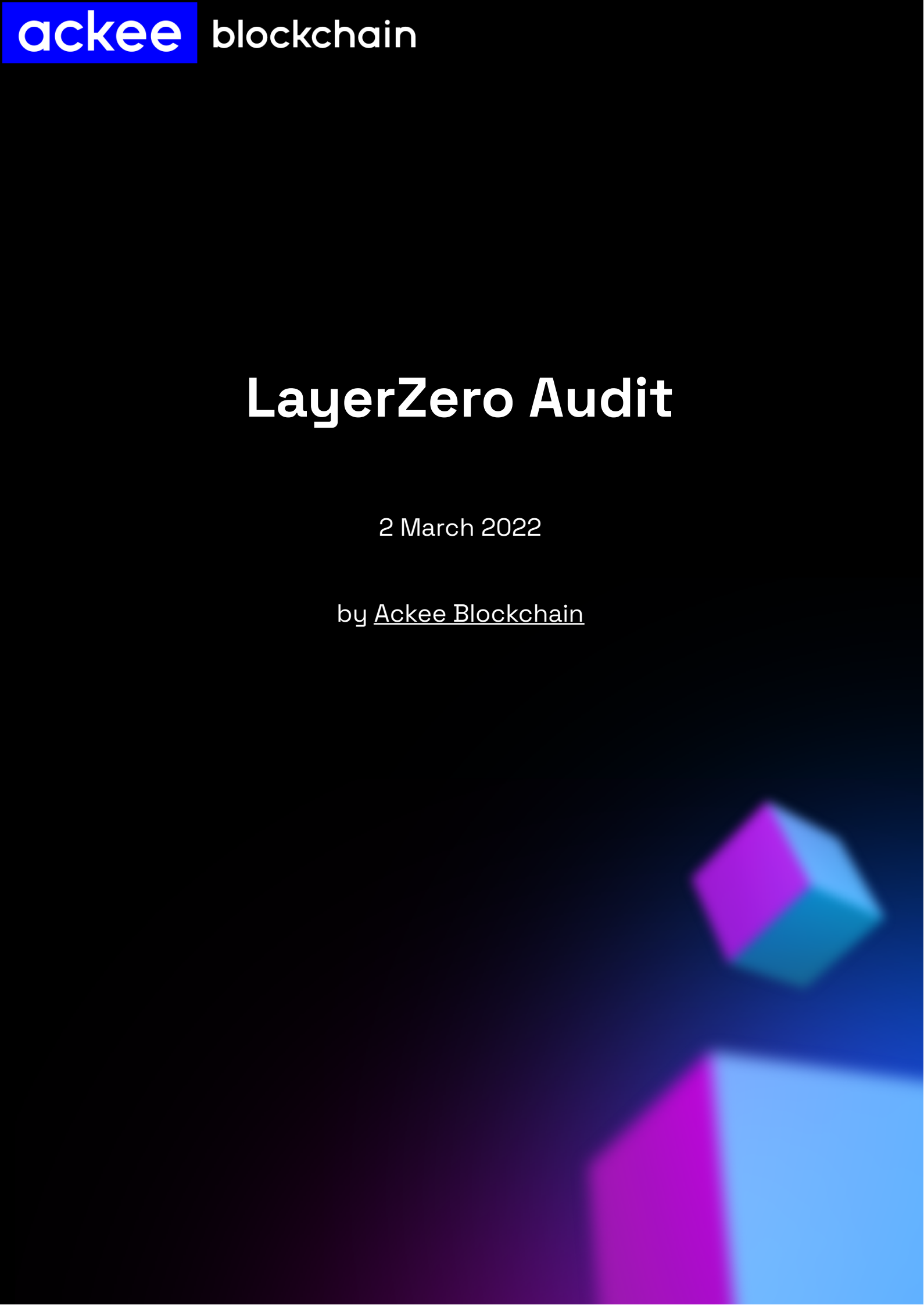
ackee blockchain

LayerZero Audit

2 March 2022

by Ackee Blockchain



Contents

1. Document Revisions

Revision	Date	Description
1.0	2 Mar 2022	Initial revision

2. Overview

This document presents our findings in reviewed contracts.

2.1 Ackee Blockchain

Ackee Blockchain is an auditing company based in Prague, Czech Republic, specialized in audits and security assessments. Our mission is to build a stronger blockchain community by sharing knowledge – we run a free certification course Summer School of Solidity and teach at the Czech Technical University in Prague. Ackee Blockchain is backed by the largest VC fund focused on blockchain and DeFi in Europe, Rockaway Blockchain Fund.

2.2 Audit Methodology

1. **Technical specification/documentation** - a brief overview of the system is requested from the client and the scope of the audit is defined.
2. **Tool-based analysis** - deep check with automated Solidity analysis tools and Slither is performed.
3. **Manual code review** - the code is checked line by line for common vulnerabilities, code duplication, best practices. The code architecture is reviewed.
4. **Local deployment + hacking** - contracts are deployed locally and we try to attack the system and break it.
5. **Unit testing** - run unit tests to ensure that the system works as expected, potentially we write our own unit tests for specific suspicious scenarios.

2.3 Review team

Member's Name	Position
Štěpán Šonský	Lead Auditor
Lukáš Böhm	Auditor
Dominik Teiml	Audit Supervisor

2.4 Disclaimer

We have put our best effort to find all vulnerabilities in the system. However, our findings shouldn't be considered as a complete list of all existing issues. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them.

3. Executive Summary

LayerZero engaged Ackee Blockchain to conduct a security review of LayerZero protocol with a total time donation of 12 engineering days.

The scope included the following repository with a given commit:

- Private repository
- a441281fa6bebeec8b9ebe170df1c5920781d196

We began our review by using static analysis tools and then took a deep dive into the logic of the contracts. During the review, we paid special attention to:

- checking if nobody can exploit the protocol
- ensuring access controls are not too weak
- checking the protocol architecture
- checking the code quality and Solidity best practices
- and looking for common issues such as data validation

LayerZero protocol is a unique approach to achieve cross-chain messages. The architecture is well designed and the overall code quality is very good. The protocol is well documented in the whitepaper, Gitbook documentation, and in the code. It helps during the audit process to focus on actual issues finding. However, we also advise using NatSpec format documentation inside the code.

During our intensive code review which was performed by two auditors in cooperation, we have not found any direct security threats. We have identified only a few hypothetical issues that aren't directly exploitable, but we have to point them out. These are general recommendations rather than security issues.

Ackee Blockchain recommends LayerZero to:

- Design some Oracle & Relayer control mechanism for independence
- Use compiler >0.8 with native SafeMath instead of library
- Use compiler no more than six months old
- Use the same compiler version across the whole project
- Do not use floating pragma
- Use 3rd party libraries wisely
- Use assembly code wisely
- Remove unused code

4. System Overview

This section contains an outline of the audited contracts. Note that this is meant for understandability purposes and does not replace project documentation.

4.1 Contracts

Contracts we find important for better understanding are described in the following section.

Endpoint
The contract handles message transmission between two different blockchains. It communicates directly with `UltraLightNode`. If the message is not successfully transmitted, the destination endpoint stores the message and blocks any future messages from the same source endpoint, until the stored message is successfully delivered, or deleted (in case of unsuccessful message resend).

User application calls `Endpoint` contract `send()` function with payload and information required to transfer to the destination blockchain.

Endpoint also maintains library configuration settings.

Endpoint contract inherits from interface `ILayerZeroEndpoint`, `ILayerZeroLibraryReciever` and Openzeppelin `Ownable`, `ReentrancyGuard`.

Custom `validVersion` modifier provides correct library version settings management.

Relayer
The contract validates transaction proof by calling ULN's `validateTransactionProof()` function. It also can be notified and provide a calculated fee amount.

The contract inherits from interface `ILayerZeroRelayer`, Openzeppelin `Ownable`, `ReentrancyGuard` and Hardhat's `Proxied` contract.

Custom modifier `onlyApproved()` requires `msg.sender` to be one of the approved owners of `Relayer`.

RelayerWithdraw
The contract works as an authenticator for `Realyer` and to withdraw protocol fees from ULN.

Treasury
The contract sets and gets native or ZRO fees. It also withdraws treasury protocol fees in native or ZRO token by calling ULN's withdraw functions.

Contract inherits from interface `ILayerZeroTreasury` and Openzeppelin `Ownable`.

UltraLightNode
`UltraLightNode` sits between `Endpoint` and users' aplication's `Relayer`. It forwards messages from `Endpoint` by `send()` function, which pays protocol, oracle and relayer fees and notify oracle with encoded data containing chain ID, library version, remote ULN address, outbound block confirmations and payload hash.
In the opposite direction, ULN's `validateTransactionProof()` function is called by `Relayer` to validate LayerZero message data and forward them to the `Endpoint`. It can be done because ULN maintains library configuration for the source chains and destination addresses.

The contract provides external fees withdraw functions for `Treasury`, `Oracle` and `Realyer`. There are also configuration and view functions for managing libraries and fees settings.

ULN contract inherits from interface `ILayerZeroEndpointLibrary`, `ILayerZeroUltraLightNodeV1` and Openzeppelin `Ownable`, `ReentrancyGuard`.

Modifier `onlyEndpoint()` requires `msg.sender` to be the `Endpoint` address.

chainlink/ChainlinkOracleClient
This contract inherits from `ILayerZeroOracle` interface, `ChainlinkClient` and from OpenZeppelin `Ownable` and `ReentrancyGuard`.

Inside the contract, there are two defined structs - `Job` and `CachedAddressString`.

Modifier `onlyULN()` requires `msg.sender` to be the address of `UltraLightNode`. This modifier is used only for `notifyOracleOfBlock()` function which sends `Chainlink.Request` to the `job.oracle` address using `sendChainlinkRequestTo()`.

The contract contains whitelisted address map `approvedAddresses` which are able to update block hashes of chains. In the `constructor`, the `msg.sender` is added into this whitelist. Also only the owner can edit this whitelist.

libraries/Buffer
A library for working with mutable byte buffers. It's basically a fork of ensdomains/buffer with an additional function `writeRawBytes()`, which is almost a 1:1 copy of `write()` method.

libraries/EVMValidator
EVMValidator is actually a contract, not a library. And contains `validateProof()` function which provides Merkle Patricia Trie data validation and returns `LayerZeroPacket.Packet`.

libraries/LayerZeroPacket
Library which contains `Packet` struct with all necessary transmission data like src/dst chain IDs, nonce, src/dst addresses, ULN address and payload. Also contains a function `getPacket()` for composing a `Packet` object from input params.

utils/Decoder
Just a simple wrapper of `EthereumDecoder.sol` library.

4.2 Actors

Owner / LayerZero
The owner deploys contracts to the network and has extra privileges in some contracts.

- Endpoint
  - Library upgrade, versioning
- Treasury
  - Set fees
  - Set BP
- UltraLightNode
  - Configuration
  - Set LayerZero token address
  - Set treasury address
  - Set relayer fee contract
  - Set libraries' addresses
  - Set adapter params for chain
  - Set remote Ultra Light Node
  - Set chain address size
- ChainlinkOracleClient
  - Approve token
  - Withdraw
  - Set Ultra Light Node

- Withdraw tokens
- Withdraw oracle quoted fee
- Set job
- Set price
- Set approved addresses

Oracle
Oracles are third party, off-chain entities, which has ability to read and transfer the block headers between chains.

Relayer
Relayers are off-chain entities, similar to oracles. But instead of block headers, they fetch transaction proofs between chains.

User
User role means any external address in the network, which can interact with the protocol.

4.3 Trust model

Users need to put trust in the Oracle and Relayer, if they have not implemented it itself. On the other hand, flow from the Endpoint to the Oracle/Relayer is trustless. Owners of the smart contracts have only the privilege to manage libraries configuration and to manage fees.

The critical point of LayerZero trust model is relayer and oracle independence. In case that both parties are cooperating, cross-chain messages could be manipulated.

5. Vulnerabilities risk methodology

Each finding contains an *Impact* and *Likelihood* ratings.

If we have found a scenario in which the issue is exploitable, it will be assigned an impact of *High*, *Medium*, or *Low*, based on the direness of the consequences it has on the system. If we haven't found a way, or the issue is only exploitable given a change in configuration (such as deployment scripts, compiler configuration, use of multi-signature wallets for owners, etc.) or given a change in the codebase, then it will be assigned an impact rating of *Warning* or *Informational*.

Low to *High* impact issues also have a *Likelihood* that measures the probability of exploitability during runtime.

5.1 Finding classification

The full definitions are as follows:

Impact

High
Conditions that activates the issue will lead to undefined or catastrophic consequences for the system.

Medium
Conditions that activates the issue will result in consequences of serious substance.

Low
Conditions that activates the issue will have outcomes on the system that are either recoverable or don't jeopardize its regular functioning.

Warning
The issue cannot be exploited given the current code and/or configuration (such as deployment scripts, compiler configuration, use of multisignature wallets for owners, etc.) but could be a security vulnerability if these were to change slightly. If we haven't found a way to exploit the issue given the time constraints, it might be marked as "Warning" or higher, based on our best estimate of whether it is currently exploitable.

Informational
The issue is on the borderline between code quality and security. Examples

include insufficient logging for critical operations. Another example is that the issue would be security-related if code or configuration (see above) was to change.

Likelihood

High
The issue is exploitable by virtually anyone under virtually any circumstance.

Medium
Exploiting the issue currently requires non-trivial preconditions.

Low
Exploiting the issue requires strict preconditions.

6. Findings

This section contains the list of discovered findings. Unless overridden for purposes of readability, each finding contains:

- a *Description*,
- an *Exploit scenario*, and
- a *Recommendation*

Many times, there might be multiple ways to solve or alleviate the issue, with varying requirements in terms of the necessary changes to the codebase. In that case, we will try to enumerate them all, making clear which solves the underlying issue better (albeit possibly only with architectural changes) than others. Issues can be also acknowledged by developers as not a risk.

Summary of Findings

ID		Type	Impact	Likelihood	Status
H1	Oracle & Relayer conspiracy	Trust model	High	Low	Reported
W1	Incorrect use of SafeMath library	Overflow	Warning	N/A	Reported
W2	Outdated compiler	Compiler	Warning	N/A	Reported
W3	Many different compiler versions	Compiler	Warning	N/A	Reported
W4	Usage of third party libs	Library	Warning	N/A	Reported
W5	Unintended feature - Renounce ownership	Access control	Warning	N/A	Reported
I1	EVMValidator is not a library	Syntax	Info	N/A	Reported
I2	Functions naming	Syntax	Info	N/A	Reported
I3	Constant naming	Syntax	Info	N/A	Reported

H1 - Oracle + Relayer conspiracy

Impact:	High	**Likelihood:**	Low
Target:	Oracle, Relayer	**Type:**	Trust model

Description
To achieve trustless message delivery, the protocol requires Oracle & Relayer to be independent of each other. In case they aren't, protocol can be manipulated, so applications built on top of it as well. This requirement is also mentioned in the white paper.

Exploiting scenario
This hypothetical issue is not a direct threat from our point of view, but the probability of creating different exploiting scenarios is non-zero.

Recommendation
We recommend auditing of applications built on top of LayerZero and giving them some kind of trustworth status when Oracles & Relayers are demonstrably independent. Or try to design some algorithmic control mechanism of independence.

W1 - Incorrect use of SafeMath library

Impact:	Warning	**Likelihood:**	N/A
Target:	Relayer.sol	**Type:**	Overflow

Description
Endpoint uses SafeMath library for uint64, uint128. Usage of SafeMath is necessary only if solidity version <0.8. SafeMath operations are implemented only for uint256. It can lead to overflow of smaller size uint variables. It also consumes more gas due to casting. However in this specific case, all equations results are assigned to uint256 which prevents overflow.

Exploiting scenario
uint8 x = 200;
uint8 y = 100;
uint8 result = uint8(x.add(y));

Result overflows without transaction revert. However, remember that typecasting overflow is still present in Solidity >0.8.

Recommendation
Avoid using smaller integer types with SafeMath library, always use uint256. Even better is to use newer Solidity versions. In Solidity > 0.8.0 integer overflow is checked by the compiler.

W2 - Outdated compiler

Impact:	Warning	**Likelihood:**	N/A
Target:	/**/*	**Type:**	Compiler

Description
The project uses Solidity compiler <0.8, which does not contain the latest security fixes and native overflow/underflow handling.

Exploiting scenario
Bytecode compiled with an outdated compiler can contain critical security issues, which could be exploited by the attacker or could lead to system misbehavior.

Recommendation
We recommend using compiler 0.8 at minimum, containing the latest bug fixes and integer overflow/underflow handling.

W3 - Many different compiler versions

Impact:	Warning	**Likelihood:**	N/A
Target:	/**/*	**Type:**	Compiler

Description
The project uses many different Solidity compiler versions, even with floating pragma.

Exploiting scenario
During the deployment, contracts could be compiled with a different compiler version than they were developed and tested with. This could lead to unexpected behaviour or even security threats.

Recommendation
Use the same, fixed compiler version across the whole project.

W4 - Usage of third-party library

Impact:	Warning	**Likelihood:**	N/A
Target:	Buffer.sol	**Type:**	Library

Description

A few years ago, a critical issue was discovered in Buffer.sol library init function by ConsenSys. The issue has been fixed, and it is not exploitable anymore.

Recommendation

It is strongly recommended not to use third-party libraries unless they are heavily used and well-debugged ones (OpenZeppelin etc.). Especially third party code containing a lot of assembly code should be handled very carefully. We also recommend removing unused code from this library.

H2 - Unintended feature - Renounce ownership

Impact:	High	**Likelihood:**	Low
Target:	*.sol implements Ownable	**Type:**	Access control

Description
The OpenZeppelin's Ownable pattern contains `renounceOwnership()` which sets the owner address to `address(0)`. This could lead to irreversible damage of the contract.

Exploiting scenario
This is not directly exploitable issue, but can be considered as unintended feature of the system, which can be called accidentally.

Recommendation
We recommend using multisig wallet for the owner to avoid accidental `renounceOwnership()` call. Or it can be handled by overriding the `renounceOwnership()` function in contracts inherited from `Ownable`.

I1 - Not a Library

Impact:	Warning	**Likelihood:**	N/A
Target:	EVMValidator.sol	**Type:**	Syntax

Description
EVMValidator is in ./library/ directory, but it is actually not a library.

Recommendation
In ./library/ directory, only libraries should be stored, not contracts to avoid confusion.

I2 - Function naming convention

Impact:	Warning	**Likelihood:**	N/A
Target:	ChainlinkOracleClient.sol	**Type:**	Naming convention

Description
The contract uses incorrect function naming convention (underscores).

Recommendation
Use camelCase naming in solidity functions.

I3 - Lower case constant variable

Impact:	Warning	**Likelihood:**	N/A
Target:	EVMValidator.sol	**Type:**	Naming convention

Description
Contract's variable name `bytes32 public constant PacketSignature` is in Lower Case.

Recommendation
All constant variable names should be UPPER_CASE.

ackee blockchain

Thank You

Ackee Blockchain a.s.

Prague, Czech Republic

hello@ackeeblockchain.com

https://discord.gg/z4KDUbuPxq