ackee blockchain

Axelar

Interchain Token Linker

by Ackee Blockchain

16.2.2023

Contents

1. Document Revisions

1.0	Final report	February 16, 2023

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 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, RockawayX.

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 Woke 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. Finding classification

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.

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.

2.4. Review team

Member's Name	Position
Štěpán Šonský	Lead Auditor
Josef Gattermayer, Ph.D.	Audit Supervisor

2.5. 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

Revision 1.0

Axelar engaged Ackee Blockchain to perform a security review of the Interchain Token Linker with a total time donation of 5 engineering days in a period between February 8 and February 16, 2022 and the lead auditor was Štěpán Šonský.

The audit has been performed on the commit dd676b1.

We began our review by using static analysis tools, namely Woke. Then we took a deep dive into the codebase and set the following goals:

- check access controls,
- check correctness of LinkedTokenData operations using Woke fuzzer,
- check cross-chain data integrity (e.g. IDs, decimals…),
- detect possible reentrancies in the code,
- look for common issues such as data validation.

Our review resulted in 7 findings, ranging from Info to Critical severity. The most severe issue C1 has been found in routing logic, which is the most important part of the protocol.

Ackee Blockchain recommends Axelar:

- be careful of bugs in the routing logic
- be aware of potentially malicious token contracts
- remove commented-out code,
- remove duplicated code,

- add detailed documentation.

See Revision 1.0 for the system overview of the codebase.

4. Summary of Findings

The following table summarizes the findings we identified during our review. Unless overridden for purposes of readability, each finding contains:

- a *Description*,
- an *Exploit scenario*,
- a *Recommendation* and if applicable
- a *Solution*.

There might often be multiple ways to solve or alleviate the issue, with varying requirements regarding 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.

	Severity	Reported	Status
C1: Hardcoded Moonbeam origin	Critical	1.0	Reported
M1: Malicious token registration	Medium	1.0	Reported
W1: Duplicated code	Warning	1.0	Reported
W2: Missing contract ID validations	Warning	1.0	Reported
W3: Duplicated code	Warning	1.0	Reported
W4: Usage of `solc` optimizer	Warning	1.0	Reported
I1: Missing documentation	Info	1.0	Reported

Table 2. Table of Findings

5. Report revision 1.0

5.1. 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.

Contracts

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

InterchainTokenLinker.sol

The core contract of the protocol. Allows sending, receiving and deploying tokens. Receiving functions are protected by `onlySelf` modifier.

LinkerRouter.sol

Provides supported token address validations using the `validateSender` function.

LinkedTokenData.sol

A library for creating and reading `bytes32 tokenData`. It uses bitmasks for various flags, e.g. `IS_ORIGIN_MASK`, `IS_GATEWAY_MASK`, `IS_REMOTE_GATEWAY_MASK`.

Actors

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

Owner

The owner has total control over the supported tokens and associated validations, namely the following privileges in the contracts:

InterchainToken

- Register origin gateway token
- Register remote gateway token

LinkerRouter

- Add trusted address
- Remove trusted address
- Add gateway-supported chains
- Remove gateway-supported chains

User

The user (any EOA or contract) can interact with the protocol in following ways:

- Send token
- Send token with data
- Register origin token
- Register origin token and deploy remote tokens
- Deploy remote tokens

5.2. Trust model

The Interchain Token Linker inherits the security of Axelar GMP and adds some `onlyOwner` privileges on top of it. Users can register their own (potentially malicious) token.

C1: Hardcoded Moonbeam origin

Critical severity issue

Impact:	High	Likelihood:	High
Target:	InterchainTokenLinker.sol	Type:	Bug in logic

Description

Payload creation in function `_deployRemoteTokens` contains hardcoded `"Moonbeam"` as an origin parameter. This bug leads to the protocol malfunction.

```
bytes memory payload = abi.encodeWithSelector(
this.selfDeployToken.selector, tokenId, 'Moonbeam', name, symbol, decimals,
tokenData.isGateway());
```

Vulnerability scenario

- A user calls `deployRemoteTokens` to other chains, the hardcoded `"Moonbeam"` string is passed into the payload.
- On target chains, the tokens are deployed and their origin set to `"Moonbeam"` instead of the real origin chain name.
- Function `_sendToken` (on target chains), which's logic strongly depends on `originalChain[tokenId]` (ie. in `_callContract` and `_callContractWithToken`) behaves unexpectedly and sends tokens to a different chain.

Recommendation

Use the correct origin chain name when deploying tokens on other chains.

Go back to Findings Summary

M1: Malicious token registration

Medium severity issue

Impact:	High	Likelihood:	Low
Target:	InterchainTokenLinker.sol	Type:	Trust model

Description

Anyone can register their own ERC-20 token implementation to the Interchain Token Linker. This feature opens a large variety of potential malicious scenarios, which could affect the protocol's reputation in case it's misused.

Vulnerability scenario

We did not identify any reentrancy scenario using malicious token implementation. However, keep in mind that attackers can be very creative in token development, e.g.:

- The attacker deploys the malicious token.
- The attacker registers the token to the Interchain Token Linker and uses it to deploy to other chains.
- Users transfer the tokens to other chains.
- The attacker rug pulls tokens from the Linker.
- Users are not able to transfer tokens back to the original chain.

Recommendation

We recommend Axelar perform a code review of the tokens registered into Interchain Token Linker to avoid these scenarios.

Go back to Findings Summary

W1: Duplicated code

Impact:	Warning	Likelihood:	N/A
Target:	InterchainTokenLinker.sol	Type:	Best practices

Description

`InterchainTokenLinker` function `_giveTokenWithData` contains the same code as `_giveToken`. Code duplications are generally bad practice and could lead to errors during future development.

```
bytes32 tokenData = tokenDatas[tokenId];
address tokenAddress = tokenData.getAddress();
if (tokenData.isOrigin() || tokenData.isGateway()) {
    _transfer(tokenAddress, to, amount);
} else {
    _mint(tokenAddress, to, amount);
}
```

Recommendation

Call `_giveToken` from `_giveTokenWithData` to improve the architecture and code readability.

```
function _giveTokenWithData(
    bytes32 tokenId,
    address to,
    uint256 amount,
    string calldata sourceChain,
     bytes memory sourceAddress,
    bytes memory data
) internal {
    _giveToken(tokenId, to, amount);
    ITokenLinkerCallable(to).processToken(tokenAddress, sourceChain,
sourceAddress, amount, data);
}
```

Go back to Findings Summary

W2: Missing contract ID validations

Impact:	Warning	Likelihood:	N/A
Target:	InterchainTokenLinker.sol	Type:	Data validations

Description

In the `InterchainTokenLinker` constructor, there are only zero-address checks for `immutable gasService` and `remoteAddressValidator` state variables.

Recommendation

We recommend contract type validation using `contractId`, which is already implemented in these contracts, e.g.:

```
if (ILinkerRouter(remoteAddressValidatorAddress_).contractId() !=
0x5d9f4d5e6bb737c289f92f2a319c66ba484357595194acb7c2122e48550eda7c) revert
InvalidContractType();
```

Go back to Findings Summary

W3: Duplicated code

Impact:	Warning	Likelihood:	N/A
Target:	InterchainTokenLinker.sol	Type:	Best practices

Description

`InterchainTokenLinker` contains commented-out code, which is not a good practice.

```
//bytes32 public immutable chainNameHash;
```

```
//chainNameHash = keccak256(bytes(chainName_));
```

Recommendation

Remove unused and commented-out parts of code.

Go back to Findings Summary

W4: Usage of `solc` optimizer

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

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 documentation

Impact:	Info	Likelihood:	N/A
Target:	`**/*`	Type:	Best practices

Description

Although the code is realatively simple and easy to read, the project is missing detailed documentation.

Recommendation

We strongly recommend covering the code by NatSpec. High-quality documentation has to be an essential part of any professional project.

Go back to Findings Summary

Appendix A: How to cite

Please cite this document as:

Ackee Blockchain, Axelar: Interchain Token Linker, 16.2.2023.

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