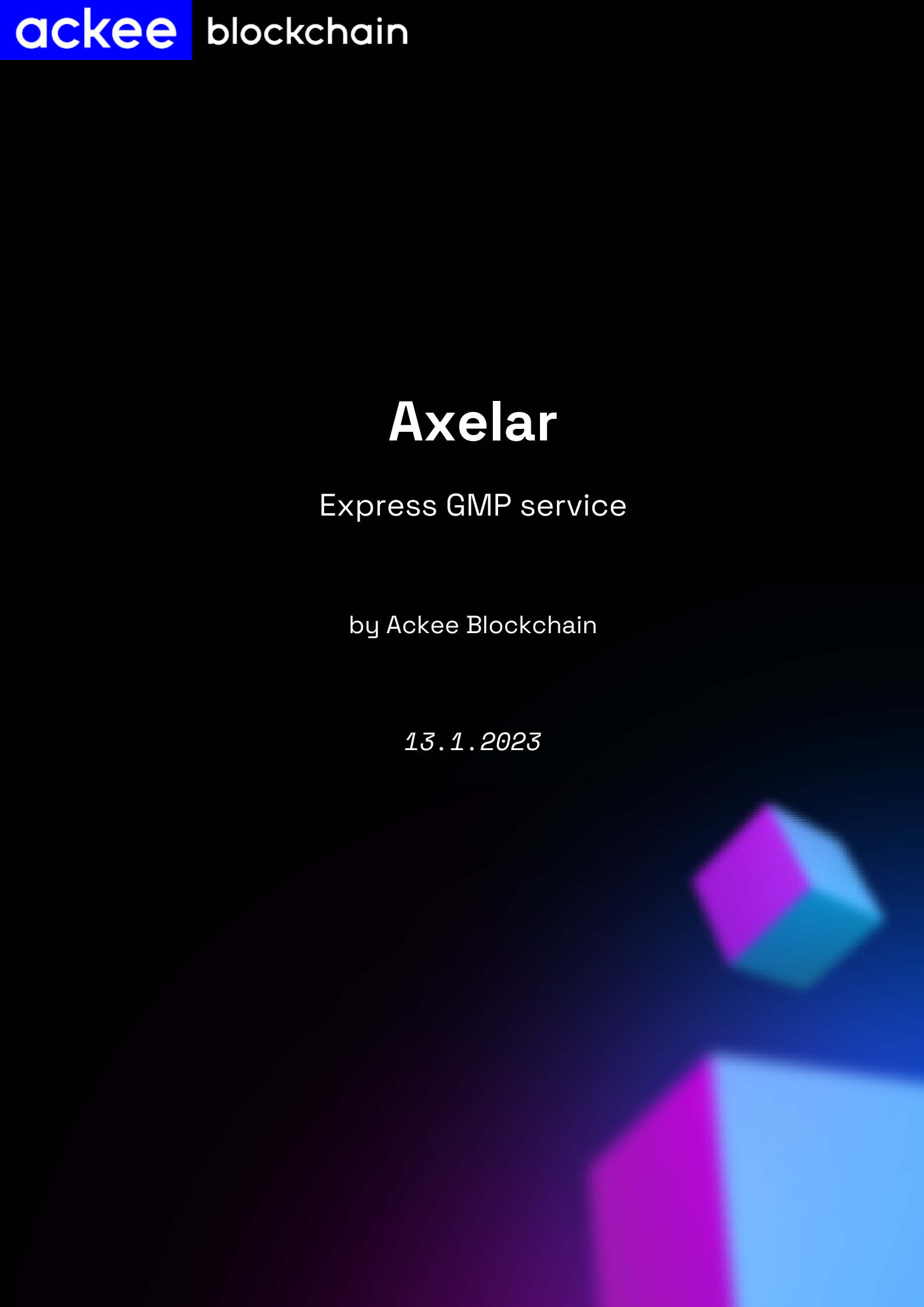
ackee blockchain

# Axelar

Express GMP service

by Ackee Blockchain

*13.1.2023*



# Contents

# 1. Document Revisions

| | | |
|---|---|---|
| 1.0 | Final report | January 13, 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 |
|---|---|
| Lukáš Böhm | Lead Auditor |
| Štěpán Šonský | 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

Axelar is a cross-chain communication protocol. It allows token transfers and cross-chain contract calls. The new feature of the protocol allows express message delivery for privileged users.

## Revision 1.0

Axelar engaged Ackee Blockchain to perform a security review of the new GMP express module with a total time donation of 5 engineering days in a period between January 4 and June 13, 2023 and the lead auditor was Lukáš Böhm.

The audit has been performed on the following two pull requests:

- Executable contracts, `46298f2`
- GMP express service, `cfd57e3`

The first introduces Executable contracts with its proxy and registry. The second pull request implements an express version of the General Message Protocol (GMP).

The code review process originally started on older commits. However, the codebase was updated during the review process. The final commits that were reviewed are mentioned above.

We began our review using static analysis tools, namely Slither and Woke. We then took a deep dive into the logic of the contracts. During the review, we paid particular attention to:

- ensuring the express logic does not allow to double-spend sent tokens,
- lost of express sender funds in more edge case scenarios,

- detecting possible reentrancies in the code,
- the correctness of access controls,
- proper data validation,
- message ID handling.

Our review resulted in 7 findings, ranging from Info to Critical severity.

Ackee Blockchain recommends Axelar:

- pay more attention to function visibility,
- address all other reported issues.

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: Arbitrary deploy salt | Critical | 1.0 | Reported |
| M1: GMPExpressService constructor data validation | Medium | 1.0 | Reported |
| W1: Usage of `solc` optimizer | Warning | 1.0 | Reported |
| W2: Lack of events | Warning | 1.0 | Reported |
| I1: Lack of in-code documentation | Info | 1.0 | Reported |
| I2: Make reused code into a library | Info | 1.0 | Reported |
| I3: Validation of token address | 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.

#### GMPExpressService

The contract inherits from `AxelarExecutable` and extends it with the functionality `callWithToken`, which implements the logic for express message handling. The tokens are lent to the destination contract and are returned after the classical GMP call is finalized. The contract allows programmatic deployment of ExpressExecutableProxy and its implementation.

The modifier `onlyOperator` is used to restrict access to the address of Axelar service operator. It is a contract designed to be used only by trusted partners and maintained by Axelar's microservice.

#### ExpressExecutableProxy

The contract provides functionalities to execute GMP express messages in the final destination. Functionalities for properly handling messages are extended to the registry contract.

#### ExpresRegistry

The contract is used for registering and processing calls with tokens. The functions are callable only by a proxy address.

### Actors

#### Owner

The owner address owns the GMPExpressService contract. The owner has permission to upgrade the logic contract. The ownership can also be transferred to another address.

#### Express operator

The operator is a trusted party responsible for triggering the express messages in the GMPExpressService contract and withdrawing the tokens from the contracts.

## 5.2. Trust model

The current trust model for the GMPExpressService has issues from the service-user (service has trust assumptions about the user) side. The destination contract is expected to return the tokens, but this is not enforced programmatically and is based on pure trust.

From the user-service (user has trust assumptions about the service) side, the service can provide arbitrary data in the GMP express calls, and the user applications have to trust that the data is valid. This could become problematic if the service is compromised.

# C1: Arbitrary deploy salt

*Critical severity issue*

| Impact: | High | Likelihood: | High |
|---|---|---|---|
| Target: | GMPExpressService.sol | Type: | Function visibility |

## Description

The main entry points for creating a new contract are functions `deployExpressProxy` and `deployExpressExecutable`. In these functions, the variable `deploySalt` is calculated based on the message sender address and then passed to the function `_deployExpressProxy`.

```
bytes32 deploySalt = keccak256(abi.encode(msg.sender, salt));
return _deployExpressProxy(deploySalt, implementationAddress, owner,
setupParam
```

The problem is in the function `_deployExpressProxy` visibility, which is set to `public`. This means that anyone can call this function and pass an arbitrary `deploySalt` value in it.

## Vulnerability scenario

In combination with a front-running a malicious actor can steal the address of the newly created contract. This can be done by calling the function `_deployExpressProxy` with the same `deploySalt` value as the one calculated for the victim in the `deployExpressProxy` function. This will cause the contract to be deployed at the same address as the one that was intended to deploy by the victim.

In a more creative scenario, the attacker can deploy the contract with the

victim's ownership but with harmful setup parameters. This can trick the victim that deployment was successful and the contract is ready to use, even though the victim's transaction fail because it was front-runned.

### Recommendation

Change the visibility of the function `_deployExpressProxy` to `internal`.

Go back to Findings Summary

# M1: GMPExpressService constructor data validation

*Medium severity issue*

| Impact: | High | Likelihood: | Low |
|---|---|---|---|
| Target: | GMPExpressService.sol | Type: | Data validation |

## Description

GMPExpressService lacks data validation in its constructor.

```
constructor(
    address gateway_,
    address gasService_,
    address expressOperator_,
    address serviceProxy_,          // <<<<<
    string memory currentChain
) AxelarExecutable(gateway_) {
    if (gasService_ == address(0)) revert InvalidAddress();
    if (expressOperator_ == address(0)) revert InvalidOperator();

    gasService = IAxelarGasService(gasService_);
    expressOperator = expressOperator_;
    serviceProxy = serviceProxy_;   // <<<<<
    expressProxyCodeHash = address(new ExpressExecutableProxy(serviceProxy,
gateway_)).codehash;
    currentChainHash = keccak256(bytes(currentChain));
}
```

During the creation of the ExpressExecutableProxy in the constructor, the wrong `serviceProxy_` address will not cause any harm because the variable `gateway_` cannot be zero. Thus the `serviceProxy_` will not be used inside the ExpressExecutableProxy constructor. However, the variable `serviceProxy` is later used in `_deployExpressProxy`.

```
function _deployExpressProxy(
    bytes32 deploySalt,
    address implementationAddress,
    address owner,
    bytes memory setupParams
) public returns (address) {
    // Passing address(0) for automatic gateway lookup. Allows to have the
same proxy address across chains
    ExpressExecutableProxy proxy = new ExpressExecutableProxy{ salt:
deploySalt }(serviceProxy, address(0)); // <<<<<

    proxy.init(implementationAddress, owner, setupParams);

    return address(proxy);
}
```

Because `address(0)` is passed as the second argument into the ExpressExecutableProxy constructor, the `serviceProxy` variable is used under the name `gmpExpressService_` to resolve the gateway address.

```
if (gateway_ == address(0)) {
    resolvedGateway = IGMPExpressService(gmpExpressService_).gateway(); //
<<<<<
} else {
    resolvedGateway = IAxelarGateway(gateway_);
}
```

### Exploit scenario

An incorrect value of `serviceProxy_` is passed to the constructor. Instead of reverting, the call succeeds. If such a mistake is not discovered quickly and the contracts are not redeployed, the protocol will not be able to provide any of the `deploy` functionalities.

### Recommendation

Because the ExpressExecutableProxy contract does not exist when calling GMPExpressService constructor, the validation with contract ID does not seem to be an option. However, the ExpressExecutableProxy constructor can call `gmpExpressService_` to get the ID and ensure the correctness of the address.

Long term, Axelar could deploy a library or an additional service that would validate whether the provided address corresponds to a valid Axelar component. Validating an EOA is harder. At least this might include a zero address check.

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: Lack of events

| Impact: | Warning | Likelihood: | N/A |
|---|---|---|---|
| Target: | GMPExpressService.sol | Type: | Logging |

### Description

The GMPExpressService.sol contract doesn't emit events on any of the important actions. This makes it hard to monitor that the tokens are borrowed and returned correctly.

### Exploit scenario

The destination `forecallable` contract doesn't return borrowed tokens or any other unexpected event that occurs. Due to a lack of logging, it takes a longer time to detect the root of this issue.

### Recommendation

Add events to the important functions that relate to borrowing and returning tokens and contract deployment. Logging will help to monitor anomalies and detect issues faster.

Go back to Findings Summary

# I1: Lack of in-code documentation

| Impact: | Info | Likelihood: | N/A |
|---|---|---|---|
| Target: | `**/*` | Type: | Function visibility |

### Description

The system is currently lacking high-level in-code documentation. While some contracts, such as the ExpressExecutableProxy contain some lines of descriptive text, non of the project contracts are fully NatSpec documented This hinders readability and makes onboarding onto the system more difficult.

### Recommendation

Add contract-level and function-level NatSpec to all contracts and functions to increase readability and maintainability. Ideally document the codebase while writing it. This will ensure maximum transparency and ease of use for developers, users, and auditors.

Go back to Findings Summary

# I2: Make reused code into a library

| Impact: | Info | Likelihood: | N/A |
|---|---|---|---|
| Target: | GMPExpressService.sol | Type: | Code copying |

### Description

GMPExpressService and other Axelar components contain the same functions `_safeTransfer` and `_safeTransferFrom`. Those functions are copied each time they are needed.

### Recommendation

To avoid code duplication, the functions should be moved to a library. This will have the following benefits:

- the code will be easier to maintain, as it will be in a single place,
- the code will be easier to audit, as it will be in a single place,
- no copy-paste error will be possible,
- the code will be more readable and elegant.

Go back to Findings Summary

# I3: Validation of token address

| Impact: | Info | Likelihood: | N/A |
|---|---|---|---|
| Target: | ExpressExecutableProxy.sol | Type: | Data validation |

### Description

The ExpressExecutableProxy supports the `expressExecuteWithToken` method which allows express message execution with the supplied token. The service fetches the token address from the gateway and calls `_safeTransferFrom`.

```
  _safeTransferFrom(token, msg.sender, amount);
  _executeWithToken(sourceChain, sourceAddress, payload, tokenSymbol,
amount);
```

If the given token is not supported by the gateway, `address(0)` will be returned and passed to `_safeTransferFrom`. There the function will revert with an ambiguous error message.

### Recommendation

Check that the token is supported by the gateway before token transfering, i.e. check that the returned token address is not `address(0)`.

Go back to Findings Summary

# Appendix A: How to cite

Please cite this document as:

Ackee Blockchain, Axelar: Express GMP service, 13.1.2023.

# 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.

ackee

# Thank You

Ackee Blockchain a.s.



Prague, Czech Republic


hello@ackeeblockchain.com


https://discord.gg/z4KDUbuPxq