BlocBite

SMART CONTRACT AUDIT REPORT

Prepared by:

BlockAudit

Date Of Enrollment:

July 11th, 2024 - July 19th, 2024

Visit : www.blockaudit.report

TABLE OF CONTENTS

INTRODUCTION 2-3

- Summary 2
- Overview 3

FINDINGS 4-12

- **Finding Overview 4**
- M-01 5
- L-01 6
- L-02 7-8
- L-03 9
- L-04 10
- L-05 11
- G-01 12

APPENDIX 13

DISCLAIMER 15

ABOUT 17

SUMMARY

This Audit Report mainly focuses on the extensive security of **BlocBite** Smart Contracts. With this report, we attempt to ensure the reliability and correctness of the smart contract by complete and rigorous assessment of the system's architecture and the smart contract codebase.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts producedby industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified inpublic;
- Provide more transparency on privileged activities once the protocol is live.

OVERVIEW

Project Summary

Project Name	BlocBite
Logo	
Platform	-
Language	Solidity
Contract Address	-

File Summary

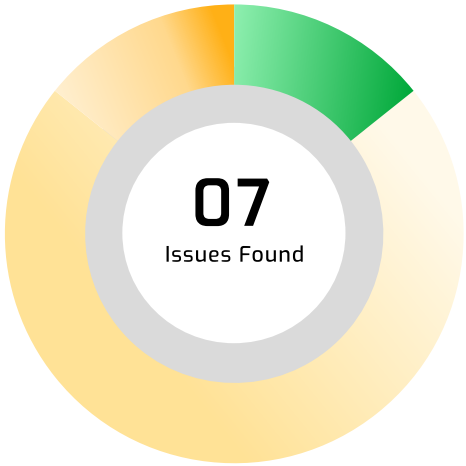
ID	File Name	Audit Status
BLOC	BlocBite.sol	Pass

Audit Summary

Date of Delivery	19 July 2024
Audit Methodology	Code Analysis. Automatic Assesment, Manual Review
Audit Result	**Passed** ✓✓
Audit Team	BlockAudit Report Team

FINDINGS

Critical	0	0.0%
High	0	0.0%
Medium	1	14%
Low	5	70%
Informational	0	0.0%
Ownership	0	0.0%
Gas Optimization	1	14%



Vulnerability Findings Summary

ID	Type	Instances	Severity	Status
M-01	Use SafeERC20 Library	-	Medium	Acknowledged
L-01	Centralization Risks	-	Low	Acknowledged
L-02	Missing slippage/min-return check in swapTokensForETH() function	-	Low	Acknowledged
L-03	Lack of 2-step transfer of ownership	-	Low	Acknowledged
L-04	Use a more recent version of Solidity	-	Low	Acknowledged
L-05	Use of unnamed Mappings	-	Low	Acknowledged
G-01	Use Custom Errors instead of Revert Strings to save Gas	-	Gas Optimisation	Acknowledged

M-01

Type	Use SafeERC20 Library
Severity	Medium
File	Bloc.sol
Line	-
Status	**Acknowledged**

Description

ERC20 standard allows the transfer function of some contracts to return bool or return nothing. Some tokens such as USDT return nothing.
This could lead to funds stuck in the contract without the possibility to retrieve them. Using safeTransferFrom of SafeERC20.sol is recommended instead.

Remediation

We recommend using OpenZeppelin's SafeERC20 versions with the safeTransfer and safeTransferFrom functions that handle the return value check as well as non-standard-compliant tokens.

Snapshot

```
1078    function transferTokens(address tokenAddress, address to, uint256 amount) public onlyOwner {
1079:           IERC20(tokenAddress).transfer(to, amount); //@audit use safeTransfer instead of transfer
1080        }
```

L-01

Type	Centralization Risks
Severity	Low
File	Bloc.sol
Line	-
Status	**Acknowledged**

Description

The protocol has an owner with privileged rights to perform admin tasks that can affect users. While the protocol owner is regarded as a trusted party, the owner can change some critical parameters like AutomatedMarketMakerPair, Enable/Disable swap, SwapTokensAtAmount, etc. Also, the owner of the contract can transfer any ERC20 tokens and ETH out of the contract.

Remediation

Make the owner a multi-sig and/or introduce a timelock for improved community oversight. Also, if possible, Specify the owner's privileges and responsibilities in the documentation.

Snapshot

```
1078:   function transferTokens(address tokenAddress, address to, uint256 amount) public onlyOwner {
1079        IERC20(tokenAddress).transfer(to, amount);
1080    }
1081
1082:   function migrateETH(address payable recipient) public onlyOwner {
1083      require(recipient != address(0), "Zero address");
1084        recipient.transfer(address(this).balance);
1085    }
```

L-02

Type	Missing Slippage/Min-Return Check In SwapTokensForETH() Function
Severity	Low
File	Bloc.sol
Line	-
Status	**Acknowledged**

Description

The contracts are missing slippage checks which can lead to being vulnerable to sandwich attacks. Trades can happen at a bad price and lead to receiving fewer tokens than at a fair market price. The attacker's profit is the protocol's loss.

A common attack in DeFi is the sandwich attack. Upon observing a trade of asset X for asset Y, an attacker frontruns the victim trade by also buying asset Y, lets the victim execute the trade, and then backruns (executes after) the victim by trading back the amount gained in the first trade. Intuitively, one uses the knowledge that someone's going to buy an asset, and that this trade will increase its price, to make a profit. The attacker's plan is to buy this asset cheap, let the victim buy at an increased price, and then sell the received amount again at a higher price afterwards.

Remediation

Add minimum return amount checks. Accept a function parameter that can be chosen by the transaction sender, then check that the actually received amount is above this parameter.

Alternatively, check if it's feasible to send these transactions directly to a miner such that they are not visible in the public mempool.

Snapshot

```
1051:   function swapTokensForETH(uint256 tokenAmount) private {
1052        address[] memory path = new address[](2);
1053        path[0] = address(this);
1054        path[1] = uniswapV2Router.WETH();
1055
1056        _approve(address(this), address(uniswapV2Router), tokenAmount);
1057        uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(
1058            tokenAmount,
1059            0,
1060            path,
1061            address(this),
1062            block.timestamp
1063        );
1064    }
1065
```

L-03

Type	Lack Of 2-Step Transfer Of Ownership.
Severity	Low
File	Bloc.sol
Line	-
Status	**Acknowledged**

Description

Ownable2Step is safer than Ownable for smart contracts because the owner cannot accidentally transfer smart contract ownership to a mistyped address. Rather than directly transferring to the new owner, the transfer is only completed when the new owner accepts ownership.
Also, If the nominated EOA account is not valid, the owner may accidentally transfer ownership to an uncontrolled account, breaking all functions with the onlyOwner() modifier.

Remediation

Recommend considering implementing a two-step process where the owner nominates an account and the nominated account needs to call an acceptOwnership() function for the transfer of ownership to fully succeed.

Snapshot

```
990: contract BlocBite is ERC20, Ownable {
```

L-04

Type	Use A More Recent Version Of Solidity
Severity	Low
File	Bloc.sol
Line	-
Status	**Acknowledged**

Description

When deploying contracts, you should use the latest released version of Solidity. Furthermore, breaking changes, as well as new features, are introduced regularly.

Remediation

Consider using the latest version of solidity i.e. 0.8.25

Snapshot

```
3: pragma solidity 0.8.17;
```

L-05

Type	Use Of Unnamed Mapping
Severity	Low
File	Bloc.sol
Line	-
Status	**Acknowledged**

Description

The contracts use unnamed mappings, which can make it difficult to understand the purpose of each mapping. Consider using named mappings to make it easier to understand the purpose of each mapping. This can make the code less readable and maintainable.

Remediation

Rename the mappings to make it clear what the purpose of each mapping is.

Snapshot

```
659:     mapping(address => uint256) private _balances;
661:     mapping(address => mapping(address => uint256)) private _allowances;
1003:  mapping(address => bool) public automatedMarketMakerPairs;
```

G-01

Type	Use Custom Errors Instead Of Revert Strings To Save Gas
Severity	Gas Optimisation
File	Bloc.sol
Line	-
Status	**Acknowledged**

Description

Custom errors from Solidity 0.8.4 are cheaper than revert strings (cheaper deployment cost and runtime cost when the revert condition is met). Custom errors are defined using the error statement, which can be used inside and outside of contracts (including interfaces and libraries).

Remediation

We suggest replacing revert strings with custom errors.

Snapshot

```
491:        require(owner() == _msgSender(), "Ownable: caller is not the owner");
510:        require(newOwner != address(0), "Ownable: new owner is the zero address");
820:        require(currentAllowance >= subtractedValue, "ERC20: decreased allowance below zero");
843:        require(from != address(0), "ERC20: transfer from the zero address");
844:        require(to != address(0), "ERC20: transfer to the zero address");
849:        require(fromBalance >= amount, "ERC20: transfer amount exceeds balance");
872:        require(account != address(0), "ERC20: mint to the zero address");
898:        require(account != address(0), "ERC20: burn from the zero address");
903:        require(accountBalance >= amount, "ERC20: burn amount exceeds balance");
929:        require(owner != address(0), "ERC20: approve from the zero address");
930:        require(spender != address(0), "ERC20: approve to the zero address");
947:        require(currentAllowance >= amount, "ERC20: insufficient allowance");
1029:       require(amount <= totalSupply(), "Amount cannot be over the total supply.");
1041:       require(pair != uniswapV2Pair, "The Uniswap pair cannot be removed from automatedMarketMakerPairs");
1046:       require(automatedMarketMakerPairs[pair] != value, "Automated market maker pair is already set to that value");
1083:       require(recipient != address(0), "Zero address");
1088:       require(sender != address(0), "ERC20: transfer from the zero address");
1089:       require(recipient != address(0), "ERC20: transfer to the zero address");
```

APPENDIX

Auditing Approach and Methodologies applied

The Block Audit Report team has performed rigorous testing of the project including the analysis of the code design patterns where we reviewed the smart contract architecture to ensure it is structured along with the safe use of standard inherited contracts and libraries. Our team also conducted a formal line by line inspection of the Smart Contract i.e., a manual review, to find potential issues including but not limited to

- Race conditions
- Zero race conditions approval attacks
- Re-entrancy
- Transaction-ordering dependence
- Timestamp dependence
- Check-effects-interaction pattern (optimistic accounting)
- Decentralized denial-of-service attacks
- Secure ether transfer pattern
- Guard check pattern
- Fail-safe mode
- Gas-limits and infinite loops
- Call Stack depth

In the Unit testing Phase, we coded/conducted custom unit tests written against each function in the contract to verify the claimed functionality from our client. In Automated Testing, we tested the Smart Contract with our standard set of multifunctional tools to identify vulnerabilities and security flaws. The code was tested in collaboration of our multiple team members and this included but not limited to;

- Testing the functionality of the Smart Contract to determine proper logic has been followed throughout the whole process.
- Analyzing the complexity of the code in depth and in detail line-by-line manual review of the code.
- Deploying the code on testnet using multiple clients to run live tests.
- Analyzing failure preparations to check how the Smart Contract performs in case of any bugs and vulnerabilities.
- Checking whether all the libraries used in the code are on the latest version.
- Analyzing the security of the on-chain data.

Issue Categories:

Every issue in this report was assigned a severity level from the following:

Critical Severity Issues

Issues of Critical Severity leaves smart contracts vulnerable to major exploits and can lead to asset loss and data loss. These can have significant impact on the functionality/performance of the smart contract.
We recommend these issues must be fixed before proceeding to MainNet..

High Severity Issues

Issues of High Severity are not as easy to exploit but they might endanger the execution of the smart contract and potentially create crucial problems.
Fixing these issues is highly recommended before proceeding to MainNet.

Medium Severity Issues

Issues on this level are not a major cause of vulnerability to the smart contract, they cannot lead to data-manipulations or asset loss but may affect funtionality.
It is important to fix these issues before proceeding to MainNet.

Low Severity Issues

Issues at this level are very low in their impact on the overall functionality and execution of the smart contract. These are mostly code-level violations or improper formatting.
These issues can be remain unfixed or can be fixed at a later date if the code is redeployed or forked.

Informational Findings

These are finding that our team comes accross when manually reviewing a smart contract which are important to know for the owners as well as users of a contract.
These issues must be acknowledged by the owners before we publish our report.

Ownership Priviledges

Owner of a smart contract can include certain rights and priviledges while deploying a smart contract that might be hidden deep inside the codebase and may make the project vulnerable to rug-pulls or other types of scams.
We at BlockAudit believe in transparency and hence we showcase Ownership priviledges separately so the owner as well as the investors can get a better understanding about the project.

Gas Optimization

Solidity gas optimization is the process of lowering the cost of operating your Solidity smart code. The term "gas" refers to the level of processing power required to perform specific tasks on the Ethereum network.
Each Ethereum transaction costs a fee since it requires the use of computer resources. It will deduct a fee anytime any function in the smart contract is invoked by the contract's owner or users.

DISCLAIMER

This is a limited report on our findings based on our analysis, in accordance with good industry practice as at the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. In order to get a full view of our analysis, it is crucial for the client to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that the client should not rely on this report and cannot claim against us on the basis of what it says or doesn't say, or how we produced it, and it is important for the client to conduct the client's own independent investigations before making any decisions. We go into more detail on this in the below disclaimer below – please make sure to read it in full.

By reading this report or any part of it, the client agrees to the terms of this disclaimer. If the client does not agree to the terms, then please immediately cease reading this report, and delete and destroy any/all copies of this report downloaded and/or printed by the client. This report is provided for information purposes only and stays on a non-reliance basis, and does not constitute investment advice. No one/ NONE shall have any rights to rely on the report or its contents, and BlockAudit and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers and other representatives).

(BlockAudit) owes no duty of care towards the client or any other person, nor does BlockAudit claim any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties or other terms of any kind except as set out in this disclaimer, and BlockAudit hereby excludes all representations, warranties, conditions and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effects in relation to the report.

Except and only to the extent that it is prohibited by law, BlockAudit hereby excludes all liability and responsibility, and neither the client nor any other person shall have any claim against BlockAudit, for any amount or kind of loss or damage that may result to the client or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of use of this report, and any reliance on this report.

The analysis of the security is purely based on the received smart contracts alone. No related/third-party smart contracts, applications or operations were reviewed for security. No product code has been reviewed.

Note: The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the **BlocBite** team put a bug bounty program in place to encourage further analysis of the smart contracts by other third parties

About BlockAudit

BlockAudit is an industry leading security organisation that helps web3 blockchain based projects with their security and correctness of their smart-contracts. With years of experience we have a dedicated team that is capable of performing audits in a wide variety of languages including HTML, PHP, JS, Node, React, Native, Solidity, Rust and other Web3 frameworks for DApps, DeFi, GameFi and Metaverse platforms.

With a mission to make web3 a safe and secure place BlockAudit is commited to provide it's partners with a budget and investor friendly security Audit Report that will increase the value of their projects significantly.

 *www.blockaudit.report*

 *team@blockaudit.report*

 *@BlockAudit*

 *github.com/Block-Audit-Report*