Axelar Gateway

Smart Contract Audit

November 22, 2021

Common Prefix

Overview

Introduction

Common Prefix was commissioned to perform a security audit on Axelar's `cgp-solidity-gateway` smart contracts, at commit hash 120b011b37ae1f5228a5302065e7c67238be9160. The files inspected are the following:

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

Executive Summary

The audited codebase is well written and of professional quality. We found no critical or high severity issues, which denote an overall healthy protocol and implementation.

We find that the current burning mechanism may not be economically practical for chains with non-negligible gas prices, since it requires the deployment and destruction of a new Burner contract upon each burn transaction. Apart from this, we mostly point out some recommendations that we believe could make the codebase clearer and simpler - thus more robust against future bugs - but also more gas efficient. A final observation is made on the decentralization of trust in regards to the upgrade mechanism.

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	Burning scheme seems unaffordable for chains with considerable gas costs
Contract(s)	`AxelarGateway.sol, Burner.sol`
Status	**Open**

Description

In the current implementation a user of the Axelar Network that wishes to burn some amount should transfer the amount to a specific (unused address) that is computed by CREATE2 operation given appropriate parameters. Under normal conditions, Axelar's gateway contracts are then triggered to construct a Burner contract at that address. This Burner contract `burn`()s all of its balance (previously transferred by the user) and finally `selfdestruct`()s. This design

seems to be related to the level of abstraction wished for the network's operation across the supported chains. However, deploying and destructing a contract for each burn operation seems to be an unaffordable solution for chains with non-negligible gas costs, such as the Ethereum blockchain.

Recommendation

This issue seems to be strongly related to the overall protocol design. However, we believe that solutions that avoid such high gas cost operations are worth exploring, especially when concerning transactions that are likely to also serve small amounts.

LOW-2	Shared storage scheme is of high complexity and gas costs
Contract(s)	*Most of the contracts affected*
Status	**Open**

Description

It is not obvious why the Eternal Storage scheme has been chosen given the complexity and the increased gas costs that it comes with.

Recommendation

Explore the alternative of the simpler Inherited Storage shared storage scheme.

Informational

INFO-1	Unnecessary special treatment of 'implementation' storage slot key
Contract(s)	`AxelarGateway.sol, AxelarGatewayProxy.sol`
Status	**Open**

Description

In the Eternal Storage schema, state variable values are stored into a mapping of appropriate type. Every key in the mapping, i.e. the final storage slot, is usually computed as the hash of a short and intuitive description, e.g.

```
bytes32 internal constant KEY_OPERATOR_EPOCH = keccak256('operator-epoch');
```

However special treatment is given to the key of the logic contract's address, following the guidelines of EIP1967:

```
/// @dev Storage slot with the address of the current factory.
/// `keccak256('eip1967.proxy.implementation') - 1`.
      bytes32 internal constant KEY_IMPLEMENTATION =

bytes32(0x360894a13ba1a3210667c828492db98dca3e2076cc3735a920a3ca505d382bbc);
```

However such a special treatment is unnecessary in the context of Eternal Storage, since storage collisions between variables defined by the proxy and the implementation contract are avoided by design. In fact, this guideline should be followed in cases of Unstructured Storage Proxies.

Recommendation

We suggest changing `keccak256('eip1967.proxy.implementation') - 1` to `keccak256('implementation')` for clarity and uniformity.

INFO-2	`setup` function in logic contract called only by a proxy
Contract(s)	`AxelarGatewayMultisig.sol, AxelarGatewaySinglesig.sol`
Status	**Open**

Description

It is important that the `setup` function in contracts `AxelarGatewayMultisig.sol`, `AxelarGatewaySinglesig.sol` should be able to be called only by a proxy contract, so that the logic contract cannot be manipulated by an attacker. Currently, this is ensured by the following requirement:

```
// Prevent setup from being called on a non-proxy (the implementation)
      require(implementation() != address(0), 'NOT_PROXY');
```

While this requirement ensures that a direct call to setup will fail, it would be clearer to define and use an onlyProxy modifier:

```
address private immutable __self = address(this);
modifier onlyProxy() {
          require(address(this) != __self, "Function must be called through
delegatecall");
          require(_getImplementation() == __self, "Function must be called
through active proxy");
    _;
}
```

Recommendation

We recommend using an onlyProxy modifier following the best practices.

INFO-3	Current upgrade mechanism rises a centralization issue
Contract(s)	*Whole protocol affected*
Status	**Open**

Description

At the heart of the protocol lies a government-like mechanism utilized for taking an important action after a sufficient number of owners (or, in some cases, operators) have voted for it. Such an action is, for example, the deployment of a new token on a chain. However, this voting procedure is not part of the upgrade mechanism. Instead, the administrators are responsible for

the protocol upgrade. More specifically, a sufficient number of administrators should agree to upgrade the protocol in respect to a specific threshold. This threshold value, as well as the administrator addresses, the logic contract itself and other sensitive parameters of the protocol are set during the initial construction of the logic contract and then altered via the upgrade mechanism. Though the administrators may be considered trusted it is still possible that some of them coalesce to gain more power over the protocol. Due to the significance of the upgrade actions it is important to decentralize the trust as much as possible.

Recommendation

We recommend adopting a voting procedure for the upgrade actions as well.

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.