Coinsult

Advanced Manual Smart Contract Audit

Project: PowerLabs

Website: https://powerlabs.network/

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

0x2a5f446f8b6A819BC3f1715156a4ddaC3b7eaf50

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	Powerlabs: Deployer	1,000,000,000	100.0000%

Source Code

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

https://bscscan.com/address/0x2a5f446f8b6a819bc3f1715156a4ddac3b7eaf50#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
    ) internal override {
        require(from != address(0), "ERC20: transfer from the zero address");
        require(to != address(0), "ERC20: transfer to the zero address");

        if(amount == 0) {
            super._transfer(from, to, 0);
            return;
        }

        bool isBuyFromLp = automatedMarketMakerPairs[from];
        bool isSelltoLp = automatedMarketMakerPairs[to];
        bool _isInLaunch = this.isInLaunch();

        uint256 currentTimestamp = !isTradingEnabled &amp;&amp; _tradingPausedTimestamp &gt; _launch

        if(!_isAllowedToTradeWhenDisabled[from] &amp;&amp; !_isAllowedToTradeWhenDisabled[to]) {
            require(isTradingEnabled, "Powerlabs: Trading is currently disabled.");
            require(! isBlocked[to], "Powerlabs: Account is blocked");
```

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 blockAccount(address account) public onlyOwner {
    uint256 currentTimestamp = _getNow();
    require(!_isBlocked[account], "Powerlabs: Account is already blocked");
    if (_isLaunched) {
        require(currentTimestamp.sub(_launchStartTimestamp) &lt; _blockedTimeLimit, &quot;Powerlabs:
    }
    _isBlocked[account] = true;
    emit BlockedAccountChange(account, true);
}
```

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.

```
constructor() DividendPayingToken("Powerlabs_Dividend_Tracker", "Powerlabs_Dividend_Tracker") {
    claimWait = 3600;
    minimumTokenBalanceForDividends = 200000000 * (10**18);
}
```

Recommendation

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

Exploit scenario

```
contract MyContract{
    uint 1_ether = 1000000000000000000;
}
```

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 activateMintOnBlockEmission(address _mintAddress, uint256 _mintAmount) public onlyOwner {
    require(!mintOnBlockEmission, "Powerlabs: Mint is already set activate");
    mintAddress = _mintAddress;
    mintAmount = _mintAmount;
    _blockTracker = block.number;
    mintOnBlockEmission = true;
}
```

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 _swapAndLiquify() private {
    uint256 contractBalance = balanceOf(address(this));
    uint256 initialBNBBalance = address(this).balance;

    uint256 amountToLiquify = contractBalance.mul(_liquidityFee).div(_totalFee).div(2);
    uint256 amountForStaking = contractBalance.mul(_stakingFee).div(_totalFee);
    uint256 amountToSwap = contractBalance.sub((amountToLiquify.add(amountForStaking)));

    _swapTokensForBNB(amountToSwap);

    uint256 bnbBalanceAfterSwap = address(this).balance.sub(initialBNBBalance);
    uint256 totalBNBFee = _totalFee.sub(_liquidityFee.div(2));
    uint256 amountBNBLiquidity = bnbBalanceAfterSwap.mul(_liquidityFee).div(totalBNBFee).div(2);
    uint256 amountBNBMarketing = bnbBalanceAfterSwap.mul(_marketingFee).div(totalBNBFee);
    uint256 amountBNBBuyBack = bnbBalanceAfterSwap.mul(_buyBackFee).div(totalBNBFee);
    uint256 amountBNBDev = bnbBalanceAfterSwap.mul(_devFee).div(totalBNBFee);
    uint256 amountBNBHolders = bnbBalanceAfterSwap.sub((amountBNBLiquidity.add(amountBNBMarketing).a

    payable(marketingWallet).transfer(amountBNBMarketing);
    payable(buyBackWallet).transfer(amountBNBBuyBack);
    payable(devWallet).transfer(amountBNBDev);
```

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.

Missing events arithmetic

Detect missing events for critical arithmetic parameters.

```
function activateMintOnBlockEmission(address _mintAddress, uint256 _mintAmount) public onlyOwner {
    require(!mintOnBlockEmission, "Powerlabs: Mint is already set activate");
    mintAddress = _mintAddress;
    mintAmount = _mintAmount;
    _blockTracker = block.number;
    mintOnBlockEmission = 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.

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.

Owner privileges

- 🟡 Owner can change max transaction amount
- 🟡 Owner can set fees higher than 25%
- 🟡 Owner can exclude from fees
- 🟡 Owner can pause the contract
- 🔴 Owner can mint new tokens
- 🔴 Owner can blacklist addresses
- ⚠️ Owner can activate trading for certain addresses even if trading is disabled
- ⚠️ Owner can cancel launch
- ⚠️ Owner can exclude addresses from dividend
- ⚠️ Owner can exclude addresses from max transaction limit
- ⚠️ Owner can exclude addresses from max wallet limit
- ⚠️ Owner is able to deactivate and activate mint on block emmision

Extra notes by the team

No notes

Notes by PowerLabs

“Powerlabs has a mint function in case we move to our dex with the same token where we would need to mint tokens as rewards and burn them to keep them deflationary.

At this moment we do not mint any tokens, all are preminted before presale. And we have plans to change that. It is entirely possible to use a 2 token structure for our dex with a new reward token for LP farmers.”

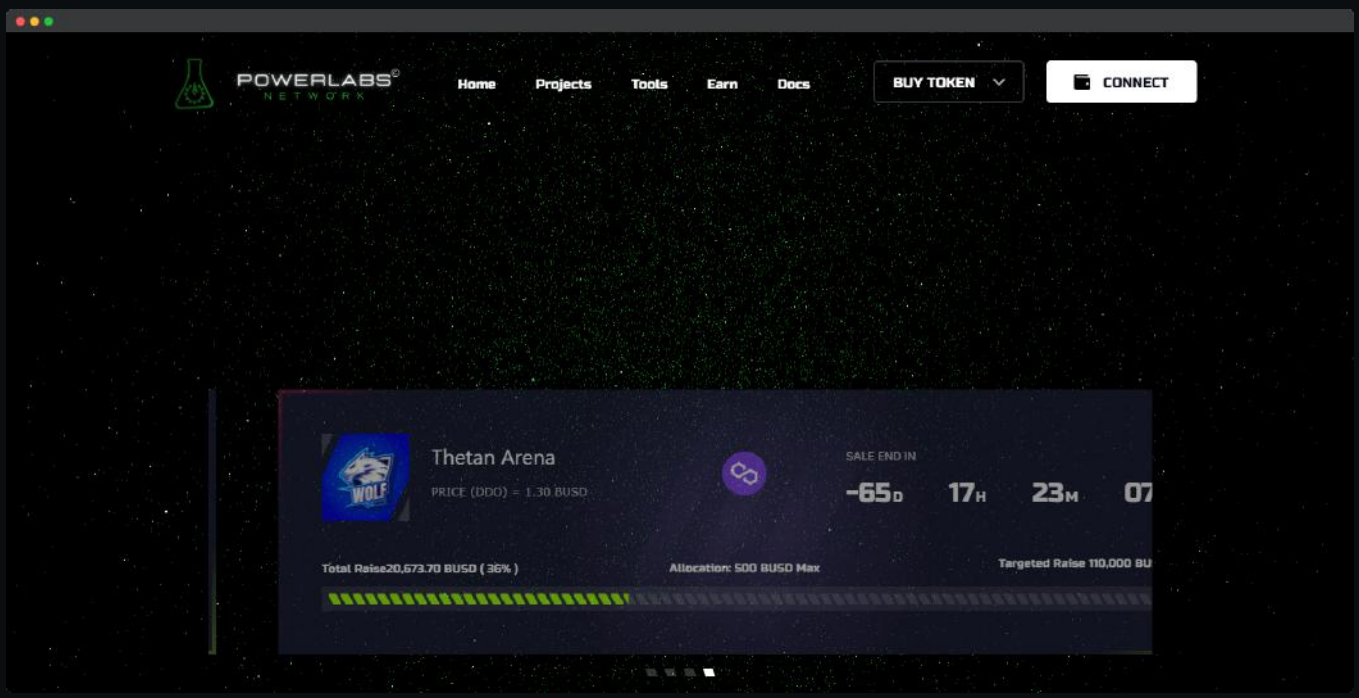
“Owner can cancel launch is a fail safe for if the launch didnt go right
Owner can exclude from dividends when the treasury become big enough you can exclude it from fees
Owner can exclude from max transaction is for the staking and farming contracts to hold the token amount that they do since it is a large percent of supply
same answer for max wallet limit
Minting function is for the dex purpose later on down in the road map but we plan on not using it unless needed
when we do pary time hours taxes go up on sell because buy in taxes are lowed to prevent mass dumping
pause trading is a function all contract should have incase of migration the ability to disable trading prevent people from buying old contract and getting rekt
activate trading for specific addresses is for adding the liquidity to the contract before launch and allowing token to be sent to presalers prior to launch”

Contract Snapshot

```
contract Powerlabs is Ownable, ERC20 {
using SafeMath for uint256;

IUniswapV2Router02 public uniswapV2Router;
address public immutable uniswapV2Pair;

string private constant _name = "Powerlabs";
string private constant _symbol = "PWL";
uint8 private constant _decimals = 9;

PowerlabsDividendTracker public dividendTracker;

bool public isTradingEnabled;
uint256 private _tradingPausedTimestamp;

// initialSupply
uint256 constant initialSupply = 1000000000 * (10**9);
```

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

PowerLabs

Audited by Coinsult.net

Date: 25 July 2022

✔ Advanced Manual Smart Contract Audit