Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project:	Binance-Peg Dogecoin Token (DOGE)
Platform:	Binance Smart Chain
Language:	Solidity
Date:	March 4th, 2026

Table of contents

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.

Email: audit@EtherAuthority.io

THIS IS SECURITY AUDIT REPORT DOCUMENT AND WHICH MAY CONTAIN INFORMATION WHICH IS CONFIDENTIAL. WHICH INCLUDES ANY POTENTIAL VULNERABILITIES AND MALICIOUS CODES WHICH CAN BE USED TO EXPLOIT THE SOFTWARE. THIS MUST BE REFERRED INTERNALLY AND ONLY SHOULD BE MADE AVAILABLE TO THE PUBLIC AFTER ISSUES ARE RESOLVED.

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

Introduction

As part of EtherAuthority's community smart contracts audit initiatives, the smart contracts of Binance-Peg Dogecoin Token (DOGE) were audited. The audit has been performed using manual analysis as well as using automated software tools. This report presents all the findings regarding the audit performed on March 4th, 2026.

The purpose of this audit was to address the following:

- Ensure that all claimed functions exist and function correctly.
- Identify any security vulnerabilities that may be present in the smart contract.

Project Background

This project implements an upgradeable proxy architecture for BEP20/ERC20 smart contracts using the transparent proxy pattern. It is built on standardized proxy contracts provided by OpenZeppelin to ensure security, reliability, and industry best practices.

The architecture separates contract logic from storage, allowing the underlying implementation to be upgraded without redeploying the proxy contract or losing state. The core Proxy contract forwards all external calls to the current implementation using the delegatecall opcode.

The UpgradeableProxy contract stores the implementation address in a standardized EIP-1967 storage slot, preventing storage collisions with the implementation contract. The TransparentUpgradeableProxy extends this functionality by introducing an admin-controlled upgrade mechanism.

Only the designated admin can perform upgrades or modify proxy configurations, ensuring controlled and secure contract evolution. Regular users interact seamlessly with the implementation through fallback delegation, without being aware of the proxy layer.

To prevent function selector conflicts, the transparent proxy pattern ensures that admin-only functions are inaccessible to regular users and that admin accounts cannot accidentally invoke implementation logic through the proxy.

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

Additionally, the integrated Address library provides safe low-level call handling and contract verification. The system also supports initialization during deployment via encoded function calls, mimicking constructor behavior for upgradeable contracts.

Event emissions such as Upgraded and AdminChanged enhance transparency and traceability of administrative actions. The BEP20UpgradeableProxy serves as a lightweight wrapper tailored for BEP20 token implementations.

Overall, this design provides a secure, flexible, and maintainable framework for building upgradeable smart contract systems while preserving state and ensuring separation of concerns.

Audit scope

Name	**Code Review and Security Analysis Report for Binance-Peg Dogecoin Token (DOGE) Smart Contract**
Platform	**Binance Smart Chain**
Language	**Solidity**
File	BEP20UpgradeableProxy.sol
Smart Contract Code	0x8ac76a51cc950d9822d68b83fe1ad97b32cd580d
Audit Date	March 4th, 2026

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Key features: • Upgradeable smart contract architecture • Transparent proxy pattern (admin/user separation) • EIP-1967 compliant storage slots • Secure delegate call execution • Admin-controlled upgrades (upgradeTo, upgradeToAndCall) • Fallback & receive delegation • Implementation contract replaceability • State persistence across upgrades • Contract address validation for upgrades • BEP20/ERC20 compatible proxy deployment	**YES, This is valid.**

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

Audit Summary

According to the standard audit assessment, the Customer`s solidity-based smart contracts are **“Secured”**. Also, these contracts contain owner control, which does not make them fully decentralized.

We used various tools like Slither, Solhint, and Remix IDE. At the same time, this finding is based on a critical analysis of the manual audit.
All issues found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the Audit Overview section. The general overview is presented in the AS-IS section and all identified issues can be found in the Audit overview section.

We found 0 critical, 0 high, 0 medium 0 low, and 2 very low level issues.

Investor Advice: A technical audit of the smart contract does not guarantee the ethical nature of the project. Any owner-controlled functions should be executed by the owner with responsibility. All investors/users are advised to do their due diligence before investing in the project.

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

Technical Quick Stats

Main Category	Subcategory	Result
Contract Programming	The solidity version is not specified	Passed
	The solidity version is too old	Moderated
	Integer overflow/underflow	Passed
	Function input parameters lack check	Passed
	Function input parameters check bypass	Passed
	Function access control lacks management	Passed
	Critical operation lacks event log	Passed
	Human/contract checks bypass	Passed
	Random number generation/use vulnerability	N/A
	Fallback function misuse	Passed
	Race condition	Passed
	Logical vulnerability	Passed
	Features claimed	Passed
	Other programming issues	Moderated
Code Specification	Function visibility not explicitly declared	Passed
	Var. storage location not explicitly declared	Passed
	Use keywords/functions to be deprecated	Passed
	Unused code	Passed
Gas Optimization	“Out of Gas” Issue	Passed
	High consumption ‘for/while’ loop	Passed
	High consumption ‘storage’ storage	Passed
	Assert() misuse	Passed
Business Risk	The maximum limit for mintage is not set	Passed
	“Short Address” Attack	Passed
	“Double Spend” Attack	Passed

Overall Audit Result: **PASSED**

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

Code Quality

This audit scope has 1 smart contract. Smart contracts contain Libraries, Smart contracts, inherits, and Interfaces. This is a compact and well-written smart contract.

The libraries in Binance-Peg Dogecoin Token (DOGE) are part of its logical algorithm. A library is a different type of smart contract that contains reusable code. Once deployed on the blockchain (only once), it is assigned a specific address and its properties/methods can be reused many times by other contracts in the Binance-Peg Dogecoin Token (DOGE).

The EtherAuthority team has no scenario and unit test scripts, which would have helped to determine the integrity of the code in an automated way.

Code parts are well commented on in the smart contracts. Ethereum's NatSpec commenting style is recommended.

Documentation

We were given Binance-Peg Dogecoin Token (DOGE) smart contract code in the form of a bscscan web link.

As mentioned above, code parts are well commented on. and the logic is straightforward. So it is easy to quickly understand the programming flow as well as complex code logic. Comments are very helpful in understanding the overall architecture of the protocol.

Use of Dependencies

As per our observation, the libraries are used in this smart contract infrastructure that are based on well-known industry standard open-source projects.

Apart from libraries, its functions are not used in external smart contract calls.

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

AS-IS overview

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	ifAdmin	modifier	Passed	No Issue
3	admin	external	ifAdmin	No Issue
4	implementation	external	ifAdmin	No Issue
5	changeAdmin	external	ifAdmin	No Issue
6	upgradeTo	external	ifAdmin	No Issue
7	upgradeToAndCall	external	ifAdmin	No Issue
8	_admin	internal	Passed	No Issue
9	_setAdmin	write	Passed	No Issue
10	_beforeFallback	internal	Passed	No Issue

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to token loss etc.
High	High-level vulnerabilities are difficult to exploit; however, they also have significant impact on smart contract execution, e.g. public access to crucial
Medium	Medium-level vulnerabilities are important to fix; however, they can't lead to tokens lose
Low	Low-level vulnerabilities are mostly related to outdated, unused etc. code snippets, that can't have significant impact on execution
Lowest / Code Style / Best Practice	Lowest-level vulnerabilities, code style violations and info statements can't affect smart contract execution and can be ignored.

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

Audit Findings

Critical Severity

No Critical severity vulnerabilities were found.

High Severity

No High severity vulnerabilities were found.

Medium

No Medium severity vulnerabilities were found.

Low

No Low severity vulnerabilities were found.

Very Low / Informational / Best practices:

(1) Missing Validation for Implementation Address:

Description:

The _setImplementation() function only checks whether the address is a contract using isContract, but does not prevent setting critical or unintended contract addresses (e.g., malicious logic contracts).

Recommendation:

Add additional validation such as access control checks (already partially present) and consider implementing a whitelist or timelock mechanism before upgrading implementations.

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

(2) Outdated Solidity Version

Description:

Contract uses Solidity v0.6.0

Recommendation:

- Upgrade to Solidity ≥0.8.x
- Remove SafeMath (built-in checks available)

Centralization

This smart contract has some functions which can be executed by the Admin (Owner) only. If the admin wallet's private key would be compromised, then it would create trouble. The following are Admin functions:

BEP20UpgradeableProxy.sol

- **admin()** → Returns the current admin address (only callable by admin)
- **implementation()** → Returns the current implementation contract address
- **changeAdmin(address newAdmin)** → Updates the proxy admin to a new address
- **upgradeTo(address newImplementation)** → Upgrades the proxy to a new implementation contract
- **upgradeToAndCall(address newImplementation, bytes data)** → Upgrades implementation and executes a function call in one transaction

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

Conclusion

We were given a contract code in the form of bscscan web links. And we have used all possible tests based on given objects as files. We observed 2 Informational issues in the smart contracts. but those are not critical. So, **it's good to go for the production**.

Since possible test cases can be unlimited for such smart contracts protocol, we provide no such guarantee of future outcomes. We have used all the latest static tools and manual observations to cover the maximum possible test cases to scan everything.

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. Smart Contract's high-level description of functionality was presented in the As-is overview section of the report.

The audit report contains all found security vulnerabilities and other issues in the reviewed code.

The security state of the reviewed smart contract, based on standard audit procedure scope, is **"Secured".**

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of the systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review:

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis:

Our audit techniques included manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's web site to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

Documenting Results:

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, and then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this, we analyze the feasibility of an attack in a live system.

Suggested Solutions:

We search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

Disclaimers

EtherAuthority.io Disclaimer

EtherAuthority team has analyzed this smart contract in accordance with the best industry practices at the date of this report, in relation to: cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment and functionality (performing the intended functions).

Due to the fact that the total number of test cases is unlimited, the audit makes no statements or warranties on the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only. We also suggest conducting a bug bounty program to confirm the high level of security of this smart contract.

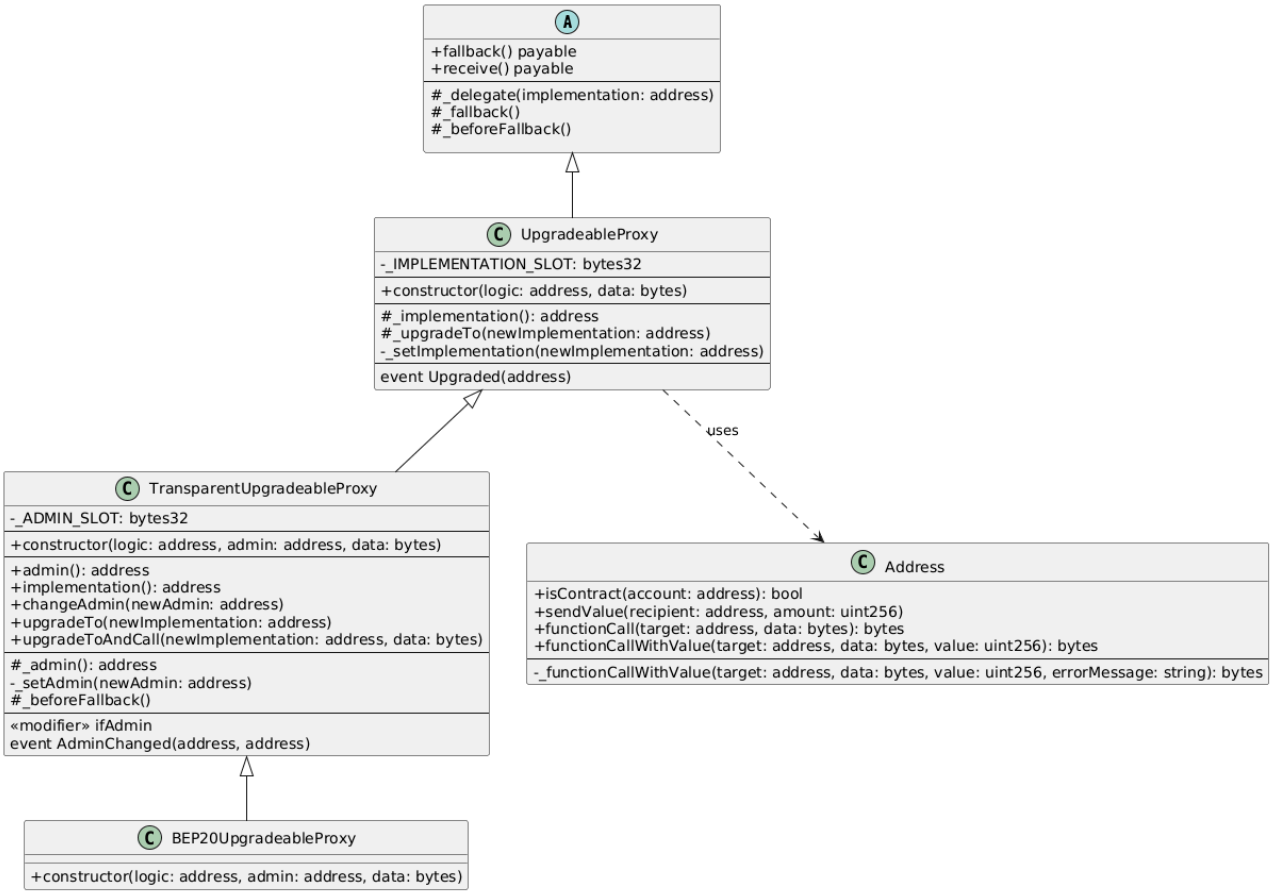
Technical Disclaimer

Smart contracts are deployed and executed on the blockchain platform. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to hacks. Thus, the audit can't guarantee the explicit security of the audited smart contracts.

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

Appendix

Code Flow Diagram - Binance-Peg Dogecoin Token (DOGE)



This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

Slither Results Log

Slither is a Solidity static analysis framework that uses vulnerability detectors, displays contract details, and provides an API for writing custom analyses. It helps developers identify vulnerabilities, improve code comprehension, and prototype custom analyses quickly. The analysis includes a report with warnings and errors, allowing developers to quickly prototype and fix issues.

We did the analysis of the project altogether. Below are the results.

Slither Log >> BEP20UpgradeableProxy.sol

```
INFO:Detectors:
TransparentUpgradeableProxy.upgradeTo(address)
(TransparentUpgradeableProxy.sol#96-100) allows admin-controlled
implementation upgrade
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#control
led-delegatecall
INFO:Detectors:
UpgradeableProxy._setImplementation(address)
(UpgradeableProxy.sol#63-75) uses low-level storage write (sstore)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#assembl
y-usage
INFO:Detectors:
Proxy._delegate(address)
(Proxy.sol#17-41) uses delegatecall inside inline assembly
Impact:
Delegated contract can fully control proxy storage.
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#low-lev
el-calls
INFO:Detectors:
TransparentUpgradeableProxy.upgradeToAndCall(address,bytes)
(TransparentUpgradeableProxy.sol#102-110) performs delegatecall with
arbitrary data
Impact:
Risk of malicious initialization logic execution.
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#delegat
ecall
INFO:Detectors:
TransparentUpgradeableProxy.changeAdmin(address)
(TransparentUpgradeableProxy.sol#86-92) missing zero-address check
for previous admin logging
Impact:
Event logs may not fully reflect secure admin transitions.
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#events-
access
```

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

```
INFO:Detectors:
Address.isContract(address)
(Address.sol#14-26) relies on extcodesize
Impact:
Returns false for contracts in construction.
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-equality
INFO:Detectors:
Address.sendValue(address,uint256)
(Address.sol#28-42) forwards all gas using call
Impact:
Potential reentrancy if not protected.
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities
INFO:Detectors:
Proxy.fallback() / receive()
(Proxy.sol#52-64) unrestricted fallback delegatecall
Impact:
All unknown calls are forwarded to implementation.
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#fallback-function
INFO:Detectors:
TransparentUpgradeableProxy._beforeFallback()
(TransparentUpgradeableProxy.sol#138-143) restricts admin fallback
Status:
✔ Proper protection implemented
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#access-control
INFO:Detectors:
UpgradeableProxy.constructor(address,bytes)
(UpgradeableProxy.sol#17-30) performs delegatecall during initialization
Impact:
Initialization logic execution risk if input is malicious.
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#constructor-calls
```

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

Solidity Static Analysis

Static code analysis is used to identify many common coding problems before a program is released. It involves examining the code manually or using tools to automate the process. Static code analysis tools can automatically scan the code without executing it.

BEP20UpgradeableProxy.sol

```
Check-effects-interaction:
Potential violation of Checks-Effects-Interaction pattern in
Address._functionCallWithValue(address,bytes,uint256,string): Could
potentially lead to re-entrancy vulnerability. Note: Modifiers are
currently not considered by this static analysis.
Pos: 206:4:

Inline assembly:
The Contract uses inline assembly, this is only advised in rare
cases. Additionally static analysis modules do not parse inline
Assembly, this can lead to wrong analysis results.
Pos: 219:16:

Low level calls:
Use of "call": should be avoided whenever possible. It can lead to
unexpected behavior if the return value is not handled properly.
Please use Direct Calls via specifying the called contract's
interface.
Pos: 144:27:

Gas costs:
Fallback function of contract BEP20UpgradeableProxy requires too much
gas (infinite). If the fallback function requires more than 2300 gas,
the contract cannot receive Ether.
Pos: 65:4:

Gas costs:
Fallback function of contract BEP20UpgradeableProxy requires too much
gas (infinite). If the fallback function requires more than 2300 gas,
the contract cannot receive Ether.
Pos: 73:4:

No return:
Proxy._implementation(): Defines a return type but never explicitly
returns a value.
Pos: 49:4:

Constant/View/Pure functions:
TransparentUpgradeableProxy._admin() : Is constant but potentially
should not be. Note: Modifiers are currently not considered by this
static analysis.
Pos: 437:4:
```

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

```
Guard conditions:
Use "assert(x)" if you never ever want x to be false, not in any
circumstance (apart from a bug in your code). Use "require(x)" if x
can be false, due to e.g. invalid input or a failing external
component.
Pos: 406:8:
```

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

Solhint Linter

Linters are the utility tools that analyze the given source code and report programming errors, bugs, and stylistic errors. For the Solidity language, there are some linter tools available that a developer can use to improve the quality of their Solidity contracts.

BEP20UpgradeableProxy.sol

```
Found more than One contract per file. 5 contracts found!
Pos: 1:4
Visibility modifier must be first in list of modifiers
Pos: 25:64
Visibility modifier must be first in list of modifiers
Pos: 24:72
Code contains empty blocks
Pos: 49:82
Use Custom Errors instead of require statements
Pos: 9:140
Error message for require is too long: 58 counted / 32 allowed
Pos: 9:144
Use Custom Errors instead of require statements
Pos: 9:144
Error message for require is too long: 38 counted / 32 allowed
Pos: 9:201
Use Custom Errors instead of require statements
Pos: 9:201
Use Custom Errors instead of require statements
Pos: 9:206
Provide an error message for require
Pos: 13:258
Use Custom Errors instead of require statements
Pos: 13:258
Error message for require is too long: 54 counted / 32 allowed
Pos: 9:299
```

Software analysis result:

These software reported many false positive results and some are informational issues. So, those issues can be safely ignored.

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.

Email: audit@EtherAuthority.io