INSPECTOR
LOVELY

SMART CONTRACT

SECURITY AUDIT

MEMECOIN

inspector.lovely.finance

INSPECTOR LOVELY

TABLE OF CONTENTS

INSPECTOR
LOVELY

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.

INSPECTOR LOVELY

AUDIT SCOPE

Name	Code Review and Security Analysis Report for Memecoin Coin Smart Contract
Platform	Ethereum
File	Memecoin.sol
File MD5 Hash	143C43BE0550CD5E639572F2FF8C4D77
Smart Contract Code	0xb131f4a55907b10d1f0a50d8ab8fa09ec342cd74
Audit Date	November 8th, 2023

INSPECTOR LOVELY

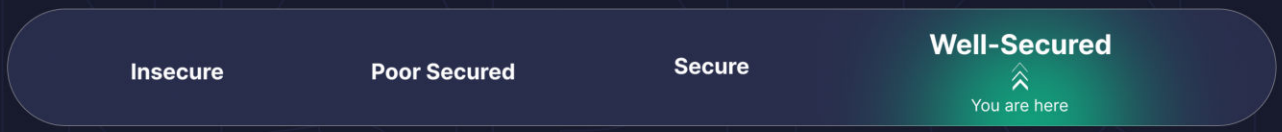
PROPOSED SMART CONTRACT FEATURES

Claimed Feature Detail	Our Observation
Tokenomics: • Name: Memecoin • Symbol: MEME • Decimals: 18 • Total Supply: 69 Billion	Validated
Owner Control: • The new TokenPool address can be set by the owner • Current owner can transfer the ownership. • Owner can renounce ownership	Validated

INSPECTOR LOVELY

AUDIT SUMMARY

According to the standard audit assessment, the 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. 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 and 0 low, and 3 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.

INSPECTOR LOVELY

KEY TECHNICAL METRICS

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

INSPECTOR LOVELY

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 Memecoin 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 Memecoin.

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 Memecoin 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 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.

INSPECTOR LOVELY

AS-IS OVERVIEW

Functions

SL.	FUNCTIONS	TYPE	OBSERVATION	CONCLUSION
1	constructor	write	Passed	No Issue
2	permit	write	Passed	No Issue
3	setTokenPool	external	access only Owner	No Issue
4	permit	write	Passed	No Issue
5	nonces	read	Passed	No Issue
6	DOMAIN_SEPARATOR	external	Passed	No Issue
7	_useNonce	internal	Passed	No Issue
8	name	read	Passed	No Issue
9	symbol	read	Passed	No Issue
10	decimals	read	Passed	No Issue
11	totalSupply	read	Passed	No Issue
12	balanceOf	read	Passed	No Issue
13	transfer	write	Passed	No Issue
14	allowance	read	Passed	No Issue
15	approve	write	Passed	No Issue
16	transferFrom	write	Passed	No Issue
17	increaseAllowance	write	Passed	No Issue
18	decreaseAllowance	write	Passed	No Issue
19	_transfer	internal	Passed	No Issue
20	_mint	internal	Passed	No Issue
21	_burn	internal	Passed	No Issue
22	_approve	internal	Passed	No Issue
23	_spendAllowance	internal	Passed	No Issue
24	_beforeTokenTransfer	internal	Passed	No Issue
25	_afterTokenTransfer	internal	Passed	No Issue
26	onlyOwner	modifier	Passed	No Issue
27	owner	write	Passed	No Issue
28	_checkOwner	internal	Passed	No Issue
29	renounceOwnership	write	access only Owner	No Issue

AS-IS OVERVIEW

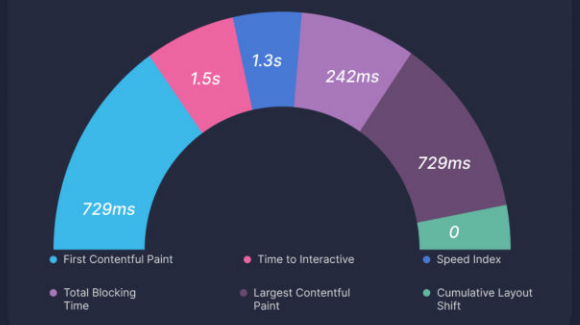
Functions

SL.	FUNCTIONS	TYPE	OBSERVATION	CONCLUSION
30	transferOwnership	write	access only Owner	No Issue
31	_transferOwnership	internal	Passed	No Issue
32	_domainSeparatorV4	internal	Passed	No Issue
33	_buildDomainSeparator	read	Passed	No Issue
34	_hashTypedDataV4	internal	Passed	No Issue
35	eip712Domain	read	Passed	No Issue

INSPECTOR LOVELY

PROJECT WEBSITE PERFORMANCE AUDIT

Performance Metrics



Browser Timings

Redirect Duration **0ms**	Connection Duration **44ms**	Backend Duration **35ms**
Time to First Byte **79ms**	First Paint **102ms**	DOM Interactive Time **334ms**
DOM Content Loaded **334ms**	Onload Time **729ms**	Fully Loaded Time **2.5s**

Grade

A	Performance 89%	Structure 100%

Web Vitals

LCP	TBT	CLS
729ms	242ms	0

Top Issues

IMPECT	AUDIT
High	Reduce Java Script execution time (TBT) — 829ms spent executing JavaScript

URL	TOTAL CPU TIME	SCRIPT EVALUATION	SCRIPT PARSE
• https://www.memecoin.org/ next/static/chunks/pages/ app-02245f6a6038b5b7.0	6.6s	247ms	17ms
• Unattributable	388ms	10ms	Orns
• https://www.memecoin.org/ next/stalm/chunks/3732.6bc2d5f2afa61747.0	256ms	228ms	8ms
• https://www.memecoin.org/ nexUstaticichunks/framework-f29e4Sae95caeSaajs	231ms	184ms	1ms
• https://www.memecoin.orgl	94ms	7ms	Orns
• https://www.memecoin.org/ nexUstatic/chunks/pages/claim-2cea65b67a33d3b4.0	67ms	64ms	3ms
• https://www.memecoin.org/ nexUstalm/chunks/6221.75c0b80ef694b6131.0	54ms	47ms	6ms

Level of Criticality

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
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.

INSPECTOR LOVELY

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:	

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:

Memecoin.sol

- setTokenPool: The new TokenPool address can be set by the owner.

Ownable.sol

- checkOwner: Throws if the sender is not the owner.
- 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.

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 observed 3 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.

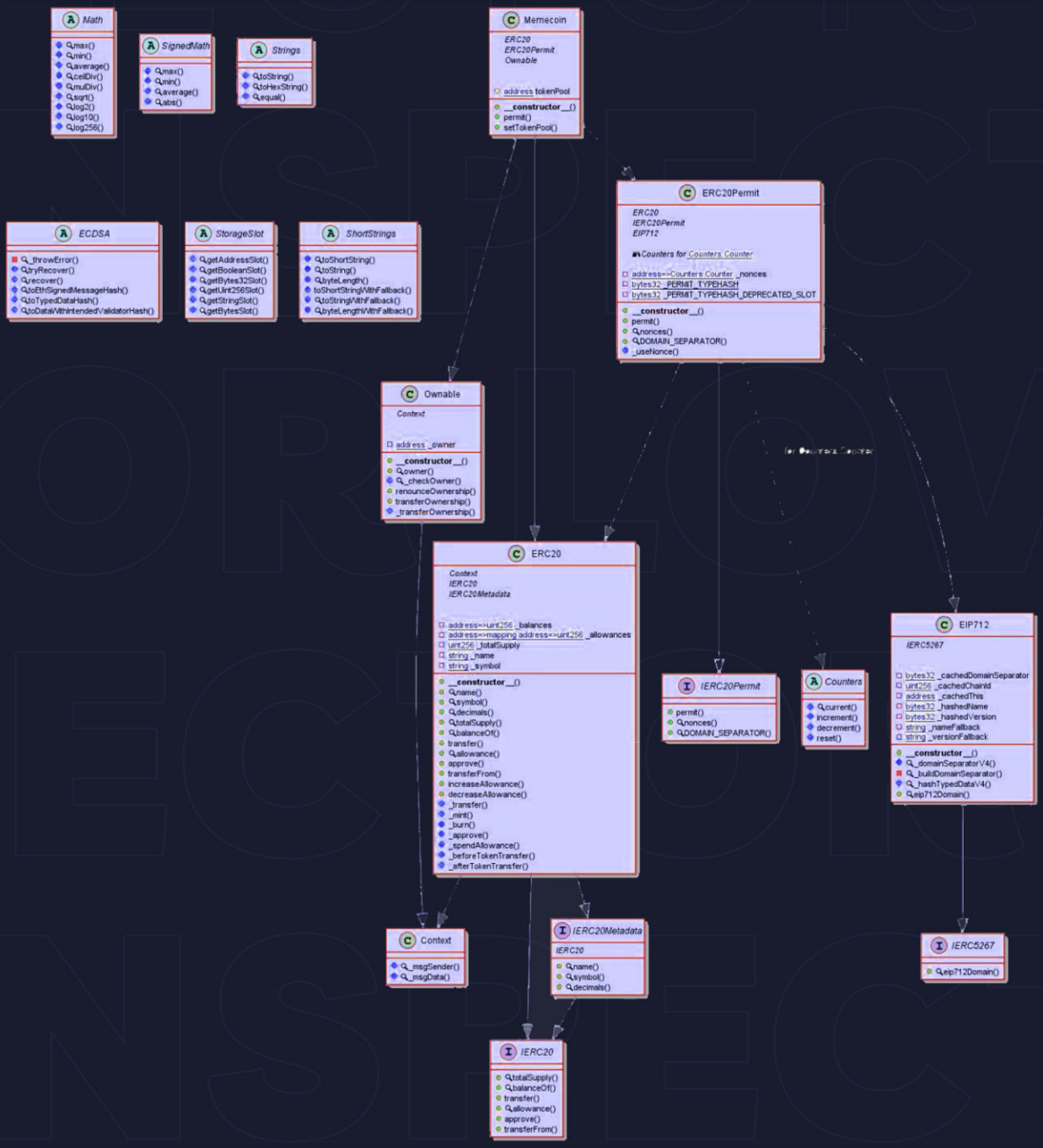
Audit report contains all found security vulnerabilities and other issues in the reviewed code.

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

INSPECTOR LOVELY

ADDENDUM

Code Flow Diagram - Memecoin



INSPECTOR LOVELY

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 >> Memecoin.sol

```
ERC20Permit.constructor(string).name (Memecoin.sol#849) shadows:
        - ERC20.name() (Memecoin.sol#708-710) (function)
        - IERC20Metadata.name() (Memecoin.sol#305) (function)
Memecoin.constructor(string,string,uint256,address).name (Memecoin.sol#894) shadows:
        - ERC20.name() (Memecoin.sol#708-710) (function)
        - IERC20Metadata.name() (Memecoin.sol#305) (function)
Memecoin.constructor(string,string,uint256,address).symbol (Memecoin.sol#894) shadows:
        - ERC20.symbol() (Memecoin.sol#712-714) (function)
        - IERC20Metadata.symbol() (Memecoin.sol#307) (function)
Memecoin.permit(address,address,uint256,uint256,uint8,bytes32,bytes32).owner (Memecoin.sol#902) shadows:
        - Ownable.owner() (Memecoin.sol#668-670) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing

Memecoin.setTokenPool(address)._tokenPool (Memecoin.sol#910) lacks a zero-check on :
                - tokenPool = _tokenPool (Memecoin.sol#911)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

ERC20Permit.permit(address,address,uint256,uint256,uint8,bytes32,bytes32) (Memecoin.sol#851-870) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp <= deadline,ERC20Permit: expired deadline) (Memecoin.sol#860)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

Math.mulDiv(uint256,uint256,uint256) (Memecoin.sol#27-76) uses assembly
        - INLINE ASM (Memecoin.sol#31-35)
        - INLINE ASM (Memecoin.sol#45-50)
        - INLINE ASM (Memecoin.sol#54-60)
Strings.toString(uint256) (Memecoin.sol#344-362) uses assembly
        - INLINE ASM (Memecoin.sol#349-351)
        - INLINE ASM (Memecoin.sol#354-356)
ECDSA.tryRecover(bytes32,bytes) (Memecoin.sol#416-430) uses assembly
        - INLINE ASM (Memecoin.sol#421-425)
ECDSA.toEthSignedMessageHash(bytes32) (Memecoin.sol#469-475) uses assembly
        - INLINE ASM (Memecoin.sol#470-474)
ECDSA.toTypedDataHash(bytes32,bytes32) (Memecoin.sol#481-489) uses assembly
```


```
ERC20Permit.constructor(string).name (Memecoin.sol#849) shadows:
        - ERC20.name() (Memecoin.sol#708-710) (function)
        - IERC20Metadata.name() (Memecoin.sol#305) (function)
Memecoin.constructor(string,string,uint256,address).name (Memecoin.sol#894) shadows:
        - ERC20.name() (Memecoin.sol#708-710) (function)
        - IERC20Metadata.name() (Memecoin.sol#305) (function)
Memecoin.constructor(string,string,uint256,address).symbol (Memecoin.sol#894) shadows:
        - ERC20.symbol() (Memecoin.sol#712-714) (function)
        - IERC20Metadata.symbol() (Memecoin.sol#307) (function)
Memecoin.permit(address,address,uint256,uint256,uint8,bytes32,bytes32).owner (Memecoin.sol#902) shadows:
        - Ownable.owner() (Memecoin.sol#668-670) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing

Memecoin.setTokenPool(address)._tokenPool (Memecoin.sol#910) lacks a zero-check on :
                - tokenPool = _tokenPool (Memecoin.sol#911)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

ERC20Permit.permit(address,address,uint256,uint256,uint8,bytes32,bytes32) (Memecoin.sol#851-870) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp <= deadline,ERC20Permit: expired deadline) (Memecoin.sol#860)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
```

```
Math.mulDiv(uint256,uint256,uint256) (Memecoin.sol#27-76) uses assembly
        - INLINE ASM (Memecoin.sol#31-35)
        - INLINE ASM (Memecoin.sol#45-50)
        - INLINE ASM (Memecoin.sol#54-60)
Strings.toString(uint256) (Memecoin.sol#344-362) uses assembly
        - INLINE ASM (Memecoin.sol#349-351)
        - INLINE ASM (Memecoin.sol#354-356)
ECDSA.tryRecover(bytes32,bytes) (Memecoin.sol#416-430) uses assembly
        - INLINE ASM (Memecoin.sol#421-425)
ECDSA.toEthSignedMessageHash(bytes32) (Memecoin.sol#469-475) uses assembly
        - INLINE ASM (Memecoin.sol#470-474)
ECDSA.toTypedDataHash(bytes32,bytes32) (Memecoin.sol#481-489) uses assembly
```

```
Math.ceilDiv(uint256,uint256) (Memecoin.sol#23-25) is never used and should be removed
Math.log10(uint256) (Memecoin.sol#157-189) is never used and should be removed
Math.log10(uint256,Math.Rounding) (Memecoin.sol#191-196) is never used and should be removed
Math.log2(uint256) (Memecoin.sol#112-148) is never used and should be removed
Math.log2(uint256,Math.Rounding) (Memecoin.sol#150-155) is never used and should be removed
Math.log256(uint256) (Memecoin.sol#198-222) is never used and should be removed
Math.log256(uint256,Math.Rounding) (Memecoin.sol#224-229) is never used and should be removed
Math.max(uint256,uint256) (Memecoin.sol#11-13) is never used and should be removed
Math.min(uint256,uint256) (Memecoin.sol#15-17) is never used and should be removed
Math.mulDiv(uint256,uint256,uint256) (Memecoin.sol#27-76) is never used and should be removed
Math.mulDiv(uint256,uint256,uint256,Math.Rounding) (Memecoin.sol#78-84) is never used and should be removed
Math.sqrt(uint256) (Memecoin.sol#86-103) is never used and should be removed
Math.sqrt(uint256,Math.Rounding) (Memecoin.sol#105-110) is never used and should be removed
SignedMath.abs(int256) (Memecoin.sol#246-250) is never used and should be removed
SignedMath.average(int256,int256) (Memecoin.sol#241-244) is never used and should be removed
SignedMath.max(int256,int256) (Memecoin.sol#233-235) is never used and should be removed
SignedMath.min(int256,int256) (Memecoin.sol#237-239) is never used and should be removed
StorageSlot.getAddressSlot(bytes32) (Memecoin.sol#521-525) is never used and should be removed
StorageSlot.getBooleanSlot(bytes32) (Memecoin.sol#527-531) is never used and should be removed
StorageSlot.getBytes32Slot(bytes32) (Memecoin.sol#533-537) is never used and should be removed
StorageSlot.getBytesSlot(bytes) (Memecoin.sol#563-567) is never used and should be removed
StorageSlot.getBytesSlot(bytes32) (Memecoin.sol#557-561) is never used and should be removed
StorageSlot.getStringSlot(bytes32) (Memecoin.sol#545-549) is never used and should be removed
StorageSlot.getStringSlot(string) (Memecoin.sol#551-555) is never used and should be removed
StorageSlot.getUint256Slot(bytes32) (Memecoin.sol#539-543) is never used and should be removed
Strings.equal(string,string) (Memecoin.sol#390-392) is never used and should be removed
Strings.toHexString(address) (Memecoin.sol#386-388) is never used and should be removed
Strings.toHexString(uint256) (Memecoin.sol#368-372) is never used and should be removed
Strings.toHexString(uint256,uint256) (Memecoin.sol#374-384) is never used and should be removed
Strings.toString(int256) (Memecoin.sol#364-366) is never used and should be removed
Strings.toString(uint256) (Memecoin.sol#344-362) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
```

```
Pragma version^0.8.0 (Memecoin.sol#2) 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

Function IERC20Permit.DOMAIN_SEPARATOR() (Memecoin.sol#284) is not in mixedCase
Function ERC20Permit.DOMAIN_SEPARATOR() (Memecoin.sol#876-878) is not in mixedCase
Variable ERC20Permit._PERMIT_TYPEHASH_DEPRECATED_SLOT (Memecoin.sol#847) is not in mixedCase
Parameter Memecoin.setTokenPool(address)._tokenPool (Memecoin.sol#910) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

ShortStrings.slitherConstructorConstantVariables() (Memecoin.sol#571-580) uses literals with too many digits:
        - _FALLBACK_SENTINEL = 0x00000000000000000000000000000000000000000000000000000000000000FF (Memecoin.sol#573)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
Memecoin.sol analyzed (17 contracts with 84 detectors), 77 result(s) found
```

INSPECTOR LOVELY

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.

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: 1012:15:

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: 1704:23:

Gas costs:

Gas requirement of function Memecoin.permit 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: 1761:11:

Similar variable names:

Memecoin.(string,string,uint256,address) : Variables have very similar names "_totalSupply" and "totalSupply_". Note: Modifiers are currently not considered by this static analysis.
Pos: 1756:31:

No return:

StorageSlot.getBytesSlot(bytes): Defines a return type but never explicitly returns a value.
Pos: 946:11:

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: 1704:15:

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: 1711:15:

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: 181:35:

INSPECTOR LOVELY

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.

Memecoin.sol

```
Compiler version ^0.8.0 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:3
Avoid using inline assembly. It is acceptable only in rare cases Pos: 13:61
Avoid using inline assembly. It is acceptable only in rare cases Pos: 13:84
Avoid using inline assembly. It is acceptable only in rare cases Pos: 13:98
Avoid using inline assembly. It is acceptable only in rare cases Pos: 13:580
Avoid using inline assembly. It is acceptable only in rare cases Pos: 17:586
Error message for revert is too long Pos: 13:659
Avoid using inline assembly. It is acceptable only in rare cases Pos: 13:691
Avoid using inline assembly. It is acceptable only in rare cases Pos: 9:797
Avoid using inline assembly. It is acceptable only in rare cases Pos: 9:827
Avoid using inline assembly. It is acceptable only in rare cases Pos: 9:877
Avoid using inline assembly. It is acceptable only in rare cases Pos: 9:887
Avoid using inline assembly. It is acceptable only in rare cases Pos: 9:897
Avoid using inline assembly. It is acceptable only in rare cases Pos: 9:907
Avoid using inline assembly. It is acceptable only in rare cases Pos: 9:917
Avoid using inline assembly. It is acceptable only in rare cases Pos: 9:927
Avoid using inline assembly. It is acceptable only in rare cases Pos: 9:937
Avoid using inline assembly. It is acceptable only in rare cases Pos: 9:947
Avoid using inline assembly. It is acceptable only in rare cases Pos: 9:1011
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:1124
Explicitly mark visibility in function (Set ignoreConstructors to
```

true if using solidity >=0.7.0)
Pos: 5:1242
Error message for require is too long Pos: 9:1284
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:1344
Error message for require is too long Pos: 9:1490
Error message for require is too long Pos: 9:1513
Error message for require is too long Pos: 9:1514
Error message for require is too long Pos: 9:1519
Error message for require is too long Pos: 9:1568
Error message for require is too long Pos: 9:1573
Error message for require is too long Pos: 9:1599
Error message for require is too long Pos: 9:1600
Code contains empty blocks
Pos: 94:1638
Code contains empty blocks
Pos: 93:1654
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:1689
Code contains empty blocks
Pos: 55:1689
Avoid making time-based decisions in your business logic Pos: 17:1703
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)

Software analysis result:

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

INSPECTOR
LOVELY

INSPECTOR LOVELY

INFO

Website: Inspector.lovely.finance

Telegram community: t.me/inspectorlovely

Twitter: twitter.com/InspectorLovely

inspector.lovely.finance