Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project:	Ocean Token
Website:	oceanprotocol.com
Platform:	Ethereum
Language:	Solidity
Date:	May 19th, 2024

Table of contents

`

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 Ocean Token smart contract from oceanprotocol.com 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 May 19th, 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

- This contract is an implementation of the Ocean Protocol ERC20 token. Let's break down its key components:
  - **Interfaces and Libraries:** It defines the ERC20 interface and uses the SafeMath library for safe arithmetic operations.
  - **Roles:** The contract includes roles for minters and pausers, allowing certain addresses to mint new tokens or pause token transfers.
  - **ERC20Mintable:** This contract extends ERC20 and adds minting functionality. Only addresses with the minter role can mint new tokens.
  - **ERC20Capped:** This contract extends ERC20Mintable and adds a cap to the total token supply. The cap is set in the constructor.
  - **ERC20Detailed:** This contract provides detailed information about the token, such as its name, symbol, and decimals.
  - **Pausable:** It implements pausable functionality, allowing the owner (or addresses with the pauser role) to pause and unpause token transfers.
  - **Ownable:** This contract defines ownership functionality, allowing the current owner to transfer ownership to another address.
  - **OceanToken:** The main token contract that inherits from ERC20Pausable, ERC20Detailed, ERC20Capped, and Ownable. It sets up the token with specific parameters in the constructor and implements additional functionality such as retrieving token holders' addresses and balances, killing the contract, and handling ether transfers.

- Overall, this contract provides a comprehensive implementation of a capped, mintable ERC20 token with pausable functionality and detailed information about the token.

Audit scope

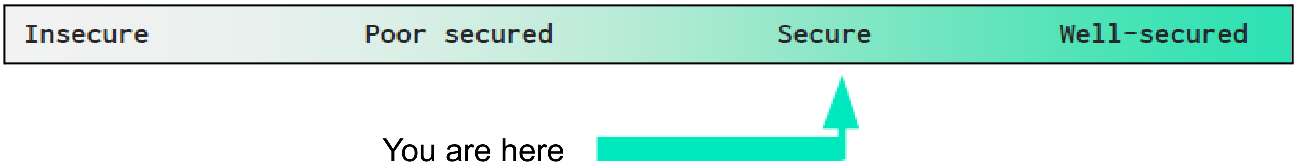
Name	**Code Review and Security Analysis Report for Ocean Token Smart Contract**
Platform	**Ethereum**
Language	**Solidity**
File	OceanToken.sol
Smart Contract Code	0x967da4048cD07aB37855c090aAF366e4ce1b9F48
Audit Date	May 19th, 2024

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Tokenomics: ● Name: Ocean Token ● Symbol: cap ● Decimals: 18 ● Total Supply: 1410 Million ● Cap: 1410 Million	**YES, This is valid.**
Ownership Control: ● Retrieve the address & token balance of token holders. ● Get the length of the account list. ● Kill the contract and destroy all tokens. ● Current owner can transfer the ownership. ● Owner can renounce ownership.	**YES, This is valid.** **We suggest renouncing ownership once the ownership functions are not needed. This is to make the smart contract 100% decentralized.**
Other Specifications: ● Adding the pauser only by the pauser owner. ● Mint the new tokens only by the minter owner.	**YES, This is valid.**

Audit Summary

According to the standard audit assessment, 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 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, 0 low and 3 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	The solidity version is not specified	Passed
	Solidity version is too old	Passed
	Integer overflow/underflow	Passed
	Function input parameters lack of 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

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?	Yes
🟢 Is it a Proxy?	No
🟢 Can Take Ownership?	Yes
🟢 Hidden Owner?	Not Detected
🟢 Self Destruction?	Not Detected
🟢 Auditor Confidence	High

Overall Audit Result: PASSED

Code Quality

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

The libraries in Ocean Token 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 Ocean Token.

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 an Ocean Token smart contract code in the form of an Etherscan 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.

AS-IS overview

OceanToken.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	transfer	write	Passed	No Issue
3	transferFrom	write	Passed	No Issue
4	getAccounts	external	access only Owner	No Issue
5	getAccountsLength	external	access only Owner	No Issue
6	kill	external	access only Owner	No Issue
7	tryToAddTokenHolder	write	Passed	No Issue
8	updateTokenHolders	write	Passed	No Issue
9	owner	read	Passed	No Issue
10	onlyOwner	modifier	Passed	No Issue
11	isOwner	read	Passed	No Issue
12	renounceOwnership	write	access only Owner	No Issue
13	transferOwnership	write	access only Owner	No Issue
14	transferOwnership	internal	Passed	No Issue
15	transfer	write	when Not Paused	No Issue
16	transferFrom	write	when Not Paused	No Issue
17	approve	write	when Not Paused	No Issue
18	increaseAllowance	write	when Not Paused	No Issue
19	decreaseAllowance	write	when Not Paused	No Issue
20	paused	read	Passed	No Issue
21	whenNotPaused	modifier	Passed	No Issue
22	whenPaused	modifier	Passed	No Issue
23	pause	write	access only Pauser	No Issue
24	unpause	write	access only Pauser	No Issue
25	onlyPauser	modifier	Passed	No Issue
26	isPauser	read	Passed	No Issue
27	addPauser	write	access only Pauser	No Issue
28	renouncePauser	write	Passed	No Issue
29	_addPauser	internal	Passed	No Issue
30	_removePauser	internal	Passed	No Issue
31	name	read	Passed	No Issue
32	symbol	read	Passed	No Issue
33	decimals	read	Passed	No Issue
34	cap	read	Passed	No Issue
35	_mint	internal	Passed	No Issue
36	mint	write	access only Minter	No Issue
37	onlyMinter	modifier	Passed	No Issue
38	isMinter	read	Passed	No Issue
39	addMinter	write	access only Minter	No Issue
40	renounceMinter	write	Passed	No Issue
41	_addMinter	internal	Passed	No Issue

42	_removeMinter	internal	Passed	No Issue
43	totalSupply	read	Passed	No Issue
44	balanceOf	read	Passed	No Issue
45	allowance	read	Passed	No Issue
46	transfer	write	Passed	No Issue
47	approve	write	Passed	No Issue
48	transferFrom	write	Passed	No Issue
49	increaseAllowance	write	Passed	No Issue
50	decreaseAllowance	write	Passed	No Issue
51	_transfer	internal	Passed	No Issue
52	_mint	internal	Passed	No Issue
53	_burn	internal	Passed	No Issue
54	_burnFrom	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 a 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 a 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.

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.

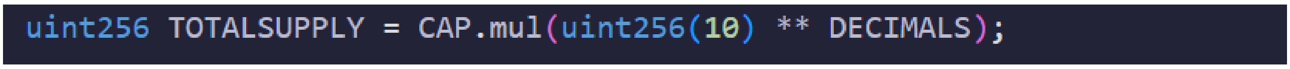
Very Low / Informational / Best practices:

(1) Missing SPDX-License-Identifier:

SPDX-License-Identifier is written but with the wrong syntax.

Resolution: We suggest adding the correct SPDX License Identifier.

(2) Make variables constant:



```
uint256 TOTALSUPPLY = CAP.mul(uint256(10) ** DECIMALS);
```

These variable values will remain unchanged.

Resolution: We suggest making them constant. It is best practice and it also saves some gas. Just add a constant keyword.

(3) Missing error message in required condition:

There is no error message added for required conditions in imported contracts and libraries

Below is the list:

- ERC20
- Roles
- Ownable
- Pausable
- PauserRole
- ERC20Capped

It is best practice to add custom error messages in every required condition, which would be helpful in debugging as well as giving a clear indication of any transaction failure.

Resolution: We suggest adding custom error messages in every required condition.

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. The following are Admin functions:

OceanToken.sol

- getAccounts: Retrieve the address & token balance of token holders by the owner.
- getAccountsLength: Get the length of the account list by the owner.
- kill: Kill the contract and destroy all tokens by the owner.

Ownable.sol

- renounceOwnership: Deleting ownership will leave the contract without an owner, removing any owner-only functionality.
- transferOwnership: Current owner can transfer ownership of the contract to a new account.

PauserRole.sol

- addPauser: Adding the pauser only by the pauser owner.

ERC20Mintable.sol

- mint: Mint the new tokens only by the minter owner.

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 Etherscan web links. And we have used all possible tests based on given objects as files. We observed 3 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 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 smart contract, based on standard audit procedure scope, is **"Secured".**

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 white box penetration testing. We look at the project's website 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, 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.

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).

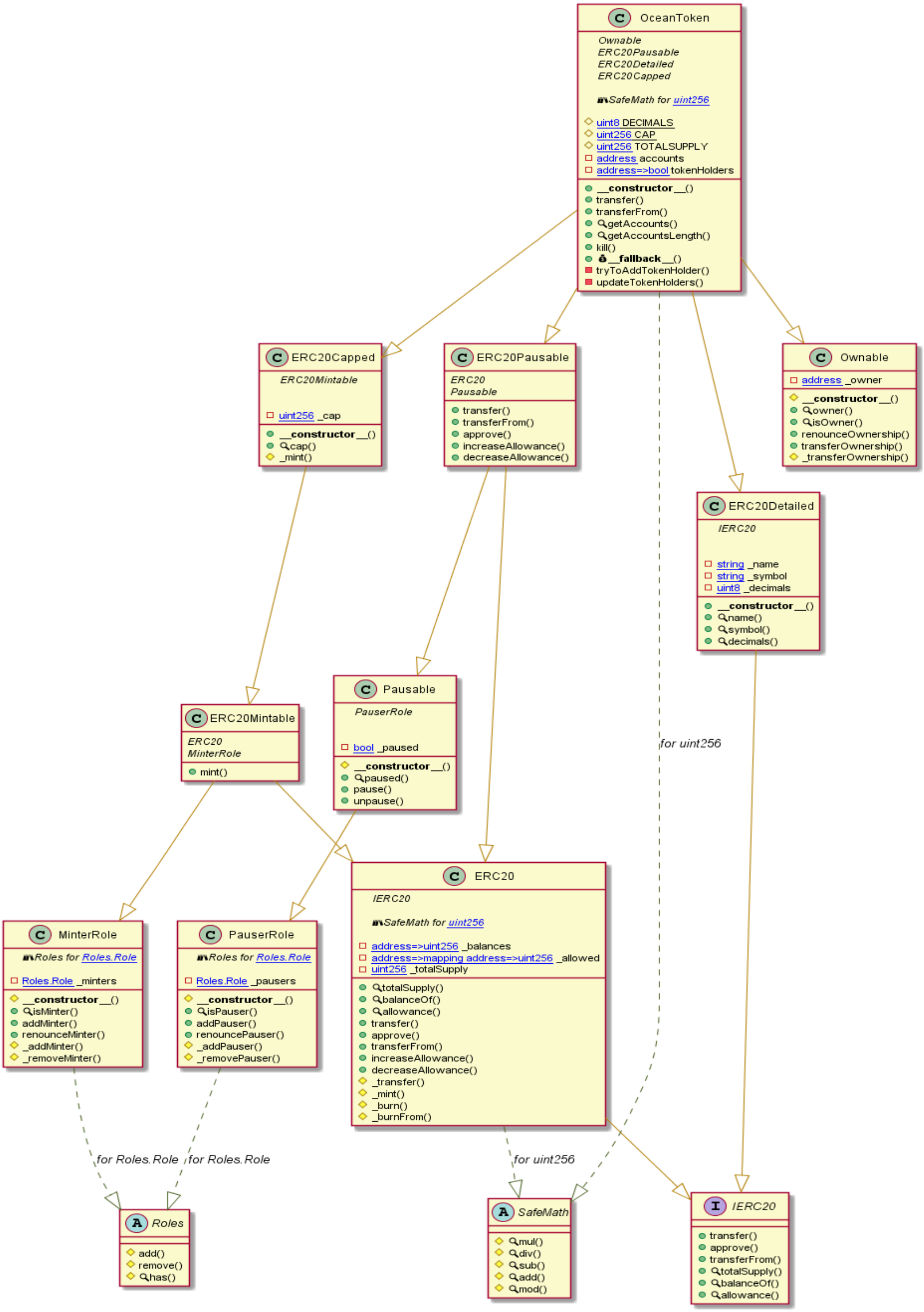
Due to the fact that the total number of test cases is unlimited, the audit makes no statements or warranties on the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. While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

Appendix

Code Flow Diagram - Ocean Token

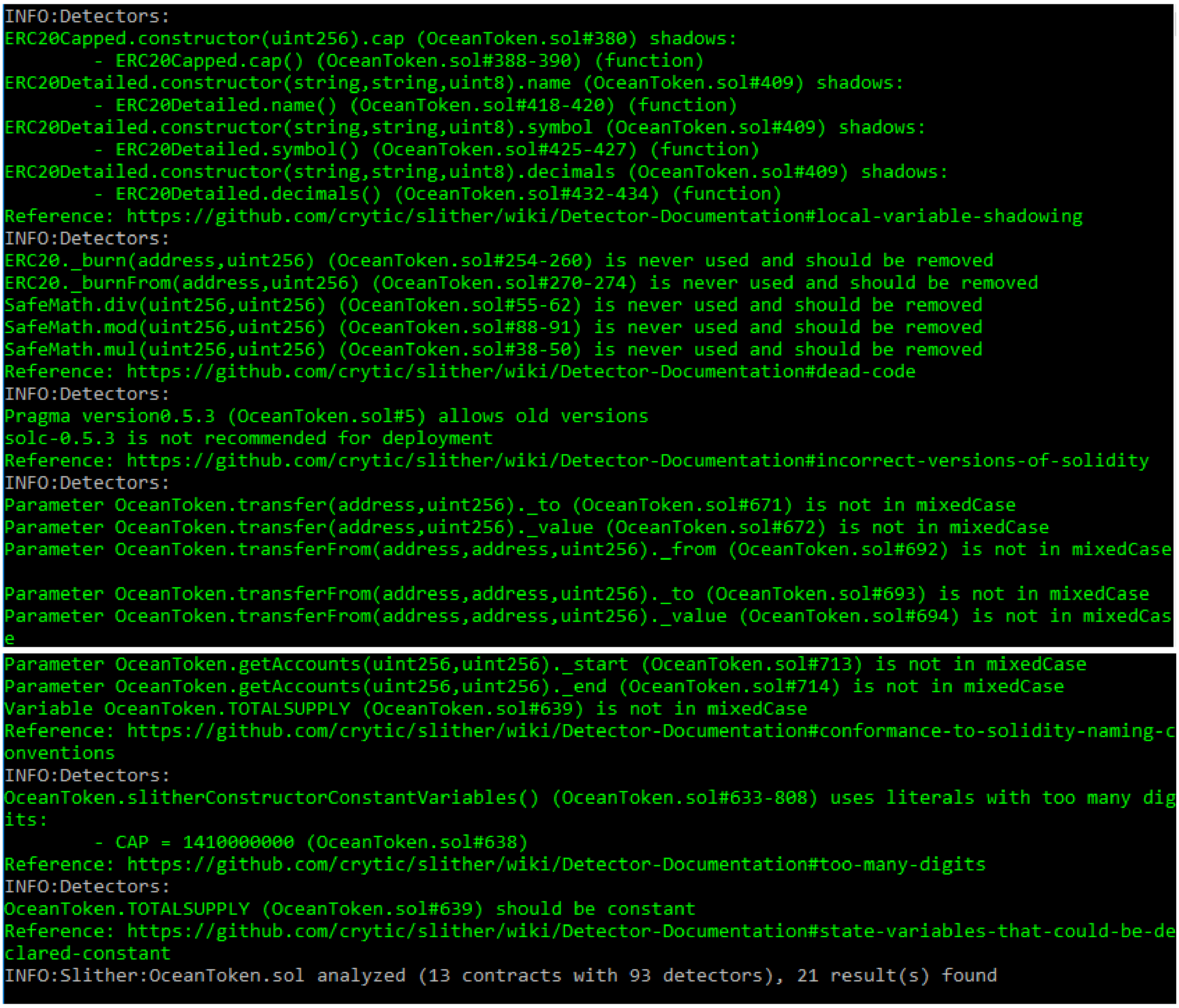


Slither Results Log

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 >> OceanToken.sol



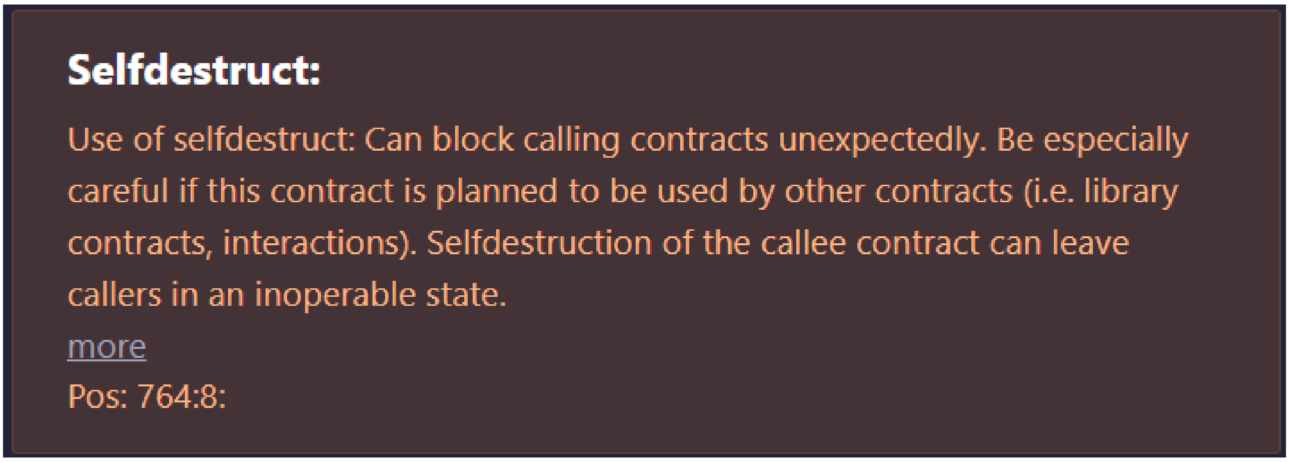
```
INFO:Detectors:
ERC20Capped.constructor(uint256).cap (OceanToken.sol#380) shadows:
        - ERC20Capped.cap() (OceanToken.sol#388-390) (function)
ERC20Detailed.constructor(string,string,uint8).name (OceanToken.sol#409) shadows:
        - ERC20Detailed.name() (OceanToken.sol#418-420) (function)
ERC20Detailed.constructor(string,string,uint8).symbol (OceanToken.sol#409) shadows:
        - ERC20Detailed.symbol() (OceanToken.sol#425-427) (function)
ERC20Detailed.constructor(string,string,uint8).decimals (OceanToken.sol#409) shadows:
        - ERC20Detailed.decimals() (OceanToken.sol#432-434) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
ERC20._burn(address,uint256) (OceanToken.sol#254-260) is never used and should be removed
ERC20._burnFrom(address,uint256) (OceanToken.sol#270-274) is never used and should be removed
SafeMath.div(uint256,uint256) (OceanToken.sol#55-62) is never used and should be removed
SafeMath.mod(uint256,uint256) (OceanToken.sol#88-91) is never used and should be removed
SafeMath.mul(uint256,uint256) (OceanToken.sol#38-50) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version0.5.3 (OceanToken.sol#5) allows old versions
solc-0.5.3 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Parameter OceanToken.transfer(address,uint256)._to (OceanToken.sol#671) is not in mixedCase
Parameter OceanToken.transfer(address,uint256)._value (OceanToken.sol#672) is not in mixedCase
Parameter OceanToken.transferFrom(address,address,uint256)._from (OceanToken.sol#692) is not in mixedCase

Parameter OceanToken.transferFrom(address,address,uint256)._to (OceanToken.sol#693) is not in mixedCase
Parameter OceanToken.transferFrom(address,address,uint256)._value (OceanToken.sol#694) is not in mixedCas
e
Parameter OceanToken.getAccounts(uint256,uint256)._start (OceanToken.sol#713) is not in mixedCase
Parameter OceanToken.getAccounts(uint256,uint256)._end (OceanToken.sol#714) is not in mixedCase
Variable OceanToken.TOTALSUPPLY (OceanToken.sol#639) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-c
onventions
INFO:Detectors:
OceanToken.slitherConstructorConstantVariables() (OceanToken.sol#633-808) uses literals with too many dig
its:
        - CAP = 1410000000 (OceanToken.sol#638)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Detectors:
OceanToken.TOTALSUPPLY (OceanToken.sol#639) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-de
clared-constant
INFO:Slither:OceanToken.sol analyzed (13 contracts with 93 detectors), 21 result(s) found
```

Solidity Static Analysis

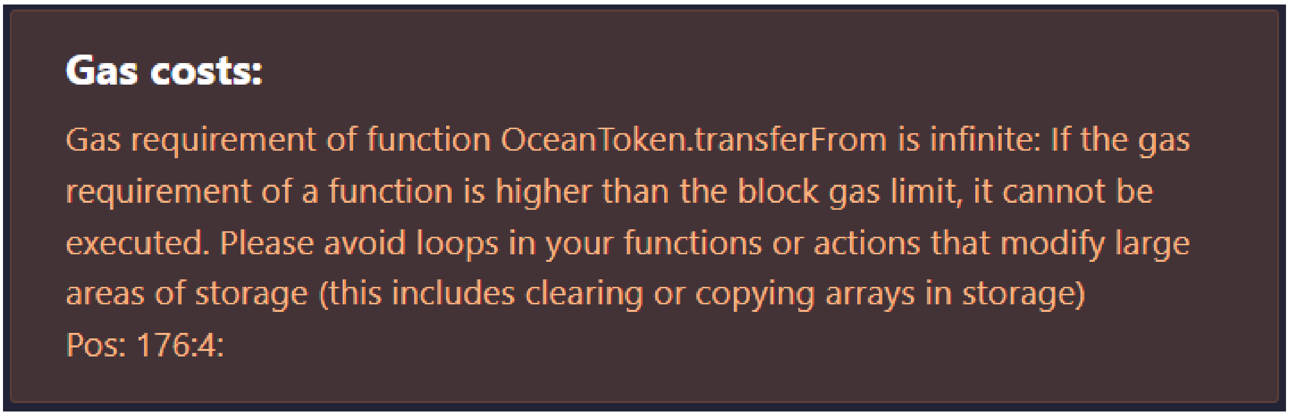
Static code analysis is used to identify many common coding problems before a program is released. It involves examining the code manually or using tools to automate the process. Static code analysis tools can automatically scan the code without executing it.

OceanToken.sol



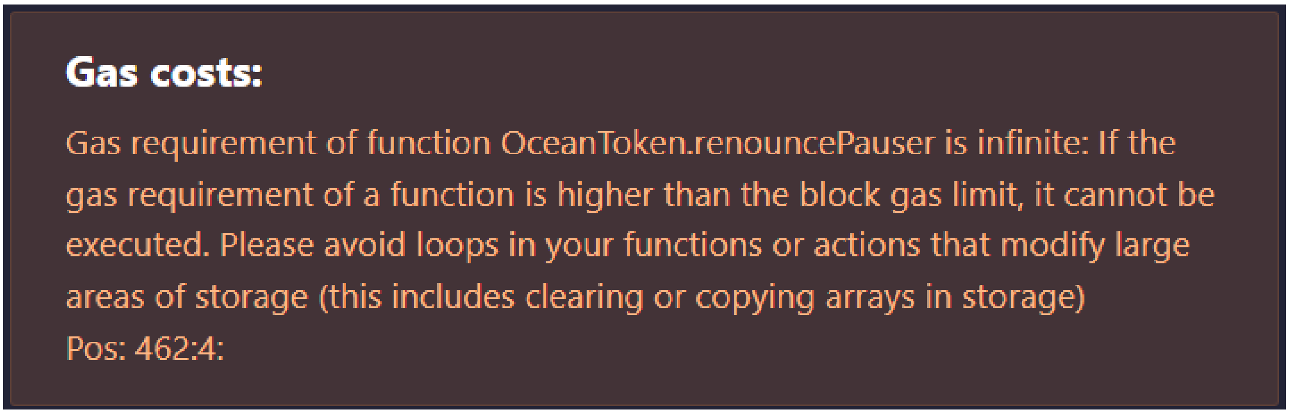
Selfdestruct:

Use of selfdestruct: Can block calling contracts unexpectedly. Be especially careful if this contract is planned to be used by other contracts (i.e. library contracts, interactions). Selfdestruction of the callee contract can leave callers in an inoperable state.
more
Pos: 764:8:



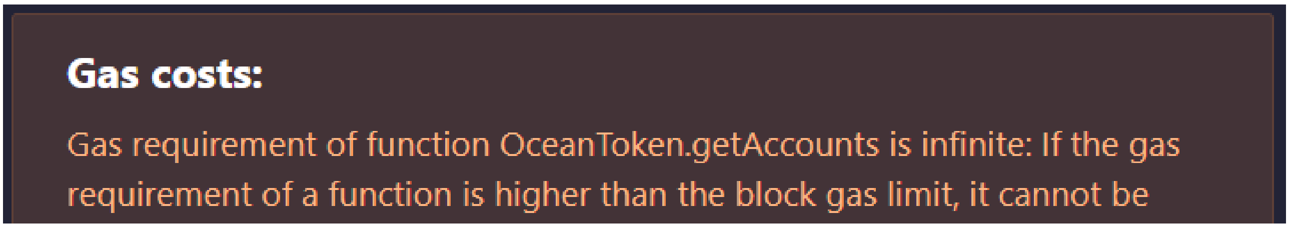
Gas costs:

Gas requirement of function OceanToken.transferFrom 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: 176:4:



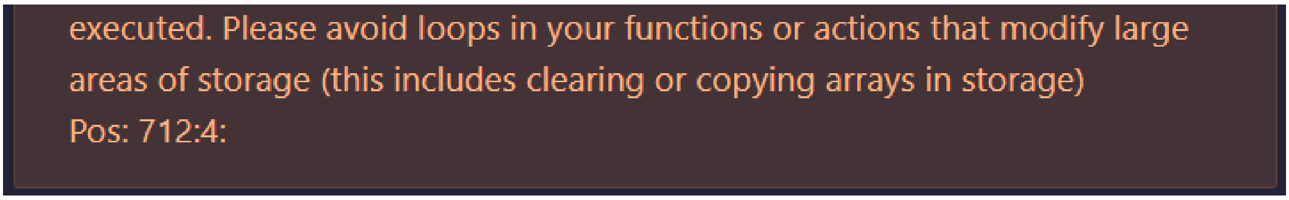
Gas costs:

Gas requirement of function OceanToken.renouncePauser 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: 462:4:

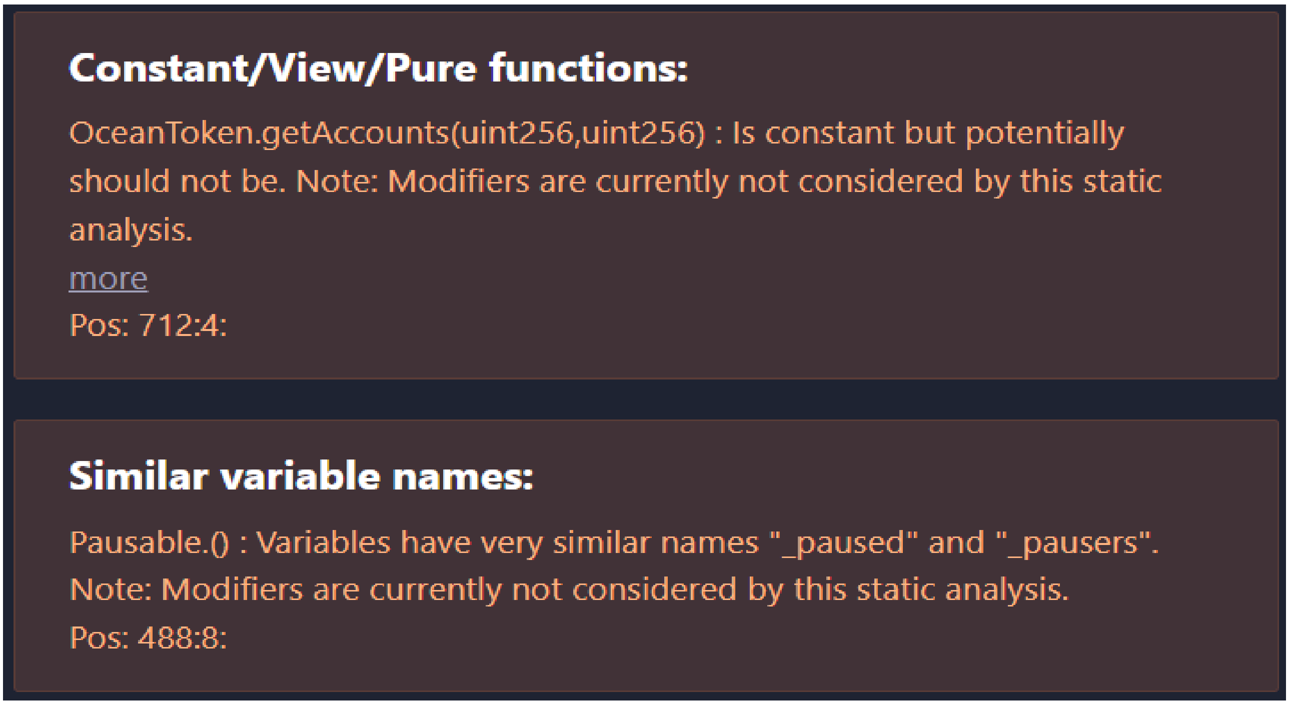


Gas costs:

Gas requirement of function OceanToken.getAccounts 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: 712:4:

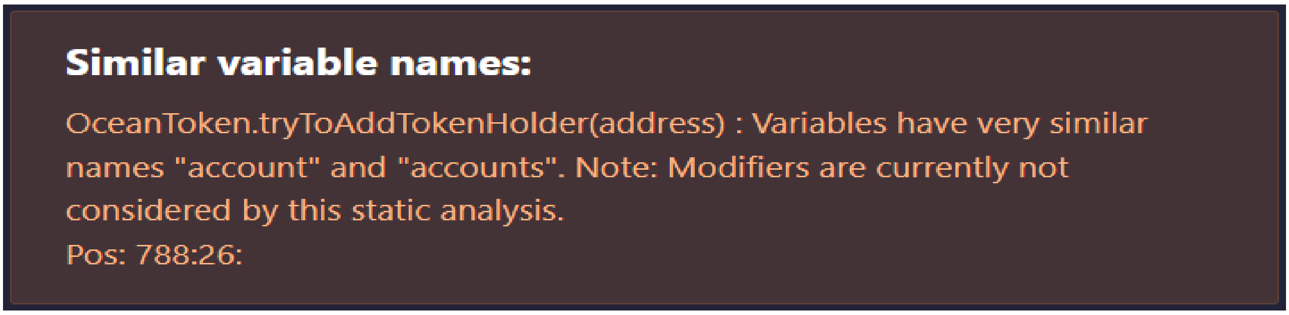


Constant/View/Pure functions:

OceanToken.getAccounts(uint256,uint256) : Is constant but potentially should not be. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 712:4:

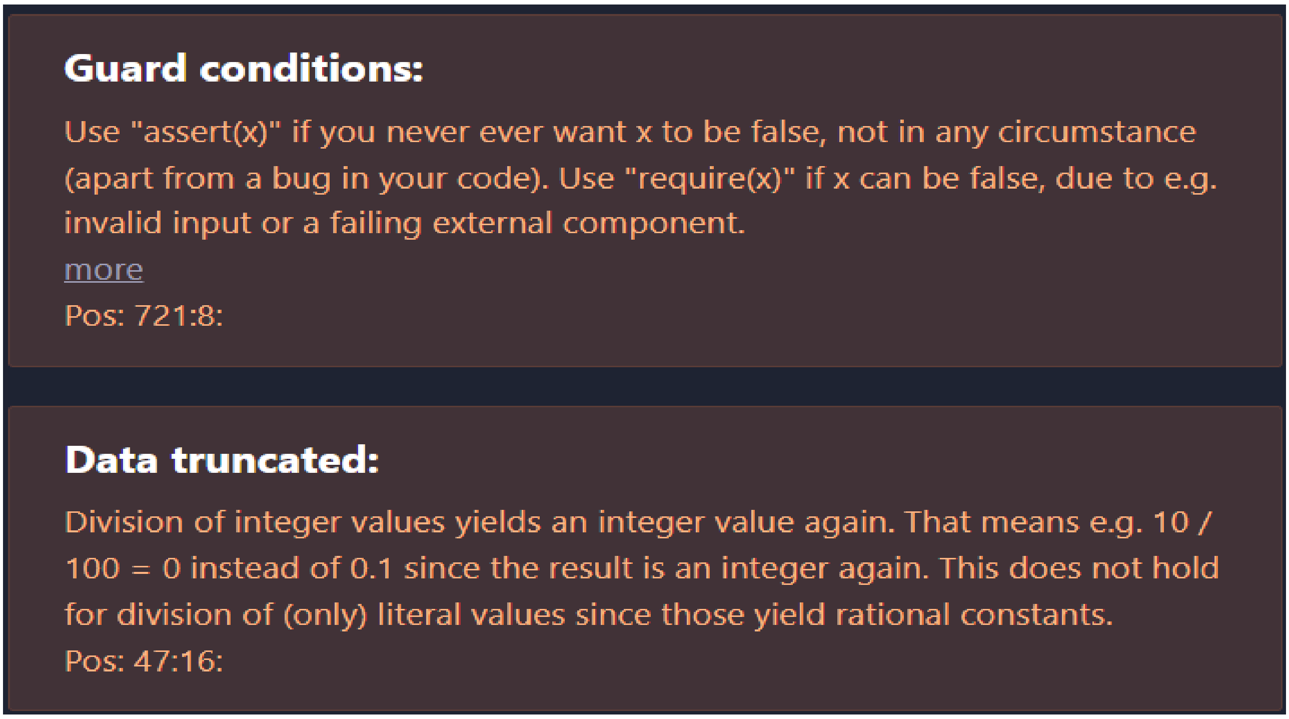
Similar variable names:

Pausable.() : Variables have very similar names "_paused" and "_pausers". Note: Modifiers are currently not considered by this static analysis.
Pos: 488:8:



Similar variable names:

OceanToken.tryToAddTokenHolder(address) : Variables have very similar names "account" and "accounts". Note: Modifiers are currently not considered by this static analysis.
Pos: 788:26:



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: 721:8:

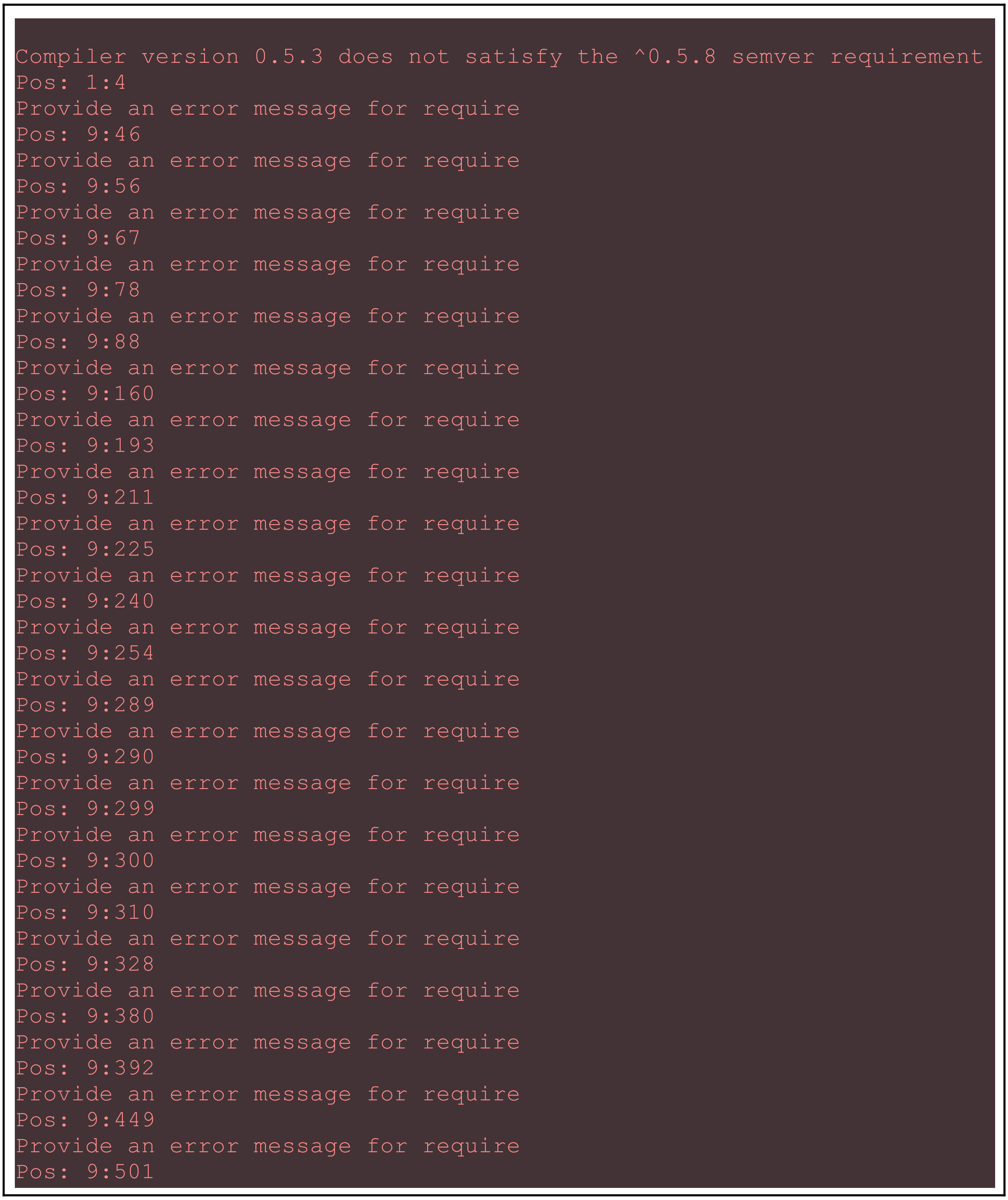
Data truncated:

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: 47:16:

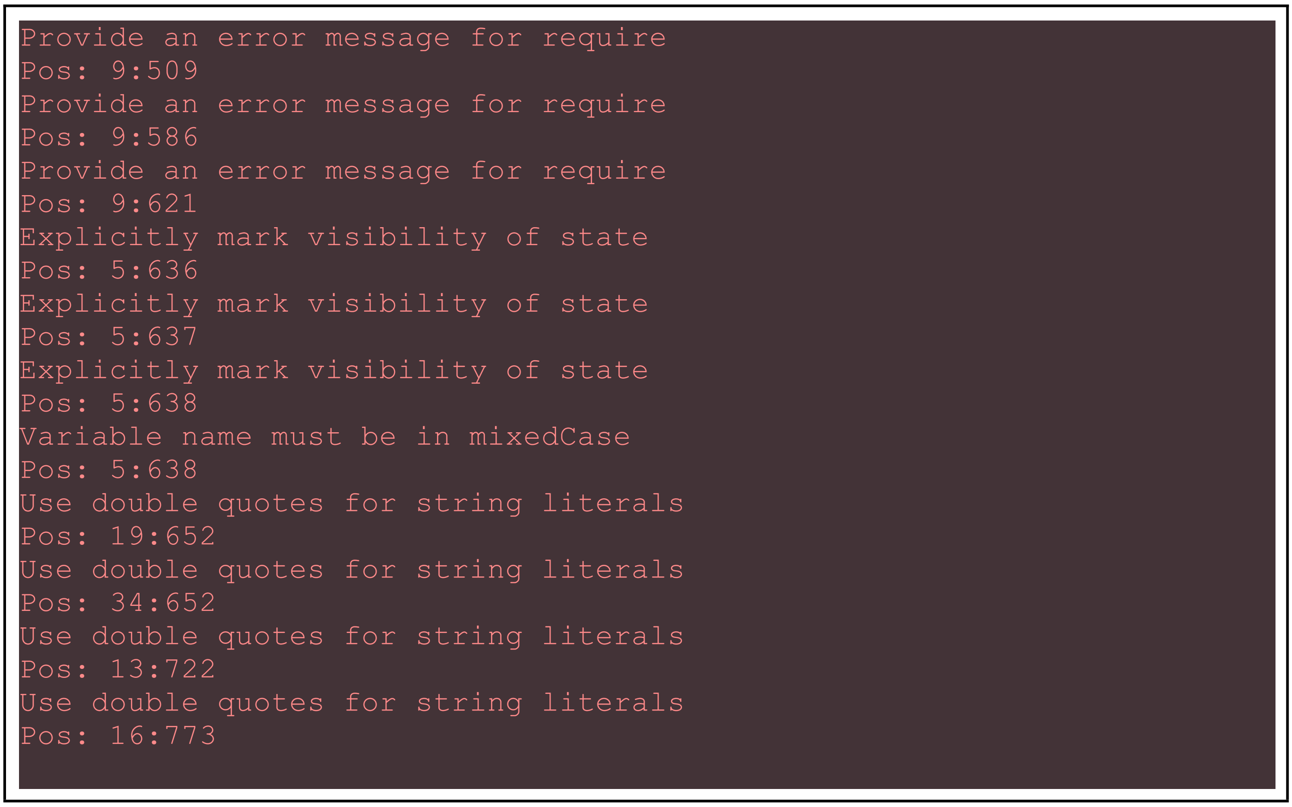
Solhint Linter

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.

OceanToken.sol



```
Compiler version 0.5.3 does not satisfy the ^0.5.8 semver requirement
Pos: 1:4
Provide an error message for require
Pos: 9:46
Provide an error message for require
Pos: 9:56
Provide an error message for require
Pos: 9:67
Provide an error message for require
Pos: 9:78
Provide an error message for require
Pos: 9:88
Provide an error message for require
Pos: 9:160
Provide an error message for require
Pos: 9:193
Provide an error message for require
Pos: 9:211
Provide an error message for require
Pos: 9:225
Provide an error message for require
Pos: 9:240
Provide an error message for require
Pos: 9:254
Provide an error message for require
Pos: 9:289
Provide an error message for require
Pos: 9:290
Provide an error message for require
Pos: 9:299
Provide an error message for require
Pos: 9:300
Provide an error message for require
Pos: 9:310
Provide an error message for require
Pos: 9:328
Provide an error message for require
Pos: 9:380
Provide an error message for require
Pos: 9:392
Provide an error message for require
Pos: 9:449
Provide an error message for require
Pos: 9:501
```



```
Provide an error message for require
Pos: 9:509
Provide an error message for require
Pos: 9:586
Provide an error message for require
Pos: 9:621
Explicitly mark visibility of state
Pos: 5:636
Explicitly mark visibility of state
Pos: 5:637
Explicitly mark visibility of state
Pos: 5:638
Variable name must be in mixedCase
Pos: 5:638
Use double quotes for string literals
Pos: 19:652
Use double quotes for string literals
Pos: 34:652
Use double quotes for string literals
Pos: 13:722
Use double quotes for string literals
Pos: 16:773
```

Software analysis result:

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

EtherAuthority

Email: audit@EtherAuthority.io