



Advanced Manual Smart Contract Audit

November 7, 2022

 CoinsultAudits

 info@coinsult.net

 coinsult.net

Audit requested by



ScamCoin (✗ Failed)

0x11aa4aE4b53988dBdD14A37FE81854800A650095

Table of Contents

1. Audit Summary

- 1.1 Audit scope
- 1.2 Tokenomics
- 1.3 Source Code

2. Disclaimer

3. Global Overview

- 3.1 Informational issues
- 3.2 Low-risk issues
- 3.3 Medium-risk issues
- 3.4 High-risk issues

4. Vulnerabilities Findings

5. Contract Privileges

- 5.1 Maximum Fee Limit Check
- 5.2 Contract Pausability Check
- 5.3 Max Transaction Amount Check
- 5.4 Exclude From Fees Check
- 5.5 Ability to Mint Check
- 5.6 Ability to Blacklist Check
- 5.7 Owner Privileges Check

6. Notes

- 6.1 Notes by Coinsult
- 6.2 Notes by ScamCoin (✗ Failed)

7. Contract Snapshot

8. Website Review

9. Certificate of Proof

Audit Summary

Project Name	ScamCoin (✗ Failed)
Website	https://www.scamcoin.art/
Blockchain	Binance Smart Chain
Smart Contract Language	Solidity
Contract Address	0x11aa4aE4b53988dBdD14A37FE81854800A650095
Audit Method	Static Analysis, Manual Review
Date of Audit	7 November 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.

Audit Scope

Source Code



Coinsult was commissioned by ScamCoin (✗ Failed) to perform an audit based on the following code:

<https://bscscan.com/address/0x11aa4aE4b53988dBdD14A37FE81854800A650095#code>

Note that we only audited the code available to us on this URL at the time of the audit. If the URL is not from any block explorer (main net), it may be subject to change. Always check the contract address on this audit report and compare it to the token you are doing research for.

Audit not passed.

Tokenomics

Rank	Address	Quantity (Token)	Percentage
1	 0x51c50a24338c88f76d827458b17c801691bb950a	7,422,500,000	74.2250%
2	 Pinksale: PinkLock V2	2,277,500,000	22.7750%
3	0xc95af430e76d5b5885e31a5bc35eb324f70d10b2	300,000,000	3.0000%

Audit Method

Coinsult's manual smart contract audit is an extensive methodical examination and analysis of the smart contract's code that is used to interact with the blockchain. This process is conducted to discover errors, issues and security vulnerabilities in the code in order to suggest improvements and ways to fix them.

➔ Automated Vulnerability Check

Coinsult uses software that checks for common vulnerability issues within smart contracts. We use automated tools that scan the contract for security vulnerabilities such as integer-overflow, integer-underflow, out-of-gas-situations, unchecked transfers, etc.

➔ Manual Code Review

Coinsult's manual code review involves a human looking at source code, line by line, to find vulnerabilities. Manual code review helps to clarify the context of coding decisions. Automated tools are faster but they cannot take the developer's intentions and general business logic into consideration.

➔ Used Tools

- Slither: Solidity static analysis framework
- Remix: IDE Developer Tool
- CWE: Common Weakness Enumeration
- SWC: Smart Contract Weakness Classification and Test Cases
- DEX: Testnet Blockchains

Risk Classification

Coinsult uses certain vulnerability levels, these indicate how bad a certain issue is. The higher the risk, the more strictly it is recommended to correct the error before using the contract.

Vulnerability Level	Description
● Informational	Does not compromise the functionality of the contract in any way
● Low-Risk	Won't cause any problems, but can be adjusted for improvement
● Medium-Risk	Will likely cause problems and it is recommended to adjust
● High-Risk	Will definitely cause problems, this needs to be adjusted

Coinsult has four statuses that are used for each risk level. Below we explain them briefly.

Risk Status	Description
Total	Total amount of issues within this category
Pending	Risks that have yet to be addressed by the team
Acknowledged	The team is aware of the risks but does not resolve them
Resolved	The team has resolved and remedied the risk

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	6	6	0	0
● Medium-Risk	0	0	0	0
● High-Risk	2	0	2	0


Centralization Risks

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 cannot 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 privileges are listed later in the report.

Error Code	Description
CS-01	State variable visibility is not set.

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

State variable visibility is not set.

```
bool inSwapAndLiquify;
```

Recommendation

It is best practice to set the visibility of state variables explicitly. The default visibility for “inSwapAndLiquify” is internal. Other possible visibility settings are public and private.

Error Code	Description
SLT: 078	Conformance to numeric notation best practices

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

Too many digits

Literals with many digits are difficult to read and review.

```
uint256 public numTokensSellToAddToLiquidity = 500000 * 10**18;  
uint256 public _maxTxAmount = 10000 * 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.

Error Code	Description
SLT: 056	Missing Zero Address Validation

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

No zero address validation for some functions

Detect missing zero address validation.

```
function setMarketingWallet(address payable newWallet) external onlyOwner() {  
    marketingWallet = newWallet;  
}
```

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, so Bob loses ownership of the contract.

Error Code	Description
SLT: 038	Imprecise arithmetic operations order

● **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 tBurn = tAmount.div(100).mul(_burnFee);
uint256 tMarketing = tAmount.div(100).mul(_marketingFee);
```

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.

Error Code	Description
SLT: 076	Costly operations in a loop

● **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 includeInReward(address account) external onlyOwner() {
    require(!_isExcluded[account], "Account is already excluded");
    for (uint256 i = 0; i < _excluded.length; i++) {
        if (_excluded[i] == account) {
            _excluded[i] = _excluded[_excluded.length - 1];
            _tOwned[account] = 0;
            _isExcluded[account] = false;
            _excluded.pop();
            break;
        }
    }
}
```

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 `SSTOREs`, which might lead to an out-of-gas.

Error Code	Description
CS: 071	Using safemath in Solidity 0.8.0+

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

Using safemath in Solidity 0.8.0+

SafeMath is generally not needed starting with Solidity 0.8, since the compiler now has built in overflow checking.

```
library SafeMath {
    /**
     * @dev Returns the addition of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
     */
    function tryAdd(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        unchecked {
            uint256 c = a + b;
            if (c < a) return (false, 0);
            return (true, c);
        }
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, with an overflow flag.
```

Recommendation

Check if you really need SafeMath and consider removing it.

Error Code	Description
CSH-01	Different use of decimals throughout the contract

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

Different use of decimals throughout the contract

```
uint256 private _tTotal = 10000 * (10**18);

uint8 private _decimals = 12;

uint256 public numTokensSellToAddToLiquidity = 500000 * 10**18;
uint256 public _maxTxAmount = 10000 * 10**18;

    bool overMinTokenBalance = contractTokenBalance >= numTokensSellToAddToLiquidity;
    if (
        overMinTokenBalance &&&
        !inSwapAndLiquify &&&
        to == uniswapV2Pair &&&
        swapAndLiquifyEnabled
    ) {
        contractTokenBalance = numTokensSellToAddToLiquidity;
        //add liquidity and send bnb to marketing wallet
        swapAndLiquify(contractTokenBalance);
    }
```

Recommendation

Since your contract decimals are 12, and you are setting the vars with 18, your math is off. numTokensSellToAddToLiquidity is bigger than your total supply, so it will never be matched.

swapAndLiquify will never be called this way.

Error Code	Description
CSH-02	Marketing and Liquidityfee not working

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

Marketing and Liquidityfee not working

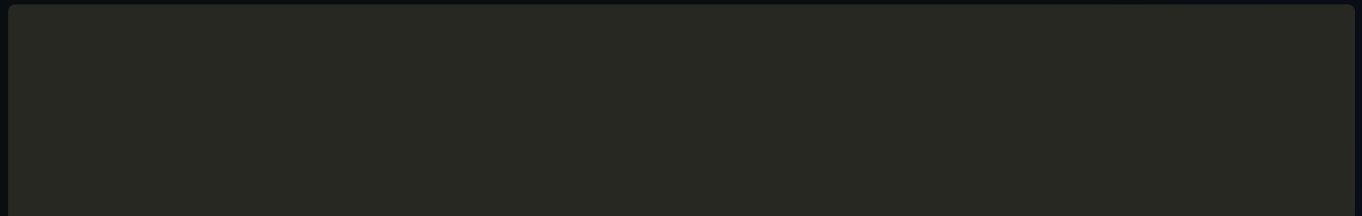
```
bool overMinTokenBalance = contractTokenBalance >= numTokensSellToAddToLiquidity;
if (
    overMinTokenBalance &&&
    !inSwapAndLiquify &&&
    to == uniswapV2Pair &&&
    swapAndLiquifyEnabled
) {
    contractTokenBalance = numTokensSellToAddToLiquidity;
    //add liquidity and send bnb to marketing wallet
    swapAndLiquify(contractTokenBalance);
}
```

Recommendation

Due to high risk error 1, marketing and liquidity fee will not work. Discussed with owner, owner said:
“That’s fine for me”

Error Code	Description
CSH-03	

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



Recommendation

No recommendation

Maximum Fee Limit Check

Error Code	Description
CEN-01	Centralization: Operator Fee Manipulation

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 cannot set the transfer fee to 25% or higher
Buy fee	● Owner cannot set the buy fee to 25% or higher
Sell fee	● Owner cannot set the sell fee to 25% or higher

Type of fee	Description
Max transfer fee	10%
Max buy fee	10%
Max sell fee	10%

Function

```
function setFeePercent(uint256 taxFee, uint256 liquidityFee, uint256 marketingFee, uint256 burnFee) external {
    require(taxFee.add(liquidityFee).add(marketingFee).add(burnFee) <= 10, "tax too high");
    _taxFee = taxFee;
    _liquidityFee = liquidityFee;
    _marketingFee = marketingFee;
    _burnFee = burnFee;
}
```

Contract Pausability Check

Error Code	Description
CEN-02	Centralization: Operator Pausability

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

Error Code	Description
CEN-03	Centralization: Operator Transaction Manipulation

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

Function

```
function setMaxTxAmount(uint256 maxTxAmount) external onlyOwner() {
    _maxTxAmount = maxTxAmount;
    require(_maxTxAmount > totalSupply().div(400), "value too low");
}
```

Exclude From Fees Check

Error Code	Description
CEN-04	Centralization: Operator Exclusion

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

Function

```
function excludeFromFee(address account) public onlyOwner {
    _isExcludedFromFee[account] = true;
}
```


Ability To Mint Check

Error Code	Description
CEN-05	Centralization: Operator Increase Supply

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

Error Code	Description
CEN-06	Centralization: Operator Dissallows Wallets

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

Error Code	Description
CEN-100	Centralization: Operator Privileges

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

⚠ Owner can set numTokensSellToAddToLiquidity without any limits

Notes

Notes by ScamCoin (✗ Failed)

No notes provided by the team.

Notes by Coinsult

Audit not passed.

After having discussions with the owner to redeploy the contract, he told us he has no budget and wanted to proceed with this one. His solution was to set all the fees to 0 so that the contract will function.

Contract Snapshot

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

```
contract ScamCoin is Context, IERC20, Ownable {
    using SafeMath for uint256;
    using Address for address;

    bool inSwapAndLiquify;
    bool public swapAndLiquifyEnabled = true;

    mapping (address => uint256) private _rOwned;
    mapping (address => uint256) private _tOwned;
    mapping (address => mapping (address => uint256)) private _allowances;
    mapping (address => bool) private _isExcludedFromFee;
    mapping (address => bool) private _isExcluded;
    address[] private _excluded;

    uint256 private constant MAX = ~uint256(0);
    uint256 private _tTotal = 10000 * (10**18);
    uint256 private _rTotal = (MAX - (MAX % _tTotal));
    uint256 private _tFeeTotal;

    string private _name = "ScamCoin";
    string private _symbol = "SCAM";
    uint8 private _decimals = 12;

    uint256 public _taxFee = 3;
    uint256 private _previousTaxFee = _taxFee;

    uint256 public _liquidityFee = 0;
    uint256 private _previousLiquidityFee = _liquidityFee;

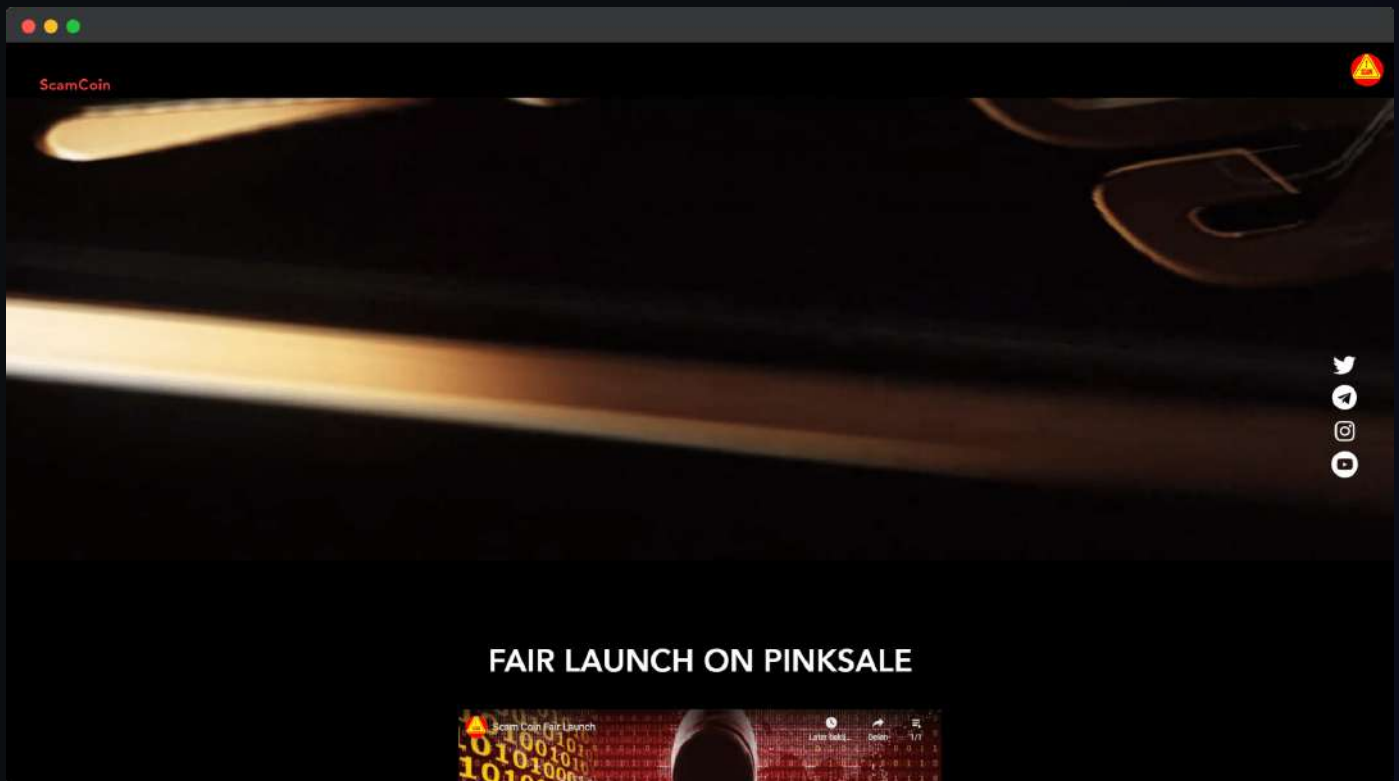
    uint256 public _burnFee = 1;
    uint256 private _previousBurnFee = _burnFee;

    address public deadAddress = 0x0000000000000000000000000000000000000000000000000000000000000000dEaD;

    uint256 public _marketingFee = 0;
    uint256 private _previousMarketingFee = _marketingFee;
```

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

ScamCoin (✗ Failed)

Audited by Coinsult.net




Date: 7 November 2022

✓ Advanced Manual Smart Contract Audit

End of report

Smart Contract Audit

 CoinsultAudits

 info@coinsult.net

 coinsult.net

Request your smart contract audit / KYC

t.me/coinsult_tg