HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: LaChain
Date: 16 Oct, 2023

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Document

Name	Smart Contract Code Review and Security Analysis Report for LaChain
Approved By	Niccolò Pozzolini | Solidity SC Auditor at Hacken OÜ Carlo Parisi | Solidity SC Auditor at Hacken OÜ
Tags	Bridge;
Platform	EVM
Language	Solidity
Methodology	Link
Changelog	18.09.2023 - Initial Review 16.10.2023 - Second Review

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Table of contents

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Introduction

Hacken OÜ (Consultant) was contracted by LaChain (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contracts.

System Overview

LaChain is a bridge protocol that provides a modular multi-directional blockchain bridge to allow data and value transfer between any number of blockchains.

For this purpose, the ChainBridge is used. ChainBridge is an extensible cross-chain communication protocol. It currently supports bridging between EVM based chains.

A bridge contract on each chain forms either side of a bridge. Handler contracts allow for customizable behavior upon receiving transactions to and from the bridge. For example, locking up an asset on one side and minting a new one on the other.

In its current state, ChainBridge operates under a trusted federation model. Deposit events on one chain are detected by a trusted set of off-chain relayers who await finality, submit events to the other chain and vote on submissions to reach acceptance triggering the appropriate handler.

Chainbridge is a relayer type bridge. The role of a relayer is to vote for the execution of a request (how many tokens to burn/release, for example). It monitors events from every chain, and votes for a proposal in the Bridge contract of the destination chain when it receives a Deposit event from a chain. A relayer calls a method in the Bridge contract to execute the proposal after the required number of votes are submitted. The bridge delegates execution to the Handler contract.

Privileged roles

- The owner can withdraw funds from the handler contract with the emergencyWithdraw(), adminWithdraw() and transferFunds() functions, it can also set variables in the Bridge.sol contract such as: fee, relayerThreshold, add and remove relayers, forwarder, deposit nonce, burnable, generic resource, resource, minimum amount to bridge, resource IDs.
- The relayers can vote, execute and reject a proposal in the Bridge.sol contract.
- The bridge can withdraw, deposit and execute a proposal in the ERC20Handler.sol contract.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Executive Summary

The score measurement details can be found in the corresponding section of the scoring methodology.

Documentation quality

The total Documentation Quality score is **10** out of **10**.

- Functional requirements are provided.
- Technical description is provided.
- Natspec is provided.
- The development environment is described.

Code quality

The total Code Quality score is **10** out of **10**.

- The development environment is configured.

Test coverage

Code coverage of the project is **94.25%** (branch coverage).

- Deployment and basic user interactions are covered with tests.
- Negative cases coverage is missed.
- Interactions by several users are not tested thoroughly.

Security score

As a result of the audit, the code contains no issues. The security score is **10** out of **10**.

All found issues are displayed in the “Findings” section.

Summary

According to the assessment, the Customer's smart contract has the following score: **9.8**. The system users should acknowledge all the risks summed up in the risks section of the report.

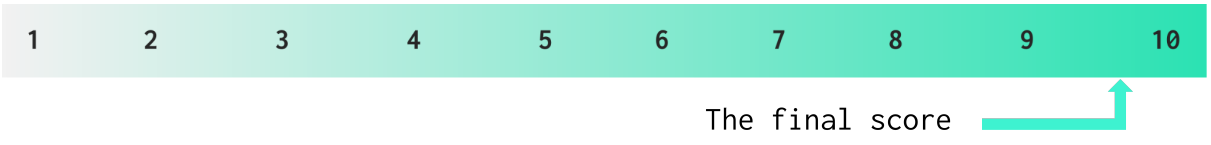

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
18 September 2023	4	2	1	4
16 October 2023	0	0	0	0

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Risks

- If the deposit is performed through WrapperLAC.wrapAndDeposit(), the sender parameter in the Deposit() event will be the WrapperLAC instead of the user.
- Users bridge funds from chain A to chain B, they transfer the funds to a contract in chain A expecting to receive them on chain B, but there is no way of knowing if chain B has enough funds, in this case the user would pay on chain A and not receive anything on chain B, relayers or the admin would need to manually unlock the funds.
- The protocol relays on relayer to work properly, if one gets compromised, the bridge could get Denial of Service attacked, if enough relayers get compromised the bridge could be drained.
- The contracts are upgradeable, the audit remains valid only if the code being audited is the one getting deployed on chain and does not get upgraded.
- There are external calls to contracts that are not in the scope, this external interactions could create problems that are not verifiable in the scope of the audit.
- The fee can be changed by the owner at any time and has no limit on the amount that can be extracted from the transaction.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Checked Items

We have audited the Customers' smart contracts for commonly known and specific vulnerabilities. Here are some items considered:

Item	Description	Status	Related Issues
Default Visibility	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	Passed	
Integer Overflow and Underflow	If unchecked math is used, all math operations should be safe from overflows and underflows.	Passed	
Outdated Compiler Version	It is recommended to use a recent version of the Solidity compiler.	Passed	
Floating Pragma	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Passed	
Unchecked Call Return Value	The return value of a message call should be checked.	Passed	
Access Control & Authorization	Ownership takeover should not be possible. All crucial functions should be protected. Users could not affect data that belongs to other users.	Passed	
SELFDESTRUCT Instruction	The contract should not be self-destructible while it has funds belonging to users.	Not Relevant	
Check-Effect-Interaction	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed	
Assert Violation	Properly functioning code should never reach a failing assert statement.	Passed	
Deprecated Solidity Functions	Deprecated built-in functions should never be used.	Passed	
Delegatecall to Untrusted Callee	Delegatecalls should only be allowed to trusted addresses.	Passed	
DoS (Denial of Service)	Execution of the code should never be blocked by a specific contract state unless required.	Passed	

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Race Conditions	Race Conditions and Transactions Order Dependency should not be possible.	Passed	
Authorization through tx.origin	tx.origin should not be used for authorization.	Not Relevant	
Block values as a proxy for time	Block numbers should not be used for time calculations.	Not Relevant	
Signature Unique Id	Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifiers should always be used. All parameters from the signature should be used in signer recovery. EIP-712 should be followed during a signer verification.	Not Relevant	
Shadowing State Variable	State variables should not be shadowed.	Passed	
Weak Sources of Randomness	Random values should never be generated from Chain Attributes or be predictable.	Passed	
Incorrect Inheritance Order	When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order.	Passed	
Calls Only to Trusted Addresses	All external calls should be performed only to trusted addresses.	Passed	
Presence of Unused Variables	The code should not contain unused variables if this is not justified by design.	Passed	
EIP Standards Violation	EIP standards should not be violated.	Passed	
Assets Integrity	Funds are protected and cannot be withdrawn without proper permissions or be locked on the contract.	Passed	
User Balances Manipulation	Contract owners or any other third party should not be able to access funds belonging to users.	Passed	
Data Consistency	Smart contract data should be consistent all over the data flow.	Passed	

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Flashloan Attack	When working with exchange rates, they should be received from a trusted source and not be vulnerable to short-term rate changes that can be achieved by using flash loans. Oracles should be used. Contracts shouldn't rely on values that can be changed in the same transaction.	Not Relevant	
Token Supply Manipulation	Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the Customer.	Passed	
Gas Limit and Loops	Transaction execution costs should not depend dramatically on the amount of data stored on the contract. There should not be any cases when execution fails due to the block Gas limit.	Not Relevant	
Style Guide Violation	Style guides and best practices should be followed.	Passed	
Requirements Compliance	The code should be compliant with the requirements provided by the Customer.	Passed	
Environment Consistency	The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Passed	
Secure Oracles Usage	The code should have the ability to pause specific data feeds that it relies on. This should be done to protect a contract from compromised oracles.	Not Relevant	
Tests Coverage	The code should be covered with unit tests. Test coverage should be sufficient, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Passed	
Stable Imports	The code should not reference draft contracts, which may be changed in the future.	Passed	

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Findings

■■■■ Critical

C01. Highly Permissive Role Access in adminWithdraw and emergencyWithdraw

Impact	High
Likelihood	High

Description: There is an highly permissive role access for the admin inside the functions adminWithdraw() and emergencyWithdraw() in the contracts Bridge.sol and ERC20HandlerHelpers.sol, the admin is able to withdraw assets that could and probably will belong to the user.

Intro: Those functions are not documented and are dangerous for the user in case of a private key leak from the admin.

Details: Both functions allow the admin to withdraw the funds that will be in the ERC20Handler.sol contract, this without having passed the normal flow of the bridge.

Impact: In case of private key leak or malicious activity from the admin the bridge could be drained without the approval of the relayers.

Risk Breakdown: The likelihood of vulnerability exploit depends on how well the private key of the admin is protected, a professional social engineering attack could pose a serious risk for the protocol.

Path: ./contracts/Bridge.sol : adminWithdraw(), transferFunds();

./contracts/handlers/ERC20HandlerHelpers.sol : emergencyWithdraw();

Recommendation: Specify this risk in the documentation very clearly and explain why the functions would be needed or remove the functions altogether.

Status: Mitigated (The client added the following to the project's documentation: If there's a misappropriation by the admin or if the admin keys are compromised, user funds could be at risk. To mitigate this, we will implement multi-signature wallet for the admin to enhance security)

C02. Proposals Mapping Key Collision

Impact	High
Likelihood	High

Description: The key used in the mapping proposal is badly crafted and can collide, resulting in proposals getting overwritten.

Intro: The proposals mapping contains the operations to be bridged. The proposals mapping uses as keys: depositNonce, originDomainID, dataHash. depositNonce and originDomainID are bitwise combined to create the key nonceAndID. This combination does not take correctly into account the variables' size, resulting in a non-unique key.

Details: The bitwise logic assumes that originDomainID is 8 bit long, while it's declared as uint64. nonceAndID is created by bitwise OR between originDomainID casted to uint72, and depositNonce first casted to uint72 and then shifted 8 bits. Everything would work if originDomainID was uint8 type: uint8 and uint64 could be joined in a uint72. Given the uint64 size, 64-8=65 bits will be ambiguous.

Impact: During a collision a proposal could "overwrite" an active proposal containing the same operation, resulting in lock of funds.

Risk Breakdown: An intentional exploit for profit could not be identified. The likelihood of the issue would depend on how many of the ambiguous bits are actively used.

Path: ./contracts/Bridge.sol : getProposal(), voteProposal(), cancelProposal(), executeProposal()

Recommendation: Two solutions are possible:

- If the domains are and will be less than 256, it suffices to make domainID as uint8 type.
- If a 64 bits size is desired, nonceAndID must be uint128 type, and in the bitwise operation depositNonce must be shifted by 64 bits instead of 8.

Status: Fixed

C03. Unverifiable Logic in ERC20HandlerHelpers and WrapperLAC

Impact	**High**
Likelihood	**High**

Description: There is a call to a contract called through the interface IWLAC inside the contracts ERC20HandlerHelpers.sol and WrapperLAC.sol, while the interface IWLAC is inside the scope, the implementation is missing. The logic of the contract cannot be checked completely without the implementation of IWLAC.sol.

Intro: The calls are performed on a contract that is not presented in the scope of the audit thus it cannot be checked.

Details: An external call can alter the functionality of the contract that performs that external call, every contract that is used in the chain of calls should be in the scope.

Impact: If the called contract present any vulnerability it could be exploited by malicious actors, it is impossible to precisely

calculate the impact of the vulnerabilities without having the code in scope.

Risk Breakdown: The likelihood of a vulnerability being present in the end contract it is impossible to calculate, for this reason it is assumed that every contract not in the scope is potentially vulnerable and should be added to the scope.

Path: ./contracts/WrapperLAC.sol : wrapAndDeposit();

./contracts/handlers/ERC20HandlerHelpers.sol : unWrapAndSendNative();

Recommendation: Add the implementation to the scope of the audit or remove the external calls to unsafe contracts.

Status: Fixed

C04. Unvalidated Deposit Amount

Impact	**High**
Likelihood	**High**

Description: When wrapping and bridging LAC, the amount to be bridged is not validated, making it possible to deposit 1 wei and withdraw the entire bridge's wrapped LAC balance on any chain connected to the bridge.

Intro: One of the entry point for bridging funds is the function wrapAndDeposit of the contract WrapperLAC. This function takes care of wrapping the native LAC and bridging it.

Details: The parameter data contains the encoded amount to be released on the other side of the bridge, it is the value that will end up in the event parameter. The issue is that this value is not validated against msg.value.

Impact: The entire bridge's wrapped LAC balance can be drained on any chain where the bridge is active.

Risk Breakdown: The vulnerability is extremely easy to exploit, it is merely a transaction call.

Path: ./contracts/WrapperLAC.sol : wrapAndDeposit()

Recommendation: Do not provide the amount to be bridged as input, derive it from msg.value instead.

Status: Fixed

■ ■ ■ High

H01. Any Realyer Could Cancel a Proposal

Impact	**High**

Likelihood	Medium

Description: When using the bridge proposals are created to transfer funds between different blockchains, those proposals need to be approved by a certain amount of relayers to be valid, but they could be rejected by a single one to be rendered invalid, there should be a certain threshold under which the proposal is not rejected.

Intro: The proposals to bridge assets between blockchains could be rejected by a single relayer.

Details: The proposals being rejected by a single relayer could create a situation where certain transactions are censored by a single relayer or the bridge encounters problem if a malicious actor gains access to one of the relayers.

Impact: The impact could be very similar to a DoS attack for a single user or for the entire protocol.

Risk Breakdown: The likelihood is not as severe as the impact, gaining access to a relayer might be complicated.

Path: ./contracts/Bridge.sol : cancelProposal();

Recommendation: Implement a threshold under which a proposal is still deemed as valid and cannot be rejected by a single relayer.

Status: Fixed

Medium

M01. Missing Validations in the Bridge Initializer

Impact	High
Likelihood	Low

Description: There are multiple fields in the bridge initializer that are not validated, the initialRelayerThreshold should not be 0, the fee should be 100% or preferably less and the initialRelayers.lenght should be maximum 200.

Intro: Human error in the initialization could create problems for the bridge.

Details: There are multiple fields with clear functional requirements that are not enforced in the code.

Impact: The consequences of human error in the initialization could create various terrible scenarios, such as:

- The fee higher than 100% would create a Denial of Service.
- The initialRelayerThreshold equal to 0 or 1 would mean that any relayer could pass any proposal.

- The initialRelayer.lenght higher than 200 would violate a functional requirement.

Risk Breakdown: The likelihood of human error is low, but not equal to 0, it is always better to allow for the code to render human error impossible rather than improbable.

Path: ./contracts/Bridge.sol : initialization(), adminChangeRelayerThreshold(), adminChangeFee(), adminAddRelayer();

Recommendation: Implement validations for the input parameters of the initializer for the Bridge.sol contract.

Status: Fixed (The fee value has not been bounded to a reasonable value, but has been described in the documentation)

M02. Missing Validations in the Fee Variable

Impact	Medium
Likelihood	Medium

Description: The fee can be 100% or higher, this should not be possible, the fee value should be a reasonable value that is preferably under 100%, but it should never be higher than 100%.

Intro: The fee value is unbounded, but in the initialization and later on when is set again by the admin.

Details: The fee should be bounded to a reasonable value and there should be validations in the code to prevent the value of the fee variable to exceed a certain threshold.

Impact: A fee over 100% would create a Denial of Service that is temporary, a fee that is 100% or close to this value would create monetary loss for the users.

Risk Breakdown: It is always better to allow for the code to render human error impossible rather than improbable.

Path: ./contracts/Bridge.sol : initialization(), adminChangeFee();

Recommendation: Implement validations for the input parameters of the initializer for the Bridge.sol contract.

Status: Fixed (The fee value has not been bounded to a reasonable value, but has been described in the documentation)

Low

L01. Redundant Storage Read

Impact	Low
Likelihood	Medium

Inside the function voteProposal(), the first if statement checks _resourceIDToHandlerAddress[resourceID] against address(0).

That storage read could be avoided since that value had already been loaded in the memory variable handler.

Path: ./contracts/Bridge.sol : voteProposal()

Recommendation: Avoid redundant storage reads, check handler against 0.

Status: Fixed

L02. Missing Error Parameter

Impact	Low
Likelihood	Medium

The error transferFailed() lacks the index of the failed transfer. If one of a huge list of transfer fails, the administrator will be required to manually check them all.

Path: ./contracts/Bridge.sol : transferFunds()

Recommendation: Add the address (or its array index) of the failed transfer to transferFailed() error.

Status: Fixed

L04. Initializers Not Disabled on the Implementation Contract

Impact	Low
Likelihood	Medium

The contracts Bridge, ERC20Handler do not follow the best practice of disabling the initializer function on the implementation contract, meant to avoid possible edge case unwanted situations.

Path: ./contracts/Bridge.sol; ./contracts/handlers/ERC20Handler.sol

Recommendation: Add a constructor and call _disableInitializers() inside of it.

Status: Fixed

Informational

I01. Style Guide Violation

The provided projects should follow the official guidelines.

Path: ./contracts/*

Recommendation: Follow the official Solidity guidelines.

Found in: 128u923

Status: Fixed

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing of this report, with 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).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

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 recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the Consultant cannot guarantee the explicit security of the audited smart contracts.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Appendix 1. Severity Definitions

When auditing smart contracts Hacken is using a risk-based approach that considers the potential impact of any vulnerabilities and the likelihood of them being exploited. The matrix of impact and likelihood is a commonly used tool in risk management to help assess and prioritize risks.

The impact of a vulnerability refers to the potential harm that could result if it were to be exploited. For smart contracts, this could include the loss of funds or assets, unauthorized access or control, or reputational damage.

The likelihood of a vulnerability being exploited is determined by considering the likelihood of an attack occurring, the level of skill or resources required to exploit the vulnerability, and the presence of any mitigating controls that could reduce the likelihood of exploitation.

Risk Level	High Impact	Medium Impact	Low Impact
High Likelihood	**Critical**	**High**	**Medium**
Medium Likelihood	**High**	**Medium**	**Low**
Low Likelihood	**Medium**	**Low**	**Low**

Risk Levels

Critical: Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation.

High: High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation.

Medium: Medium vulnerabilities are usually limited to state manipulations and, in most cases, cannot lead to asset loss. Contradictions and requirements violations. Major deviations from best practices are also in this category.

Low: Major deviations from best practices or major Gas inefficiency. These issues won't have a significant impact on code execution, don’t affect security score but can affect code quality score.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Impact Levels

High Impact: Risks that have a high impact are associated with financial losses, reputational damage, or major alterations to contract state. High impact issues typically involve invalid calculations, denial of service, token supply manipulation, and data consistency, but are not limited to those categories.

Medium Impact: Risks that have a medium impact could result in financial losses, reputational damage, or minor contract state manipulation. These risks can also be associated with undocumented behavior or violations of requirements.

Low Impact: Risks that have a low impact cannot lead to financial losses or state manipulation. These risks are typically related to unscalable functionality, contradictions, inconsistent data, or major violations of best practices.

Likelihood Levels

High Likelihood: Risks that have a high likelihood are those that are expected to occur frequently or are very likely to occur. These risks could be the result of known vulnerabilities or weaknesses in the contract, or could be the result of external factors such as attacks or exploits targeting similar contracts.

Medium Likelihood: Risks that have a medium likelihood are those that are possible but not as likely to occur as those in the high likelihood category. These risks could be the result of less severe vulnerabilities or weaknesses in the contract, or could be the result of less targeted attacks or exploits.

Low Likelihood: Risks that have a low likelihood are those that are unlikely to occur, but still possible. These risks could be the result of very specific or complex vulnerabilities or weaknesses in the contract, or could be the result of highly targeted attacks or exploits.

Informational

Informational issues are mostly connected to violations of best practices, typos in code, violations of code style, and dead or redundant code.

Informational issues are not affecting the score, but addressing them will be beneficial for the project.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Appendix 2. Scope

The scope of the project includes the following smart contracts from the provided repository:

Initial review scope

Requirements	diagram-flow-erc20.png, ReadME.md
Contracts	File: ./contracts/Bridge.sol SHA3: 66227cd12ee6a873d9ae07b3abba5e723d538b7014951a05c6f30f746b46f13f File: ./contracts/ERC20Safe.sol SHA3: 0adbd576c77960109772d2d7ee7c9a42101f5cd2cbea7bec73add5d6059c87ea File: ./contracts/WLAC.sol SHA3: a6776b3bb9a22e5e052fc985581690d1214f6e5b07f39ae54e28283dbbc3bc87 File: ./contracts/WrapperLAC.sol SHA3: 9dba03f76628f69d9053e760eb3801b979ef76156008a4aadc702cca2cde316b File: ./contracts/handlers/ERC20Handler.sol SHA3: eb7910fa9b344ffde2b09c9a415bce551c8b5eb0855be28304929a55dfce85e8 File: ./contracts/handlers/ERC20HandlerHelpers.sol SHA3: 5eba43d1c9c648b6c16ca9f16348619e5a334e0df2ab47d3fdf2af6952bbdf41 File: ./contracts/interfaces/IBridge.sol SHA3: f442ca3b8a94aa31330fd022e371b5b6fcae0304868e4e14d195fbe1f9a8d6fc File: ./contracts/interfaces/IDepositExecute.sol SHA3: 0284a26622d71cdc810d11721807a31c71ab2483fc78c7b411af91f894e8d5e7 File: ./contracts/interfaces/IERC20Handler.sol SHA3: cbaa11e6424c63d0711c0166f7e054dd81f06989f69dbb9b77da2cdd68a3ac57 File: ./contracts/interfaces/IERCHandler.sol SHA3: 8bf71477ad8d890fcea4eac73713b2c4c4a891c05264967ade810d7eb1c9ff2c File: ./contracts/interfaces/IGenericHandler.sol SHA3: 406d2e8707d168f4b3c6d8d9e70090bd28c6ffef44cac82ac7bf76626745b795 File: ./contracts/interfaces/IWLAC.sol SHA3: c166c54bad67b5ea24368de29a9c3e7bcb594fabc602c5aa0c8133285c5db03a File: ./contracts/utils/AccessControlEnumerableUpgradeable.sol SHA3: 8bf81c8c1fa89074f2014cfd5be0b900a11686eb7aa2be984c15bbcf44ea376f