Coinsult

Advanced Manual Smart Contract Audit

Project: LuckyStep

Website: https://luckystep.app/

Low-Risk	Medium-Risk	High-Risk
6 low-risk code issues found	0 medium-risk code issues found	0 high-risk code issues found

Contract Address

0xDf51B4366bB8Da2Ab27A27aEF3A692a0754B141E

Disclaimer: Coinsult is not responsible for any financial losses. Nothing in this contract audit is financial advice, please do your own research.

Disclaimer

Coinsult is not responsible if a project turns out to be a scam, rug-pull or honeypot. We only provide a detailed analysis for your own research.

Coinsult is not responsible for any financial losses. Nothing in this contract audit is financial advice, please do your own research.

The information provided in this audit is for informational purposes only and should not be considered investment advice. Coinsult does not endorse, recommend, support or suggest to invest in any project.

Coinsult can not be held responsible for when a project turns out to be a rug-pull, honeypot or scam.

Tokenomics

Rank	Address	Quantity (Token)	Percentage
1	0x6ce919d242125d74ecc53e598669744bf6879e01	60,000,000,000	60.0000%
2	0x0038601f55ba787700b904986efca3e502cd26e2	25,000,000,000	25.0000%
3	0x38f5f73d0b307c8d546d1989ad80090c269f89ce	10,000,000,000	10.0000%
4	0x356c1f5d2afc3eeb2a769a9c937332ea8dc6fc25	5,000,000,000	5.0000%

Source Code

Coinsult was comissioned by LuckyStep to perform an audit based on the following smart contract:

https://bscscan.com/address/0xdf51b4366bb8da2ab27a27aef3a692a0754b141e#code

Contract contains imported modules.

Manual Code Review

In this audit report we will highlight all these issues:

🟢 Low-Risk

6 low-risk code
issues found

🟡 Medium-Risk

0 medium-risk code
issues found

🔴 High-Risk

0 high-risk code
issues found

The detailed report continues on the next page...

🟢 **Low-Risk:** Could be fixed, will not bring problems.

Contract contains Reentrancy vulnerabilities

Additional information: This combination increases risk of malicious intent. While it may be justified by some complex mechanics (e.g. rebase, reflections, buyback).

More information: Slither

```
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 &gt; 0, "Transfer amount must be greater than zero");

    if(from != owner() &amp;&amp; to != owner())
        require(amount &lt;= _maxTxAmount, &quot;Transfer amount exceeds the maxTxAmount.&quot;);

    if(from != owner() &amp;&amp; to != owner() &amp;&amp; to != address(1) &amp;&amp; to != uniswap
        require(balanceOf(to) + amount = _maxTxAmount)
    {
        contractTokenBalance = _maxTxAmount;
    }

    bool overMinTokenBalance = contractTokenBalance &gt;= numTokensSellToAddToLiquidity;
    if (
        overMinTokenBalance &amp;&amp;
        !inSwapAndLiquify &amp;&amp;
```

Recommendation

Apply the check-effects-interactions pattern.

Exploit scenario

```
function withdrawBalance(){
    // send userBalance[msg.sender] Ether to msg.sender
    // if mgs.sender is a contract, it will call its fallback function
    if( ! (msg.sender.call.value(userBalance[msg.sender])() ) ){
        throw;
    }
    userBalance[msg.sender] = 0;
}
```

Bob uses the re-entrancy bug to call withdrawBalance two times, and withdraw more than its initial deposit to the contract.

🟢 **Low-Risk:** Could be fixed, will not bring problems.

Divide before multiply

Solidity integer division might truncate. As a result, performing multiplication before division can sometimes avoid loss of precision.

```
uint256 devBNB = newBalance.div(_devFee + _liquidityFee).mul(_devFee);
        uint256 liquidityBNB = newBalance - devBNB;

        TransferBnbToExternalAddress(_devWalletAddress, newBalance.div(_devFee + _liquidityFee).mul(
```

Recommendation

Consider ordering multiplication before division.

Exploit scenario

```
contract A {
    function f(uint n) public {
        coins = (oldSupply / n) * interest;
    }
}
```

If n is greater than `oldSupply`, `coins` will be zero. For example, with `oldSupply = 5; n = 10, interest = 2`, coins will be zero. If `(oldSupply * interest / n)` was used, `coins` would have been 1. In general, it's usually a good idea to re-arrange arithmetic to perform multiplication before division, unless the limit of a smaller type makes this dangerous.

🟢 **Low-Risk:** Could be fixed, will not bring problems.

Missing events arithmetic

Detect missing events for critical arithmetic parameters.

```
function excludeFromFee(address account) public onlyOwner {
    _isExcludedFromFee[account] = true;
}
```

Recommendation

Emit an event for critical parameter changes.

Exploit scenario

```
contract C {

  modifier onlyAdmin {
    if (msg.sender != owner) throw;
    _;
  }

  function updateOwner(address newOwner) onlyAdmin external {
    owner = newOwner;
  }
}
```

`updateOwner()` has no event, so it is difficult to track off-chain changes in the buy price.

🟢 **Low-Risk:** Could be fixed, will not bring problems.

Conformance to Solidity naming conventions

Allow _ at the beginning of the mixed_case match for private variables and unused parameters.

```
Parameter LUCKYSTEP.setSwapAndLiquifyEnabled(bool)._enabled (LUCKYSTEP.sol#298) is not in mixedCase
Function LUCKYSTEP.TransferBnbToExternalAddress(address,uint256) (LUCKYSTEP.sol#366-368) is not in m
Parameter LUCKYSTEP.calculateTaxFee(uint256)._amount (LUCKYSTEP.sol#370) is not in mixedCase
Parameter LUCKYSTEP.calculateDevFee(uint256)._amount (LUCKYSTEP.sol#374) is not in mixedCase
Parameter LUCKYSTEP.calculateLiquidityFee(uint256)._amount (LUCKYSTEP.sol#378) is not in mixedCase
Variable LUCKYSTEP._taxFee (LUCKYSTEP.sol#93) is not in mixedCase
Variable LUCKYSTEP._devFee (LUCKYSTEP.sol#96) is not in mixedCase
Variable LUCKYSTEP._liquidityFee (LUCKYSTEP.sol#99) is not in mixedCase
Variable LUCKYSTEP._maxTxPercent (LUCKYSTEP.sol#108) is not in mixedCase
Variable LUCKYSTEP._maxWalletPercent (LUCKYSTEP.sol#109) is not in mixedCase
Variable LUCKYSTEP._maxWalletDecimal (LUCKYSTEP.sol#110) is not in mixedCase
Variable LUCKYSTEP._maxTxAmount (LUCKYSTEP.sol#112) is not in mixedCase
Variable LUCKYSTEP._maxWalletAmount (LUCKYSTEP.sol#115) is not in mixedCase
Function IUniswapV2Pair.DOMAIN_SEPARATOR() (uniswap.sol#36) is not in mixedCase
Function IUniswapV2Pair.PERMIT_TYPEHASH() (uniswap.sol#37) is not in mixedCase
Function IUniswapV2Pair.MINIMUM_LIQUIDITY() (uniswap.sol#54) is not in mixedCase
Function IUniswapV2Router01.WETH() (uniswap.sol#74) is not in mixedCase
```

Recommendation

Follow the Solidity naming convention.

Rule exceptions

- Allow constant variable name/symbol/decimals to be lowercase (`ERC20`).
- Allow _ at the beginning of the `mixed_case` match for private variables and unused parameters.

🟢 **Low-Risk:** Could be fixed, will not bring problems.

Redundant Statements

Detect the usage of redundant statements that have no effect.

```
function _msgData() internal view virtual returns (bytes memory) {
    this; // silence state mutability warning without generating bytecode - see https://github.com/e
     return msg.data;
}
```

Recommendation

Remove redundant statements if they congest code but offer no value.

Exploit scenario

```
contract RedundantStatementsContract {

    constructor() public {
        uint; // Elementary Type Name
        bool; // Elementary Type Name
        RedundantStatementsContract; // Identifier
    }

    function test() public returns (uint) {
        uint; // Elementary Type Name
        assert; // Identifier
        test; // Identifier
        return 777;
    }
}
```

Each commented line references types/identifiers, but performs no action with them, so no code will be generated for such statements and they can be removed.

🟢 **Low-Risk:** Could be fixed, will not bring problems.

Costly operations inside a loop

Costly operations inside a loop might waste gas, so optimizations are justified.

```
function includeInReward(address account) external onlyOwner {
    require(_isExcluded[account], "Account is already included");
    for (uint256 i = 0; i &lt; _excluded.length; i++) {
        if (_excluded[i] == account) {
            _excluded[i] = _excluded[_excluded.length - 1];
            _tOwned[account] = 0;
            _isExcluded[account] = false;
            _excluded.pop();
            break;
        }
    }
}
```

Recommendation

Use a local variable to hold the loop computation result.

Exploit scenario

```
contract CostlyOperationsInLoop{

    function bad() external{
        for (uint i=0; i < loop_count; i++){
            state_variable++;
        }
    }

    function good() external{
      uint local_variable = state_variable;
      for (uint i=0; i < loop_count; i++){
        local_variable++;
      }
      state_variable = local_variable;
    }
}
```

Incrementing `state_variable` in a loop incurs a lot of gas because of expensive `SSTORE`s, which might lead to an `out-of-gas`.

Owner privileges

- Owner cannot set fees higher than 25%
- Owner cannot pause trading
- Owner can change max transaction amount
- Owner can exclude from fees
- ⚠️ Owner can set max wallet percentage

Extra notes by the team

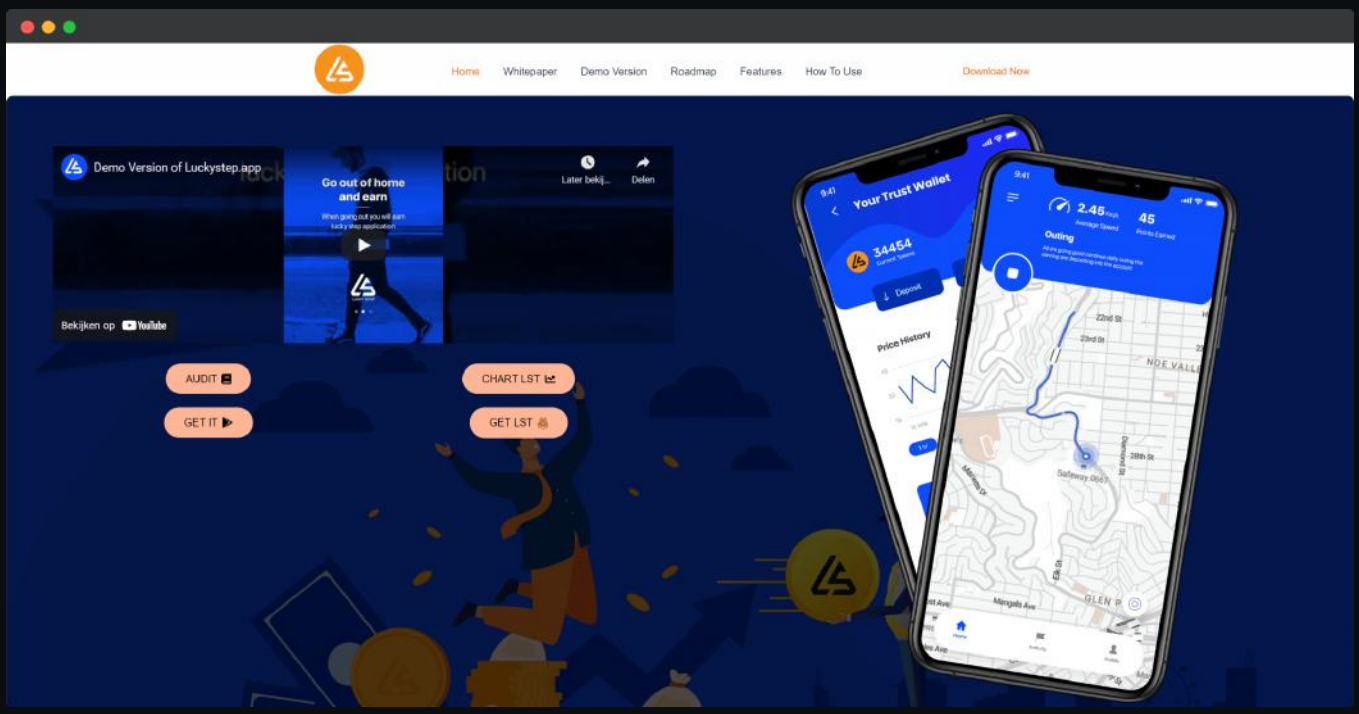
No notes

Contract Snapshot

```
contract LUCKYSTEP is Context, IBEP20, Ownable {
using SafeMath for uint256;
using Address for address;

mapping (address =&gt; uint256) private _rOwned;
mapping (address =&gt; uint256) private _tOwned;
mapping (address =&gt; mapping (address =&gt; uint256)) private _allowances;

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

address private _devWalletAddress = 0x38F5f73d0b307C8D546D1989AD80090C269F89Ce;

uint256 private constant MAX = ~uint256(0);
uint256 private _tTotal = 1 * 10**11 * 10**9;
uint256 private _rTotal = (MAX - (MAX % _tTotal));
uint256 private _tFeeTotal;

string private _name = "LUCKYSTEP";
string private _symbol = "LST";
uint8 private _decimals = 9;

uint256 public _taxFee = 1;
uint256 private _previousTaxFee = _taxFee;

uint256 public _devFee = 2;
uint256 private _previousDevFee = _devFee;

uint256 public _liquidityFee = 2;
uint256 private _previousLiquidityFee = _liquidityFee;

IUniswapV2Router02 public immutable uniswapV2Router;
address public immutable uniswapV2Pair;

bool inSwapAndLiquify;
bool public swapAndLiquifyEnabled = true;
```

Website Review

Coinsult checks the website completely manually and looks for visual, technical and textual errors. We also look at the security, speed and accessibility of the website. In short, a complete check to see if the website meets the current standard of the web development industry.



- Mobile Friendly
- Does not contain jQuery errors
- SSL Secured
- No major spelling errors

Project Overview

Not KYC verified by Coinsult