Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project:	Arbitrum (ARB) Token
Website:	arbitrum.foundation
Platform:	Ethereum
Language:	Solidity
Date:	April 12th, 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 contract audit initiatives, the smart contract of Arbitrum Token from arbitrum.foundation 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 April 12th, 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 Arbitrum token is a Permit token registered with an Arb One and Nova counterpart.
- The `L1ArbitrumToken` contract combines standard ERC20 features with permit functionality and integrates with the Arbitrum bridge infrastructure, facilitating secure token bridging between Ethereum (L1) and Arbitrum (L2) networks. It supports:
    - Initialization with specific gateway and router addresses.
    - Minting and burning of tokens controlled by the Arbitrum One gateway.
    - Registration on both Arbitrum One and Nova networks.
    - Standard ERC20 balance and transfer functions with additional interface compatibility.
- This design ensures robust and secure handling of ERC20 tokens that interact with the Arbitrum network, providing flexibility and security for cross-layer operations.

Audit scope

Name	Code Review and Security Analysis Report for Arbitrum (ARB) Token Smart Contract
Platform	**Ethereum**
File	L1ArbitrumToken.sol
Smart Contract Code	0xad0c361ef902a7d9851ca7dcc85535da2d3c6fc7
Audit Date	April 12th, 2024

Email: audit@EtherAuthority.io

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Tokenomics: • Name: Arbitrum • Symbol: ARB • Decimals: 18	**YES, This is valid.**
Ownership control: • The mint address and amount can be set by the only L1-ARB gateway. • Burn address and amount can be set by the only L1 arb gateway. • The current admin can check the condition the current admin returns. • Returns the current implementation by the admin. • Changes the admin of the proxy by the admin. • Upgrade the implementation of the proxy by the admin. • Upgrade the implementation of the proxy, and then call a function from the new implementation as specified by `data` by the admin.	**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, 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. A 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 4 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	Solidity version not specified	Passed
	Solidity version too old	Moderated
	Integer overflow/underflow	Passed
	Function input parameters lack of 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 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 Proxy?	Yes
Can Take Ownership?	Not Detected
Hidden Owner?	Not Detected
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 Arbitrum 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 Arbitrum 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 not well commented on in the smart contract. Ethereum's NatSpec commenting style is recommended.

Documentation

We were given an Arbitrum Token smart contract code in the form of an Etherscan web link.

As mentioned above, code parts are not well commented on. but 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 are 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

L1ArbitrumToken.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	initialize	write	initializer	No Issue
3	isArbitrumEnabled	external	Passed	No Issue
4	onlyArbOneGateway	modifier	Passed	No Issue
5	bridgeMint	write	access only Arb One Gateway	No Issue
6	bridgeBurn	write	access only Arb One Gateway	No Issue
7	registerTokenOnL2	write	Passed	No Issue
8	balanceOf	read	Passed	No Issue
9	transferFrom	write	Passed	No Issue
10	__ERC20_init	internal	access only Initializing	No Issue
11	__ERC20_init_unchained	internal	access only Initializing	No Issue
12	name	read	Passed	No Issue
13	symbol	read	Passed	No Issue
14	decimals	read	Passed	No Issue
15	totalSupply	read	Passed	No Issue
16	balanceOf	read	Passed	No Issue
17	transfer	write	Passed	No Issue
18	allowance	read	Passed	No Issue
19	approve	write	Passed	No Issue
20	transferFrom	write	Passed	No Issue
21	increaseAllowance	write	Passed	No Issue
22	decreaseAllowance	write	Passed	No Issue
23	_transfer	internal	Passed	No Issue
24	_mint	internal	Passed	No Issue
25	_burn	internal	Passed	No Issue
26	_approve	internal	Passed	No Issue
27	_spendAllowance	internal	Passed	No Issue
28	_beforeTokenTransfer	internal	Passed	No Issue
29	_afterTokenTransfer	internal	Passed	No Issue
30	__ERC20Permit_init	internal	access only Initializing	No Issue
31	__ERC20Permit_init_unchained	internal	access only Initializing	No Issue
32	permit	write	Timestamp	Refer Audit Findings
33	nonces	read	Passed	No Issue
34	DOMAIN_SEPARATOR	external	Passed	No Issue

35	_useNonce	internal	Passed	No Issue
36	transferAndCall	write	Passed	No Issue
37	contractFallback	write	Passed	No Issue
38	isContract	read	Passed	No Issue
39	initializer	modifier	Passed	No Issue
40	reinitializer	modifier	Passed	No Issue
41	onlyInitializing	modifier	Passed	No Issue
42	_disableInitializers	internal	Passed	No Issue
43	__Context_init	internal	access only Initializing	No Issue
44	__Context_init_unchained	internal	access only Initializing	No Issue
45	_msgSender	internal	Passed	No Issue
46	_msgData	internal	Passed	No Issue
47	__EIP712_init	internal	access only Initializing	No Issue
48	__EIP712_init_unchained	internal	access only Initializing	No Issue
49	_domainSeparatorV4	internal	Passed	No Issue
50	_buildDomainSeparator	read	Passed	No Issue
51	_hashTypedDataV4	internal	Passed	No Issue
52	_EIP712NameHash	internal	Passed	No Issue
53	_EIP712VersionHash	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) Error message for require is too long:

L1ArbitrumToken.sol

Ethereum has a gas limit for each block. This limit includes the gas used by all transactions and contract executions within that block. When a required statement fails, it results in an exception, and the error message, along with the gas used up to that point, is included in the transaction's revert message.

Resolution: We suggest writing short and clear messages in the required statements.

(2) Timestamp: **L1ArbitrumToken.sol**

```
ERC20PermitUpgradeable.permit(address,address,uint256,uint256,uint8,bytes32,bytes32) (L1ArbitrumToken.sol#667-68
6) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp <= deadline,ERC20Permit: expired deadline) (L1ArbitrumToken.sol#6
76)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
```

Dangerous usage of block.timestamp. block.timestamp can be manipulated by miners.

Resolution: Avoid relying on block.timestamp.

(3) pragma:

L1ArbitrumToken.sol

```
Different versions of Solidity are used:
        - Version used: ['0.8.16', '>=0.6.9<0.9.0', '^0.8.0', '^0.8.1', '^0.8.16', '^0.8.2']
        - 0.8.16 (L1ArbitrumToken.sol#2)
        - >=0.6.9<0.9.0 (ICustomToken.sol#20)
        - ^0.8.0 (ContextUpgradeable.sol#4)
        - ^0.8.0 (CountersUpgradeable.sol#4)
        - ^0.8.0 (ECDSAUpgradeable.sol#4)
        - ^0.8.0 (ERC20Upgradeable.sol#4)
        - ^0.8.0 (IERC20MetadataUpgradeable.sol#4)
        - ^0.8.0 (IERC20Upgradeable.sol#4)
        - ^0.8.0 (StringsUpgradeable.sol#4)
        - ^0.8.0 (draft-EIP712Upgradeable.sol#4)
        - ^0.8.0 (draft-ERC20PermitUpgradeable.sol#4)
        - ^0.8.0 (draft-IERC20PermitUpgradeable.sol#4)
        - ^0.8.1 (AddressUpgradeable.sol#4)
        - ^0.8.16 (TransferAndCallToken.sol#3)
        - ^0.8.2 (Initializable.sol#4)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used
```

Initializable.sol

```
Different versions of Solidity are used:
        - Version used: ['^0.8.1', '^0.8.2']
        - ^0.8.1 (AddressUpgradeable.sol#4)
        - ^0.8.2 (Initializable.sol#4)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used
```

Detected different Solidity versions are used.

Resolution: Use one Solidity version.

(4) sold-version: **ICustomToken.sol**

```
Pragma version>=0.6.9<0.9.0 (ICustomToken.sol#20) is too complex
solc-0.8.16 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
```

The contract has an unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to

differing compiler version numbers. This can lead to ambiguity when debugging as compiler-specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Resolution: Use a simple pragma version. Consider using the latest version of Solidity for testing.

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. Following are Admin functions:

L1ArbitrumToken.sol

- bridgeMint: Mint address and amount can be set by the only l1 arb one gateway.
- bridgeBurn: Burn address and amount can be set by the only l1 arb one gateway.

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 4 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 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 are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree 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.

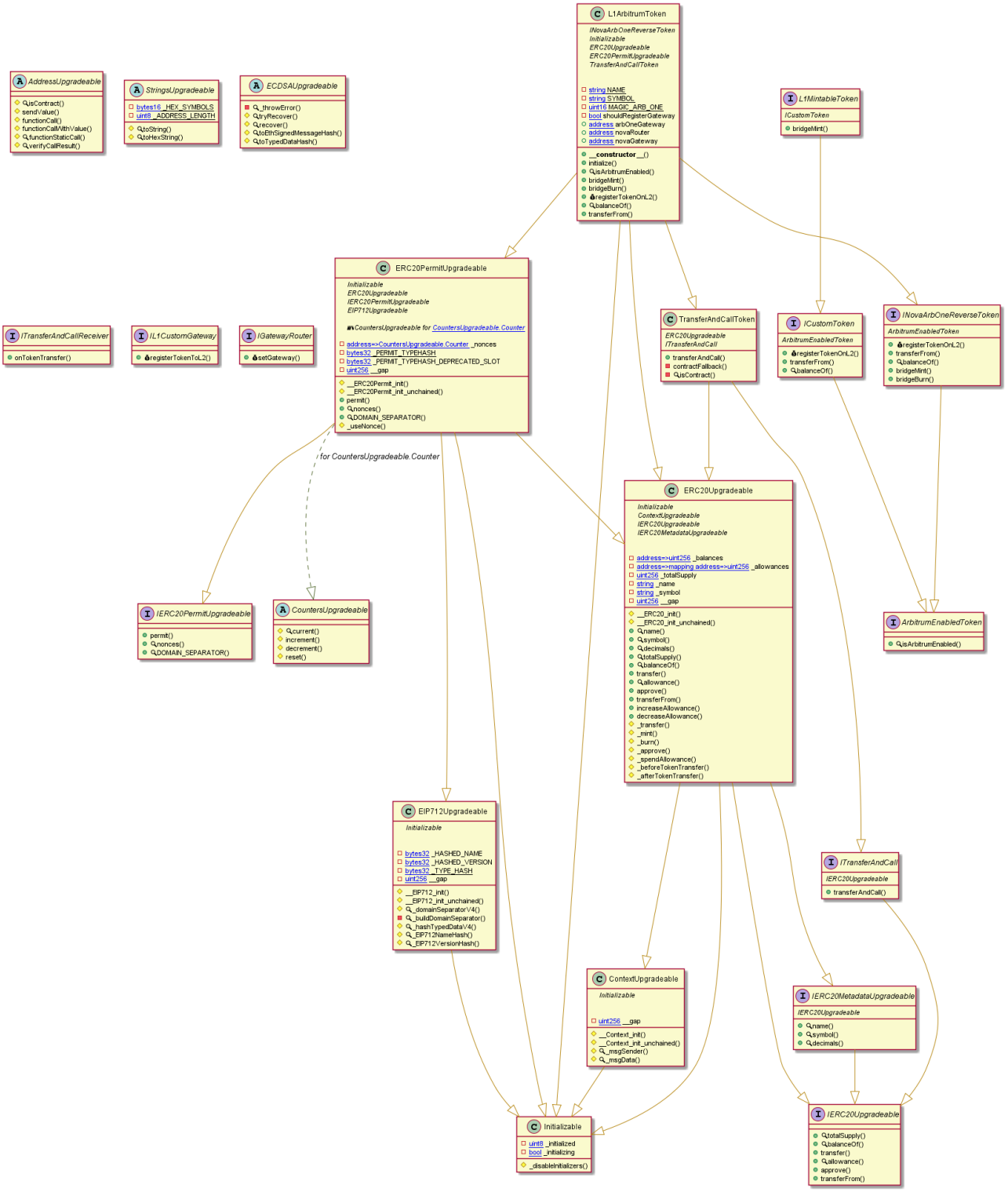
Technical Disclaimer

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 - Arbitrum Token

L1Arbitrum Token Diagram



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 >> L1ArbitrumToken.sol

```
ERC20PermitUpgradeable.__ERC20Permit_init(string).name (L1ArbitrumToken.sol#661) shadows:
        - ERC20Upgradeable.name() (L1ArbitrumToken.sol#453-455) (function)
        - IERC20MetadataUpgradeable.name() (L1ArbitrumToken.sol#319) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing

ERC20PermitUpgradeable.permit(address,address,uint256,uint256,uint8,bytes32,bytes32) (L1ArbitrumToken.sol#667-686) uses timesta
mp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp <= deadline,ERC20Permit: expired deadline) (L1ArbitrumToken.sol#676)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

AddressUpgradeable.verifyCallResult(bool,bytes,string) (L1ArbitrumToken.sol#106-123) uses assembly
        - INLINE ASM (L1ArbitrumToken.sol#115-118)
ECDSAUpgradeable.tryRecover(bytes32,bytes) (L1ArbitrumToken.sol#231-245) uses assembly
        - INLINE ASM (L1ArbitrumToken.sol#236-240)
TransferAndCallToken.isContract(address) (L1ArbitrumToken.sol#643-649) uses assembly
        - INLINE ASM (L1ArbitrumToken.sol#645-647)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

AddressUpgradeable.functionCall(address,bytes) (L1ArbitrumToken.sol#58-60) is never used and should be removed
AddressUpgradeable.functionCall(address,bytes,string) (L1ArbitrumToken.sol#62-68) is never used and should be removed
AddressUpgradeable.functionCallWithValue(address,bytes,uint256) (L1ArbitrumToken.sol#70-76) is never used and should be removed
AddressUpgradeable.functionCallWithValue(address,bytes,uint256,string) (L1ArbitrumToken.sol#78-89) is never used and should be
removed
AddressUpgradeable.functionStaticCall(address,bytes) (L1ArbitrumToken.sol#91-93) is never used and should be removed
AddressUpgradeable.functionStaticCall(address,bytes,string) (L1ArbitrumToken.sol#95-104) is never used and should be removed
AddressUpgradeable.sendValue(address,uint256) (L1ArbitrumToken.sol#51-56) is never used and should be removed
AddressUpgradeable.verifyCallResult(bool,bytes,string) (L1ArbitrumToken.sol#106-123) is never used and should be removed
ContextUpgradeable.__Context_init() (L1ArbitrumToken.sol#374-375) is never used and should be removed
ContextUpgradeable.__Context_init_unchained() (L1ArbitrumToken.sol#377-378) is never used and should be removed
ContextUpgradeable._msgData() (L1ArbitrumToken.sol#383-385) is never used and should be removed
CountersUpgradeable.decrement(CountersUpgradeable.Counter) (L1ArbitrumToken.sol#141-147) is never used and should be removed
CountersUpgradeable.reset(CountersUpgradeable.Counter) (L1ArbitrumToken.sol#149-151) is never used and should be removed
ECDSAUpgradeable.recover(bytes32,bytes) (L1ArbitrumToken.sol#247-251) is never used and should be removed
```

```
ECDSAUpgradeable.recover(bytes32,bytes32,bytes32) (L1ArbitrumToken.sol#263-271) is never used and should be removed
ECDSAUpgradeable.toEthSignedMessageHash(bytes) (L1ArbitrumToken.sol#309-311) is never used and should be removed
ECDSAUpgradeable.toEthSignedMessageHash(bytes32) (L1ArbitrumToken.sol#305-307) is never used and should be removed
ECDSAUpgradeable.tryRecover(bytes32,bytes) (L1ArbitrumToken.sol#231-245) is never used and should be removed
ECDSAUpgradeable.tryRecover(bytes32,bytes32,bytes32) (L1ArbitrumToken.sol#253-261) is never used and should be removed
EIP712Upgradeable.__EIP712_init(string,string) (L1ArbitrumToken.sol#396-398) is never used and should be removed
ERC20PermitUpgradeable.__ERC20Permit_init_unchained(string) (L1ArbitrumToken.sol#665) is never used and should be removed
StringsUpgradeable.toHexString(address) (L1ArbitrumToken.sol#203-205) is never used and should be removed
StringsUpgradeable.toHexString(uint256) (L1ArbitrumToken.sol#178-189) is never used and should be removed
StringsUpgradeable.toHexString(uint256,uint256) (L1ArbitrumToken.sol#191-201) is never used and should be removed
StringsUpgradeable.toString(uint256) (L1ArbitrumToken.sol#158-176) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version^0.8.19 (L1ArbitrumToken.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0
.7.6/0.8.16
solc-0.8.19 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) (L1ArbitrumToken.sol#51-56):
        - (success) = recipient.call{value: amount}() (L1ArbitrumToken.sol#54)
Low level call in AddressUpgradeable.functionCallWithValue(address,bytes,uint256,string) (L1ArbitrumToken.sol#78-89):
        - (success,returndata) = target.call{value: value}(data) (L1ArbitrumToken.sol#87)
Low level call in AddressUpgradeable.functionStaticCall(address,bytes,string) (L1ArbitrumToken.sol#95-104):
        - (success,returndata) = target.staticcall(data) (L1ArbitrumToken.sol#102)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
```

```
Function IERC20PermitUpgradeable.DOMAIN_SEPARATOR() (L1ArbitrumToken.sol#41) is not in mixedCase
Function ContextUpgradeable.__Context_init() (L1ArbitrumToken.sol#374-375) is not in mixedCase
Function ContextUpgradeable.__Context_init_unchained() (L1ArbitrumToken.sol#377-378) is not in mixedCase
Variable ContextUpgradeable.__gap (L1ArbitrumToken.sol#387) is not in mixedCase
Function EIP712Upgradeable.__EIP712_init(string,string) (L1ArbitrumToken.sol#396-398) is not in mixedCase
Function EIP712Upgradeable.__EIP712_init_unchained(string,string) (L1ArbitrumToken.sol#400-405) is not in mixedCase
Function EIP712Upgradeable._EIP712NameHash() (L1ArbitrumToken.sol#423-425) is not in mixedCase
Function EIP712Upgradeable._EIP712VersionHash() (L1ArbitrumToken.sol#427-429) is not in mixedCase
```

```
Function EIP712Upgradeable._EIP712VersionHash() (L1ArbitrumToken.sol#427-429) is not in mixedCase
Variable EIP712Upgradeable._HASHED_NAME (L1ArbitrumToken.sol#391) is not in mixedCase
Variable EIP712Upgradeable._HASHED_VERSION (L1ArbitrumToken.sol#392) is not in mixedCase
Variable EIP712Upgradeable.__gap (L1ArbitrumToken.sol#431) is not in mixedCase
Function ERC20Upgradeable.__ERC20_init(string,string) (L1ArbitrumToken.sol#444-446) is not in mixedCase
Function ERC20Upgradeable.__ERC20_init_unchained(string,string) (L1ArbitrumToken.sol#448-451) is not in mixedCase
Variable ERC20Upgradeable.__gap (L1ArbitrumToken.sol#606) is not in mixedCase
Parameter TransferAndCallToken.transferAndCall(address,uint256,bytes)._to (L1ArbitrumToken.sol#623) is not in mixedCase
Parameter TransferAndCallToken.transferAndCall(address,uint256,bytes)._value (L1ArbitrumToken.sol#623) is not in mixedCase
Parameter TransferAndCallToken.transferAndCall(address,uint256,bytes)._data (L1ArbitrumToken.sol#623) is not in mixedCase
Parameter TransferAndCallToken.contractFallback(address,uint256,bytes)._to (L1ArbitrumToken.sol#638) is not in mixedCase
Parameter TransferAndCallToken.contractFallback(address,uint256,bytes)._value (L1ArbitrumToken.sol#638) is not in mixedCase
Parameter TransferAndCallToken.contractFallback(address,uint256,bytes)._data (L1ArbitrumToken.sol#638) is not in mixedCase
Parameter TransferAndCallToken.isContract(address)._addr (L1ArbitrumToken.sol#643) is not in mixedCase
Function ERC20PermitUpgradeable.__ERC20Permit_init(string) (L1ArbitrumToken.sol#661-663) is not in mixedCase
Function ERC20PermitUpgradeable.__ERC20Permit_init_unchained(string) (L1ArbitrumToken.sol#665) is not in mixedCase
Function ERC20PermitUpgradeable.DOMAIN_SEPARATOR() (L1ArbitrumToken.sol#692-694) is not in mixedCase
Variable ERC20PermitUpgradeable._PERMIT_TYPEHASH_DEPRECATED_SLOT (L1ArbitrumToken.sol#659) is not in mixedCase
Variable ERC20PermitUpgradeable.__gap (L1ArbitrumToken.sol#702) is not in mixedCase
Parameter L1ArbitrumToken.initialize(address,address,address)._arbOneGateway (L1ArbitrumToken.sol#802) is not in mixedCase
Parameter L1ArbitrumToken.initialize(address,address,address)._novaRouter (L1ArbitrumToken.sol#802) is not in mixedCase
Parameter L1ArbitrumToken.initialize(address,address,address)._novaGateway (L1ArbitrumToken.sol#802) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
L1ArbitrumToken.sol analyzed (22 contracts with 84 detectors), 68 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.

L1ArbitrumToken.sol

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in AddressUpgradeable.functionCallWithValue(address,bytes,uint256,string): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 78:4:

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: 645:8:

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: 676:16:

Gas costs:

Gas requirement of function L1ArbitrumToken.bridgeMint 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: 828:4:

Constant/View/Pure functions:

L1MintableToken.bridgeMint(address,uint256) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 732:4:

Similar variable names:

L1ArbitrumToken.bridgeBurn(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.
Pos: 841:23:

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: 824:8:

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.

L1ArbitrumToken.sol

```
Compiler version ^0.8.0 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:2
Function name must be in mixedCase
Pos: 5:40
Error message for require is too long
Pos: 9:54
Error message for require is too long
Pos: 9:83
Error message for require is too long
Pos: 9:99
Avoid using inline assembly. It is acceptable only in rare cases
Pos: 17:114
Error message for revert is too long
Pos: 13:224
Error message for revert is too long
Pos: 13:226
Avoid using inline assembly. It is acceptable only in rare cases
Pos: 13:235
Error message for require is too long
Pos: 9:334
Error message for require is too long
Pos: 9:350
Error message for require is too long
Pos: 9:359
Error message for require is too long
Pos: 9:364
Function name must be in mixedCase
Pos: 5:373
Code contains empty blocks
Pos: 57:373
Function name must be in mixedCase
Pos: 5:376
Code contains empty blocks
Pos: 67:376
Variable name must be in mixedCase
Pos: 5:390
Variable name must be in mixedCase
Pos: 5:391
Function name must be in mixedCase
Pos: 5:395
Function name must be in mixedCase
Pos: 5:399
```

```
Function name must be in mixedCase
Pos: 5:422
Function name must be in mixedCase
Pos: 5:426
Function name must be in mixedCase
Pos: 5:443
Function name must be in mixedCase
Pos: 5:447
Error message for require is too long
Pos: 9:508
Error message for require is too long
Pos: 9:521
Error message for require is too long
Pos: 9:522
Error message for require is too long
Pos: 9:527
Error message for require is too long
Pos: 9:551
Error message for require is too long
Pos: 9:556
Error message for require is too long
Pos: 9:572
Error message for require is too long
Pos: 9:573
Avoid using inline assembly. It is acceptable only in rare cases
Pos: 9:644
Function name must be in mixedCase
Pos: 5:664
Code contains empty blocks
Pos: 84:664
Avoid making time-based decisions in your business logic
Pos: 17:675
Function name must be in mixedCase
Pos: 5:691
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:797
Error message for require is too long
Pos: 9:805
Error message for require is too long
Pos: 9:806
Error message for require is too long
Pos: 9:807
Error message for require is too long
Pos: 9:818
Error message for require is too long
Pos: 9:823
```

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