ackee blockchain

Axelar

Ethereum contracts

by Ackee Blockchain

July 25, 2022

Contents

1. Document Revisions

0.1	Draft report	July 13, 2022
1.0	Final report	July 14, 2022
1.1	Fix review	July 25, 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

Axelar is a protocol that allows users to send tokens and interact with contracts across multiple chains.

Axelar engaged Ackee Blockchain to perform a security review of changes on several Ethereum contracts with a total time donation of 5 engineering days in a period between July 4 and June 12, 2022 and the lead auditor was Jan Kalivoda.

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

- Solidity CGP Gateway - 9f9ca0d
  - contracts/AxelarAuthMultisig.sol
- XC20 Wrapper - 327c543
  - contracts/*

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

- validating the upgradeability pattern,
- 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 9 findings, ranging from Info to High severity.

Ackee Blockchain recommends Axelar:

- create documentation including NatSpec comments,

- reconsider the current upgradeability pattern,
- write unit tests for XC20 Wrapper,
- address all other reported issues.

Update July 25, 2022: Axelar provided an updated codebase that addresses some issues from this report. See Appendix D 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.

AxelarAuthMultisig

Separated authentication mechanism from the gateway. Allows changing ownership of the auth contract and operatorship of the protocol.

XC20Wrapper

An upgradeable contract that allows XC20Wrapper owner to add tokens to be wrapped and everyone to wrap or unwrap these tokens.

4.2. Actors

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

AxelarAuthMultisig owner

The owner can transfer operatorship to an arbitrary set of addresses (excluding zero-address that is sanitized) and change the threshold. Also, his ownership can be transferred.

XC20Wrapper owner

The owner can add an arbitrary token with the matching interface and upgrade the contract.

4.3. Trust model

Users have to trust the owners of both contracts. Their actions can make the contracts completely unusable. More description is in the corresponding findings tagged as Trust model issues.

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

Severity	Impact	Likelihood
Informational	Informational	N/A
Warning	Warning	N/A
Low	Low	Low
Medium	Low	Medium
Medium	Low	High
Medium	Medium	Medium
High	Medium	High
Medium	High	Low

Severity	Impact	Likelihood
High	High	Medium
Critical	High	High

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 "Warning" or higher, based on our best estimate of whether it is currently exploitable.

Info

The issue is on the border-line 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: Ignored return values on LocalAsset interface	High	Medium	High
M1: Floating dependency on AxelarGateway	Medium	High	Low
W1: Pitfalls of upgradeability	Warning	Warning	N/A
W2: The owner can change arbitrarily operatorship and potentially cause DoS	Warning	Warning	N/A
W3: XC20Wrapper owner has escalated priviliges	Warning	Warning	N/A
W4: Missing unit tests	Warning	Warning	N/A

	Severity	Impact	Likelihood
W5: Usage of `solc` optimizer	Warning	Warning	N/A
I1: Typo in the variable name	Info	Info	N/A
I2: Missing events	Info	Info	N/A

Table 2. Table of Findings

H1: Ignored return values on LocalAsset interface

High severity issue

Impact:	Medium	Likelihood:	High
Target:	XC20 Wrapper/contracts/XC20Wrapper.sol	Type:	Ignored return values

Listing 1. Excerpt from XC20Wrapper.addWrapping

```
        LocalAsset(xc20Token).set_team(address(this), address(this),
address(this));
        LocalAsset(xc20Token).set_metadata(newName, newSymbol, IERC20
(axelarToken).decimals());
```

Listing 2. Excerpt from XC20Wrapper.wrap

```
        LocalAsset(wrappedToken).mint(msg.sender, amount);
```

Listing 3. Excerpt from XC20Wrapper.unwrap

```
        LocalAsset(wrappedToken).burn(msg.sender, amount);
```

Listing 4. Excerpt from XC20Wrapper._executeWithToken

```
        } else {
            LocalAsset(xc20).mint(receiver, amount);
        }
```

Description

The interface `LocalAsset` is returning a boolean value about success for each function. It should be checked in the contract. If it is not important, it should be documented why. The severity of this issue can vary from warning to critical, depending on the individual case.

A critical scenario would be for example that the user can call `unwrap` (see Listing 3.) without burning his/her wrapped tokens and thus repeatadly drain Axelar tokens.

Exploit scenario

A token is not minted and in a given context it does not revert but returns false and it will remain unnoticed.

Recommendation

Short term, add validation of return values to mentioned functions above.

Long term, handle all return values and use Slither to detect them.

Go back to Findings Summary

M1: Floating dependency on AxelarGateway

Medium severity issue

Impact:	High	Likelihood:	Low
Target:	XC20 Wrapper	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 go step by step with README in the repository to deploy its contracts. So, he/she 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.

Go back to Findings Summary

W1: Pitfalls of upgradeability

Impact:	Warning	Likelihood:	N/A
Target:	[XC20 Wrapper](#)/contracts/XC20Wrapper.sol	Type:	Upgradeability

Description

The setup function in the logic contract has no access controls, except the check if `implementation` is equal to zero-address. This approach **is safe** until a mistake occurs (e.g., calling `upgrade` on the logic contract), and `implementation` address will be changed to something else in the contract's storage.

Exploit scenario

A pre-condition for this attack is that `implementation` is not equal to zero-address.

An attacker starts interacting directly with the logic contract and calling `setup` function with `params`, which will define him as [XC20Wrapper owner](#).

Then he will deploy a following contract:

```
contract Exploit {
    function setup(bytes calldata params) external {
        selfdestruct(payable(address(0)));
    }
}
```

After the attacker can change the codehash variable and call `upgrade` function with `newImplementation` set to `Exploit`. This call will enter `Exploit`, but the execution will be delegated to the logic contract. The logic contract will be destroyed.

Since the proxy has only an empty `setup` function and `fallback`, it will cause it can’t be further upgraded.

As a result, the protocol is frozen in its actual state.

Recommendation

First of all, the proxy should have an existence check (`isContract` function or `extcodesize` instruction can be used).

Solidity docs	The low-level call, `delegatecall` and `callcode` will return success if the called account is non-existent, as part of the design of EVM. Existence must be checked prior to calling if desired.

Secondly and more importantly, we must be sure that the `setup` function can be called only once. It can be implemented in various ways. We recommend using initializer modifier or another alternative like `onlyInitialized`, which is described in Appendix C: Theory of Upgradeability.

Go back to Findings Summary

W2: The owner can change arbitrarily operatorship and potentially cause DoS

Impact:	Warning	Likelihood:	N/A
Target:	Solidity CGP Gateway/contracts/AxelarAuthMultisig.sol	Type:	Trust model

Description

The owner can transfer operatorship to an arbitrary set of addresses (excluding zero-address that is sanitized) and change the threshold. Unwise or malicious choices can block operations on the protocol or affect it dangerously in various ways (e.g. not enough relevant addresses above the threshold, intentional change to singlesig, …).

Recommendation

Multisignature address for the owner could partly mitigate the risk, but generally is a good practice that the owner has not so escalated privileges.

Go back to Findings Summary

W3: XC20Wrapper owner has escalated priviliges

Impact:	Warning	Likelihood:	N/A
Target:	XC20 Wrapper/contracts/XC20Wrapper.sol	Type:	Trust model

Description

The owner can add an arbitrary token with the matching interface. This token can have potentially malicious behavior, through calls in `_safeTransfer` and `_safeTranferFrom` functions. Also, he can do any call (to a given address) from the contract's context using an `invoke` function.

At last, he can change the codehash variable and seriously affect the protocol as it is described in W1 finding.

Recommendation

Multisignature address for the owner could partly mitigate the risk, but generally is a good practice that the owner has not so escalated privileges.

Go back to Findings Summary

W4: Missing unit tests

Impact:	Warning	Likelihood:	N/A
Target:	XC20 Wrapper	Type:	Quality Assurance

Description

The project is completely missing the unit tests. Unit tests should be an essential part of every development because they can discover many hidden issues in the system and avoid human errors during routine refactoring.

Exploit scenario

Without unit tests, any code changes can lead to unexpected system behavior or new security issues.

Recommendation

We recommend implementing unit tests and achieving the best possible coverage.

Go back to Findings Summary

W5: 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: Typo in the variable name

Impact:	Info	Likelihood:	N/A
Target:	XC20 Wrapper/contracts/Proxy.sol	Type:	Typo

Listing 5. Excerpt from Proxy.fallback

```
    fallback() external payable {
        address implementaion_ = implementation();
```

Description

The variable `implementaion_` should be `implementation_`.

Recommendation

Fix the typo.

Go back to Findings Summary

I2: Missing events

Impact:	Info	Likelihood:	N/A
Target:	XC20 Wrapper	Type:	Logging

Description

Many places in the codebase lack logging (like `addWrapping`, `removeWrapping`, …). Lack of logging can make it difficult to observe and debug the contract and make incident analysis difficult.

Recommendation

Add event emissions to all important state-changing functions.

Long-term, log all important operations. This will ensure that all the protocol users can be effectively informed about everything happening in the system.

Go back to Findings Summary

Appendix A: How to cite

Please cite this document as:

Ackee Blockchain, Axelar: Ethereum contracts, July 25, 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: Theory of Upgradeability

This appendix lays out the most common pitfalls of upgradeability in Solidity and Ethereum generally, as well as our general recommendations. This is primarily for education purposes; for recommendations about the reviewed project, please refer to the **Recommendation** section of each finding.

Upgradeability is a difficult topic, and clsoe to impossible to get right. Most commonly, the following are the greatest issues in any upgradeability mechanism:

1. Interacting with the logic contract. If the syntactic contract or any of its superclasses contain a delegatecall or selfdestruct instruction, it could be possible to destroy the logic contract. See the [Parity 2](#) vulnerability.
2. Front-running the initialization function (syntactically in the logic contract) on the proxy. This is $usually$ a non-issue and easy to solve, just check that the init function has not been called in the smart contract, and finally require the init function have succeeeded in the deployment script (e.g. JavaScript or Python).
3. Front-running other functions (syntactically in the logic contract) on the proxy. This can be a problem if the deployment of the proxy and the initialization of the logic contract are not atomic. In that case, it may be possible for an attacker to front-run the initialization, and call some function. If there are no access controls in this case, the call would succeed without the deployment script failing, and the attacker might have a backdoor in to the contract without anyone realizing it.

There tend to be two possible solutions for (1) and (3):

C.1. Manual review

For (1), this involves checking that it, or any of its ancestors, don't contain the delegatecall or selfdestruct instructions.

For (3), this involves checking that if any mutating function is called before the init function has been called, the call would not change any state in the system, e.g. by reverting.

C.2. Programmatic approach

The best way to accomplish both (1) and (3) (while preserving (2)) is to:

1. Ensure that no function on the deployed logic contract can be called until the initialization function is called.
2. Make sure that once the logic contract is constructed, its initialization function cannot be called.
3. Ensure that the initialization function can be called on the oroxy.
4. All functions can be called on the proxy once it has been initialized.

As an example, here is a way to accomplish that on a primitive upgradeable contract:

1. Ensure the initialization function can only be called once, e.g. by using OpenZeppelin's `initializer` modifier (see [Listing 7](#)).
2. Also add the `initializer` modifier to the constructor of the logic contract (see [Listing 8](#)).
   - The constructor is called on the deployed logic contract, but not on the proxy.
3. Add a `initialized` state variable ([Listing 6](#)) that gets set to `true` on initialization (see [Listing 7](#)). Note that we have to define a new variable,

since OpenZeppelin's `_initialized` is marked as `private`.

4. Add a require to every mutating function in the logic contract that it has been initialized (see [Listing 9](#)).

Listing 6. For syntactic logic contract

```
bool public initialized;
```

Listing 7. For syntactic logic contract

```
function initialize() public initializer {
    initialized = true;
}
```

Listing 8. For syntactic logic contract

```
constructor() initializer {}
```

Listing 9. For every mutating function in the syntactic logic contract

```
modifier onlyInitialized() {
    require(initialized);
    _;
}
```

In summary, the process would be to:

1. Add a requirement to every mutating function that the contract has been initialized.
2. Add a requirement to the initialization function that it cannot be called on the logic contract.

Together, these will accomplish both (1) and (3) of the upgradeability requirements.

7. Appendix D: Fix Review

On July 15, 2022, Ackee Blockchain reviewed Axelar's fixes for the issues identified in this report. The following table summarizes the fix review. The updated commit for XC20 Wrapper was 4340a2f and after reporting an incorrect fix in H1F, it was changed to dd49548. In Solidity CGP Gateway were not performed any changes.

Fix log

Id	Severity	Impact	Likelihood	Status
H1: Ignored return values on LocalAsset interface	High	Medium	High	**Fixed**
M1: Floating dependency on AxelarGateway	Medium	High	Low	**Fixed**
W1: Pitfalls of upgradeability	Warning	Warning	N/A	**Acknowledged**
W2: The owner can change arbitrarily operatorship and potentially cause DoS	Warning	Warning	N/A	**Acknowledged**
W3: XC20Wrapper owner has escalated priviliges	Warning	Warning	N/A	**Acknowledged**
W4: Missing unit tests	Warning	Warning	N/A	**Fixed**
W5: Usage of `solc` optimizer	Warning	Warning	N/A	**Reported**

Id	Severity	Impact	Likelihood	Status
I1: Typo in the variable name	Info	Info	N/A	**Reported**
I2: Missing events	Info	Info	N/A	**Reported**

Table 3. Table of fixes

H1F: Ignored return values on LocalAsset interface

High severity issue

Impact:	Medium	Likelihood:	High
Target:	[H1: Ignored return values on LocalAsset interface](#)	Type:	Ignored return values

Description

The return values are now properly checked. In case of returning `false`, the transaction is reverted.

The only function which works differently is the `_executeWithToken` function.

Listing 10. The previous behavior

```
if (xc20 == address(0)) {
    _safeTransfer(tokenAddress, receiver, amount);
    } else {
        LocalAsset(xc20).mint(receiver, amount);
    }
}
```

Listing 11. The behavior in [4340a2f](#)

```
if (xc20 != address(0) || !LocalAsset(xc20).mint(receiver, amount)) {
    _safeTransfer(tokenAddress, receiver, amount);
}
```

The first fix ([4340a2f](#)) caused the transfers are happening if `xc20` is not equal to zero-address. However, from the previous behavior, it should be only if `xc20` **is** zero-address. This issue was reported to the client and fixed in the second commit ([dd49548](#)).

Listing 12. The fix in dd49548

```
if (xc20 == address(0) || !LocalAsset(xc20).mint(receiver, amount)) {
    _safeTransfer(tokenAddress, receiver, amount);
}
```

Also, transfers on failed mint are desirable.

Go back to Fix log

M1F: Floating dependency on AxelarGateway

Medium severity issue

Impact:	High	Likelihood:	Low
Target:	M1: Floating dependency on AxelarGateway	Type:	Version mismatch

Description

The floating version is removed from the configuration file. It is fixed by referencing a different repository (axelar-utils-solidity) that will only update its `IAxelarGateway`` if the deployed gateway does.

Go back to Fix log

W1F: Pitfalls of upgradeability

Impact:	Warning	Likelihood:	N/A
Target:	W2: The owner can change arbitrarily operatorship and potentially cause DoS	Type:	Upgradeability

Description

This issue was acknowledged by the client.

Client's response

"There is no way for implementation() to not be 0 in the implementation contract. Setup will be called zero times there and so ensuring that it can only be called once does not solve the exploit described."

Go back to Fix log

W2F: The owner can change arbitrarily operatorship and potentially cause DoS

Impact:	Warning	Likelihood:	N/A
Target:	[W2: The owner can change arbitrarily operatorship and potentially cause DoS](#)	Type:	Trust model

Description

This issue was acknowledged by the client.

Client's response

"We plan to use a multisig wallet for this, it has to be managed by someone to add tokens. We might remove ability to remove wrappings and remove upgradability in the future which will make the owner much less “powerful”

[Go back to Fix log](#)

W3F: XC20Wrapper owner has escalated priviliges

Impact:	Warning	Likelihood:	N/A
Target:	[W3: XC20Wrapper owner has escalated priviliges](#)	Type:	Trust model

Description

This issue was acknowledged by the client.

Client's response

"Invoke has been removed. We might hardcode the code hash variable which would restrict the owner to only tokens a) deployed by Axelar and b) xc20s deployed by moonbeam, none of which should be malicious. We might upgrade to this once Moonbeam has a finalised version of xc20s."

[Go back to Fix log](#)

W4F: Missing unit tests

Impact:	Warning	Likelihood:	N/A
Target:	W4: Missing unit tests	Type:	Trust model

Description

Tests were added, but they could have better coverage. They weren't even able to discover the issue mentioned in H1F in the first commit (4340a2f).

Go back to Fix log

Thank You

Ackee Blockchain a.s.

Prague, Czech Republic

hello@ackeeblockchain.com

https://discord.gg/z4KDUbuPxq