AUDITED BY
LOVELY INSPECTOR

SMART CONTRACT
SECURITY AUDIT

MAVERICK TOKEN

inspector.lovely.finance

LOVELY INSPECTOR

TABLE OF CONTENTS

LOVELY INSPECTOR

DISCLAIMER

This is a comprehensive report based on our automated and manual examination of cybersecurity vulnerabilities and framework flaws of the project's smart contract. Reading the full analysis report is essential to build your understanding of project's security level. It is crucial to take note, though we have done our best to perform this analysis and report, that you should not rely on the our research and cannot claim what it states or how we created it. Before making any judgments, you have to conduct your own independent research. We will discuss this in more depth in the following disclaimer - please read it fully. DISCLAIMER: You agree to the terms of this disclaimer by reading this report or any portion thereof. Please stop reading this report and remove and delete any copies of this report that you download and/or print if you do not agree to these conditions. Scan and verify report's presence in the GitHub repository by a qr-code on the title page. This report is for non-reliability information only and does not represent investment advice. No one shall be entitled to depend on the report or its contents, and Inspector Lovely and its affiliates shall not be held responsible to you or anyone else, nor shall Inspector Lovely provide any guarantee or representation to any person with regard to the accuracy or integrity of the report. Without any terms, warranties or other conditions other than as set forth in that exclusion and Inspector Lovely excludes hereby all representations, warrants, conditions and other terms (including, without limitation, guarantees implied by the law of satisfactory quality, fitness for purposes and the use of reasonable care and skills). The report is provided as "as is" and does not contain any terms and conditions. Except as legally banned, Inspector Lovely disclaims all responsibility and responsibilities and no claim against Inspector Lovely is made to any amount or type of loss or damages (without limitation, direct, indirect, special, punitive, consequential or pure economic loses or losses) that may be caused by you or any other person, or any damages or damages, including without limitations (whether innocent or negligent). Security analysis is based only on the smart contracts. No applications or operations were reviewed for security. No product code has been reviewed.

LOVELY INSPECTOR

AUDIT SCOPE

Name	Code Review and Security Analysis Report for Maverick Token Coin Smart Contract
Platform	Ethereum
Language	Solidity
File	MaverickToken.sol
Ethereum Code	0×7448c7456a97769F6cD04F1E83A4a23cCdC46aBD
Audit Date	November 8th, 2023

LOVELY INSPECTOR

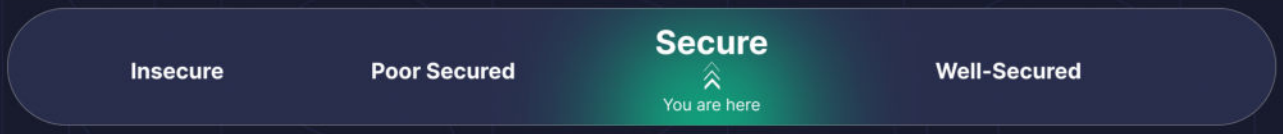
PROPOSED SMART CONTRACT FEATURES

Claimed Feature Detail	Our Observation
Tokenomics: • Name: Maverick Token • Symbol: MAV • Decimals: 18 • Total Supply: 2 billion	Validated
Ownership control: • Custom adapter parameter values can be set by the owner. • The owner can set configuration. • Sender / receiver Version can be set by the owner. • Resume Receive can be set by the owner. • Trusted Remote can be set by the owner. • Precrime addresses can be set by the owner. • Gas value can be set by the owner. • Payload size limit can be set by the owner. • Current owner can transfer the ownership. • Owner can renounce ownership.	Validated

LOVELY INSPECTOR

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 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.

LOVELY INSPECTOR

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?	Yes
Is it Proxy?	No
Can Take Ownership?	Yes
Hidden Owner?	Not Detected
Self Destruction?	Not Detected
Auditor Confidence	High

Overall Audit Result: PASSED

LOVELY INSPECTOR

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 Maverick 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 Maverick 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 a Maverick 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: https://www.mav.xyz which provided rich information about the project architecture and tokenomics.

USE OF DEPENDENCIES

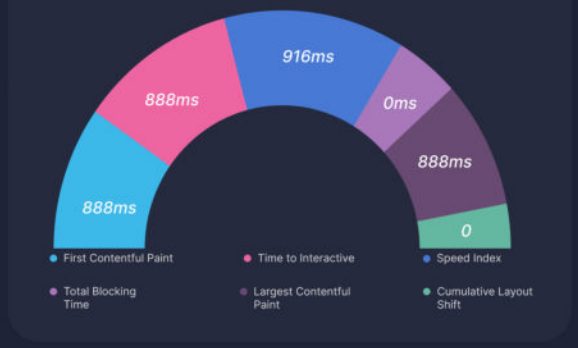
As per our observation, the libraries are used in this smart contract infrastructure that are based on well known industry standard open source projects.

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 **111ms**	Backend Duration **181ms**
Time to First Byte **292ms**	First Paint **449ms**	DOM Interactive Time **450ms**
DOM Content Loaded **863ms**	Onload Time **889ms**	Fully Loaded Time **2.9s**

Grade

A	Performance 98%	Structure 100%

Web Vitals

LCP	TBT	CLS
888ms	0ms	0

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 Maverick 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.

LOVELY INSPECTOR

CENTRALIZATION

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

OFTCore.sol

- setUseCustomAdapterParams: Custom adapter parameter values can be set by the owner.

LzApp.sol

- setConfig: The owner can set configuration.
- setSendVersion: Send Version can be set by the owner.
- setReceiveVersion: Receiver Version can be set by the owner.
- forceResumeReceive: Resume Receive can be set by the owner.
- setTrustedRemote: Trusted Remote can be set by the owner.
- setTrustedRemoteAddress: Trusted Remote address can be set by the owner.
- setPrecrime: Precrime address can be set by the owner.
- setMinDstGas: Gas value can be set by the owner.
- setPayloadSizeLimit: Payload size limit can be set by the owner.

Ownable.sol

- renounce Ownership: Deleting ownership will leave the contract without an owner, removing any owner-only functionality.
- transferOwnership: The 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.

LOVELY INSPECTOR

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 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.

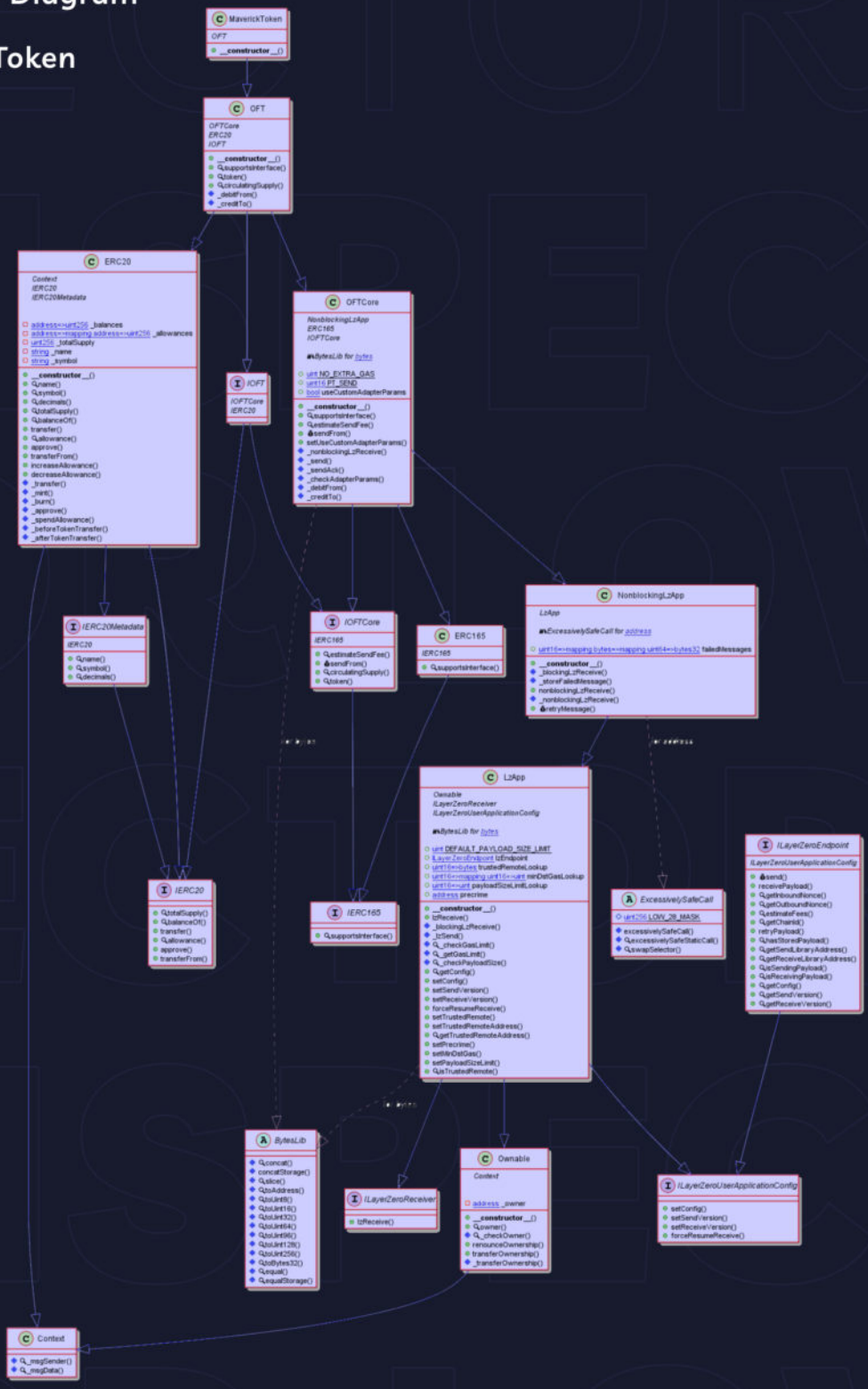
Security state of the reviewed smart contract, based on standard audit procedure scope, is "Secured".

LOVELY INSPECTOR

ADDENDUM

Code Flow Diagram

Maverick Token



LOVELY INSPECTOR

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 >> MaverickToken.sol

```
OFT.constructor(string,string,address)._name (MaverickToken.sol#1578) shadows:
        - ERC20._name (MaverickToken.sol#926) (state variable)
OFT.constructor(string,string,address)._symbol (MaverickToken.sol#1578) shadows:
        - ERC20._symbol (MaverickToken.sol#927) (state variable)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing

LzApp.setPrecrime(address)._precrime (MaverickToken.sol#1372) lacks a zero-check on :
                - precrime = _precrime (MaverickToken.sol#1373)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

Variable 'BytesLib.concatStorage(bytes,bytes).sc_concatStorage_asm_0 (MaverickToken.sol#344)' in BytesLib.concatStorage(bytes,by
tes) (MaverickToken.sol#288-423) potentially used before declaration: sc_concatStorage_asm_0 = keccak256(uint256,uint256)(0x0,0x
20) + slength_concatStorage_asm_0 / 32 (MaverickToken.sol#392)
Variable 'BytesLib.concatStorage(bytes,bytes).submod_concatStorage_asm_0 (MaverickToken.sol#358)' in BytesLib.concatStorage(byte
s,bytes) (MaverickToken.sol#288-423) potentially used before declaration: submod_concatStorage_asm_0 = 32 - slengthmod_concatSto
rage_asm_0 (MaverickToken.sol#401)
Variable 'BytesLib.concatStorage(bytes,bytes).submod_concatStorage_asm_0 (MaverickToken.sol#358)' in BytesLib.concatStorage(byte
s,bytes) (MaverickToken.sol#288-423) potentially used before declaration: mc_concatStorage_asm_0 = _postBytes + submod_concatSto
rage_asm_0 (MaverickToken.sol#402)
Variable 'BytesLib.concatStorage(bytes,bytes).mc_concatStorage_asm_0 (MaverickToken.sol#359)' in BytesLib.concatStorage(bytes,by
tes) (MaverickToken.sol#288-423) potentially used before declaration: mc_concatStorage_asm_0 = _postBytes + submod_concatStorage
_asm_0 (MaverickToken.sol#402)
Variable 'BytesLib.concatStorage(bytes,bytes).end_concatStorage_asm_0 (MaverickToken.sol#360)' in BytesLib.concatStorage(bytes,b
ytes) (MaverickToken.sol#288-423) potentially used before declaration: end_concatStorage_asm_0 = _postBytes + mlength_concatStor
age_asm_0 (MaverickToken.sol#403)
Variable 'BytesLib.concatStorage(bytes,bytes).mask_concatStorage_asm_0 (MaverickToken.sol#361)' in BytesLib.concatStorage(bytes,
bytes) (MaverickToken.sol#288-423) potentially used before declaration: mask_concatStorage_asm_0 = 0x100 ** submod_concatStorage
_asm_0 - 1 (MaverickToken.sol#404)
Variable 'BytesLib.concatStorage(bytes,bytes).submod_concatStorage_asm_0 (MaverickToken.sol#358)' in BytesLib.concatStorage(byte
s,bytes) (MaverickToken.sol#288-423) potentially used before declaration: mask_concatStorage_asm_0 = 0x100 ** submod_concatStora
ge_asm_0 - 1 (MaverickToken.sol#404)
Variable 'BytesLib.concatStorage(bytes,bytes).sc_concatStorage_asm_0 (MaverickToken.sol#344)' in BytesLib.concatStorage(bytes,by
tes) (MaverickToken.sol#288-423) potentially used before declaration: sstore(uint256,uint256)(sc_concatStorage_asm_0,sload(uint2
56)(sc_concatStorage_asm_0) + mload(uint256)(mc_concatStorage_asm_0) & mask_concatStorage_asm_0) (MaverickToken.sol#406)
tes) (MaverickToken.sol#288-423) potentially used before declaration: sstore(uint256,uint256)(sc_concatStorage_asm_0,mload(uint2
56)(mc_concatStorage_asm_0)) (MaverickToken.sol#415)
Variable 'BytesLib.concatStorage(bytes,bytes).sc_concatStorage_asm_0 (MaverickToken.sol#344)' in BytesLib.concatStorage(bytes,by
tes) (MaverickToken.sol#288-423) potentially used before declaration: sstore(uint256,uint256)(sc_concatStorage_asm_0,mload(uint2
56)(mc_concatStorage_asm_0)) (MaverickToken.sol#415)
Variable 'BytesLib.concatStorage(bytes,bytes).sc_concatStorage_asm_0 (MaverickToken.sol#344)' in BytesLib.concatStorage(bytes,by
tes) (MaverickToken.sol#288-423) potentially used before declaration: sc_concatStorage_asm_0 = sc_concatStorage_asm_0 + 1 (Maver
ickToken.sol#412)
Variable 'BytesLib.concatStorage(bytes,bytes).mc_concatStorage_asm_0 (MaverickToken.sol#359)' in BytesLib.concatStorage(bytes,by
tes) (MaverickToken.sol#288-423) potentially used before declaration: mc_concatStorage_asm_0 = mc_concatStorage_asm_0 + 0x20 (Ma
verickToken.sol#413)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#pre-declaration-usage-of-local-variables

Reentrancy in OFTCore._send(address,uint16,bytes,uint256,address,address,bytes) (MaverickToken.sol#1542-1551):
        External calls:
        - _lzSend(_dstChainId,lzPayload,_refundAddress,_zroPaymentAddress,_adapterParams,msg.value) (MaverickToken.sol#1548)
                - lzEndpoint.send{value: _nativeFee}(_dstChainId,trustedRemote,_payload,_refundAddress,_zroPaymentAddress,_adapt
erParams) (MaverickToken.sol#1307)
        Event emitted after the call(s):
        - SendToChain(_dstChainId,_from,_toAddress,amount) (MaverickToken.sol#1550)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

BytesLib.concat(bytes,bytes) (MaverickToken.sol#210-286) uses assembly
        - INLINE ASM (MaverickToken.sol#220-283)
BytesLib.concatStorage(bytes,bytes) (MaverickToken.sol#288-423) uses assembly
        - INLINE ASM (MaverickToken.sol#289-422)
BytesLib.slice(bytes,uint256,uint256) (MaverickToken.sol#425-492) uses assembly
        - INLINE ASM (MaverickToken.sol#439-489)
BytesLib.toAddress(bytes,uint256) (MaverickToken.sol#494-503) uses assembly
        - INLINE ASM (MaverickToken.sol#498-500)
BytesLib.toUint8(bytes,uint256) (MaverickToken.sol#505-514) uses assembly
        - INLINE ASM (MaverickToken.sol#509-511)
BytesLib.toUint16(bytes,uint256) (MaverickToken.sol#516-525) uses assembly
        - INLINE ASM (MaverickToken.sol#520-522)
BytesLib.toUint32(bytes,uint256) (MaverickToken.sol#527-536) uses assembly
```

LOVELY INSPECTOR

```
BytesLib.toUint32(bytes,uint256) (MaverickToken.sol#527-536) uses assembly
        - INLINE ASM (MaverickToken.sol#531-533)
BytesLib.toUint64(bytes,uint256) (MaverickToken.sol#538-547) uses assembly
        - INLINE ASM (MaverickToken.sol#542-544)
BytesLib.toUint96(bytes,uint256) (MaverickToken.sol#549-558) uses assembly
        - INLINE ASM (MaverickToken.sol#553-555)
BytesLib.toUint128(bytes,uint256) (MaverickToken.sol#560-569) uses assembly
        - INLINE ASM (MaverickToken.sol#564-566)
BytesLib.toUint256(bytes,uint256) (MaverickToken.sol#571-580) uses assembly
        - INLINE ASM (MaverickToken.sol#575-577)
BytesLib.toBytes32(bytes,uint256) (MaverickToken.sol#582-591) uses assembly
        - INLINE ASM (MaverickToken.sol#586-588)
BytesLib.equal(bytes,bytes) (MaverickToken.sol#593-634) uses assembly
        - INLINE ASM (MaverickToken.sol#596-631)
BytesLib.equalStorage(bytes,bytes) (MaverickToken.sol#636-706) uses assembly
        - INLINE ASM (MaverickToken.sol#646-703)
ExcessivelySafeCall.excessivelySafeCall(address,uint256,uint16,bytes) (MaverickToken.sol#729-764) uses assembly
        - INLINE ASM (MaverickToken.sol#743-762)
ExcessivelySafeCall.excessivelySafeStaticCall(address,uint256,uint16,bytes) (MaverickToken.sol#781-815) uses assembly
        - INLINE ASM (MaverickToken.sol#795-813)
ExcessivelySafeCall.swapSelector(bytes4,bytes) (MaverickToken.sol#826-841) uses assembly
        - INLINE ASM (MaverickToken.sol#832-840)
LzApp._getGasLimit(bytes) (MaverickToken.sol#1317-1322) uses assembly
        - INLINE ASM (MaverickToken.sol#1319-1321)
OFTCore._nonblockingLzReceive(uint16,bytes,uint64,bytes) (MaverickToken.sol#1529-1540) uses assembly
        - INLINE ASM (MaverickToken.sol#1531-1533)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

BytesLib.concat(bytes,bytes) (MaverickToken.sol#210-286) is never used and should be removed
BytesLib.concatStorage(bytes,bytes) (MaverickToken.sol#288-423) is never used and should be removed
BytesLib.equal(bytes,bytes) (MaverickToken.sol#593-634) is never used and should be removed
BytesLib.equalStorage(bytes,bytes) (MaverickToken.sol#636-706) is never used and should be removed
BytesLib.toBytes32(bytes,uint256) (MaverickToken.sol#582-591) is never used and should be removed
BytesLib.toUint128(bytes,uint256) (MaverickToken.sol#560-569) is never used and should be removed
BytesLib.toUint16(bytes,uint256) (MaverickToken.sol#516-525) is never used and should be removed
BytesLib.toUint256(bytes,uint256) (MaverickToken.sol#571-580) is never used and should be removed
BytesLib.toUint32(bytes,uint256) (MaverickToken.sol#527-536) is never used and should be removed
BytesLib.toUint64(bytes,uint256) (MaverickToken.sol#538-547) is never used and should be removed
BytesLib.toUint8(bytes,uint256) (MaverickToken.sol#505-514) is never used and should be removed
BytesLib.toUint96(bytes,uint256) (MaverickToken.sol#549-558) is never used and should be removed
Context._msgData() (MaverickToken.sol#849-851) is never used and should be removed
ExcessivelySafeCall.excessivelySafeStaticCall(address,uint256,uint16,bytes) (MaverickToken.sol#781-815) is never used and should
 be removed
ExcessivelySafeCall.swapSelector(bytes4,bytes) (MaverickToken.sol#826-841) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version>=0.5.0 (MaverickToken.sol#5) allows old versions
solc-0.8.0 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Parameter BytesLib.concat(bytes,bytes)._preBytes (MaverickToken.sol#211) is not in mixedCase
Parameter BytesLib.concat(bytes,bytes)._postBytes (MaverickToken.sol#212) is not in mixedCase
Parameter BytesLib.concatStorage(bytes,bytes)._preBytes (MaverickToken.sol#288) is not in mixedCase
Parameter BytesLib.concatStorage(bytes,bytes)._postBytes (MaverickToken.sol#288) is not in mixedCase
Parameter BytesLib.slice(bytes,uint256,uint256)._bytes (MaverickToken.sol#426) is not in mixedCase
Parameter BytesLib.slice(bytes,uint256,uint256)._start (MaverickToken.sol#427) is not in mixedCase
Parameter BytesLib.slice(bytes,uint256,uint256)._length (MaverickToken.sol#428) is not in mixedCase
Parameter BytesLib.toAddress(bytes,uint256)._bytes (MaverickToken.sol#494) is not in mixedCase
Parameter BytesLib.toAddress(bytes,uint256)._start (MaverickToken.sol#494) is not in mixedCase
Parameter BytesLib.toUint8(bytes,uint256)._bytes (MaverickToken.sol#505) is not in mixedCase
Parameter BytesLib.toUint8(bytes,uint256)._start (MaverickToken.sol#505) is not in mixedCase
Parameter BytesLib.toUint16(bytes,uint256)._bytes (MaverickToken.sol#516) is not in mixedCase
Parameter BytesLib.toUint16(bytes,uint256)._start (MaverickToken.sol#516) is not in mixedCase
Parameter BytesLib.toUint32(bytes,uint256)._bytes (MaverickToken.sol#527) is not in mixedCase
Parameter BytesLib.toUint32(bytes,uint256)._start (MaverickToken.sol#527) is not in mixedCase

Parameter NonblockingLzApp.nonblockingLzReceive(uint16,bytes,uint64,bytes)._nonce (MaverickToken.sol#1420) is not in mixedCase
Parameter NonblockingLzApp.nonblockingLzReceive(uint16,bytes,uint64,bytes)._payload (MaverickToken.sol#1420) is not in mixedCase
Parameter NonblockingLzApp.retryMessage(uint16,bytes,uint64,bytes)._srcChainId (MaverickToken.sol#1429) is not in mixedCase
Parameter NonblockingLzApp.retryMessage(uint16,bytes,uint64,bytes)._srcAddress (MaverickToken.sol#1429) is not in mixedCase
Parameter NonblockingLzApp.retryMessage(uint16,bytes,uint64,bytes)._nonce (MaverickToken.sol#1429) is not in mixedCase
Parameter NonblockingLzApp.retryMessage(uint16,bytes,uint64,bytes)._payload (MaverickToken.sol#1429) is not in mixedCase
Parameter OFTCore.estimateSendFee(uint16,bytes,uint256,bool,bytes)._dstChainId (MaverickToken.sol#1514) is not in mixedCase
Parameter OFTCore.estimateSendFee(uint16,bytes,uint256,bool,bytes)._toAddress (MaverickToken.sol#1514) is not in mixedCase
Parameter OFTCore.estimateSendFee(uint16,bytes,uint256,bool,bytes)._amount (MaverickToken.sol#1514) is not in mixedCase
Parameter OFTCore.estimateSendFee(uint16,bytes,uint256,bool,bytes)._useZro (MaverickToken.sol#1514) is not in mixedCase
Parameter OFTCore.estimateSendFee(uint16,bytes,uint256,bool,bytes)._adapterParams (MaverickToken.sol#1514) is not in mixedCase
Parameter OFTCore.sendFrom(address,uint16,bytes,uint256,address,address,bytes)._from (MaverickToken.sol#1520) is not in mixedCas
e
Parameter OFTCore.sendFrom(address,uint16,bytes,uint256,address,address,bytes)._dstChainId (MaverickToken.sol#1520) is not in mi
xedCase
Parameter OFTCore.sendFrom(address,uint16,bytes,uint256,address,address,bytes)._toAddress (MaverickToken.sol#1520) is not in mix
edCase
Parameter OFTCore.sendFrom(address,uint16,bytes,uint256,address,address,bytes)._amount (MaverickToken.sol#1520) is not in mixedC
ase
Parameter OFTCore.sendFrom(address,uint16,bytes,uint256,address,address,bytes)._refundAddress (MaverickToken.sol#1520) is not in
 mixedCase
Parameter OFTCore.sendFrom(address,uint16,bytes,uint256,address,address,bytes)._zroPaymentAddress (MaverickToken.sol#1520) is no
t in mixedCase
Parameter OFTCore.sendFrom(address,uint16,bytes,uint256,address,address,bytes)._adapterParams (MaverickToken.sol#1520) is not in
 mixedCase
Parameter OFTCore.setUseCustomAdapterParams(bool)._useCustomAdapterParams (MaverickToken.sol#1524) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

BytesLib.toAddress(bytes,uint256) (MaverickToken.sol#494-503) uses literals with too many digits:
        - tempAddress = mload(uint256)(_bytes + 0x20 + _start) / 0x1000000000000000000000000 (MaverickToken.sol#499)
ExcessivelySafeCall.slitherConstructorConstantVariables() (MaverickToken.sol#710-842) uses literals with too many digits:
        - LOW_28_MASK = 0x00000000ffffffffffffffffffffffffffffffffffffffffffffffffffffffff (MaverickToken.sol#711-712)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
MaverickToken.sol analyzed (19 contracts with 84 detectors), 158 result(s) found
root@server:/chetan/gaza/mycontracts/Akiyo/p2#
```

LOVELY INSPECTOR

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.

MaverickToken.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: 1531:8:

Gas costs:

Gas requirement of function OFT.supportsInterface 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: 1580:4:

This on local calls:

Use of "this" for local functions: Never use "this" to call functions in the same contract, it only consumes more gas than normal local calls.
more
Pos: 1408:119:

LOVELY INSPECTOR

Constant/View/Pure functions:

ExcessivelySafeCall.excessivelySafeStaticCall(address,uint256,uint16,bytes) : Is constant but potentially should not be. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 781:4:

Similar variable names:

ERC20._burn(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.
Pos: 1173:28:

No return:

ILayerZeroEndpoint.getSendLibraryAddress(address): Defines a return type but never explicitly returns a value.
Pos: 179:4:

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.
more
Pos: 1313:8:

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.

MaverickToken.sol

```
Compiler version >=0.5.0 does not satisfy the ^0.5.8 semver requirement
Pos: 1:4
Avoid using inline assembly. It is acceptable only in rare cases
Pos: 9:585
Avoid using inline assembly. It is acceptable only in rare cases
Pos: 9:595
Avoid using inline assembly. It is acceptable only in rare cases
Pos: 9:645
Explicitly mark visibility of state
Pos: 5:710
Avoid using inline assembly. It is acceptable only in rare cases
Pos: 9:742
Avoid using inline assembly. It is acceptable only in rare cases
Pos: 9:794
Provide an error message for require
Pos: 9:829
Avoid using inline assembly. It is acceptable only in rare cases
Pos: 9:831
Explicitly mark visibility in function (Set ignoreConstructors to true if using solidity >=0.7.0)
Pos: 5:861
Error message for require is too long
Pos: 9:903
Explicitly mark visibility in function (Set ignoreConstructors to true if using solidity >=0.7.0)
Pos: 5:934
Error message for require is too long
Pos: 9:1080
Error message for require is too long
Pos: 9:1103
Error message for require is too long
Pos: 9:1104
Error message for require is too long
Pos: 9:1109
Error message for require is too long
Pos: 9:1158
Error message for require is too long
Pos: 9:1163
Error message for require is too long
Pos: 9:1189
Error message for require is too long
Pos: 9:1190
Code contains empty blocks
Pos: 94:1228
```

LOVELY INSPECTOR

```
Code contains empty blocks
Pos: 93:1244
Explicitly mark visibility in function (Set ignoreConstructors to true if using solidity >=0.7.0)
Pos: 5:1284
Error message for require is too long
Pos: 9:1294
Error message for require is too long
Pos: 9:1304
Check result of "send" call
Pos: 9:1306
Avoid using inline assembly. It is acceptable only in rare cases
Pos: 9:1318
Explicitly mark visibility in function (Set ignoreConstructors to true if using solidity >=0.7.0)
Pos: 5:1398
Code contains empty blocks
Pos: 53:1398
Error message for require is too long
Pos: 9:1421
Error message for require is too long
Pos: 9:1431
Error message for require is too long
Pos: 9:1432
Code contains empty blocks
Pos: 1:1492
Explicitly mark visibility in function (Set ignoreConstructors to true if using solidity >=0.7.0)
Pos: 5:1507
Code contains empty blocks
Pos: 68:1507
Avoid using inline assembly. It is acceptable only in rare cases
Pos: 9:1530
Error message for require is too long
Pos: 13:1565
Explicitly mark visibility in function (Set ignoreConstructors to true if using solidity >=0.7.0)
Pos: 5:1577
Code contains empty blocks
Pos: 125:1577
Explicitly mark visibility in function (Set ignoreConstructors to true if using solidity >=0.7.0)
Pos: 5:1606
```

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