Ether Authority
etherauthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project: Coinbase Wrapped Staked ETH
Website: coinbase.com
Platform: Base Chain Network
Language: Solidity
Date: June 4th, 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.

Email: audit@EtherAuthority.io

Introduction

As part of EtherAuthority's community smart contracts audit initiatives, the Coinbase Wrapped Staked ETH smart contract from coinbase.com 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 June 4th, 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

Website Details

coinbase

Coinbase is a trusted platform for buying, selling, and managing various cryptocurrencies, including Bitcoin, Ethereum, and Dogecoin. It offers tools for individuals, businesses, and developers, including advanced trading options, crypto wallets, and institutional solutions. Users can earn rewards, access staking, and participate in learning programs. Coinbase prioritizes security with best-in-class storage solutions. The platform also provides educational resources on crypto basics and market updates.

Code Details

- The `UpgradeableOptimismMintableERC20` contract is an upgradeable version of the `OptimismMintableERC20` contract, which enables the minting and burning of tokens by a bridge contract. This version integrates OpenZeppelin's upgradeable contracts to facilitate contract upgrades without disrupting the contract state.
- The `UpgradeableOptimismMintableERC20` contract is designed to facilitate token minting and burning through a bridge in an upgradeable manner. By incorporating OpenZeppelin's upgradeable contracts and providing a structured way to add future upgrades, this contract ensures both functionality and maintainability in a dynamic blockchain environment.

Audit scope

Name	**Code Review and Security Analysis Report for Coinbase Wrapped Staked ETH Smart Contract**
Platform	**Base Chain Network**
Language	**Solidity**
File	UpgradeableOptimismMintableERC20.sol
Smart Contract Code	0x07a71b9b835c9eba242836704d17da0953324e1f
Audit Date	June 4th,2024
Audit Result	**Passed**

Code Audit History

0 Total Findings	0 Critical	0 High	0 Medium	0 Low	0 Informational

Severity Definitions

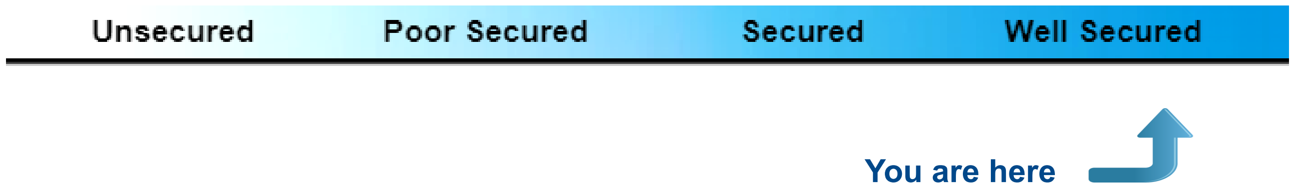
0	**Critical**	Critical vulnerabilities are usually straightforward to exploit and can lead to token loss etc.
0	**High**	High-level vulnerabilities are difficult to exploit; however, they also have significant impact on smart contract execution, e.g. Public access is crucial.
0	**Medium**	Medium-level vulnerabilities are important to fix; however, they can't lead to tokens lose
0	**Low**	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets, that can't have a significant impact on execution
0	**Lowest / Informational / Best Practice**	Lowest-level vulnerabilities, code style violations, and info statements can't affect smart contract execution and can be ignored.

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Key Modifications 1. Inheritance from Upgradeable Contracts: ● Uses `ERC20Upgradeable` instead of `ERC20` from OpenZeppelin to support upgradeability. ● Uses `IERC165Upgradeable` instead of `IERC165` for interface support. 2. New Functions and Modifiers: ● `initialize` function: Initializes the contract, setting the ERC20 name and symbol. ● `onlyBridge` modifier: Restricts minting and burning functions to be callable only by the bridge. 3. Storage and State Management: ● Added an immutable variable `_DECIMALS` for decimal representation. ● Added a storage gap (`__gap`) to facilitate future upgrades without shifting storage slots. 4. Interface Support: ● The `supportsInterface` method is marked as `virtual` to allow overriding by inheriting contracts. 5. Legacy Support: ● Added legacy getter functions (`l1Token`, `l2Bridge`, `remoteToken`, `bridge`) to maintain compatibility with existing interfaces.	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, the Customer`s solidity-based smart contracts are **"Well 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 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, 0 low, and 0 very low-level issue.

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	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	Passed
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 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?	Yes
Is it used Open Source?	No
External Call Risk?	No
Balance Modifiable?	No
Can Take Ownership?	No
Ownership Renounce?	No
Hidden Owner?	Not Detected
Self Destruction?	Not Detected
Auditor Confidence	High

Overall Audit Result: PASSED

Code Quality

This audit scope has 1 smart contract. Smart contracts contain Libraries, Smart contracts, inherits, and Interfaces. This is a compact and well-written smart contract.

The libraries in Coinbase Wrapped Staked ETH 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 Coinbase Wrapped Staked ETH.

The EtherAuthority team has not provided 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 Coinbase Wrapped Staked ETH smart contract code in the form of a basescan 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

UpgradeableOptimismMintableERC20.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	onlyBridge	modifier	Passed	No Issue
3	initialize	external	initializer	No Issue
4	mint	external	access only Bridge	No Issue
5	burn	external	access only Bridge	No Issue
6	supportsInterface	external	Passed	No Issue
7	l1Token	read	Passed	No Issue
8	l2Bridge	read	Passed	No Issue
9	remoteToken	read	Passed	No Issue
10	bridge	read	Passed	No Issue
11	decimals	read	Passed	No Issue
12	version	read	Passed	No Issue
13	__ERC20_init	internal	access only Initializing	No Issue
14	__ERC20_init_unchained	internal	access only Initializing	No Issue
15	name	read	Passed	No Issue
16	symbol	read	Passed	No Issue
17	decimals	read	Passed	No Issue
18	totalSupply	read	Passed	No Issue
19	balanceOf	read	Passed	No Issue
20	transfer	write	Passed	No Issue
21	allowance	read	Passed	No Issue
22	approve	write	Passed	No Issue
23	transferFrom	write	Passed	No Issue
24	increaseAllowance	write	Passed	No Issue
25	decreaseAllowance	write	Passed	No Issue
26	_transfer	internal	Passed	No Issue
27	_mint	internal	Passed	No Issue
28	_burn	internal	Passed	No Issue
29	_approve	internal	Passed	No Issue
30	_spendAllowance	internal	Passed	No Issue
31	_beforeTokenTransfer	internal	Passed	No Issue
32	_afterTokenTransfer	internal	Passed	No Issue

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:

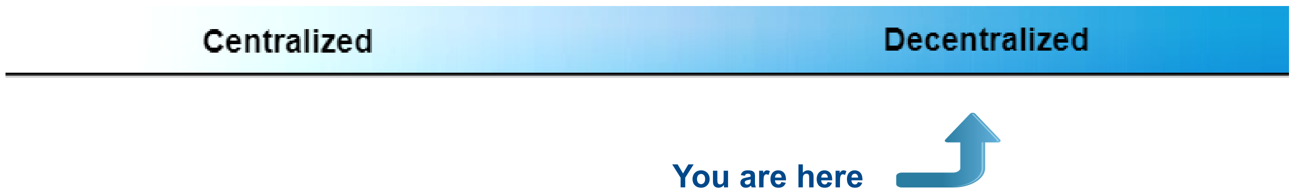
No Very-Low-severity vulnerabilities were found.

Email: audit@EtherAuthority.io

Centralization

The Coinbase Wrapped Staked ETH (cbETH) 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 a basescan web link. And we have used all possible tests based on given objects as files. We observed no issue in the smart contracts. 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 **"Well Secured"**.

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of the systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review:

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis:

Our audit techniques included manual code analysis, user interface interaction, and white box penetration testing. We look at the project's website to get a high-level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

Documenting Results:

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyze the feasibility of an attack in a live system.

Suggested Solutions:

We search for immediate mitigations that live deployments can take, and finally, we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

Email: audit@EtherAuthority.io

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).

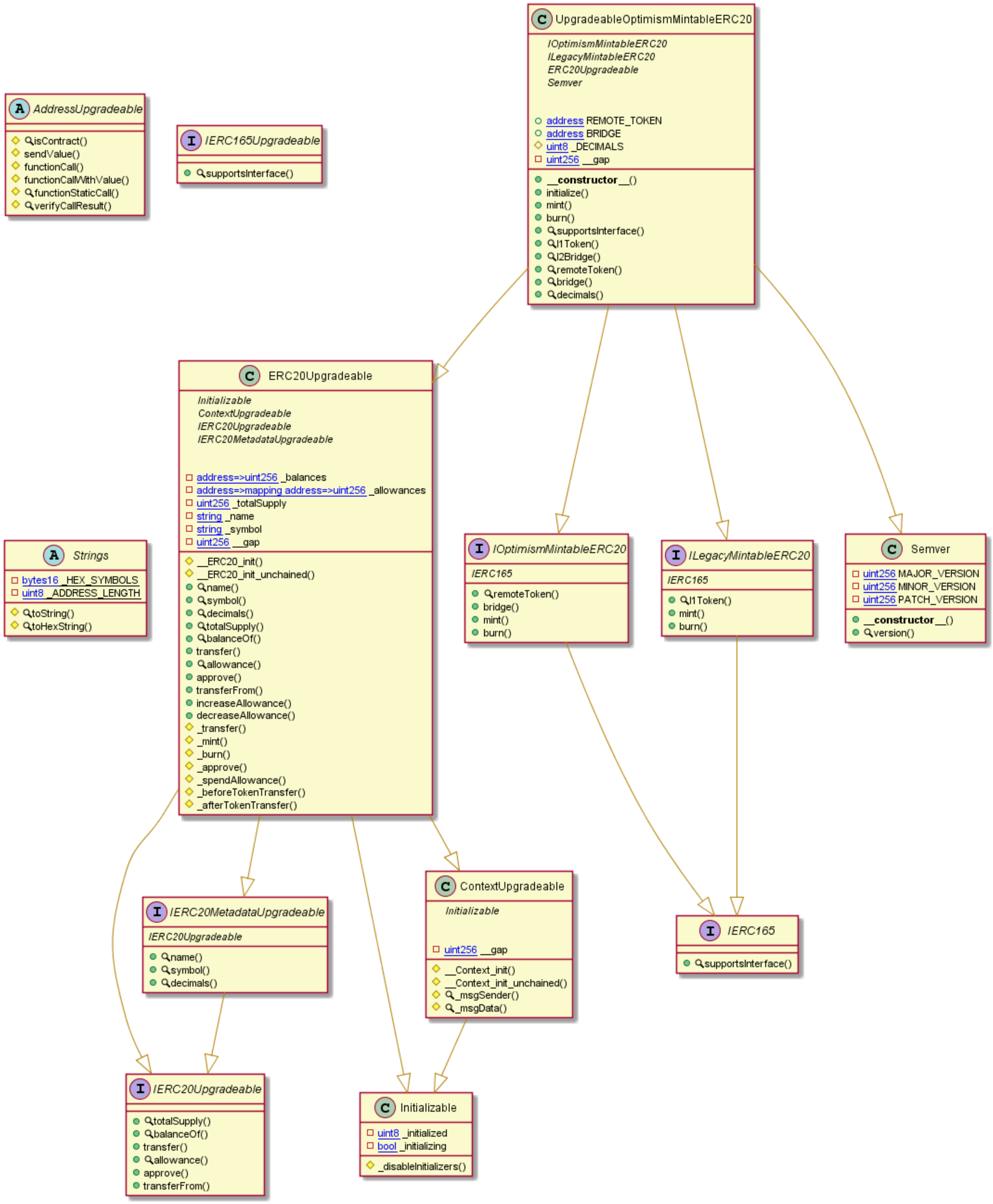
Due to the fact that the total number of test cases is unlimited, the audit makes no statements or warranties on the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. While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



Email: audit@EtherAuthority.io

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.

UpgradeableOptimismMintableERC20.sol

```
INFO:Detectors:
UpgradeableOptimismMintableERC20.initialize(string,string)._name
(UpgradeableOptimismMintableERC20.sol#1115) shadows:
	- ERC20Upgradeable._name (UpgradeableOptimismMintableERC20.sol#512) (state
variable)
UpgradeableOptimismMintableERC20.initialize(string,string)._symbol
(UpgradeableOptimismMintableERC20.sol#1116) shadows:
	- ERC20Upgradeable._symbol (UpgradeableOptimismMintableERC20.sol#513) (state
variable)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
UpgradeableOptimismMintableERC20.constructor(address,address,uint8)._remoteToken
(UpgradeableOptimismMintableERC20.sol#1101) lacks a zero-check on :
		- REMOTE_TOKEN = _remoteToken (UpgradeableOptimismMintableERC20.sol#1104)
UpgradeableOptimismMintableERC20.constructor(address,address,uint8)._bridge
(UpgradeableOptimismMintableERC20.sol#1100) lacks a zero-check on :
		- BRIDGE = _bridge (UpgradeableOptimismMintableERC20.sol#1105)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Pragma version0.8.15 (UpgradeableOptimismMintableERC20.sol#11) allows old versions
solc-0.8.15 is not recommended for deployment
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Variable Semver.MAJOR_VERSION (UpgradeableOptimismMintableERC20.sol#1016) is not in
mixedCase
Variable Semver.MINOR_VERSION (UpgradeableOptimismMintableERC20.sol#1019) is not in
mixedCase
Variable Semver.PATCH_VERSION (UpgradeableOptimismMintableERC20.sol#1022) is not in
mixedCase
```


Email: audit@EtherAuthority.io

```
Parameter UpgradeableOptimismMintableERC20.initialize(string,string)._name
(UpgradeableOptimismMintableERC20.sol#1115) is not in mixedCase
Parameter UpgradeableOptimismMintableERC20.initialize(string,string)._symbol
(UpgradeableOptimismMintableERC20.sol#1116) is not in mixedCase
Parameter UpgradeableOptimismMintableERC20.mint(address,uint256)._to
(UpgradeableOptimismMintableERC20.sol#1124) is not in mixedCase
Parameter UpgradeableOptimismMintableERC20.mint(address,uint256)._amount
(UpgradeableOptimismMintableERC20.sol#1124) is not in mixedCase
Parameter UpgradeableOptimismMintableERC20.burn(address,uint256)._from
(UpgradeableOptimismMintableERC20.sol#1137) is not in mixedCase
Parameter UpgradeableOptimismMintableERC20.burn(address,uint256)._amount
(UpgradeableOptimismMintableERC20.sol#1137) is not in mixedCase
Parameter UpgradeableOptimismMintableERC20.supportsInterface(bytes4)._interfaceId
(UpgradeableOptimismMintableERC20.sol#1150) is not in mixedCase
Variable UpgradeableOptimismMintableERC20.REMOTE_TOKEN
(UpgradeableOptimismMintableERC20.sol#1071) is not in mixedCase
Variable UpgradeableOptimismMintableERC20.BRIDGE
(UpgradeableOptimismMintableERC20.sol#1074) is not in mixedCase
Variable UpgradeableOptimismMintableERC20._DECIMALS
(UpgradeableOptimismMintableERC20.sol#1077) is not in mixedCase
Variable UpgradeableOptimismMintableERC20.__gap
(UpgradeableOptimismMintableERC20.sol#1196) is not in mixedCase
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-c
onventions
INFO:Detectors:
Variable UpgradeableOptimismMintableERC20.REMOTE_TOKEN
(UpgradeableOptimismMintableERC20.sol#1071) is too similar to
UpgradeableOptimismMintableERC20.constructor(address,address,uint8)._remoteToken
(UpgradeableOptimismMintableERC20.sol#1101)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-too-similar
INFO:Detectors:
UpgradeableOptimismMintableERC20.__gap (UpgradeableOptimismMintableERC20.sol#1196)
is never used in UpgradeableOptimismMintableERC20
(UpgradeableOptimismMintableERC20.sol#1069-1198)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variable
INFO:Slither:UpgradeableOptimismMintableERC20.sol analyzed (13 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.

UpgradeableOptimismMintableERC20.sol

Check-effects-interaction:
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.
Pos: 305:16:

Gas costs:
Gas requirement of function UpgradeableOptimismMintableERC20.l2Bridge 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: 1167:4:

Constant/View/Pure functions:
UpgradeableOptimismMintableERC20.initialize(string,string) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.
Pos: 1114:4:

Similar variable names:
Semver.version() : Variables have very similar names "MAJOR_VERSION" and "MINOR_VERSION". Note: Modifiers are currently not considered by this static analysis.
Pos: 1045:37:

No return:
ILegacyMintableERC20.l1Token(): Defines a return type but never explicitly returns a value.
Pos: 932:4:

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: 1091:8:

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.

UpgradeableOptimismMintableERC20.sol

```
Compiler version 0.8.15 does not satisfy the ^0.5.8 semver requirement
Pos: 1:10
Error message for require is too long
Pos: 9:182
Error message for require is too long
Pos: 9:252
Error message for require is too long
Pos: 9:280
Avoid to use inline assembly. It is acceptable only in rare cases
Pos: 17:304
Error message for require is too long
Pos: 9:387
Error message for require is too long
Pos: 9:415
Error message for require is too long
Pos: 9:428
Error message for require is too long
Pos: 9:439
Function name must be in mixedCase
Pos: 5:458
Code contains empty blocks
Pos: 57:458
Function name must be in mixedCase
Pos: 5:461
Code contains empty blocks
Pos: 67:461
Function name must be in mixedCase
Pos: 5:523
Function name must be in mixedCase
Pos: 5:527
Error message for require is too long
Pos: 9:677
Error message for require is too long
Pos: 9:704
Error message for require is too long
Pos: 9:705
```


Email: audit@EtherAuthority.io

```
Error message for require is too long
Pos: 9:710
Error message for require is too long
Pos: 9:754
Error message for require is too long
Pos: 9:759
Error message for require is too long
Pos: 9:788
Error message for require is too long
Pos: 9:789
Code contains empty blocks
Pos: 24:835
Code contains empty blocks
Pos: 24:855
Variable name must be in mixedCase
Pos: 5:1015
Variable name must be in mixedCase
Pos: 5:1018
Variable name must be in mixedCase
Pos: 5:1021
Explicitly mark visibility in function (Set ignoreConstructors to true if using solidity >=0.7.0)
Pos: 5:1026
Variable name must be in mixedCase
Pos: 5:1070
Variable name must be in mixedCase
Pos: 5:1073
Variable name must be in mixedCase
Pos: 5:1076
Error message for require is too long
Pos: 9:1090
Explicitly mark visibility in function (Set ignoreConstructors to true if using solidity >=0.7.0)
Pos: 5:1098
```

Software analysis result:

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

Email: audit@EtherAuthority.io