Axelar

Token Linker & Forecall Service

by Ackee Blockchain

12.8.2022

Contents

1. Document Revisions

1.0	Final report	12.8.2022

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, 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 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 first 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. The second objective of the audit is a Forecall Service that allows an application that receives messages from Axelar to accept messages before they have been approved on Gateway.

Axelar engaged Ackee Blockchain to perform a security review of the Token Linker and the Forecall Service with a total time donation of 5 engineering days in a period between August 1 and August 5, 2022 and the lead auditor was Jan Kalivoda.

The audit was performed on two repositories with the following commits and files.

- Token Linker - 5e1d4bb
  - contracts/*.sol
- Forecall Service - db238d7
  - contracts/executables/AxelarForecallable.sol

We began our review by using static analysis tools, namely Slither and the solc compiler. This yielded issues such as M1: The `tokenAddress` is missing zero-address check. We then took a deep dive into the logic of the contracts. During the review, we paid special attention to:

- execution logic in Forecall Service is matching requirements,
- token linking is not leading to unauthorized access to funds,
- 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 High severity. The most severe one is a violation of an intended behavior in Forecall Service (see H1: The `forecall` and `forecallWithToken` can be called repeatedly with a same payload).

Ackee Blockchain recommends Axelar:

- add documentation including Natspec comments,
- write a more extensive test suite,
- address all other reported issues.

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.

TokenLinker

An abstract contract for other token linker contracts. It is a subclass of AxelarExecutable. It allows to call `execute` which triggers `_giveToken` function that is responsible for token retrieval and `sendToken` function that is responsible for token sending. Both of these functions (`_giveToken` and `_takeToken`) are implemented in child contracts.

TokenLinkerLockUnlock

The contract is for a pre-existing ERC20, that needs to be locked/unlocked on a chain. The `_giveToken` and `_takeToken` functions are implemented as a simple call on `tokenAddress`.

TokenLinkerMintBurn

The contract is for a newly deployed ERC20, which can be minted/burned by Axelar's token linker (and only this token linker). Since it is ERC20 token, the `_giveToken` and `_takeToken` functions are implemented as a simple burn and mint function on ERC20 contract

TokenLinkerNative

The contract is for a native currency of a chain, such as ETH for Ethereum. Via `_giveToken` token can be transferred to the user and the deposit is working through the payable function `updateBalance`, or `sendToken` to call atomically with `_takeToken`.

TokenLinkerProxy

The proxy contract with the upgradeability pattern from Axelar core project. This proxy is planned to be deployed on each chain (that is going to be linked) with a same address.

AxelarForecallable

It allows an application that receives messages from Axelar to accept messages before they have been approved on Gateway from a configured forecaller address. This could be useful when receiving messages from chains with long confirmation times, such as Ethereum.

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 for validating calls in `execute` functions.

4.3. Trust model

The contracts don’t have any ownership or other escalated privileges. The only thing that should be ensured is a correctly chosen Gateway and thus users have to trust Gateway and specific implementations such as AxelarForecallable.

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
H1: The `forecall` and `forecallWithToken` can be called repeatedly with a same payload	High	Medium	High
M1: The `tokenAddress` is missing zero-address check	Medium	High	Low
M2: TokenLinker has insufficient data validation	Medium	High	Low
W1: Usage of `solc` optimizer	Warning	Warning	N/A
W2: Floating dependency on AxelarGateway	Warning	Warning	N/A

	Severity	Impact	Likelihood
W3: Mulitple ways to receive ether can lead to loss of funds	Warning	Warning	N/A
W4: The `forecall` function is missing any checks by default	Warning	Warning	N/A
I1: Typo in the error name	Info	Info	N/A

Table 2. Table of Findings

H1: The `forecall` and `forecallWithToken` can be called repeatedly with a same payload

High severity issue

Impact:	Medium	Likelihood:	High
Target:	Forecall Service/contracts/executables/AxelarForecallable.sol	Type:	Data validation

Listing 1. Excerpt from AxelarForecallable.forecall

```
43     function forecall(
44         string calldata sourceChain,
45         string calldata sourceAddress,
46         bytes calldata payload,
47         address forecaller
48     ) external {
49         _checkForecall(sourceChain, sourceAddress, payload, forecaller);
50         if (getForecaller(sourceChain, sourceAddress, payload) !=
   address(0)) revert AlreadyForecalled();
51         _setForecaller(sourceChain, sourceAddress, payload, forecaller);
52         _execute(sourceChain, sourceAddress, payload);
53     }
```

Description

The contract is not checking if the `forecaller` address is not equal to zero-address. As a result, the check preventing double execution on line 50 (see Listing 1) can be bypassed. This violation can be performed by anyone and anytime since `forecall` (resp. `forecallWithToken`) is a publicly-accessible function. However, after discussion with the team, double execution should not cause any critical scenario.

Exploit scenario

Bob calls the `forecall` function with the `forecaller` address equal to zero-address and some specific payload A (the remaining function parameters). This payload gets executed. Since he passed `forecaller` as a zero-address he executes payload A again and repeatedly[1].

Recommendation

Add a zero-address check for the `forecaller` address in both functions (`forecall` and `forecallWithToken`).

Go back to Findings Summary

M1: The `tokenAddress` is missing zero-address check

Medium severity issue

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

Listing 2. Excerpt from TokenLinkerLockUnlock.constructor

```
constructor(
    address gatewayAddress_,
    address gasServiceAddress_,
    address tokenAddress_
) TokenLinker(gatewayAddress_, gasServiceAddress_) {
    tokenAddress = tokenAddress_;
}
```

Description

The contract does not perform any data validation of `tokenAddress` in its constructor.

Exploit scenario

A zero-address is passed to the constructor in place of `tokenAddress`. Instead of reverting, the call succeeds.

Recommendation

Short term, add a zero-address check for `tokenAddress` in the constructor.

Long term, use Slither to detect this common issue.

Go back to Findings Summary

M2: TokenLinker has insufficient data validation

Medium severity issue

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

Listing 3. Excerpt from TokenLinker.constructor

```
constructor(address gatewayAddress_, address gasServiceAddress_) {
    gatewayAddress = gatewayAddress_;
    gasService = IAxelarGasService(gasServiceAddress_);
}
```

Description

The contract and its subclasses do not perform any data validation of `gatewayAddress_` in its constructor. The `gasService` parameter should be also more validated.

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 `gasService`). At the very least this would include a zero-address check. 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[2]. This will ensure the call reverts for most incorrectly passed values.

Go back to Findings Summary

W1: 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

W2: Floating dependency on AxelarGateway

Impact:	Warning	Likelihood:	N/A
Target:	Token Linker	Type:	Version mismatch

Description

The configuration file `package.json` is holding floating dependency on AxelarGateway (one of the main contracts of the protocol). There is a possibility of deployment with an unwished version if minor or patch updates are not properly tested.

Exploit scenario

A developer will use `npm i` instead of `npm ci` (clean install) which will overwrite the lockfile. Contracts are deployed on an untested version and due to that contracts have different behavior than it's intended.

Recommendation

Fix the version to the one that is properly tested and functional within the protocol or ensure that lockfile isn't overwritten during the deployment.

Go back to Findings Summary

W3: Mulitple ways to receive ether can lead to loss of funds

Impact:	Warning	Likelihood:	N/A
Target:	Token Linker/contracts/TokenLinkerNative.sol	Type:	Front-running

Description

The contract is using two balances. The first one is a real balance of the account, in Solidity known as `address(this).balance` and the second balance is held in a storage slot. This storage slot is updated at the end of each relevant function of the contract. However, the proxy can receive ether or any other native currency via `receive` and that can cause inconsistency in these balances between transactions if they are not atomic.

Moreover, unlike using only the `sendToken` function, there is an option to call `updateBalance` with sending ether to it and then call `sendToken` with the null amount. This introduces non-atomic behavior which can be front-runned and the potential user can lose his funds.

Exploit scenario

Bob will call `updateBalance` with sending 1 ether to it (or sending it directly to the proxy). Alice will notice that the contract has some balance in a storage slot (or some balance at all). Alice will call `sendToken` without sending any ether and it will pass (or call `updateBalance` to update the balance in a storage slot and then call `sendToken`).

Recommendation

Ensure that the users will send tokens atomically.

Go back to Findings Summary

W4: The `forecall` function is missing any checks by default

Impact:	Warning	Likelihood:	N/A
Target:	[Forecall Service](#)/contracts/executables/AxelarForecallable.sol	Type:	Data validation

Listing 4. Excerpt from [AxelarForecallable.forecall](#)

```
    function forecall(
        string calldata sourceChain,
        string calldata sourceAddress,
        bytes calldata payload,
        address forecaller
    ) external {
        _checkForecall(sourceChain, sourceAddress, payload, forecaller);
        if (getForecaller(sourceChain, sourceAddress, payload) != address(0)) revert AlreadyForecalled();
        _setForecaller(sourceChain, sourceAddress, payload, forecaller);
        _execute(sourceChain, sourceAddress, payload);
    }
```

Description

The contract is abstract and the `_checkForecall` function is left unimplemented. This is presenting a risk because there are not by default performed any checks and thus execution logic can be arbitrarily triggered by anyone.

Recommendation

Ensure that the implementation of this contract will be properly validated as it is done externally in `AxelarExecutable` in [Gateway](#).

[Go back to Findings Summary](#)

I1: Typo in the error name

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

Listing 5. Excerpt from TokenLinkerNative._takeToken

```
        if (balance + amount > address(this).balance) revert
TranferFromNativeFailed();
```

Description

The error name `TranferFromNativeFailed` should be probably `TransferFromNativeFailed`.

Recommendation

Fix the typo.

Go back to Findings Summary

Endnotes

[1] In a happy scenario, he wouldn't be able to execute payload A again, because the `forecaller` will be a non-zero-address in the contract's storage.

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

Appendix A: How to cite

Please cite this document as:

Ackee Blockchain, Axelar: Token Linker & Forecall Service, 12.8.2022.

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.

Thank You

Ackee Blockchain a.s.

Prague, Czech Republic

hello@ackeeblockchain.com

https://discord.gg/z4KDUbuPxq