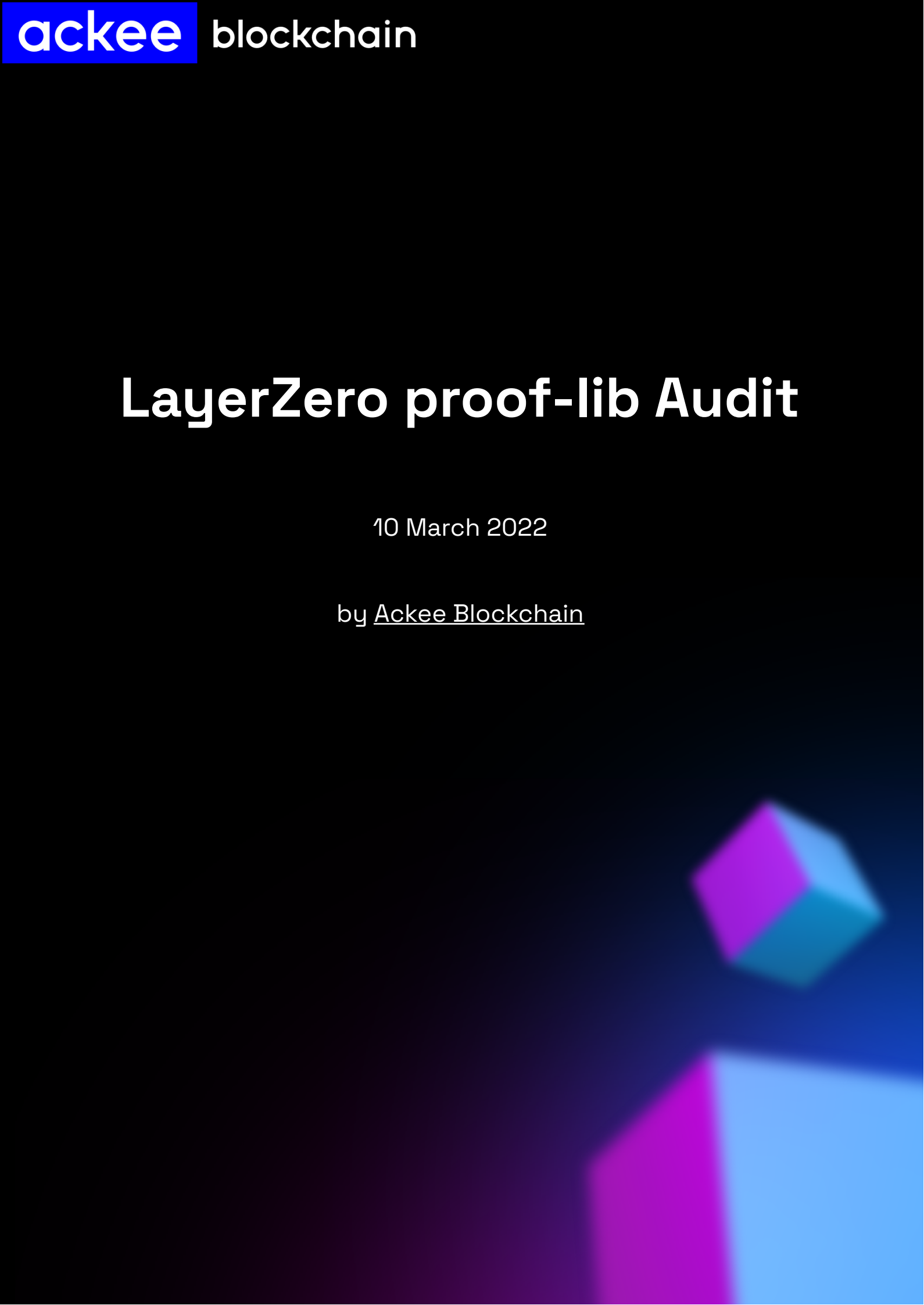
ackee blockchain

LayerZero proof-lib Audit

10 March 2022

by Ackee Blockchain



Contents

1. Document Revisions

Revision	Date	Description
1.0	10 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
Lukáš Böhm	Auditor
Josef Gattermayer	Audit Supervisor

2.4 Disclaimer

We have put our best effort to find all vulnerabilities in the system. However, our findings should not 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 4 engineering days.

The scope included the following repository with a given commit:

- Private repository
- ~~86ea33e73afe15673d9aa66af7f926ee4f6b480e~~
- fa312693b62e4b7dda6a5bfeab6ebb22c1a43374

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 the access control of the contract,
- looking for common issues such as data validation,
- checking the code quality and Solidity best practices,
- ensuring the library logic works as expected.

LayerZero proof-lib contains one contract, one interface and four libraries. RLPDecode.sol and Buffer.sol are imported third-party libraries. Buffer library does not follow the best solidity practices, but the code quality of the rest is very good. Complicated parts of the code are well documented.

During our intensive code review, which one auditor performed, no direct security threat was found. However, RLPDecode library contains high severity uint underflow. It is not directly exploitable in the current audit scope, but it is still recommended to fix.

Ackee Blockchain recommends LayerZero to:

- Make library functions secure for potential future usage.
- Use compiler >0.8 with native SafeMath instead of the library.
- Use compiler no more than six months old.
- Choose better naming conventions.
- Use third-party libraries wisely.
- Use assembly code wisely.
- Remove dead 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.

MPTValidator
The contract contains three functions and a custom structure `ULNLog`. Function `validateProof` is called from ULN (knowledge from previous audit scope) to validate a transaction proof. Inside this function, input data are abi decoded to fill `_getVerifiedLog` function, which handles the verification logic and returns `ULNLog` structure as a result. Then inside `validateProof` packet signature is checked, and `ULNLog` is returned.
The third function is simply calling `_getVerifiedLog` function straight with input data.

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

utility/RLPDecode (library)
The library for RLP coding operations (modified hamdiallam/solidity-rlp library). It defines two structures `RLPItem` and `Iterator`. These structures are used in functions for handling RLP encoded objects. The library can even create one of these structures.

utility/UltraLightNodeEVMDecoder (library)
The Library to get the receipt log. Structure `Log` is defined inside the library. Function `toReceiptLog()` creates log from argument bytes. Function `getReceiptLog()` looking for an item on a third position in the RLP Iterator object created from input argument bytes data. Then `toReceiptLog()` is called.

utility/LayerZeroPacket (library)
This library defines `Packet` structure that stores all necessary Layer Zero packet information. Function `getPacket()` uses the assembly part to load the data, and the Buffer library is used for creating a complete `Packet` to return.

4.2 Actors

In the current scope, there is only one contract `MPTValidator`. No actor has any additional privilege.

4.3 Trust model

There is only one contract in the audit scope, as said before. The contract is made in a trustless way. Libraries are trustless by default.

5. Vulnerabilities risk methodology

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

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 have not 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 complete definitions are as follows:

Impact

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

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

Low
Conditions that activate the issue will have outcomes on the system that are either recoverable or do not 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 multi-signature wallets for owners, etc.) but could be a security vulnerability if these were to change slightly. If we have not 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, clarifying 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	Unsigned integer underflow	Underflow	High	low	Acknowledged
W1	Outdated compiler	Compiler	Warning	N/A	Acknowledged
W2	Usage of third party libraries	Library	Warning	N/A	Acknowledged
I1	Commented out code	Syntax	Info	N/A	Fixed

H1 - Unsigned integer underflow

Impact:	High	**Likelihood:**	Low
Target:	RLPDecode.sol	**Type:**	Underflow

Description
If library functions `toRLPItem()` or `typeOffset()` are called with the input equal to zero length bytes, underflow occurs in variable `len`. RLPItem structure is then returned with `RLPItem.len` parameter with value of MAX_INT.

Exploiting scenario
Both functions are always called with `iterator()` function in the current scope. This function requires an item to be a list. It handles `isList()` function, which returns `False` for zero-length bytes.
This issue is not a direct threat to this audit scope.

Recommendation
Nevertheless, we recommend using zero-length checks in the functions or the safeMath library for uint. It is a good practice to make the library secure and ready for potential future usage.

Client's response
Function toRPLItem() is used only in this scope.

W1 - 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.

Client's response
The bug fix of solidity >0.8 does not affect our implementation to our understanding. And in our benchmark, it consumes more gas in general. So we decide to stay at 0.7.6

W2 - Usage of the 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.

The structure is not named in CapWorld format and has the same name as the library itself. In structure, there is a defined variable called buff. It causes a confusing piece of codes like:
`Buffer.buffer.buf`
The structure `buffer` is an argument to functions named `buf`. It causes even more confusing code:
`bytes memory oldbuf = buf.buf;`
Assembly parts of code do not use camelCase for variable naming.

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

Client's response
Acknowledged. It is an import library and we want to keep its code and license.

I1 - Commented out code

Impact:	Informational	**Likelihood:**	N/A
Target:	RLPDecode.sol	**Type:**	Dead dode

Description
Code of the default `toRLPItem()` function from hamdiallam/solidity-rlp is commented out above the actual function.

Recommendation
Useless code or comments should not be a part of the contract/library. Delete all dead code.

Client's response
Noted.

ackee blockchain

Thank You

Ackee Blockchain a.s.

Prague, Czech Republic

hello@ackeeblockchain.com

https://discord.gg/z4KDUbuPxq