Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project:	stETH Token
Website:	https://lido.fi/
Platform:	Ethereum
Language:	Solidity
Date:	May 10th, 2024

Table of contents

`

Introduction

As part of EtherAuthority's community smart contract audit initiatives, the smart contract of stETH Token from lido.fi was 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 10th, 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 seems to be a part of the Aragon framework, specifically handling proxy contracts for upgradeable apps. Here's a breakdown of its functionality:
    - **Unstructured Storage Library:** This library provides functions to interact with unstructured storage. It allows storing and retrieving data at arbitrary storage slots.
    - **Interface Definitions:**
        - **IACL:** An interface defining the Access Control List for permissions management.
        - **IVaultRecoverable:** An interface for contracts that support the recovery of tokens to a vault.
    - **AppStorage Contract:** This contract defines storage slots for storing the kernel address and app ID.
    - **IsContract Contract:** This contract provides a function isContract to check if an address corresponds to a contract.
    - **ERCProxy Contract:** This contract defines an interface for proxy contracts, specifying functions to get the proxy type and implementation address.
    - **DelegateProxy Contract:** This contract extends ERCProxy and provides a function delegatedFwd to perform delegate calls.
    - **DepositableStorage Contract:** This contract defines a storage slot to indicate whether a contract accepts deposits.

    - **DepositableDelegateProxy Contract:** This contract combines the functionality of DepositableStorage and DelegateProxy, allowing for delegate calls and handling of deposits.
    - **KernelConstants Contract:** This contract defines constants related to the Aragon kernel, such as the app IDs for the core kernel, default ACL, and default vault.
    - **KernelNamespaceConstants Contract:** This contract defines constants related to kernel namespaces.
    - **AppProxyBase Contract:** This contract is the base contract for Aragon app proxies. It sets up the proxy with a kernel reference, app ID, and optional initialization payload.
    - **AppProxyUpgradeable Contract:** This contract extends AppProxyBase and implements the ERC897 standard for upgradeable proxies. It defines functions to get the implementation address and proxy type.
- Overall, these contracts provide a framework for creating upgradeable proxies for Aragon apps, allowing for efficient storage management and upgradeability.
- stETH is a transferable rebasing utility token representing a share of the total ETH staked through the LIDO protocol, which consists of user deposits and staking rewards. Because stETH rebases daily, it communicates the position of the share daily.

Audit scope

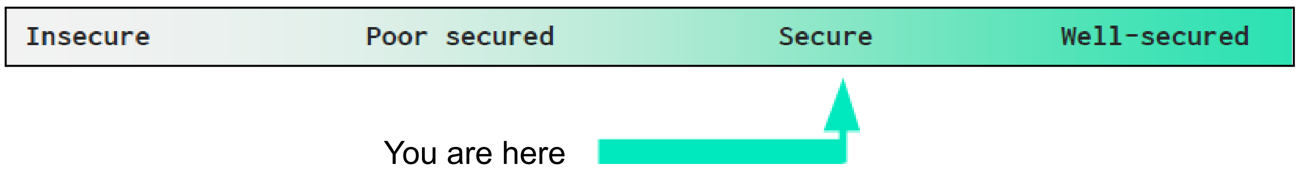
Name	**Code Review and Security Analysis Report for stETH Token Smart Contract**
Platform	**Ethereum**
File	AppProxyUpgradeable.sol
Smart Contract Code	0xae7ab96520de3a18e5e111b5eaab095312d7fe84
Audit Date	May 10th, 2024

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Tokenomics: • Name: Liquid staked Ether 2.0 st • Symbol: stETH • Decimals: 18	**YES, This is valid.**
Ownership Control: • There are no owner functions, which makes it 100% decentralized.	**YES, This is valid.**

Audit Summary

According to the standard audit assessment, Customer`s solidity based smart contract is **"Secured"**.This token contract does not have any ownership control, hence it is 100% 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 4 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	Moderated
	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 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?	Not Detected
● Max Tax?	No
● Is it Anti-whale?	Not Detected
● Is Anti-bot?	Not Detected
● Is iter5re a Blacklist?	Not Detected
● Blacklist Check	No
● Can Mint?	No
● Is it Proxy?	Yes
● Can Take Ownership?	Not Detected
● 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 stETH 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 stETH 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 not well commented on in the smart contract. Ethereum's NatSpec commenting style is recommended.

Documentation

We were given a stETH Token smart contract code in the form of an Etherscan web link.

As mentioned above, code parts are not well commented on. but 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 that 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

AppProxyUpgradeable.sol : Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Incorrect placement of the visibility modifier, Empty Blocks in Code	Refer Audit Findings
2	implementation	read	Passed	No Issue
3	proxyType	write	Passed	No Issue
4	getAppBase	internal	Passed	No Issue
5	isDepositable	read	Passed	No Issue
6	setDepositable	internal	Passed	No Issue
7	delegatedFwd	internal	Passed	No Issue
8	proxyType	write	Passed	No Issue
9	implementation	read	Passed	No Issue
10	isContract	internal	Passed	No Issue
11	kernel	read	Passed	No Issue
12	appId	read	Passed	No Issue
13	setKernel	internal	Passed	No Issue
14	setAppId	internal	Passed	No Issue
15	acl	read	Passed	No Issue
16	hasPermission	read	Passed	No Issue
17	setApp	write	Passed	No Issue
18	getApp	read	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 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) Functions in interfaces should be declared external:

```
interface IACL {
    function initialize(address permissionsCreator) external;

    // TODO: this should be external
    // See https://github.com/ethereum/solidity/issues/4832
    function hasPermission(address who, address where, bytes32 what,
bytes how) public view returns (bool);
}
```

When working with interfaces in code, it's essential to declare functions as external within the interface definition. This warning occurs because one or more functions in your interface are not declared as external.

Resolution: Ensure that each function declaration within the interface is preceded by the "external" keyword.

(2) Use the latest solidity version:

```
pragma solidity ^0.4.24;
```

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

Resolution: Please use versions greater than 0.8.7.

(3) Incorrect placement of the visibility modifier:

```
constructor(IKernel _kernel, bytes32 _appId, bytes
_initializePayload)
        AppProxyBase(_kernel, _appId, _initializePayload)
        public // solium-disable-line visibility-first
    {
        // solium-disable-previous-line no-empty-blocks
    }
```

During the code review, it was observed that the constructor in the smart contract contains a visibility modifier (public) that is placed after other modifiers and the constructor's base constructor call. According to Solidity's syntax conventions, visibility modifiers should be the first modifiers in the list.

Resolution: We recommend adhering to Solidity's syntax guidelines by moving the visibility modifier to the beginning of the list of modifiers.

(4) Empty Blocks in Code:

```
constructor(IKernel _kernel, bytes32 _appId, bytes
_initializePayload)
        AppProxyBase(_kernel, _appId, _initializePayload)
        public // solium-disable-line visibility-first
    {
        // solium-disable-previous-line no-empty-blocks
    }
```

During the code review, it was identified that the constructor in the smart contract contains an empty block. Empty blocks serve no functional purpose and can lead to confusion for developers reviewing the code. Code readability and maintainability are essential for smart contracts, and empty blocks should be avoided.

Resolution: We recommend removing the empty block to improve code clarity and maintainability.

Centralization Risk

The stETH Token smart contract does not have any ownership control, **hence it is 100% decentralized.**

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

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 4 informational issues in the smart contract. 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 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.

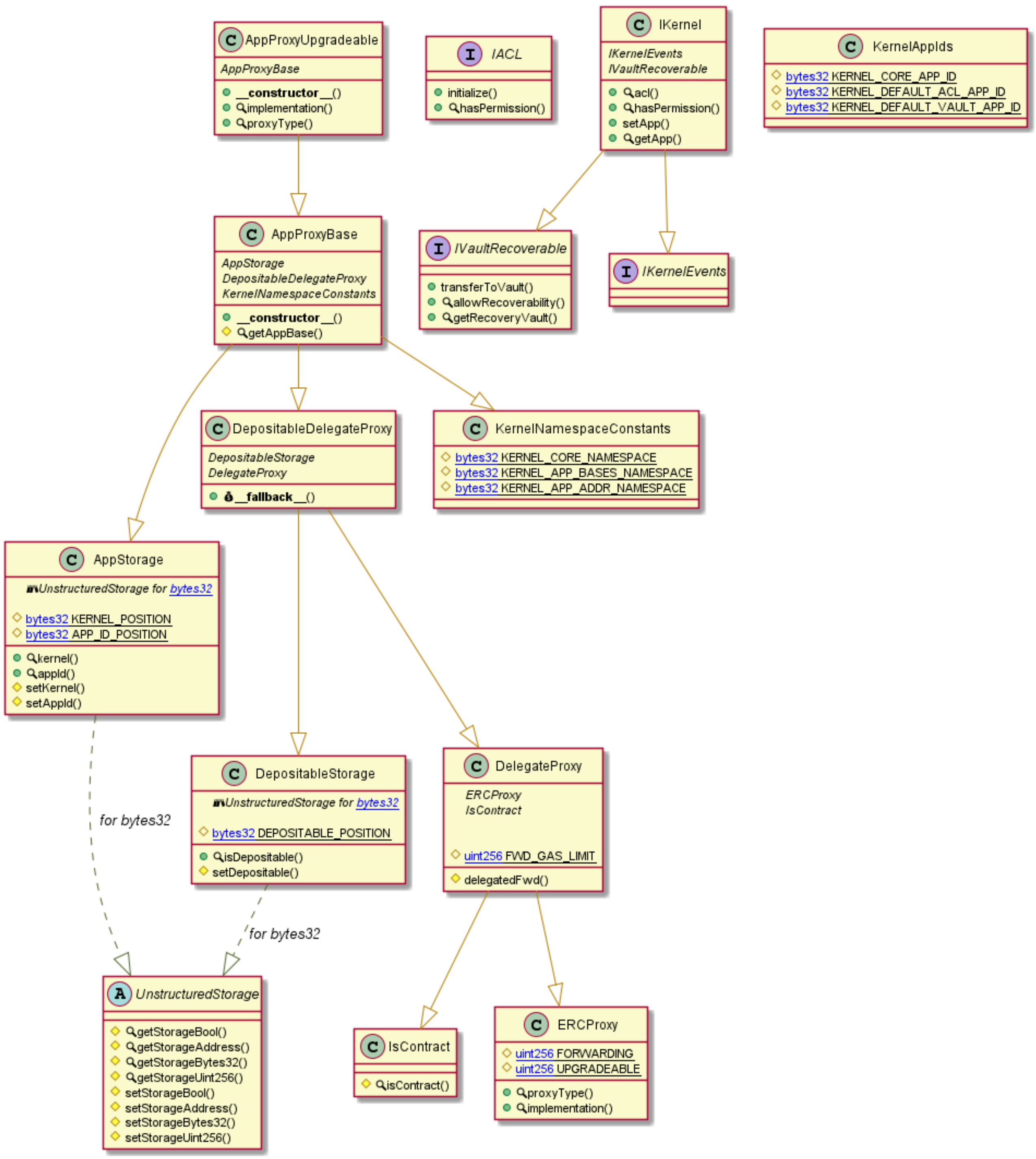
Technical Disclaimer

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 - stETH Token

AppProxyUpgradeable Diagram

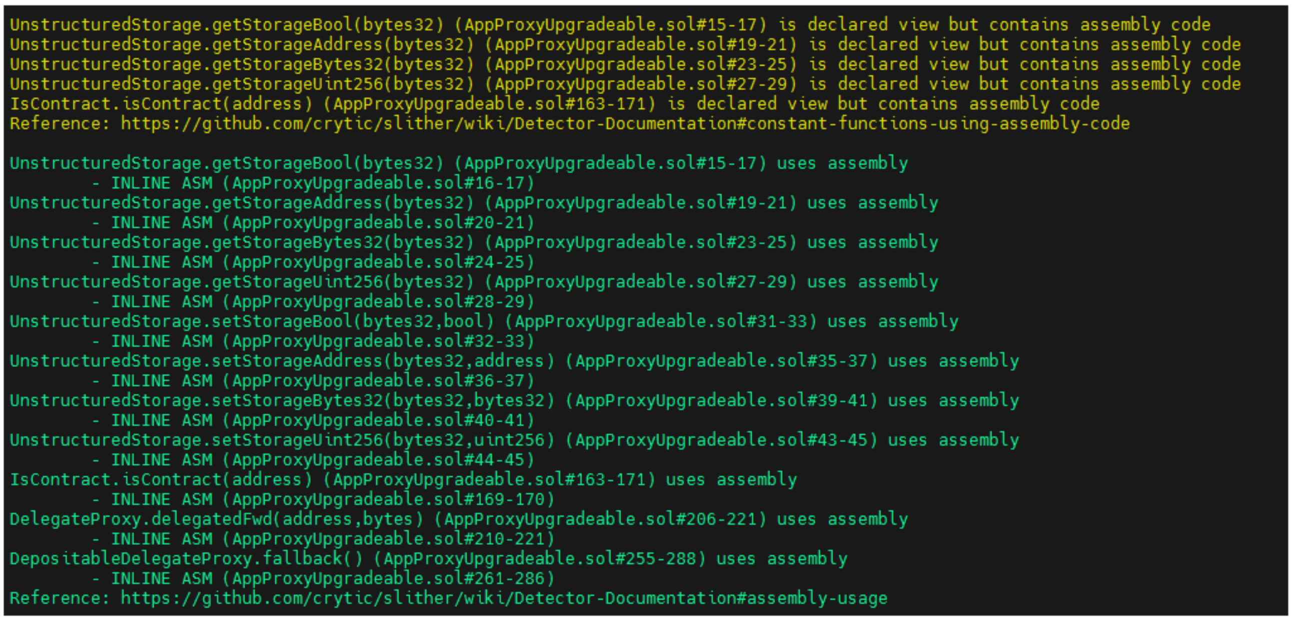


Slither Results Log

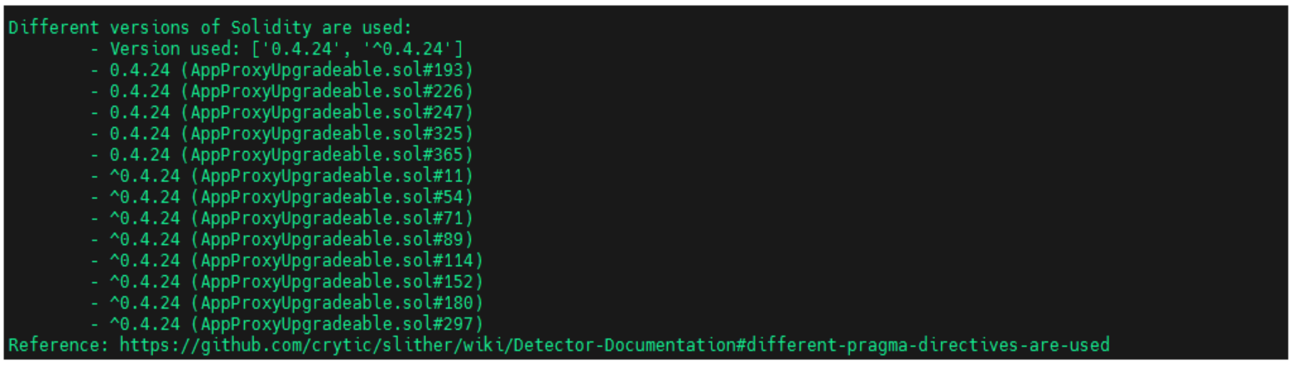
Slither is a Solidity static analysis framework that uses vulnerability detectors, displays contract details, and provides an API for writing custom analyses. It helps developers identify vulnerabilities, improve code comprehension, and prototype custom analyses quickly. The analysis includes a report with warnings and errors, allowing developers to quickly prototype and fix issues.

We did the analysis of the project altogether. Below are the results.

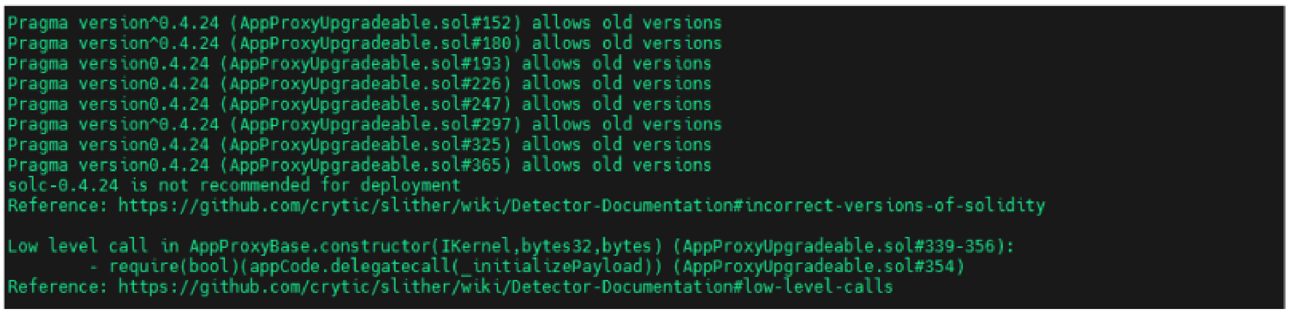
Slither Log >> AppProxyUpgradeable.sol



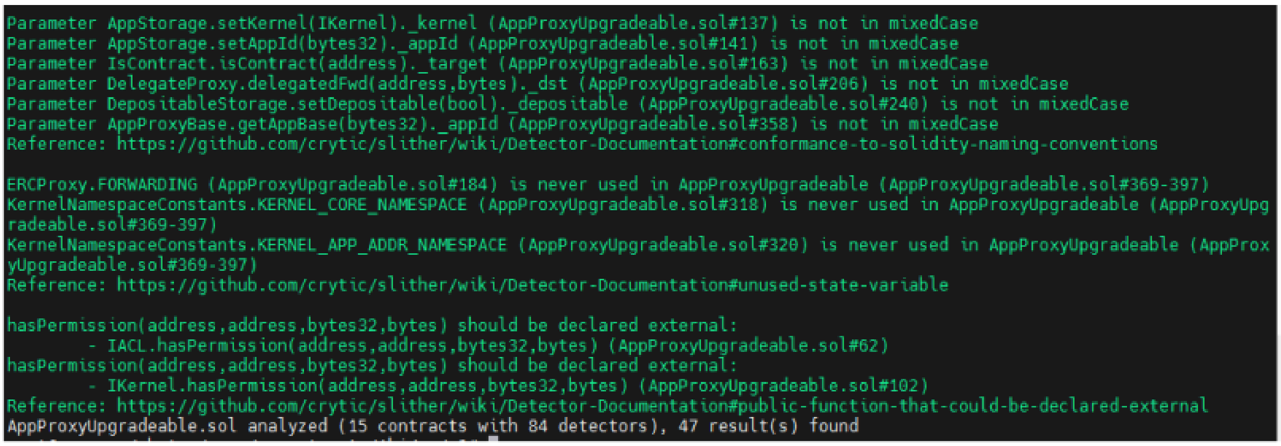
```
UnstructuredStorage.getStorageBool(bytes32) (AppProxyUpgradeable.sol#15-17) is declared view but contains assembly code
UnstructuredStorage.getStorageAddress(bytes32) (AppProxyUpgradeable.sol#19-21) is declared view but contains assembly code
UnstructuredStorage.getStorageBytes32(bytes32) (AppProxyUpgradeable.sol#23-25) is declared view but contains assembly code
UnstructuredStorage.getStorageUint256(bytes32) (AppProxyUpgradeable.sol#27-29) is declared view but contains assembly code
IsContract.isContract(address) (AppProxyUpgradeable.sol#163-171) is declared view but contains assembly code
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#constant-functions-using-assembly-code

UnstructuredStorage.getStorageBool(bytes32) (AppProxyUpgradeable.sol#15-17) uses assembly
        - INLINE ASM (AppProxyUpgradeable.sol#16-17)
UnstructuredStorage.getStorageAddress(bytes32) (AppProxyUpgradeable.sol#19-21) uses assembly
        - INLINE ASM (AppProxyUpgradeable.sol#20-21)
UnstructuredStorage.getStorageBytes32(bytes32) (AppProxyUpgradeable.sol#23-25) uses assembly
        - INLINE ASM (AppProxyUpgradeable.sol#24-25)
UnstructuredStorage.getStorageUint256(bytes32) (AppProxyUpgradeable.sol#27-29) uses assembly
        - INLINE ASM (AppProxyUpgradeable.sol#28-29)
UnstructuredStorage.setStorageBool(bytes32,bool) (AppProxyUpgradeable.sol#31-33) uses assembly
        - INLINE ASM (AppProxyUpgradeable.sol#32-33)
UnstructuredStorage.setStorageAddress(bytes32,address) (AppProxyUpgradeable.sol#35-37) uses assembly
        - INLINE ASM (AppProxyUpgradeable.sol#36-37)
UnstructuredStorage.setStorageBytes32(bytes32,bytes32) (AppProxyUpgradeable.sol#39-41) uses assembly
        - INLINE ASM (AppProxyUpgradeable.sol#40-41)
UnstructuredStorage.setStorageUint256(bytes32,uint256) (AppProxyUpgradeable.sol#43-45) uses assembly
        - INLINE ASM (AppProxyUpgradeable.sol#44-45)
IsContract.isContract(address) (AppProxyUpgradeable.sol#163-171) uses assembly
        - INLINE ASM (AppProxyUpgradeable.sol#169-170)
DelegateProxy.delegatedFwd(address,bytes) (AppProxyUpgradeable.sol#206-221) uses assembly
        - INLINE ASM (AppProxyUpgradeable.sol#210-221)
DepositableDelegateProxy.fallback() (AppProxyUpgradeable.sol#255-288) uses assembly
        - INLINE ASM (AppProxyUpgradeable.sol#261-286)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
```



```
Different versions of Solidity are used:
        - Version used: ['0.4.24', '^0.4.24']
        - 0.4.24 (AppProxyUpgradeable.sol#193)
        - 0.4.24 (AppProxyUpgradeable.sol#226)
        - 0.4.24 (AppProxyUpgradeable.sol#247)
        - 0.4.24 (AppProxyUpgradeable.sol#325)
        - 0.4.24 (AppProxyUpgradeable.sol#365)
        - ^0.4.24 (AppProxyUpgradeable.sol#11)
        - ^0.4.24 (AppProxyUpgradeable.sol#54)
        - ^0.4.24 (AppProxyUpgradeable.sol#71)
        - ^0.4.24 (AppProxyUpgradeable.sol#89)
        - ^0.4.24 (AppProxyUpgradeable.sol#114)
        - ^0.4.24 (AppProxyUpgradeable.sol#152)
        - ^0.4.24 (AppProxyUpgradeable.sol#180)
        - ^0.4.24 (AppProxyUpgradeable.sol#297)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used
```



```
Pragma version^0.4.24 (AppProxyUpgradeable.sol#152) allows old versions
Pragma version^0.4.24 (AppProxyUpgradeable.sol#180) allows old versions
Pragma version0.4.24 (AppProxyUpgradeable.sol#193) allows old versions
Pragma version0.4.24 (AppProxyUpgradeable.sol#226) allows old versions
Pragma version0.4.24 (AppProxyUpgradeable.sol#247) allows old versions
Pragma version^0.4.24 (AppProxyUpgradeable.sol#297) allows old versions
Pragma version0.4.24 (AppProxyUpgradeable.sol#325) allows old versions
Pragma version0.4.24 (AppProxyUpgradeable.sol#365) allows old versions
solc-0.4.24 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Low level call in AppProxyBase.constructor(IKernel,bytes32,bytes) (AppProxyUpgradeable.sol#339-356):
        - require(bool)(appCode.delegatecall(_initializePayload)) (AppProxyUpgradeable.sol#354)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
```

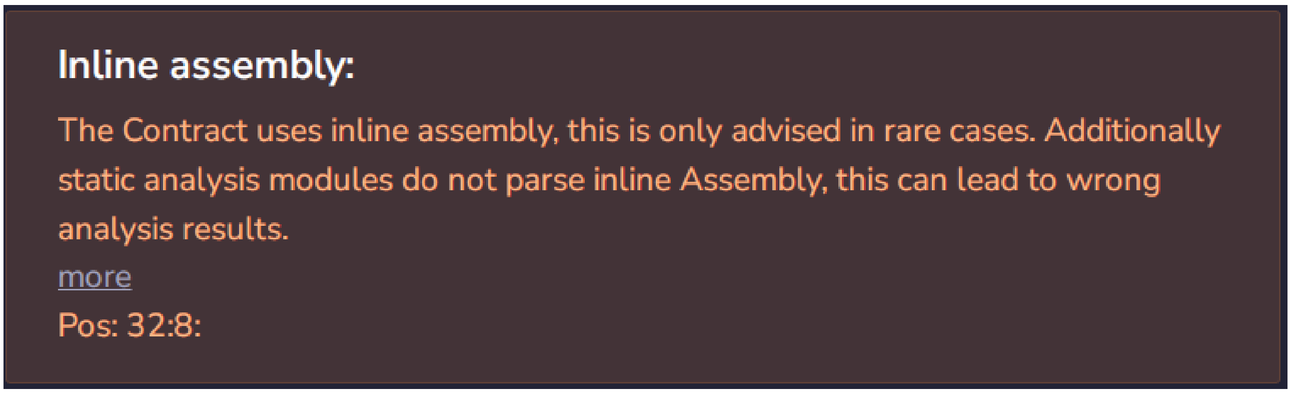


```
Parameter AppStorage.setKernel(IKernel)._kernel (AppProxyUpgradeable.sol#137) is not in mixedCase
Parameter AppStorage.setAppId(bytes32)._appId (AppProxyUpgradeable.sol#141) is not in mixedCase
Parameter IsContract.isContract(address)._target (AppProxyUpgradeable.sol#163) is not in mixedCase
Parameter DelegateProxy.delegatedFwd(address,bytes)._dst (AppProxyUpgradeable.sol#206) is not in mixedCase
Parameter DepositableStorage.setDepositable(bool)._depositable (AppProxyUpgradeable.sol#240) is not in mixedCase
Parameter AppProxyBase.getAppBase(bytes32)._appId (AppProxyUpgradeable.sol#358) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

ERCProxy.FORWARDING (AppProxyUpgradeable.sol#184) is never used in AppProxyUpgradeable (AppProxyUpgradeable.sol#369-397)
KernelNamespaceConstants.KERNEL_CORE_NAMESPACE (AppProxyUpgradeable.sol#318) is never used in AppProxyUpgradeable (AppProxyUpg
radeable.sol#369-397)
KernelNamespaceConstants.KERNEL_APP_ADDR_NAMESPACE (AppProxyUpgradeable.sol#320) is never used in AppProxyUpgradeable (AppProx
yUpgradeable.sol#369-397)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variable

hasPermission(address,address,bytes32,bytes) should be declared external:
        - IACL.hasPermission(address,address,bytes32,bytes) (AppProxyUpgradeable.sol#62)
hasPermission(address,address,bytes32,bytes) should be declared external:
        - IKernel.hasPermission(address,address,bytes32,bytes) (AppProxyUpgradeable.sol#102)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
AppProxyUpgradeable.sol analyzed (15 contracts with 84 detectors), 47 result(s) found
```

Solidity Static Analysis

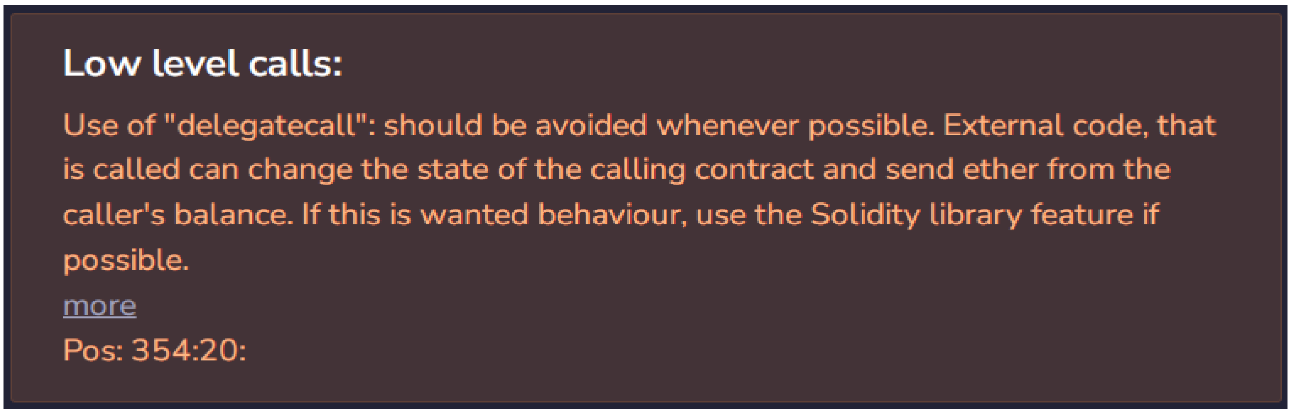
Static code analysis is used to identify many common coding problems before a program is released. It involves examining the code manually or using tools to automate the process. Static code analysis tools can automatically scan the code without executing it.

AppProxyUpgradeable.sol



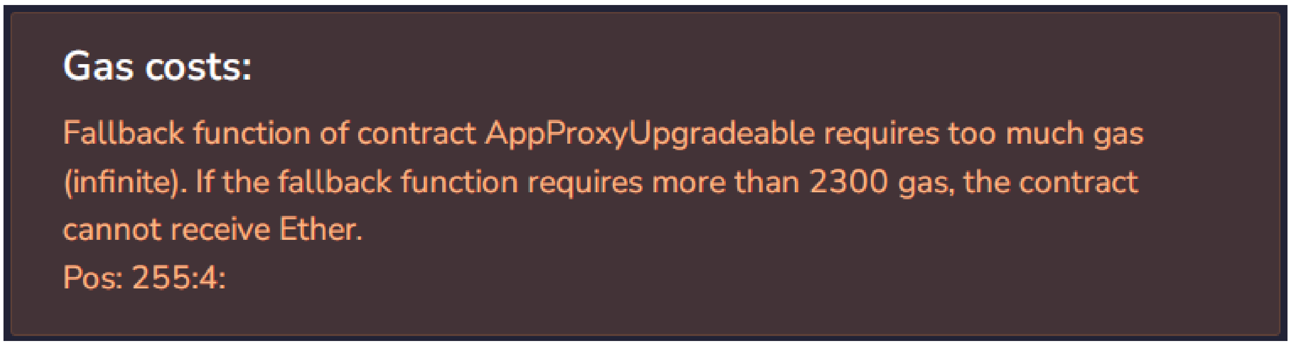
Inline assembly:

The Contract uses inline assembly, this is only advised in rare cases. Additionally static analysis modules do not parse inline Assembly, this can lead to wrong analysis results.
more
Pos: 32:8:



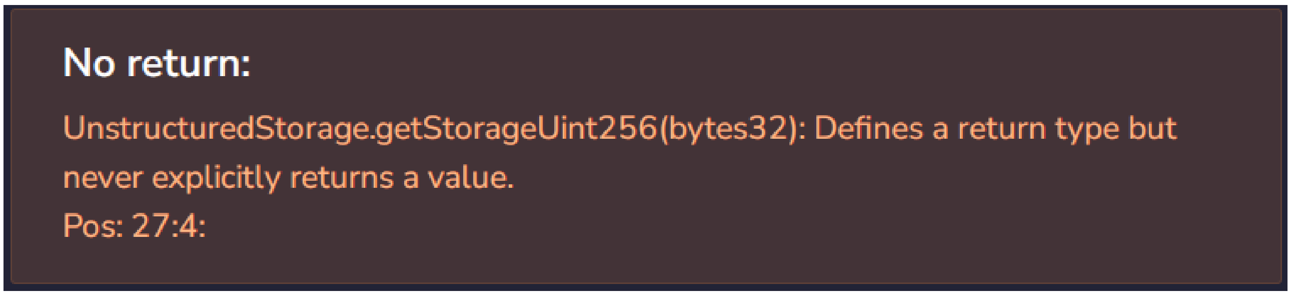
Low level calls:

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: 354:20:



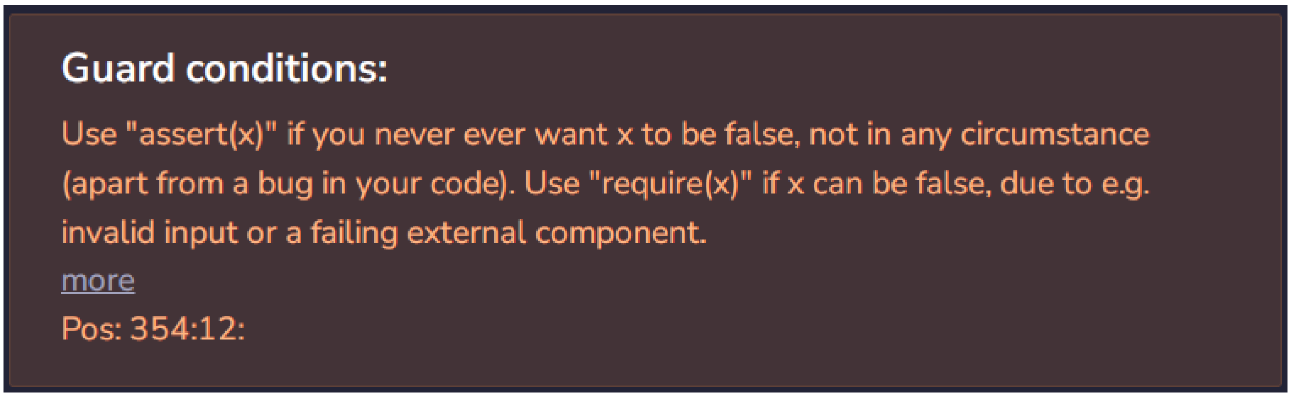
Gas costs:

Fallback function of contract AppProxyUpgradeable requires too much gas (infinite). If the fallback function requires more than 2300 gas, the contract cannot receive Ether.
Pos: 255:4:



No return:

UnstructuredStorage.getStorageUint256(bytes32): Defines a return type but never explicitly returns a value.
Pos: 27:4:



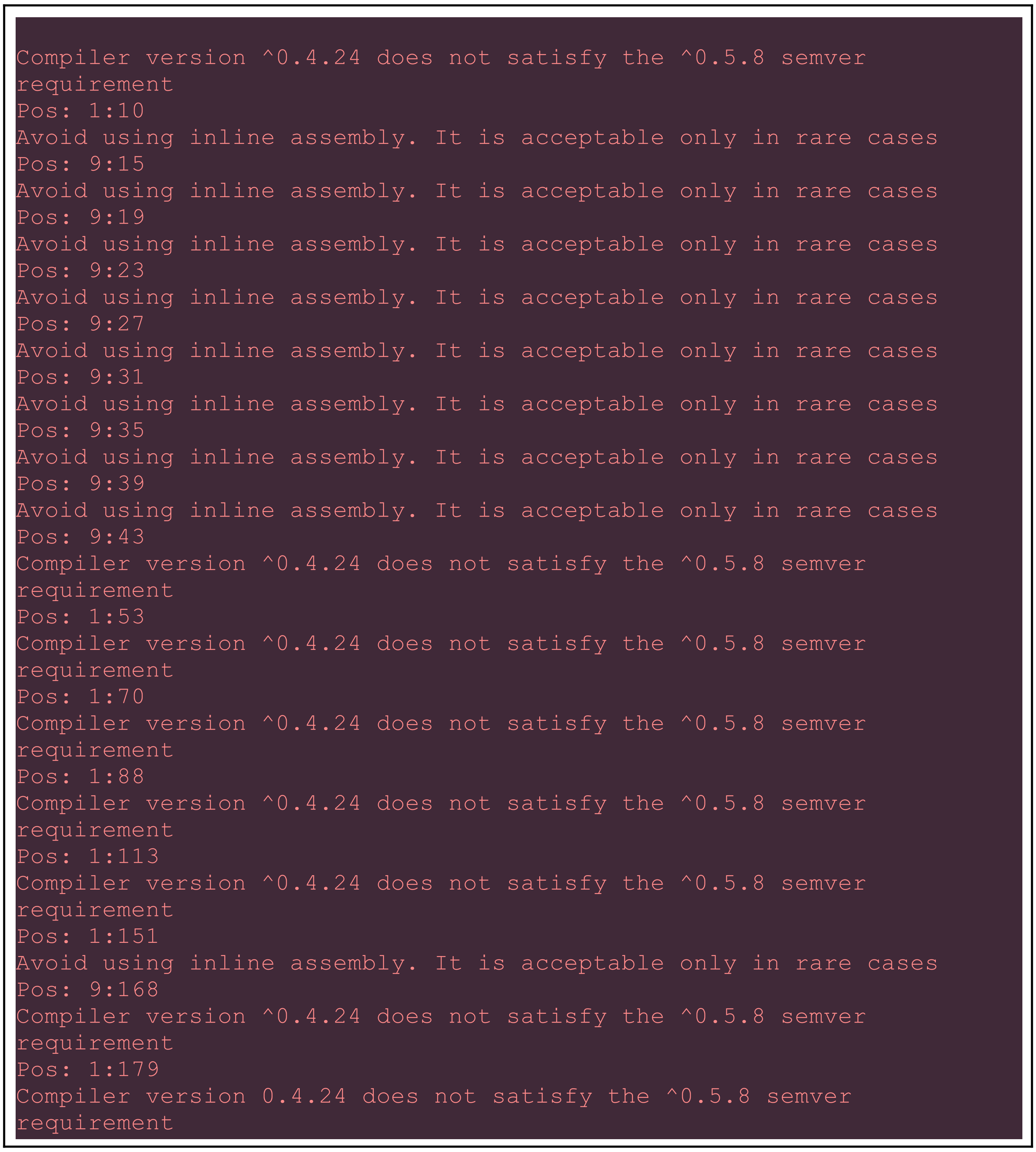
Guard conditions:

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: 354:12:

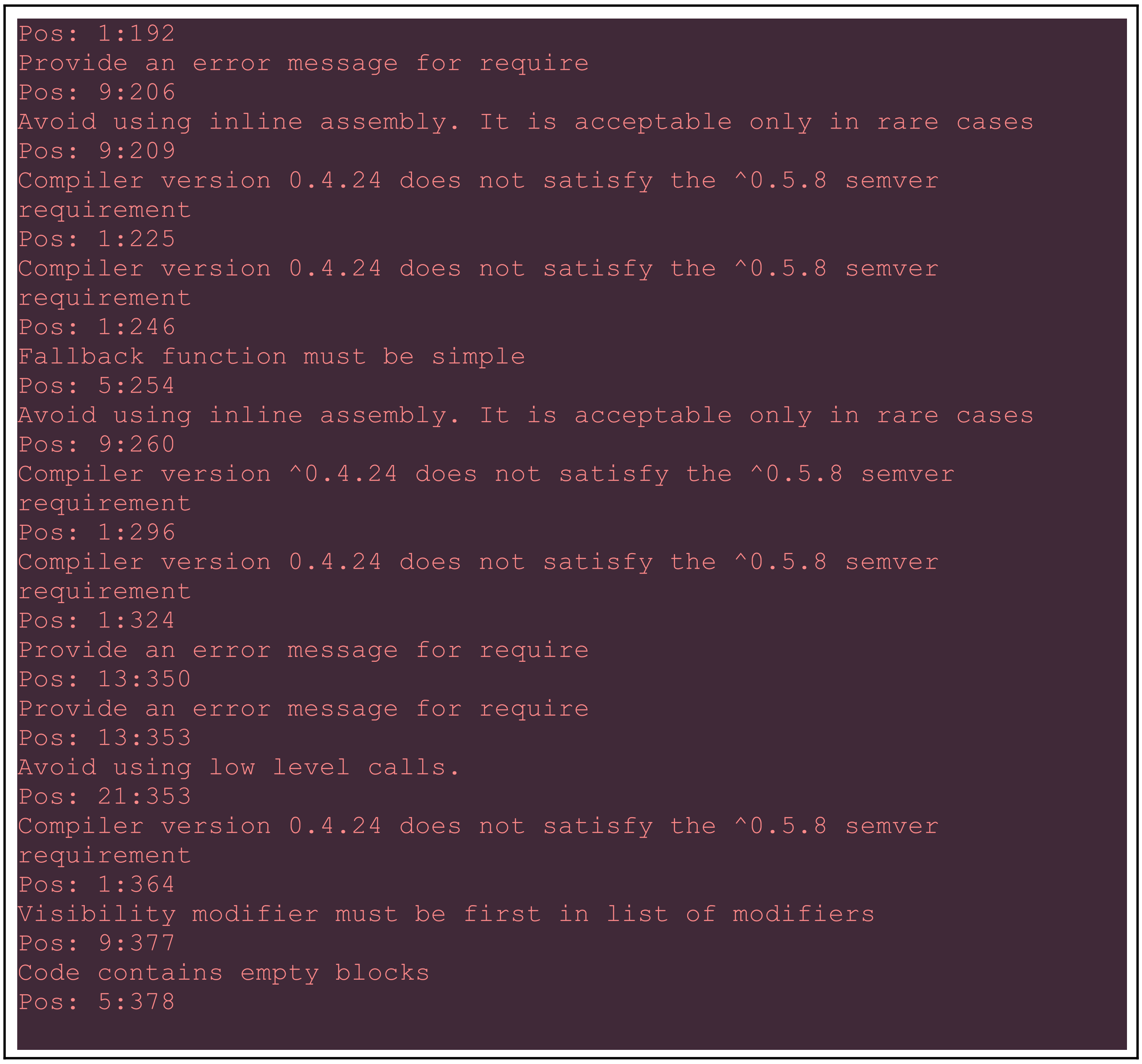
Solhint Linter

Solhint 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.

AppProxyUpgradeable.sol



```
Compiler version ^0.4.24 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:10
Avoid using inline assembly. It is acceptable only in rare cases
Pos: 9:15
Avoid using inline assembly. It is acceptable only in rare cases
Pos: 9:19
Avoid using inline assembly. It is acceptable only in rare cases
Pos: 9:23
Avoid using inline assembly. It is acceptable only in rare cases
Pos: 9:27
Avoid using inline assembly. It is acceptable only in rare cases
Pos: 9:31
Avoid using inline assembly. It is acceptable only in rare cases
Pos: 9:35
Avoid using inline assembly. It is acceptable only in rare cases
Pos: 9:39
Avoid using inline assembly. It is acceptable only in rare cases
Pos: 9:43
Compiler version ^0.4.24 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:53
Compiler version ^0.4.24 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:70
Compiler version ^0.4.24 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:88
Compiler version ^0.4.24 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:113
Compiler version ^0.4.24 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:151
Avoid using inline assembly. It is acceptable only in rare cases
Pos: 9:168
Compiler version ^0.4.24 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:179
Compiler version 0.4.24 does not satisfy the ^0.5.8 semver
requirement
```



```
Pos: 1:192
Provide an error message for require
Pos: 9:206
Avoid using inline assembly. It is acceptable only in rare cases
Pos: 9:209
Compiler version 0.4.24 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:225
Compiler version 0.4.24 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:246
Fallback function must be simple
Pos: 5:254
Avoid using inline assembly. It is acceptable only in rare cases
Pos: 9:260
Compiler version ^0.4.24 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:296
Compiler version 0.4.24 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:324
Provide an error message for require
Pos: 13:350
Provide an error message for require
Pos: 13:353
Avoid using low level calls.
Pos: 21:353
Compiler version 0.4.24 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:364
Visibility modifier must be first in list of modifiers
Pos: 9:377
Code contains empty blocks
Pos: 5:378
```

Software analysis result:

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