AUDITED BY

LOVELY INSPECTOR

SMART CONTRACT

SECURITY AUDIT

ARBITRUM TOKEN

inspector.lovely.finance

LOVELY INSPECTOR

TABLE OF CONTENTS

LOVELY INSPECTOR

DISCLAIMER

LOVELY INSPECTOR

AUDIT SCOPE

Name	Code Review and Security Analysis Report for Arbitrum Token Coin Smart Contract
Platform	Ethereum
Language	Solidity
File	L1ArbitrumToken.sol
Ethereum Code	0xad0c361ef902a7d9851ca7dcc85535da2d3c6fc7
Audit Date	November 8th, 2023

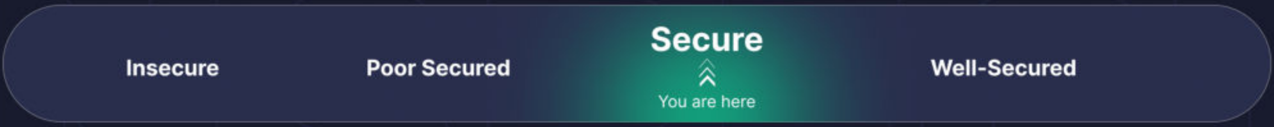
LOVELY INSPECTOR

PROPOSED SMART CONTRACT FEATURES

Claimed Feature Detail	Our Observation
Tokenomics: • Name: Arbitrum • Symbol: ARB • Decimals: 18	Validated
Ownership control: • Allow the Arb One bridge to mint tokens by only l1 arb one gateway. • Allow the Arb One bridge to burn tokens by only l1 arb one gateway.	Validated

AUDIT SUMMARY

According to the standard audit assessment, 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. General overview is presented in AS-IS section and all identified issues can be found in the Audit overview section.

We found 0 critical, 0 high, 0 medium and 0 low, and 0 very low level issues.

Investors Advice: 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.

KEY TECHNICAL METRICS

MAIN CATEGORY	SUBCATEGORY	RESULT
Contract Programming	Solidity version is not specified	Passed
	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

LOVELY INSPECTOR

BUSINESS RISK ANALYSIS

CATEGORY	RESULT
● Buy Tax	0%
● Sell Tax	0%
● Cannot Buy	Not Detected
● Cannot Sell	Not Detected
● Max Tax	0%
● Modify Tax	Not Detected
● Fee Check	No
● Is Honeypot	Not Detected
● Trading Cooldown	Not Detected
● Can Pause Trade?	No
● Pause Transfer?	No
● Max Tax?	No
● Is it Anti-whale?	No
● Is Anti-bot?	Not Detected
● Is it a Blacklist?	Not Detected
● Blacklist Check	No
● Can Mint?	No
● Is it Proxy?	No
● Can Take Ownership?	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 contract contains 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 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 an Arbitrum 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.

Another source of information was its official website: arbitrum.io which provided rich information about the project architecture and tokenomics.

USE OF DEPENDENCIES

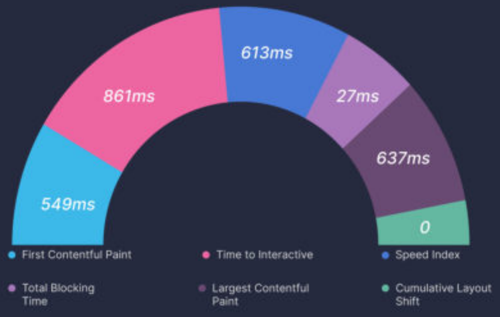
As per our observation, the libraries are used in this smart contract infrastructure that are based on well known industry standard open source projects.

Apart from libraries, its functions are not used in external smart contract calls.

LOVELY INSPECTOR

PROJECT WEBSITE PERFORMANCE AUDIT

Performance Metrics



Browser Timings

● Redirect Duration 0ms	● Connection Duration 95ms	● Backend Duration 229ms
● Time to First Byte 324ms	● First Paint 470ms	● DOM Interactive Time 471ms
● DOM Content Loaded 550ms	● Onload Time 652ms	● Fully Loaded Time 1.3s

Grade

	Performance	Structure
A	100%	88%

Web Vitals

LCP	TBT	CLS
637ms	27ms	0

IMPECT	AUDIT
High	Enable Keep-Alive (FCP) (LCP)
URL WITHOUT KEEP-ALIVE ENABLED	. HTTP://ARBITRUM.IO/
Low	Allow back/forward cache restoration
Low	Avoid an excessive DOM size (TBT)
Low	Avoid enormous network payloads (LCP)
Low	Avoid multiple page redirects (FCP) (LCP)

Level of Criticality

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 significant impact on smart contract execution, e.g. public access to crucial
Med	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 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.

LOVELY INSPECTOR

AUDIT FINDINGS TABLE

	Total	Resolved	UnResolved	Acknowledged
High Severity Issues Found	0	0	0	0
Moderate Severity Issues Found	0	0	0	0
Medium Severity Issues	0	0	0	0
Low Severity Issues	0	0	0	0
Informational Observations	0	0	0	0

The Arbitrum Token - Audit report identifies 0 issues with varying severity levels, discovered through manual review and static analysis techniques, alongside rigorous code reviews, highlighting the need for further investigation and vulnerability identification.

The smart contract is considered to **pass the audit**, as of the audit date, if no high-severity or moderate-severity issues are found.

LOVELY INSPECTOR

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.

CENTRALIZATION

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

L1ArbitrumToken.sol

- bridgeMint: Allow the Arb One bridge to mint tokens by only l1 arb one gateway.
- bridgeBurn: Allow the Arb One bridge to burn tokens by 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 had not observed any issues 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.

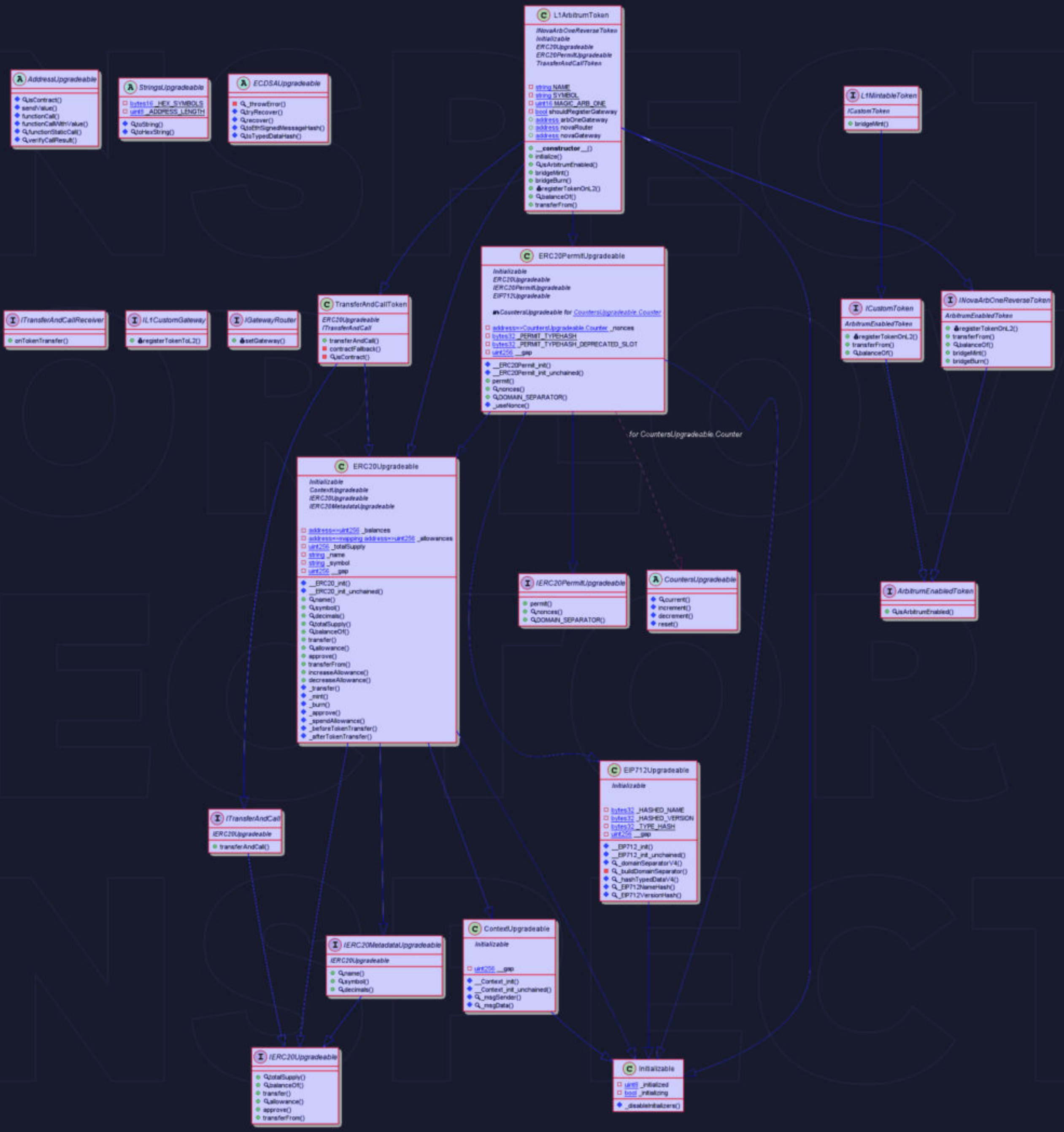
Audit report contains all found security vulnerabilities and other issues in the reviewed code.

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

ADDENDUM

Code Flow Diagram

Arbitrum Token

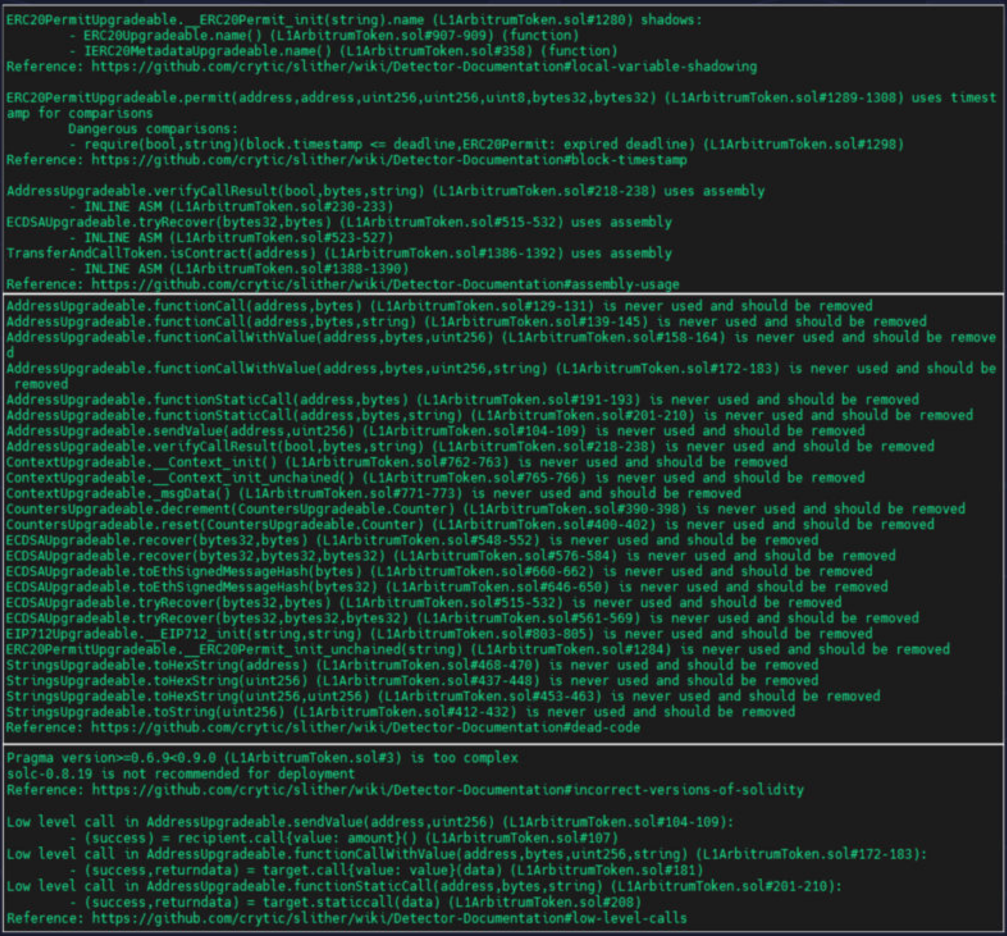


SECURITY ASSESSMENT REPORT

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 together. Below are the results.

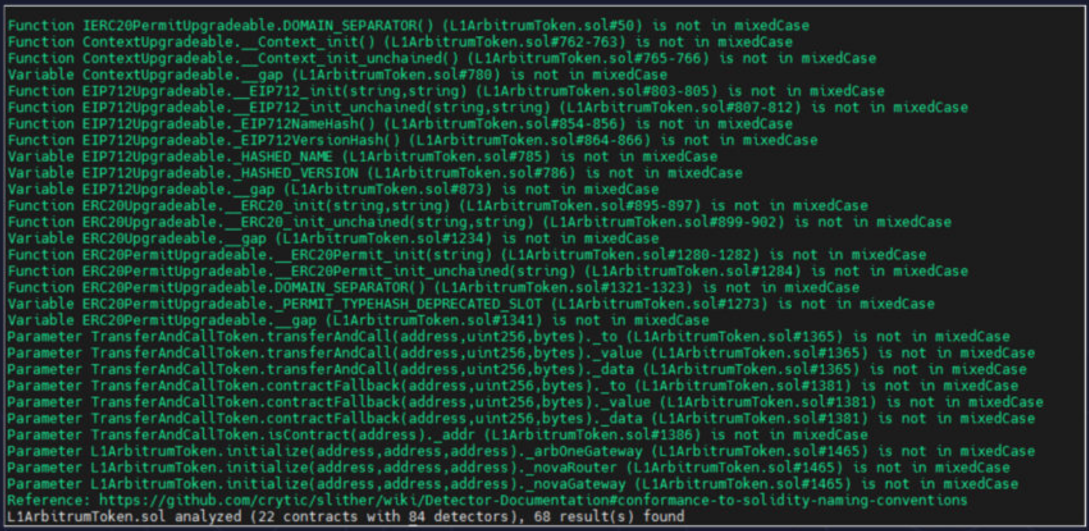
Slither Log >> L1ArbitrumToken.sol



```
ERC20PermitUpgradeable.__ERC20Permit_init(string).name (L1ArbitrumToken.sol#1280) shadows:
        - ERC20Upgradeable.name() (L1ArbitrumToken.sol#907-909) (function)
        - IERC20MetadataUpgradeable.name() (L1ArbitrumToken.sol#358) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing

ERC20PermitUpgradeable.permit(address,address,uint256,uint256,uint8,bytes32,bytes32) (L1ArbitrumToken.sol#1289-1308) uses timest
amp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp <= deadline,ERC20Permit: expired deadline) (L1ArbitrumToken.sol#1298)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

AddressUpgradeable.verifyCallResult(bool,bytes,string) (L1ArbitrumToken.sol#218-238) uses assembly
        - INLINE ASM (L1ArbitrumToken.sol#230-233)
ECDSAUpgradeable.tryRecover(bytes32,bytes) (L1ArbitrumToken.sol#515-532) uses assembly
        - INLINE ASM (L1ArbitrumToken.sol#523-527)
TransferAndCallToken.isContract(address) (L1ArbitrumToken.sol#1386-1392) uses assembly
        - INLINE ASM (L1ArbitrumToken.sol#1388-1390)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
```

```
AddressUpgradeable.functionCall(address,bytes) (L1ArbitrumToken.sol#129-131) is never used and should be removed
AddressUpgradeable.functionCall(address,bytes,string) (L1ArbitrumToken.sol#139-145) is never used and should be removed
AddressUpgradeable.functionCallWithValue(address,bytes,uint256) (L1ArbitrumToken.sol#158-164) is never used and should be remove
d
AddressUpgradeable.functionCallWithValue(address,bytes,uint256,string) (L1ArbitrumToken.sol#172-183) is never used and should be
 removed
AddressUpgradeable.functionStaticCall(address,bytes) (L1ArbitrumToken.sol#191-193) is never used and should be removed
AddressUpgradeable.functionStaticCall(address,bytes,string) (L1ArbitrumToken.sol#201-210) is never used and should be removed
AddressUpgradeable.sendValue(address,uint256) (L1ArbitrumToken.sol#104-109) is never used and should be removed
AddressUpgradeable.verifyCallResult(bool,bytes,string) (L1ArbitrumToken.sol#218-238) is never used and should be removed
ContextUpgradeable.__Context_init() (L1ArbitrumToken.sol#762-763) is never used and should be removed
ContextUpgradeable.__Context_init_unchained() (L1ArbitrumToken.sol#765-766) is never used and should be removed
ContextUpgradeable._msgData() (L1ArbitrumToken.sol#771-773) is never used and should be removed
CountersUpgradeable.decrement(CountersUpgradeable.Counter) (L1ArbitrumToken.sol#390-398) is never used and should be removed
CountersUpgradeable.reset(CountersUpgradeable.Counter) (L1ArbitrumToken.sol#400-402) is never used and should be removed
ECDSAUpgradeable.recover(bytes32,bytes) (L1ArbitrumToken.sol#548-552) is never used and should be removed
ECDSAUpgradeable.recover(bytes32,bytes32,bytes32) (L1ArbitrumToken.sol#576-584) is never used and should be removed
ECDSAUpgradeable.toEthSignedMessageHash(bytes) (L1ArbitrumToken.sol#660-662) is never used and should be removed
ECDSAUpgradeable.toEthSignedMessageHash(bytes32) (L1ArbitrumToken.sol#646-650) is never used and should be removed
ECDSAUpgradeable.tryRecover(bytes32,bytes) (L1ArbitrumToken.sol#515-532) is never used and should be removed
ECDSAUpgradeable.tryRecover(bytes32,bytes32,bytes32) (L1ArbitrumToken.sol#561-569) is never used and should be removed
EIP712Upgradeable.__EIP712_init(string,string) (L1ArbitrumToken.sol#803-805) is never used and should be removed
ERC20PermitUpgradeable.__ERC20Permit_init_unchained(string) (L1ArbitrumToken.sol#1284) is never used and should be removed
StringsUpgradeable.toHexString(address) (L1ArbitrumToken.sol#468-470) is never used and should be removed
StringsUpgradeable.toHexString(uint256) (L1ArbitrumToken.sol#437-448) is never used and should be removed
StringsUpgradeable.toHexString(uint256,uint256) (L1ArbitrumToken.sol#453-463) is never used and should be removed
StringsUpgradeable.toString(uint256) (L1ArbitrumToken.sol#412-432) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
```

```
Pragma version>=0.6.9<0.9.0 (L1ArbitrumToken.sol#3) is too complex
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#104-109):
        - (success) = recipient.call{value: amount}() (L1ArbitrumToken.sol#107)
Low level call in AddressUpgradeable.functionCallWithValue(address,bytes,uint256,string) (L1ArbitrumToken.sol#172-183):
        - (success,returndata) = target.call{value: value}(data) (L1ArbitrumToken.sol#181)
Low level call in AddressUpgradeable.functionStaticCall(address,bytes,string) (L1ArbitrumToken.sol#201-210):
        - (success,returndata) = target.staticcall(data) (L1ArbitrumToken.sol#208)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
```

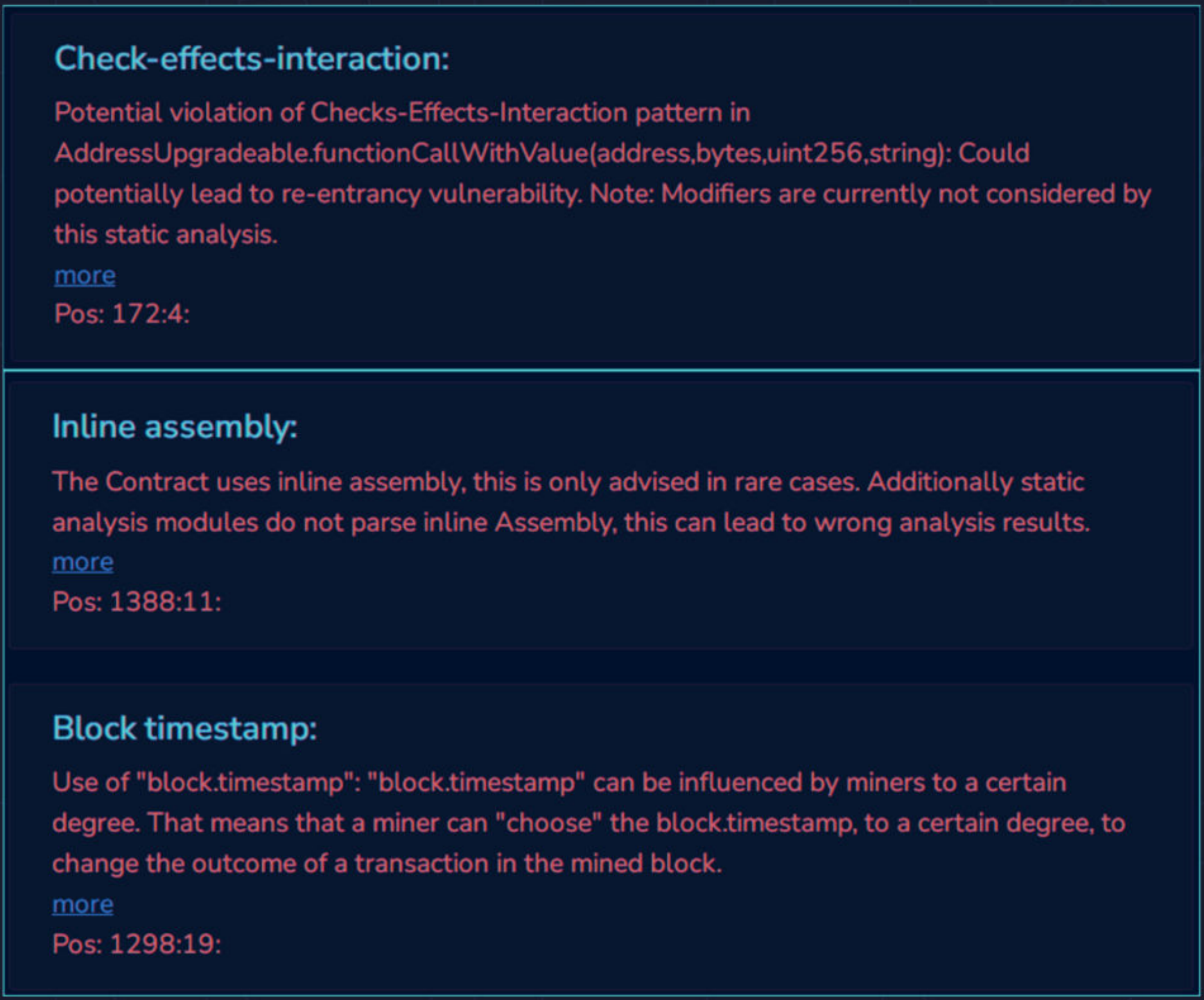


```
Function IERC20PermitUpgradeable.DOMAIN_SEPARATOR() (L1ArbitrumToken.sol#50) is not in mixedCase
Function ContextUpgradeable.__Context_init() (L1ArbitrumToken.sol#762-763) is not in mixedCase
Function ContextUpgradeable.__Context_init_unchained() (L1ArbitrumToken.sol#765-766) is not in mixedCase
Variable ContextUpgradeable.__gap (L1ArbitrumToken.sol#780) is not in mixedCase
Function EIP712Upgradeable.__EIP712_init(string,string) (L1ArbitrumToken.sol#803-805) is not in mixedCase
Function EIP712Upgradeable.__EIP712_init_unchained(string,string) (L1ArbitrumToken.sol#807-812) is not in mixedCase
Function EIP712Upgradeable._EIP712NameHash() (L1ArbitrumToken.sol#854-856) is not in mixedCase
Function EIP712Upgradeable._EIP712VersionHash() (L1ArbitrumToken.sol#864-866) is not in mixedCase
Variable EIP712Upgradeable._HASHED_NAME (L1ArbitrumToken.sol#785) is not in mixedCase
Variable EIP712Upgradeable._HASHED_VERSION (L1ArbitrumToken.sol#786) is not in mixedCase
Variable EIP712Upgradeable.__gap (L1ArbitrumToken.sol#873) is not in mixedCase
Function ERC20Upgradeable.__ERC20_init(string,string) (L1ArbitrumToken.sol#895-897) is not in mixedCase
Function ERC20Upgradeable.__ERC20_init_unchained(string,string) (L1ArbitrumToken.sol#899-902) is not in mixedCase
Variable ERC20Upgradeable.__gap (L1ArbitrumToken.sol#1234) is not in mixedCase
Function ERC20PermitUpgradeable.__ERC20Permit_init(string) (L1ArbitrumToken.sol#1280-1282) is not in mixedCase
Function ERC20PermitUpgradeable.__ERC20Permit_init_unchained(string) (L1ArbitrumToken.sol#1284) is not in mixedCase
Function ERC20PermitUpgradeable.DOMAIN_SEPARATOR() (L1ArbitrumToken.sol#1321-1323) is not in mixedCase
Variable ERC20PermitUpgradeable._PERMIT_TYPEHASH_DEPRECATED_SLOT (L1ArbitrumToken.sol#1273) is not in mixedCase
Variable ERC20PermitUpgradeable.__gap (L1ArbitrumToken.sol#1341) is not in mixedCase
Parameter TransferAndCallToken.transferAndCall(address,uint256,bytes)._to (L1ArbitrumToken.sol#1365) is not in mixedCase
Parameter TransferAndCallToken.transferAndCall(address,uint256,bytes)._value (L1ArbitrumToken.sol#1365) is not in mixedCase
Parameter TransferAndCallToken.transferAndCall(address,uint256,bytes)._data (L1ArbitrumToken.sol#1365) is not in mixedCase
Parameter TransferAndCallToken.contractFallback(address,uint256,bytes)._to (L1ArbitrumToken.sol#1381) is not in mixedCase
Parameter TransferAndCallToken.contractFallback(address,uint256,bytes)._value (L1ArbitrumToken.sol#1381) is not in mixedCase
Parameter TransferAndCallToken.contractFallback(address,uint256,bytes)._data (L1ArbitrumToken.sol#1381) is not in mixedCase
Parameter TransferAndCallToken.isContract(address)._addr (L1ArbitrumToken.sol#1386) is not in mixedCase
Parameter L1ArbitrumToken.initialize(address,address,address)._arbOneGateway (L1ArbitrumToken.sol#1465) is not in mixedCase
Parameter L1ArbitrumToken.initialize(address,address,address)._novaRouter (L1ArbitrumToken.sol#1465) is not in mixedCase
Parameter L1ArbitrumToken.initialize(address,address,address)._novaGateway (L1ArbitrumToken.sol#1465) 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: 172:4:

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

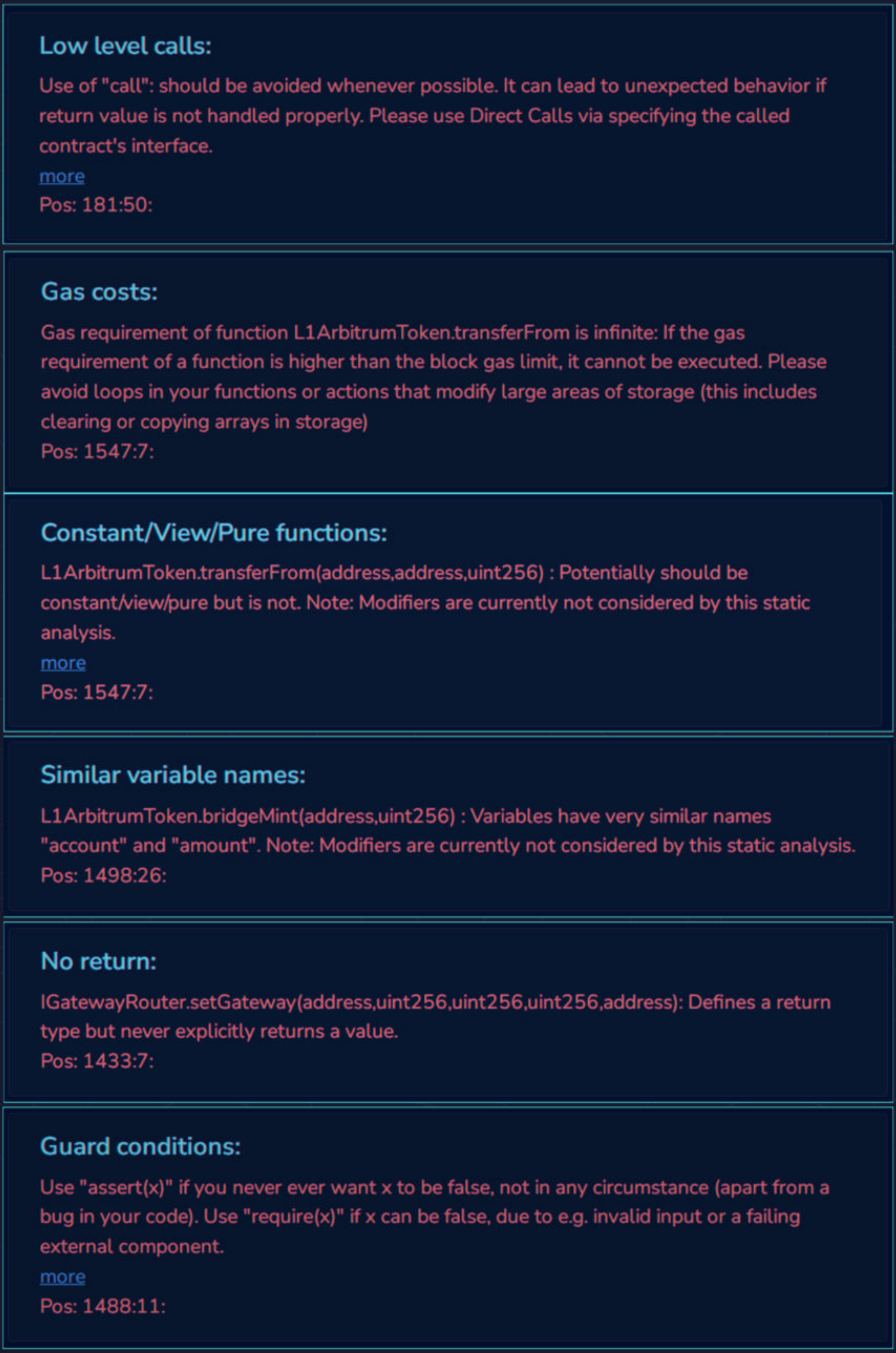
Pos: 1388:11:

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: 1298:19:



Low level calls:

Use of "call": should be avoided whenever possible. It can lead to unexpected behavior if return value is not handled properly. Please use Direct Calls via specifying the called contract's interface.
more
Pos: 181:50:

Gas costs:

Gas requirement of function L1ArbitrumToken.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: 1547:7:

Constant/View/Pure functions:

L1ArbitrumToken.transferFrom(address,address,uint256) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 1547:7:

Similar variable names:

L1ArbitrumToken.bridgeMint(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.
Pos: 1498:26:

No return:

IGatewayRouter.setGateway(address,uint256,uint256,uint256,address): Defines a return type but never explicitly returns a value.
Pos: 1433:7:

Guard conditions:

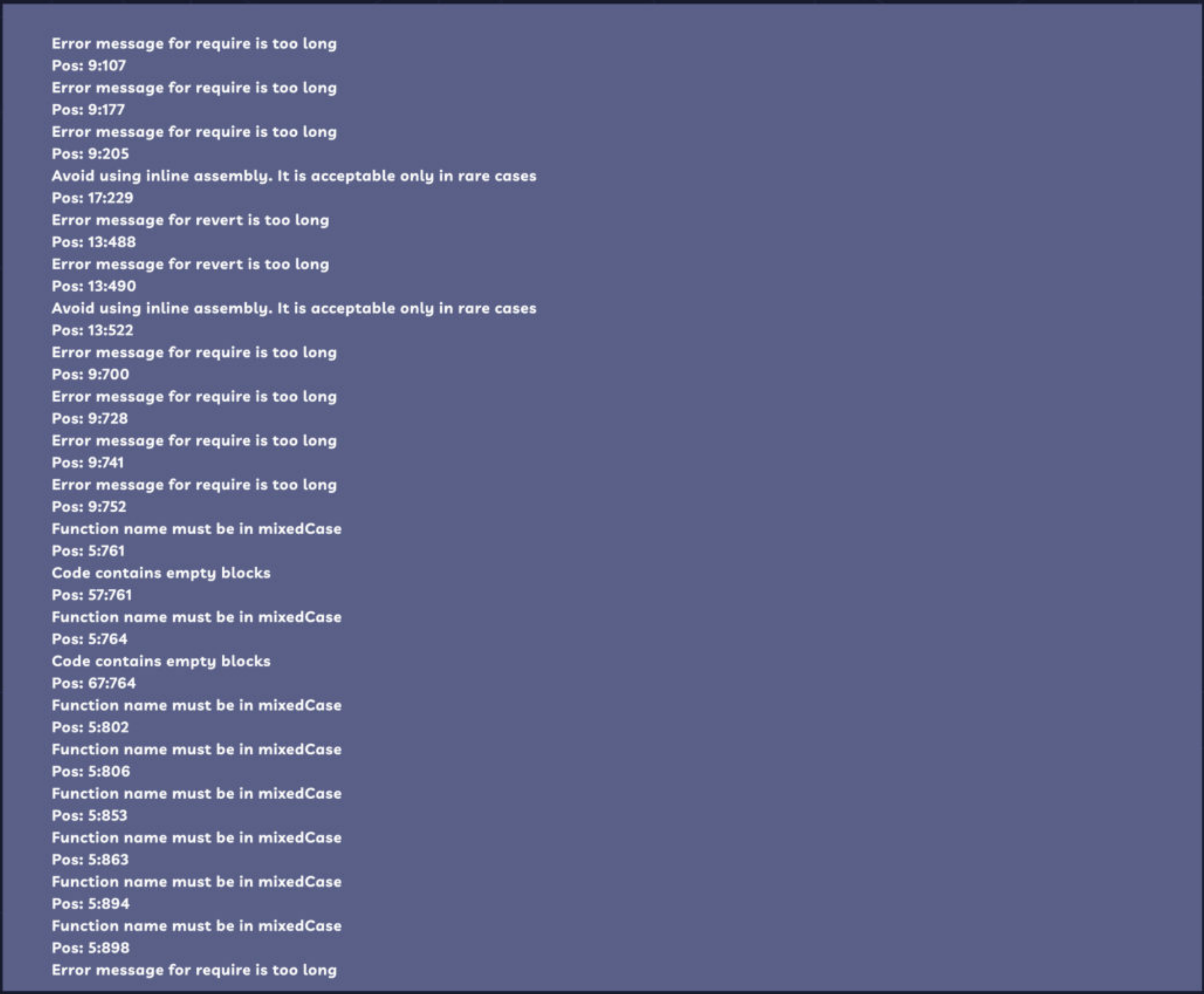
Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 1488:11:

LOVELY INSPECTOR

COMPLIANCE ANALYSIS

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



```
Error message for require is too long
Pos: 9:107
Error message for require is too long
Pos: 9:177
Error message for require is too long
Pos: 9:205
Avoid using inline assembly. It is acceptable only in rare cases
Pos: 17:229
Error message for revert is too long
Pos: 13:488
Error message for revert is too long
Pos: 13:490
Avoid using inline assembly. It is acceptable only in rare cases
Pos: 13:522
Error message for require is too long
Pos: 9:700
Error message for require is too long
Pos: 9:728
Error message for require is too long
Pos: 9:741
Error message for require is too long
Pos: 9:752
Function name must be in mixedCase
Pos: 5:761
Code contains empty blocks
Pos: 57:761
Function name must be in mixedCase
Pos: 5:764
Code contains empty blocks
Pos: 67:764
Function name must be in mixedCase
Pos: 5:802
Function name must be in mixedCase
Pos: 5:806
Function name must be in mixedCase
Pos: 5:853
Function name must be in mixedCase
Pos: 5:863
Function name must be in mixedCase
Pos: 5:894
Function name must be in mixedCase
Pos: 5:898
Error message for require is too long
```

```
Pos: 9:1048
Error message for require is too long
Pos: 9:1075
Error message for require is too long
Pos: 9:1076
Error message for require is too long
Pos: 9:1081
Error message for require is too long
Pos: 9:1125
Error message for require is too long
Pos: 9:1130
Error message for require is too long
Pos: 9:1159
Error message for require is too long
Pos: 9:1160
Code contains empty blocks
Pos: 24:1206
Code contains empty blocks
Pos: 24:1226
Function name must be in mixedCase
Pos: 5:1279
Function name must be in mixedCase
Pos: 5:1283
Code contains empty blocks
Pos: 84:1283
Avoid making time-based decisions in your business logic
Pos: 17:1297
Avoid using inline assembly. It is acceptable only in rare cases
Pos: 9:1387
Explicitly mark visibility in function (Set ignoreConstructors to true if using solidity >=0.7.0)
Pos: 5:1460
Error message for require is too long
Pos: 9:1468
Error message for require is too long
Pos: 9:1469
Error message for require is too long
Pos: 9:1470
Error message for require is too long
Pos: 9:1482
Error message for require is too long
Pos: 9:1487
```

SOFTWARE ANALYSIS RESULT

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

AUDITED BY
LOVELY
INSPECTOR

LOVELY INSPECTOR

INFO

Website: Inspector.lovely.finance

Telegram community: t.me/inspectorlovely

Twitter: twitter.com/InspectorLovely

inspector.lovely.finance