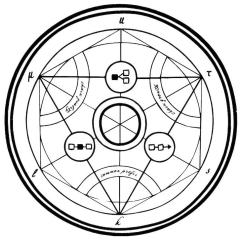
Axelar Gateway v2.1

Smart Contract Audit

March 01, 2022

Common Prefix



Overview

Introduction

Common Prefix was commissioned to perform a security audit on Axelar's `cgp-solidity-gateway-2.1.0` smart contracts. The files inspected are the following:

`AxelarGatewayProxyMultisig.sol`
`AxelarGatewayProxy.sol`
`ERC20.sol`
`Ownable.sol`
`ERC20Permit.sol`
`AxelarGatewayProxySinglesig.sol`
`AxelarGatewayMultisig.sol`
`AxelarGatewaySinglesig.sol`
`ECDSA.sol`
`DepositHandler.sol`
`AdminMultisigBase.sol`
`MintableCappedERC20.sol`
`Context.sol`
`AxelarGateway.sol`
`EternalStorage.sol`
`BurnableMintableCappedERC20.sol`

Findings Severity Breakdown

The findings are classified under the following severity categories according to the impact and the likelihood of an attack.

Level	Description
Critical	Logical errors or implementation bugs that are easily exploited and may lead to any kind of loss of funds
High	Logical errors or implementation bugs that are likely to be exploited and may have disadvantageous economic impact or contract failure
Medium	Issues that may break the intended contract logic or lead to DoS attacks
Low	Issues harder to exploit (exploitable with low probability), clumsy logic or implementation that lead to poor contract performance
Informational	Advisory comments and recommendations that could help make the codebase clearer, more readable and easier to maintain

Disclaimer

Note that this audit does not give any warranties on the bug-free status of the given smart contracts, i.e. the evaluation result does not guarantee the nonexistence of any further findings of security issues. This audit report is intended to be used for discussion purposes only. We always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of the project.

Findings

Critical

None found.

High

None found.

Medium

None found.

Low

LOW-1	Clumsy implementation of ERC20 inheritance
Contract(s)	`ERC20Permit`, `MintableCappedERC20`
Status	**Open**

Description

In MintableCappedERC20.sol the ERC20 contract is essentially imported and inherited twice, since ERC20Permit.sol also imports it and inherits from it. At the same time, ERC20Permit contract does not construct an ERC20 instance, though it should.

```
/// MintableCappedERC20.sol
import { ERC20 } from './ERC20.sol'; // CP: also imported in ERC20Permit
import { ERC20Permit } from './ERC20Permit.sol';
import { Ownable } from './Ownable.sol';
```

```
// CP: ERC20 should omitted
contract MintableCappedERC20 is ERC20, ERC20Permit, Ownable {
    uint256 public cap;

    constructor(
        string memory name,
        string memory symbol,
        uint8 decimals,
        uint256 capacity
        // CP: ERC20 should be constructed in ERC20Permit instead
    ) ERC20(name, symbol, decimals) ERC20Permit(name) Ownable() {
        cap = capacity;
    }

    // ...
}


/// ERC20Permit.sol
import { ERC20 } from './ERC20.sol';

abstract contract ERC20Permit is ERC20 {

    constructor(string memory name) {
        DOMAIN_SEPARATOR = keccak256(
            abi.encode(
                DOMAIN_TYPE_SIGNATURE_HASH,
                keccak256(bytes(name)),
                keccak256(bytes('1')),
                block.chainid,
                address(this)
            )
         );
     }
     /// ...
}
```

Recommendation

We suggest removing the duplicates import/inheritance and construct the ERC20 instance in the constructor of ERC20Permit contract.

Informational-Suggestions

INFO-2	Inconsistent way of checking the validOwners and validOperators
Contract(s)	`AxelarGateWayMutlisig`
Status	**Open**

Description

In AxelarGateWayMutlisig::_execute valid owners of the current active ownerEpoch are checked in a separate function than the rest of the recent enough epochs. On the contrary, the validity of the operators either of the current or of any other recent enough operatorEpoch is performed within a single function. However, there seems to be no reason for such a differentiation, since function `_areValidOwnersInEpoch()` is nowhere else used.

```
if (signersRole == Role.Owner) {
    areValidCurrentOwners = _areValidOwnersInEpoch(_ownerEpoch(), signers);
    areValidRecentOwners = areValidCurrentOwners || areValidPreviousOwners(signers);
} else if (signersRole == Role.Operator) {
    areValidRecentOperators = _areValidRecentOperators(signers);
}
```

Under this scope, we see no reason not to have a simpler _areValidRecentOwners() function, just like the one checking the validity of operators.

Recommendation

We suggest substituting function `AxelarGateWayMutlisig::areValidPreviousOwners` with one similar to `AxelarGateWayMutlisig::areValidRecentOperators` and thus enhance code consistency but also simplify the checks in `AxelarGateWayMutlisig::_execute`.

About Common Prefix

Common Prefix is a blockchain research, development, and consulting company consisting of a small number of scientists and engineers specializing in many aspects of blockchain science. We work with industry partners who are looking to advance the state-of-the-art in our field to help them analyze and design simple but rigorous protocols from first principles, with provable security in mind.

Our consulting and audits pertain to theoretical cryptographic protocol analyses as well as the pragmatic auditing of implementations in both core consensus technologies and application layer smart contracts.