Coinsult

Advanced Manual Smart Contract Audit

Project: Redpill

Website: https://redpillplay.com/

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

Contract Address

0x80D3A1226FE1a185834e8CB44C66d5F7859a03d1

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	PancakeSwap V2: RPP 10	461,916.3788	26.0740%
2	Magic Forest: Deployer	192,268.8	10.8531%
3	0x28655727c08bf984b67b995b01d3fbf941913b41	106,188	5.9940%
4	0x7035d620aaa0fd44de684ecdb7e37b7858533a25	33,362.284	1.8832%
5	0x6b38ef3519828725a2225227af11596693fa5a1d	31,443.4656	1.7749%

Source Code

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

https://bscscan.com/address/0x80d3a1226fe1a185834e8cb44c66d5f7859a03d1#code

Manual Code Review

In this audit report we will highlight all these issues:

🟢 Low-Risk

10 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 _transferFrom(
    address sender,
    address recipient,
    uint256 amount
) internal returns (bool) {
    if (inSwap || isFullExempt[sender] || isFullExempt[recipient]) {
        return _basicTransfer(sender, recipient, amount);
    }

    if (!authorizations[sender] &amp;amp;&amp;amp; !authorizations[recipient]) {
        require(tradingOpen, "Trading not open yet");
    }

    uint256 rAmount = amount.mul(rate);

    if (
        !authorizations[sender] &amp;amp;&amp;amp;
        recipient != address(this) &amp;amp;&amp;amp;
        recipient != address(DEAD) &amp;amp;&amp;amp;
        recipient != pair &amp;amp;&amp;amp;
        recipient != marketingFeeReceiver &amp;amp;&amp;amp;
        recipient != devFeeReceiver &amp;amp;&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.

Avoid relying on block.timestamp

block.timestamp can be manipulated by miners.

```
shareholderClaims[shareholder] = 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 public maxRoomRent = 8000000000;
```

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 setFeeReceivers(
    address _autoLiquidityReceiver,
    address _marketingFeeReceiver,
    address _devFeeReceiver
) external authorized {
    autoLiquidityReceiver = _autoLiquidityReceiver;
    marketingFeeReceiver = _marketingFeeReceiver;
    devFeeReceiver = _devFeeReceiver;
}
```

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 swapBack() internal swapping {
    uint256 dynamicLiquidityFee = isOverLiquified(
        targetLiquidity,
        targetLiquidityDenominator
    )
        ? 0
        : liquidityFee;
    uint256 tokensToSell = swapThreshold.div(rate);

    uint256 amountToLiquify = tokensToSell
        .div(totalFee)
        .mul(dynamicLiquidityFee)
        .div(2);
    uint256 amountToSwap = tokensToSell.sub(amountToLiquify);

    address[] memory path = new address[](2);
    path[0] = address(this);
    path[1] = WBNB;

    uint256 balanceBefore = address(this).balance;

    router.swapExactTokensForETHSupportingFeeOnTransferTokens(
```

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.

Unchecked transfer

The return value of an external transfer/transferFrom call is not checked.

```
function distributeDividend(address shareholder) internal {
    if (shares[shareholder].amount == 0) {
        return;
    }

    uint256 amount = getUnpaidEarnings(shareholder);
    if (amount &gt; 0) {
        totalDistributed = totalDistributed.add(amount);
        RWRD.transfer(shareholder, amount);
        shareholderClaims[shareholder] = block.timestamp;
        shares[shareholder].totalRealised = shares[shareholder]
            .totalRealised
            .add(amount);
        shares[shareholder].totalExcluded = getCumulativeDividends(
            shares[shareholder].amount
        );
    }
}
```

Recommendation

Use `SafeERC20`, or ensure that the transfer/transferFrom return value is checked.

Exploit scenario

```
contract Token {
    function transferFrom(address _from, address _to, uint256 _value) public returns (bool success);
}
contract MyBank{
    mapping(address => uint) balances;
    Token token;
    function deposit(uint amount) public{
        token.transferFrom(msg.sender, address(this), amount);
        balances[msg.sender] += amount;
    }
}
```

Several tokens do not revert in case of failure and return false. If one of these tokens is used in `MyBank`, `deposit` will not revert if the transfer fails, and an attacker can call `deposit` for free..

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

Write after write

Variables that are written but never read and written again.

```
try distributor.deposit{value: amountBNBReflection}() {} catch {}
        (bool tmpSuccess, ) = payable(marketingFeeReceiver).call{
            value: amountBNBMarketing,
            gas: 30000
        }("");
        (tmpSuccess, ) = payable(devFeeReceiver).call{
            value: amountBNBDev,
            gas: 30000
        }("");

        // only to supress warning msg
        tmpSuccess = false;
```

Recommendation

Fix or remove the writes.

Exploit scenario

```
```solidity
contract Buggy{
    function my_func() external initializer{
        // ...
        a = b;
        a = c;
        // ..
    }
}
```

`a` is first asigned to `b`, and then to `c`. As a result the first write does nothing.
```

🟢 **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 feeAmount = rAmount.div(feeDenominator * 100).mul(totalFee).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 setDistributionCriteria(
    uint256 _minPeriod,
    uint256 _minDistribution
) external override onlyToken {
    minPeriod = _minPeriod;
    minDistribution = _minDistribution;
}
```

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.

Costly operations inside a loop

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

```
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 &lt; gas &amp;&amp; iterations = shareholderCount) {
            currentIndex = 0;
        }

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

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 can change max transaction amount
- 🟡 Owner can exclude from fees
- 🟡 Owner can pause the contract
- ⚠️ Owner can exclude addresses from dividend
- ⚠️ Owner can set a sell multiplier which will be applied on sell fees

Extra notes by the team

No notes

Contract Snapshot

```
contract RedPill is IBEP20, Auth {
using SafeMath for uint256;
using SafeMathInt for int256;

address WBNB = 0xbb4CdB9CBd36B01bD1cBaEBF2De08d9173bc095c;
address DEAD = 0x000000000000000000000000000000000000dEaD;
address ZERO = 0x0000000000000000000000000000000000000000;

string constant _name = "RedPill";
string constant _symbol = "RPP";
uint8 constant _decimals = 4;

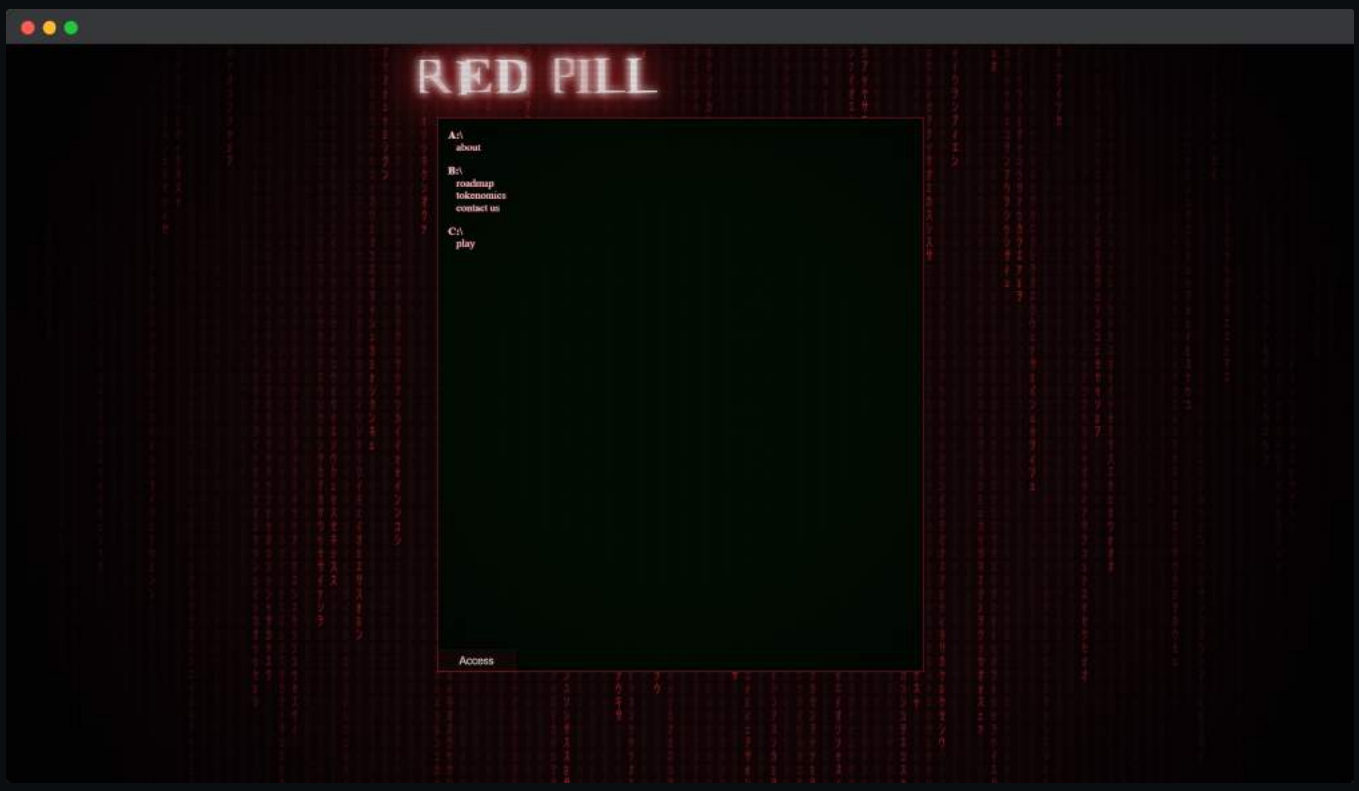
//mapping (address =&gt; uint256) _balances;
mapping(address =&gt; uint256) _rBalance;
mapping(address =&gt; mapping(address =&gt; uint256)) _allowances;

mapping(address =&gt; bool) public isFeeExempt;
mapping(address =&gt; bool) public isTxLimitExempt;
mapping(address =&gt; bool) public isTimelockExempt;
mapping(address =&gt; bool) public isDividendExempt;
mapping(address =&gt; bool) public isFullExempt;

uint256 public liquidityFee = 0;
uint256 public reflectionFee = 0;
uint256 public marketingFee = 4;
uint256 public devFee = 2;
uint256 public totalFee =
    marketingFee + reflectionFee + liquidityFee + devFee;
uint256 public feeDenominator = 100;
```

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