Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project:	Cronos Coin
Website:	cronos-pos.org
Platform:	Ethereum
Language:	Solidity
Date:	March 15th, 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 Cronos coin smart contract from cronos-pos.org was audited extensively. The audit has been performed using manual analysis as well as using automated software tools. This report presents all the findings regarding the audit performed on March 15th, 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 Cronos Coin is a standard smart contract.
- **CroToken Contract:** The main token contract inheriting from `ReleasableToken`, `MintableToken`, and `UpgradeableToken`.
    - **UpdatedTokenInformation:** Event for updating token information.
    - **name, symbol, decimals:** Token details.
    - Reserve wallets for various purposes.
    - The constructor initializes the token with initial supply and reserve wallets.
    - Overrides `releaseTokenTransfer` and `canUpgrade` to integrate functionalities.
- The contract is deployed with the token name, symbol, initial supply, decimals, mintable status, and addresses for reserve wallets.
- If the token is not mintable, minting is finished after the initial supply is allocated.
- This contract provides a comprehensive template for an ERC20 token with advanced features, suitable for various use cases in decentralized applications.
- Overall, the contract hierarchy provides a comprehensive implementation of a token with features like minting, releasing, upgrading, and various reserve wallets for different purposes.

Audit scope

Name	**Code Review and Security Analysis Report for Cronos Coin Smart Contract**
Platform	**Ethereum**
File	CroToken.sol
Smart Contract Code	0xa0b73e1ff0b80914ab6fe0444e65848c4c34450b
Audit Date	March 15th, 2024

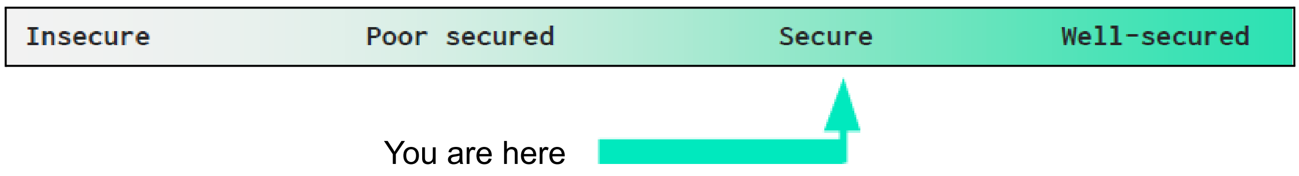
Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Tokenomics: • Name: CRO • Symbol: CRO • Decimals: 8 • Total supply: 100 billion • Get Upgrade state: 2	**YES, This is valid.**
Release Agent Control: • The token is released to be transferable. • Release the tokens to the wild. **Upgrade Master Control:** • Set the upgrade master address. **Ownership Control:** • Update Release agent address. • Stop minting new tokens. • Update Transfer agent address. • Mint tokens by the mint permission owner. • The current owner can transfer the ownership. • The owner can renounce ownership.	**YES, This is valid.** **We suggest renouncing ownership once the ownership functions are not needed. This is to make the smart contract 100% decentralized.**
Auxiliary Contracts: • **UpgradeAgent:** Interface for the upgrade agent. • **originalSupply:** Original supply of the token. • **isUpgradeAgent and upgradeFrom:** Functions to be implemented by the	**YES, This is valid.**

upgrade agent.	
Key Functionalities: ● **Minting:** The owner can mint new tokens until the minting is finished. ● **Releasing:** Tokens can be restricted from transfer until released by a release agent. ● **Upgrading:** Tokens can be upgraded to a new contract using an upgrade agent. ● **Multiple Reserve Wallets:** Initial supply is divided among different reserve wallets for various purposes.	**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. The 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, 1 low, and 7 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 not set	Moderated
	“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?	Detected
● Is Anti-bot?	Not Detected
● Is it a Blacklist?	Not Detected
● Blacklist Check	No
● Can Mint?	Yes
● Is it Proxy?	Not Detected
● 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 Cronos Coin 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 Cronos Coin.

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 a Cronos Coin 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 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

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Upgradeable Token	No Issue
2	releaseTokenTransfer	write	access only Release Agent	No Issue
3	canUpgrade	read	Passed	No Issue
4	totalSupply	read	Passed	No Issue
5	upgrade	write	Passed	No Issue
6	setUpgradeAgent	external	Passed	No Issue
7	getUpgradeState	read	Passed	No Issue
8	setUpgradeMaster	write	Passed	No Issue
9	canUpgrade	read	Passed	No Issue
10	canTransfer	modifier	Lengthy require Error Message	Refer Audit Findings
11	setReleaseAgent	write	Missing zero address validation	Refer Audit Findings
12	setTransferAgent	write	Missing zero address validation	Refer Audit Findings
13	releaseTokenTransfer	write	access only Release Agent	No Issue
14	inReleaseState	modifier	Passed	No Issue
15	onlyReleaseAgent	modifier	Passed	No Issue
16	transfer	write	can Transfer	No Issue
17	transferFrom	write	Missing required error message	Refer Audit Findings
18	canMint	modifier	Missing required error message	Refer Audit Findings
19	hasMintPermission	modifier	Missing required error message	Refer Audit Findings
20	mint	write	Unlimited token minting	Refer Audit Findings
21	finishMinting	write	access only Owner	No Issue
22	transferFrom	write	Passed	No Issue
23	approve	write	Passed	No Issue
24	allowance	read	Passed	No Issue
25	increaseApproval	write	Passed	No Issue
26	decreaseApproval	write	Passed	No Issue
27	allowance	read	Passed	No Issue
28	transferFrom	write	Passed	No Issue
29	approve	write	Passed	No Issue
30	totalSupply	read	Passed	No Issue
31	transfer	write	Missing required error message	Refer Audit Findings
32	balanceOf	read	Passed	No Issue
33	totalSupply	read	Passed	No Issue

34	balanceOf	read	Passed	No Issue
35	transfer	write	Passed	No Issue
36	onlyOwner	modifier	Missing required error message	Refer Audit Findings
37	renounceOwnership	write	access only Owner	No Issue
38	transferOwnership	write	access only Owner	No Issue
39	_transferOwnership	internal	Missing required error message	Refer Audit Findings
40	isUpgradeAgent	write	Passed	No Issue
41	upgradeFrom	write	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

(1) Unlimited token minting: MintableToken.sol

```
function mint(
    address _to,
    uint256 _amount
  )
    public
    hasMintPermission
    canMint
    returns (bool)
  {
    totalSupply_ = totalSupply_.add(_amount);
    balances[_to] = balances[_to].add(_amount);
    emit Mint(_to, _amount);
    emit Transfer(address(0), _to, _amount);
    return true;
  }
```

Token minting without any maximum limit is considered inappropriate for tokenomics.

Resolution: We recommend placing some limit on token minting to mitigate this issue.

Very Low / Informational / Best practices:

(1) Use the latest solidity version: CroToken.sol

```
pragma solidity ^0.4.13;
```

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) Missing required error message:

StandardToken.sol

```
function transferFrom(
    address _from,
    address _to,
    uint256 _value
  )
    public
    returns (bool)
  {
    require(_value <= balances[_from]);
    require(_value <= allowed[_from][msg.sender]);
    require(_to != address(0));

    balances[_from] = balances[_from].sub(_value);
    balances[_to] = balances[_to].add(_value);
    allowed[_from][msg.sender] = allowed[_from][msg.sender].sub(_value);
    emit Transfer(_from, _to, _value);
    return true;
  }
```

Ownable.sol

```
modifier onlyOwner() {
    require(msg.sender == owner);
    _;
  }
function _transferOwnership(address _newOwner) internal {
    require(_newOwner != address(0));
    emit OwnershipTransferred(owner, _newOwner);
    owner = _newOwner;
  }
```

BasicToken.sol

```
function transfer(address _to, uint256 _value) public returns (bool) {
    require(_value <= balances[msg.sender]);
    require(_to != address(0));

    balances[msg.sender] = balances[msg.sender].sub(_value);
    balances[_to] = balances[_to].add(_value);
    emit Transfer(msg.sender, _to, _value);
    return true;
  }
```

MintableToken.sol

```
modifier canMint() {
    require(!mintingFinished);
    _;
  }

  modifier hasMintPermission() {
    require(msg.sender == owner);
    _;
  }
```

There is no error message set in the required condition.

Resolution: We suggest setting relevant error messages to identify the failure of the transaction.

(3) Explicit Visibility for State Variables Warning: **MintableToken.sol**

```
bool canUpgrade_ = true;
```

The warning is related to the visibility of state variables in your Solidity code.

Resolution: We recommend updating the code to explicitly mark the visibility of state variables using the internal or public keyword, depending on the intended visibility.

(4) Make variable constant: **CroToken.sol**

```
string public name;
string public symbol;
uint8 public decimals;
```

These variables' values will remain unchanged. so, we suggest making them constant. It is best practice and it also saves some gas. Just add a constant keyword.

Resolution: We suggest declaring this variable as constant to save some gas.

(5) Missing zero address validation: **ReleasableToken.sol**

```
function setTransferAgent(address addr, bool state) public onlyOwner
inReleaseState(false) {
        transferAgents[addr] = state;
    }
function setReleaseAgent(address addr) public onlyOwner
inReleaseState(false) {

        // We don't do interface check here as we might want to a
normal wallet address to act as a release agent
        releaseAgent = addr;
    }
```

Detects missing zero address validation.

Resolution: We suggest first Check that the address is not zero.

(6) Lengthy require Error Message: **ReleasableToken.sol**

```
modifier canTransfer(address _sender) {
        require(released || transferAgents[_sender], "For the token
to be able to transfer: it's required that the crowdsale is in
released state; or the sender is a transfer agent.");
        _;
    }
```

It could potentially lead to gas exhaustion, user confusion, or any other security concerns.

Resolution: We suggest breaking down complex conditions into smaller, more readable required statements or providing more user-friendly error messages.

(7) Unused event and enum:

CroToken.sol

```
event UpdatedTokenInformation(string newName, string newSymbol);
```

UpgradeableToken.sol

```
enum UpgradeState {Unknown, NotAllowed, WaitingForAgent,
ReadyToUpgrade
```

The event UpdatedTokenInformation is declared but never used and enum UpgradeState is declared but never used.

Resolution: We suggest removing unused events and enums from smart contracts.

Centralization

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

CroToken.sol

- releaseTokenTransfer: The token is released to be transferable by the Release Agent.

UpgradeableToken.sol

- setUpgradeMaster: The upgrade master address can be updated by the upgrade master.

ReleasableToken.sol

- setReleaseAgent: Release agent address can be updated by the owner.
- setTransferAgent: The transfer agent address can be updated by the owner.
- releaseTokenTransfer: Release the tokens to the wild by the Release Agent.

MintableToken.sol

- mint: mint tokens by the mint permission owner.
- finishMinting: Stop minting new tokens by the owner.

Ownable.sol

- renounceOwnership: Allows the current owner to relinquish control of the contract.
- transferOwnership: Allows the current owner to transfer control of the contract to a new 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 1 low and 7 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.

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

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

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of 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).

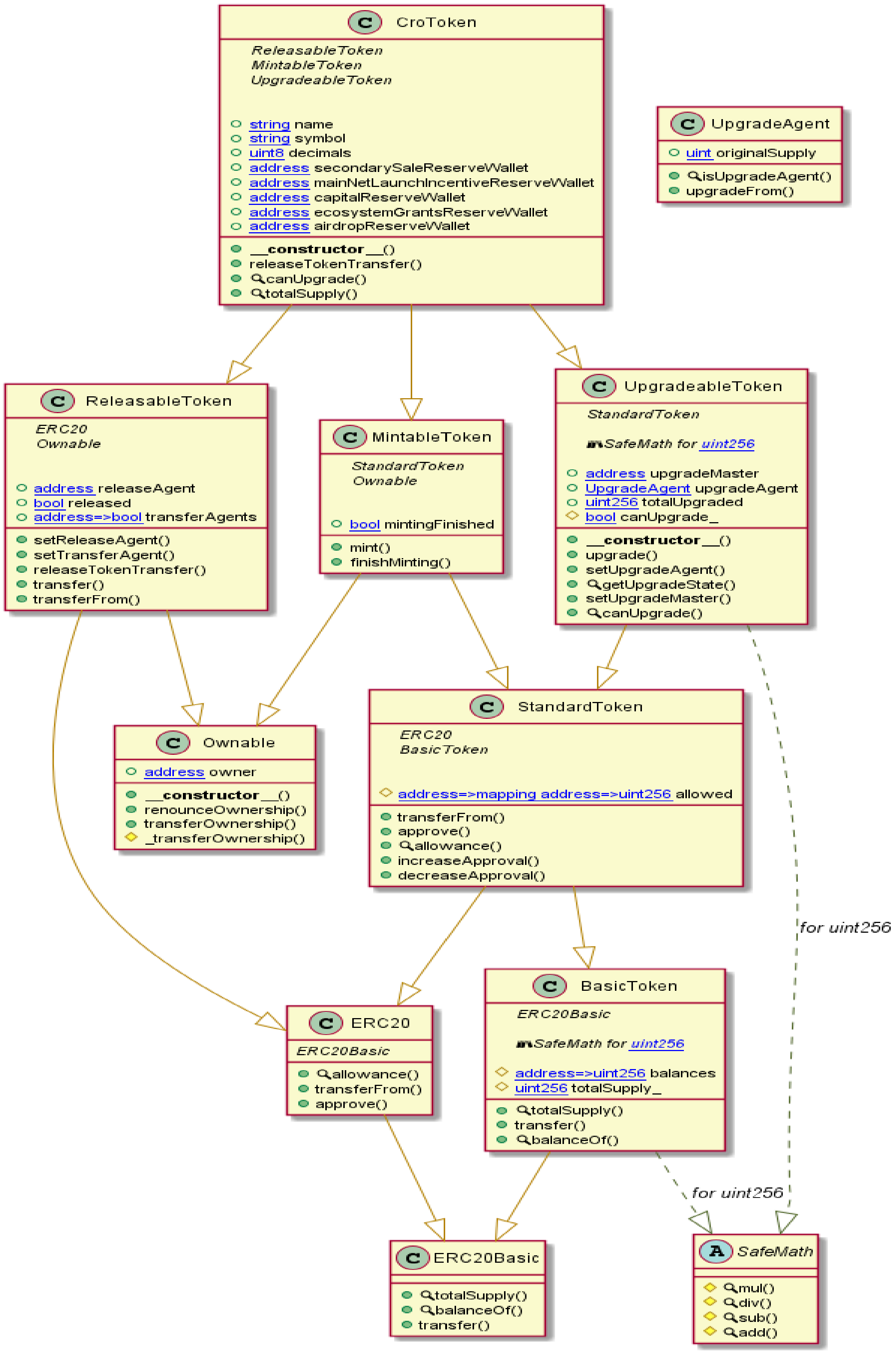
Due to the fact that the total number of test cases is unlimited, the audit makes no statements or warranties on the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bug-free status or any other statements of the contract. While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

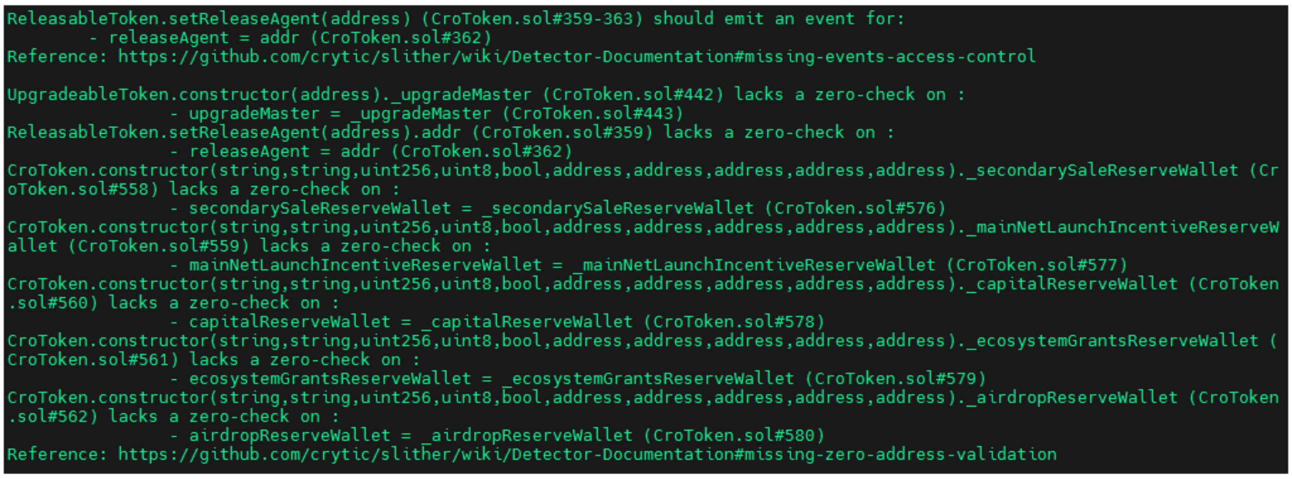


Slither Results Log

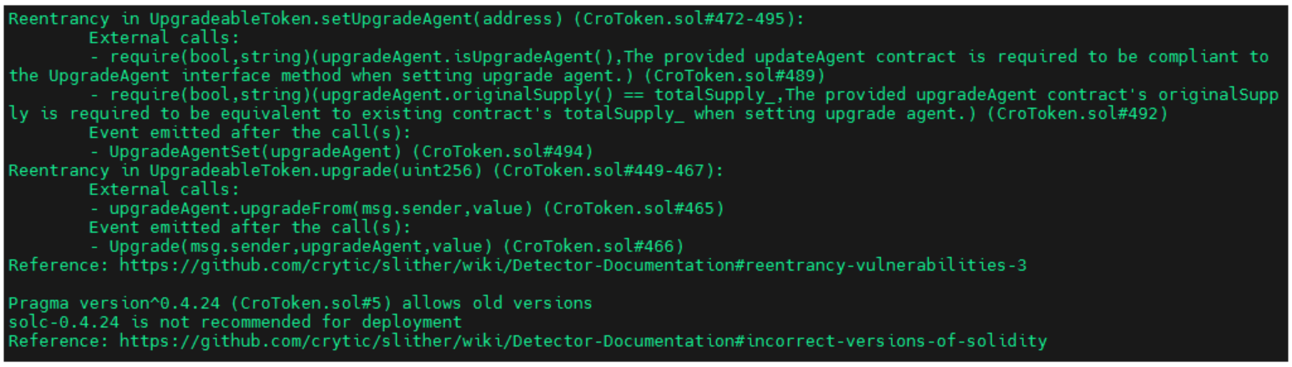
Slither is a Solidity static analysis framework that uses vulnerability detectors, displays contract details, and provides an API for writing custom analyses. It helps developers identify vulnerabilities, improve code comprehension, and prototype custom analyses quickly. The analysis includes a report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

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

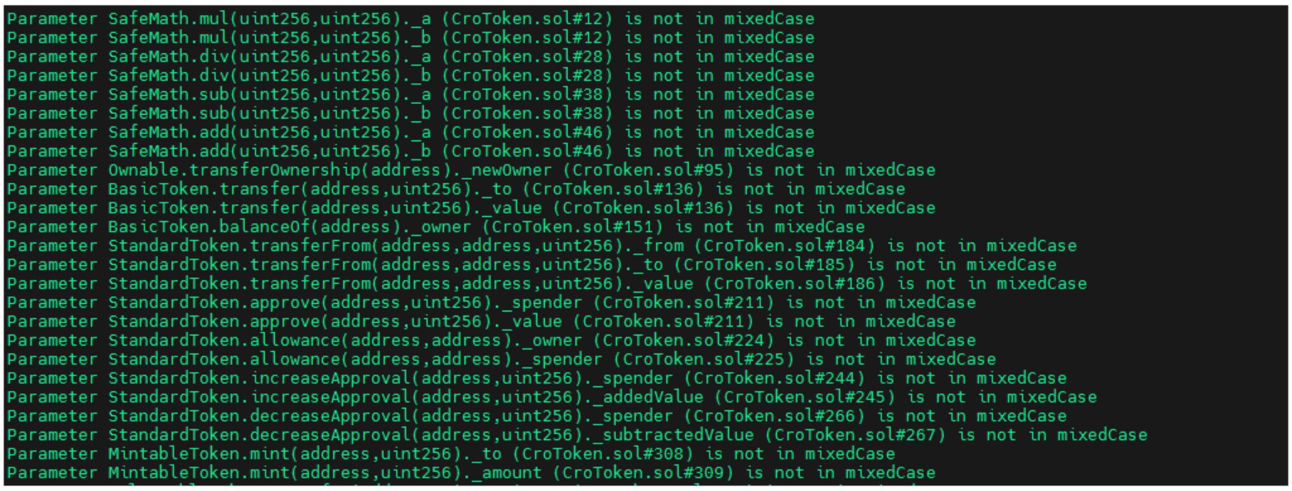
Slither Log >> CroToken.sol



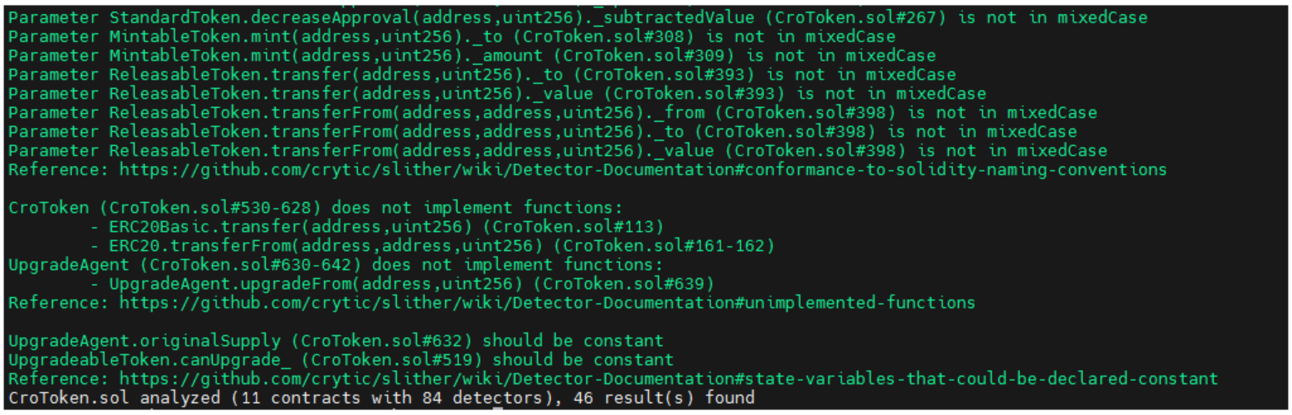
```
ReleasableToken.setReleaseAgent(address) (CroToken.sol#359-363) should emit an event for:
        - releaseAgent = addr (CroToken.sol#362)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-access-control

UpgradeableToken.constructor(address)._upgradeMaster (CroToken.sol#442) lacks a zero-check on :
                - upgradeMaster = _upgradeMaster (CroToken.sol#443)
ReleasableToken.setReleaseAgent(address).addr (CroToken.sol#359) lacks a zero-check on :
                - releaseAgent = addr (CroToken.sol#362)
CroToken.constructor(string,string,uint256,uint8,bool,address,address,address,address,address)._secondarySaleReserveWallet (Cr
oToken.sol#558) lacks a zero-check on :
                - secondarySaleReserveWallet = _secondarySaleReserveWallet (CroToken.sol#576)
CroToken.constructor(string,string,uint256,uint8,bool,address,address,address,address,address)._mainNetLaunchIncentiveReserveW
allet (CroToken.sol#559) lacks a zero-check on :
                - mainNetLaunchIncentiveReserveWallet = _mainNetLaunchIncentiveReserveWallet (CroToken.sol#577)
CroToken.constructor(string,string,uint256,uint8,bool,address,address,address,address,address)._capitalReserveWallet (CroToken
.sol#560) lacks a zero-check on :
                - capitalReserveWallet = _capitalReserveWallet (CroToken.sol#578)
CroToken.constructor(string,string,uint256,uint8,bool,address,address,address,address,address)._ecosystemGrantsReserveWallet (
CroToken.sol#561) lacks a zero-check on :
                - ecosystemGrantsReserveWallet = _ecosystemGrantsReserveWallet (CroToken.sol#579)
CroToken.constructor(string,string,uint256,uint8,bool,address,address,address,address,address)._airdropReserveWallet (CroToken
.sol#562) lacks a zero-check on :
                - airdropReserveWallet = _airdropReserveWallet (CroToken.sol#580)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
```



```
Reentrancy in UpgradeableToken.setUpgradeAgent(address) (CroToken.sol#472-495):
        External calls:
        - require(bool,string)(upgradeAgent.isUpgradeAgent(),The provided updateAgent contract is required to be compliant to
the UpgradeAgent interface method when setting upgrade agent.) (CroToken.sol#489)
        - require(bool,string)(upgradeAgent.originalSupply() == totalSupply_,The provided upgradeAgent contract's originalSupp
ly is required to be equivalent to existing contract's totalSupply_ when setting upgrade agent.) (CroToken.sol#492)
        Event emitted after the call(s):
        - UpgradeAgentSet(upgradeAgent) (CroToken.sol#494)
Reentrancy in UpgradeableToken.upgrade(uint256) (CroToken.sol#449-467):
        External calls:
        - upgradeAgent.upgradeFrom(msg.sender,value) (CroToken.sol#465)
        Event emitted after the call(s):
        - Upgrade(msg.sender,upgradeAgent,value) (CroToken.sol#466)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

Pragma version^0.4.24 (CroToken.sol#5) 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
```



```
Parameter SafeMath.mul(uint256,uint256)._a (CroToken.sol#12) is not in mixedCase
Parameter SafeMath.mul(uint256,uint256)._b (CroToken.sol#12) is not in mixedCase
Parameter SafeMath.div(uint256,uint256)._a (CroToken.sol#28) is not in mixedCase
Parameter SafeMath.div(uint256,uint256)._b (CroToken.sol#28) is not in mixedCase
Parameter SafeMath.sub(uint256,uint256)._a (CroToken.sol#38) is not in mixedCase
Parameter SafeMath.sub(uint256,uint256)._b (CroToken.sol#38) is not in mixedCase
Parameter SafeMath.add(uint256,uint256)._a (CroToken.sol#46) is not in mixedCase
Parameter SafeMath.add(uint256,uint256)._b (CroToken.sol#46) is not in mixedCase
Parameter Ownable.transferOwnership(address)._newOwner (CroToken.sol#95) is not in mixedCase
Parameter BasicToken.transfer(address,uint256)._to (CroToken.sol#136) is not in mixedCase
Parameter BasicToken.transfer(address,uint256)._value (CroToken.sol#136) is not in mixedCase
Parameter BasicToken.balanceOf(address)._owner (CroToken.sol#151) is not in mixedCase
Parameter StandardToken.transferFrom(address,address,uint256)._from (CroToken.sol#184) is not in mixedCase
Parameter StandardToken.transferFrom(address,address,uint256)._to (CroToken.sol#185) is not in mixedCase
Parameter StandardToken.transferFrom(address,address,uint256)._value (CroToken.sol#186) is not in mixedCase
Parameter StandardToken.approve(address,uint256)._spender (CroToken.sol#211) is not in mixedCase
Parameter StandardToken.approve(address,uint256)._value (CroToken.sol#211) is not in mixedCase
Parameter StandardToken.allowance(address,address)._owner (CroToken.sol#224) is not in mixedCase
Parameter StandardToken.allowance(address,address)._spender (CroToken.sol#225) is not in mixedCase
Parameter StandardToken.increaseApproval(address,uint256)._spender (CroToken.sol#244) is not in mixedCase
Parameter StandardToken.increaseApproval(address,uint256)._addedValue (CroToken.sol#245) is not in mixedCase
Parameter StandardToken.decreaseApproval(address,uint256)._spender (CroToken.sol#266) is not in mixedCase
Parameter StandardToken.decreaseApproval(address,uint256)._subtractedValue (CroToken.sol#267) is not in mixedCase
Parameter MintableToken.mint(address,uint256)._to (CroToken.sol#308) is not in mixedCase
Parameter MintableToken.mint(address,uint256)._amount (CroToken.sol#309) is not in mixedCase
```

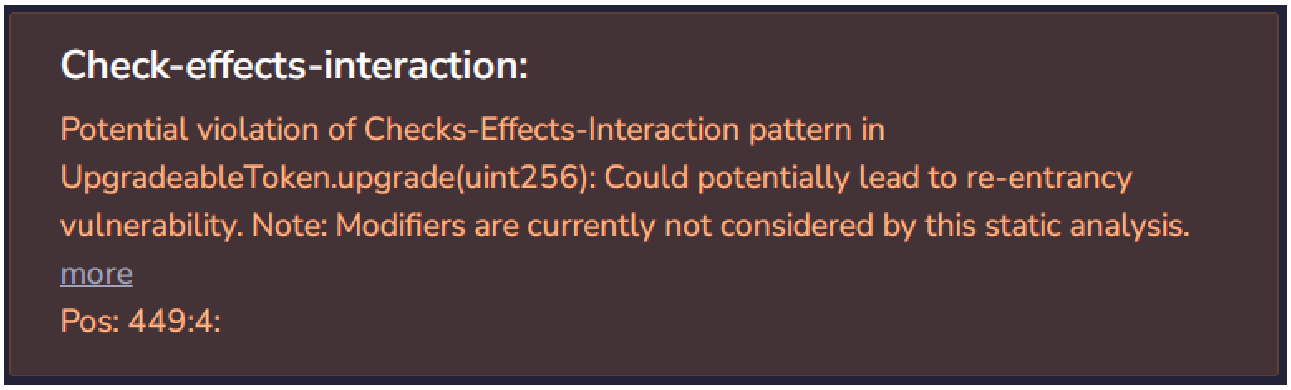


```
Parameter StandardToken.decreaseApproval(address,uint256)._subtractedValue (CroToken.sol#267) is not in mixedCase
Parameter MintableToken.mint(address,uint256)._to (CroToken.sol#308) is not in mixedCase
Parameter MintableToken.mint(address,uint256)._amount (CroToken.sol#309) is not in mixedCase
Parameter ReleasableToken.transfer(address,uint256)._to (CroToken.sol#393) is not in mixedCase
Parameter ReleasableToken.transfer(address,uint256)._value (CroToken.sol#393) is not in mixedCase
Parameter ReleasableToken.transferFrom(address,address,uint256)._from (CroToken.sol#398) is not in mixedCase
Parameter ReleasableToken.transferFrom(address,address,uint256)._to (CroToken.sol#398) is not in mixedCase
Parameter ReleasableToken.transferFrom(address,address,uint256)._value (CroToken.sol#398) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

CroToken (CroToken.sol#530-628) does not implement functions:
        - ERC20Basic.transfer(address,uint256) (CroToken.sol#113)
        - ERC20.transferFrom(address,address,uint256) (CroToken.sol#161-162)
UpgradeAgent (CroToken.sol#630-642) does not implement functions:
        - UpgradeAgent.upgradeFrom(address,uint256) (CroToken.sol#639)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unimplemented-functions

UpgradeAgent.originalSupply (CroToken.sol#632) should be constant
UpgradeableToken.canUpgrade_ (CroToken.sol#519) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
CroToken.sol analyzed (11 contracts with 84 detectors), 46 result(s) found
```

Solidity Static Analysis

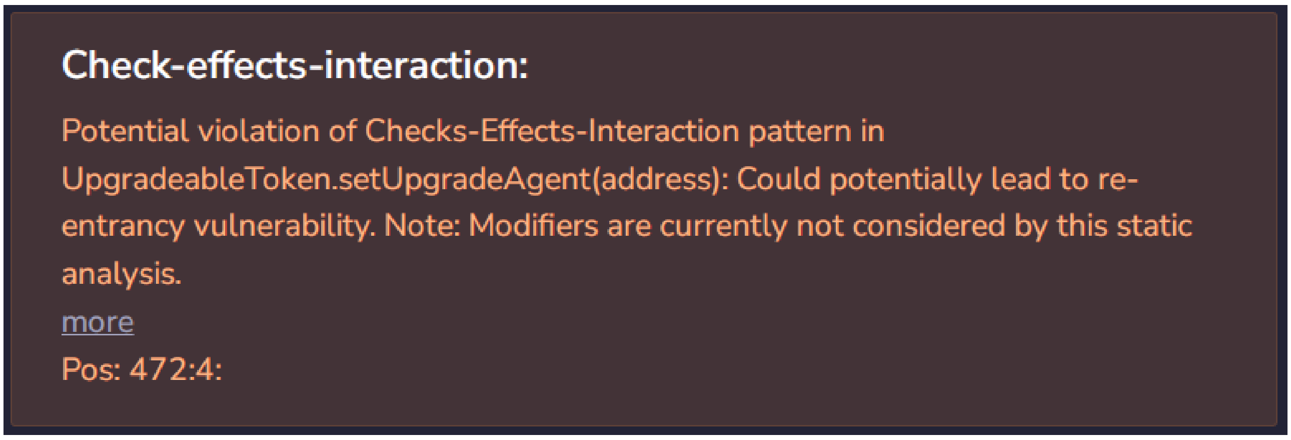
Static code analysis is used to identify many common coding problems before a program is released. It involves examining the code manually or using tools to automate the process. Static code analysis tools can automatically scan the code without executing it.

CroToken.sol



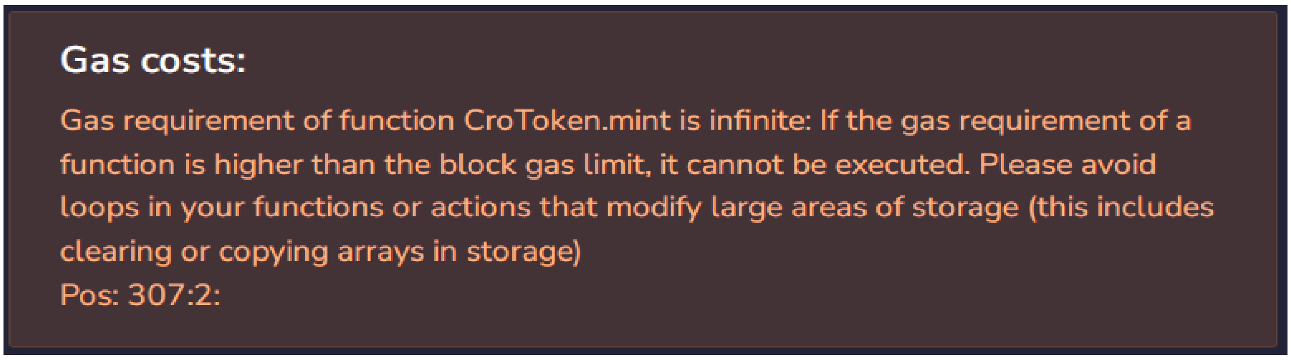
Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in UpgradeableToken.upgrade(uint256): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 449:4:



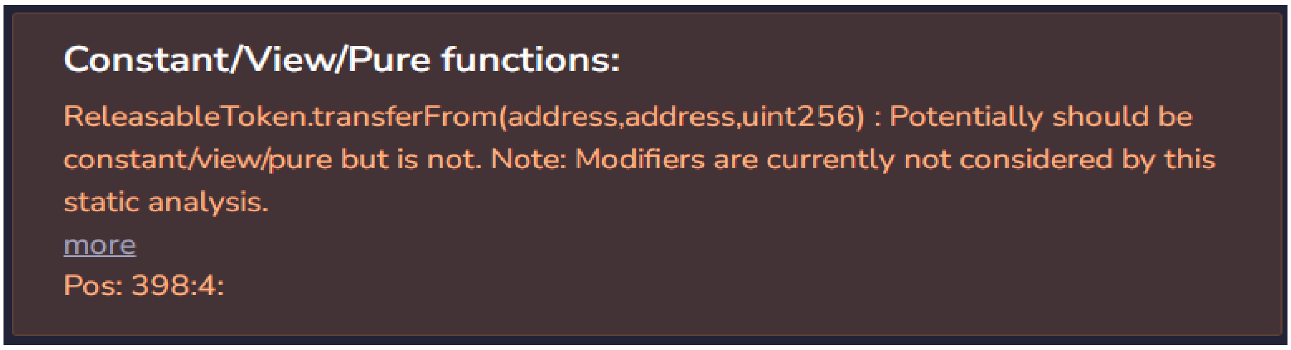
Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in UpgradeableToken.setUpgradeAgent(address): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 472:4:



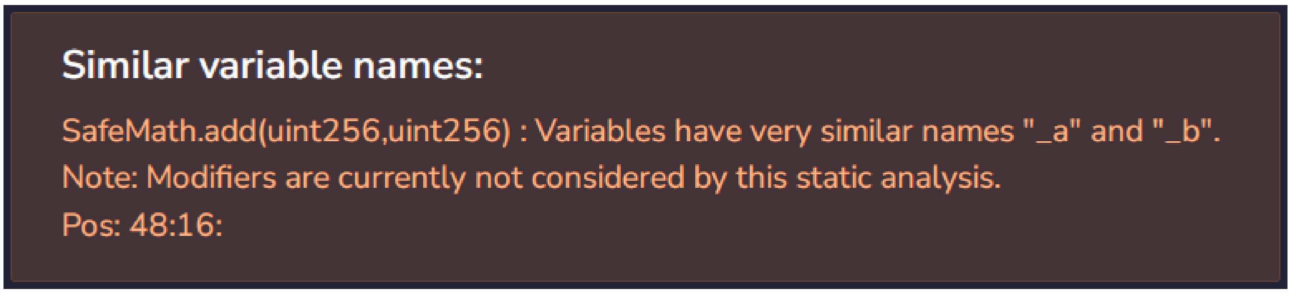
Gas costs:

Gas requirement of function CroToken.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: 307:2:



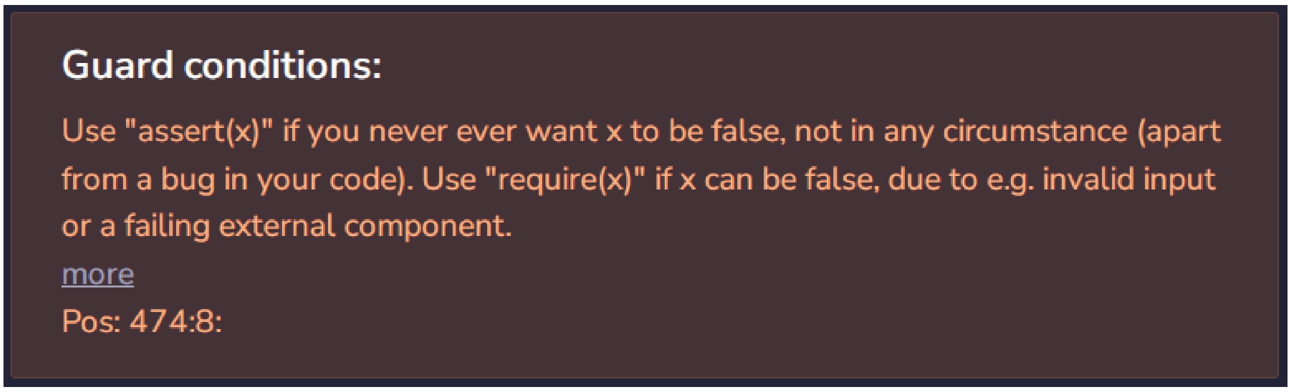
Constant/View/Pure functions:

ReleasableToken.transferFrom(address,address,uint256) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 398:4:



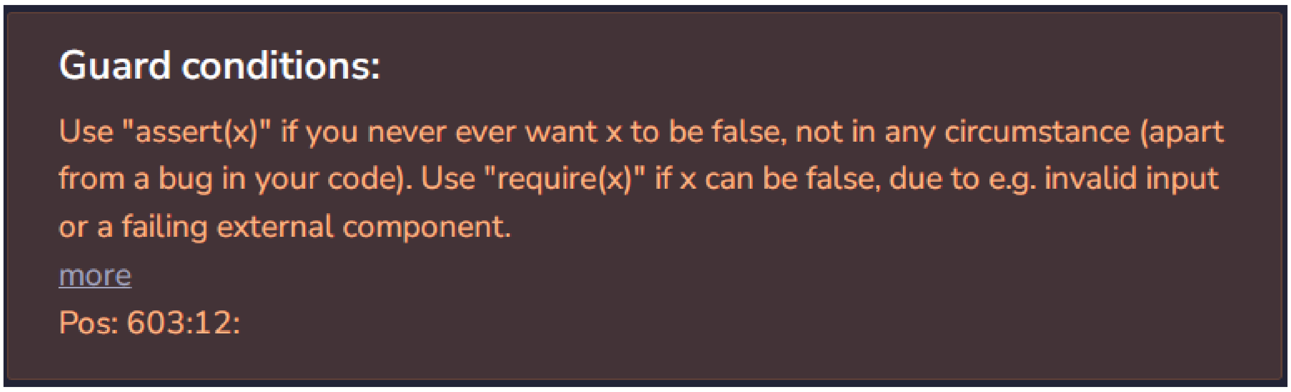
Similar variable names:

SafeMath.add(uint256,uint256) : Variables have very similar names "_a" and "_b". Note: Modifiers are currently not considered by this static analysis.
Pos: 48:16:



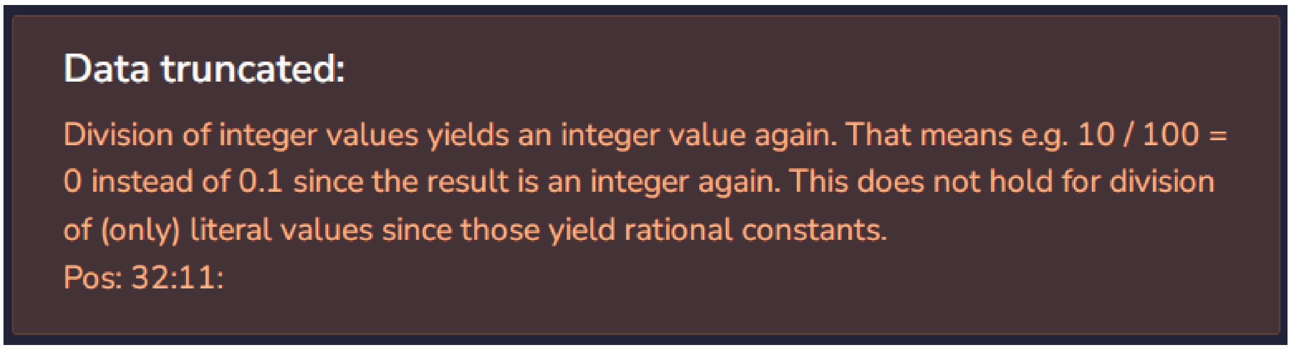
Guard conditions:

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: 474:8:



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: 603:12:



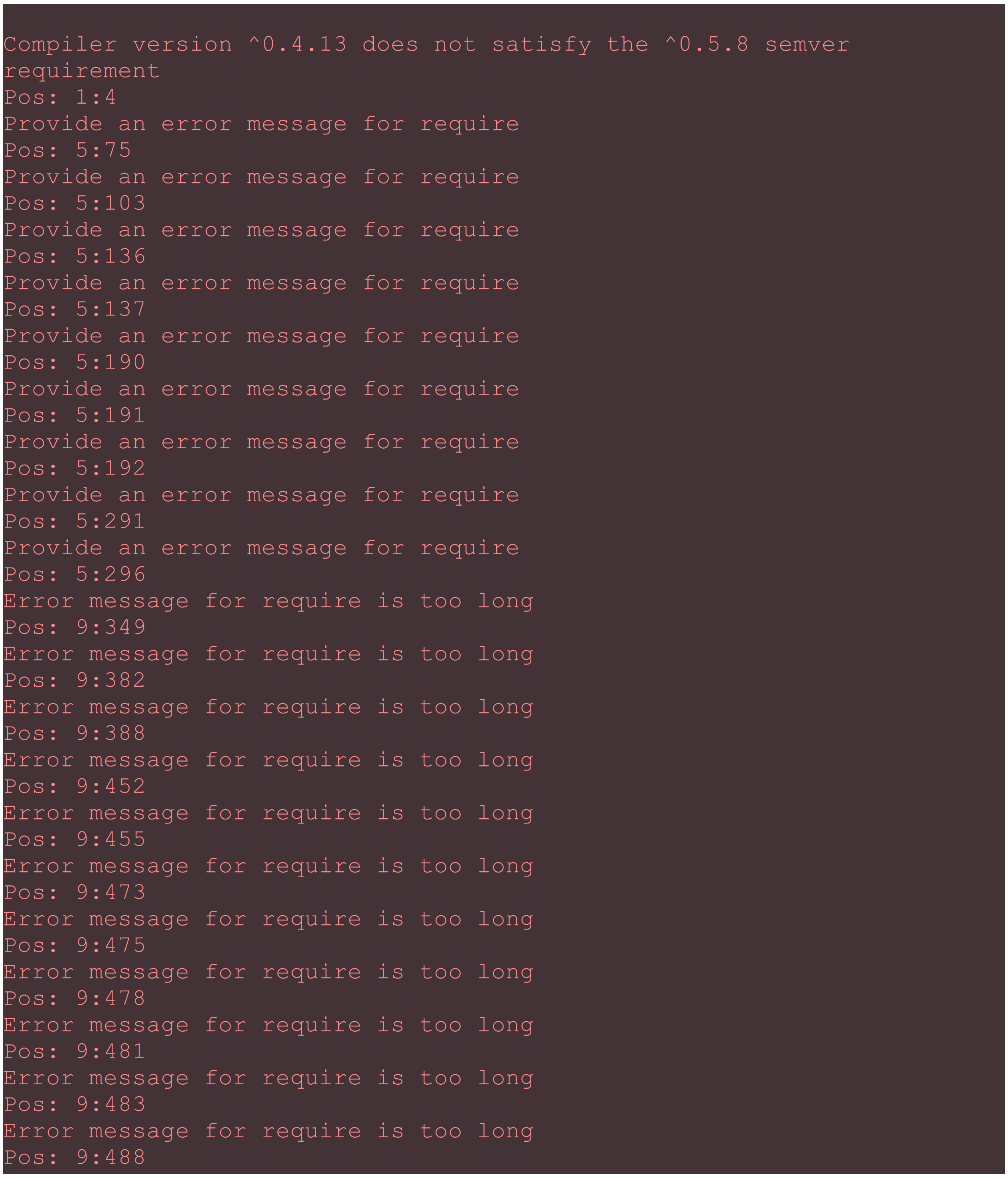
Data truncated:

Division of integer values yields an integer value again. That means e.g. 10 / 100 = 0 instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants.
Pos: 32:11:

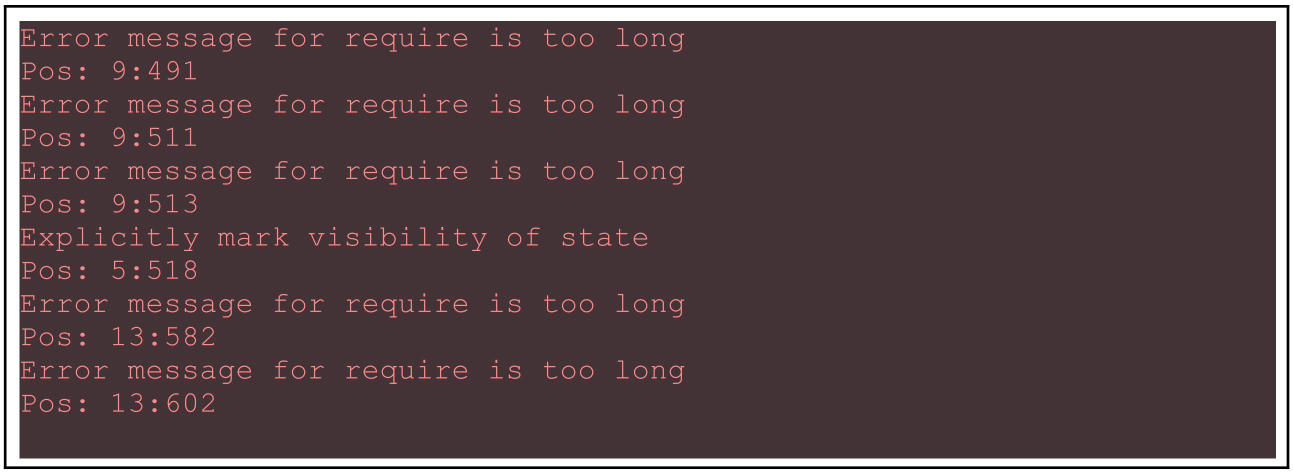
Solhint Linter

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.

CroToken.sol



```
Compiler version ^0.4.13 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:4
Provide an error message for require
Pos: 5:75
Provide an error message for require
Pos: 5:103
Provide an error message for require
Pos: 5:136
Provide an error message for require
Pos: 5:137
Provide an error message for require
Pos: 5:190
Provide an error message for require
Pos: 5:191
Provide an error message for require
Pos: 5:192
Provide an error message for require
Pos: 5:291
Provide an error message for require
Pos: 5:296
Error message for require is too long
Pos: 9:349
Error message for require is too long
Pos: 9:382
Error message for require is too long
Pos: 9:388
Error message for require is too long
Pos: 9:452
Error message for require is too long
Pos: 9:455
Error message for require is too long
Pos: 9:473
Error message for require is too long
Pos: 9:475
Error message for require is too long
Pos: 9:478
Error message for require is too long
Pos: 9:481
Error message for require is too long
Pos: 9:483
Error message for require is too long
Pos: 9:488
```



```
Error message for require is too long
Pos: 9:491
Error message for require is too long
Pos: 9:511
Error message for require is too long
Pos: 9:513
Explicitly mark visibility of state
Pos: 5:518
Error message for require is too long
Pos: 13:582
Error message for require is too long
Pos: 13:602
```

Software analysis result:

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