Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project:	Mantle Token
Website:	mantle.xyz
Platform:	Ethereum
Language:	Solidity
Date:	May 14th, 2024

Table of contents

Introduction 4

Project Background 4

Audit Scope 5

Claimed Smart Contract Features 6

Audit Summary 7

Technical Quick Stats 8

Business Risk Analysis 9

Code Quality 10

Documentation 10

Use of Dependencies 10

AS-IS overview 11

Severity Definitions 14

Audit Findings 15

Conclusion 17

Our Methodology 18

Disclaimers 20

Appendix

- Code Flow Diagram 21
- Slither Results Log 22
- Solidity static analysis 24
- Solhint Linter 26

Email: audit@EtherAuthority.io

`

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 contract audit initiatives, the smart contract of Mantle Token from mantle.xyz was audited. The audit has been performed using manual analysis as well as using automated software tools. This report presents all the findings regarding the audit performed on May 14th, 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 Solidity smart contract implements an ERC20 token with additional functionalities like minting, burning, and governance using OpenZeppelin's upgradeable contracts. The contract, `L1MantleToken`, is designed to allow minting with specific constraints and includes governance features such as votes. Here is a detailed breakdown of its structure and functionality:
- The contract imports several modules from OpenZeppelin's library:
    - ERC20Upgradeable: The basic ERC20 token functionality.
    - ERC20BurnableUpgradeable: Adds burn functionality.
    - OwnableUpgradeable: Adds ownership and access control.
    - ERC20PermitUpgradeable: Adds EIP-2612 permits for gas-less approvals.
    - ERC20VotesUpgradeable: Adds voting functionality.
    - Initializable: Support for upgradeable contracts.
- The `L1MantleToken` contract is a comprehensive implementation of an ERC20 token with additional features for minting, burning, and governance. It ensures strict control over minting operations to prevent inflation and includes upgradeable features to allow future modifications.

Audit scope

Name	Code Review and Security Analysis Report for Mantle (MNT) Token Smart Contract
Platform	**Ethereum**
File	L1MantleToken.sol
Smart Contract Code	0xcd368c1d80120b0dd92447c87eb570154f8e685c
Audit Date	May 14th, 2024

Email: audit@EtherAuthority.io

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Tokenomics: • Name: Mantle • Symbol: MNT • Decimals: 18 • Min Mint Interval: 365 Days • Mint Cap Denominator: 10,000 • Mint Cap Max Numerator: 200	**YES, This is valid.**
Ownership control: • Allows the owner to mint new tokens and increase this token's total supply. • The `setMintCapNumerator` function allows the owner to set the `mintCapNumerator`. It ensures the new numerator does not exceed the maximum allowed value and emits a `MintCapNumeratorChanged` event. • 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.**

Audit Summary

According to the standard audit assessment, the Customer`s solidity-based smart contract is **“Secured”**. Also, this contract contains owner control, which does not make it 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, 0 low, and 1 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	Passed
	Human/contract checks bypass	Passed
	Random number generation/use vulnerability	N/A
	Fallback function misuse	Passed
	Race condition	Passed
	Logical vulnerability	Passed
	Features claimed	Passed
	Other programming issues	Moderated
Code Specification	Function visibility not explicitly declared	Passed
	Var. storage location not explicitly declared	Passed
	Use keywords/functions to be deprecated	Passed
	Unused code	Passed
Gas Optimization	“Out of Gas” Issue	Passed
	High consumption ‘for/while’ loop	Passed
	High consumption ‘storage’ storage	Passed
	Assert() misuse	Passed
Business Risk	The maximum limit for mintage is not set	Passed
	“Short Address” Attack	Passed
	“Double Spend” Attack	Passed

Overall Audit Result: **PASSED**

Email: audit@EtherAuthority.io

Business Risk Analysis

Category	Result
Buy Tax	0%
Sell Tax	0%
Cannot Buy	No
Cannot Sell	No
Max Tax	0%
Modify Tax	Not Detected
Fee Check	No
Is Honeypot	Not Detected
Trading Cooldown	Not Detected
Can Pause Trade?	No
Pause Transfer?	Not Detected
Max Tax?	No
Is it Anti-whale?	Not Detected
Is Anti-bot?	Not Detected
Is it a Blacklist?	Not Detected
Blacklist Check	No
Can Mint?	Yes
Is it a Proxy?	Yes
Can Take Ownership?	Yes
Hidden Owner?	No
Self Destruction?	Not Detected
Auditor Confidence	High

Overall Audit Result: **PASSED**

Email: audit@EtherAuthority.io

Code Quality

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

The libraries in Mantle 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 contract in the Mantle 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 Mantle 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

L1MantleToken.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	__ERC20Votes_init	internal	access only Initializing	No Issue
3	__ERC20Votes_init_unchained	internal	access only Initializing	No Issue
4	checkpoints	read	Passed	No Issue
5	numCheckpoints	read	Passed	No Issue
6	delegates	read	Passed	No Issue
7	getVotes	read	Passed	No Issue
8	getPastVotes	read	Passed	No Issue
9	getPastTotalSupply	read	Passed	No Issue
10	_checkpointsLookup	read	Passed	No Issue
11	delegate	write	Passed	No Issue
12	delegateBySig	write	Passed	No Issue
13	_maxSupply	internal	Passed	No Issue
14	_mint	internal	Passed	No Issue
15	_burn	internal	Passed	No Issue
16	_afterTokenTransfer	internal	Passed	No Issue
17	_delegate	internal	Passed	No Issue
18	_moveVotingPower	write	Passed	No Issue
19	_writeCheckpoint	write	Passed	No Issue
20	_add	write	Passed	No Issue
21	_subtract	write	Passed	No Issue
22	_unsafeAccess	write	Passed	No Issue
23	_useNonce	internal	Passed	No Issue
24	DOMAIN_SEPARATOR	external	Passed	No Issue
25	nonces	read	Passed	No Issue
26	permit	write	Passed	No Issue
27	__ERC20Permit_init_unchained	internal	access only Initializing	No Issue
28	__ERC20Permit_init	internal	access only Initializing	No Issue
29	__ERC20Burnable_init	internal	access only Initializing	No Issue
30	__ERC20Burnable_init_unchained	internal	access only Initializing	No Issue
31	burn	write	Passed	No Issue
32	burnFrom	write	Passed	No Issue
33	__ERC20_init	internal	access only Initializing	No Issue

34	__ERC20_init_unchained	internal	access only Initializing	No Issue
35	name	read	Passed	No Issue
36	symbol	read	Passed	No Issue
37	decimals	read	Passed	No Issue
38	totalSupply	read	Passed	No Issue
39	balanceOf	read	Passed	No Issue
40	transfer	write	Passed	No Issue
41	allowance	read	Passed	No Issue
42	approve	write	Passed	No Issue
43	transferFrom	write	Passed	No Issue
44	increaseAllowance	write	Passed	No Issue
45	decreaseAllowance	write	Passed	No Issue
46	_transfer	internal	Passed	No Issue
47	_mint	internal	Passed	No Issue
48	_burn	internal	Passed	No Issue
49	_approve	internal	Passed	No Issue
50	_spendAllowance	internal	Passed	No Issue
51	_beforeTokenTransfer	internal	Passed	No Issue
52	_afterTokenTransfer	internal	Passed	No Issue
53	__Ownable_init	internal	access only Initializing	No Issue
54	__Ownable_init_unchained	internal	access only Initializing	No Issue
55	onlyOwner	modifier	Passed	No Issue
56	owner	read	Passed	No Issue
57	_checkOwner	internal	Passed	No Issue
58	renounceOwnership	write	Passed	No Issue
59	transferOwnership	write	Passed	No Issue
60	_transferOwnership	internal	Passed	No Issue
61	__EIP712_init	internal	access only Initializing	No Issue
62	__EIP712_init_unchained	internal	access only Initializing	No Issue
63	_domainSeparatorV4	internal	Passed	No Issue
64	_buildDomainSeparator	read	Passed	No Issue
65	_hashTypedDataV4	internal	Passed	No Issue
66	_EIP712NameHash	internal	Passed	No Issue
67	_EIP712VersionHash	internal	Passed	No Issue
68	initialize	write	initializer	No Issue
69	mint	write	Centralized Ownership and Privileges Management	Refer Audit Findings
70	setMintCapNumerator	write	Centralized Ownership and Privileges Management	Refer Audit Findings

71	_afterTokenTransfer	internal	Passed	No Issue
72	_mint	internal	Passed	No Issue
73	_burn	internal	Passed	No Issue
74	initializer	modifier	Passed	No Issue
75	reinitializer	modifier	Passed	No Issue
76	onlyInitializing	modifier	Passed	No Issue
77	_disableInitializers	internal	Passed	No Issue
78	_getInitializedVersion	internal	Passed	No Issue
79	_isInitializing	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.

Email: audit@EtherAuthority.io

Audit Findings

Critical Severity

No Critical severity vulnerabilities were found.

High Severity

No high-severity vulnerabilities were found.

Medium

No Medium-severity vulnerabilities were found.

Low

No low-severity vulnerabilities were found.

Very Low / Informational / Best practices:

(1) Centralized Ownership and Privileges Management:

Some functions of this smart contract are only called by the owner.

L1MantleToken.sol

- mint
- setMintCapNumerator

Resolution: We suggest making your smart contract 100% decentralized.

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:

L1MantleToken.sol

- mint: Allows the owner to mint new tokens and increase this token's total supply.
- setMintCapNumerator: Mint Cap Numerator value can be set by the owner.

Ownable.sol

- renounceOwnership: Deleting ownership will leave the contract without an owner, removing any owner-only functionality.
- transferOwnership: Current owner can transfer ownership of the contract to a new account.

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 informational issue in the smart contract. And these issues are not critical. So, **it's good to go for the production**.

Since possible test cases can be unlimited for such smart contracts protocol, we provide no such guarantee of future outcomes. We have used all the latest static tools and manual observations to cover the maximum possible test cases to scan everything.

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. Smart Contract's high-level description of functionality was presented in the As-is overview section of the report.

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

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

Our Methodology

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

Manual Code Review:

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

Vulnerability Analysis:

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

Documenting Results:

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

Suggested Solutions:

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

Disclaimers

EtherAuthority.io Disclaimer

EtherAuthority team has analyzed this smart contract in accordance with the best industry practices at the date of this report, in relation to: cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment and functionality (performing the intended functions).

Due to the fact that the total number of test cases 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.

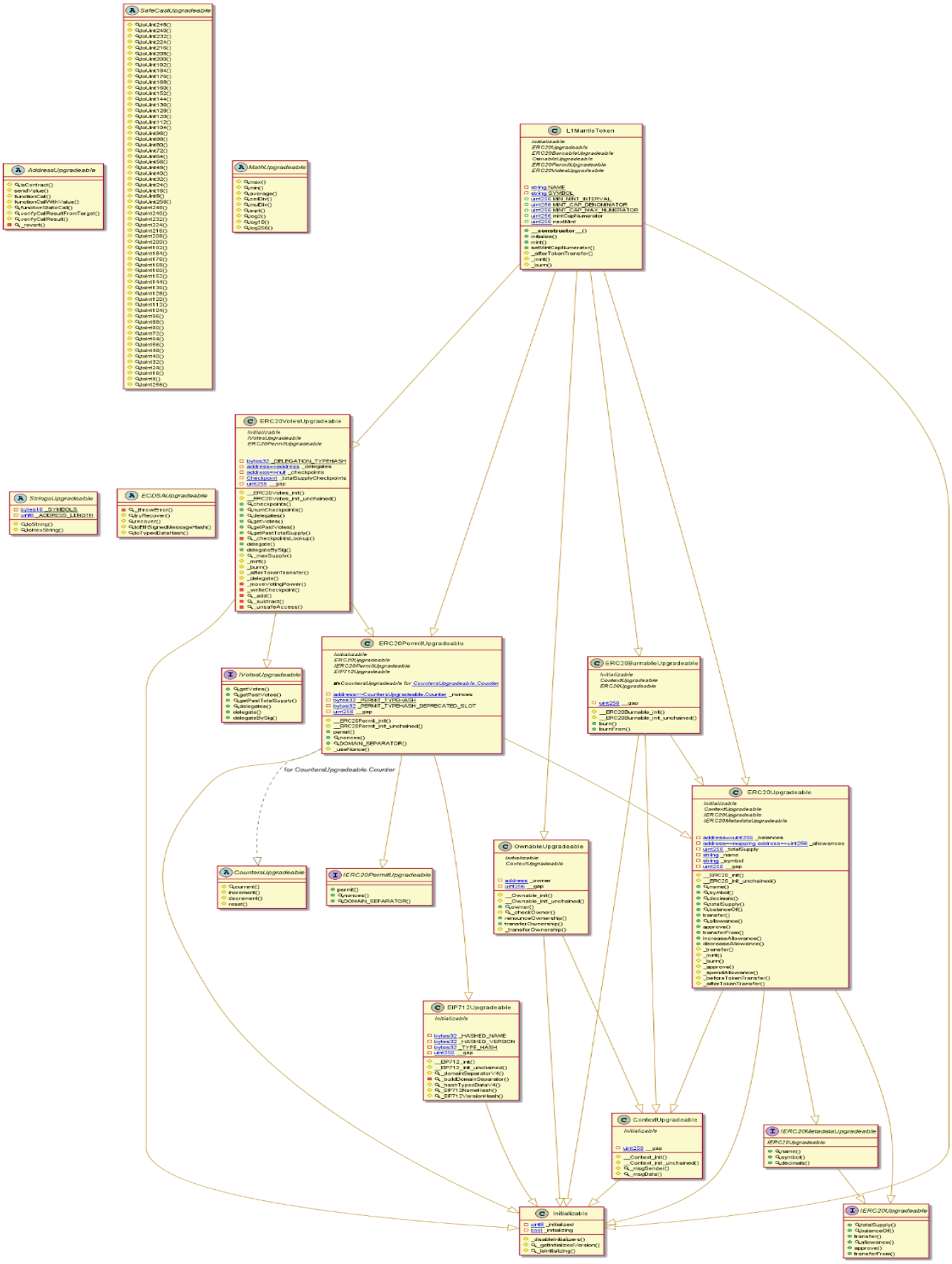
Technical Disclaimer

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 explicit security of the audited smart contracts.

Appendix

Code Flow Diagram - Mantle Token

L1Mantle Token Diagram



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.

Slither Log >> L1MantleToken.sol

```
ERC20PermitUpgradeable.__ERC20Permit_init(string).name (L1MantleToken.sol#2903) shadows:
        - ERC20UpgradeableName() (L1MantleToken.sol#2505-2507) (function)
        - IERC20MetadataUpgradeable.name() (L1MantleToken.sol#1916) (function)
L1MantleToken.initialize(uint256,address)._owner (L1MantleToken.sol#3300) shadows:
        - OwnableUpgradeable._owner (L1MantleToken.sol#2399) (state variable)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing

Variable 'ERC20VotesUpgradeable._moveVotingPower(address,address,uint256).oldWeight (L1MantleToken.sol#3175)' in ERC20VotesUpgr
adeable._moveVotingPower(address,address,uint256) (L1MantleToken.sol#3168-3184) potentially used before declaration: (oldWeight
,newWeight) = _writeCheckpoint(_checkpoints[dst],_add,amount) (L1MantleToken.sol#3180)
Variable 'ERC20VotesUpgradeable._moveVotingPower(address,address,uint256).newWeight (L1MantleToken.sol#3175)' in ERC20VotesUpgr
adeable._moveVotingPower(address,address,uint256) (L1MantleToken.sol#3168-3184) potentially used before declaration: (oldWeight
,newWeight) = _writeCheckpoint(_checkpoints[dst],_add,amount) (L1MantleToken.sol#3180)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#pre-declaration-usage-of-local-variables

ERC20PermitUpgradeable.permit(address,address,uint256,uint256,uint8,bytes32,bytes32) (L1MantleToken.sol#2912-2931) uses timesta
mp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp <= deadline,ERC20Permit: expired deadline) (L1MantleToken.sol#2921)
ERC20VotesUpgradeable.delegateBySig(address,uint256,uint256,uint8,bytes32,bytes32) (L1MantleToken.sol#3093-3110) uses timestamp
 for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp <= expiry,ERC20Votes: signature expired) (L1MantleToken.sol#3101)
L1MantleToken.mint(address,uint256) (L1MantleToken.sol#3325-3334) uses timestamp for comparisons
        Dangerous comparisons:
        - block.timestamp < nextMint (L1MantleToken.sol#3330)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
```

```
AddressUpgradeable._revert(bytes,string) (L1MantleToken.sol#206-218) uses assembly
        - INLINE ASM (L1MantleToken.sol#211-214)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (L1MantleToken.sol#1484-1564) uses assembly
        - INLINE ASM (L1MantleToken.sol#1495-1499)
        - INLINE ASM (L1MantleToken.sol#1515-1522)
        - INLINE ASM (L1MantleToken.sol#1529-1538)
StringsUpgradeable.toString(uint256) (L1MantleToken.sol#1783-1803) uses assembly
        - INLINE ASM (L1MantleToken.sol#1789-1791)
        - INLINE ASM (L1MantleToken.sol#1795-1797)
ECDSAUpgradeable.tryRecover(bytes32,bytes) (L1MantleToken.sol#2002-2019) uses assembly
        - INLINE ASM (L1MantleToken.sol#2010-2014)
ERC20VotesUpgradeable._unsafeAccess(ERC20VotesUpgradeable.Checkpoint[],uint256) (L1MantleToken.sol#3216-3221) uses assembly
        - INLINE ASM (L1MantleToken.sol#3217-3220)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
```

```
SafeCastUpgradeable.toUint96(uint256) (L1MantleToken.sol#605-608) is never used and should be removed
StringsUpgradeable.toHexString(address) (L1MantleToken.sol#1832-1834) is never used and should be removed
StringsUpgradeable.toHexString(uint256) (L1MantleToken.sol#1808-1812) is never used and should be removed
StringsUpgradeable.toHexString(uint256,uint256) (L1MantleToken.sol#1817-1827) is never used and should be removed
StringsUpgradeable.toString(uint256) (L1MantleToken.sol#1783-1803) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version^0.8.18 (L1MantleToken.sol#4) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7
.6/0.8.16
solc-0.8.18 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Low level call in AddressUpgradeable.sendValue(address,uint256) (L1MantleToken.sol#60-65):
        - (success) = recipient.call{value: amount}() (L1MantleToken.sol#63)
Low level call in AddressUpgradeable.functionCallWithValue(address,bytes,uint256,string) (L1MantleToken.sol#128-137):
        - (success,returndata) = target.call{value: value}(data) (L1MantleToken.sol#135)
Low level call in AddressUpgradeable.functionStaticCall(address,bytes,string) (L1MantleToken.sol#155-162):
        - (success,returndata) = target.staticcall(data) (L1MantleToken.sol#160)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

Function IERC20PermitUpgradeable.DOMAIN_SEPARATOR() (L1MantleToken.sol#266) is not in mixedCase
Function ContextUpgradeable.__Context_init() (L1MantleToken.sol#2283-2284) is not in mixedCase
Function ContextUpgradeable.__Context_init_unchained() (L1MantleToken.sol#2286-2287) is not in mixedCase
Variable ContextUpgradeable.__gap (L1MantleToken.sol#2301) is not in mixedCase
Function EIP712Upgradeable.__EIP712_init(string,string) (L1MantleToken.sol#2324-2326) is not in mixedCase
Function EIP712Upgradeable.__EIP712_init_unchained(string,string) (L1MantleToken.sol#2328-2333) is not in mixedCase
Function EIP712Upgradeable._EIP712NameHash() (L1MantleToken.sol#2375-2377) is not in mixedCase
```

```
Function EIP712Upgradeable._EIP712NameHash() (L1MantleToken.sol#2375-2377) is not in mixedCase
Function EIP712Upgradeable._EIP712VersionHash() (L1MantleToken.sol#2385-2387) is not in mixedCase
Variable EIP712Upgradeable._HASHED_NAME (L1MantleToken.sol#2306) is not in mixedCase
Variable EIP712Upgradeable._HASHED_VERSION (L1MantleToken.sol#2307) is not in mixedCase
Variable EIP712Upgradeable.__gap (L1MantleToken.sol#2394) is not in mixedCase
Function OwnableUpgradeable.__Ownable_init() (L1MantleToken.sol#2406-2408) is not in mixedCase
Function OwnableUpgradeable.__Ownable_init_unchained() (L1MantleToken.sol#2410-2412) is not in mixedCase
Variable OwnableUpgradeable.__gap (L1MantleToken.sol#2471) is not in mixedCase
Function ERC20Upgradeable.__ERC20_init(string,string) (L1MantleToken.sol#2493-2495) is not in mixedCase
Function ERC20Upgradeable.__ERC20_init_unchained(string,string) (L1MantleToken.sol#2497-2500) is not in mixedCase
Variable ERC20Upgradeable.__gap (L1MantleToken.sol#2838) is not in mixedCase
Function ERC20BurnableUpgradeable.__ERC20Burnable_init() (L1MantleToken.sol#2843-2844) is not in mixedCase
Function ERC20BurnableUpgradeable.__ERC20Burnable_init_unchained() (L1MantleToken.sol#2846-2847) is not in mixedCase
Variable ERC20BurnableUpgradeable.__gap (L1MantleToken.sol#2878) is not in mixedCase
Function ERC20PermitUpgradeable.__ERC20Permit_init(string) (L1MantleToken.sol#2903-2905) is not in mixedCase
Function ERC20PermitUpgradeable.__ERC20Permit_init_unchained(string) (L1MantleToken.sol#2907) is not in mixedCase
Function ERC20PermitUpgradeable.DOMAIN_SEPARATOR() (L1MantleToken.sol#2944-2946) is not in mixedCase
Variable ERC20PermitUpgradeable._PERMIT_TYPEHASH_DEPRECATED_SLOT (L1MantleToken.sol#2896) is not in mixedCase
Variable ERC20PermitUpgradeable.__gap (L1MantleToken.sol#2964) is not in mixedCase
Function ERC20VotesUpgradeable.__ERC20Votes_init() (L1MantleToken.sol#2970-2971) is not in mixedCase
Function ERC20VotesUpgradeable.__ERC20Votes_init_unchained() (L1MantleToken.sol#2973-2974) is not in mixedCase
Variable ERC20VotesUpgradeable.__gap (L1MantleToken.sol#3228) is not in mixedCase
Parameter L1MantleToken.initialize(uint256,address)._initialSupply (L1MantleToken.sol#3300) is not in mixedCase
Parameter L1MantleToken.initialize(uint256,address)._owner (L1MantleToken.sol#3300) is not in mixedCase
Parameter L1MantleToken.mint(address,uint256)._recipient (L1MantleToken.sol#3325) is not in mixedCase
Parameter L1MantleToken.mint(address,uint256)._amount (L1MantleToken.sol#3325) is not in mixedCase
Parameter L1MantleToken.setMintCapNumerator(uint256)._mintCapNumerator (L1MantleToken.sol#3342) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
L1MantleToken.sol analyzed (19 contracts with 84 detectors), 166 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.

L1MantleToken.sol

Inline assembly:

The Contract uses inline assembly, this is only advised in rare cases. Additionally static analysis modules do not parse inline Assembly, this can lead to wrong analysis results.
more
Pos: 3217:15:

Block timestamp:

Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.
more
Pos: 3332:26:

Gas costs:

Gas requirement of function L1MantleToken.burnFrom 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: 2868:11:

Similar variable names:

L1MantleToken._burn(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.
Pos: 3368:36:

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: 2647:15:

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: 3326:43:

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.

L1MantleToken.sol

```
Compiler version ^0.8.1 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:3
Error message for require is too long
Pos: 9:63
Function name must be in mixedCase
Pos: 5:2972
Code contains empty blocks
Pos: 70:2972
Avoid making time-based decisions in your business logic
Pos: 17:3100
Error message for require is too long
Pos: 9:3123
Avoid using inline assembly. It is acceptable only in rare cases
Pos: 9:3216
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:3286
Avoid making time-based decisions in your business logic
Pos: 20:3309
Avoid making time-based decisions in your business logic
Pos: 13:3329
Avoid making time-based decisions in your business logic
Pos: 88:3329
Avoid making time-based decisions in your business logic
Pos: 20:3331
```

Software analysis result:

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

Email: audit@EtherAuthority.io