

etherauthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project: Heroes of Mavia
Website: mavia.com
Platform: Base Chain Network
Language: Solidity
Date: May 29th, 2024

Table of contents

Introduction	4
Project Background	4
Audit Scope	5
Code Audit History	6
Severity Definitions	6
Claimed Smart Contract Features	7
Audit Summary	8
Technical Quick Stats	9
Business Risk Analysis	10
Code Quality	11
Documentation	11
Use of Dependencies	11
AS-IS overview	12
Audit Findings	13
Conclusion	16
Our Methodology	17
Disclaimers	19
Appendix	
• Code Flow Diagram	20
• Slither Results Log	21
• Solidity static analysis	23
• Solhint Linter	25

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.

Introduction

As part of EtherAuthority's community smart contracts audit initiatives, the Heroes of Maviasmart contract from mavia.com was audited extensively. The audit has been performed using manual analysis as well as using automated software tools. This report presents all the findings regarding the audit performed on May 29th, 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

Website Details



Heroes of Mavia is a mobile strategy game where players build and defend bases, command armies, and earn in-game resources like Ruby, Gold, and Oil. The game emphasizes strategic base-building, resource management, and combat with a variety of infantry, vehicle, and air units. Players can also participate in governance through the Mavia DAO and trade items in the Ruby Legendary Marketplace. The game is available on iOS and Android and features high-quality 3D art developed by Skrice Studios.

Code Details

- The provided Solidity contract, `MaviaOFT`, is an implementation of an Omnichain Fungible Token (OFT) using LayerZero's technology for cross-chain interoperability. Here's a detailed explanation of its components and functionality:
- Key Components:
 - Imports:
 - OFT from LayerZero Labs: This import brings in the functionality for Omnichain Fungible Tokens, enabling the token to operate across multiple blockchains.
 - The LayerZero endpoint enables seamless cross-chain communication, allowing tokens to be transferred and utilized on various supported chains.
- This contract is designed to create a token that can be easily used and transferred across different blockchain networks, providing enhanced functionality and flexibility for token holders and developers.

Audit scope

Name	Code Review and Security Analysis Report for Heroes of Mavia Smart Contract
Platform	Base Chain Network
Language	Solidity
File	MaviaOFT.sol
Smart Contract Code	0x24fcFC492C1393274B6bcd568ac9e225BEc93584
Audit Date	May 29th, 2024
Audit Result	Passed

Code Audit History



0
Total Findings

0
Critical






0
High

0
Medium

0
Low

0
Informational

Severity Definitions

0		Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to token loss etc.
0		High	High-level vulnerabilities are difficult to exploit; however, they also have significant impact on smart contract execution, e.g. Public access is crucial.
0		Medium	Medium-level vulnerabilities are important to fix; however, they can't lead to tokens loss
0		Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets, that can't have a significant impact on execution
0		Lowest / Informational / Best Practice	Lowest-level vulnerabilities, code style violations, and info statements can't affect smart contract execution and can be ignored.

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
<p>Tokenomics:</p> <ul style="list-style-type: none"> • Name: Heroes of Mavia • Symbol: MAVIA • Decimals: 18 	<p>YES, This is valid.</p>
<p>Advantages:</p> <ul style="list-style-type: none"> • Cross-Chain Interoperability: By leveraging LayerZero's OFT, `MaviaOFT` can interact with multiple blockchains, facilitating a more versatile token use case. • Ownership Control: Using OpenZeppelin's `Ownable`, the contract ensures that sensitive operations can only be performed by the owner, enhancing security. 	<p>YES, This is valid.</p>
<p>Owner Specifications:</p> <ul style="list-style-type: none"> • Sets the message inspector address for the OFT. • Sets the preCrime contract address. • Sets the enforced options for specific endpoint and message type combinations. • Sets the peer address (OApp instance) for a corresponding endpoint. • Sets the delegate address for the OApp. • The current owner can transfer the ownership. • The owner can renounce ownership. 	<p>YES, This is valid.</p> <p>We advise renouncing ownership once the ownership functions are not needed. This is to make the smart contract 100% decentralized.</p>

Audit Summary

According to the standard audit assessment, the Customer's solidity-based smart contracts are **"Well Secured"**. Also, these contracts contain owner control, which does not make them fully decentralized.

Unsecured

Poor Secured

Secured

Well Secured

You are here



We used various tools like Slither, Solhint, and Remix IDE. At the same time, this finding is based on a critical analysis of the manual audit.

All issues found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the Audit Overview section. The general overview is presented in the AS-IS section and all identified issues can be found in the Audit overview section.

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

Investor Advice: A technical audit of the smart contract does not guarantee the ethical nature of the project. Any owner-controlled functions should be executed by the owner with responsibility. All investors/users are advised to do their due diligence before investing in the project.

Technical Quick Stats

Main Category	Subcategory	Result
Contract Programming	The solidity version is not specified	Passed
	The solidity version is too old	Passed
	Integer overflow/underflow	Passed
	Function input parameters lack check	Passed
	Function input parameters check bypass	Passed
	Function access control lacks management	Passed
	Critical operation lacks event log	Passed
	Human/contract checks bypass	Passed
	Random number generation/use vulnerability	N/A
	Fallback function misuse	Passed
	Race condition	Passed
	Logical vulnerability	Passed
	Features claimed	Passed
	Other programming issues	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**

Business Risk Analysis

Category	Result
● Buy Tax	0%
● Sell Tax	0%
● Cannot Buy	No
● Cannot Sell	No
● Max Tax	0%
● Modify Tax	No
● Fee Check	Not Detected
● Is Honeypot	Not Detected
● Trading Cooldown	Not Detected
● Can Pause Trade?	Not Detected
● Pause Transfer?	No
● Max Tax?	No
● Is it Anti-whale?	Not Detected
● Is Anti-bot?	Not Detected
● Is it a Blacklist?	No
● Blacklist Check	No
● Can Mint?	No
● Is it a Proxy Contract?	No
● Is it used Open Source?	No
● External Call Risk?	No
● Balance Modifiable?	No
● Can Take Ownership?	Yes
● Ownership Renounce?	Yes
● Hidden Owner?	Not Detected
● Self Destruction?	Not Detected
● Auditor Confidence	High

Overall Audit Result: PASSED

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.

Email: audit@EtherAuthority.io

Code Quality

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

The libraries in Heroes of Mavia 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 Heroes of Mavia.

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 Heroes of Mavia smart contract code in the form of a [basescan](#) web link.

As mentioned above, code parts are well commented on. And the logic is straightforward. So it is easy to quickly understand the programming flow as well as complex code logic. Comments are very helpful in understanding the overall architecture of the protocol.

Use of Dependencies

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

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

AS-IS overview

MaviaOFT.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	oftVersion	external	Passed	No Issue
3	token	external	Passed	No Issue
4	approvalRequired	external	Passed	No Issue
5	_debit	internal	Passed	No Issue
6	credit	internal	Passed	No Issue
7	sharedDecimals	write	Passed	No Issue
8	setMsgInspector	write	access only Owner	No Issue
9	quoteOFT	external	Passed	No Issue
10	quoteSend	external	Passed	No Issue
11	send	external	Passed	No Issue
12	_buildMsgAndOptions	internal	Passed	No Issue
13	_lzReceive	internal	Passed	No Issue
14	_lzReceiveSimulate	internal	Passed	No Issue
15	isPeer	internal	Passed	No Issue
16	_removeDust	internal	Passed	No Issue
17	toLD	internal	Passed	No Issue
18	toSD	internal	Passed	No Issue
19	debitView	internal	Passed	No Issue
20	_debit	internal	Passed	No Issue
21	credit	internal	Passed	No Issue

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 which can be executed by the Admin (Owner) only. If the admin wallet's private key would be compromised, then it would create trouble.

Centralized

Decentralized

You are here



The following are Owner functions:

OFTCore.sol

- setMsgInspector: Sets the message inspector address for the OFT.

OAppPreCrimeSimulator.sol

- setPreCrime: Sets the preCrime contract address.

OAppOptionsType3.sol

- setEnforcedOptions: Sets the enforced options for specific endpoint and message type combinations,

OAppCore.sol

- setPeer: Sets the peer address (OApp instance) for a corresponding endpoint.
- setDelegate: Sets the delegate address for the OApp.

Ownable.sol

- transferOwnership: Allows the current owner to transfer control of the contract to a newOwner.
- renounceOwnership: Leaves the contract without the owner. It will not be possible to call `onlyOwner` functions anymore. Can only be called by the current owner.

To make the smart contract 100% decentralized, we suggest renouncing ownership in the smart contract once its function is completed.

Conclusion

We were given a contract code in the form of a [basescan](#) web link. And we have used all possible tests based on given objects as files. We observed no issue 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.

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

The security state of the reviewed smart contract, based on standard audit procedure scope, is **“Well 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 the 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

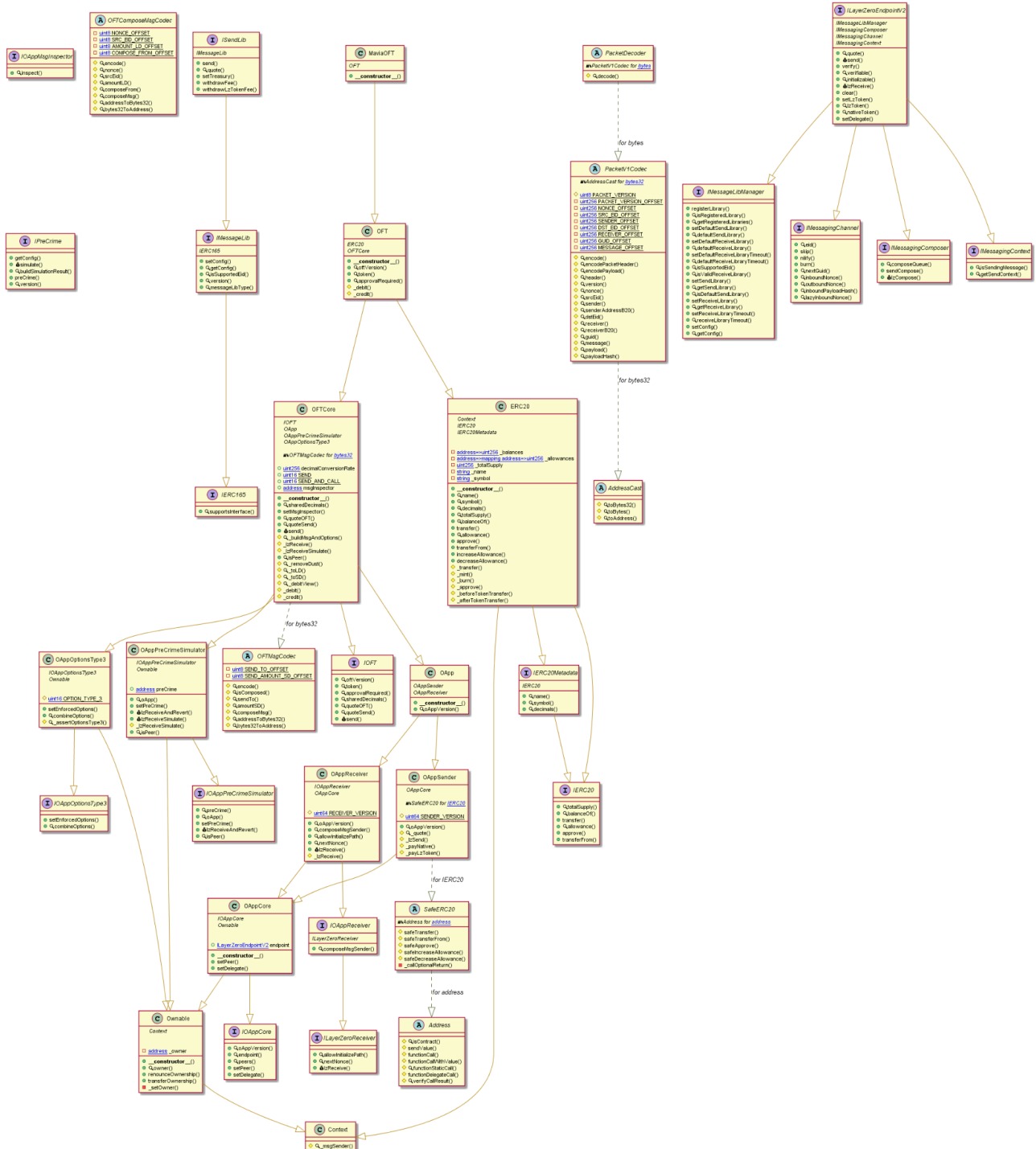
EtherAuthority team has analyzed this smart contract in accordance with the best industry practices at the date of this report, in relation to: cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation,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

Code Flow Diagram - Heroes of Mavia



This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.

Email: audit@EtherAuthority.io

Slither Results Log

Slither is a Solidity static analysis framework that uses vulnerability detectors, displays contract details, and provides an API for writing custom analyses. It helps developers identify vulnerabilities, improve code comprehension, and prototype custom analyses quickly. The analysis includes a report with warnings and errors, allowing developers to quickly prototype and fix issues.

We did the analysis of the project altogether. Below are the results.

MaviaOFT.sol

INFO:Detectors:

OFT.constructor(string,string,address,address)._name (MaviaOFT.sol#1480) shadows:

- ERC20._name (MaviaOFT.sol#859) (state variable)

OFT.constructor(string,string,address,address)._symbol (MaviaOFT.sol#1481) shadows:

- ERC20._symbol (MaviaOFT.sol#860) (state variable)

MaviaOFT.constructor(string,string,address,address)._name (MaviaOFT.sol#1522) shadows:

- ERC20._name (MaviaOFT.sol#859) (state variable)

MaviaOFT.constructor(string,string,address,address)._symbol (MaviaOFT.sol#1523) shadows:

- ERC20._symbol (MaviaOFT.sol#860) (state variable)

MaviaOFT.constructor(string,string,address,address)._owner (MaviaOFT.sol#1525) shadows:

- Ownable._owner (MaviaOFT.sol#818) (state variable)

Reference:

<https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing>

INFO:Detectors:

OFTCore.setMsgInspector(address)._msgInspector (MaviaOFT.sol#1329) lacks a zero-check on :

- msgInspector = _msgInspector (MaviaOFT.sol#1330)

OAppPreCrimeSimulator.setPreCrime(address)._preCrime (MaviaOFT.sol#1244) lacks a zero-check on :

- preCrime = _preCrime (MaviaOFT.sol#1245)

Reference:

<https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation>

INFO:Detectors:

OAppPreCrimeSimulator.lzReceiveAndRevert(InboundPacket[]) (MaviaOFT.sol#1249-1265) has external calls inside a loop: this.lzReceiveSimulate{value:

packet.value}(packet.origin,packet.guid,packet.message,packet.executor,packet.extraData) (MaviaOFT.sol#1255-1261)

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop>

INFO:Detectors:

Reentrancy in OFTCore._lzReceive(Origin,bytes32,bytes,address,bytes)

(MaviaOFT.sol#1402-1424):

External calls:

- endpoint.sendCompose(toAddress,_guid,0,composeMsg) (MaviaOFT.sol#1420)

Event emitted after the call(s):

- OFTReceived(_guid,_origin.srcEid,toAddress,amountReceivedLD) (MaviaOFT.sol#1423)

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerability>

INFO:Detectors:

Pragma version^0.8.0 (MaviaOFT.sol#3) 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>

INFO:Detectors:

Parameter OFTCore.send(SendParam,MessagingFee,address)._refundAddress

(MaviaOFT.sol#1370) is not in mixedCase

Reference:

<https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions>

INFO:Detectors:

MaviaOFT (MaviaOFT.sol#1520-1531) does not implement functions:

- IOAppCore.peers(uint32) (MaviaOFT.sol#616)

Reference:

<https://github.com/crytic/slither/wiki/Detector-Documentation#unimplemented-functions>

INFO:Detectors:

OFTComposeMsgCodec.NONCE_OFFSET (MaviaOFT.sol#262) is never used in

OFTComposeMsgCodec (MaviaOFT.sol#261-303)

OFTComposeMsgCodec.SRC_EID_OFFSET (MaviaOFT.sol#263) is never used in

OFTComposeMsgCodec (MaviaOFT.sol#261-303)

OFTComposeMsgCodec.AMOUNT_LD_OFFSET (MaviaOFT.sol#264) is never used in

OFTComposeMsgCodec (MaviaOFT.sol#261-303)

OFTComposeMsgCodec.COMPOSE_FROM_OFFSET (MaviaOFT.sol#265) is never used in

OFTComposeMsgCodec (MaviaOFT.sol#261-303)

OFTMsgCodec.SEND_TO_OFFSET (MaviaOFT.sol#306) is never used in OFTMsgCodec

(MaviaOFT.sol#305-343)

OAppOptionsType3.OPTION_TYPE_3 (MaviaOFT.sol#1118) is never used in MaviaOFT

(MaviaOFT.sol#1520-1531)

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variable>

INFO:Slither:MaviaOFT.sol analyzed (37 contracts with 93 detectors), 132 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.

MaviaOFT.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.

Pos: 124:12:

Low level calls:

Use of "delegatecall": should be avoided whenever possible. External code, that is called can change the state of the calling contract and send ether from the caller's balance. If this is wanted behaviour, use the Solidity library feature if possible.

Pos: 85:50:

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.

Pos: 1255:12:

For loop over dynamic array:

Loops that do not have a fixed number of iterations, for example, loops that depend on storage values, have to be used carefully. Due to the block gas limit, transactions can only consume a certain amount of gas. The number of iterations in a loop can grow beyond the block gas limit which can cause the complete contract to be stalled at a certain point. Additionally, using unbounded loops incurs in a lot of avoidable gas costs. Carefully test how many items at maximum you can pass to such functions to make it successful.

Pos: 798:8:

For loop over dynamic array:

Loops that do not have a fixed number of iterations, for example, loops that depend on storage values, have to be used carefully. Due to the block gas limit, transactions can only consume a certain amount of gas. The number of iterations in a loop can grow beyond the block gas limit which can cause the complete contract to be stalled at a certain point. Additionally, using unbounded loops incurs in a lot of avoidable gas costs. Carefully test how many items at maximum you can pass to such functions to make it successful.

Pos: 1250:8:

Similar variable names:

OFTCore._toSD(uint256) : Variables have very similar names "_amountLD" and "amountSD".

Note: Modifiers are currently not considered by this static analysis.

Pos: 1449:22:

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.

Pos: 989:8:

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.

MaviaOFT.sol

```
Compiler version ^0.8.0 does not satisfy the ^0.5.8 semver requirement
Pos: 1:2
Avoid to use inline assembly. It is acceptable only in rare cases
Pos: 9:12
Error message for require is too long
Pos: 9:82
Avoid to use low level calls.
Pos: 51:84
Avoid to use inline assembly. It is acceptable only in rare cases
Pos: 17:98
Explicitly mark visibility in function (Set ignoreConstructors to true if using solidity >=0.7.0)
Pos: 5:1012
Variable "_refundAddress" is unused
Pos: 9:1096
Variable "messageValue" is unused
Pos: 9:1098
Variable "_eid" is unused
Pos: 9:1131
Variable "_msgType" is unused
Pos: 9:1132
Code contains empty blocks
Pos: 81:1149
Explicitly mark visibility in function (Set ignoreConstructors to true if using solidity >=0.7.0)
Pos: 5:1290
Code contains empty blocks
Pos: 86:1290
Explicitly mark visibility in function (Set ignoreConstructors to true if using solidity >=0.7.0)
Pos: 3:1520
```

Software analysis result:

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



This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.

Email: audit@EtherAuthority.io