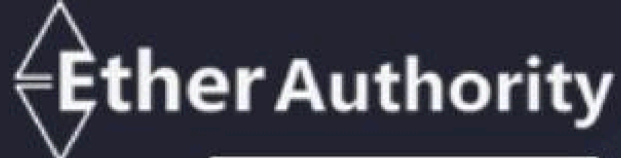


etherauthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project: Pyth Network
Website: pyth.network
Platform: Base Chain Network
Language: Solidity
Date: June 1st, 2024

Table of contents

Introduction 4
Project Background 4
Audit Scope 5
Code Audit History 6
Severity Definitions 6
Claimed Smart Contract Features 7
Audit Summary 8
Technical Quick Stats 9
Business Risk Analysis 10
Code Quality 11
Documentation 11
Use of Dependencies 11
AS-IS overview 12
Audit Findings 13
Conclusion 15
Our Methodology 16
Disclaimers 18
Appendix

- Code Flow Diagram 19
- Slither Results Log 20
- Solidity static analysis 21
- Solhint Linter 22

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.

Introduction

As part of EtherAuthority's community smart contracts audit initiatives, the Pyth Network smart contract from pyth.network was audited extensively. 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 June 1st, 2024.

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

Website Details

Pyth Network provides real-time market data for smart contracts, covering cryptocurrencies, equities, forex, and commodities. It integrates with over 50 blockchains and is supported by major financial institutions like Jane Street and CBOE.
Pyth uses a pull oracle model, allowing applications to request data on-demand, ensuring precise and timely updates. It offers robust APIs for seamless integration, enabling developers to access accurate and low-latency market data. Pyth is used by various DeFi applications for secure and reliable data feeds.

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 Details

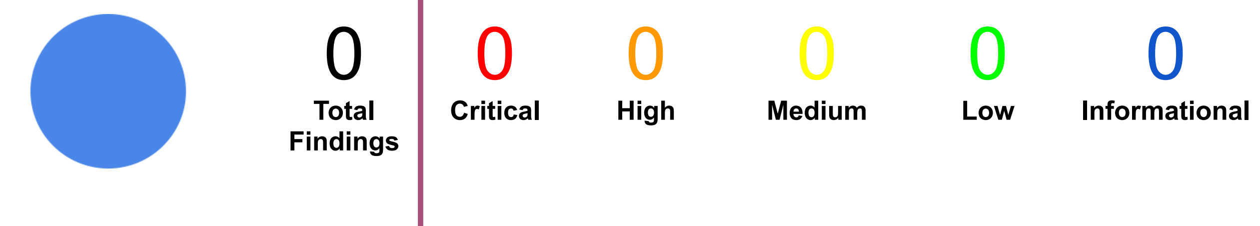
- The provided Solidity code is a comprehensive implementation of a proxy contract system. It includes:
  - **StorageSlot Library:** Defines structs for different data types and provides functions to access storage slots.
  - **Address Library:** Offers utilities for address-related functions, such as checking if an address is a contract and performing low-level calls.
  - **Proxy Contract:** An abstract contract for delegating calls to an implementation contract.
  - **ERC1967Upgrade:** Provides functions for upgrading and managing the implementation and admin addresses according to the EIP-1967 standard.
  - **BeaconProxy:** A proxy contract that delegates calls to an implementation returned by a beacon contract.
  - **BridgeToken:** A concrete implementation of the BeaconProxy contract.
- This structure allows for upgradable proxy contracts, enabling changes to implementation logic without altering the proxy address.

Audit scope

Name	**Code Review and Security Analysis Report for Pyth Network Smart Contract**
Platform	**Base Chain Network**
Language	**Solidity**
File	BridgeToken.sol
Smart Contract Code	0x4c5d8A75F3762c1561D96f177694f67378705E98
Audit Date	June 1st,2024
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 Audit History



Severity Definitions

	Severity	Definition
0	**Critical**	Critical vulnerabilities are usually straightforward to exploit and can lead to token loss etc.
0	**High**	High-level vulnerabilities are difficult to exploit; however, they also have significant impact on smart contract execution, e.g. Public access is crucial.
0	**Medium**	Medium-level vulnerabilities are important to fix; however, they can't lead to tokens lose
0	**Low**	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets, that can't have a significant impact on execution
0	**Lowest / Informational / 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

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Ownership Control: • There are no owner functions, which makes it 100% decentralized.	**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.

Audit Summary

According to the standard audit assessment, the Customer`s solidity-based smart contracts are **"Well Secured"**.This token contract does not have any ownership control, hence it is 100% decentralized.

Unsecured	Poor Secured	Secured	Well Secured

You are here

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 0 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	Passed
	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	Passed
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

Business Risk Analysis

Category	Result
Buy Tax	0%
Sell Tax	0%
Cannot Buy	No
Cannot Sell	No
Max Tax	0%
Modify Tax	No
Fee Check	Not Detected
Is Honeypot	Not Detected
Trading Cooldown	Not Detected
Can Pause Trade?	Not Detected
Pause Transfer?	No
Max Tax?	No
Is it Anti-whale?	Not Detected
Is Anti-bot?	Not Detected
Is it a Blacklist?	No
Blacklist Check	No
Can Mint?	No
Is it a Proxy Contract?	Yes
Is it used Open Source?	No
External Call Risk?	No
Balance Modifiable?	No
Can Take Ownership?	No
Ownership Renounce?	No
Hidden Owner?	Not Detected
Self Destruction?	Not Detected
Auditor Confidence	High

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 Pyth Network 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 Pyth Network.

The EtherAuthority team has not provided 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 a Pyth Network smart contract code in the form of a basescan 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 used in this smart contract infrastructure 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

BridgeToken.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	_beacon	internal	Passed	No Issue
3	_implementation	internal	Passed	No Issue
4	_setBeacon	internal	Passed	No Issue
5	_getImplementation	internal	Passed	No Issue
6	_setImplementation	write	Passed	No Issue
7	_upgradeTo	internal	Passed	No Issue
8	_upgradeToAndCall	internal	Passed	No Issue
9	_upgradeToAndCallSecure	internal	Passed	No Issue
10	_getAdmin	internal	Passed	No Issue
11	_setAdmin	write	Passed	No Issue
12	_changeAdmin	internal	Passed	No Issue
13	_getBeacon	internal	Passed	No Issue
14	_setBeacon	write	Passed	No Issue
15	_upgradeBeaconToAndCall	internal	Passed	No Issue
16	_delegate	internal	Passed	No Issue
17	_implementation	internal	Passed	No Issue
18	_fallback	internal	Passed	No Issue
19	fallback	external	Passed	No Issue
20	receive	external	Passed	No Issue
21	_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

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:

No Very-Low-severity vulnerabilities were found.

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

Centralization

The Pyth Network (PYTH) Token smart contract does not have any ownership control, **hence it is 100% decentralized.**

Therefore, there is **no** centralization risk.

Centralized | Decentralized

You are here

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 a basescan web link. And we have used all possible tests based on given objects as files. We observed no issue in the smart contracts. 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.

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 **"Well 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 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, 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.

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).

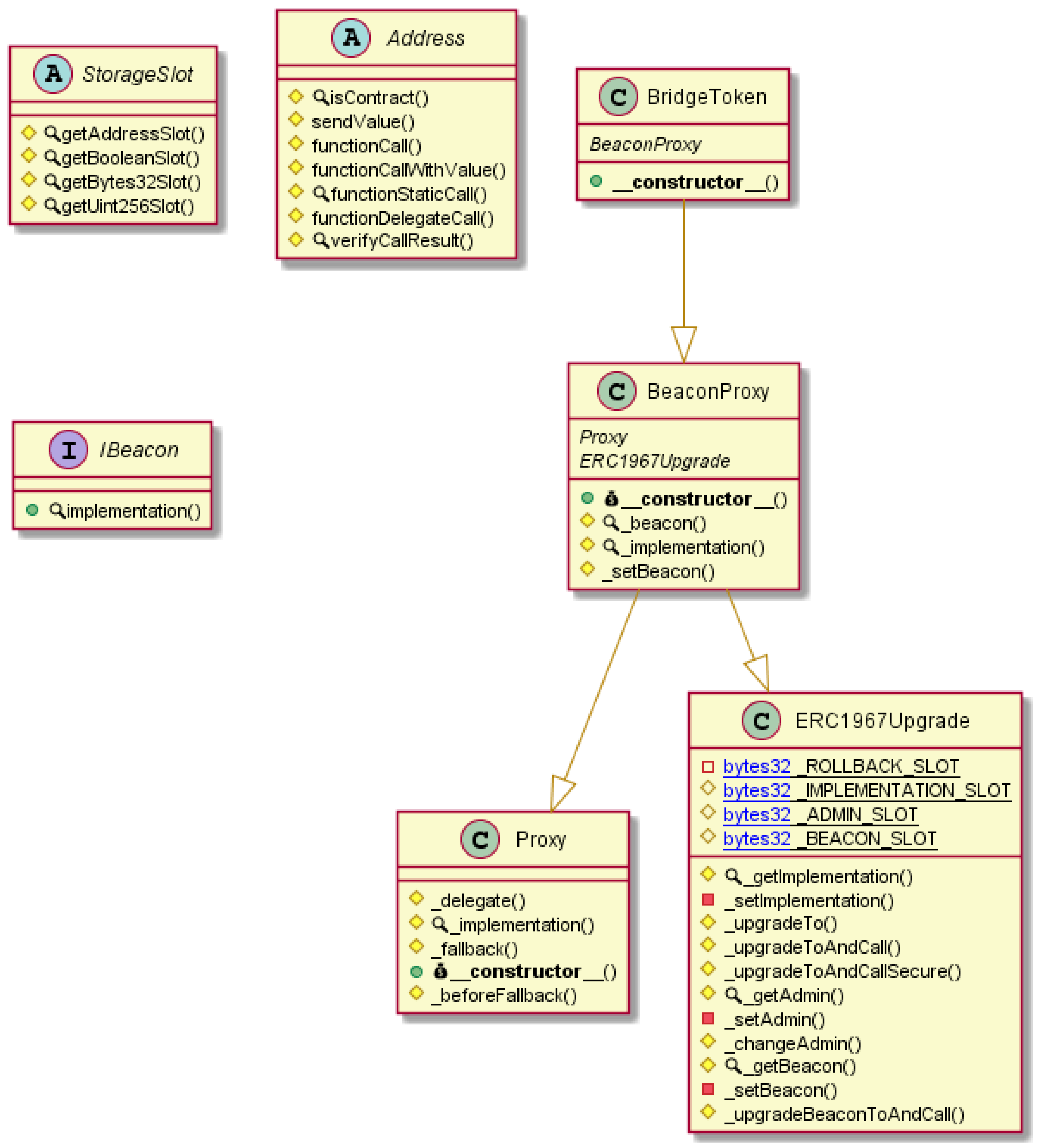
Due to the fact that the total number of test cases is unlimited, the audit makes no statements or warranties on the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. While 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 explicit security of the audited smart contracts.

Appendix

Code Flow Diagram - Pyth Network



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.

BridgeToken.sol

```
INFO:Detectors:
Pragma version^0.8.0 (BridgeToken.sol#3) allows old versions
solc-0.8.25 is not recommended for deployment
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in Address.sendValue(address,uint256) (BridgeToken.sol#105-110):
        - (success) = recipient.call{value: amount}() (BridgeToken.sol#108)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string)
(BridgeToken.sol#173-184):
        - (success,returndata) = target.call{value: value}(data) (BridgeToken.sol#182)
Low level call in Address.functionStaticCall(address,bytes,string) (BridgeToken.sol#202-211):
        - (success,returndata) = target.staticcall(data) (BridgeToken.sol#209)
Low level call in Address.functionDelegateCall(address,bytes,string) (BridgeToken.sol#229-238):
        - (success,returndata) = target.delegatecall(data) (BridgeToken.sol#236)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Slither:BridgeToken.sol analyzed (7 contracts with 93 detectors), 38 result(s) found
```

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.

BridgeToken.sol

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: 287:8:

Low level calls:
Use of "delegatecall": should be avoided whenever possible. External code, that is called can change the state of the calling contract and send ether from the caller's balance. If this is wanted behaviour, use the Solidity library feature if possible.
Pos: 236:50:

Gas costs:
Fallback function of contract BeaconProxy requires too much gas (infinite). If the fallback function requires more than 2300 gas, the contract cannot receive Ether.
Pos: 331:4:

No return:
Proxy._implementation(): Defines a return type but never explicitly returns a value.
Pos: 315:4:

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: 551:8:

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.

BridgeToken.sol

```
Compiler version ^0.8.0 does not satisfy the ^0.5.8 semver requirement
Pos: 1:2
Avoid to use inline assembly. It is acceptable only in rare cases
Pos: 9:52
Avoid to use inline assembly. It is acceptable only in rare cases
Pos: 9:82
Error message for require is too long
Pos: 9:206
Error message for require is too long
Pos: 9:233
Avoid to use low level calls.
Pos: 51:235
Avoid to use inline assembly. It is acceptable only in rare cases
Pos: 17:257
Avoid to use inline assembly. It is acceptable only in rare cases
Pos: 9:286
Code contains empty blocks
Pos: 49:348
Error message for require is too long
Pos: 9:386
Error message for require is too long
Pos: 13:445
Error message for require is too long
Pos: 9:474
```

Software analysis result:

This software reported many false positive results and some were 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