Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project:	CRO Token
Website:	cronos-pos.org
Platform:	Ethereum
Language:	Solidity
Date:	May 7th, 2024

Table of contents

`

Introduction

As part of EtherAuthority's community smart contracts audit initiatives, the CRO Token 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 May 7th, 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 inherits from several other contracts and libraries to implement functionalities like access control, ERC20 token standards, pausing, burning, and minting.
- Here's a breakdown of what this contract does:
  - **Libraries:** The contract imports and uses various libraries like EnumerableSet, SafeMath, and Address to implement set operations, safe arithmetic operations, and address-related functions.
  - **AccessControl:** The contract defines roles (MINTER_ROLE and PAUSER_ROLE) and assigns them to specific accounts. These roles control who can mint new tokens and pause/unpause the contract.
  - **ERC20 Token:** The contract implements the ERC20 token standard with functionalities like transferring tokens, approving spending, allowance management, total supply, balance inquiries, etc.
  - **Burnable:** This contract allows tokens to be burned (destroyed) by the token owner or by another authorized account.
  - **Pausable:** The contract can be paused and unpaused, preventing token transfers while paused to avoid potential issues or attacks.
  - **Constructor:** The constructor initializes the contract by setting up the default admin role and assigning the minter and pauser roles to the contract deployer.

- Overall, this contract provides a standard ERC20 token with additional features such as access control, burning, and pausing.

Audit scope

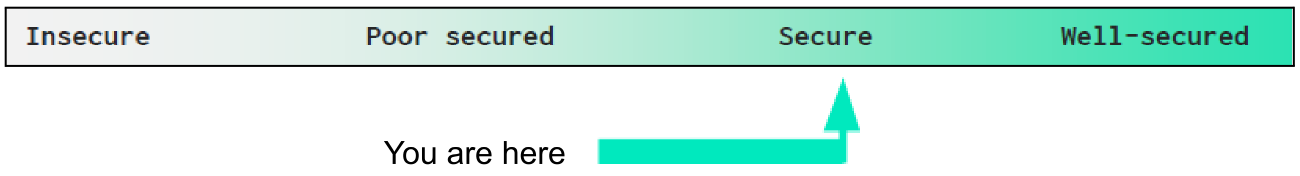
Name	**Code Review and Security Analysis Report for CRO Token Smart Contract**
Platform	**Ethereum**
Language	**Solidity**
File	CroToken.sol
Smart Contract Code	0xa0b73e1ff0b80914ab6fe0444e65848c4c34450b
Audit Date	May 7th, 2024

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Tokenomics: • Name: CRO • Symbol: CRO • Decimals: 8 • Total Supply: 100 billion	**YES, This is valid.**
Other Specifications: • The ReleaseAgent can release tokens before transferable. • An upgrade agent address be updated by the upgrade Master. • An upgrade master address be updated by the current upgradeMaster. • ReleaseAgent can release the tokens to the wild.	**YES, This is valid.**
Owner Specifications: • The owner can be minting new tokens • The owner can stop minting new tokens. • ReleaseAgent address can be updated by the owner • The owner can allow a particular address (a crowd sale contract) to transfer tokens despite the lock-up period • An upgrade master address be updated by the current upgradeMaster. • Renouncing ownership will leave the contract without an owner. • Allows the current owner to transfer control of the contract to a new owner	**YES, This is valid. We suggest renouncing ownership once the ownership functions are not needed. This is to make the smart contract 100% decentralized.**

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	Passed
	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	Moderated
	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	No
● Fee Check	Not Detected
● Is Honeypot	Not Detected
● Trading Cooldown	Not Detected
● Can Pause Trade?	Not Detected
● Pause Transfer?	No
● Max Tax?	No
● Is it Anti-whale?	Not Detected
● Is Anti-bot?	Not Detected
● Is it a Blacklist?	No
● Blacklist Check	No
● Can Mint?	Yes
● Is it a Proxy Contract?	No
● Can Take Ownership?	Yes
● Creator Percentage?	0.00%
● 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 contract contain Libraries, Smart contracts, inherits, and Interfaces. This is a compact and well-written smart contract.

The libraries in CRO 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 CRO 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 a CRO 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

CroToken.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	releaseTokenTransfer	write	access only Release Agent	No Issue
3	canUpgrade	read	Passed	No Issue
4	totalSupply	read	Passed	No Issue
5	isUpgradeAgent	write	Passed	No Issue
6	upgradeFrom	write	Passed	No Issue
7	upgrade	write	Passed	No Issue
8	setUpgradeAgent	external	Passed	No Issue
9	getUpgradeState	read	Passed	No Issue
10	setUpgradeMaster	write	Critical operation lacks event log	Refer Audit Findings
11	canUpgrade	read	Passed	No Issue
12	canTransfer	modifier	Passed	No Issue
13	setReleaseAgent	write	Critical operation lacks event log, Missing zero address validation	Refer Audit Findings
14	setTransferAgent	write	access only Owner	No Issue
15	releaseTokenTransfer	write	access only Release Agent	No Issue
16	inReleaseState	modifier	Passed	No Issue
17	onlyReleaseAgent	modifier	Passed	No Issue
18	transfer	write	can Transfer	No Issue
19	transferFrom	write	can Transfer	No Issue
20	canMint	modifier	Passed	No Issue
21	hasMintPermission	modifier	Passed	No Issue
22	mint	write	Centralization Risk	Refer Audit Findings
23	finishMinting	write	access only Owner	No Issue
24	transferFrom	write	Passed	No Issue
25	approve	write	Passed	No Issue
26	allowance	read	Passed	No Issue
27	increaseApproval	write	Passed	No Issue
28	decreaseApproval	write	Passed	No Issue
29	allowance	read	Passed	No Issue
30	transferFrom	write	Passed	No Issue
31	approve	write	Passed	No Issue
32	totalSupply	read	Passed	No Issue
33	transfer	write	Passed	No Issue
34	balanceOf	read	Passed	No Issue

35	totalSupply	read	Passed	No Issue
36	balanceOf	read	Passed	No Issue
37	transfer	write	Passed	No Issue
38	onlyOwner	modifier	Passed	No Issue
39	renounceOwnership	write	access only Owner	No Issue
40	transferOwnership	write	access only Owner	No Issue
41	_transferOwnership	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

(1) Critical operation lacks event log:

```
 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;
    }
function setUpgradeMaster(address master) public {
        require(master != address(0), "The provided upgradeMaster is
required to be a non-empty address when setting upgrade master.");
        require(msg.sender == upgradeMaster, "Message sender is
required to be the original upgradeMaster when setting (new) upgrade
master.");

        upgradeMaster = master;
    }
```

Missing event log contract functions like:

- ReleasableToken: setReleaseAgent()
- UpgradeableToken: setUpgradeMaster()

Recommendation: We suggest considering adding events for sensitive actions, and emit them in the functions.

Very Low / Informational / Best practices:

(1) Missing SPDX-License-Identifier:
SPDX-License-Identifier is not written.

Recommendation: We suggest adding the SPDX License Identifier.

(2) Missing zero address validation:

```
 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 in setReleaseAgent() in the RelesableToken contract. Addresses should be checked before assignment or external call to make sure they are not zero addresses.

Recommendation: We suggest adding a zero-check for the passed-in address value to prevent unexpected errors.

(3) Missing error message in required condition:
It is best practice to add custom error messages in every required condition, which would be helpful in debugging as well as giving a clear indication of any transaction failure.
Some of the contracts that do not add error messages in the required conditions are:

- Ownable Contracts
- BasicToken Contracts
- StandardToken Contracts
- MintableToken Contracts

Recommendation: Add custom error messages in every required condition.

(4) Visibility can be external over the public:

Any functions which are not called internally should be declared as external. This saves some gas and is considered a good practice.

https://ethereum.stackexchange.com/questions/19380/external-vs-public-best-practices

(5) Unused event:

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

UpdatedTokenInformation event is defined, but not used in code.

Recommendation: We suggest removing unused events.

(6) Make variables constant:

UpgradeAgent Contract:

```
uint public originalSupply;
```

UpgradeableToken Contract:

```
bool canUpgrade_ = true;
```

These variables will be unchanged. So, please make it constant. It will save some gas.

Recommendation: Declare those variables as constant. Just put a constant keyword.

(7) Centralization Risk:

```
  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;
  }
```

The hasMintPermission can mint unlimited tokens.

Recommendation: We suggest carefully managing these account's private keys to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices.

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 ReleaseAgent can release tokens before transferable.

UpgradeableToken.sol

- setUpgradeAgent: An upgrade agent address is updated by the upgrade Master.
- setUpgradeMaster: An upgrade master address be updated by the current upgradeMaster.

ReleasableToken.sol

- setReleaseAgent: ReleaseAgent address can be updated by the owner.
- setTransferAgent: The owner can allow a particular address (a crowd sale contract) to transfer tokens despite the lock-up period.
- releaseTokenTransfer: ReleaseAgent can release the tokens to the wild.

MintableToken.sol

- mint: The owner can mint new tokens.
- finishMinting: The owner can stop minting new tokens.

Ownable.sol

- renounceOwnership: Renouncing ownership will leave the contract without an owner.
- 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 in 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. but those 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 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, and 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).

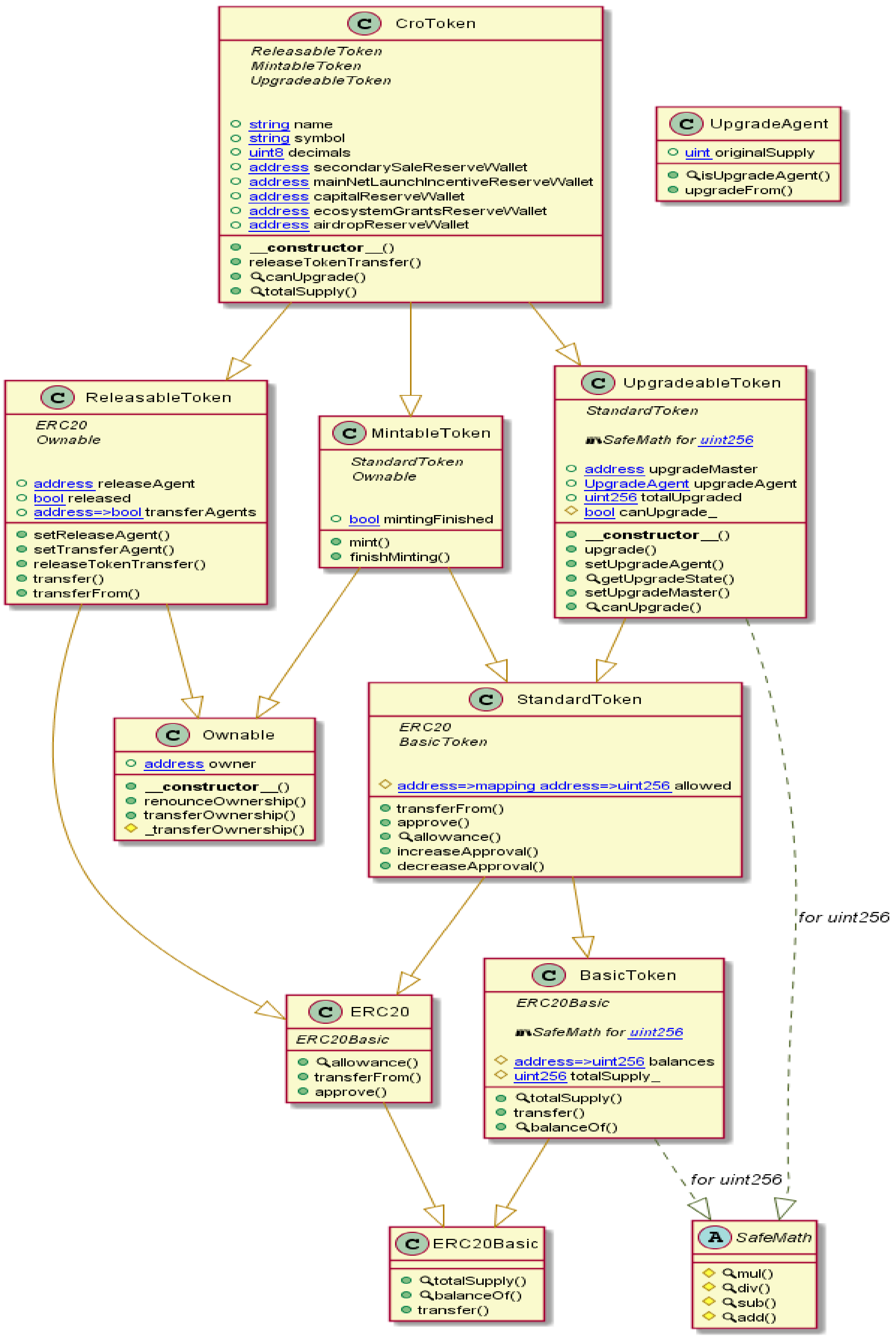
Due to the fact that the total number of test cases is unlimited, the audit makes no statements or warranties on the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. While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



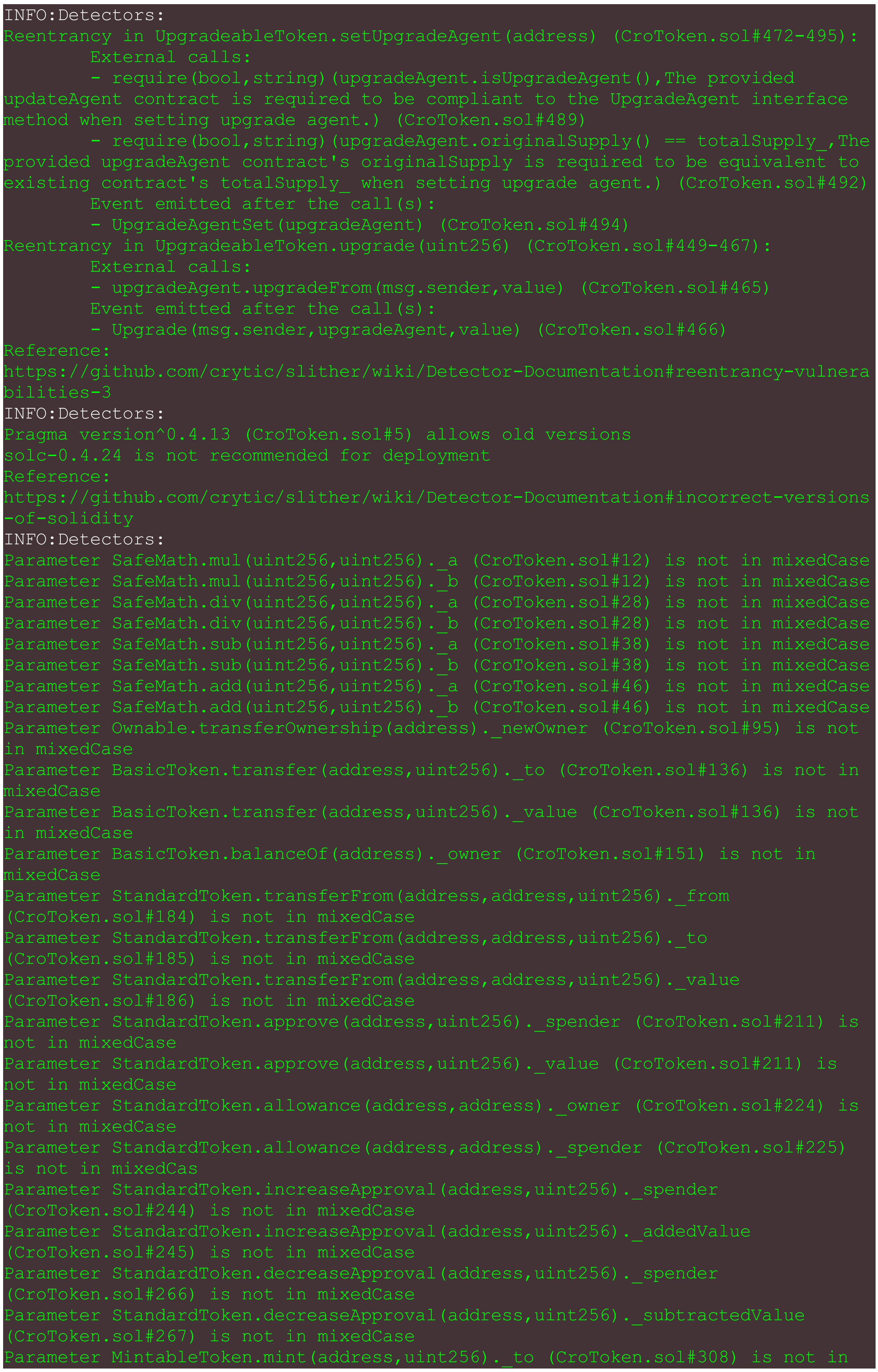
Slither Results Log

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.

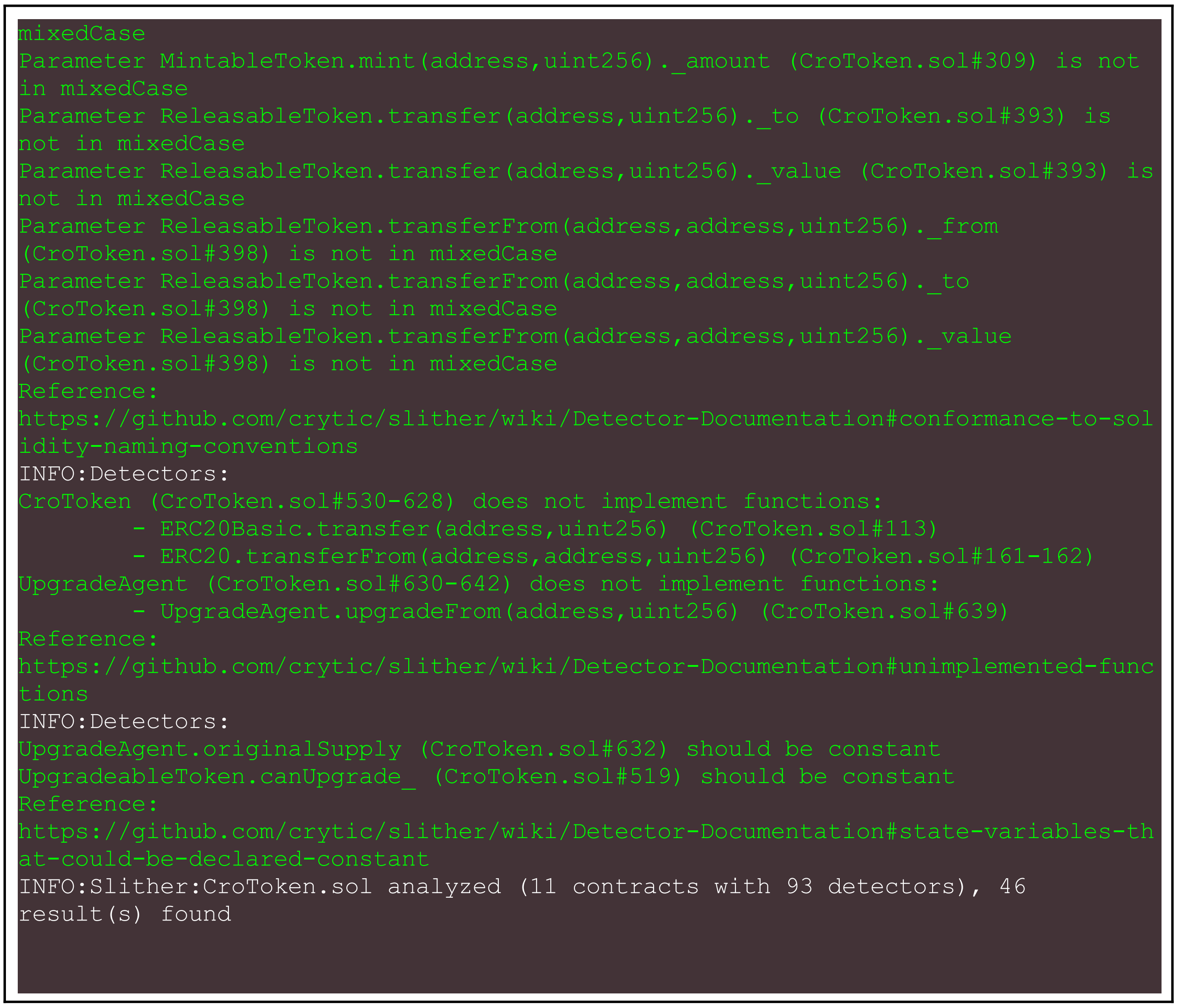
We did the analysis of the project altogether. Below are the results.

CroToken.sol

```
INFO:Detectors:
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-acc
ess-control
INFO:Detectors:
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,ad
dress,address)._secondarySaleReserveWallet (CroToken.sol#558) lacks a zero-check
on :
                - secondarySaleReserveWallet = _secondarySaleReserveWallet
(CroToken.sol#576)
CroToken.constructor(string,string,uint256,uint8,bool,address,address,address,ad
dress,address)._mainNetLaunchIncentiveReserveWallet (CroToken.sol#559) lacks a
zero-check on :
                - mainNetLaunchIncentiveReserveWallet =
_mainNetLaunchIncentiveReserveWallet (CroToken.sol#577)
CroToken.constructor(string,string,uint256,uint8,bool,address,address,address,ad
dress,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,ad
dress,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,ad
dress,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-addre
ss-validation
```



```
INFO:Detectors:
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 originalSupply 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-vulnera
bilities-3
INFO:Detectors:
Pragma version^0.4.13 (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
INFO:Detectors:
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 mixedCas
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
```

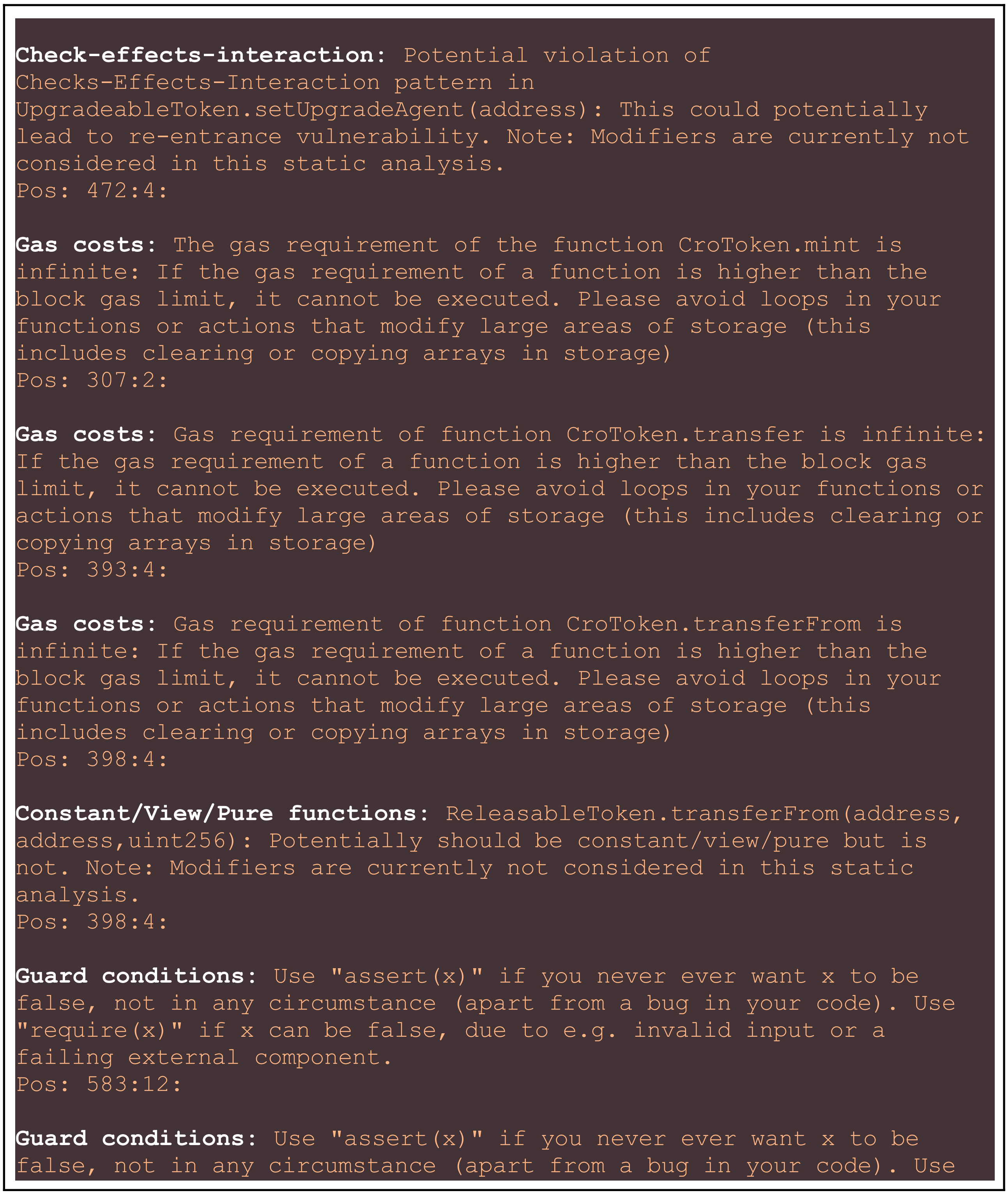


```
mixedCase
Parameter MintableToken.mint(address,uint256)._amount (CroToken.sol#309) is not
in mixedCase
Parameter ReleasableToken.transfer(address,uint256)._to (CroToken.sol#393) is
not in mixedCase
Parameter ReleasableToken.transfer(address,uint256)._value (CroToken.sol#393) is
not in mixedCase
Parameter 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-sol
idity-naming-conventions
INFO:Detectors:
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-func
tions
INFO:Detectors:
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-th
at-could-be-declared-constant
INFO:Slither:CroToken.sol analyzed (11 contracts with 93 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.setUpgradeAgent(address): This could potentially
lead to re-entrance vulnerability. Note: Modifiers are currently not
considered in this static analysis.
Pos: 472:4:

Gas costs: The gas requirement of the 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:

Gas costs: Gas requirement of function CroToken.transfer 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: 393:4:

Gas costs: Gas requirement of function CroToken.transferFrom 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: 398:4:

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

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.
Pos: 583:12:

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
```

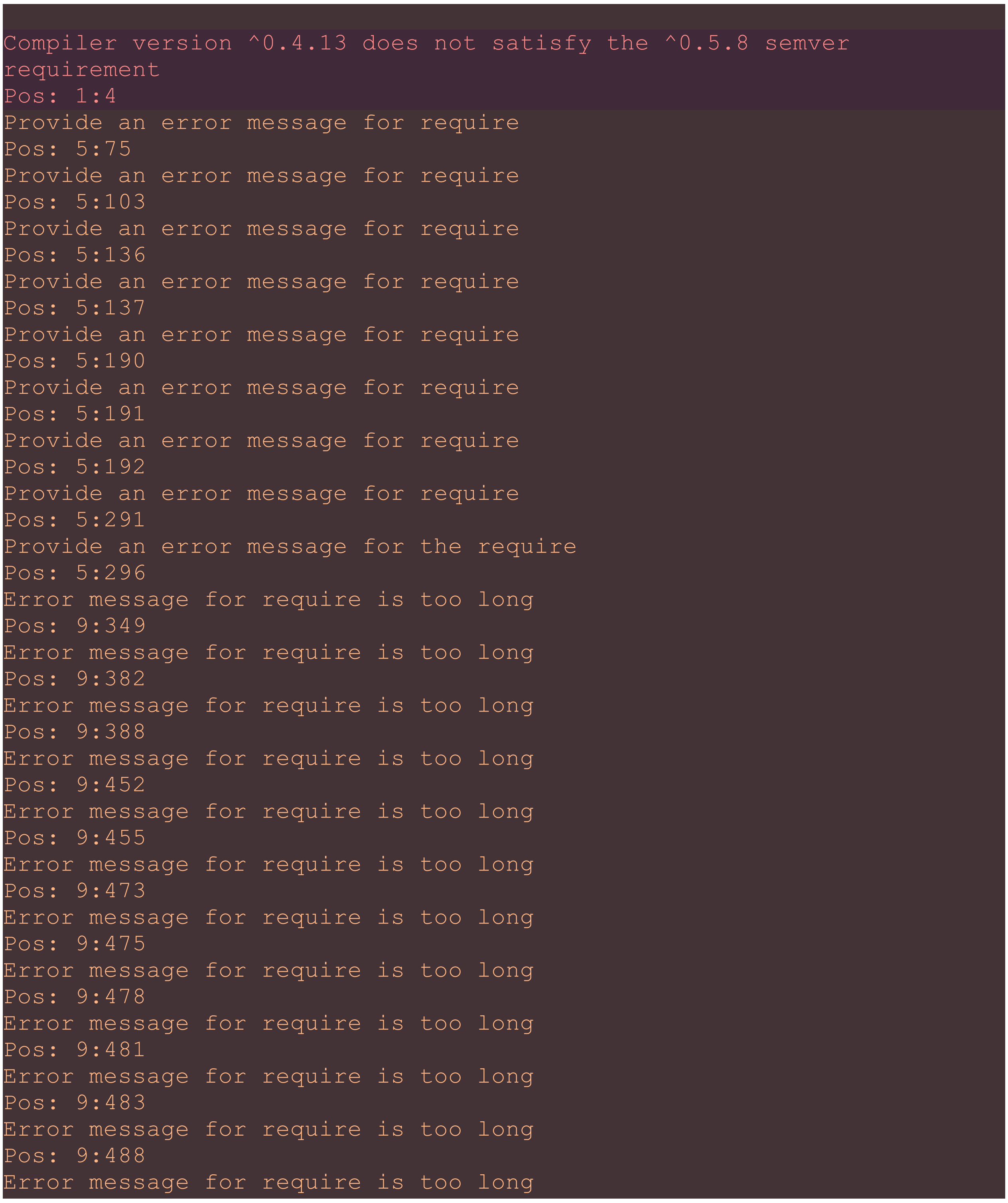
```
"require(x)" if x can be false, due to e.g. invalid input or a
failing external component.
Pos: 603:12:

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 the division of (only)
literal values since those yield rational constants.
Pos: 32:11:
```

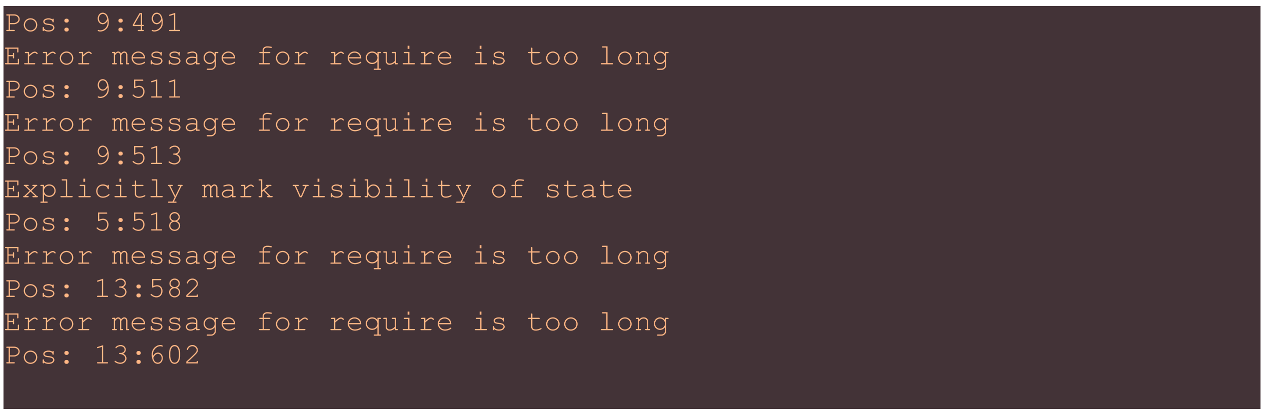
Solhint Linter

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 the 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.