

etherauthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project: Extra Finance (EXTRA)
Website: app.extrafi.io/lend
Platform: Base Chain Network
Language: Solidity
Date: June 24th, 2024

Table of contents

Introduction	4
Project Background	4
Audit Scope	6
Code Audit History	7
Severity Definitions	7
Claimed Smart Contract Features	8
Audit Summary	10
Technical Quick Stats	11
Business Risk Analysis	12
Code Quality	13
Documentation	13
Use of Dependencies	13
AS-IS overview	14
Audit Findings	16
Conclusion	19
Our Methodology	20
Disclaimers	22
Appendix	
• Code Flow Diagram	23
• Slither Results Log	24
• Solidity static analysis	26
• Solhint Linter	27

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 Extra Finance smart contract from app.extrafi.io/lend 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 June 24th, 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

The logo for Extrafi, featuring the word "Extrafi" in a blue, sans-serif font. The "E" is significantly larger and more stylized than the other letters. The logo is centered on a black rectangular background.

- Extra Finance is a leveraged yield strategy & lending protocol built on Optimism/Base.
- Extra Finance offers leveraged yield farming, allowing users to maximize earnings by leveraging stable pools like \$ETH/\$USDC for attractive yield rates and depositing assets in the Lending Pool for steady passive income. However, careful risk assessment and active management are crucial to mitigate potential drawbacks and ensure a successful farming experience.

Code Details

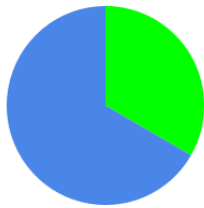
- The provided code implements a cross-chain token called `EXTRAoft` that leverages the LayerZero protocol to facilitate sending tokens between different blockchain networks. This functionality is achieved through several abstract and base contracts that define essential methods and events for handling cross-chain messaging and token operations. Here's a detailed breakdown of the key components and their roles:
- **Contract Components:**
 - EXTRAoft Contract:
 - Inherits `OFT`.
 - Initializes the token with the name "Extra Finance" and symbol "EXTRA".
 - Configures the contract to use a specified LayerZero endpoint for crosschain operations.
 - OFT Contract:
 - Inherits `OFTCore`, `ERC20`, and `IOFT`.
 - Implements the ERC20 token standard alongside the crosschain functionalities provided by `OFTCore`.
 - Defines `_debitFrom` and `_creditTo` methods to handle burning and minting tokens during crosschain transfers.
 - Provides additional methods to support ERC165 interface checks and tokenspecific details like total supply.
 - OFTCore Contract:
 - Inherits `NonblockingLzApp`, `ERC165`, and `IOFTCore`.
 - Provides core functionalities for an Omnichain Fungible Token (OFT), including support for estimating send fees and sending tokens across chains.
 - Implements the `_send` and `_sendAck` methods to handle token transfers and acknowledgments.
 - Includes abstract methods `_debitFrom` and `_creditTo` for debiting and crediting tokens, to be defined in derived contracts.
 - NonblockingLzApp Contract:
 - Inherits `LzApp`.

- Adds functionality for handling message failures and retries.
 - Stores failed messages and emits events when messages fail or succeed on retry.
 - Implements `_blockingLzReceive` to safely call the nonblocking receive method and store failed messages if necessary.
 - LzApp Contract:
 - Inherits `Ownable`, `ILayerZeroReceiver`, and `ILayerZeroUserApplicationConfig`.
 - Manages the configuration and communication with the LayerZero endpoint.
 - Contains mappings to store trusted remote addresses, minimum destination gas requirements, and payload size limits.
 - Defines methods to receive messages (`lzReceive`), send messages (`_lzSend`), and configure various settings such as trusted remotes and gas limits.
- The `EXTRAoft` contract extends a sophisticated framework for creating a cross-chain token that can be sent and received across different blockchain networks using the LayerZero protocol. The modular design allows for easy customization and extension while providing robust mechanisms for handling cross-chain messaging and token management.

Audit scope

Name	Code Review and Security Analysis Report for Extra Finance Smart Contract
Platform	Base Chain Network
Language	Solidity
File	EXTRAoft.sol
Smart Contract Code	0x2dad3a13ef0c6366220f989157009e501e7938f8
Audit Date	June 24th,2024
Audit Result	Passed

Code Audit History



3
Total Findings

0
Critical








0
High

0
Medium

1
Low

2
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
1		Low		Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets, that can't have a significant impact on execution
2		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: Extra Finance • Symbol: EXTRA • Decimals: 18 	<p>YES, This is valid.</p>
<p>Owner Specifications:</p> <ul style="list-style-type: none"> • Allows the current owner to transfer control of the contract to a new owner. • The current owner Leaves the contract without the owner. • Set configurations. • Set the send/receive version. • Set the trusted path for cross-chain communication. • Set a trusted remote address • Set precrime. • Set min dst gas. • Set the payload size limit. 	<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>
<p>Key Functionalities:</p> <ul style="list-style-type: none"> • CrossChain Messaging: The `LzApp` and `NonblockingLzApp` contracts manage crosschain message sending and receiving, ensuring messages are only accepted from trusted sources and handling potential message failures with retries. • Token Transfers: The `OFTCore` and `OFT` contracts implement methods to transfer tokens between chains by encoding the transfer details in the payload and ensuring the tokens are correctly debited from the sender and credited to 	<p>YES, This is valid.</p>

the receiver on the destination chain.

- Configuration and Security: The contracts include various configuration methods to set up trusted remotes, gas limits, payload size limits, and precrime addresses. These settings help ensure secure and efficient crosschain operations.

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.



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, 1 low, and 2 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	Moderated
	Human/contract checks bypass	Passed
	Random number generation/use vulnerability	N/A
	Fallback function misuse	Passed
	Race condition	Passed
	Logical vulnerability	Passed
	Features claimed	Passed
	Other programming issues	Moderated
Code Specification	Function visibility not explicitly declared	Passed
	Var. storage location not explicitly declared	Passed
	Use keywords/functions to be deprecated	Passed
	Unused code	Moderated
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 Extra Finance 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 Extra Finance.

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 Extra Finance 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

EXTRAoft.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Empty Blocks in Code	Refer Audit Findings
2	supportsInterface	read	Passed	No Issue
3	token	read	Passed	No Issue
4	circulatingSupply	read	Passed	No Issue
5	_debitFrom	internal	Passed	No Issue
6	_creditTo	internal	Passed	No Issue
7	supportsInterface	read	Passed	No Issue
8	estimateSendFee	read	Passed	No Issue
9	sendFrom	write	Passed	No Issue
10	setUseCustomAdapterParams	write	access only Owner	No Issue
11	_nonblockingLzReceive	internal	Passed	No Issue
12	send	internal	Passed	No Issue
13	_sendAck	internal	Passed	No Issue
14	checkAdapterParams	internal	Passed	No Issue
15	debitFrom	internal	Passed	No Issue
16	creditTo	internal	Passed	No Issue
17	_blockingLzReceive	internal	Passed	No Issue
18	_storeFailedMessage	internal	Passed	No Issue
19	nonblockingLzReceive	write	Passed	No Issue
20	_nonblockingLzReceive	internal	Passed	No Issue
21	retryMessage	write	Passed	No Issue
22	lzReceive	write	Passed	No Issue
23	_blockingLzReceive	internal	Unused function	Refer Audit Findings
24	_lzSend	internal	Passed	No Issue
25	checkGasLimit	internal	Passed	No Issue
26	_getGasLimit	internal	Passed	No Issue
27	checkPayloadSize	internal	Passed	No Issue
28	getConfig	external	Passed	No Issue
29	setConfig	external	Passed	No Issue
30	setSendVersion	external	access only Owner	No Issue
31	setReceiveVersion	external	access only Owner	No Issue
32	forceResumeReceive	external	access only Owner	No Issue
33	setTrustedRemote	external	access only Owner	No Issue

34	setTrustedRemoteAddress	external	access only Owner	No Issue
35	getTrustedRemoteAddress	external	Passed	No Issue
36	setPrecrime	external	access only Owner	No Issue
37	setMinDstGas	external	access only Owner	No Issue
38	setPayloadSizeLimit	external	Critical operation lacks event log	Refer Audit Findings
39	isTrustedRemote	external	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

[L-01] Critical operation lacks event log:

Description:

IMissing event log for setPayloadSizeLimit() in LzApp contract.

Recommendation: We suggest writing an event log for the listed event.

Very Low / Informational / Best practices:

[I-01] Unused function:

```
// overriding the virtual function in LzReceiver
function _blockingLzReceive(uint16 _srcChainId, bytes memory
_srcAddress, uint64 _nonce, bytes memory _payload) internal virtual
override {
    (bool success, bytes memory reason) =
address(this).excessivelySafeCall(gasleft(), 150,
abi.encodeWithSelector(this.nonblockingLzReceive.selector,
_srcChainId, _srcAddress, _nonce, _payload));
    // try-catch all errors/exceptions
```



```
    if (!success) {
        _storeFailedMessage(_srcChainId, _srcAddress, _nonce,
_payload, reason);
    }
}
```

Description:

The `_blockingLzReceive()` is an internal function defined in the `NonblockingLzApp` contract but never used in code.

Recommendation: We suggest removing unused functions.

[I-02] Empty Blocks in Code:

```
constructor(address _lzEndpoint) NonblockingLzApp(_lzEndpoint) {}
```

Description:

During the code review, it was identified that the constructor in the smart contract contains an empty block. Empty blocks serve no functional purpose and can lead to confusion for developers reviewing the code. Code readability and maintainability are essential for smart contracts, and empty blocks should be avoided.

Recommendation: We recommend removing the empty block to improve code clarity and maintainability.

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 Admin functions:

LzApp.sol

- setConfig: The owner can set configurations.
- setSendVersion: The owner can set the send version.
- setReceiveVersion: The owner can set the receive version.
- forceResumeReceive: The owner can resume receiving the address.
- setTrustedRemote: Set the trusted path for the cross-chain communication by the owner.
- setTrustedRemoteAddress: The owner can set a trusted remote address.
- setPrecrime: The owner can set precrime.
- setMinDstGas: The owner can set min dst gas.
- setPayloadSizeLimit: The owner can set the payload size limit.

Ownable.sol

- transferOwnership: Allows the current owner to transfer control of the contract to a newOwner.
- renounceOwnership: The current owner Leaves the contract without the 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. We have used all possible tests based on given objects as files. We observed 1 low and 2 Informational issues in the smart contracts. but those 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.

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 **"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 white box penetration testing. We look at the project's website 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, and 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

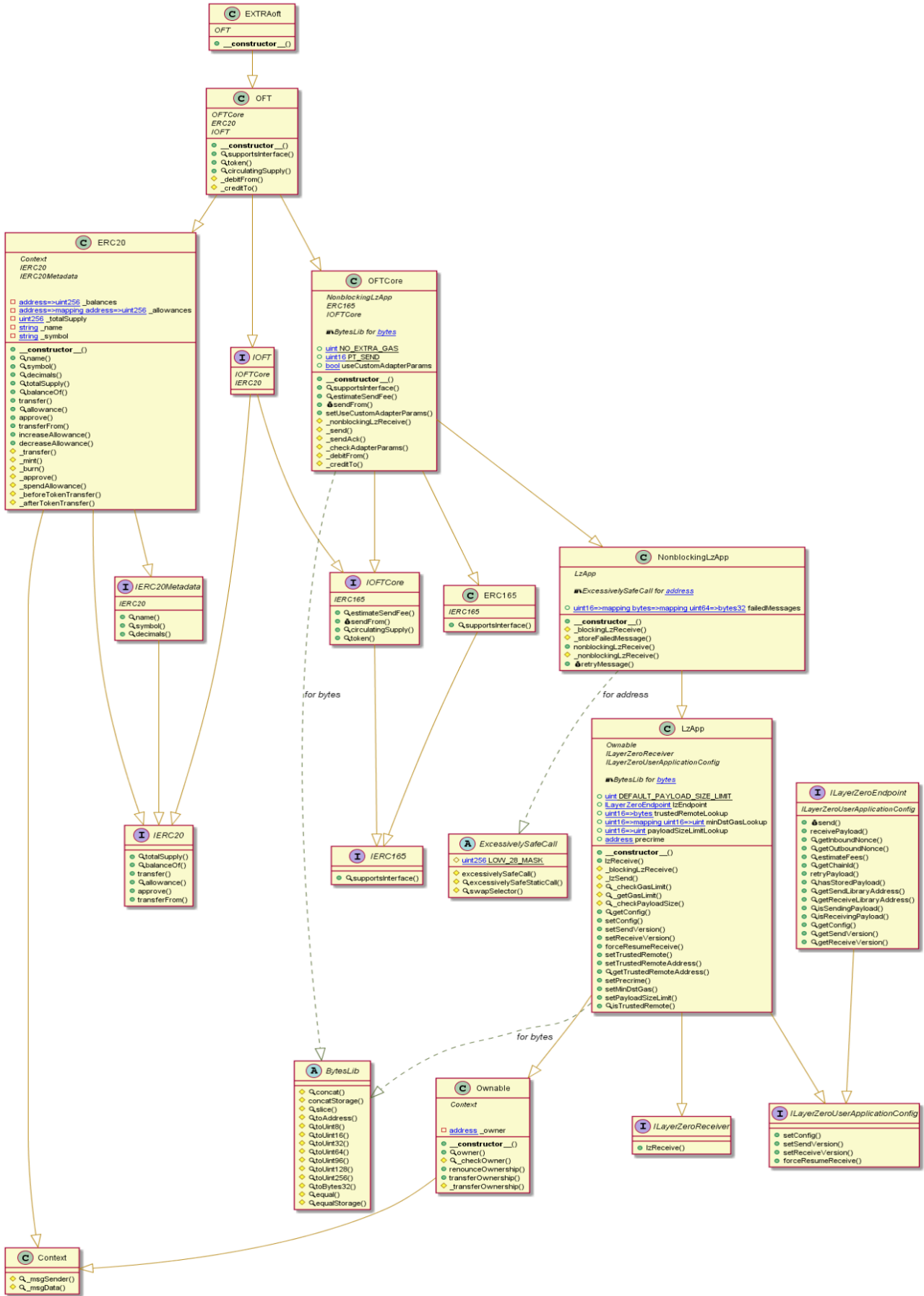
EtherAuthority team has analyzed this smart contract in accordance with the best industry practices at the date of this report, in relation to: cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation,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

Due to the fact that the total number of test cases is unlimited, the audit makes no statements or warranties on the security of the code. It also cannot be considered a sufficient assessment regarding the utility and safety of the code, bug-free 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 the explicit security of the audited smart contracts.

Code Flow Diagram - Extra Finance



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.

EXTRAoft.sol

```
INFO:Detectors:
OFT.constructor(string,string,address)._name (EXTRAoft.sol#1586) shadows:
  - ERC20._name (EXTRAoft.sol#837) (state variable)
OFT.constructor(string,string,address)._symbol (EXTRAoft.sol#1586) shadows:
  - ERC20._symbol (EXTRAoft.sol#838) (state variable)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
LzApp.setPrecrime(address)._precrime (EXTRAoft.sol#1430) lacks a zero-check on :
  - precrime = _precrime (EXTRAoft.sol#1431)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Reentrancy in OFTCore._send(address,uint16,bytes,uint256,address,address,bytes)
(EXTRAoft.sol#1550-1559):
  External calls:
  -
  _LzSend(_dstChainId,lzPayload,_refundAddress,_zroPaymentAddress,_adapterParams,msg.value)
  (EXTRAoft.sol#1556)
    - lzEndpoint.send{value:
    _nativeFee}(_dstChainId,trustedRemote,_payload,_refundAddress,_zroPaymentAddress,_adapter
    Params) (EXTRAoft.sol#1365)
    Event emitted after the call(s):
    - SendToChain(_dstChainId,_from,_toAddress,amount) (EXTRAoft.sol#1558)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
LzApp._getGasLimit(bytes) (EXTRAoft.sol#1375-1380) uses assembly
  - INLINE ASM (EXTRAoft.sol#1377-1379)
OFTCore._nonblockingLzReceive(uint16,bytes,uint64,bytes) (EXTRAoft.sol#1537-1548) uses
```


assembly

- INLINE ASM (EXTRAoft.sol#1539-1541)

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

INFO:Detectors:

ExcessivelySafeCall.excessivelySafeStaticCall(address,uint256,uint16,bytes)

(EXTRAoft.sol#669-703) is never used and should be removed

ExcessivelySafeCall.swapSelector(bytes4,bytes) (EXTRAoft.sol#714-729) is never used and should be removed

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

INFO:Detectors:

Pragma version^0.8.0 (EXTRAoft.sol#4) allows old versions

solc-0.8.25 is not recommended for deployment

Reference:

<https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity>

INFO:Detectors:

Parameter OFTCore.estimateSendFee(uint16,bytes,uint256,bool,bytes)._dstChainId (EXTRAoft.sol#1522) is not in mixedCase

Parameter OFTCore.estimateSendFee(uint16,bytes,uint256,bool,bytes)._toAddress (EXTRAoft.sol#1522) is not in mixedCase

Parameter OFTCore.estimateSendFee(uint16,bytes,uint256,bool,bytes)._amount (EXTRAoft.sol#1522) is not in mixedCase

Parameter OFTCore.estimateSendFee(uint16,bytes,uint256,bool,bytes)._useZro (EXTRAoft.sol#1522) is not in

mixedCaseOFTCore.sendFrom(address,uint16,bytes,uint256,address,address,bytes)._refundAddress (EXTRAoft.sol#1528) is not in mixedCase

Parameter

OFTCore.sendFrom(address,uint16,bytes,uint256,address,address,bytes)._zroPaymentAddress (EXTRAoft.sol#1528) is not in mixedCase

Parameter

OFTCore.sendFrom(address,uint16,bytes,uint256,address,address,bytes)._adapterParams (EXTRAoft.sol#1528) is not in mixedCase

Parameter OFTCore.setUseCustomAdapterParams(bool)._useCustomAdapterParams (EXTRAoft.sol#1532) is not in mixedCase

Reference:

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

INFO:Detectors:

ExcessivelySafeCall.slitherConstructorConstantVariables() (EXTRAoft.sol#598-730) uses literals with too many digits:

- LOW_28_MASK = 0x00000000ffffffffffffffffffffffffffffffffffffffff

(EXTRAoft.sol#599-600)

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits>

INFO:Slither:EXTRAoft.sol analyzed (19 contracts with 93 detectors), 135 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.

EXTRAoft.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: 1539:8:

Gas costs:

Gas requirement of function EXTRAoft.setPrecrime 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: 1430:4:

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.

Pos: 1471:119:

Constant/View/Pure functions:

ExcessivelySafeCall.swapSelector(bytes4,bytes) : Is constant but potentially should not be. Note: Modifiers are currently not considered by this static analysis.

Pos: 714:4:

Constant/View/Pure functions:

LzApp._getGasLimit(bytes) : Is constant but potentially should not be. Note: Modifiers are currently not considered by this static analysis.

Pos: 1375:4:

No return:

OFTCore._debitFrom(address,uint16,bytes,uint256): Defines a return type but never explicitly returns a value.

Pos: 1579:4:

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.

Pos: 1575:12:

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.

EXTRAoft.sol

```
Compiler version ^0.8.0 does not satisfy the ^0.5.8 semver requirement
Pos: 1:3
Check result of "send" call
Pos: 9:1364
Avoid to use inline assembly. It is acceptable only in rare cases
Pos: 9:1376
Explicitly mark visibility in function (Set ignoreConstructors to true if using solidity >=0.7.0)
Pos: 5:1515
Code contains empty blocks
Pos: 68:1515
Avoid to use inline assembly. It is acceptable only in rare cases
Pos: 9:1538
Error message for require is too long
Pos: 13:1574
Explicitly mark visibility in function (Set ignoreConstructors to true if using solidity >=0.7.0)
Pos: 5:1585
Code contains empty blocks
Pos: 125:1585
Explicitly mark visibility in function (Set ignoreConstructors to true if using solidity >=0.7.0)
Pos: 5:1614
Code contains empty blocks
Pos: 57:1616
```

Software analysis result:

This software reported many false positive results and some were 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