Coinsult

Advanced Manual Smart Contract Audit

Project: Scuba Inu

Website: http://www.scubainubsc.com

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

Contract Address

0x8A5e6Fc2d406EAA3086fb45fBFb3B685F950f992

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	0xaf9b4fdc57f9ddb56823bdc8106ba334d2ff834a	100,000,000,000	100.0000%

Source Code

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

https://bscscan.com/address/0x8a5e6fc2d406eaa3086fb45fbfb3b685f950f992#code

Manual Code Review

In this audit report we will highlight all these issues:

Low-Risk

7 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), "BEP20: transfer from the zero address");
        require(to != address(0), "BEP20: 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 (launchedAt == 0 &amp;&amp; from == owner() &amp;&amp; automatedMarketMakerPairs[to]) {
    launchedAt = block.number;
}

        if (!automatedMarketMakerPairs[to] &amp;&amp; !_isExcludedFromMaxWallet[to]) {
    require(balanceOf(to).add(amount) = _maxTxAmount)
        {
            contractTokenBalance = _maxTxAmount;
        }
```

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.

Avoid relying on block.timestamp

block.timestamp can be manipulated by miners.

```
function addLiquidity(uint256 tokenAmount, uint256 BNBAmount) private {
    // approve token transfer to cover all possible scenarios
    _approve(address(this), address(pancakeSwapV2Router), tokenAmount);

    pancakeSwapV2Router.addLiquidityETH{value: BNBAmount}(
        address(this),
        tokenAmount,
        0, // slippage is unavoidable
        0, // slippage is unavoidable
        owner(),
        block.timestamp
    );
}
```

Recommendation

Do not use `block.timestamp`, now or `blockhash` as a source of randomness

Exploit scenario

```
contract Game {

    uint reward_determining_number;

    function guessing() external{
      reward_determining_number = uint256(block.blockhash(10000)) % 10;
    }
}
```

Eve is a miner. Eve calls `guessing` and re-orders the block containing the transaction. As a result, Eve wins the game.

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

Too many digits

Literals with many digits are difficult to read and review.

```
uint256 private constant _tTotal = 100000 * 10**6 * 10**18;
```

Recommendation

Use: Ether suffix, Time suffix, or The scientific notation

Exploit scenario

```
contract MyContract{
    uint 1_ether = 10000000000000000000;
}
```

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

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

No zero address validation for some functions

Detect missing zero address validation.

```
function setCharityWallet(address payable charityWallet) external onlyOwner() {
     _charityWallet = charityWallet;
}
```

Recommendation

Check that the new address is not zero.

Exploit scenario

```
contract C {

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

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

Bob calls `updateOwner` without specifying the `newOwner`, soBob loses ownership of the contract.

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

Missing events arithmetic

Detect missing events for critical arithmetic parameters.

```
function setTaxFeePercent(uint256 taxFee) external onlyOwner() {
     _taxFee = taxFee;
}
```

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.

Redundant Statements

Detect the usage of redundant statements that have no effect.

```
function _msgData() internal view virtual returns (bytes calldata) {
    this; // silence state mutability warning without generating bytecode - see https://github.com/et
    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 not excluded");
    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 pause trading
- 🟡 Owner can change max transaction amount
- 🟡 Owner can set fees higher than 25%
- 🟡 Owner can exclude from fees
- ⚠️ Owner can exclude address from MaxWallet function
- ⚠️ Owner can set max wallet amount

Extra notes by the team

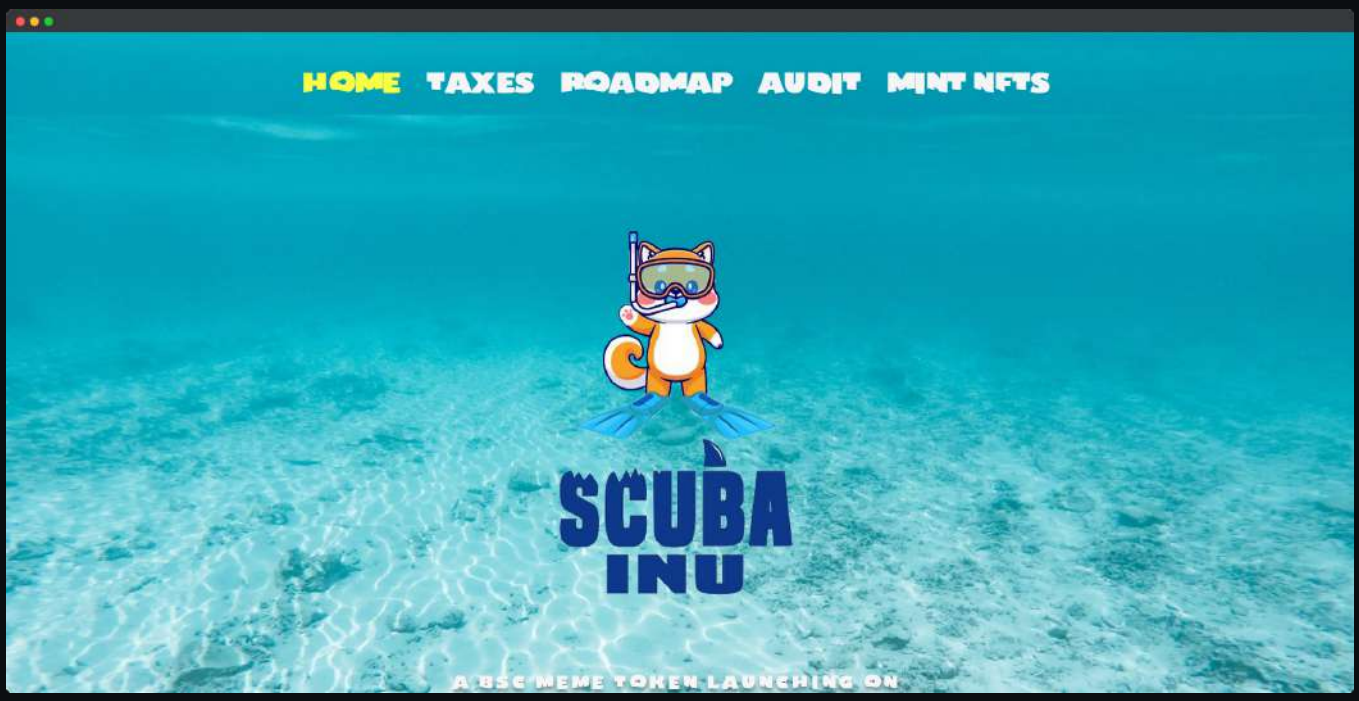
No notes

Contract Snapshot

```
contract SCUBA 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) public automatedMarketMakerPairs;
mapping(address =&gt; bool) public _isExcludedFromMaxWallet;
```

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

Scuba Inu

Audited by Coinsult.net

Date: 29 June 2022

Advanced Manual Smart Contract Audit