



SAULIDITY



COOREST

2022

SMART CONTRACT
SECURITY ANALYSIS

PREPARED BY
Saulidity

PRESENTED TO
Coorest.io



2022

SAULIDITY

SECURITY REPORT



Smart Contract
Audit



saulidity.com



Saulidity



@Saulidity

DISCLAIMER

This report does not constitute financial advice, and Saulidity is not accountable or liable for any negative consequences resulting from this report, nor may Saulidity be held liable in any way. You agree to the terms of this disclaimer by reading any part of the report. If you do not agree to the terms, please stop reading this report immediately and delete and destroy any and all copies of this report that you have downloaded and/or printed. This report was entirely based on information given by the audited party and facts that existed prior to the audit. Saulidity and its auditors cannot be held liable for any outcome, including modifications (if any) made to the contract(s) for the audit that was completed. No modifications have been made to the contract(s) by the Saulidity team, but if it does, it will be indicated explicitly. The audit does not include the project team, website, logic, or tokenomics, but if it does, it will be indicated explicitly. The security is evaluated only on the basis of smart contracts only. There were no security checks performed on any apps or activities. There hasn't been a review of any product codes. It is assumed by Saulidity that the information and materials given were not tampered with, censored, or misrepresented. Even if this report exists and Saulidity makes every effort to uncover any security flaws, you should not rely completely on it and should conduct your own independent research. Saulidity hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against Saulidity, for any amount or kind of loss or damage that may result to you or any other person or any kind of company, community, association and institution. Saulidity is the exclusive owner of this report, and it is published by Saulidity. Without Saulidity's express written authorization, use of this report for any reason other than a security interest in the individual contacts, or use of sections of this report, is forbidden.

Table of Contents

| | |
|-----------|--|
| 02 | Saulidity |
| 03 | Introduction |
| 04 | Scope |
| 05 | Audit & Project Information |
| 06 | Summary Table |
| 07 | Executive Summary |
| 08 | Inheritance |
| 13 | Call Graph |
| 17 | Analysis |
| 30 | Testing Standards |

Saulidity

Saulidity is a renowned cybersecurity firm specializing in the analysis and development of Smart contracts. Saulidity, as a full-service security organization, can help with a variety of audits and project development.

In a market where confidence and trust are key, a genuine project may simply increase its user base enormously with an official audit performed by Saulidity.

Introduction

For a thorough understanding of the audit, please read the entire document.

The goal of the audit was to find any potential smart contract security problems and vulnerabilities.

The information in this report should be used to understand the smart contract's risk exposure and as a guide to improving the smart contract's security posture by addressing the concerns that were discovered.

During our audit, we conducted a thorough inquiry using automated analysis and manual review approaches.

The security specialists did a complete study independently of one another in order to uncover any security issues in the contracts as comprehensively as feasible. For optimum security and professionalism, all of our audits are undertaken by at least two independent auditors.

The audit was carried out on contracts that had not yet been deployed. The project's website, logic, or tokenomics have not been vetted by the Saulidity team.








Scope

We analyze smart contracts for both well-known and more specific vulnerabilities.

Here are some of the most well-known vulnerabilities that are taken into account but not limited to:

- Reentrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with (Unexpected) Throw
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Style guide violation
- Transfer forwards all gas
- ERC20 API violation
- Compiler version not fixed
- Unchecked external call - Unchecked math
- Unsafe type inference
- Implicit visibility level

Audit & Project Information

| | | |
|---|---------------------|--|
|  | Project Name | Coorest |
|  | Contract Name | Coorest.sol TrueCoorest.sol CC02.sol PoCC.sol |
|  | Report ID | SAUL91000 V1.1 |
|  | Website | Coorest.io |
|  | Contact | Nick Zwaneveld |
|  | Contact Information | nick@coorest.eu |
|  | Code language | Solidity |

Summary Table

| SEVERITY | FOUND |
|---|-------|
| Critical | 0 |
| High | 0 |
| Medium | 0 |
| Low | 0 |
| Lowest / Code Style / Optimized Practice | 13 |

Executive Summary

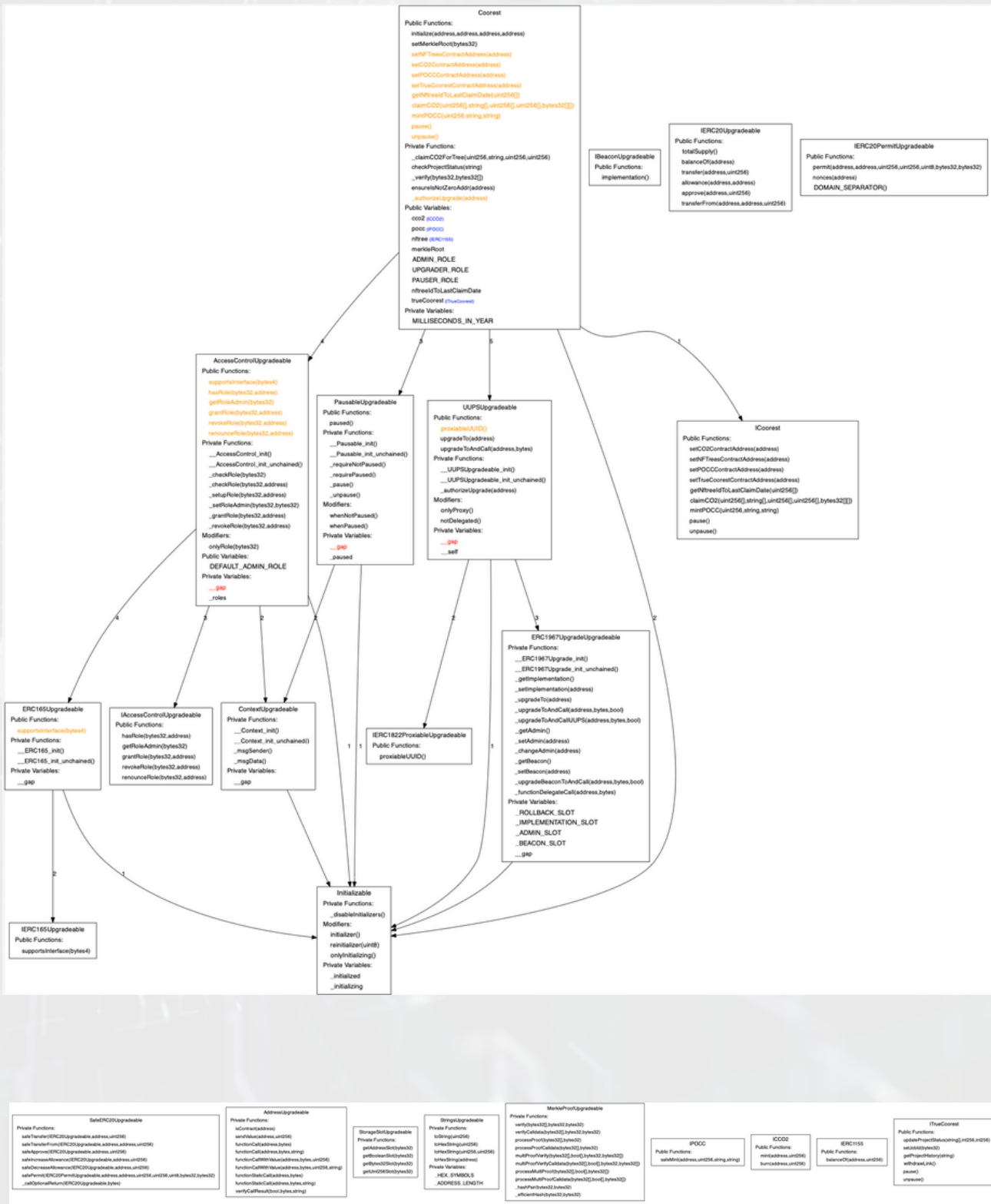
ACCORDING TO THE ANALYSIS, THERE ARE **0 CRITICAL, 0 HIGH, 0 MEDIUM AND 0 LOW SEVERITY SECURITY VULNERABILITIES.**

IT SHOULD BE NOTED THAT ALL FINDINGS WERE MITIGATED BY THE CLIENT.

ALL ISSUES FOUND DURING ANALYSIS WERE MANUALLY REVIEWED, AND FALSE POSITIVES WERE ELIMINATED. THE FINDINGS ARE PRESENTED IN THE ANALYSIS SECTION OF THE REPORT.

Inheritance

Coorest.sol



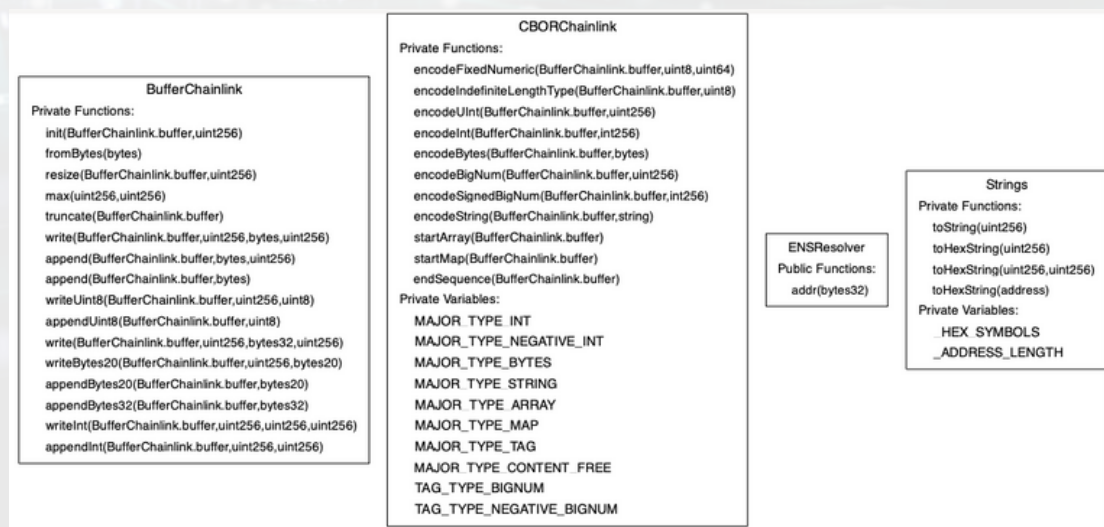
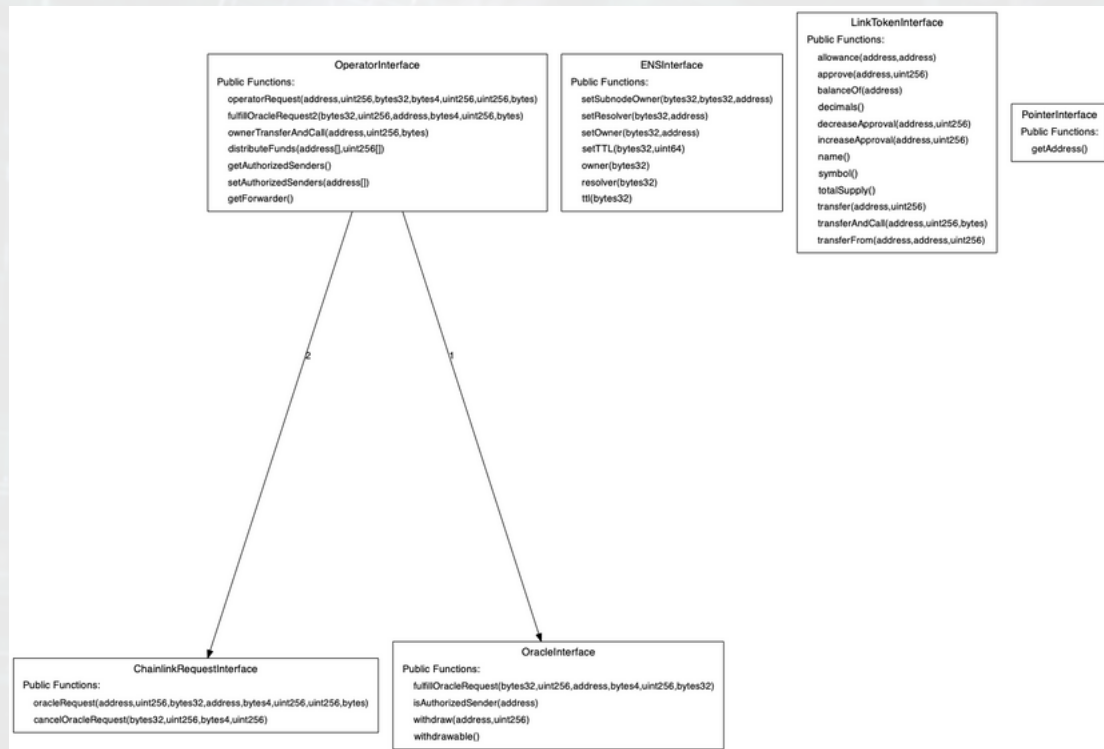
Inheritance

TrueCoorest.sol

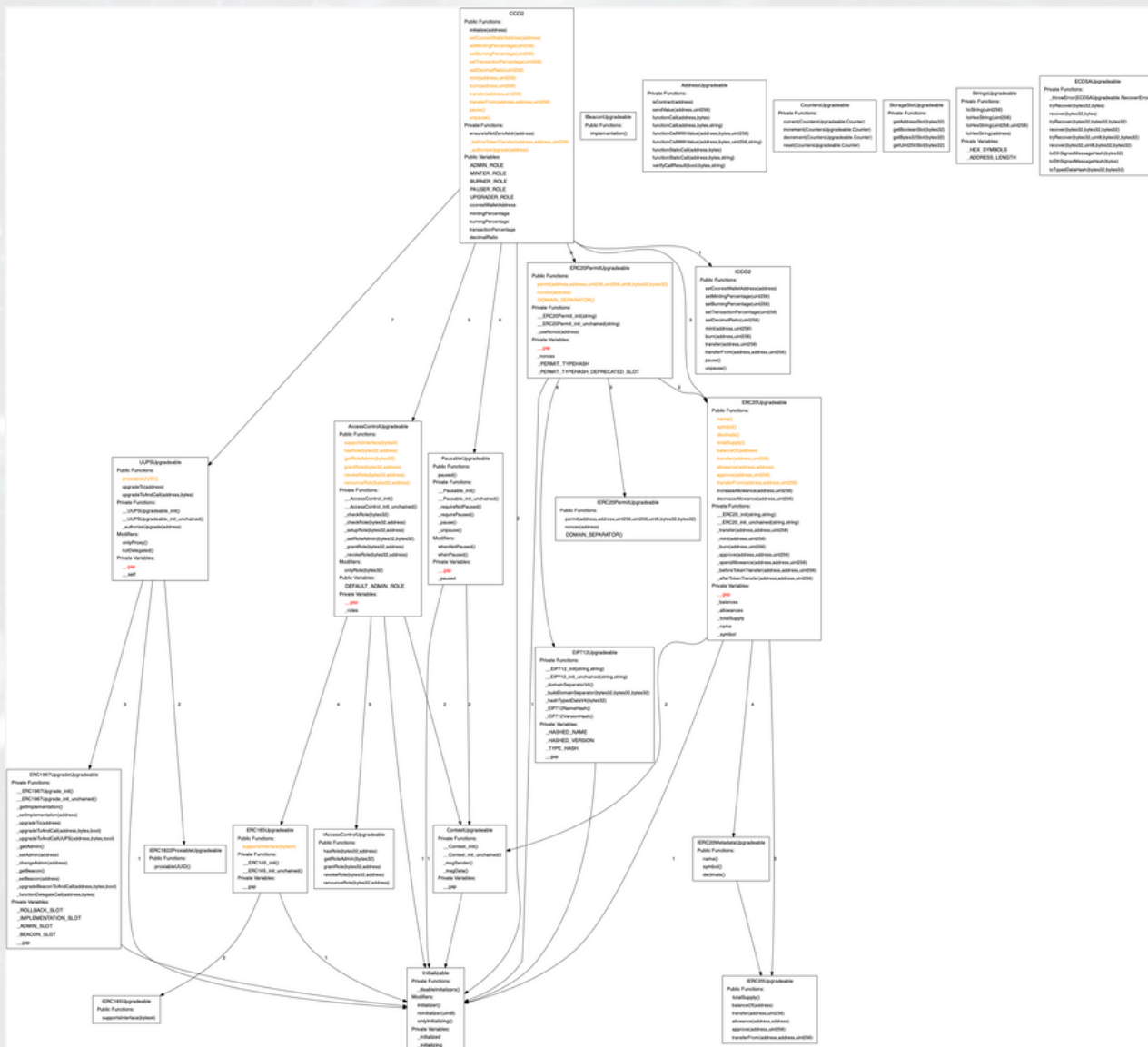


Inheritance

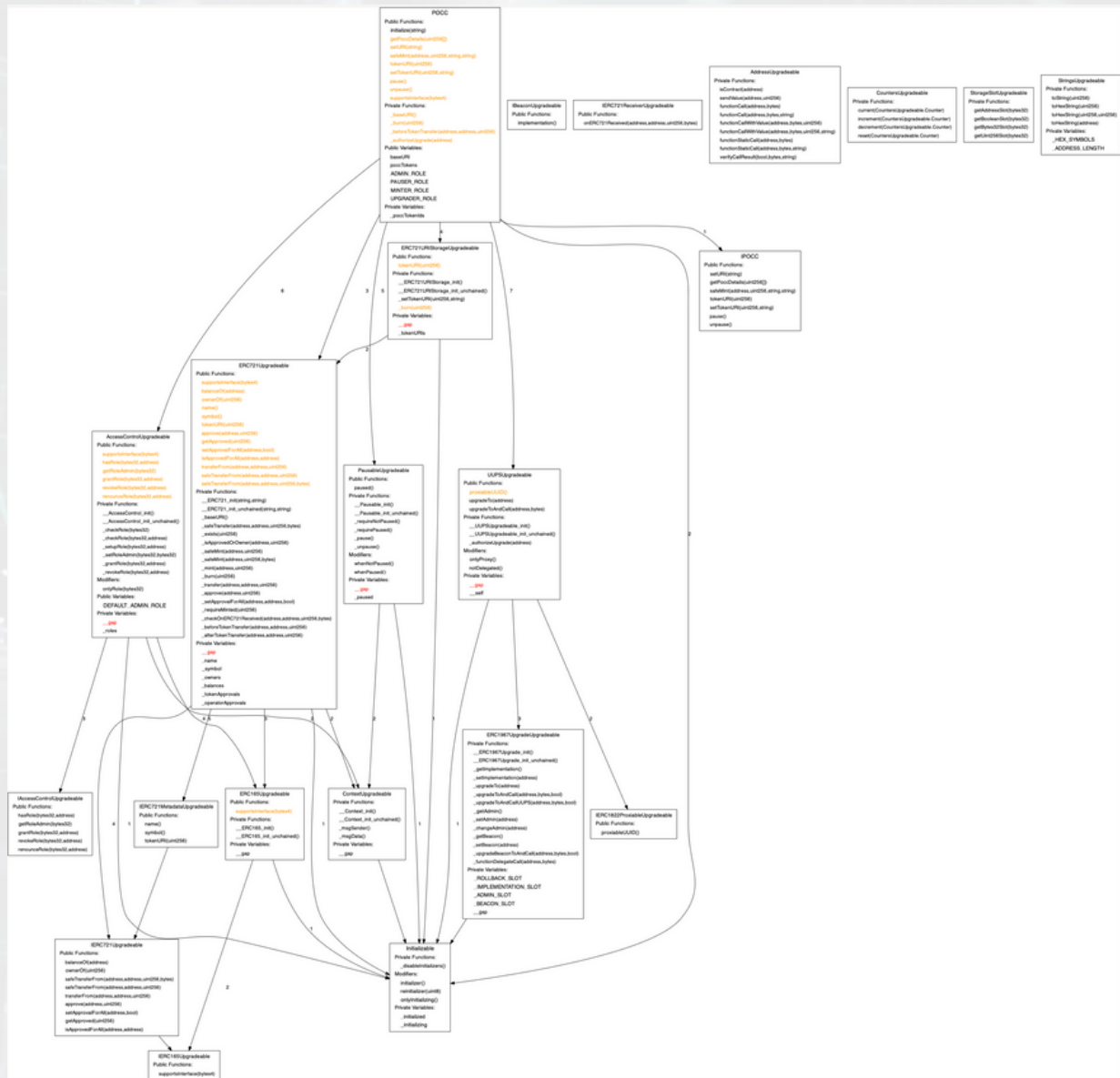
TrueCoorest.sol



CCO2.sol

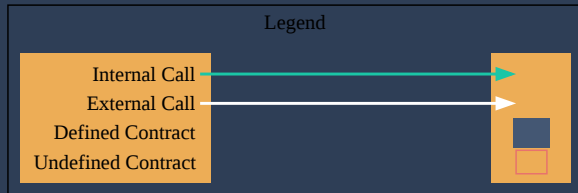


POCC.sol

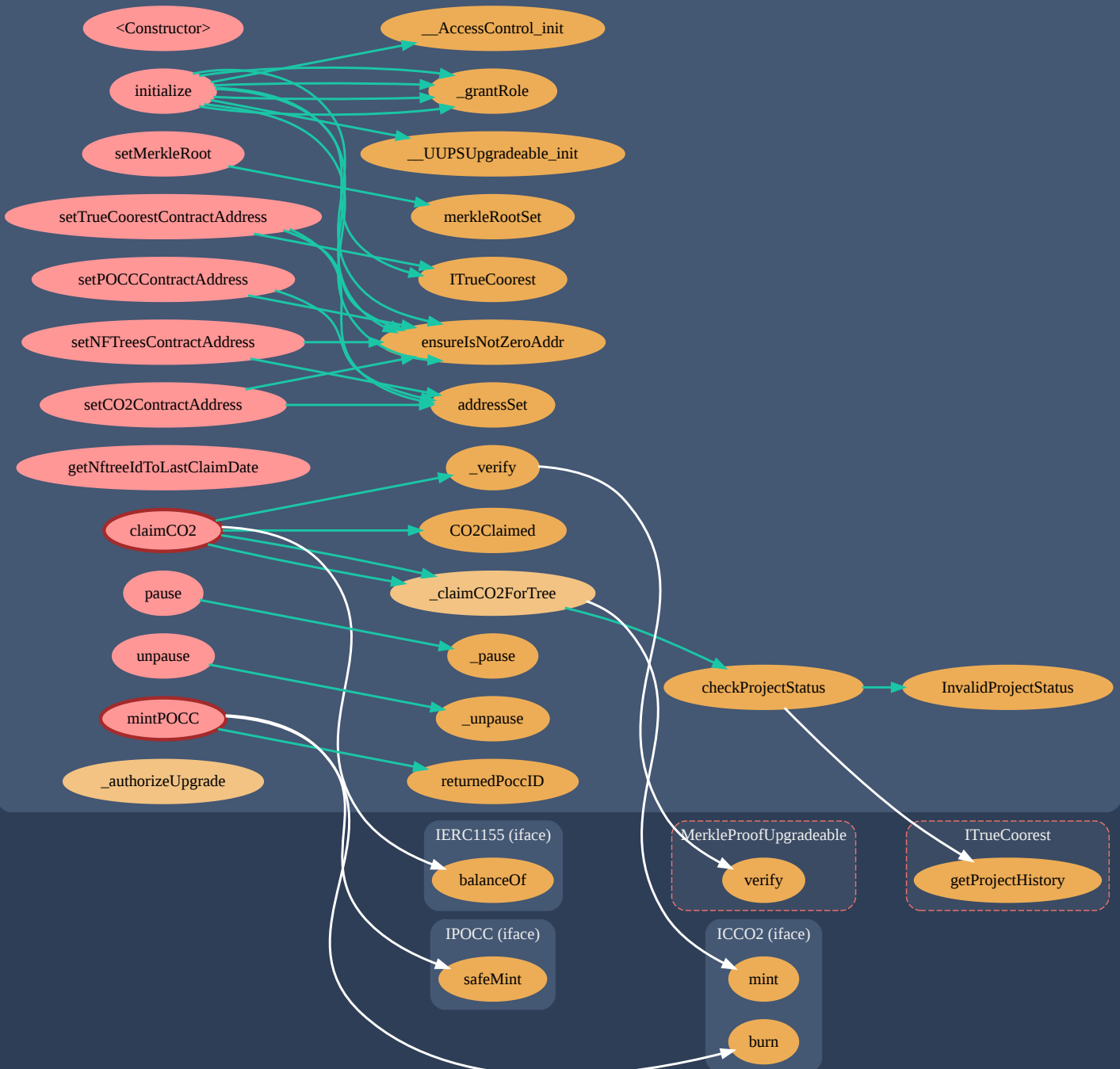


Call Graph

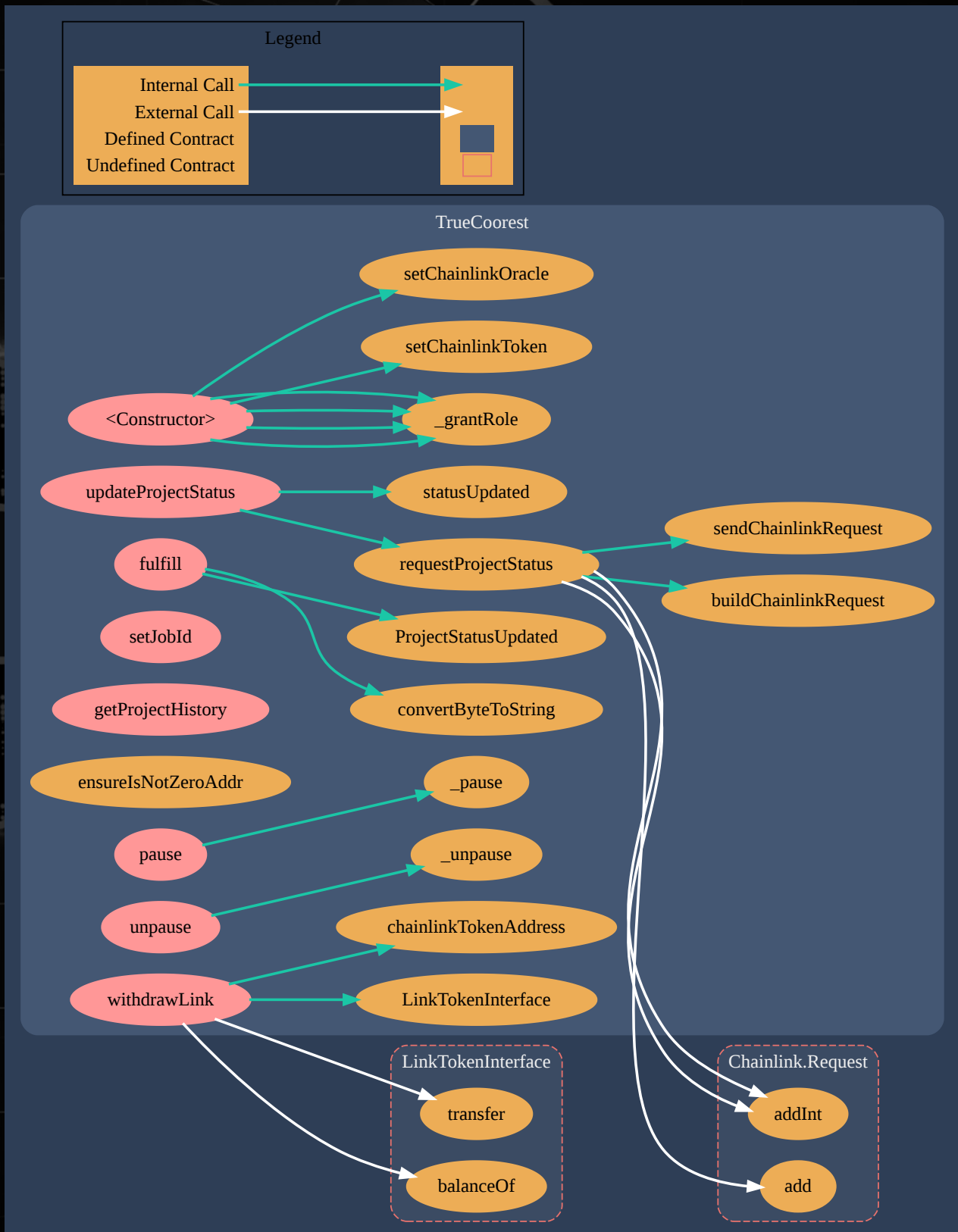
Coorest.sol



Coorest

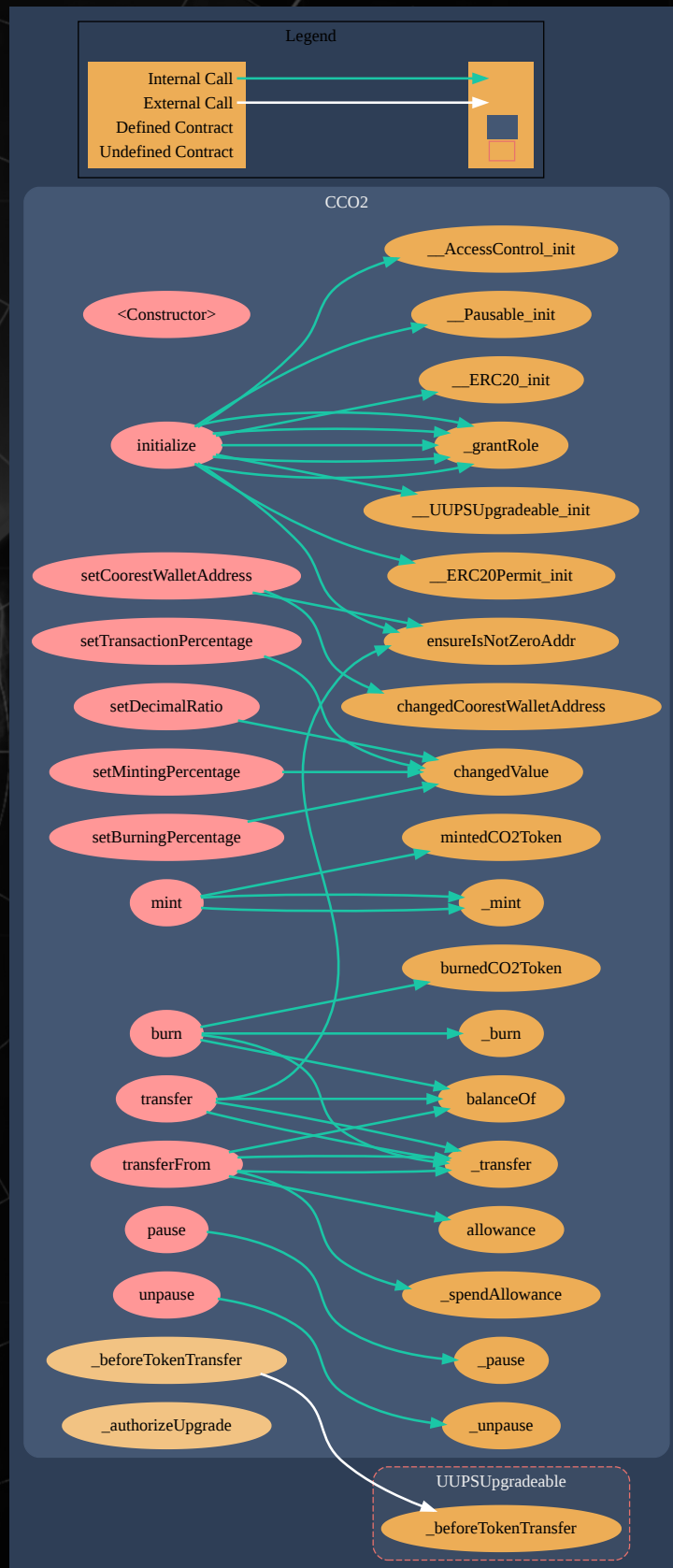


TrueCoorest.sol



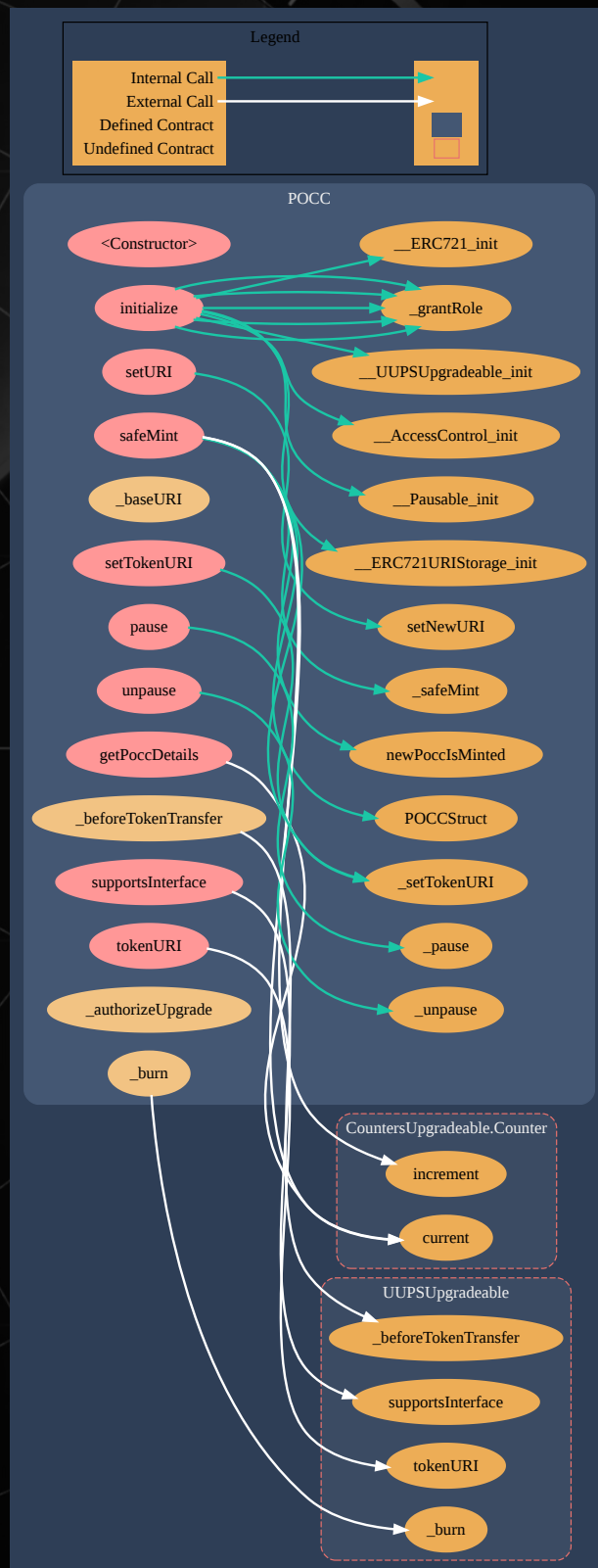
Call Graph

CCO2.sol



Call Graph

POCC.sol



Analysis

Coorest.sol

Issue: Usage of too many digits,

Severity: Lowest / Clean Coding / Optimized Practice

Location: L34

Description: Literals with many digits are difficult to read and review.

```
uint256 constant MILLISECONDS_IN_YEAR = 31536000000;
```

Recommendation: We recommend using the scientific notation with ether units suffix instead (ie: 50e6 ether).

- Ether suffix
- Time suffix
- The scientific notation

Status: Mitigated

Analysis

Coorest.sol

Issue: Possibility to declare public function as external.

Severity: Lowest / Code Style / Optimized Practice

Location: L51-76, L78-81, L83-92, L94-103, L107-115, L117-125, L129-141, L171-222, L225-235, L239-241, L243-245

Description: In order to save gas, public functions that are never called by the contract could be declared external.

Comment: We recommend to consider using the external attribute for functions that are never called from the contract.

Status: Mitigated

Analysis

Coorest.sol

Issue: Version of Solidity.

Severity: Lowest / Clean Coding / Optimized Practice

Location: General

Description: It's worth noting that some pragma versions are more frequently used by Solidity programmers and have been thoroughly tested in several security audits. There is a json file in the Solitidy Github repository that lists all defects found in various compiler versions.

Comment: If at all feasible, use the most recent stable pragma version that has been carefully tested to avoid any unknown vulnerabilities.

Consider deploying with any of the following Solidity versions:

- 0.5.16 - 0.5.17
- 0.6.11 - 0.6.12
- 0.7.5 - 0.7.6
- 0.8.4 - 0.8.7

Status: Mitigated

Analysis

Coorest.sol

Issue: Lock pragmas to a specific compiler version.

Severity: Lowest / Clean Coding / Optimized Practice

Location: General

Description: The current pragma Solidity directive is not locked to a fixed version.

```
pragma solidity >=0.4.22 <0.9.0;
```

Comment: It's best to specify a fixed compiler version to guarantee that the bytecode generated is consistent across builds. Locking the pragma helps ensure that contracts do not accidentally get deployed using, for example, that latest compiler, which may have a higher risker of undiscovered bugs. This is especially significant if the code is verified at the bytecode level.

Status: Mitigated

Analysis

TrueCoorest.sol

Issue: Possibility to declare public function as external.

Severity: Lowest / Code Style / Optimized Practice

Location: L48-53, L76-78, L82-84, L97-103, L107-109, L111-113

Description: In order to save gas, public functions that are never called by the contract could be declared external.

Comment: We recommend to consider using the external attribute for functions that are never called from the contract.

Status: Mitigated

Analysis

TrueCoorest.sol

Issue: Version of Solidity.

Severity: Lowest / Clean Coding / Optimized Practice

Location: General

Description: It's worth noting that some pragma versions are more frequently used by Solidity programmers and have been thoroughly tested in several security audits. There is a json file in the Solidity Github repository that lists all defects found in various compiler versions.

Comment: If at all feasible, use the most recent stable pragma version that has been carefully tested to avoid any unknown vulnerabilities.

Consider deploying with any of the following Solidity versions:

- 0.5.16 - 0.5.17
- 0.6.11 - 0.6.12
- 0.7.5 - 0.7.6
- 0.8.4 - 0.8.7

Status: Mitigated

Analysis

TrueCoorest.sol

Issue: Lock pragmas to a specific compiler version.

Severity: Lowest / Clean Coding / Optimized Practice

Location: General

Description: The current pragma Solidity directive is not locked to a fixed version.

```
pragma solidity >=0.4.22 <0.9.0;
```

Comment: It's best to specify a fixed compiler version to guarantee that the bytecode generated is consistent across builds. Locking the pragma helps ensure that contracts do not accidentally get deployed using, for example, that latest compiler, which may have a higher risker of undiscovered bugs. This is especially significant if the code is verified at the bytecode level.

Status: Mitigated

Analysis

CCO2.sol

Issue: Possibility to declare public function as external.

Severity: Lowest / Code Style / Optimized Practice

Location: L37-59, L63-71, L73-80, L82-89, L91-98, L100-107, L111-125, L127-142, L192-194, L196-198

Description: In order to save gas, public functions that are never called by the contract could be declared external.

Comment: We recommend to consider using the external attribute for functions that are never called from the contract.

Status: Mitigated

Analysis

CCO2.sol

Issue: Version of Solidity.

Severity: Lowest / Clean Coding / Optimized Practice

Location: General

Description: It's worth noting that some pragma versions are more frequently used by Solidity programmers and have been thoroughly tested in several security audits. There is a json file in the Solidity Github repository that lists all defects found in various compiler versions.

Comment: If at all feasible, use the most recent stable pragma version that has been carefully tested to avoid any unknown vulnerabilities.

Consider deploying with any of the following Solidity versions:

- 0.5.16 - 0.5.17
- 0.6.11 - 0.6.12
- 0.7.5 - 0.7.6
- 0.8.4 - 0.8.7

Status: Mitigated

Analysis

CCO2.sol

Issue: Lock pragmas to a specific compiler version.

Severity: Lowest / Clean Coding / Optimized Practice

Location: General

Description: The current pragma Solidity directive is not locked to a fixed version.

```
pragma solidity >=0.4.22 <0.9.0;
```

Comment: It's best to specify a fixed compiler version to guarantee that the bytecode generated is consistent across builds. Locking the pragma helps ensure that contracts do not accidentally get deployed using, for example, that latest compiler, which may have a higher risker of undiscovered bugs. This is especially significant if the code is verified at the bytecode level.

Status: Mitigated

Analysis

PoCC.sol

Issue: Possibility to declare public function as external.

Severity: Lowest / Code Style / Optimized Practice

Location: L37-51, L53-65, L67-77, L83-103, L114-120, L132-134, L136-138

Description: In order to save gas, public functions that are never called by the contract could be declared external.

Comment: We recommend to consider using the external attribute for functions that are never called from the contract.

Status: Mitigated

Analysis

PoCC.sol

Issue: Version of Solidity.

Severity: Lowest / Clean Coding / Optimized Practice

Location: General

Description: It's worth noting that some pragma versions are more frequently used by Solidity programmers and have been thoroughly tested in several security audits. There is a json file in the Solidity Github repository that lists all defects found in various compiler versions.

Comment: If at all feasible, use the most recent stable pragma version that has been carefully tested to avoid any unknown vulnerabilities.

Consider deploying with any of the following Solidity versions:

- 0.5.16 - 0.5.17
- 0.6.11 - 0.6.12
- 0.7.5 - 0.7.6
- 0.8.4 - 0.8.7

Status: Mitigated

Analysis

PoCC.sol

Issue: Lock pragmas to a specific compiler version.

Severity: Lowest / Clean Coding / Optimized Practice

Location: General

Description: The current pragma Solidity directive is not locked to a fixed version.

```
pragma solidity >=0.4.22 <0.9.0;
```

Comment: It's best to specify a fixed compiler version to guarantee that the bytecode generated is consistent across builds. Locking the pragma helps ensure that contracts do not accidentally get deployed using, for example, that latest compiler, which may have a higher risker of undiscovered bugs. This is especially significant if the code is verified at the bytecode level.

Status: Mitigated

Testing Standards

The goal of the audit was to find any potential smart contract security problems and vulnerabilities. The information in this report should be used to understand the smart contract's risk exposure and as a guide to improving the smart contract's security posture by addressing the concerns that were discovered.

The blockchain platform is used to deploy and execute smart contracts. The platform, its programming language, and other smart contract-related applications all have vulnerabilities that may be exploited. As a result, the audit cannot ensure the audited smart contract(s) explicit security. Audits can't make statements or warranties on security of the code. It also cannot be deemed an adequate assessment of the code's utility and safety, bug-free status, or any statements of the smart contract.

While we did our best in completing the study and publishing this report, it is crucial to emphasize that you should not rely only on it; we advocate all projects doing many independent audits and participating in a public bug bounty program to assure smart contract security.

Testing Standards

1. Gather all relevant data.
2. Perform a preliminary visual examination of all documents and contracts.
3. Find security holes with specialist tools & manual review with independent experts.
4. Create and distribute a report.



SAULIDITY



Smart Contract
Audit



saulidity.com
Saulidity
@Saulidity