Coinsult

Request your audit at coinsult.net

Advanced Manual
Smart Contract Audit

September 15, 2022

Audit requested by

CryptoSwaps

0x1ef5d98f556af1fd39ce045d591d0a37e9816b26

Table of Contents

Audit Summary

Audit Scope

Project Name	CryptoSwaps
Website	https://criptoswap.finance/
Blockchain	Binance Smart Chain
Smart Contract Language	Solidity
Contract Address	0x1ef5d98f556af1fd39ce045d591d0a37e9816b26
Audit Method	Static Analysis, Manual Review
Date of Audit	15 September 2022

This audit report has been prepared by Coinsult's experts at the request of the client. In this audit, the results of the static analysis and the manual code review will be presented. The purpose of the audit is to see if the functions work as intended, and to identify potential security issues within the smart contract.

The information in this report should be used to understand the risks associated with the smart contract. This report can be used as a guide for the development team on how the contract could possibly be improved by remediating the issues that were identified.

Tokenomics

Rank	Address	Quantity (Token)	Percentage
1	0x7f0f14b2ec480423e1fd89772bdd558dafde00c9	100,000,000	100.0000%

Source Code

Coinsult was comissioned by CryptoSwaps to perform an audit based on the following code:

https://bscscan.com/address/0x1ef5d98f556af1fd39ce045d591d0a37e9816b26#code

Disclaimer

This audit report has been prepared by Coinsult's experts at the request of the client. In this audit, the results of the static analysis and the manual code review will be presented. The purpose of the audit is to see if the functions work as intended, and to identify potential security issues within the smart contract.

The information in this report should be used to understand the risks associated with the smart contract. This report can be used as a guide for the development team on how the contract could possibly be improved by remediating the issues that were identified.

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.

Global Overview

Manual Code Review

In this audit report we will highlight the following issues:

Vulnerability Level	Total	Pending	Acknowledged	Resolved
● Informational	0	0	0	0
● Low-Risk	7	0	7	0
● Medium-Risk	0	0	0	0
● High-Risk	1	0	1	0

Privilege Overview

Coinsult checked the following privileges:

Contract Privilege	Description
Owner can mint?	● Owner cannot mint new tokens
Owner can blacklist?	● Owner cannot blacklist addresses
Owner can set fees > 25%?	● Owner can set the sell fee to 25% or higher
Owner can exclude from fees?	● Owner can exclude from fees
Owner can pause trading?	● Owner cannot pause the contract
Owner can set Max TX amount?	● Owner can set max transaction amount

More owner priviliges are listed later in the report.

● **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) {
    uint256 burnFeeAmount = amount.mul(burnFee).div(feeDenominator);
    uint256 amountWithFee = amount.sub(burnFeeAmount);

    if(inSwap){
        if (burnFeeAmount &gt; 0) { _burn(sender, burnFeeAmount); }
        return _basicTransfer(sender, recipient, amountWithFee);
    }

    checkTxLimit(sender, amount);

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

    if(shouldAutoBuyback()){ triggerAutoBuyback(); }

    if(!launched() &amp;&amp; recipient == pair){ require(_balances[sender] &gt; 0); launch(); }

    _balances[sender] = _balances[sender].sub(amountWithFee, "Insufficient Balance");
    if (burnFeeAmount &gt; 0) { _burn(sender, burnFeeAmount); }

    uint256 amountReceived = shouldTakeFee(sender) ? takeFee(sender, recipient, amountWithFee) : amount
    _balances[recipient] = _balances[recipient].add(amountReceived);
```

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.

Too many digits

Literals with many digits are difficult to read and review.

```
uint256 _totalSupply = 100000000 * (10 ** _decimals);
```

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 transferOwnership(address payable adr) public onlyOwner {owner = adr;
    authorizations[adr] = true;
    emit OwnershipTransferred(adr);
}
```

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 : li
    uint256 amountToLiquify = swapThreshold.mul(dynamicLiquidityFee).div(totalFee).div(2);
    uint256 amountToSwap = swapThreshold.sub(amountToLiquify);

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

    uint256 balanceBefore = address(this).balance;

    router.swapExactTokensForETHSupportingFeeOnTransferTokens(amountToSwap,0,path,address(this),block.t
    uint256 amountBNB = address(this).balance.sub(balanceBefore);
    uint256 totalBNBFee = totalFee.sub(dynamicLiquidityFee.div(2));
    uint256 amountBNBLiquidity = amountBNB.mul(dynamicLiquidityFee).div(totalBNBFee).div(2);
    uint256 amountBNBReflection = amountBNB.mul(reflectionFee).div(totalBNBFee);
    uint256 amountBNBMarketing = amountBNB.mul(marketingdevelopmentFee).div(totalBNBFee);

    try distributor.deposit{value: amountBNBReflection}() {} catch {}
    (bool success, /* bytes memory data */) = payable(marketingFeeReceiver).call{value: amountBNBMarket
    require(success, "receiver rejected ETH transfer");
```

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);
        REWARD.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.

Missing events arithmetic

Detect missing events for critical arithmetic parameters.

```
function setDistributionCriteria(uint256 _minPeriod, uint256 _minDistribution) external override onlyTo
    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]);
        }
        gasUsed = gasUsed.add(gasLeft.sub(gasleft()));
        gasLeft = gasleft();
        currentIndex++;
        iterations++;
    }
}
```

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`.

● **High-Risk:** Must be fixed, will bring problems.

Cooldown between trades without maximum

```
// enable cooldown between trades
function cooldownEnabled(bool _status, uint8 _interval) public onlyOwner {
    buyCooldownEnabled = _status;
    cooldownTimerInterval = _interval;
}
```

Recommendation

Owner can set cooldownTimeInterval without a constraint. Implement a require statement to prevent accidental errors, or high interval amounts.

Contract Privileges

Maximum Fee Limit Check

Coinsult tests if the owner of the smart contract can set the transfer, buy or sell fee to 25% or more. It is bad practice to set the fees to 25% or more, because owners can prevent healthy trading or even stop trading when the fees are set too high.

Type of fee	Description
Transfer fee	🔴 Owner can set the transfer fee to 25% or higher
Buy fee	🔴 Owner can set the buy fee to 25% or higher
Sell fee	🔴 Owner can set the sell fee to 25% or higher

Type of fee	Description
Max transfer fee	100%%
Max buy fee	100%%
Max sell fee	100%%

Contract Pausability Check

Coinsult tests if the owner of the smart contract has the ability to pause the contract. If this is the case, users can no longer interact with the smart contract; users can no longer trade the token.

Privilege Check	Description
Can owner pause the contract?	🟢 Owner cannot pause the contract

Max Transaction Amount Check

Coinsult tests if the owner of the smart contract can set the maximum amount of a transaction. If the transaction exceeds this limit, the transaction will revert. Owners could prevent normal transactions to take place if they abuse this function.

Privilege Check	Description
Can owner set max tx amount?	🔴 Owner can set max transaction amount

Exclude From Fees Check

Coinsult tests if the owner of the smart contract can exclude addresses from paying tax fees. If the owner of the smart contract can exclude from fees, they could set high tax fees and exclude themselves from fees and benefit from 0% trading fees. However, some smart contracts require this function to exclude routers, dex, cex or other contracts / wallets from fees.

Privilege Check	Description
Can owner exclude from fees?	● Owner can exclude from fees

Ability To Mint Check

Coinsult tests if the owner of the smart contract can mint new tokens. If the contract contains a mint function, we refer to the token’s total supply as non-fixed, allowing the token owner to “mint” more tokens whenever they want.

A mint function in the smart contract allows minting tokens at a later stage. A method to disable minting can also be added to stop the minting process irreversibly.

Minting tokens is done by sending a transaction that creates new tokens inside of the token smart contract. With the help of the smart contract function, an unlimited number of tokens can be created without spending additional energy or money.

Privilege Check	Description
Can owner mint?	🟢 Owner cannot mint new tokens

Ability To Blacklist Check

Coinsult tests if the owner of the smart contract can blacklist accounts from interacting with the smart contract. Blacklisting methods allow the contract owner to enter wallet addresses which are not allowed to interact with the smart contract.

This method can be abused by token owners to prevent certain / all holders from trading the token. However, blacklists might be good for tokens that want to rule out certain addresses from interacting with a smart contract.

Privilege Check	Description
Can owner blacklist?	🟢 Owner cannot blacklist addresses

Other Owner Privileges Check

Coinsult lists all important contract methods which the owner can interact with.

⚠️ Owner can authorize multiple addresses

⚠️ Owner can exclude addresses form dividend

⚠️ Owner can exclude addresses from timelock

⚠️ Owner can create cooldown period between trades

Notes

Notes by CryptoSwaps

No notes provided by the team.

Notes by Coinsult

✅ No notes provided by Coinsult

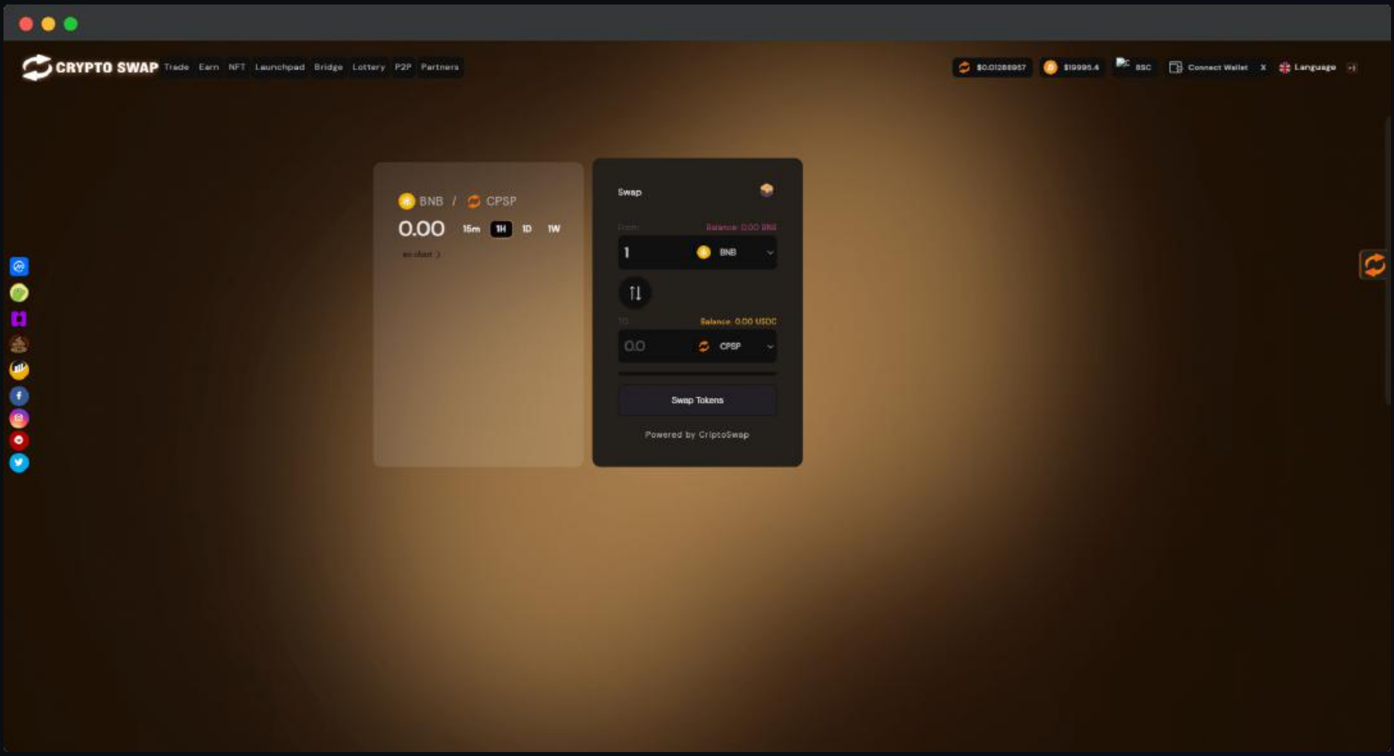
Contract Snapshot

This is how the constructor of the contract looked at the time of auditing the smart contract.

```
contract CryptoSwap is IBEP20, Auth {
using SafeMath for uint256;

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

string constant _name = "Crypto Swap V2";
string constant _symbol = "CPSP";
uint8 constant _decimals = 18;

uint256 _totalSupply = 100000000 * (10 ** _decimals);
uint256 public _maxTxAmount = _totalSupply / 1; //
uint256 public _maxWallet = 100000000 * 10**_decimals;

mapping (address =&gt; uint256) _balances;
mapping (address =&gt; mapping (address =&gt; uint256)) _allowances;
```

Website Review

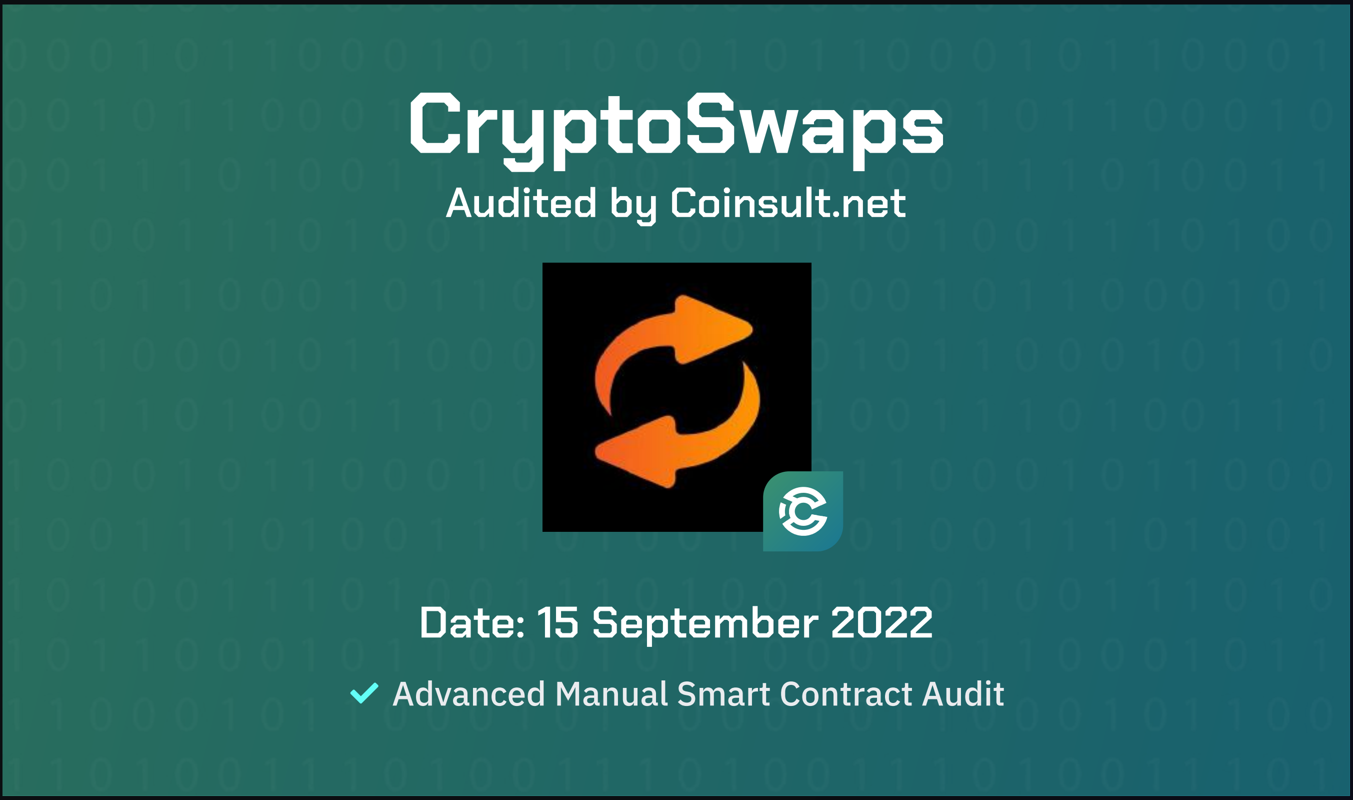
Coinsult checks the website completely manually and looks for visual, technical and textual errors. We also look at the security, speed and accessibility of the website. In short, a complete check to see if the website meets the current standard of the web development industry.



Type of check	Description
Mobile friendly?	● The website is mobile friendly
Contains jQuery errors?	● The website does not contain jQuery errors
Is SSL secured?	● The website is SSL secured
Contains spelling errors?	● The website does not contain spelling errors

Certificate of Proof

● Not KYC verified by Coinsult



CryptoSwaps

Audited by Coinsult.net

Date: 15 September 2022

✓ Advanced Manual Smart Contract Audit

Coinsult

coinsult.net

End of report

Smart Contract Audit

Request your smart contract audit / KYC

t.me/coinsult_tg