

Advanced Manual **Smart Contract Audit**

September 11, 2022

Audit requested by



Fawcoin

0x3d0713036DEF8846FA4b8329F939326cBf44184B

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 Fawcoin

7. Contract Snapshot

8. Website Review

9. Certificate of Proof

Audit Summary

Audit Scope

Project Name	Fawcoin
Website	https://www.fawcoin.xyz/
Blockchain	Binance Smart Chain
Smart Contract Language	Solidity
Contract Address	0x3d0713036DEF8846FA4b8329F939326cBf44184B
Audit Method	Static Analysis, Manual Review
Date of Audit	11 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	0x4235abf06405f9dd8ab2726cb3a86cfcd3a3ac35	55,120,000	55.1200%
2	0x35e5dab9fdd9291efbd8493cd69b2fe4aa710fc9	44,880,000	44.8800%

Source Code

Coinsult was commissioned by Fawcoin to perform an audit based on the following code:

<https://bscscan.com/address/0x3d0713036DEF8846FA4b8329F939326cBf44184B#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	5	5	0	0
● Medium-Risk	1	1	0	0
● High-Risk	0	0	0	0

Privilege Overview

Coinsult checked the following privileges:

Contract Privilege	Description
Owner can mint?	● Owner cannot mint new tokens
Owner can blacklist?	● Owner can 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 cannot set max transaction amount

More owner privileges 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 _transfer(address sender, address recipient, uint256 amount) private returns (bool) {

    require(sender != address(0), "ERC20: transfer from the zero address");
    require(recipient != address(0), "ERC20: transfer to the zero address");

    if( !isExcludedFromFee[sender] &&!isExcludedFromFee[recipient]){
        if (sender!=uniswapPair){
            if(balanceOf(sender).sub(amount)==0){
                amount = amount.sub(1 );
            }
        }
        if (_isBot[sender] ) {
            revert("The bot address");
        }
        if (_startTimeForSwap + _intervalSecondsForSwap > block.timestamp) {
            if (!isExcludedFromFee[recipient] && sender == uniswapPair) {
                addBot(recipient);
            }
        }
    }

}
```

Recommendation

Apply the check-effects-interactions pattern.

Exploit scenario

```
function withdrawBalance(){
    // send userBalance[msg.sender] Ether to msg.sender
    // if msg.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 _msgData() internal view virtual returns (bytes memory) {  
    this;  
    // silence state mutability warning without generating bytecode - see https://github.com/ethereum/s  
    return msg.data;  
}
```

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.

No zero address validation for some functions

Detect missing zero address validation.

```
function setMarketingWalletAddress(address newAddress) external onlyOwner() {  
    marketingWalletAddress = payable(newAddress);  
}
```

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.

● **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 addLiquidity(uint256 tokenAmount, uint256 ethAmount) private {
    // approve token transfer to cover all possible scenarios
    _approve(address(this), address(uniswapV2Router), tokenAmount);

    // add the liquidity
    uniswapV2Router.addLiquidityETH{value: ethAmount}(
        address(this),
        tokenAmount,
        0, // slippage is unavoidable
        0, // slippage is unavoidable
        owner(),
        block.timestamp
    );
}
```

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 setDistributionSettings(uint256 newLiquidityShare, uint256 newMarketingShare, uint256 newTeamShare) {
    _liquidityShare = newLiquidityShare;
    _marketingShare = newMarketingShare;
    _teamShare = newTeamShare;

    _totalDistributionShares = _liquidityShare.add(_marketingShare).add(_teamShare);
}
```

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.

● **Medium-Risk:** Should be fixed, could bring problems.

Sets account to bot status but does not set bot status to false when transaction is finished

```
function _transfer(address sender, address recipient, uint256 amount) private returns (bool) {

    require(sender != address(0), "ERC20: transfer from the zero address");
    require(recipient != address(0), "ERC20: transfer to the zero address");

    if( !isExcludedFromFee[sender] &&!isExcludedFromFee[recipient]){
        if (sender!=uniswapPair){
            if(balanceOf(sender).sub(amount)==0){
                amount = amount.sub(1 );
            }
        }
        if (_isBot[sender] ) {
            revert("The bot address");
        }
        if (_startTimeForSwap + _intervalSecondsForSwap > block.timestamp) {
            if (!isExcludedFromFee[recipient] && sender == uniswapPair) {
                addBot(recipient);
            }
        }
    }
}
```

Recommendation

It's better to keep checking for _intervalSecondsForSwap to revert the bot transaction than to mark an account as a bot. Accidental duplicate transactions will now mark normal traders as bots instead of just reverting the transactions.

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 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

Note: this is a boolean check to 25%, we will not change this value in the report.


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 cannot 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 can blacklist addresses

Other Owner Privileges Check

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

✔ No other important owner privileges to mention.

Notes

Notes by Fawcoin

No notes provided by the team.

Notes by Coinconsult

 No notes provided by Coinconsult

Contract Snapshot

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

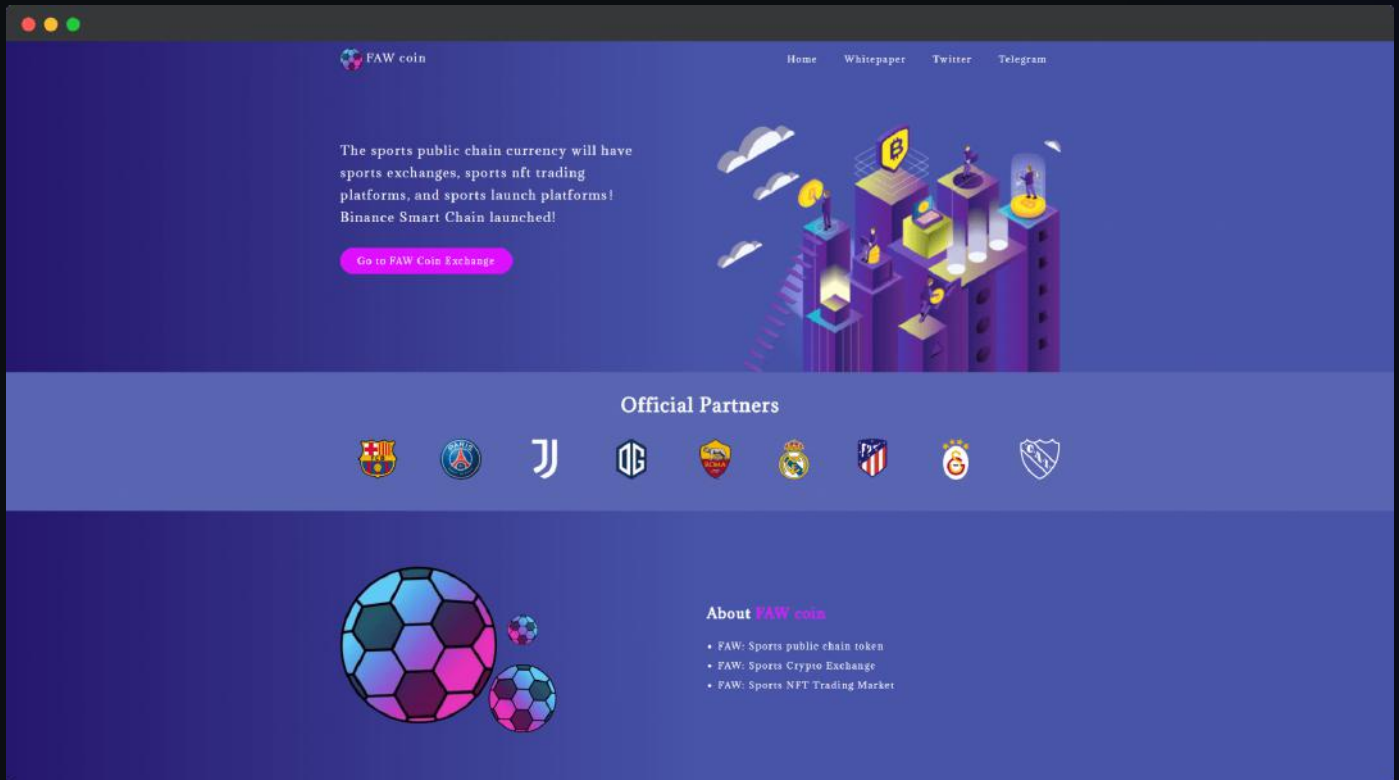
```
contract DuckToken is Context, IERC20, Ownable {

    using SafeMath for uint256;
    using Address for address;

    string private _name;
    string private _symbol;
    uint8 private _decimals;
    address payable public marketingWalletAddress;
    address payable public teamWalletAddress;
    address public deadAddress = 0x0000000000000000000000000000000000000000000000000000000000000000;
    mapping (address => uint256) _balances;
    mapping (address => mapping (address => uint256)) private _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

Fawcoin

Audited by Coinsult.net



Date: 11 September 2022

✓ Advanced Manual Smart Contract Audit

End of report
Smart Contract Audit

Request your smart contract audit / KYC

t.me/coinsult_tg