ackee blockchain

Axelar

Crosschain DEX

3 June 2022

by Ackee Blockchain

Contents

1. Document Revisions

1.0	Final report	Jun 03, 2022
1.1	Fix review	Jun 07, 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
Štěpán Šonský	Lead Auditor
Jan Šmolík	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 engaged Ackee Blockchain to conduct a security review of the Crosschain-Dex (private repository) with a total time donation of four engineering days. The review took place between May 31 and June 3, 2022.

We started with the commit `5739e73bcfa469c2822c59b76d73ffb1cbf213c5` and during the audit the code has been slightly changed to improve readability, so the final commit was `faedfd700ccc0c004cd204059c68d88e109cf4ee`.

The code quality is good, smart contracts are easy to read. Understanding the whole system requires basic knowledge of Axelar protocol and Uniswap V2 router. One of the biggest concerns we have identified is missing unit tests, which should be essential for every development.

Regarding the findings, we have found 10 issues ranked from informational to high.

Our conclusions regarding the Crosschain-Dex project:

- missing unit tests,
- potentially dangerous low-level calls and assembly code,
- code quality is good, but duplicities should be removed,
- the provided documentation is sufficient for audit,
- code documentation is missing.

We recommend fixing or clarifying the issues and performing a fix review.

Update June 7, 2022: Axelar provided an updated codebase that addresses some issues from this report. See Appendix B 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. Overview

Axelar Crosschain-Dex allows users to send any token from the source chain and get any token at the destination chain in a single transaction.

- The token sent/received must be either a cross-chain token (a token that can be transferred cross-chain by Axelar) or there must be a route and liquidity to swap to/from the cross-chain token.
- The cross-chain token transfer from one chain to another (and a potential swap of the cross-chain token on the destination chain) is handled off-chain by Axelar protocol.

4.2. Contracts

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

SquidSwapExecutable.sol

The `SquidSwapExecutable` is the main component we reviewed. It implements the `IAxelarExecutable` interface.

`SquidSwapExecutable` can

- perform the swap to the cross-chain token on the source chain,
- communicate with the `AxelarGateway` to transfer the cross-chain token and send the cross-chain message to swap on the destination chain,

- receive the cross-chain token on the destination chain, and
- receive the potential cross-chain message to swap on the destination chain.

The following functions can be called externally on a source chain:

sendTrade()

This function sends a cross-chain token to the gateway contract on the source chain. After receiving the cross-chain token on the destination chain, it is swapped to the desired token there.

tradeSend()

This function swaps a token into a cross-chain token on the source chain and sends it to the gateway contract. On the destination chain, the cross-chain token will go to the given recipient's address directly.

tradeSendTrade()

This function swaps a token into a cross-chain token on the source chain and sends it to the gateway contract, and after receiving the cross-chain token on the destination chain, it is swapped to the desired token there.

DistributionENSExecutable.sol

`DistributionENSExecutable` is a contract for sending and receiving tokens cross-chain to many receivers using the ENS.

SquidToken.sol

The `SquidToken` is an ERC-20 token.

We suppose it is used as a test token because there is a public `mint()` function.

4.3. Actors

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

Owner

Owner of the `SquidSwapExecutable` contract can:

- transfer or renounce the ownership of the contract,
- add `siblings` in the `SquidSwapExecutable` contract.

User

User role means any external address. It has the following permissions in audited contracts.

`DistributionENSExecutable`

- `sendToMany()`
- `executeLocal()`

`SquidSwapExecutable`

- `sendTrade()`
- `tradeSend()`
- `tradeSendTrade()`

`SquidToken`

- mint tokens for himself,
- perform all standard ERC20 operations.

Axelar

Axelar protocol is responsible for transferring cross-chain messages. In the current scope, it is considered as a black box. Axelar can call receiving

functions on the destination chain.

4.4. Trust model

Users of the Crosschain-Dex need to trust Axelar to handle cross-chain transactions off-chain correctly.

5. Vulnerabilities risk methodology

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

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 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 *Informational*.

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:

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.

Informational

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

	Type	Impact	Likelihood
H1: Unhandled return value	Data validation	High	High
L1: Payload manipulation	Data validation	Low	Low
L2: Unchecked transfer	Data validation	Low	Low
W1: Code duplication	Code quality	Warning	N/A
W2: Renounce ownership	Access controls	Warning	N/A
W3: Missing unit tests	Quality Assurace	Warning	N/A
W4: External mint function	Access controls	Warning	N/A
I1: Commented out code	Code quality	Informational	N/A
I2: State variable access	Code quality	Informational	N/A

	Type	Impact	Likelihood
I3: Missing code documentation	Documentation	Informational	N/A

Table 1. Table of Findings

H1: Unhandled return value

Impact:	High	Likelihood:	High
Target:	SquidSwapExecutable.sol	Type:	Data validation

Description

In the `tradeSend()` function there is unchecked return value of the `_tradeSrc()`. The function `_tradeSrc()` returns the result of the low-level call to Uniswap V2 router, which could fail and return `false` instead of `revert`.

Exploit scenario

The user calls the `tradeSend()` function. In the `_tradeSrc()` function, tokens or the native asset gets transferred from the user into the contract. When the Uniswap V2 router call fails, and the `tradeSend()` successfully executes the rest of the code, the user's assets stay in the contract.

Recommendation

Add the return value check for the _tradeSrc() call inside the `tradeSend()` function.

```
require(_tradeSrc(tradeData), "TRADE_FAILED");
```

or

```
if (!_tradeSrc(tradeData)) revert("TRADE_FAILED");
```

Go back to Findings Summary

L1: Payload manipulation

Impact:	Low	Likelihood:	Low
Target:	SquidSwapExecutable.sol	Type:	Data validation

Description

`tradeData` in the `_receiveTrade()` inside the `assembly` code on the destination chain can be manipulated by the user from the source chain using `inputPos` and `amount` values.

Exploit scenario

The attacker could design specific parameters to inject malicious data inside the tradeData and manipulate the trade. However, according to the client's response, this vulnerability can cause damage only to the attacker. So we have decreased the severity from high to low.

Recommendation

We recommend avoiding the ability to manipulate input parameters of the assembly code from external addresses. This kind of issue often opens many backdoors for the attacker.

Client's response

"We considered this but since the smart contract does not hold any value, and merely manipulates tokens received as part of the `callContractWithToken`, a malicious user could at worst lose their own tokens."

Go back to Findings Summary

L2: Unchecked transfer

Impact:	Low	Likelihood:	Low
Target:	SquidSwapExecutable.sol	Type:	Data validation

Description

The contract uses unchecked `ERC20.transferFrom()`. It can be a problem in the case of non-standard ERC-20 tokens, which do not return the `bool` value.

The return value is not a part of a function selector. So `transfer()` without a return value and a function `transfer() returns(bool)` have the same sighash, but they are different.

Exploit scenario

There are many different scenarios where contracts interacting with these non-standard ERC-20 tokens can lead to fatal consequences when using unchecked transfers. But in this case, we have not identified it as a high severity threat.

Recommendation

Use OpenZeppelin `SafeERC20` extension and `safeTransfer()`, `safeTransferFrom()` functions to handle non-standard ERC-20 tokens.

Go back to Findings Summary

W1: Code duplication

Impact:	Warning	Likelihood:	N/A
Target:	SquidSwapExecutable.sol	Type:	Code quality

Description

In the SquidSwapExecutable.sol there is a block of code that is used in both `sendTrade()` and `tradeSendTrade()`.

Duplicated code decreases the readability of the code and increases the space for human errors.

```
bytes memory payload = abi.encode(
    tradeData,
    traceId,
    fallbackRecipient,
    inputPos
);
gasReceiver.payNativeGasForContractCallWithToken{value: msg.value}(
    address(this),
    destinationChain,
    siblings[destinationChain],
    payload,
    symbol,
    amount,
    msg.sender
);
IERC20(token).approve(address(gateway), amount);
gateway.callContractWithToken(
    destinationChain,
    siblings[destinationChain],
    payload,
    symbol,
    amount
);
```

Recommendation

Move this block of code into a new private function.

Go back to Findings Summary

W2: Renounce ownership

Impact:	Warning	Likelihood:	N/A
Target:	SquidSwapExecutable.sol	Type:	Access controls

Description

The SquidSwapExecutable.sol is an `Ownable` contract. The owner of the contract can add `siblings` via `addSibling()`. The ownership can be transferred and renounced by the owner.

Exploit scenario

The owner accidentally calls `renounceOwnership()`. Then nobody will ever be able to add `siblings`.

Recommendation

We recommend overriding the `renounceOwnership()` method to disable this feature if it is not intended. Otherwise, ignore this issue.

Go back to Findings Summary

W3: Missing unit tests

Impact:	Warning	Likelihood:	N/A
Target:	*/*	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 to implement unit tests and achieve 90% coverage at minimum.

Go back to Findings Summary

W4: External mint function

Impact:	Warning	Likelihood:	N/A
Target:	*/*	Type:	Access Controls

Description

The SquidToken.sol contract contains `external` `mint()` function with the comment:

```
// Anyone can mint token for test.
```

Exploit scenario

It is not clear if this is an intended feature for public release, so `SquidToken` is considered as a public testing token without any value, or if this function is planned to be removed before deployment to mainnets.

In the first case, no action is needed. In the second case, this would be a critical issue.

Recommendation

If some of the contract logic is meant to be only for the testing phase and is planned to be removed before the deployment, it is a good practice to create another inherited contract with the prefix `Test` or `Mock` to ensure that forgotten testing code does not go into the production.

Go back to Findings Summary

I1: Commented out code

Impact:	Informational	Likelihood:	N/A
Target:	DistributionENSExecutable.sol	Type:	Code quality

Description

In DistributionENSExecutable, there is a commented out code:

```
// doesn't seem to work
// int256 a =
~0x0000000000000000000000000fffffffffffffffffffffffffffffffffffffff;
// if (uint256(recipients[i]) & uint256(a) == 0) {
//     ensResolvedAddress = address(uint160(recipients[i]));
// } else {
//     IENS ens = IENS(ensRegistryAddress);
//     IENSResolver resolver = ens.resolver(recipients[i]);
//     ensResolvedAddress = resolver.addr(recipients[i]);
// }
```

Recommendation

We understand that the codebase is currently under development, but we recommend following the developer's best practices and continuously deleting the unused code.

Go back to Findings Summary

I2: State variable access

Impact:	Informational	Likelihood:	N/A
Target:	SquidSwapExecutable.sol	Type:	Code quality

Description

In DistributionENSExecutable.sol, the `gasReceiver` variable is set only in the constructor, so it can be marked as `immutable`.

```
IAxelarGasReceiver gasReceiver;
```

Also, consider the visibility of the gasReceiver` variable in both contracts DistributionENSExecutable.sol and SquidSwapExecutable.sol.

Recommendation

Define state variables as immutable if they are set only in the constructor.

If nobody needs to access these state variables from outside, change the visibility to `private`.

Go back to Findings Summary

I3: Missing code documentation

Impact:	Informational	Likelihood:	N/A
Target:	*/*	Type:	Documentation

Description

The code is missing detailed documentation.

Recommendation

Although the client provided brief project documentation and the code is relatively simple and understandable, we recommend using NatSpec documentation. High-quality documentation has to be an essential part of any professional project.

Go back to Findings Summary

7. Appendix A

7.1. How to cite

Please cite this document as:

Ackee Blockchain, Axelar Crosschain-Dex, June 3, 2022.

If an individual issue is referenced, please use the following identifier:

`ABCH-{project_identifer}-{finding_id}`,

where `{project_identifier}` for this project is `AXELAR-CROSSCHAIN-DEX` and `{finding_id}` is the id which can be found in Summary of Findings. For example, to cite H1 issue, we would use `ABCH-AXELAR-CROSSCHAIN-DEX-H1`.

8. Appendix B: Fix Review

On June 7, 2022, Ackee Blockchain reviewed Axelar's fixes for the issues identified in this report. The following table summarizes the fix review.

Fix log

Id		Type	Impact	Likelihood	Status
H1F	H1F: Unhandled return value	Data validation	High	High	**Fixed**
L1F	L1: Payload manipulation	Data validation	Low	Low	**Acknowledged**
L2F	L2F: Unchecked transfer	Data validation	Low	Low	**Fixed**
W1F	W1F: Code duplication	Code quality	Warning	N/A	**Fixed**
W2F	W2F: Renounce ownership	Access controls	Warning	N/A	**Fixed**
W3F	W3: Missing unit tests	Quality assurance	Warning	N/A	**Acknowledged**
W4F	W4: External mint function	Access controls	Warning	N/A	**Acknowledged**
I1F	I1: Commented out code	Code quality	Info	N/A	**Acknowledged**
I2F	I2: State variable access	Code quality	Info	N/A	**Acknowledged**
I3F	I3: Missing code documentation	Documentation	Info	N/A	**Acknowledged**

Table 2. Table of fixes

H1F: Unhandled return value

Impact:	High	Likelihood:	High
Target:	H1: Unhandled return value	Type:	Data validation

Description

The return value of the low-level call is now being checked in the `tradeSend()` function.

```
require(_tradeSrc(tradeData), "TRADE_FAILED");
```

Therefore, if the low-level router call in `_tradeSrc()` fails, the whole `tradeSend()` transaction will safely revert.

Go back to Fix log

L2F: Unchecked transfer

Impact:	Low	Likelihood:	Low
Target:	L2: Unchecked transfer	Type:	Data validation

Description

In the SquidSwapExecutable contract, the OpenZeppelin `SafeERC20` library is now used for all `ERC20` transfers, as we recommended.

Go back to Fix log

W1F: Code duplication

Impact:	Warning	Likelihood:	N/A
Target:	W1: Code duplication	Type:	Code quality

Description

The duplicated block of code used in `sendTrade()` and `tradeSendTrade()` was moved into a new private function `_sendTrade()`:

```
function _sendTrade(
        string memory destinationChain,
        string memory symbol,
        uint256 amount,
        bytes memory tradeData,
        bytes32 traceId,
        address fallbackRecipient,
        uint256 inputPos
    ) private {
        ...
    }
```

Go back to Fix log

W2F: Renounce ownership

Impact:	Warning	Likelihood:	N/A
Target:	W2: Renounce ownership	Type:	Access controls

Description

The `renounceOwnership()` function has been overridden:

```
function renounceOwnership() public override {}
```

It is no longer possible to call the function accidentally and loose the ownership of the contract.

Go back to Fix log

ackee

Thank You

Ackee Blockchain a.s.

Prague, Czech Republic

hello@ackeeblockchain.com

https://discord.gg/z4KDUbuPxq