Coinsult

Request your audit at coinsult.net

Advanced Manual
Smart Contract Audit

October 13, 2022

Audit requested by

Bloom

0xbe6DD68261501C1A8dD2d7FF3c2EFA8827b08DCd

Table of Contents

Audit Summary

Audit Scope

Project Name	Bloom
Website	https://bloomsocial.io/
Blockchain	Binance Smart Chain
Smart Contract Language	Solidity
Contract Address	0xbe6DD68261501C1A8dD2d7FF3c2EFA8827b08DCd
Audit Method	Static Analysis, Manual Review
Date of Audit	13 October 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	0xbb059140fe74c21fcd903aeb3e5f1ae0280af40a	69,321,600	69.3216%
2	Pinksale: PinkLock V2	30,678,400	30.6784%

Source Code

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

https://bscscan.com/address/0xbe6DD68261501C1A8dD2d7FF3c2EFA8827b08DCd#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	4	4	0	0
Medium-Risk	0	0	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 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 cannot set max transaction amount

More owner priviliges are listed later in the report.

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

Avoid relying on block.timestamp

block.timestamp can be manipulated by miners.

```
function setLpPair(address pair, bool enabled) external onlyOwner {
    if (!enabled) {
        lpPairs[pair] = false;
        protections.setLpPair(pair, false);
    } else {
        if (timeSinceLastPair != 0) {
            require(block.timestamp - timeSinceLastPair &gt; 3 days, "3 Day cooldown.!");
        }
        lpPairs[pair] = true;
        timeSinceLastPair = block.timestamp;
        protections.setLpPair(pair, 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.

```
if (ratios.marketing &gt; 0) {
    (success, ) = _taxWallets.marketing.call{
        value: marketingBalance,
        gas: 35000
    }("");
}
if (ratios.BFund &gt; 0) {
    (success, ) = _taxWallets.BFund.call{
        value: BFundBalance,
        gas: 35000
    }("");
}
if (ratios.LP &gt; 0) {
    (success, ) = _taxWallets.LP.call{value: LPBalance, gas: 35000}("");
}
```

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.

Missing events arithmetic

Detect missing events for critical arithmetic parameters.

```
function setTaxes(
    uint16 buyFee,
    uint16 sellFee,
    uint16 transferFee
) external onlyOwner {
    require(!taxesAreLocked, "Taxes are locked.");
    require(
        buyFee &lt;= maxBuyTaxes &amp;&amp;
            sellFee &lt;= maxSellTaxes &amp;&amp;
            transferFee &lt;= maxTransferTaxes,
        &quot;Cannot exceed maximums.&quot;
    );
    _taxRates.buyFee = buyFee;
    _taxRates.sellFee = sellFee;
    _taxRates.transferFee = transferFee;
}
```

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 multiSendTokens(address[] memory accounts, uint256[] memory amounts) external onlyOwner {
    require(accounts.length == amounts.length, "Lengths do not match.");
    for (uint16 i = 0; i = amounts[i]*10**_decimals, "Not enough tokens.");
        finalizeTransfer(msg.sender, accounts[i], amounts[i]*10**_decimals, false, false, true);
    }
}
```

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`.

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

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

Function

```
function setTaxes(
    uint16 buyFee,
    uint16 sellFee,
    uint16 transferFee
) external onlyOwner {
    require(!taxesAreLocked, "Taxes are locked.");
    require(
        buyFee &lt;= maxBuyTaxes &amp;&amp;
            sellFee &lt;= maxSellTaxes &amp;&amp;
            transferFee &lt;= maxTransferTaxes,
        &quot;Cannot exceed maximums.&quot;
    );
    _taxRates.buyFee = buyFee;
    _taxRates.sellFee = sellFee;
    _taxRates.transferFee = transferFee;
}
```

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

Function

```
function setExcludedFromFees(address account, bool enabled)
    public
    onlyOwner
{
    _isExcludedFromFees[account] = enabled;
}
```

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 set external ‘Protections’ address

⚠️ Owner can remove accounts from sniper protection

⚠️ Trading disabled by default, owner can enable it but never pause it again

⚠️ Owner can transfer all BNB within the contract

Notes

Notes by Bloom

No notes provided by the team.

Notes by Coinsult

✅ No notes provided by Coinsult

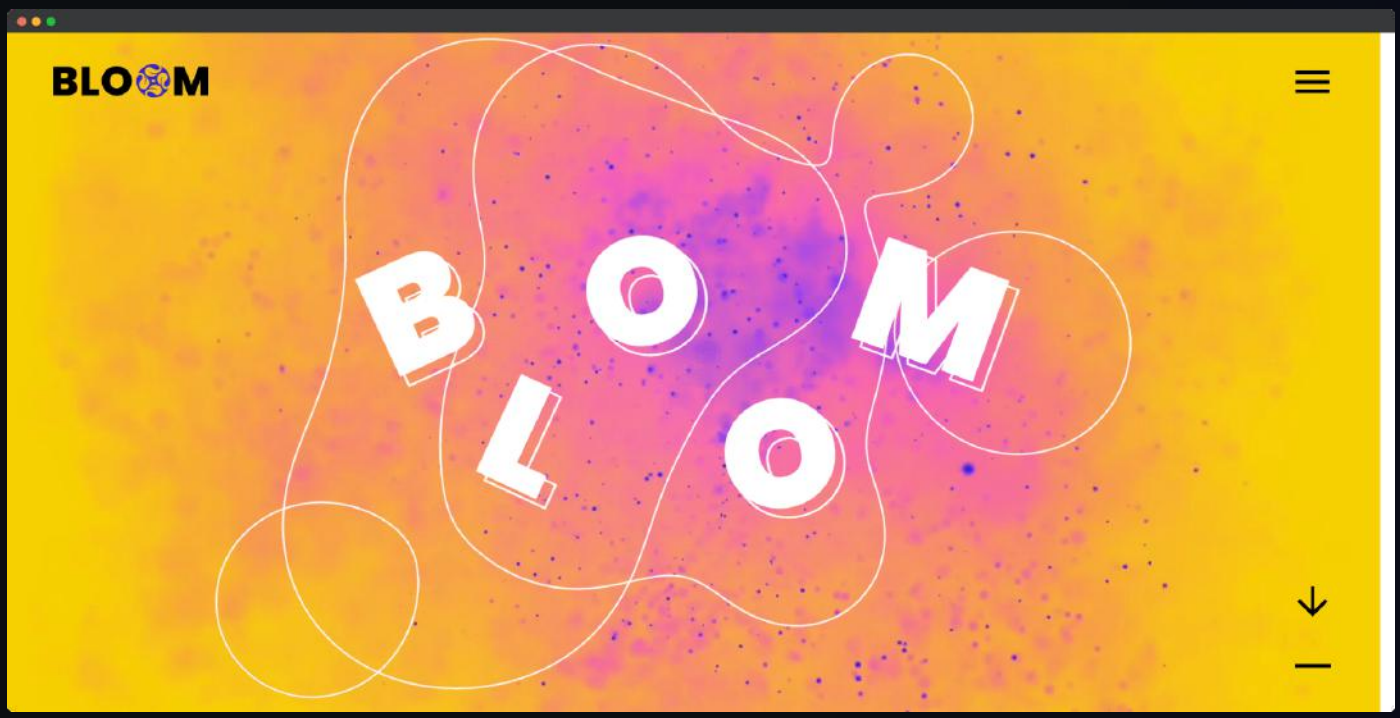
Contract Snapshot

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

```
contract Bloom is IERC20 {
mapping(address =&gt; uint256) private _tOwned;
mapping(address =&gt; bool) lpPairs;
uint256 private timeSinceLastPair = 0;
mapping(address =&gt; mapping(address =&gt; uint256)) private _allowances;

mapping(address =&gt; bool) private _liquidityHolders;
mapping(address =&gt; bool) private _isExcludedFromProtection;
mapping(address =&gt; bool) private _isExcludedFromFees;
mapping(address =&gt; bool) private presaleAddresses;
bool private allowedPresaleExclusion = true;

uint256 private constant startingSupply = 100_000_000;
string private constant _name = "Bloom";
string private constant _symbol = "BLM";
uint8 private constant _decimals = 9;
uint256 private constant _tTotal = startingSupply * 10**_decimals;

struct Fees {
    uint16 buyFee;
    uint16 sellFee;
    uint16 transferFee;
}

struct Ratios {
    uint16 marketing;
    uint16 BFund;
    uint16 LP;
    uint16 totalSwap;
}

Fees public _taxRates = Fees({buyFee: 600, sellFee: 600, transferFee: 600});

Ratios public _ratios =
    Ratios({marketing: 400, BFund: 100, LP: 100, totalSwap: 600});

uint256 public constant maxBuyTaxes = 1000;
```

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.



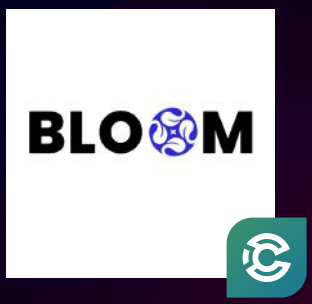
Type of check	Description
Mobile friendly?	The website is mobile friendly
Contains jQuery errors?	The website does not contain jQuery errors
Is SSL secured?	The website is SSL secured
Contains spelling errors?	The website does not contain spelling errors

Certificate of Proof

Not KYC verified by Coinsult

Bloom

Audited by Coinsult.net



Date: 13 October 2022

Advanced Manual Smart Contract Audit

Coinsult

coinsult.net

End of report
Smart Contract Audit