Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project: Telegraph
Platform: Ethereum
Language: Solidity
Date: October 24th, 2022

Table of contents

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.

Email: audit@EtherAuthority.io

Introduction

EtherAuthority was contracted by Telegraph protocol to perform the Security audit of the Telegraph protocol smart contracts code. 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 October 24th, 2022.

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

Telegraph is a platform where anyone can become a node by paying fees. Fee is an ERC20 token.

Audit scope

Name	**Code Review and Security Analysis Report for Telegraph Protocol Smart Contracts**
Platform	**Ethereum / Solidity**
File 1	Telegraph.sol
File 1 MD5 Hash	138A457D6BFA2408052EE5268F1FBB1A
File 2	PortContract.sol
File 2 MD5 Hash	8185CC78831CD45D8D3485E7B5EED51E
Audit Date	October 24th, 2022

Email: audit@EtherAuthority.io

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
File 1 PortContract.sol ● Name: Telegraph ● Symol: MSG ● Decimals: 18 ● These contracts can set below addresses and values: ○ Set the threshold ○ Entrance fees ○ Bridge address ○ Chain identification ○ Contract status ○ Fee Destinations ○ Distribution Contract ○ Price Mapping	**YES, This is valid.**

Audit Summary

According to the standard audit assessment, Customer`s solidity smart contracts are **“Secured”**. Also, these contracts do 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 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. General overview is presented in AS-IS section and all identified issues can be found in the Audit overview section.

We found 0 critical, 0 high, 0 medium and 3 low and some very low level issues.

Investors Advice: 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.

Technical Quick Stats

Main Category	Subcategory	Result
Contract Programming	Solidity version not specified	Passed
	Solidity version too old	Passed
	Integer overflow/underflow	Passed
	Function input parameters lack of check	Moderated
	Function input parameters check bypass	Passed
	Function access control lacks management	Passed
	Critical operation lacks event log	Moderated
	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	Moderated
Gas Optimization	“Out of Gas” Issue	Passed
	High consumption ‘for/while’ loop	Moderated
	High consumption ‘storage’ storage	Passed
	Assert() misuse	Passed
Business Risk	The maximum limit for mintage not set	Passed
	“Short Address” Attack	Passed
	“Double Spend” Attack	Passed

Overall Audit Result: **PASSED**

Code Quality

This audit scope has 2 smart contract files. Smart contracts contain Libraries, Smart contracts, inherits and Interfaces. This is a compact and well written smart contract.

The libraries in Telegraph Protocol 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 Telegraph Protocol.

The Telegraph team has provided unit test scripts, which would have helped to determine the integrity of the code in an automated way.

All code parts are not well commented on smart contracts.

Documentation

We were given a Telegraph smart contract code in the form of an etherscan weblink. The hash of that code is mentioned above in the table.

As mentioned above, code parts are not well commented. So it is not 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 contracts infrastructure that are based on well known industry standard open source projects.

Apart from libraries, its functions are used in external smart contract calls.

AS-IS overview

Telegraph.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	fallback	external	Passed	No Issue

PortContract.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	ERCProxyConstructor	internal	Passed	No Issue
3	name	read	Passed	No Issue
4	symbol	read	Passed	No Issue
5	decimals	read	Passed	No Issue
6	totalSupply	read	Passed	No Issue
7	balanceOf	read	Passed	No Issue
8	transfer	write	Passed	No Issue
9	allowance	read	Passed	No Issue
10	approve	write	Passed	No Issue
11	transferFrom	write	Passed	No Issue
12	increaseAllowance	write	Passed	No Issue
13	decreaseAllowance	write	Passed	No Issue
14	_transfer	internal	Passed	No Issue
15	_mint	internal	Passed	No Issue
16	_burn	internal	Passed	No Issue
17	_approve	internal	Passed	No Issue
18	_beforeTokenTransfer	internal	Passed	No Issue
19	setOwnableConstructor	internal	Passed	No Issue
20	owner	read	Passed	No Issue
21	onlyOwner	modifier	Passed	No Issue
22	renounceOwnership	write	access only Owner	No Issue
23	transferOwnership	write	access only Owner	No Issue
24	updateCodeAddress	internal	Passed	No Issue
25	proxiableUUID	write	Passed	No Issue
26	proxyConstructor	write	Duplicate Code, Function input parameters lack of check, Anyone can initialize contract	Refer Audit Findings
27	updateCode	write	Function input parameters lack of check	Refer Audit Findings
28	receive	external	Passed	No Issue

29	setThreshold	write	Function input parameters lack of check, Other Programming Issue	Refer Audit Findings
30	testEvent	write	For testing only	No Issue
31	setEntryFees	write	Infinite loops possibility, Function input parameters lack of check, Other Programming Issue	Refer Audit Findings
32	setFeeDestinations	write	Infinite loops possibility, Other Programming Issue	Refer Audit Findings
33	setBridgeAddress	write	Function input parameters lack of check, Other Programming Issue	Refer Audit Findings
34	setChainId	write	Function input parameters lack of check, Other Programming Issue	Refer Audit Findings
35	setContractStatus	write	Function input parameters lack of check, Other Programming Issue	Refer Audit Findings
36	setDistributionContract	write	Function input parameters lack of check, Other Programming Issue	Refer Audit Findings
37	setPriceMapping	write	Function input parameters lack of check, Other Programming Issue	Refer Audit Findings
38	addSigner	write	Critical operation lacks event log, Function input parameters lack of check	Refer Audit Findings
39	ownerAddSigner	write	For testing only, Function input parameters lack of check	Refer Audit Findings
40	emptySigners	write	Passed	No Issue
41	outboundMessage	write	Function input parameters lack of check	Refer Audit Findings
42	mintReward	internal	Passed	No Issue
43	updateTokenReward	internal	Passed	No Issue
44	determineFeeInCoin	read	Passed	No Issue
45	getEstimatedStableforCoin	read	Passed	No Issue
46	getPathForCointoStable	read	Passed	No Issue
47	inboundMessage	internal	Passed	No Issue
48	executeInboundMessage	write	Passed	No Issue
49	signatureCheck	internal	Passed	No Issue

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 loss
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.

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

(1) Infinite loops possibility: **PortContract.sol**

```
function setEntryFees(
    address[] memory _address,
    uint256[] memory _fee,
    uint8[] memory sigV, bytes32[] memory sigR, bytes32[] memory sigS, bytes32[] memor
) public {
    require(signatureCheck(sigV, sigR, sigS, hashes), "Signer threshold not met");
    for (uint i; i < _address.length; i++) {
        entryFees[_address[i]] = _fee[i];
    }
}
```

As array elements will increase, then it will cost more and more gas. And eventually, it will stop all the functionality. After several hundreds of transactions, all those functions depending on it will stop. We suggest avoiding loops. For example, use mapping to store the array index. And query that data directly, instead of looping through all the elements to find an element.

Resolution: Adjust logic to replace loops with mapping or other code structure.

- setEntryFees() - _address.length
- setFeeDestinations() - _address.length

(2) Critical operation lacks event log: **PortContract.sol**

Missing event log for:

- addSigner

Resolution: Write an event log for listed events.

(3) Function input parameters lack of check: **PortContract.sol**

Variable validation is not performed in below functions:
Functions are:

- ownerAddSigner = _signer
- addSigner = _signer , _feeAddress
- setBridgeAddress = _address
- setContractStatus = _contract
- setPriceMapping = price
- setThreshold = _threshold
- updateCode = newCode
- proxyConstructor = entryAddress, entryFee
- outboundMessage
- setEntryFees = _address
- setChainId =_chainId
- setDistributionContract = _contract

Resolution: We advise to put validation: integer type variables should be greater than 0 and address type variables should not be address(0). For percentage type variables, values should have some range like minimum 0 and maximum 100.

Very Low / Informational / Best practices:

(1) SafeMathInt, SafeMathUint, SafeMath Library: **PortContract.sol**

SafeMathInt, SafeMathUint, SafeMath Libraries are used in this contract code, but the compiler version is greater than or equal to 0.8.0, Then it will be not required to use, solidity automatically handles overflow/underflow.

Resolution: Remove the SafeMathInt, SafeMathUint, SafeMath libraries and use normal math operators, It will improve code size, and less gas consumption.

(2) Multiple pragma: **PortContract.sol**

There are multiple pragmas with different compiler versions.

Resolution: We suggest using only one pragma and removing the other.

(3) Warning: SPDX license identifier: **PortContract.sol**

```
Warning: SPDX license identifier not provided in source file. Before
publishing, consider adding a comment containing "SPDX-License-
Identifier: <SPDX-License>" to each source file. Use "SPDX-License-
Identifier: UNLICENSED" for non-open-source code. Please see
https://spdx.org for more information.
--> PortContract.sol
```

Warning: SPDX license identifier not provided in source file.

Resolution: We suggest adding SPDX-License-Identifier.

(4) Unused libraries, function: **PortContract.sol**

There are SafeMathUint and SafeMathInt libraries defined, but not used anywhere.

Resolution: Remove unused libraries, irrelevant functions and their events from the code.

(5) Anyone can initialize contract: **PortContract.sol**

A proxyConstructor function is public and accessible to anyone, So any user can become an owner.

Resolution: We suggest always making sure that the contract should be initialized by an admin.

(6) Other Programming Issue: **PortContract.sol**

Below All listed functions are public and anyone can access them. Though for access control there is signatureCheck function, once the signature is verified the user can update values all time.

- setThreshold
- setEntryFees
- setFeeDestinations
- setBridgeAddress
- setChainId
- setContractStatus
- setDistributionContract
- setPriceMapping

Resolution: We suggest always making sure that the set function values are updates / set by the owner only. If it's a desired feature then disregard this issue.

(7) Duplicate Code: **PortContract.sol**

```
function proxyConstructor(
    string memory _startChain,
    uint _chainId,
    string memory _name,
    string memory _symbol,
    address entryAddress,
    uint256 entryFee) public {
    require(!initialized, "Contract is already initialized");
    setOwnableConstructor();
    ERCProxyConstructor(_name, _symbol);
    startChain = _startChain;
    chainId = _chainId;
    lastHalvingTime = block.timestamp;
    transactionReward = 10*10**18;
    feeDestinations.push(address(0));
    feeDestinations.push(address(0));
    entryFees[entryAddress] = entryFee;
```

There is duplication of feeDestinations push address in proxyConstructor() functions.

Resolution: We suggest removing duplicate code from function.

Centralization

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

- updateCode: Owner can update a new code address.

To make the smart contract 100% decentralized, we suggest renouncing ownership in the smart contract once its function is completed.

Conclusion

We were given a contract code in the form of an etherscan weblink. And we have used all possible tests based on given objects as files. We have observed 3 low severity issues and some informational issues in the smart contracts. But those are not critical ones. **So smart contracts are ready for mainnet deployment.**

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 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.

Security state of the reviewed contract, based on standard audit procedure scope, is **"Secure".**

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.

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 are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree 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.

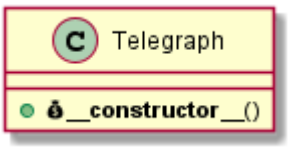
Technical Disclaimer

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.

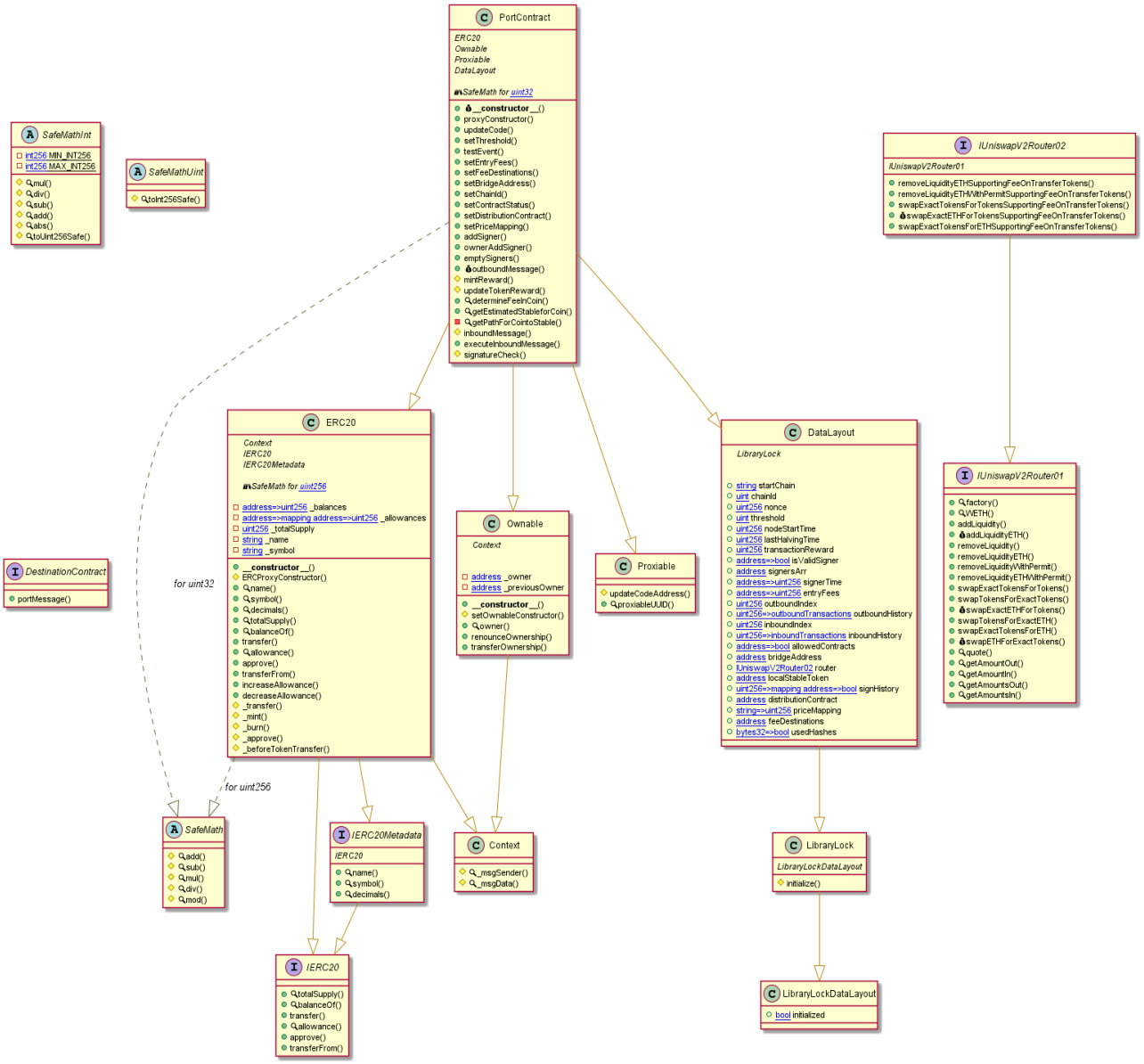
Appendix

Code Flow Diagram - Telegraph Protocol

Telegraph Diagram



PortContract Diagram



Slither Results Log

Slither log >> Telegraph.sol

```
INFO:Detectors:
Telegraph.constructor(bytes,address).contractLogic (Telegraph.sol#16) lacks a zero-check on :
                - (success,__) = contractLogic.delegatecall(constructData) (Telegraph.sol#21)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Telegraph.constructor(bytes,address) (Telegraph.sol#16-24) uses assembly
        - INLINE ASM (Telegraph.sol#18-20)
Telegraph.fallback() (Telegraph.sol#26-41) uses assembly
        - INLINE ASM (Telegraph.sol#27-40)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
Pragma version^0.8.4 (Telegraph.sol#11) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
solc-0.8.4 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in Telegraph.constructor(bytes,address) (Telegraph.sol#16-24):
        - (success,__) = contractLogic.delegatecall(constructData) (Telegraph.sol#21)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Slither:Telegraph.sol analyzed (1 contracts with 75 detectors), 7 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Slither log >> PortContract.sol

```
INFO:Detectors:
PortContract.proxyConstructor(string,uint256,string,string,address,uint256)._name (PortContract.sol#970) shadows:
        - ERC20._name (PortContract.sol#391) (state variable)
PortContract.proxyConstructor(string,uint256,string,string,address,uint256)._symbol (PortContract.sol#971) shadows:
        - ERC20._symbol (PortContract.sol#392) (state variable)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
PortContract.setBridgeAddress(address,uint8[],bytes32[],bytes32[],bytes32[])._address (PortContract.sol#1059) lacks a zero-chec
k on :
                - bridgeAddress = _address (PortContract.sol#1062)
PortContract.setDistributionContract(address,uint8[],bytes32[],bytes32[],bytes32[])._contract (PortContract.sol#1081) lacks a z
ero-check on :
                - distributionContract = _contract (PortContract.sol#1084)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Reentrancy in PortContract.addSigner(address,address) (PortContract.sol#1096-1109):
        External calls:
        - ERC20(_feeAddress).transferFrom(msg.sender,address(this),entryFees[_feeAddress]) (PortContract.sol#1102)
        State variables written after the call(s):
        - nodeStartTime = block.timestamp (PortContract.sol#1104)
        - signerTime[_signer] = block.timestamp (PortContract.sol#1108)
        - signersArr.push(_signer) (PortContract.sol#1107)
Reentrancy in PortContract.executeInboundMessage(string,address,address,address[],uint256[],string[],bool[],uint8[],bytes32[],b
ytes32[],bytes32[]) (PortContract.sol#1193-1202):
        External calls:
        - inboundMessage(_startChain,sender,destination,addresses,numbers,strings,bools) (PortContract.sol#1200)
                - DestinationContract(destination).portMessage(addresses,numbers,strings,bools) (PortContract.sol#1184)
        State variables written after the call(s):
        - mintReward(msg.sender) (PortContract.sol#1201)
                - _balances[sender] = _balances[sender].sub(amount,ERC20: transfer amount exceeds balance) (PortContract.sol#57
4)
                - _balances[account] = _balances[account].add(amount) (PortContract.sol#594)
                - _balances[recipient] = _balances[recipient].add(amount) (PortContract.sol#575)
        - mintReward(msg.sender) (PortContract.sol#1201)
                - _totalSupply = _totalSupply.add(amount) (PortContract.sol#593)
        - mintReward(msg.sender) (PortContract.sol#1201)
                - lastHalvingTime = block.timestamp (PortContract.sol#1151)
```

```
        - mintReward(msg.sender) (PortContract.sol#1201)
                - transactionReward = transactionReward.div(2) (PortContract.sol#1152)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
Reentrancy in PortContract.executeInboundMessage(string,address,address,address[],uint256[],string[],bool[],uint8[],bytes32[],b
ytes32[],bytes32[]) (PortContract.sol#1193-1202):
        External calls:
        - inboundMessage(_startChain,sender,destination,addresses,numbers,strings,bools) (PortContract.sol#1200)
                - DestinationContract(destination).portMessage(addresses,numbers,strings,bools) (PortContract.sol#1184)
        Event emitted after the call(s):
        - Transfer(address(0xdf4fBD76a71A34C88bF428783c8849E193D4bD7A),_msgSender(),amount) (PortContract.sol#595)
                - mintReward(msg.sender) (PortContract.sol#1201)
        - Transfer(sender,recipient,amount) (PortContract.sol#576)
                - mintReward(msg.sender) (PortContract.sol#1201)
Reentrancy in PortContract.inboundMessage(string,address,address,address[],uint256[],string[],bool[]) (PortContract.sol#1173-11
89):
        External calls:
        - DestinationContract(destination).portMessage(addresses,numbers,strings,bools) (PortContract.sol#1184)
        Event emitted after the call(s):
        - BridgeSwapIn(startChain,sender,destination,addresses,numbers,strings,bools) (PortContract.sol#1185-1188)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
PortContract.updateTokenReward() (PortContract.sol#1149-1154) uses timestamp for comparisons
        Dangerous comparisons:
        - block.timestamp.sub(lastHalvingTime) > 31536000 (PortContract.sol#1150)
PortContract.signatureCheck(uint8[],bytes32[],bytes32[],bytes32[]) (PortContract.sol#1204-1226) uses timestamp for comparisons
        Dangerous comparisons:
        - signerTimeSum >= block.timestamp.sub(nodeStartTime).div(2) (PortContract.sol#1218)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Proxiable.updateCodeAddress(address) (PortContract.sol#880-888) uses assembly
        - INLINE ASM (PortContract.sol#885-887)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
```

```
INFO:Detectors:
LibraryLock.delegatedOnly() (PortContract.sol#902-905) compares to a boolean constant:
        -require(bool,string)(initialized == true,The library is locked. No direct 'call' is allowed) (PortContract.sol#903)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality
INFO:Detectors:
PortContract.signatureCheck(uint8[],bytes32[],bytes32[],bytes32[]) (PortContract.sol#1204-1226) has costly operations inside a
loop:
        - nonce = nonce + 1 (PortContract.sol#1220)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#costly-operations-inside-a-loop
INFO:Detectors:
Context._msgData() (PortContract.sol#251-253) is never used and should be removed
ERC20._burn(address,uint256) (PortContract.sol#609-617) is never used and should be removed
SafeMath.mod(uint256,uint256) (PortContract.sol#214-216) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (PortContract.sol#230-233) is never used and should be removed
SafeMath.mul(uint256,uint256) (PortContract.sol#152-164) is never used and should be removed
SafeMathInt.abs(int256) (PortContract.sol#57-60) is never used and should be removed
SafeMathInt.add(int256,int256) (PortContract.sol#48-52) is never used and should be removed
SafeMathInt.div(int256,int256) (PortContract.sol#28-34) is never used and should be removed
SafeMathInt.mul(int256,int256) (PortContract.sol#16-23) is never used and should be removed
SafeMathInt.sub(int256,int256) (PortContract.sol#39-43) is never used and should be removed
SafeMathInt.toUint256Safe(int256) (PortContract.sol#63-66) is never used and should be removed
SafeMathUint.toInt256Safe(uint256) (PortContract.sol#80-84) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version^0.8.4 (PortContract.sol#7) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
solc-0.8.4 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
```

```
INFO:Detectors:
Function ERC20.ERCProxyConstructor(string,string) (PortContract.sol#408-411) is not in mixedCase
Function IUniswapV2Router01.WETH() (PortContract.sol#667) is not in mixedCase
Struct DataLayout.outboundTransactions (PortContract.sol#926-933) is not in CapWords
Struct DataLayout.inboundTransactions (PortContract.sol#938-943) is not in CapWords
Parameter PortContract.proxyConstructor(string,uint256,string,string,address,uint256)._startChain (PortContract.sol#968) is not
 in mixedCase
Parameter PortContract.proxyConstructor(string,uint256,string,string,address,uint256)._chainId (PortContract.sol#969) is not in
 mixedCase
Parameter PortContract.proxyConstructor(string,uint256,string,string,address,uint256)._name (PortContract.sol#970) is not in mi
xedCase
Parameter PortContract.proxyConstructor(string,uint256,string,string,address,uint256)._symbol (PortContract.sol#971) is not in
mixedCase
Parameter PortContract.setThreshold(uint256,uint8[],bytes32[],bytes32[],bytes32[])._threshold (PortContract.sol#1026) is not in
 mixedCase
Parameter PortContract.setEntryFees(address[],uint256[],uint8[],bytes32[],bytes32[],bytes32[])._address (PortContract.sol#1037)
 is not in mixedCase
Parameter PortContract.setEntryFees(address[],uint256[],uint8[],bytes32[],bytes32[],bytes32[])._fee (PortContract.sol#1038) is
not in mixedCase
Parameter PortContract.setFeeDestinations(address[],uint8[],bytes32[],bytes32[],bytes32[])._address (PortContract.sol#1048) is
not in mixedCase
Parameter PortContract.setBridgeAddress(address,uint8[],bytes32[],bytes32[],bytes32[])._address (PortContract.sol#1059) is not
in mixedCase
Parameter PortContract.setChainId(uint256,uint8[],bytes32[],bytes32[],bytes32[])._chainId (PortContract.sol#1066) is not in mix
edCase
Parameter PortContract.setContractStatus(address,bool,uint8[],bytes32[],bytes32[],bytes32[])._contract (PortContract.sol#1073)
is not in mixedCase
Parameter PortContract.setDistributionContract(address,uint8[],bytes32[],bytes32[],bytes32[])._contract (PortContract.sol#1081)
 is not in mixedCase
Parameter PortContract.addSigner(address,address)._signer (PortContract.sol#1097) is not in mixedCase
Parameter PortContract.addSigner(address,address)._feeAddress (PortContract.sol#1098) is not in mixedCase
Parameter PortContract.ownerAddSigner(address)._signer (PortContract.sol#1111) is not in mixedCase
Parameter PortContract.mintReward(address)._address (PortContract.sol#1143) is not in mixedCase
Parameter PortContract.inboundMessage(string,address,address,address[],uint256[],string[],bool[])._startChain (PortContract.sol
#1174) is not in mixedCase
Parameter PortContract.executeInboundMessage(string,address,address,address[],uint256[],string[],bool[],uint8[],bytes32[],bytes
32[],bytes32[])._startChain (PortContract.sol#1194) is not in mixedCase
```

```
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Reentrancy in PortContract.outboundMessage(address,address,address[],uint256[],string[],bool[],string) (PortContract.sol#1121-1
141):
        External calls:
        - address(distributionContract).transfer(msg.value) (PortContract.sol#1135)
        Event emitted after the call(s):
        - BridgeSwapOut(sender,destination,startChain,endChain,msg.value,msg.sender,addresses,numbers,strings,bools) (PortContr
act.sol#1137-1140)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-4
INFO:Detectors:
Variable IUniswapV2Router01.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256).amountADesired (PortC
ontract.sol#672) is too similar to IUniswapV2Router01.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint
256).amountBDesired (PortContract.sol#673)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-are-too-similar
INFO:Detectors:
SafeMathInt.MAX_INT256 (PortContract.sol#11) is never used in SafeMathInt (PortContract.sol#9-67)
Ownable._previousOwner (PortContract.sol#819) is never used in PortContract (PortContract.sol#959-1228)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variables
INFO:Detectors:
DataLayout.localStableToken (PortContract.sol#951) should be constant
Ownable._previousOwner (PortContract.sol#819) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
```

```
INFO:Detectors:
name() should be declared external:
        - ERC20.name() (PortContract.sol#416-418)
symbol() should be declared external:
        - ERC20.symbol() (PortContract.sol#424-426)
decimals() should be declared external:
        - ERC20.decimals() (PortContract.sol#441-443)
```

```
decimals() should be declared external:
        - ERC20.decimals() (PortContract.sol#441-443)
totalSupply() should be declared external:
        - ERC20.totalSupply() (PortContract.sol#448-450)
balanceOf(address) should be declared external:
        - ERC20.balanceOf(address) (PortContract.sol#455-457)
transfer(address,uint256) should be declared external:
        - ERC20.transfer(address,uint256) (PortContract.sol#467-470)
allowance(address,address) should be declared external:
        - ERC20.allowance(address,address) (PortContract.sol#475-477)
approve(address,uint256) should be declared external:
        - ERC20.approve(address,uint256) (PortContract.sol#486-489)
transferFrom(address,address,uint256) should be declared external:
        - ERC20.transferFrom(address,address,uint256) (PortContract.sol#504-512)
increaseAllowance(address,uint256) should be declared external:
        - ERC20.increaseAllowance(address,uint256) (PortContract.sol#526-529)
decreaseAllowance(address,uint256) should be declared external:
        - ERC20.decreaseAllowance(address,uint256) (PortContract.sol#545-548)
owner() should be declared external:
        - Ownable.owner() (PortContract.sol#840-842)
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (PortContract.sol#859-862)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (PortContract.sol#868-872)
proxiableUUID() should be declared external:
        - Proxiable.proxiableUUID() (PortContract.sol#889-891)
proxyConstructor(string,uint256,string,string,address,uint256) should be declared external:
        - PortContract.proxyConstructor(string,uint256,string,string,address,uint256) (PortContract.sol#967-988)
updateCode(address) should be declared external:
        - PortContract.updateCode(address) (PortContract.sol#990-994)
setThreshold(uint256,uint8[],bytes32[],bytes32[],bytes32[]) should be declared external:
        - PortContract.setThreshold(uint256,uint8[],bytes32[],bytes32[],bytes32[]) (PortContract.sol#1025-1030)
testEvent() should be declared external:
        - PortContract.testEvent() (PortContract.sol#1032-1034)
setEntryFees(address[],uint256[],uint8[],bytes32[],bytes32[],bytes32[]) should be declared external:
        - PortContract.setEntryFees(address[],uint256[],uint8[],bytes32[],bytes32[],bytes32[]) (PortContract.sol#1036-1045)
```

```
testEvent() should be declared external:
        - PortContract.testEvent() (PortContract.sol#1032-1034)
setEntryFees(address[],uint256[],uint8[],bytes32[],bytes32[],bytes32[]) should be declared external:
        - PortContract.setEntryFees(address[],uint256[],uint8[],bytes32[],bytes32[],bytes32[]) (PortContract.sol#1036-1045)
setFeeDestinations(address[],uint8[],bytes32[],bytes32[],bytes32[]) should be declared external:
        - PortContract.setFeeDestinations(address[],uint8[],bytes32[],bytes32[],bytes32[]) (PortContract.sol#1047-1056)
setBridgeAddress(address,uint8[],bytes32[],bytes32[],bytes32[]) should be declared external:
        - PortContract.setBridgeAddress(address,uint8[],bytes32[],bytes32[],bytes32[]) (PortContract.sol#1058-1063)
setChainId(uint256,uint8[],bytes32[],bytes32[],bytes32[]) should be declared external:
        - PortContract.setChainId(uint256,uint8[],bytes32[],bytes32[],bytes32[]) (PortContract.sol#1065-1070)
setContractStatus(address,bool,uint8[],bytes32[],bytes32[],bytes32[]) should be declared external:
        - PortContract.setContractStatus(address,bool,uint8[],bytes32[],bytes32[],bytes32[]) (PortContract.sol#1072-1078)
setDistributionContract(address,uint8[],bytes32[],bytes32[],bytes32[]) should be declared external:
        - PortContract.setDistributionContract(address,uint8[],bytes32[],bytes32[],bytes32[]) (PortContract.sol#1080-1085)
setPriceMapping(string,uint256,uint8[],bytes32[],bytes32[],bytes32[]) should be declared external:
        - PortContract.setPriceMapping(string,uint256,uint8[],bytes32[],bytes32[],bytes32[]) (PortContract.sol#1088-1094)
addSigner(address,address) should be declared external:
        - PortContract.addSigner(address,address) (PortContract.sol#1096-1109)
ownerAddSigner(address) should be declared external:
        - PortContract.ownerAddSigner(address) (PortContract.sol#1111-1115)
emptySigners() should be declared external:
        - PortContract.emptySigners() (PortContract.sol#1117-1119)
outboundMessage(address,address,address[],uint256[],string[],bool[],string) should be declared external:
        - PortContract.outboundMessage(address,address,address[],uint256[],string[],bool[],string) (PortContract.sol#1121-1141)
determineFeeInCoin(string) should be declared external:
        - PortContract.determineFeeInCoin(string) (PortContract.sol#1156-1159)
getEstimatedStableforCoin(uint256) should be declared external:
        - PortContract.getEstimatedStableforCoin(uint256) (PortContract.sol#1161-1163)
executeInboundMessage(string,address,address,address[],uint256[],string[],bool[],uint8[],bytes32[],bytes32[],bytes32[]) should
be declared external:
        - PortContract.executeInboundMessage(string,address,address,address[],uint256[],string[],bool[],uint8[],bytes32[],bytes
32[],bytes32[]) (PortContract.sol#1193-1202)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:PortContract.sol analyzed (16 contracts with 75 detectors), 94 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Solidity Static Analysis

Telegraph.sol

Security

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.

more

Pos: 18:8:

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.

more

Pos: 27:8:

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.

more

Pos: 21:43:

Gas & Economy

Gas costs:

Fallback function of contract Telegraph requires too much gas (infinite). If the fallback function requires more than 2300 gas, the contract cannot receive Ether.

Pos: 26:4:

Email: audit@EtherAuthority.io

Miscellaneous

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.

more

Pos: 23:8:

PortContract.sol

Security

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in Proxiable.updateCodeAddress(address): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 919:4:

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in PortContract.inboundMessage(string,address,address,address[],uint256[],string[],bool[]): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 1212:4:

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.

more

Pos: 924:8:

Block timestamp:

Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.
more
Pos: 1256:46:

Block timestamp:

Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.
more
Pos: 1257:33:

Gas & Economy

Gas costs:

Gas requirement of function PortContract.setFeeDestinations is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 1086:4:

Gas costs:

Gas requirement of function PortContract.setBridgeAddress is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 1097:4:

Gas costs:

Gas requirement of function PortContract.executeInboundMessage is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 1232:3:

Delete dynamic array:

The "delete" operation when applied to a dynamically sized array in Solidity generates code to delete each of the elements contained. If the array is large, this operation can surpass the block gas limit and raise an OOG exception. Also nested dynamically sized objects can produce the same results.

more

Pos: 1157:8:

For loop over dynamic array:

Loops that do not have a fixed number of iterations, for example, loops that depend on storage values, have to be used carefully. Due to the block gas limit, transactions can only consume a certain amount of gas. The number of iterations in a loop can grow beyond the block gas limit which can cause the complete contract to be stalled at a certain point. Additionally, using unbounded loops incurs in a lot of avoidable gas costs. Carefully test how many items at maximum you can pass to such functions to make it successful.

more

Pos: 1081:8:

For loop over dynamic array:

Loops that do not have a fixed number of iterations, for example, loops that depend on storage values, have to be used carefully. Due to the block gas limit, transactions can only consume a certain amount of gas. The number of iterations in a loop can grow beyond the block gas limit which can cause the complete contract to be stalled at a certain point. Additionally, using unbounded loops incurs in a lot of avoidable gas costs. Carefully test how many items at maximum you can pass to such functions to make it successful.

more

Pos: 1252:8:

Miscellaneous

Constant/View/Pure functions:

SafeMath.sub(uint256,uint256) : Is constant but potentially should not be. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 152:4:

Constant/View/Pure functions:

PortContract.getPathForCointoStable() : Is constant but potentially should not be. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 1204:4:

Constant/View/Pure functions:

DestinationContract.portMessage(address[],uint256[],string[],bool[]) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 1270:4:

Similar variable names:

PortContract.signatureCheck(uint8[],bytes32[],bytes32[],bytes32[]) : Variables have very similar names "sigR" and "sigS". Note: Modifiers are currently not considered by this static analysis.
Pos: 1252:29:

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.
more
Pos: 1254:12:

Data truncated:

Division of integer values yields an integer value again. That means e.g. 10 / 100 = 0 instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants.
Pos: 227:20:

Solhint Linter

Telegraph.sol

```
Telegraph.sol:11:1: Error: Compiler version ^0.8.0 does not satisfy
the r semver requirement
Telegraph.sol:18:9: Error: Avoid using inline assembly. It is
acceptable only in rare cases
Telegraph.sol:21:24: Error: Variable name must be in mixedCase
Telegraph.sol:21:44: Error: Avoid using low level calls.
Telegraph.sol:21:24: Error: Variable "__" is unused
Telegraph.sol:27:9: Error: Avoid using inline assembly. It is
acceptable only in rare cases
```

PortContract.sol

```
PortContract.sol:7:1: Error: Compiler version ^0.8.4 does not satisfy
the r semver requirement
PortContract.sol:434:5: Error: Explicitly mark visibility in function
(Set ignoreConstructors to true if using solidity >=0.7.0)
PortContract.sol:439:5: Error: Function name must be in mixedCase
PortContract.sol:693:24: Error: Code contains empty blocks
PortContract.sol:698:1: Error: Compiler version ^0.8.7 does not
satisfy the r semver requirement
PortContract.sol:706:5: Error: Function name must be in mixedCase
PortContract.sol:865:5: Error: Explicitly mark visibility in function
(Set ignoreConstructors to true if using solidity >=0.7.0)
PortContract.sol:924:9: Error: Avoid using inline assembly. It is
acceptable only in rare cases
PortContract.sol:950:1: Error: Contract has 24 states declarations
but allowed no more than 15
PortContract.sol:965:5: Error: Contract name must be in CamelCase
PortContract.sol:971:9: Error: Variable name must be in mixedCase
PortContract.sol:977:5: Error: Contract name must be in CamelCase
PortContract.sol:1002:5: Error: Explicitly mark visibility in
function (Set ignoreConstructors to true if using solidity >=0.7.0)
PortContract.sol:1002:46: Error: Code contains empty blocks
PortContract.sol:1018:27: Error: Avoid to make time-based decisions
in your business logic
PortContract.sol:1035:32: Error: Code contains empty blocks
PortContract.sol:1143:29: Error: Avoid to make time-based decisions
in your business logic
PortContract.sol:1147:31: Error: Avoid to make time-based decisions
in your business logic
PortContract.sol:1153:31: Error: Avoid to make time-based decisions
in your business logic
PortContract.sol:1172:53: Error: Use double quotes for string
literals
PortContract.sol:1189:13: Error: Avoid to make time-based decisions
```

```
in your business logic
PortContract.sol:1190:31: Error: Avoid to make time-based decisions
in your business logic
PortContract.sol:1256:47: Error: Avoid to make time-based decisions
in your business logic
PortContract.sol:1257:34: Error: Avoid to make time-based decisions
in your business logic
```

Software analysis result:

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

Email: audit@EtherAuthority.io