ackee blockchain

Axelar

Token Linker

by Ackee Blockchain

26.9.2022

Contents

1. Document Revisions

0.1	Draft report	12.9.2022
1.0	Final report	14.9.2022
1.1	Fix review	26.9.2022

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, specializing in audits and security assessments. Our mission is to build a stronger blockchain community by sharing knowledge – we run free certification courses School School of Solana, 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 and the code architecture is reviewed.
4. **Local deployment + hacking** - the contracts are deployed locally and we try to attack the system and break it.
5. **Unit and fuzzy testing** - run unit tests to ensure that the system works as expected, potentially write missing unit or fuzzy tests.

2.3. Review team

Member's Name	Position
Jan Kalivoda	Lead Auditor
Josef Gattermayer, Ph.D.	Audit Supervisor

2.4. Disclaimer

We've 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

The objective of the audit is Token Linker, a set of contracts that is used to link any tokens across two or more different EVM-compatible chains on a one-to-one basis using only Axelar's general message passing.

Axelar engaged Ackee Blockchain to perform a security review of the Token Linker changes with a total time donation of 4 engineering days in a period between September 5 and September 9, 2022 and the lead auditor was Jan Kalivoda.

The full-repository audit of Token Linker was performed on the commit 58acfd0.

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

- the correctness of the upgradeability pattern and lookup for the implementations,
- the correctness of the deployment process via the factory,
- detecting possible reentrancies in the code,
- ensuring access controls are not too relaxed or too strict,
- looking for common issues such as data validation.

Our review resulted in 8 findings, ranging from Info to Medium severity.

Ackee Blockchain recommends Axelar:

- update documentation including Natspec comments,
- do not merge or release the code that is not being used,

- address all other reported issues.

Update September 26, 2022

Axelar provided an updated codebase that addresses some issues from this report. See Appendix C for a detailed discussion of the exact status of each issue.

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.

TokenLinkerFactory.sol

The contract allows deploying token linkers. The token linker can be deployed as managed by the factory or not. The factory-managed token linker looks up for implementation to the factory (FactoryLookupProxy). Otherwise, the token linker is deployed as an upgradeable contract (SelfLookupProxy). Also, there is an option to deploy the token linker on multiple chains by passing remote deployment data to the function and calling Gateway.

4.2. Actors

This part describes actors of the system, their roles, and permissions.

Gateway

Gateway is Solidity CGP Gateway project by Axelar. It is used only for the `callContract` function on the Gateway.

4.3. Trust model

The contracts don't have any ownership or other escalated privileges.

5. Vulnerabilities risk methodology

A *Severity* rating of each finding is determined as a synthesis of two sub-ratings: *Impact* and *Likelihood*. It ranges from *Informational* to *Critical*.

If we have found a scenario in which an issue is exploitable, it will be assigned an impact rating 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 *Info*.

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

5.1. Finding classification

The full definitions are as follows:

Severity

		Likelihood			
		High	**Medium**	**Low**	**-**
Impact	**High**	Critical	High	Medium	-
	Medium	High	Medium	Medium	-
	Low	Medium	Medium	Low	-
	Warning	-	-	-	Warning
	Info	-	-	-	Info

Table 1. Severity of findings

Impact

- **High** - Code that activates the issue will lead to undefined or catastrophic consequences for the system.
- **Medium** - Code that activates the issue will result in consequences of serious substance.
- **Low** - Code 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 multi-signature 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 a "Warning" or higher, based on our best estimate of whether it is currently exploitable.
- **Info** - 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 overriden 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 solve the underlying issue better (albeit possibly only with architectural changes) than others.

Summary of Findings

	Severity	Impact	Likelihood
M1: TokenLinker has insufficient data validation	Medium	High	Low
M2: Looping through arrays with different lengths	Medium	High	Low
W1: Same contract names	Warning	Warning	N/A
W2: Usage of `solc` optimizer	Warning	Warning	N/A
I1: Missing implementation and documentation	Info	Info	N/A
I2: Unused custom error	Info	Info	N/A
I3: Inconsistent usage of (pre/post)incrementation	Info	Info	N/A

	Severity	Impact	Likelihood
I4: Typos	Info	Info	N/A

Table 2. Table of Findings

M1: TokenLinker has insufficient data validation

Medium severity issue

Impact:	High	Likelihood:	Low
Target:	Token Linker/contracts/token-linkers/TokenLinker.sol	Type:	Data validation

Listing 1. Excerpt from TokenLinkerFactory.init

```
46     function init(
47         address gatewayAddress_,
48         address gasServiceAddress_,
49         address[] memory factoryManagedImplementations_,
50         address[] memory upgradableImplementations_
51     ) external {
52         if (gatewayAddress != address(0)) revert AlreadyInitialized();
53         if (gatewayAddress_ == address(0) || gasServiceAddress_ ==
   address(0)) revert ZeroAddress();
54         gatewayAddress = gatewayAddress_;
55         gasService = IAxelarGasService(gasServiceAddress_);
```

Description

The `TokenLinker` contract and its subclasses do not perform any data validation of `gatewayAddress_` and `gasServiceAddress_` in its constructor.

Exploit scenario

By accident, an incorrect `gatewayAddress_` is passed to the constructor. Instead of reverting, the call succeeds.

Recommendation

Add more stringent data validation for `gatewayAddress_` and

`gasServiceAddress_`). At the very least this would include a zero-address check. Like it is done in `TokenLinkerFactory` (see Listing 1 line 53).

Ideally, we recommend defining a getter such as `contractId()` (which is already implemented in token linker contracts) that would return a hash of an identifier unique to the (project, contract) tuple[1]. This will ensure the call reverts for most incorrectly passed values.

Go back to Findings Summary

M2: Looping through arrays with different lengths

Medium severity issue

Impact:	High	Likelihood:	Low
Target:	Token Linker/contracts/TokenLinkerFactory.sol	Type:	Data validation

Listing 2. Excerpt from TokenLinkerFactory.init

```
56          uint256 length = factoryManagedImplementations_.length;
57          for (uint256 i; i < length; ++i) {
58              _checkImplementation(factoryManagedImplementations_[i], i);
59              _checkImplementation(upgradableImplementations_[i], i);
60          }
```

Description

The for-loop in the `init` function (see Listing 2) is iterating through two different arrays that can have different lengths.

Exploit scenario

The `factoryManagedImplementations_` array is shorter than the `upgradableImplementations_` array. As a result, the `init` function will not revert and the `upgradableImplementations_` array remains not fully initialized (and undetected).

Recommendation

Add check for the same size of arrays, if they are going to be always the same. If not, use two different for-loops.

Go back to Findings Summary

W1: Same contract names

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

Description

The project is using Hardhat, so in this specific case, it is not an issue.

However, if somebody will want to reuse the code or if the framework will be changed in the future, it could be an issue. Truffle has a serious bug if a project has two or more contracts with the same names. The compilation artifacts will contain only the last created artifact because it overwrote the previous ones. And for example in Brownie it is not possible to compile the project at all.

Vulnerability scenario

Bob is using Truffle for the compilation of his project (instead of hardhat) and he has two contracts, both named `AToken`. During the compilation, Truffle overwrites the first generated artifact for `AToken`. As a result, it can cause an unknown behavior for the protocol.

Recommendation

Avoid using the same names for contracts.

Go back to Findings Summary

W2: Usage of `solc` optimizer

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

Description

The project uses `solc` optimizer. Enabling `solc` optimizer may lead to unexpected bugs.

The Solidity compiler was audited in November 2018, and the audit concluded that the optimizer may not be safe.

Vulnerability scenario

A few months after deployment, a vulnerability is discovered in the optimizer. As a result, it is possible to attack the protocol.

Recommendation

Until the `solc` optimizer undergoes more stringent security analysis, opt-out using it. This will ensure the protocol is resilient to any existing bugs in the optimizer.

Go back to Findings Summary

I1: Missing implementation and documentation

Impact:	Info	Likelihood:	N/A
Target:	**/*	Type:	Quality assurance

Listing 3. Excerpt from TokenLinkerRouter.TokenLinkerRouter

```
 8 contract TokenLinkerRouter {
 9     bytes32[] public tokenLinkerIds;
10
11     ITokenLinkerFactory public immutable factory;
12
13     constructor(bytes32[] memory tokenLinkerIds_) {
14         factory = ITokenLinkerFactory(msg.sender);
15         tokenLinkerIds = tokenLinkerIds_;
16     }
17 }
```

Description

The project contains unfinished contracts (see Listing 3) and uncomplete documentation (the `README.md` file).

Recommendation

Write the documentation to match the code and delete the unused code.

Go back to Findings Summary

I2: Unused custom error

Impact:	Info	Likelihood:	N/A
Target:	Token Linker/contracts/TokenLinker MintBurnExternal.sol	Type:	Quality assurance

Listing 4. Excerpt from TokenLinkerMintBurnExternal._takeToken

```
36          if (!transferred || tokenAddress.code.length == 0) revert
   ('BurnFailed()');
```

Description

The contract has defined a custom error called `BurnFailed`, but it is not used. Instead of it, simple revert message is passed (see Listing 4).

Recommendation

Write the documentation to match the code and delete the unused code.

Go back to Findings Summary

I3: Inconsistent usage of (pre/post)incrementation

Impact:	Info	Likelihood:	N/A
Target:	Token Linker/contracts/TokenLinkerFactory.sol	Type:	Gas optimization

Listing 5. Excerpt from TokenLinkerFactory.init

```
57          for (uint256 i; i < length; ++i) {
```

Listing 6. Excerpt from TokenLinkerFactory.deployMultichain

```
129          for (uint256 i; i < length; i++) {
```

Description

The contract is using (pre/post)incrementation inconsistently in its for-loops. Pre-incrementation is the preferred way since it is cheaper for execution.

Recommendation

Replace post-incrementation with pre-incrementation in for-loops (see Listing 6).

Go back to Findings Summary

I4: Typos

Impact:	Info	Likelihood:	N/A
Target:	Token Linker/contracts/TokenLinkerNative.sol	Type:	Quality assurance

Listing 7. Excerpt from TokenLinkerNative._execute

```
84             emit ReveivingWithData(sourceChain, recipient, amount, from,
   data);
```

Listing 8. Excerpt from TokenLinkerNative._execute

```
133             if (gasAmount < address(this).balance) revert
    InsufficinetAmountForGas();
```

Description

- Listing 7 - The error name `ReveivingWithData` should be probably `ReceivingWithData`.
- Listing 8 - The event name `InsufficinetAmountForGas` should be probably `InsufficientAmountForGas`.

Recommendation

Fix the typos.

Go back to Findings Summary

Endnotes

[1] An example would be `keccak256("Axelar - Solidity CGP Gateway")`

Appendix A: How to cite

Please cite this document as:

Ackee Blockchain, Axelar: Token Linker, 26.9.2022.

Appendix B: Glossary of terms

The following terms might be used throughout the document:

Superclass/Ancestor of C

A contract that C inherits/derives from.

Subclass/Child of C

A contract that inherits/derives from C.

Syntactic contract

A Solidity contract. May have an inheritance chain, and may be deployed.

Deployed contract

An EVM account with non-zero code. If its source was written in Solidity, it was created through at least one syntactic contract. If that contract had superclasses (parents), it would be composed of multiple syntactic contracts.

Init/initialization function

A non-constructor function that serves as an initializer. Often used in upgradeable contracts.

External entrypoint

A `public` or `external` function.

Public/Publicly-accessible function/entrypoint

An `external` or `public` function that can be successfully executed by any network account.

Mutating function

A non-`view` and non-`pure` function.

Appendix C: Fix Review

On September 26, 2022, Ackee Blockchain reviewed Axelar's fixes for the issues identified in this report. The following table summarizes the fix review. The updated commit was `0a9aab2z`.

Fix log

Id	Severity	Impact	Likelihood	Status
M1: TokenLinker has insufficient data validation	Medium	High	Low	Fixed
M2: Looping through arrays with different lengths	Medium	High	Low	Fixed
W1: Same contract names	Warning	Warning	N/A	Acknowledged
W2: Usage of `solc` optimizer	Warning	Warning	N/A	Acknowledged
I1: Missing implementation and documentation	Info	Info	N/A	Acknowledged
I2: Unused custom error	Info	Info	N/A	Fixed
I3: Inconsistent usage of (pre/post)incrementation	Info	Info	N/A	Fixed
I4: Typos	Info	Info	N/A	Fixed

Table 3. Table of fixes

M1F: TokenLinker has insufficient data validation

Medium severity issue

Impact:	High	Likelihood:	Low
Target:	M1: TokenLinker has insufficient data validation	Type:	Data validation

Description

The issue is fixed by adding the following zero-address checks.

```
if(gatewayAddress_ == address(0) || gasServiceAddress_ == address(0))
revert ZeroAddress();
```

However, we recommend adding more stringent validation in the future such as proposed in the original finding.

Go back to Fix log

M2F: Looping through arrays with different lengths

Medium severity issue

Impact:	High	Likelihood:	Low
Target:	M2: Looping through arrays with different lengths	Type:	Data validation

Description

The issue is fixed by adding the following condition.

```
if(length != upgradableImplementations_.length) revert LengthMismatch();
```

Go back to Fix log

ackee

Thank You

Ackee Blockchain a.s.


Prague, Czech Republic


hello@ackeeblockchain.com


https://discord.gg/z4KDUbuPxq