Coinsult

Advanced Manual Smart Contract Audit

Project: Patriot Coin
Website: https://patriotft.com

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

Contract address
0xA4b3a505A091f0633e45D1D5e3157dd276e9739c

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

Not deployed yet

Source code

Coinsult was commissioned by Patriot Coin to perform an audit based on the following smart contract:

TESTNET CODE
https://testnet.bscscan.com/address/0x712Bc50149Fa899b1E073926936e5367B3d9863c#code

Manual Code Review

🟢 Low-risk

4 low-risk code issues found.
Could be fixed, will not bring problems.

- Contract contains Reentrancy vulnerabilities:
  _transferFrom(address,address,uint256)
  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) {

        require(!blacklist[sender] && !blacklist[recipient], "in_blacklist");

        if (inSwap) {
            return _basicTransfer(sender, recipient, amount);
        }
        if (shouldRebase()) {
           rebase();
        }

        if (shouldAddLiquidity()) {
            addLiquidity();
        }

        if (shouldSwapBack()) {
            swapBack();
        }

        uint256 gonAmount = amount.mul(_gonsPerFragment);
        _gonBalances[sender] = _gonBalances[sender].sub(gonAmount);
        uint256 gonAmountReceived = shouldTakeFee(sender, recipient)
            ? takeFee(sender, recipient, gonAmount)
            : gonAmount;
        _gonBalances[recipient] = _gonBalances[recipient].add(
            gonAmountReceived
        );

        if(!isDividendExempt[sender]){ try distributor.setShare(sender, balanceOf(sender)) {} catch {} }
        if(!isDividendExempt[recipient]){ try distributor.setShare(recipient, balanceOf(recipient)) {} catch {}
}

        try distributor.process(distributorGas) {} catch {}

        emit Transfer(
            sender,
            recipient,
            gonAmountReceived.div(_gonsPerFragment)
        );
        return true;
    }
```

- Function which sends eth to arbitrary destination

  Ensure that an arbitrary user cannot withdraw unauthorized funds. More information: Slither

```
        (bool success, ) = payable(treasuryReceiver).call{
            value: amountETHToTreasuryAndSIF.mul(treasuryFee).div(
                treasuryFee.add(Test9422DivFee)
            ),
            gas: 30000
        }("");
```

- Block.timestamp can be manipulated by miners.

  Avoid relying on block.timestamp.

  More information: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

```
    function shouldRebase() internal view returns (bool) {
        return
            _autoRebase &&
            (_totalSupply < MAX_SUPPLY) &&
            msg.sender != pair  &&
            !inSwap &&
            block.timestamp >= (_lastRebasedTime + 15 minutes);
    }
```

- Missing zero address validation.

  Check that the new address is not zero.

```
    function setFeeReceivers(
        address _autoLiquidityReceiver,
        address _treasuryReceiver,
        address _powerBurn
    ) external onlyOwner {
        autoLiquidityReceiver = _autoLiquidityReceiver;
        treasuryReceiver = _treasuryReceiver;
        powerBurn = _powerBurn;
    }
```

🟡 Medium-risk

0 medium-risk code issues found.
Should be fixed, could bring problems.

🔴 High-risk

0 high-risk code issues found
Must be fixed, and will bring problems.

Extra notes by the team

🟢 Owner can not change the fees

🟡 The ownership is not renounced.

🟡 Owner can blacklist contract addresses

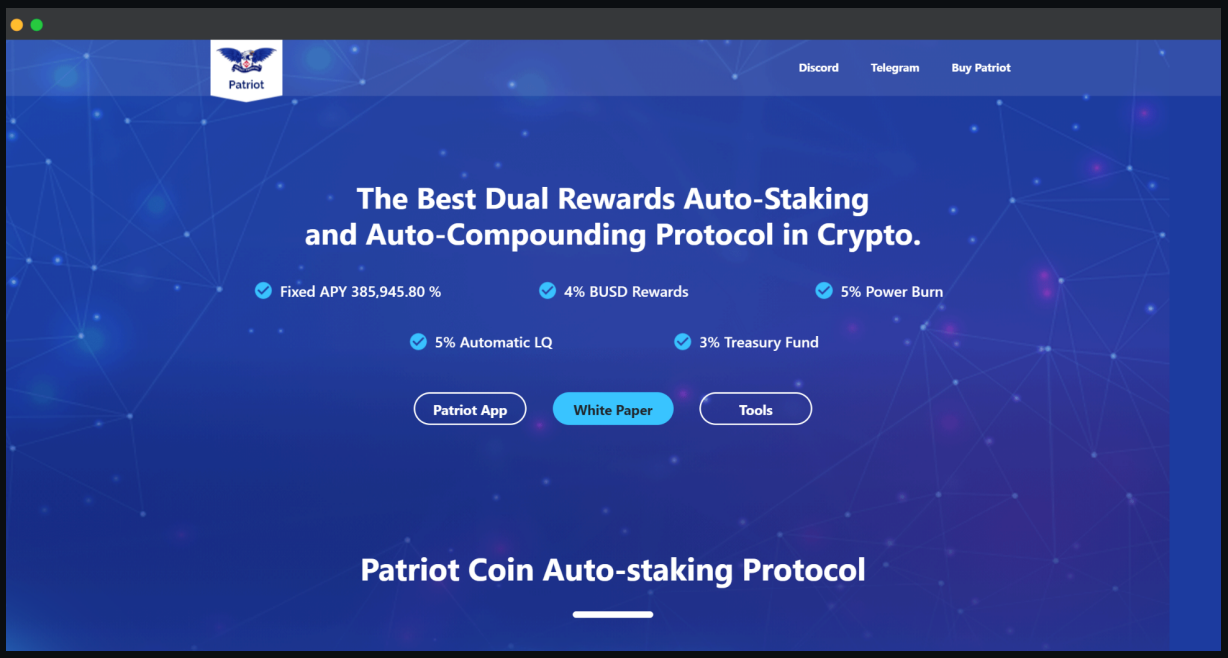
🟡 Contract uses rebase

```
    function rebase() internal {

        if ( inSwap ) return;
        uint256 rebaseRate;
        uint256 deltaTimeFromInit = block.timestamp - _initRebaseStartTime;
        uint256 deltaTime = block.timestamp - _lastRebasedTime;
        uint256 times = deltaTime.div(15 minutes);
        uint256 epoch = times.mul(15);

            if (deltaTimeFromInit < (365 days)) {
        rebaseRate = 2355;
            } else if (deltaTimeFromInit >= (7 * 365 days)) {
        rebaseRate = 2;
            } else if (deltaTimeFromInit >= ((15 * 365 days) / 10)) {
        rebaseRate = 14;
            } else if (deltaTimeFromInit >= (365 days)) {
        rebaseRate = 211;
        }

        for (uint256 i = 0; i < times; i++) {
            _totalSupply = _totalSupply
                .mul((10**RATE_DECIMALS).add(rebaseRate))
                .div(10**RATE_DECIMALS);
        }

        _gonsPerFragment = TOTAL_GONS.div(_totalSupply);
        _lastRebasedTime = _lastRebasedTime.add(times.mul(15 minutes));

        pairContract.sync();

        emit LogRebase(epoch, _totalSupply);
    }
```

Contract Snapshot

```
contract Test9422 is ERC20Detailed, Ownable {

    using SafeMath for uint256;
    using SafeMathInt for int256;

    event LogRebase(uint256 indexed epoch, uint256 totalSupply);

    IPancakeSwapPair public pairContract;
    mapping(address => bool) _isFeeExempt;

    modifier validRecipient(address to) {
        require(to != address(0x0));
        _;
    }

    uint256 public constant DECIMALS = 7;
    uint256 public constant MAX_UINT256 = ~uint256(0);
    uint8 public constant RATE_DECIMALS = 7;

    uint256 private constant INITIAL_FRAGMENTS_SUPPLY =
        70 * 10**5 * 10**DECIMALS;

    uint256 public liquidityFee = 50;
    uint256 public treasuryFee = 30;
    uint256 public Test9422DivFee = 40;
    uint256 public sellFee = 20;
    uint256 public powerBurnFee = 50;
    uint256 public totalFee =
        liquidityFee.add(treasuryFee).add(Test9422DivFee).add(
            powerBurnFee
        );
    uint256 public feeDenominator = 1000;

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

    address public autoLiquidityReceiver;
    address public treasuryReceiver;

    DividendDistributor distributor;
```

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
- Contains no jQuery errors
- SSL Secured
- No major spelling errors

Loading speed: 89%

Rug-pull Review

Based on the available information analyzed by us, we come to the following conclusions:

🟡 No locked Liquidity - No liquidity yet

🟡 Large unlocked wallets - Tokens not yet distributed

🟡 No doxxed Team yet

Honeypot Review

Based on the available information analyzed by us, we come to the following conclusions:

🟢 Ability to sell

🟢 Owner is not able to pause the contract

🟡 Router can be changed in the contract

Note: Please check the disclaimer above and note, the audit makes no statements or warranties on business model, investment attractiveness or code sustainability. The report is provided for the only contract mentioned in the report and does not include any other potential contracts deployed by the project owner.