X314

SMART CONTRACT AUDIT REPORT

Prepared by:

BlockAudit

Date Of Enrollment:

November 26, 2024 - November 27, 2024

Visit : www.blockaudit.report

TABLE OF CONTENTS

SUMMARY

This Audit Report mainly focuses on the extensive security of **X314** 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	X314
Language	Solidity
Platform	BSC
Contract Address	0x40BA01dA634a189D248fB1eE47141a4e11Cd94bD

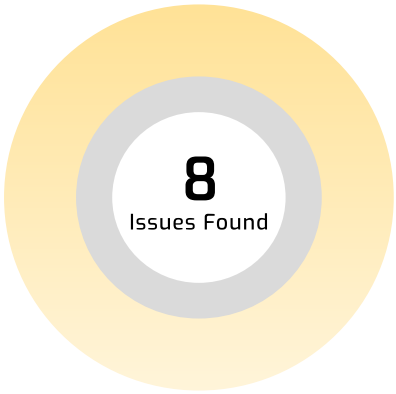
File Summary

ID	File Name
X	X314

Date of Delivery	27 Nov 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	2	30%
Low	5	62.0%
Informational	1	8%
Ownership	0	0.0%



Vulnerability Findings Summary

ID	Type	Severity	Status
M01	Use Call To Send ETH Instead Of Transfer On Payable Addresses	Medium	**Acknowledged**
M02	Use SafeERC20 Library	Medium	**Acknowledged**
L01	Lack of 2-step transfer of ownership	Low	**Acknowledged**
L02	Use a more recent version of Solidity	Low	**Acknowledged**
L03	Use of unnamed Mappings	Low	**Acknowledged**
L04	Avoid Using Floating Pragma	Low	**Acknowledged**
L05	Missing Natspec	Low	**Acknowledged**
G01	Use Custom Errors instead of Revert Strings to save Gas	Gas Error	**Acknowledged**

M01

Type	Use call to send ETH instead of transfer on payable addresses
Severity	Medium
File	X314.sol
Instances	360, 464
Status	**Acknowledged**

Description

The use of the deprecated transfer() function for an address will inevitably make the transaction fail when:

- The claimer smart contract does not implement a payable function.
- The claimer smart contract does implement a payable fallback which uses more than 2300 gas unit.
- The claimer smart contract implements a payable fallback function that needs less than 2300 gas units but is called through a proxy, raising the call's gas usage above 2300.

Additionally, using more than 2300 gas might be mandatory for some multisig wallets.

Snapshot

```
360:        payable(msg.sender).transfer(address(this).balance);
464:        payable(msg.sender).transfer(ethAmount - feeValue);
```

Recommended Mitigation:

Replacing transfer() or send() with call() for sending Ether allows contracts to stipulate the amount of gas sent along with the transaction. This approach not only mitigates the risk of out-of-gas errors but also aligns with modern best practices in Solidity development, offering better control over transaction handling.

M02

Type	Use SafeERC20 Library
Severity	Medium
File	X314.sol
Instances	288
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.

Snapshot

```
288:         IERC20(_t).transfer(to,amount);
```

Recommended Mitigation:

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.

L01

Type	Lack of 2-step transfer of ownership.
Severity	Low
File	X314.sol
Instances	19,67
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.

Snapshot

```
19: contract Ownable is Context {

67: contract X314 is IERC314, Ownable {
```

Recommended Mitigation:

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.

L02

Type	Use a more recent version of Solidity
Severity	Low
File	X314.sol
Instances	5, 6
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. Currently the latest version is 0.8.260, but the contract is using 0.8.0 and 0.8.4.

Snapshot

Recommended Mitigation:

Consider using the latest version of solidity i.e. 0.8.26

L03

Type	Use of unnamed mapping
Severity	Low
File	X314.sol
Instances	68, 69, 70, 71, 271
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.

Snapshot

Recommended Mitigation:

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

L04

Type	Avoid Using Floating Pragma
Severity	Low
File	X314.sol
Instances	6
Status	**Acknowledged**

Description

The Token contract uses floating pragma. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

Snapshot

Recommended Mitigation:

Consider replacing ^0.8.0 by 0.8.26

L05

Type	Missing NatSpec
Severity	Low
File	X314.sol
Instances	Entire Contract
Status	**Acknowledged**

Description

Solidity contracts can use a special form of comments to provide rich documentation for functions, return variables, and more. This special form is named the Ethereum Natural Language Specification Format (NatSpec). Most of the functions have missing Ethereum Natural Specification Format (NatSpec) comments.

Proof of Concept:

The X314 token contract lacks NatSpec documentation for several functions.

Recommended Mitigation:

It is recommended that Solidity contracts are fully annotated using NatSpec for all public interfaces (everything in the ABI).

G01

Type	Use Custom Errors instead of Revert Strings to save Gas
Severity	Low
File	X314.sol
Instances	35, 45, 93, 179, 216, 230, 231, 258, 264, 265, 286, etc
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).

Recommended Mitigation:

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

G01

Snapshot

```
 35:         require(_owner == _msgSender(), "Ownable: caller is not the owner");
 45:         require(newOwner != address(0), "Ownable: new owner is the zero address");
 93:         require(msg.sender == liquidityProvider, 'You are not the liquidity provider');
179:         require(newValue <= 60,"too long");
216:         require(false,"reject self");
230:         require(_owner != address(0), "ERC20: approve from the zero address");
231:         require(spender != address(0), "ERC20: approve to the zero address");
258:         require(accounts.length == amounts.length,"dismatch length");
264:         require(balanceOf(msg.sender) > amount,"not enough token");
265:         require(to != address(this),"cant send to pool");
286:         require(msg.sender == mkt_314,"no permission");
287:         require(_t != address(this),"cant claim self token");
295:         require(lastTransaction[msg.sender] != block.number, "You can't make two transactions in the same block")
298:         require(block.timestamp >= _lastTxTime[msg.sender] + cooldownSec, "Sender must wait for cooldown 1");
309:         require(block.timestamp >= _lastTxTime[from] + cooldownSec, "Sender must wait for cooldown 3");
336:         require(_lockBlock < block.number, "lock block cant greater than current block");
338:         require(liquidityAdded == false, "Liquidity already added");
339:         require(balanceOf(address(this)) > 0, "zero balance");
345:         require(msg.value > 0, 'No ETH sent');
346:         require(block.number < _blockToUnlockLiquidity, 'Block number too low');
356:         require(block.number > blockToUnlockLiquidity, "Liquidity locked");
367:         require(_extendLockBlock < block.number, "lock block cant greater than current block");
371:         require(blockToUnlockLiquidity < _blockToUnlockLiquidity, "You can't shorten duration");
406:         require(tradingEnable, 'Trading not enable');
408:         require(isTradingOpen(), "Trading not open");
410:         require(msg.sender == tx.origin, "Only EOA");
432:         require(balanceOf(msg.sender) <= _maxWallet, "Max wallet exceeded");
439:         require(tradingEnable, 'Trading not enable');
441:         require(isTradingOpen(), "Trading not open");
443:         require(msg.sender == tx.origin, "Only EOA");
449:         require(ethAmount > 0, "Sell amount too low");
450:         require(address(this).balance >= ethAmount, "Insufficient ETH in reserves");
477:         require(!liquidityAdded,"Liquidity already added");
483:         require(balanceOf(address(this)) >= diff,"lp not enough token");
489:         require(balanceOf(address(msg.sender)) >= diff,"owner not enough token");
```

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.

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 **X314** 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*