allowances;

    bool public blacklistMode = true;
    mapping (address => bool) public isBlacklisted;

    mapping (address => bool) isFeeExempt;
    mapping (address => bool) isTxLimitExempt;
    mapping (address => bool) isTimelockExempt;
    mapping (address => bool) isDividendExempt;

    uint256 public liquidityFee    = 1;
    uint256 public reflectionFee   = 0;
    uint256 public marketingFee    = 5;
    uint256 public teamfee    = 2;
    uint256 public burnFee         = 0;
    uint256 private totalFee        = marketingFee + reflectionFee + liquidityFee +
    uint256 public feeDenominator  = 100;
```

Resolution status: Fully resolved before deployment to main net.

Vulnerability 2: Gas Optimisation Issue

Threat level: Informational

Vulnerability 2: Gas optimisation

Threat level: Informational

Description: Does not seem like a honeypot.
This can always change! Do your own due diligence.
INFO! There is no liquidity with BNB. Honeypot added liquidity for test. Results with non-BNB pair may differ. If the token is not live yet, results may be different once the token is live. It is common for tokens to have 0% taxes before launching on DEX!

SEEDCOURT (SEC)
Max TX: 20000000 SEC (~? BNB)
Gas used for Buying: 352,037
Gas used for Selling: 178,997
Buy Tax: 8%
Sell Tax: 7.9%

Recommendation:

The contract can be modified so that it can be done via a single call to save gas.

```
function process(uint256 gas) external override onlyToken {
    uint256 shareholderCount = shareholders.length;

    if(shareholderCount == 0) { return; }

    uint256 gasUsed = 0;
    uint256 gasLeft = gasleft();

    uint256 iterations = 0;

    while(gasUsed < gas && iterations < shareholderCount) {
        if(currentIndex >= shareholderCount){
            currentIndex = 0;
        }

        if(shouldDistribute(shareholders[currentIndex])){
            distributeDividend(shareholders[currentIndex]);
        }

        gasUsed = gasUsed.add(gasLeft.sub(gasleft()));
        gasLeft = gasleft();
        currentIndex++;
        iterations++;
    }
```

No	Issue Description	Checking Status.
1	Compiler Errors.	Passed
2	Oracle Cells.	Passed
3	Race Conditions and Reentrancy. Cross-function race condition.	Passed
4	Possible delay in data delivery.	Passed
5	Front Runing.	Passed
6	TimeStamp Dependence.	Passed
7	Integal Overflow.	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 The Declarations.	Passed
16	Uninitialized Storage Pointers	Passed
17	Arithmetic Accuracy.	Passed
18	Design Logic.	Passed
19	Cross-function Race conditions.	Passed
20	Safe Zeppelin Model	Passed
21	Fallback Function Security.	Passed

Audit Result

PASSED

Conclusion

During the Vital block Audit process, the Seedcourt contract was analysed by manual review and automated testing. All issues identified pre-launch were resolved before deployment to main net. By submitting the contract for audit pre-launch, the team have displayed a strong commitment to security.

Whilst there are no obvious vulnerabilities or security risks identified within the main net contract, it is beyond the scope of this Vital Block Audit to comment upon any risks associated with tokenomics, adoption or platform longevity. Before placing funds in any defi protocol Vital Block encourages potential investors to exercise due diligence and research all projects thoroughly to assess plans for ongoing development and financial sustainability.

Appendix

Finding Categories

Gas Optimization

Gas Optimization findings refer to exhibits that do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Mathematical Operations

Mathematical Operation exhibits entail findings that relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

Logical Issue findings are exhibits that detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectely on certain edge cases that may result in avulnerability.

Data Flow

Data Flow findings describe faults in the way data is handled at rest and in memory, such as the result of a structassignment operation affecting an in-memory struct rather than an instorage one.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of private or delete.

Coding Style

Coding Style findings usually do not affect the generated byte-code and comment on how to make the codebase more legible and as a result easily maintainable.

Appendix

Inconsistency
Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a constructor assignment imposing different require statements on the input variables than a setter function.

Magic Numbers
Magic Number findings refer to numeric literals that are expressed in the codebase in their raw format and should otherwise be specified as constant contract variables aiding in their legibility and maintainability.

Compiler Error
Compiler Error findings refer to an error in the structure of the code that renders it impossible to compile using the specified version of the project.

Dead Code
Code that otherwise does not affect the functionality of the codebase and can be safely omitted.

VitalBlock

WWW.VITALBLOCK.XYZ

Decentralized Smart Contract Auditing Firm.