Coinsult

Advanced Manual Smart Contract Audit

Project: Aptos Chain

Website: -

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

0x72c863C31cF59abd4820b1D04026DFFe08486B98

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	0xecbc0c45df202b01fa180e5c71820d24e9c43eb1	5,500,000	55.0000%
2	0x267c69f35a95da03db3e90d900e6d3aae9bd295f	4,500,000	45.0000%

Source Code

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

https://bscscan.com/address/0x72c863C31cF59abd4820b1D04026DFFe08486B98#code

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 sender, address recipient, uint256 amount) private returns (bool) {

    require(sender != address(0), "ERC20: transfer from the zero address");
    require(recipient != address(0), "ERC20: transfer to the zero address");
    require(!_bAddress[sender] &amp;amp;&amp;amp; !_bAddress[recipient], "ERC20: transfer from | to bAddress

    if(inSwapAndLiquify)
    {
        return _basicTransfer(sender, recipient, amount);
    }
    else
    {
        if(!isTxLimitExempt[sender] &amp;amp;&amp;amp; !isTxLimitExempt[recipient]) {
            require(amount = _minimumTokensBeforeSwap;

        if (overMinimumTokenBalance &amp;amp;&amp;amp; !inSwapAndLiquify &amp;amp;&amp;amp; !isMarketPair[sender] &amp
        {
            if(swapAndLiquifyByLimitOnly)
                contractTokenBalance = _minimumTokensBeforeSwap;
            swapAndLiquify(contractTokenBalance);
        }
```

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.

No zero address validation for some functions

Detect missing zero address validation.

```
function setMarketingWalletAddress(address newAddress) external onlyOwner() {
    marketingWalletAddress = payable(newAddress);
}
```

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.

Functions that send Ether to arbitrary destinations

Unprotected call to a function sending Ether to an arbitrary address.

```
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
        marketingWalletAddress,
        block.timestamp
    );
}
```

Recommendation

Ensure that an arbitrary user cannot withdraw unauthorized funds.

Exploit scenario

```
contract ArbitrarySend{
    address destination;
    function setDestination(){
        destination = msg.sender;
    }

    function withdraw() public{
        destination.transfer(this.balance);
    }
}
```

Bob calls `setDestination` and `withdraw`. As a result he withdraws the contract's balance.

🟢 **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.

```
_minimumTokensBeforeSwap = supply.div(10000) * 10**_decimals;
```

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 updateKillBlockNum(uint num) public onlyOwner {
    killBlockNum = num;
}
```

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 memory) {
    this;
    // silence state mutability warning without generating bytecode - see https://github.com/ethereu
    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.

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 disable wallet limits
- ⚠️ Owner can exclude addresses from max transaction amount
- ⚠️ Owner can set max wallet balance

Extra notes by the team

No notes

Contract Snapshot

```
contract APTOS is Context, IERC20, Ownable {

using SafeMath for uint256;
using Address for address;

string private _name;
string private _symbol;
uint8 private _decimals;
address payable public marketingWalletAddress;
address payable public teamWalletAddress;
address public immutable deadAddress = 0x000000000000000000000000000000000000dEaD;
```

Project Overview

Not KYC verified by Coinsult