



Building the Futuristic **Blockchain Ecosystem**

SECURITY AUDIT REPORT

Bridge Contract Audit

TABLE OF CONTENTS

02	Table of Contents	_____
03	Overview	_____
04	Audit Summary	_____
06	Contract Details	_____
07	Audit Methodology	_____
08	Vulnerabilities Checklist	_____
09	Risk Classification	_____
10	Ownership Privileges	_____
14	Functions	_____
16	Inheritance Graph	_____
17	Centralization Privileges	_____
18	Manual Review	_____
19	Findings	_____
24	About Expelee	_____
25	Disclaimer	_____

OVERVIEW

The Expelee team has performed a line-by-line manual analysis and automated review of the smart contract. The smart contract was analysed mainly for common smart contract vulnerabilities, exploits, and manipulation hacks. According to the smart contract audit:

Overall Security	Contract is Safe to Deploy
Audit Date	16 March 2024

AUDIT SUMMARY

Version : v1.0

- Layout Project
- Automated- /Manual-Security Testing
- Summary

Note - The following audit report presents a comprehensive security analysis of the smart contract utilized in the project that includes outside manipulation of the contract's functions in a malicious way. This analysis did not include functional testing (or unit testing) of the contract/s logic. We cannot guarantee 100% logical correctness of the contract as we did not functionally test it. This includes internal calculations in the formulae used in the contract

File Overview

The Team provided us with the files that should be tested in the security assessment. This audit covered the following files listed below with an SHA-1 Hash.

File Name	SHA-1 Hash
src/Bridge/Bridge.sol	1716a1aaf0dd61d37307824c2f848d4e2ac085e1

Note - Files with a different hash value than in this table have been modified after the security check, either intentionally or unintentionally. A different hash value may (but need not) be an indication of a changed state or potential vulnerability that was not the subject of this scan.

AUDIT SUMMARY

Imported packages

Used code from other Frameworks/Smart Contracts (direct imports).

Dependency / Import Path	Count
@layerzero/contracts/interfaces/ILayerZeroEndpoint.sol	1
@layerzero/contracts/interfaces/ILayerZeroReceiver.sol	1
@openzeppelin/contracts/access/AccessControl.sol	1
@openzeppelin/contracts/proxy/utils/Initializable.sol	1
@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol	1

Note for Investors: We only audited contracts mentioned in the scope above. All contracts related to the project apart from that are not a part of the audit, and we cannot comment on its security and are not responsible for it in any way

CONTRACT DETAILS

Token Name: --

Symbol: --

Network: --

Language: Solidity

Contract Address: --

Owner's Wallet: --

Deployer's Wallet: --

AUDIT METHODOLOGY

Auditing Strategy and Techniques Applied

Throughout the review process, care was taken to check the repository for security related issues, code quality, and compliance with specifications and best practices. To this end, our team of experienced pen-testers and smart contract developers reviewed the code line by line and documented any issues discovered.

We check every file manually. We use automated tools only so that they help us achieve faster and better results.

Methodology

The auditing process follows a routine series of steps:

1. Code review that includes the following:

- Reviewing the specifications, sources, and instructions provided to Expelee to ensure we understand the size, scope, and functionality of the smart contract.
- Manual review of the code, i.e. reading the source code line by line to identify potential vulnerabilities.
- Comparison to the specification, i.e. verifying that the code does what is described in the specifications, sources, and instructions provided to Expelee

2. Testing and automated analysis that includes the following:

- Test coverage analysis, which determines whether test cases actually cover code and how much code is executed when those test cases are executed.
- Symbolic execution, which is analyzing a program to determine what inputs causes each part of a program to execute.

3. Review best practices, i.e., review smart contracts to improve efficiency, effectiveness, clarity, maintainability, security, and control based on best practices, recommendations, and research from industry and academia.

4. Concrete, itemized and actionable recommendations to help you secure your smart contracts.

VULNERABILITY CHECKS

Design Logic	Passed
Compiler warnings	Passed
Private user data leaks	Passed
Timestamps dependence	Passed
Integer overflow and underflow	Passed
Race conditions & reentrancy. Cross-function race conditions	Passed
Possible delays in data delivery	Passed
Oracle calls	Passed
Front Running	Passed
DoS with Revert	Passed
DoS with block gas limit	Passed
Methods execution permissions	Passed
Economy model	Passed
Impact of the exchange rate on the logic	Passed
Malicious event log	Passed
Scoping and declarations	Passed
Uninitialized storage pointers	Passed
Arithmetic accuracy	Passed
Cross-function race conditions	Passed
Safe Zepplin module	Passed

RISK CLASSIFICATION

When performing smart contract audits, our specialists look for known vulnerabilities as well as logical and access control issues within the code. The exploitation of these issues by malicious actors may cause serious financial damage to projects that failed to get an audit in time. We categorize these vulnerabilities by the following levels:

High Risk

Issues on this level are critical to the smart contract's performance/functionality and should be fixed before moving to a live environment.

Medium Risk

Issues on this level are critical to the smart contract's performance/functionality and should be fixed before moving to a live environment.

Low Risk

Issues on this level are minor details and warnings that can remain unfixed.

Informational

Issues on this level are minor details and warnings that can remain unfixed.

OWNERSHIP PRIVILEGES

These functions can be dangerous. Please note that abuse can lead to financial loss. We have a guide where you can learn more about these Functions.

Minting tokens

Minting tokens refers to the process of creating new tokens in a cryptocurrency or blockchain network. This process is typically performed by the project's owner or a designated authority, who has the ability to add new tokens to the network's total supply.

Contract owner cannot mint new tokens

 **The owner cannot mint new tokens**

Description :

The owner is not able to mint new tokens once the contract is deployed

Burning tokens

Burning tokens is the process of permanently destroying a certain number of tokens, reducing the total supply of a cryptocurrency or token. This is usually done to increase the value of the remaining tokens, as the reduced supply can create scarcity and potentially drive up demand.

Contract owner cannot burn tokens

 **The owner cannot burn tokens**

Description

The owner is not able burn tokens without any allowances.

OWNERSHIP PRIVILEGES

Blacklist addresses

Blacklisting addresses in smart contracts is the process of adding a certain address to a blacklist, effectively preventing them from accessing or participating in certain functionalities or transactions within the contract. This can be useful in preventing fraudulent or malicious activities, such as hacking attempts or money laundering.

Contract owner cannot blacklist addresses

 **The owner cannot blacklist addresses**

Description :

The owner is not able blacklist addresses to lock funds.

Upgradeability

Contract is not an upgradeable

 **Deployer cannot update the contract with new functionalities**

Description :

The contract is not an upgradeable contract. The deployer is not able to change or add any functionalities to the contract after deploying

OWNERSHIP PRIVILEGES

Fees and Tax

In some smart contracts, the owner or creator of the contract can set fees for certain actions or operations within the contract. These fees can be used to cover the cost of running the contract, such as paying for gas fees or compensating the contract's owner for their time and effort in developing and maintaining the contract.

Contract owner cannot set fees more than 25%



The owner cannot levy unfair taxes

Description :

The owner is not able to set the fees above 25%

Lock User Funds

In a smart contract, locking refers to the process of restricting access to certain tokens or assets for a specified period of time. When tokens or assets are locked in a smart contract, they cannot be transferred or used until the lock-up period has expired or certain conditions have been met.

Owner cannot lock the contract



The owner cannot lock the contract

Description

The owner is not able to lock the contract by any functions or updating any variables.

OWNERSHIP PRIVILEGES

Ownership

The ownership is not renounced

X The owner is not renounce

Description :

The owner has not renounced the ownership that means that the owner retains control over the contract's operations, including the ability to execute functions that may impact the contract's users or stakeholders. This can lead to several potential issues, including:

- Centralizations
- The owner has significant control over contract's operations

Note - If the contract is not deployed then we would consider the ownership to be not renounced. Moreover, if there are no ownership functionalities then the ownership is automatically considered renounced

FUNCTIONS

External/Public functions

External/public functions are functions that can be called from outside of a contract, i.e., they can be accessed by other contracts or external accounts on the blockchain. These functions are specified using the function declaration's external or public visibility modifier.

State variables



State variables are variables that are stored on the blockchain as part of the contract's state. They are declared at the contract level and can be accessed and modified by any function within the contract. State variables can be defined with a visibility modifier, such as public, private, or internal, which determines the access level of the variable

Components

 Contracts	 Libraries	 Interfaces	 Abstract
1	0	1	0


Exposed Functions

This section lists functions that are explicitly declared public or payable. Please note that getter methods for public stateVars are not included.

 Public	 Payable
15	1




External	Internal	Private	Pure	View
9	16	0	4	3

State Variable

Total	 Public
5	5

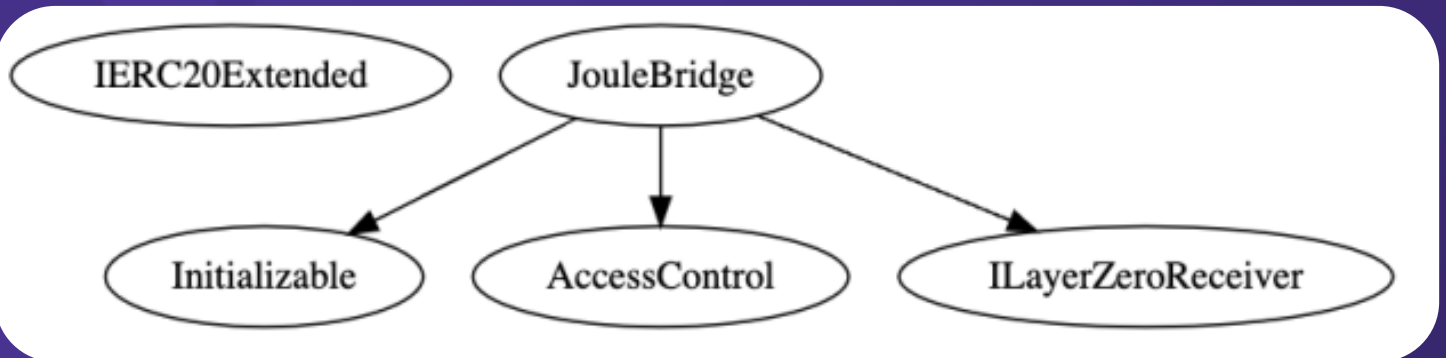
FUNCTIONS

Capabilities

Solidity Versions observed	Transfers ETH	 Can Receive Funds	 Uses Assembl y	 Has Destroyable Contracts
<code>^0.8.13</code>	Yes	Yes		

INHERITANCE GRAPH

An inheritance graph is a graphical representation of the inheritance hierarchy among contracts. In object-oriented programming, inheritance is a mechanism that allows one class (or contract, in the case of Solidity) to inherit properties and methods from another class. It shows the relationships between different contracts and how they are related to each other through inheritance.



CENTRALIZATION PRIVILEGES

Centralization can arise when one or more parties have privileged access or control over the contract's functionality, data, or decision-making. This can occur, for example, if the contract is controlled by a single entity or if certain participants have special permissions or abilities that others do not.

In the project there are authorities that has the authority over the following functions:

File	Privileges
Bridge.sol	<ul style="list-style-type: none">• List and Delist tokens from the bridge• Set the source LayerZero Implementation address• Set Send and Receive message library versions

Recommendations

To avoid potential hacking risks, it is advisable for the client to manage the private key of the privileged account with care. Additionally, we recommend enhancing the security practices of centralized privileges or roles in the protocol through a decentralized mechanism or smart contract-based accounts, such as multi-signature wallets.

Here are some suggestions what the client can do.

- Consider using multi-signature wallets: Multi-signature wallets require multiple parties to sign off on a transaction before it can be executed, providing an extra layer of security e.g. Gnosis Safe
- Use of a timelock at least with a latency of e.g. 48-72 hours for awareness on privileged operations
- Introduce a DAO/Governance/Voting module to increase transparency and user involvement
- Consider Renouncing the ownership so that the owner cannot modify any state variables of the contract anymore. Make sure to set up everything before renouncing.

MANUAL REVIEW

Severity Criteria

Expelee assesses the severity of disclosed vulnerabilities according to methodology based on OWASP standarts.

Vulnerabilities are dividend into three primary risk categorieis:

High

Medium

Low

High-level considerations for vulnerabilities span the following key areas when conducting assessments:

- Malicious input handling
- Escalation of privileges
- Arithmetic
- Gas use

Overall Risk Severity				
Impact	HIGH	Medium	High	Critical
	MEDIUM	Low	Medium	High
	LOW	Note	Low	Medium
		LOW	MEDIUM	HIGH
	Likelihood			

RISK FINDINGS

Findings	Severity	Found
High Risk	● High	0
Medium Risk	● Medium	1
Low Risk	● Low	1
Suggestion & discussion	● Informational	2

MEDIUM RISK FINDING

Owner can drain tokens

File: Main

Severity: Medium

Location: L220

Status: Open

Description

The contract owner can drain the contract's complete balance by calling this function.

Remediation

We recommend to make it impossible to withdraw the tokens that are sent for cross-chain transfers and only accidentally sent tokens should be withdrawable by the owners

LOW RISK FINDING

Owner can drain tokens

File: Main

Severity: Low

Location: L74, 78, 88

Status: Open

Description

Make sure to emit events for all the critical parameter changes in the contract to ensure the transparency and trackability of all the state variable changes.

INFORMATIONAL RISK FINDING

NatSpec documentation missing

File: Main

Severity: Informational

Status: Open

Description

If you started to comment on your code, comment on all other functions, variables etc.

INFORMATIONAL RISK FINDING

Floating Pragma

File: Main

Severity: Informational

Status: Open

Description

The contracts should be deployed with the same compiler version and flag that they have been tested thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using other versions.

ABOUT EXPELEE

Expelee is a product-based aspirational Web3 start-up. Coping up with numerous solutions for blockchain security and constructing a Web3 ecosystem from deal making platform to developer hosting open platform, while also developing our own commercial and sustainable blockchain.

 www.expelee.com

 [expeleeofficial](https://twitter.com/expeleeofficial)

 [expelee](https://medium.com/expelee)

 [Expelee](https://t.me/Expelee)

 [expelee](https://in.linkedin.com/company/expelee)

 [expelee_official](https://www.instagram.com/expelee_official)

 [expelee-co](https://github.com/expelee-co)

expelee

Building the Futuristic **Blockchain Ecosystem**

DISCLAIMER

All the content provided in this document is for general information only and should not be used as financial advice or a reason to buy any investment. Team provides no guarantess against the sale of team tokens or the removal of liquidity by the project audited in this document.

Always do your own research and project yourselves from being scammed. The Expelee team has audited this project for general information and only expresses their opinion based on similar projects and checks from popular diagnostic tools.

Under no circumstances did Expelee receive a payment to manipulate those results or change the awarding badge that we will be adding in our website. Alway do your own research and protect yourselves from scams.

This document should not be presented as a reason to buy or not buy any particular token. The Expelee team disclaims any liability for the resulting losses.

The logo for Expelee, featuring the word "expelee" in a stylized font. The "ex" is in white, and "pelee" is in orange. The letters are bold and modern.

Building the Futuristic **Blockchain Ecosystem**