EtherAuthority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project:	OxyO2 Token
Platform:	Binance Smart Chain
Language:	Solidity
Date:	March 24th, 2022

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

EtherAuthority was contracted by the OxyO2 team to perform the Security audit of the OxyO2 Token smart contract 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 March 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

OxyO2 is a standard BEP20 token smart contract. This audit only considers the OxyO2 token smart contract, and does not cover any other smart contracts on the platform.

Audit scope

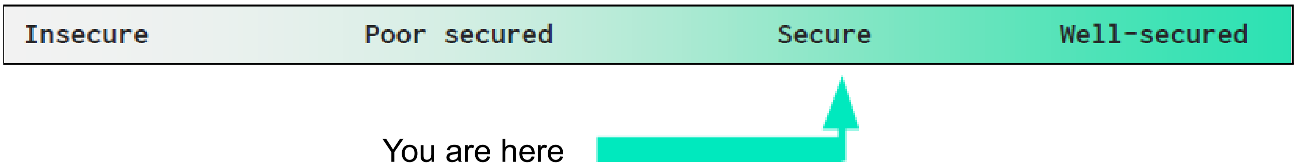
Name	**Code Review and Security Analysis Report for OxyO2 Token Smart Contract**
Platform	**BSC / Solidity**
File	OxyO2.sol
File MD5 Hash	7320B0214EA977A219073499E67A49A3
Online Code Link	0xbc10b37baff5819d5065312e019bd9c3dc68c90a
Audit Date	March 24th, 2022
Updated MD5 Hash	6E4123E86EACAD95D1324D8A141B96C9
Updated Online Code Link:	d33c784adc0ca0f12994a49a0c5213fa5810635e
Revise Audit Date	April 7th, 2022
Updated MD5 Hash	316F9B25347CC0068D7C6391410F0645
Updated Online Code Link:	0x4ff08F7F52Ddba3E78C7754331c1baE737b0C50d
Revise Audit Date	May 5th, 2022

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Tokenomics: ● Name: oxyO2 ● Symbol: OX2 ● Decimals: 18 ● Total Supply: 1 Billion	YES, This is valid.

Audit Summary

According to the standard audit assessment, Customer`s solidity smart contracts are **"Secured"**. This token contract does contain owner control, which does not make it 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 0 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	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 not set	Passed
	“Short Address” Attack	Passed
	“Double Spend” Attack	Passed

Overall Audit Result: PASSED

Code Quality

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

The libraries in OxyO2 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 OxyO2 Token.

The OxyO2 Token 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 **not well** commented on smart contracts.

Documentation

We were given a OxyO2 Token smart contract code in the form of a BSCScan Web Link.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 contract 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

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	recoverBEP20	write	Removed	
3	snapshot	write	access only Owner	No Issue
4	pause	write	access only Owner	No Issue
5	unpause	write	access only Owner	No Issue
6	_beforeTokenTransfer	internal	Passed	No Issue
7	_afterTokenTransfer	internal	Passed	No Issue
8	_mint	internal	Passed	No Issue
9	_burn	internal	Passed	No Issue
10	paused	read	Passed	No Issue
11	whenNotPaused	modifier	Passed	No Issue
12	whenPaused	modifier	Passed	No Issue
13	_pause	internal	Passed	No Issue
14	_unpause	internal	Passed	No Issue
15	owner	read	Passed	No Issue
16	onlyOwner	modifier	Passed	No Issue
17	renounceOwnership	read	access only Owner	No Issue
18	transferOwnership	write	access only Owner	No Issue
19	_transferOwnership	internal	Passed	No Issue
20	name	read	Passed	No Issue
21	symbol	read	Passed	No Issue
22	decimals	read	Passed	No Issue
23	totalSupply	read	Passed	No Issue
24	balanceOf	read	Passed	No Issue
25	transfer	write	Passed	No Issue
26	allowance	read	Passed	No Issue
27	approve	write	Passed	No Issue
28	transferFrom	write	Passed	No Issue
29	increaseAllowance	write	Passed	No Issue
30	decreaseAllowance	write	Passed	No Issue
31	_transfer	internal	Passed	No Issue
32	_mint	internal	Passed	No Issue
33	_burn	internal	Passed	No Issue
34	_approve	internal	Passed	No Issue
35	_beforeTokenTransfer	internal	Passed	No Issue
36	_afterTokenTransfer	internal	Passed	No Issue
37	permit	write	Passed	No Issue
38	nonces	read	Passed	No Issue
39	DOMAIN_SEPARATOR	external	Passed	No Issue
40	_useNonce	internal	Passed	No Issue
41	checkpoints	read	Passed	No Issue
42	numCheckpoints	read	Passed	No Issue

43	delegates	read	Passed	No Issue
44	getVotes	read	Passed	No Issue
45	getPastVotes	read	Passed	No Issue
46	getPastTotalSupply	read	Passed	No Issue
47	_checkpointsLookup	read	Passed	No Issue
48	delegate	write	Passed	No Issue
49	delegateBySig	write	Passed	No Issue
50	_maxSupply	internal	Passed	No Issue
51	_mint	internal	Passed	No Issue
52	_burn	internal	Passed	No Issue
53	_afterTokenTransfer	internal	Passed	No Issue
54	_delegate	internal	Passed	No Issue
55	_moveVotingPower	write	Passed	No Issue
56	_writeCheckpoint	write	Passed	No Issue
57	_add	write	Passed	No Issue
58	_subtract	write	Passed	No Issue
59	_snapshot	internal	Passed	No Issue
60	_getCurrentSnapshotId	internal	Passed	No Issue
61	balanceOfAt	read	Passed	No Issue
62	totalSupplyAt	read	Passed	No Issue
63	_beforeTokenTransfer	internal	Passed	No Issue
64	_valueAt	read	Passed	No Issue
65	_updateAccountSnapshot	write	Passed	No Issue
66	_updateTotalSupplySnapshot	write	Passed	No Issue
67	_updateSnapshot	write	Passed	No Issue
68	_lastSnapshotId	read	Passed	No Issue
69	burn	write	Passed	No Issue
70	burnFrom	write	Passed	No Issue
71	withdrawCoin	write	Removed	

Severity Definitions

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

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) Critical operation lacks event log:

Missing event log for: recoverBEP20

Resolution: We suggest writing an event log for listed events.

Status: Fixed

(2) Function input parameters lack of check:

Variable validation is not performed in the function “recoverBEP20”.

Resolution: We suggest using a validation for the address type variables that should not be address(0).

Status: Fixed

(3) Owner cannot drain BNB:

withdrawCoin function is used to drain BNB of the contract but as the contract does not accept BNB, the owner cannot drain BNB from the contract.

Resolution: We suggest adding the fallback function to accept BNB.

Status: Fixed

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:

- stakersbl: The Owner can blacklist a wallet address.
- stakersubl: The Owner can remove a wallet address from blacklist.

Conclusion

We were given a contract code. And we have used all possible tests based on given objects as files. We have not observed any major issues. So, **it's good to go to 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 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 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).

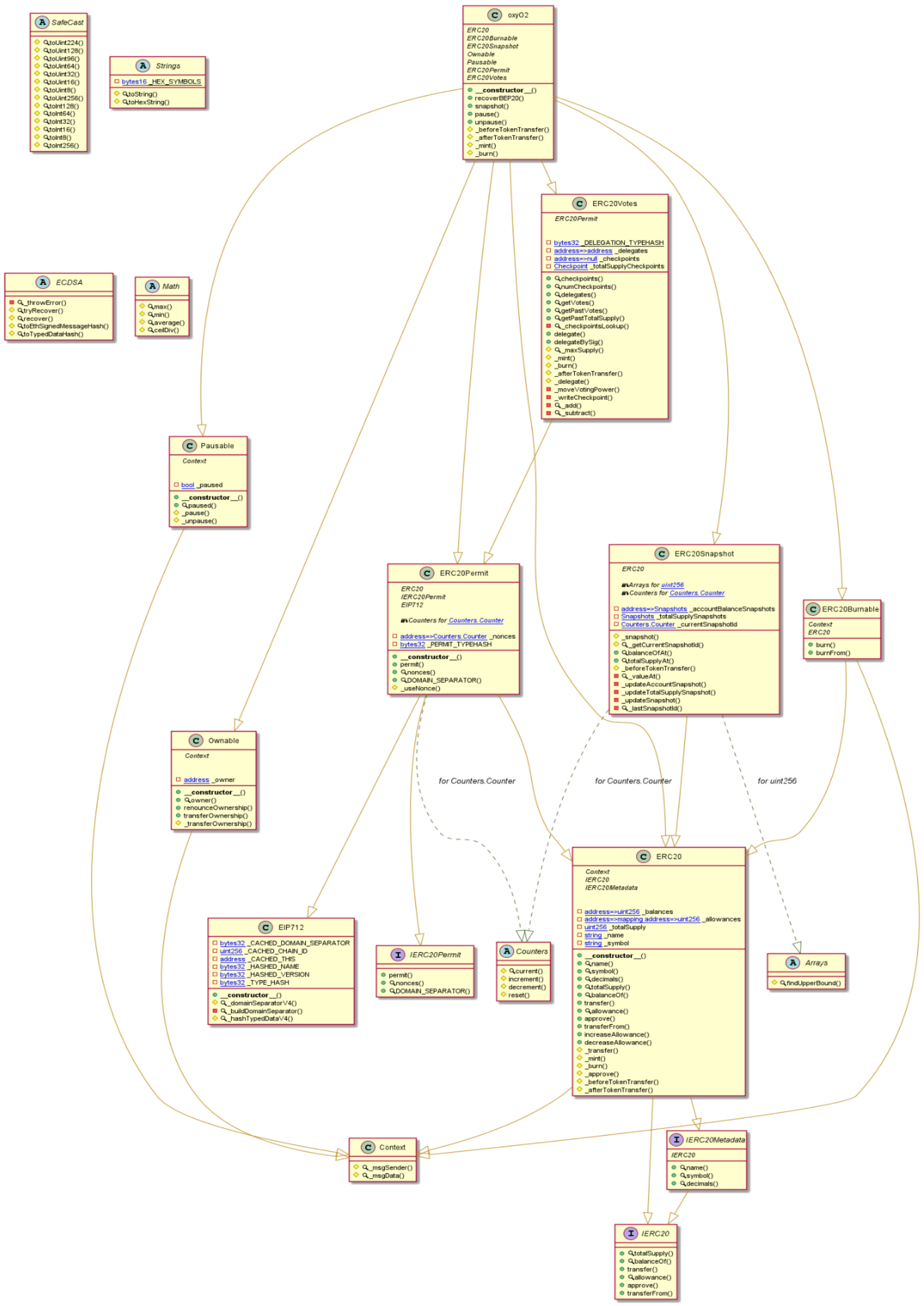
Due to the fact that the total number of test cases are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we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

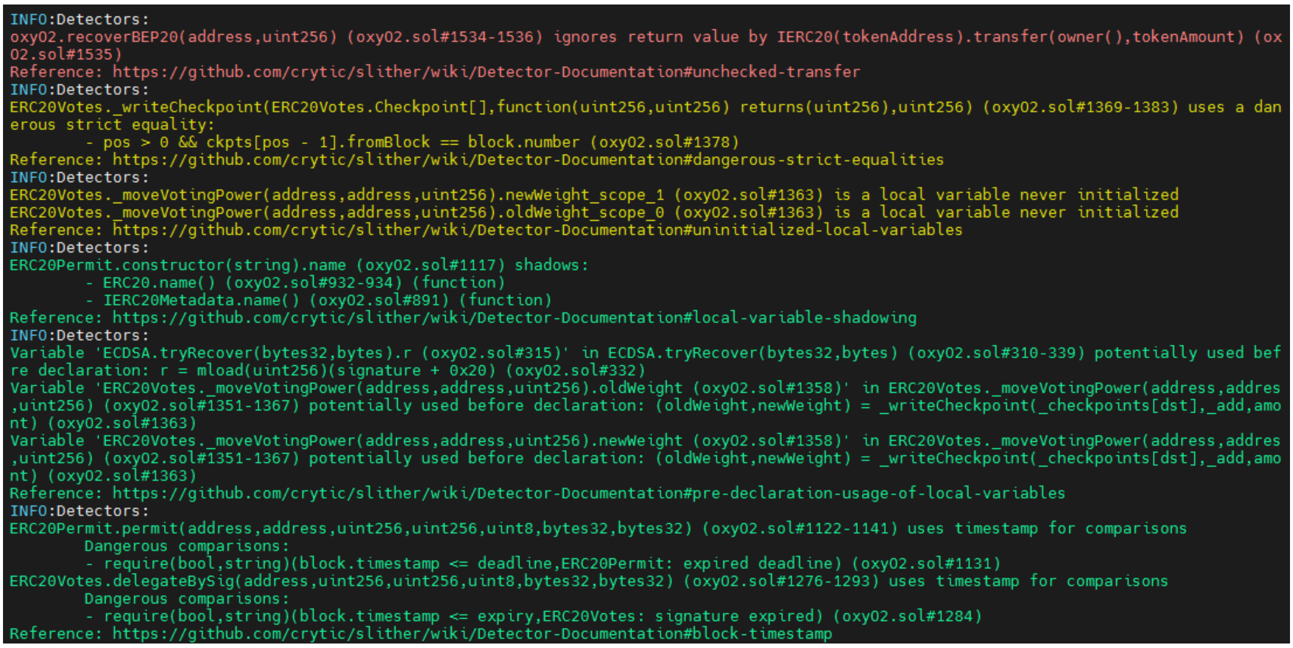
Appendix

Code Flow Diagram - OxyO2 Token

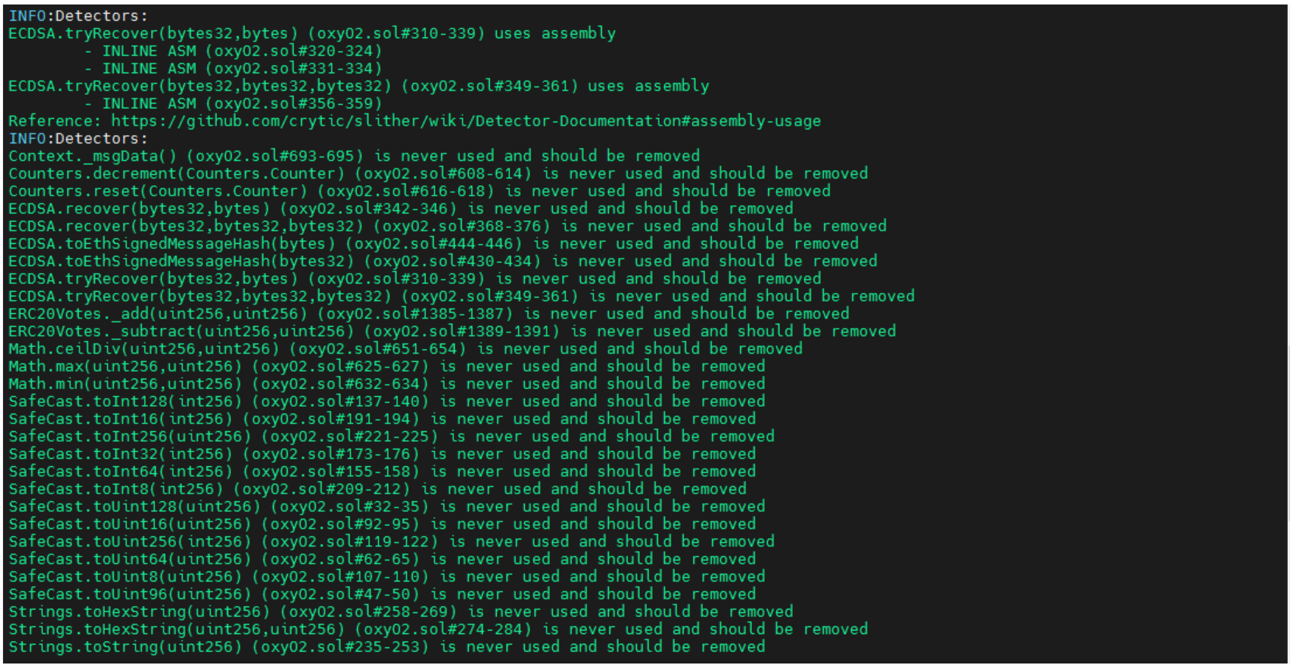


Slither Results Log

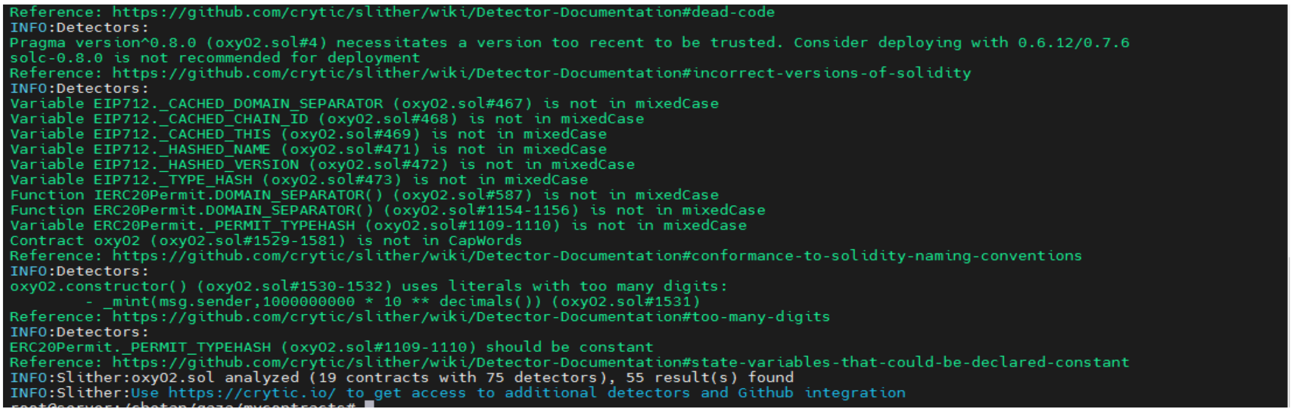
Slither log >> OxyO2.sol



```
INFO:Detectors:
oxy02.recoverBEP20(address,uint256) (oxy02.sol#1534-1536) ignores return value by IERC20(tokenAddress).transfer(owner(),tokenAmount) (ox
02.sol#1535)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer
INFO:Detectors:
ERC20Votes._writeCheckpoint(ERC20Votes.Checkpoint[],function(uint256,uint256) returns(uint256),uint256) (oxy02.sol#1369-1383) uses a dan
erous strict equality:
        - pos > 0 && ckpts[pos - 1].fromBlock == block.number (oxy02.sol#1378)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
ERC20Votes._moveVotingPower(address,address,uint256).newWeight_scope_1 (oxy02.sol#1363) is a local variable never initialized
ERC20Votes._moveVotingPower(address,address,uint256).oldWeight_scope_0 (oxy02.sol#1363) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables
INFO:Detectors:
ERC20Permit.constructor(string).name (oxy02.sol#1117) shadows:
        - ERC20.name() (oxy02.sol#932-934) (function)
        - IERC20Metadata.name() (oxy02.sol#891) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
Variable 'ECDSA.tryRecover(bytes32,bytes).r (oxy02.sol#315)' in ECDSA.tryRecover(bytes32,bytes) (oxy02.sol#310-339) potentially used bef
re declaration: r = mload(uint256)(signature + 0x20) (oxy02.sol#332)
Variable 'ERC20Votes._moveVotingPower(address,address,uint256).oldWeight (oxy02.sol#1358)' in ERC20Votes._moveVotingPower(address,addres
,uint256) (oxy02.sol#1351-1367) potentially used before declaration: (oldWeight,newWeight) = _writeCheckpoint(_checkpoints[dst],_add,amo
nt) (oxy02.sol#1363)
Variable 'ERC20Votes._moveVotingPower(address,address,uint256).newWeight (oxy02.sol#1358)' in ERC20Votes._moveVotingPower(address,addres
,uint256) (oxy02.sol#1351-1367) potentially used before declaration: (oldWeight,newWeight) = _writeCheckpoint(_checkpoints[dst],_add,amo
nt) (oxy02.sol#1363)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#pre-declaration-usage-of-local-variables
INFO:Detectors:
ERC20Permit.permit(address,address,uint256,uint256,uint8,bytes32,bytes32) (oxy02.sol#1122-1141) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp <= deadline,ERC20Permit: expired deadline) (oxy02.sol#1131)
ERC20Votes.delegateBySig(address,uint256,uint256,uint8,bytes32,bytes32) (oxy02.sol#1276-1293) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp <= expiry,ERC20Votes: signature expired) (oxy02.sol#1284)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
```



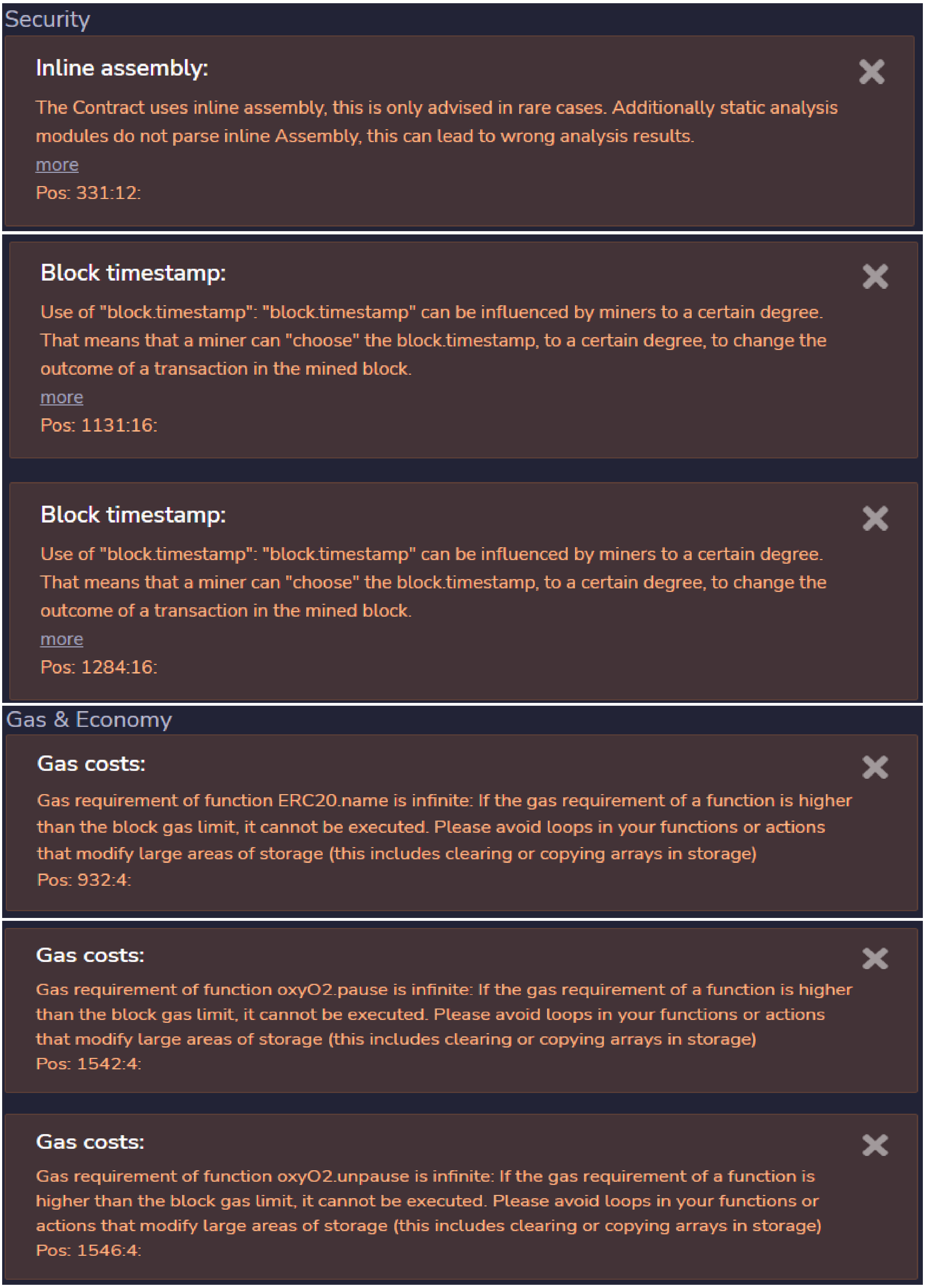
```
INFO:Detectors:
ECDSA.tryRecover(bytes32,bytes) (oxy02.sol#310-339) uses assembly
        - INLINE ASM (oxy02.sol#320-324)
        - INLINE ASM (oxy02.sol#331-334)
ECDSA.tryRecover(bytes32,bytes32,bytes32) (oxy02.sol#349-361) uses assembly
        - INLINE ASM (oxy02.sol#356-359)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
Context._msgData() (oxy02.sol#693-695) is never used and should be removed
Counters.decrement(Counters.Counter) (oxy02.sol#608-614) is never used and should be removed
Counters.reset(Counters.Counter) (oxy02.sol#616-618) is never used and should be removed
ECDSA.recover(bytes32,bytes) (oxy02.sol#342-346) is never used and should be removed
ECDSA.recover(bytes32,bytes32,bytes32) (oxy02.sol#368-376) is never used and should be removed
ECDSA.toEthSignedMessageHash(bytes) (oxy02.sol#444-446) is never used and should be removed
ECDSA.toEthSignedMessageHash(bytes32) (oxy02.sol#430-434) is never used and should be removed
ECDSA.tryRecover(bytes32,bytes) (oxy02.sol#310-339) is never used and should be removed
ECDSA.tryRecover(bytes32,bytes32,bytes32) (oxy02.sol#349-361) is never used and should be removed
ERC20Votes._add(uint256,uint256) (oxy02.sol#1385-1387) is never used and should be removed
ERC20Votes._subtract(uint256,uint256) (oxy02.sol#1389-1391) is never used and should be removed
Math.ceilDiv(uint256,uint256) (oxy02.sol#651-654) is never used and should be removed
Math.max(uint256,uint256) (oxy02.sol#625-627) is never used and should be removed
Math.min(uint256,uint256) (oxy02.sol#632-634) is never used and should be removed
SafeCast.toInt128(int256) (oxy02.sol#137-140) is never used and should be removed
SafeCast.toInt16(int256) (oxy02.sol#191-194) is never used and should be removed
SafeCast.toInt256(uint256) (oxy02.sol#221-225) is never used and should be removed
SafeCast.toInt32(int256) (oxy02.sol#173-176) is never used and should be removed
SafeCast.toInt64(int256) (oxy02.sol#155-158) is never used and should be removed
SafeCast.toInt8(int256) (oxy02.sol#209-212) is never used and should be removed
SafeCast.toUint128(uint256) (oxy02.sol#32-35) is never used and should be removed
SafeCast.toUint16(uint256) (oxy02.sol#92-95) is never used and should be removed
SafeCast.toUint256(int256) (oxy02.sol#119-122) is never used and should be removed
SafeCast.toUint64(uint256) (oxy02.sol#62-65) is never used and should be removed
SafeCast.toUint8(uint256) (oxy02.sol#107-110) is never used and should be removed
SafeCast.toUint96(uint256) (oxy02.sol#47-50) is never used and should be removed
Strings.toHexString(uint256) (oxy02.sol#258-269) is never used and should be removed
Strings.toHexString(uint256,uint256) (oxy02.sol#274-284) is never used and should be removed
Strings.toString(uint256) (oxy02.sol#235-253) is never used and should be removed
```



```
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version^0.8.0 (oxy02.sol#4) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
solc-0.8.0 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Variable EIP712._CACHED_DOMAIN_SEPARATOR (oxy02.sol#467) is not in mixedCase
Variable EIP712._CACHED_CHAIN_ID (oxy02.sol#468) is not in mixedCase
Variable EIP712._CACHED_THIS (oxy02.sol#469) is not in mixedCase
Variable EIP712._HASHED_NAME (oxy02.sol#471) is not in mixedCase
Variable EIP712._HASHED_VERSION (oxy02.sol#472) is not in mixedCase
Variable EIP712._TYPE_HASH (oxy02.sol#473) is not in mixedCase
Function IERC20Permit.DOMAIN_SEPARATOR() (oxy02.sol#587) is not in mixedCase
Function ERC20Permit.DOMAIN_SEPARATOR() (oxy02.sol#1154-1156) is not in mixedCase
Variable ERC20Permit._PERMIT_TYPEHASH (oxy02.sol#1109-1110) is not in mixedCase
Contract oxy02 (oxy02.sol#1529-1581) is not in CapWords
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
oxy02.constructor() (oxy02.sol#1530-1532) uses literals with too many digits:
        - _mint(msg.sender,1000000000 * 10 ** decimals()) (oxy02.sol#1531)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Detectors:
ERC20Permit._PERMIT_TYPEHASH (oxy02.sol#1109-1110) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
INFO:Slither:oxy02.sol analyzed (19 contracts with 75 detectors), 55 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Solidity Static Analysis

OxyO2.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: 331:12:

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: 1131:16:

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: 1284:16:

Gas & Economy

Gas costs:

Gas requirement of function ERC20.name 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: 932:4:

Gas costs:

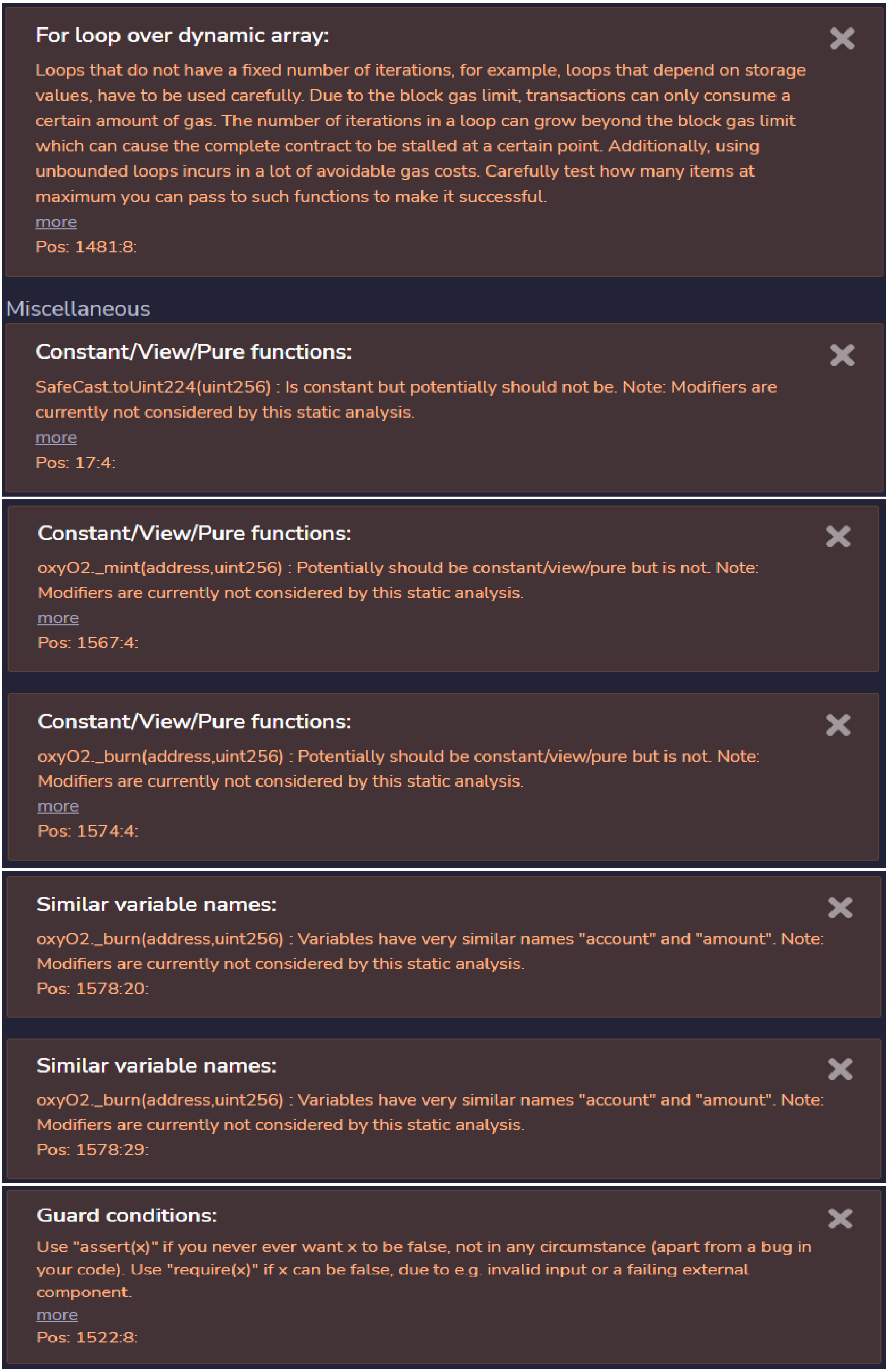
Gas requirement of function oxyO2.pause is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Pos: 1542:4:

Gas costs:

Gas requirement of function oxyO2.unpause 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: 1546:4:



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: 1481:8:

Miscellaneous

Constant/View/Pure functions:

SafeCast.toUint224(uint256) : Is constant but potentially should not be. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 17:4:

Constant/View/Pure functions:

oxyO2._mint(address,uint256) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 1567:4:

Constant/View/Pure functions:

oxyO2._burn(address,uint256) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 1574:4:

Similar variable names:

oxyO2._burn(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.

Pos: 1578:20:

Similar variable names:

oxyO2._burn(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.

Pos: 1578:29:

Guard conditions:

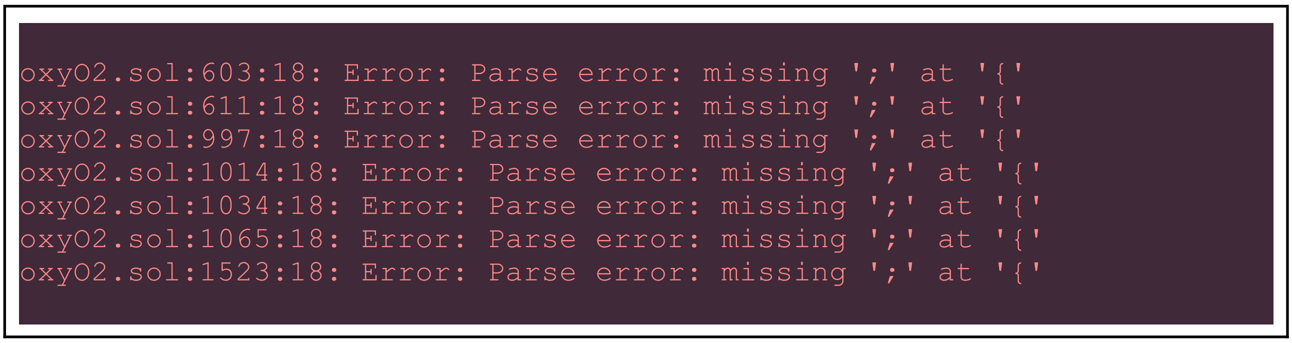
Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.

more

Pos: 1522:8:

Solhint Linter

OxyO2.sol



```
oxyO2.sol:603:18: Error: Parse error: missing ';' at '{'
oxyO2.sol:611:18: Error: Parse error: missing ';' at '{'
oxyO2.sol:997:18: Error: Parse error: missing ';' at '{'
oxyO2.sol:1014:18: Error: Parse error: missing ';' at '{'
oxyO2.sol:1034:18: Error: Parse error: missing ';' at '{'
oxyO2.sol:1065:18: Error: Parse error: missing ';' at '{'
oxyO2.sol:1523:18: Error: Parse error: missing ';' at '{'
```

Software analysis result:

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

EtherAuthority

Email: audit@EtherAuthority.io