Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project:	Immutable x Token
Website:	imx.community
Platform:	Ethereum
Language:	Solidity
Date:	May 16th, 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 smart contracts of Immutable x Token from imx.community were audited. The audit has been performed using manual analysis as well as using automated software tools. This report presents all the findings regarding the audit performed on May 16th, 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

- The Solidity code outlines IMXToken, an ERC20 token with additional functionalities, which can be broken down into its key components.
  - **ERC20Capped Extension:** The contract inherits from ERC20Capped, which extends ERC20 with a supply cap. This means the total supply of tokens cannot exceed a predefined cap. The cap is set to 2 billion IMX tokens.
  - **Access Control:** The contract uses the OpenZeppelin Access Control library to manage roles. It defines a MINTER_ROLE which is necessary to mint new tokens. The contract constructor sets up the initial minter by assigning MINTER_ROLE to the provided minter address.
  - **Minting Functionality:** The mint function allows the minting of new tokens by addresses with the MINTER_ROLE. It checks if the caller has the required role before minting tokens.
  - **Modifiers:** The checkRole modifier is used to ensure that only addresses with a specific role (in this case, the minter role) can call certain functions.
- Here's a summary of the roles and their responsibilities:
  - **DEFAULT_ADMIN_ROLE:** Admin role with the highest privileges. By default, it can grant or revoke other roles.
  - **MINTER_ROLE:** Role responsible for minting new tokens.
- Overall, this contract provides a way to create and manage a capped ERC20 token with a role-based access control mechanism for minting new tokens.

Audit scope

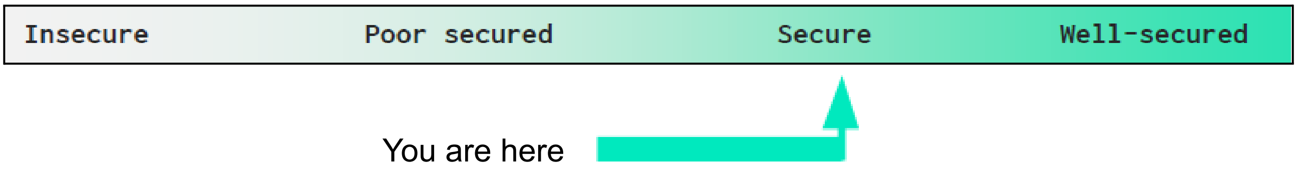
Name	Code Review and Security Analysis Report for Immutable x Token Smart Contract
Platform	Ethereum
File	IMXToken.sol
Smart Contract Code	0xf57e7e7c23978c3caec3c3548e3d615c346e79ff
Audit Date	May 16th, 2024

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Tokenomics: • Name: Immutable x • Symbol: IMX • Decimals: 18 • Total Supply: 2 billion	YES, This is valid.
Ownership control: • Mint a new token by the minter role owner. • Grants `role` to `account` can be set by the owner. • Revokes `role` from `account` by the owner. • Renounce Role from `account` by the owner.	YES, This is valid.

Audit Summary

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



We used various tools like Slither, Solhint, and Remix IDE. At the same time, this finding is based on a critical analysis of the manual audit.

All issues found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the Audit Overview section. A general overview is presented in the AS-IS section and all identified issues can be found in the Audit overview section.

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

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

Technical Quick Stats

Main Category	Subcategory	Result
Contract Programming	The solidity version is not specified	Passed
	The solidity version is too old	Moderated
	Integer overflow/underflow	Passed
	Function input parameters lack check	Passed
	Function input parameters check bypass	Passed
	Function access control lacks management	Passed
	Critical operation lacks event log	Passed
	Human/contract checks bypass	Passed
	Random number generation/use vulnerability	N/A
	Fallback function misuse	Passed
	Race condition	Passed
	Logical vulnerability	Passed
	Features claimed	Passed
	Other programming issues	Moderated
Code Specification	Function visibility not explicitly declared	Passed
	Var. storage location not explicitly declared	Passed
	Use keywords/functions to be deprecated	Passed
	Unused code	Moderated
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	Not Detected
🟢 Fee Check	No
🟢 Is Honeypot	Not Detected
🟢 Trading Cooldown	Not Detected
🟢 Can Pause Trade?	No
🟢 Pause Transfer?	No
🟢 Max Tax?	No
🟢 Is it Anti-whale?	No
🟢 Is Anti-bot?	Not Detected
🟢 Is it a Blacklist?	Not Detected
🟢 Blacklist Check	No
🟢 Can Mint?	Yes
🟢 Is it a Proxy?	Not Detected
🟢 Can Take Ownership?	No
🟢 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 contracts contain Libraries, Smart contracts, inherits, and Interfaces. This is a compact and well-written smart contract.

The libraries in Immutable x 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 Immutable x 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 Immutable x 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

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	checkRole	modifier	Passed	No Issue
3	mint	external	checkRole	No Issue
4	supportsInterface	read	Passed	No Issue
5	hasRole	read	Passed	No Issue
6	getRoleAdmin	read	Passed	No Issue
7	grantRole	write	Passed	No Issue
8	revokeRole	write	Passed	No Issue
9	renounceRole	write	Passed	No Issue
10	_setupRole	internal	Passed	No Issue
11	_setRoleAdmin	internal	Passed	No Issue
12	_grantRole	write	Passed	No Issue
13	_revokeRole	write	Passed	No Issue
14	cap	read	Passed	No Issue
15	_mint	internal	Passed	No Issue
16	supportsInterface	read	Passed	No Issue
17	name	read	Passed	No Issue
18	symbol	read	Passed	No Issue
19	decimals	read	Passed	No Issue
20	totalSupply	read	Passed	No Issue
21	balanceOf	read	Passed	No Issue
22	transfer	write	Passed	No Issue
23	allowance	read	Passed	No Issue
24	approve	write	Passed	No Issue
25	transferFrom	write	Passed	No Issue
26	increaseAllowance	write	Passed	No Issue
27	decreaseAllowance	write	Passed	No Issue
28	_transfer	internal	Passed	No Issue
29	_mint	internal	Passed	No Issue
30	_burn	internal	Passed	No Issue
31	_approve	internal	Passed	No Issue
32	_beforeTokenTransfer	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

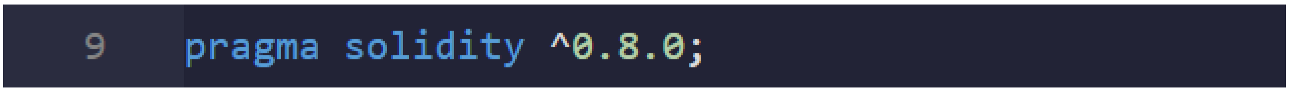
No Medium-severity vulnerabilities were found.

Low

No Low severity vulnerabilities were found.

Very Low / Informational / Best practices:

(1) Use the latest solidity version:

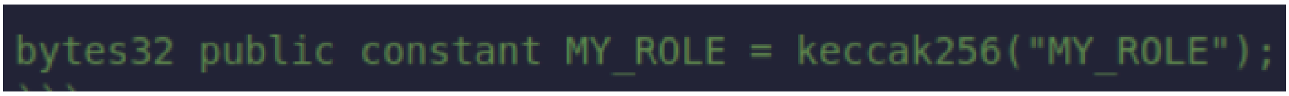


```
pragma solidity ^0.8.0;
```

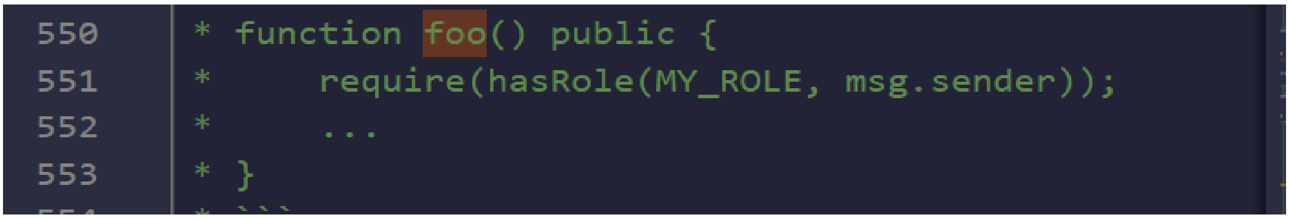
Use the latest solidity version while contract deployment to prevent any compiler version-level bugs.

Resolution: Please use versions greater than 0.8.7.

(2) Unwanted comments: **IAccessControl.sol**



```
bytes32 public constant MY_ROLE = keccak256("MY_ROLE");
```



```
* function foo() public {
*     require(hasRole(MY_ROLE, msg.sender));
*     ...
* }
```

Unwanted comments found in code.

Resolution: We suggest removing unwanted comments.

Centralization

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

AccessControl.sol

- mint: Mint a new token by the minter role owner.

AccessControl.sol

- grantRole: Grants `role` to `account` can be set by the owner.
- revokeRole: Revokes `role` from `account` by the owner.
- renounceRole: Renounce Role from `account` by the owner.

To make the smart contract 100% decentralized, we suggest renouncing ownership of 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 2 informational issues in the smart contracts. And those issues are not critical. So, **it's good to go for the production**.

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

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

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

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

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, 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).

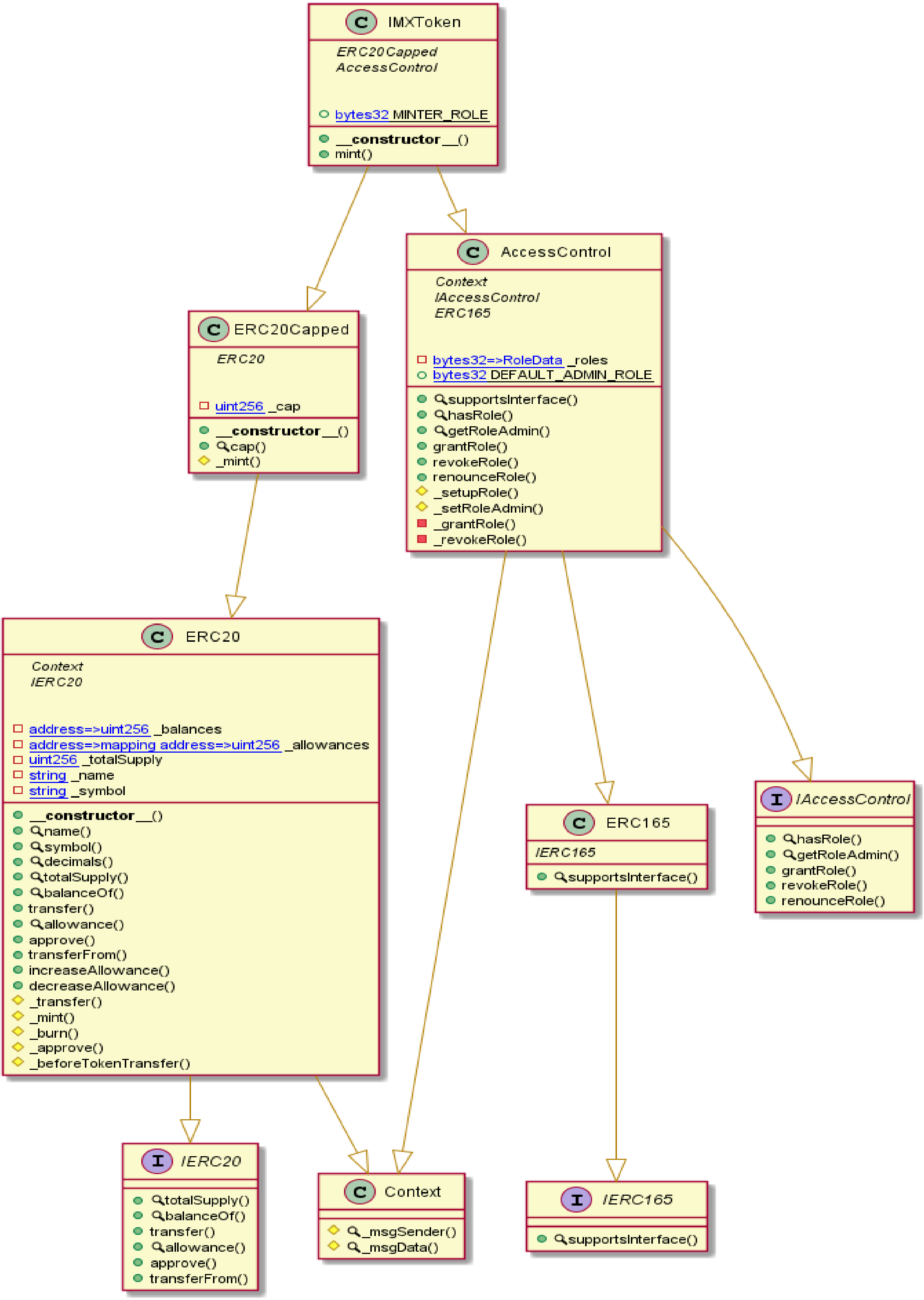
Due to the fact that the total number of test cases is unlimited, the audit makes no statements or warranties on the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. While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

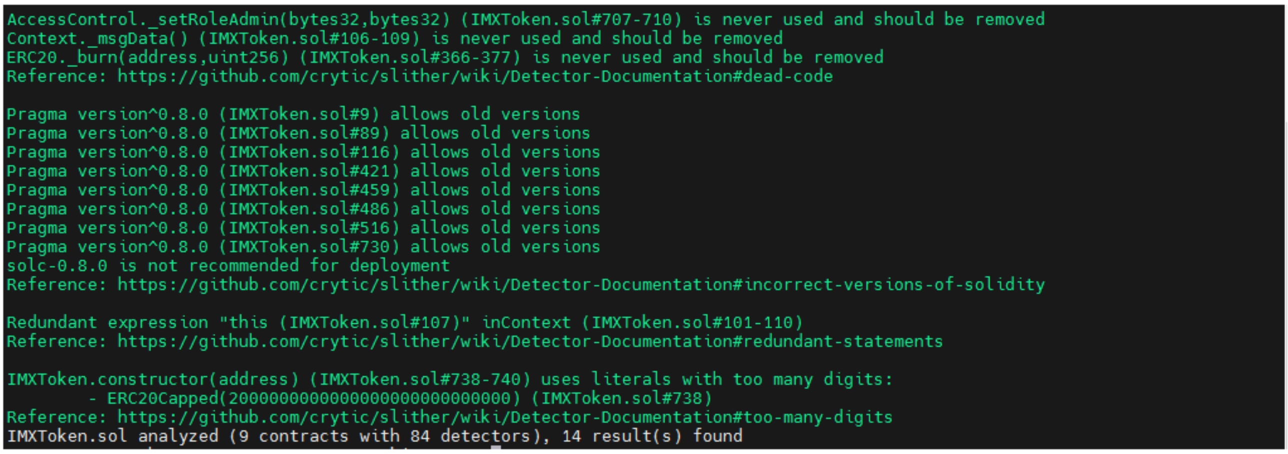


Slither Results Log

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 >> IMXToken.sol

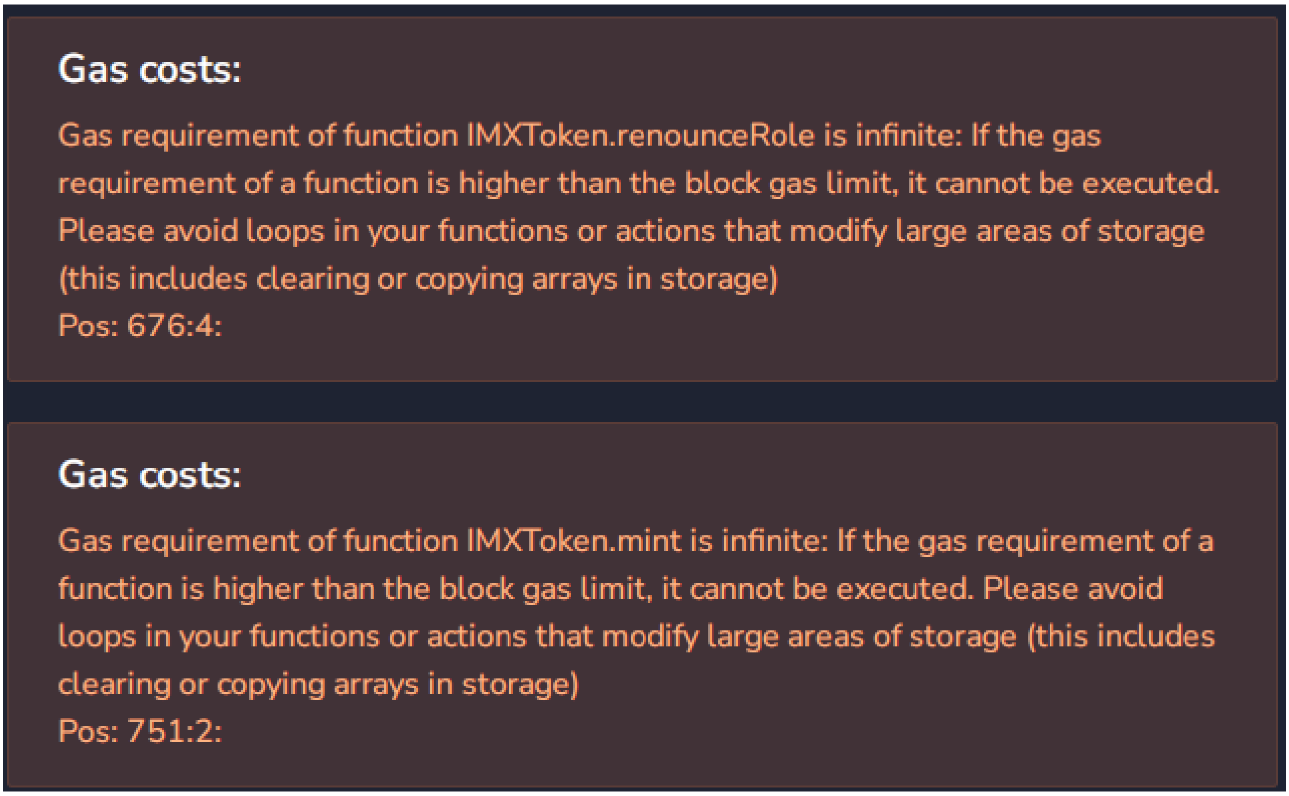


```
AccessControl._setRoleAdmin(bytes32,bytes32) (IMXToken.sol#707-710) is never used and should be removed
Context._msgData() (IMXToken.sol#106-109) is never used and should be removed
ERC20._burn(address,uint256) (IMXToken.sol#366-377) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version^0.8.0 (IMXToken.sol#9) allows old versions
Pragma version^0.8.0 (IMXToken.sol#89) allows old versions
Pragma version^0.8.0 (IMXToken.sol#116) allows old versions
Pragma version^0.8.0 (IMXToken.sol#421) allows old versions
Pragma version^0.8.0 (IMXToken.sol#459) allows old versions
Pragma version^0.8.0 (IMXToken.sol#486) allows old versions
Pragma version^0.8.0 (IMXToken.sol#516) allows old versions
Pragma version^0.8.0 (IMXToken.sol#730) allows old versions
solc-0.8.0 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Redundant expression "this (IMXToken.sol#107)" inContext (IMXToken.sol#101-110)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements

IMXToken.constructor(address) (IMXToken.sol#738-740) uses literals with too many digits:
        - ERC20Capped(2000000000000000000000000000) (IMXToken.sol#738)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
IMXToken.sol analyzed (9 contracts with 84 detectors), 14 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.

IMXToken.sol

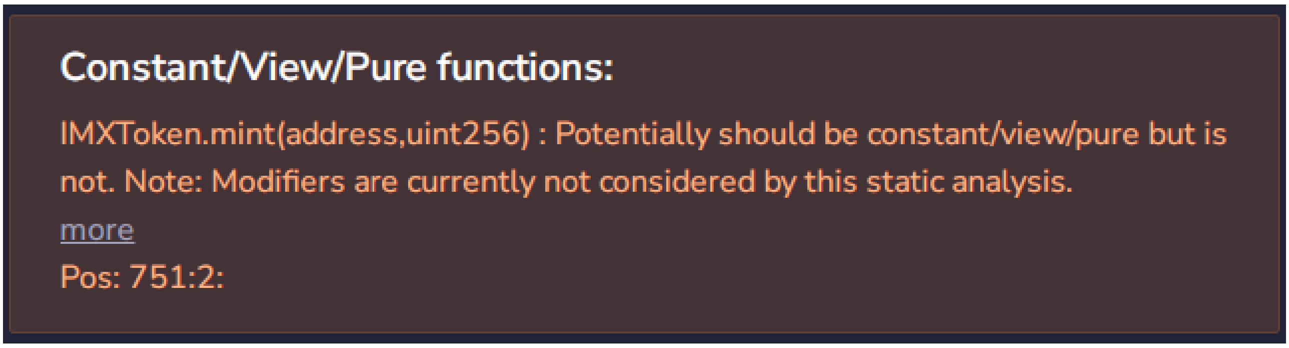


Gas costs:

Gas requirement of function IMXToken.renounceRole 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: 676:4:

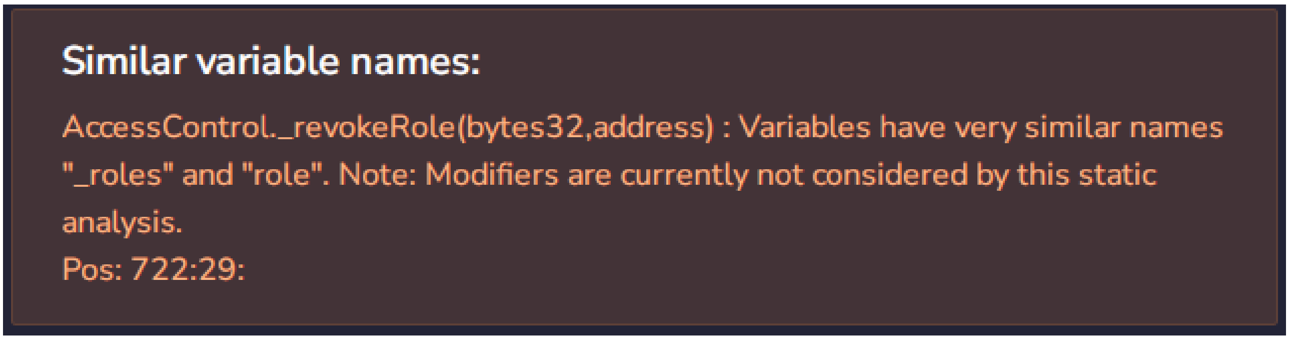
Gas costs:

Gas requirement of function IMXToken.mint 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: 751:2:



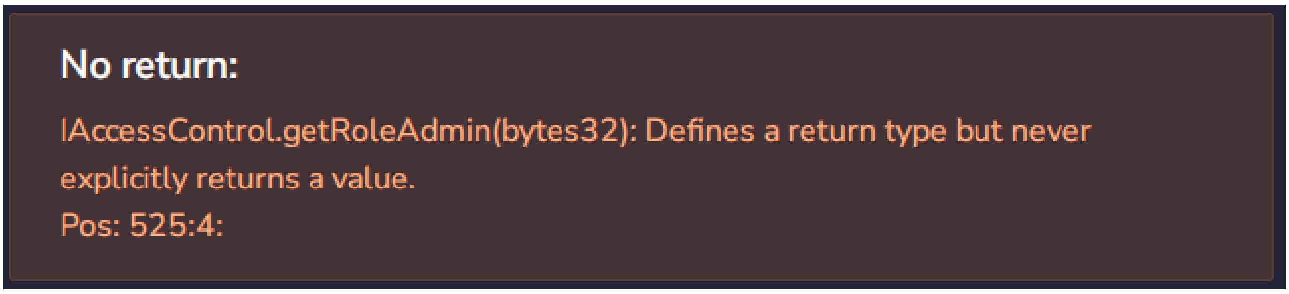
Constant/View/Pure functions:

IMXToken.mint(address,uint256) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 751:2:



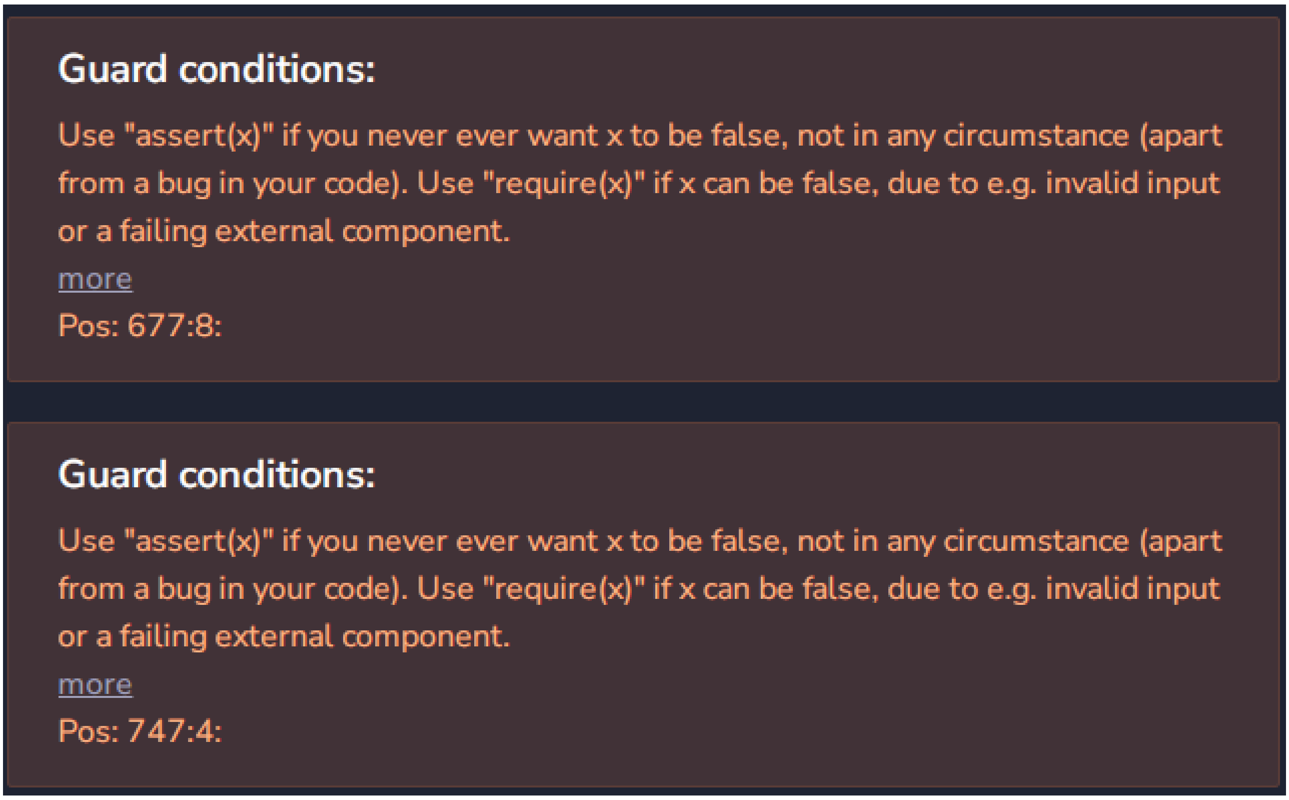
Similar variable names:

AccessControl._revokeRole(bytes32,address) : Variables have very similar names "_roles" and "role". Note: Modifiers are currently not considered by this static analysis.
Pos: 722:29:



No return:

IAccessControl.getRoleAdmin(bytes32): Defines a return type but never explicitly returns a value.
Pos: 525:4:



Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 677:8:

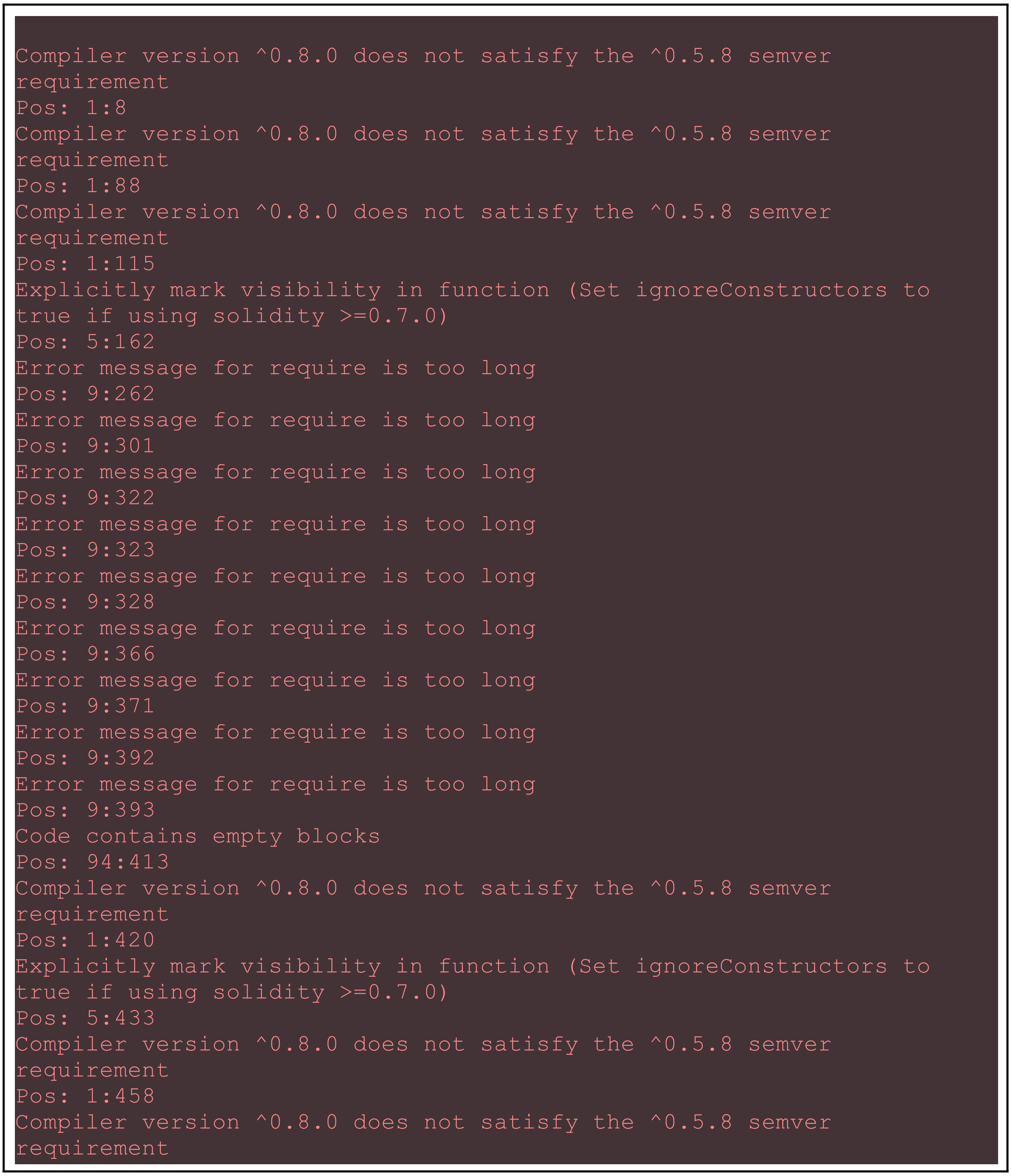
Guard conditions:

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: 747:4:

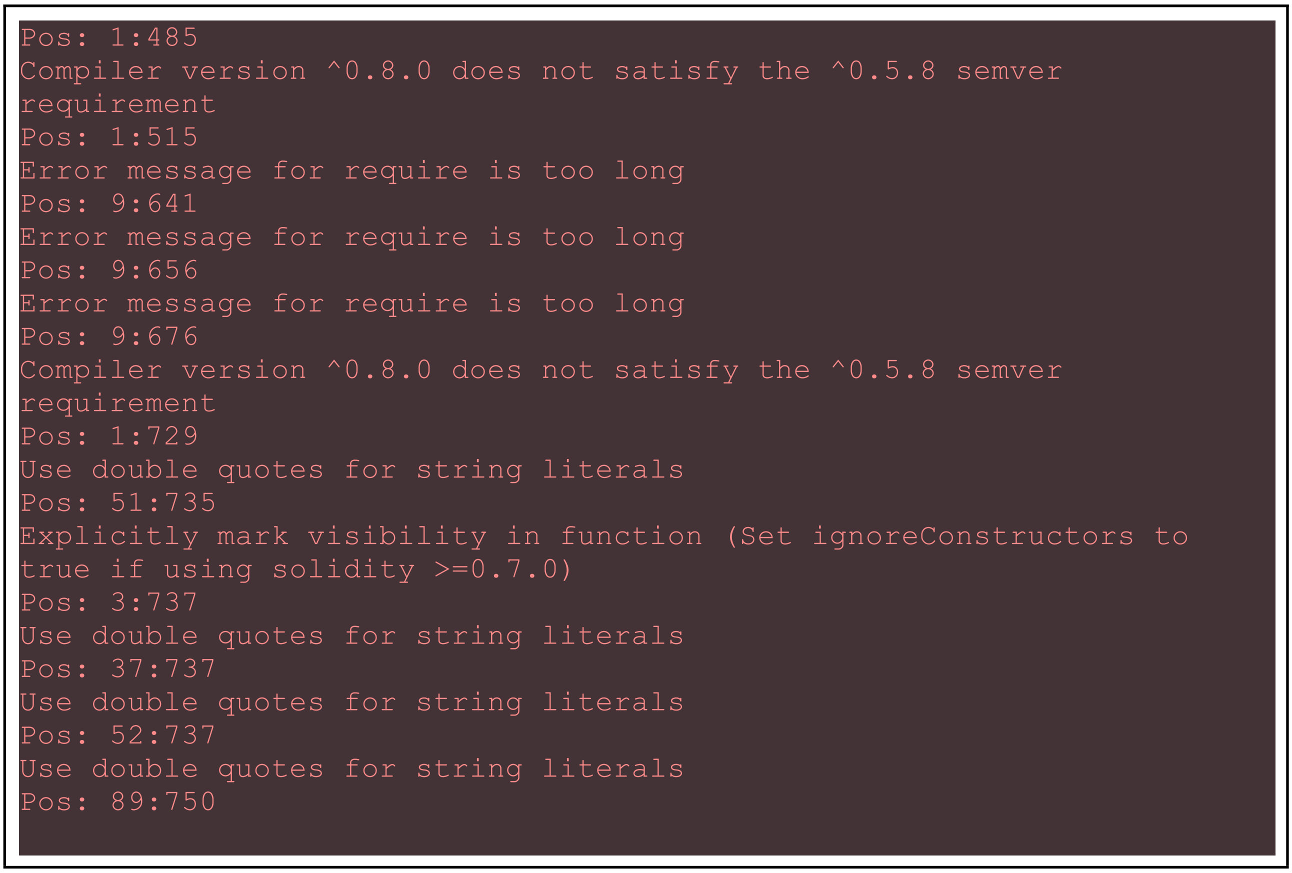
Solhint Linter

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.

IMXToken.sol



```
Compiler version ^0.8.0 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:8
Compiler version ^0.8.0 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:88
Compiler version ^0.8.0 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:115
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:162
Error message for require is too long
Pos: 9:262
Error message for require is too long
Pos: 9:301
Error message for require is too long
Pos: 9:322
Error message for require is too long
Pos: 9:323
Error message for require is too long
Pos: 9:328
Error message for require is too long
Pos: 9:366
Error message for require is too long
Pos: 9:371
Error message for require is too long
Pos: 9:392
Error message for require is too long
Pos: 9:393
Code contains empty blocks
Pos: 94:413
Compiler version ^0.8.0 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:420
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:433
Compiler version ^0.8.0 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:458
Compiler version ^0.8.0 does not satisfy the ^0.5.8 semver
requirement
```



```
Pos: 1:485
Compiler version ^0.8.0 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:515
Error message for require is too long
Pos: 9:641
Error message for require is too long
Pos: 9:656
Error message for require is too long
Pos: 9:676
Compiler version ^0.8.0 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:729
Use double quotes for string literals
Pos: 51:735
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 3:737
Use double quotes for string literals
Pos: 37:737
Use double quotes for string literals
Pos: 52:737
Use double quotes for string literals
Pos: 89:750
```

Software analysis result:

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