Axelar Transceiver Audit Report

Prepared by Cyfrin

Version 3.0

Lead Auditors

Giovanni Di Siena

0kage

Assisting Auditors

Hans

July 2, 2024

Contents

1 About Cyfrin

Cyfrin is a Web3 security company dedicated to bringing industry-leading protection and education to our partners and their projects. Our goal is to create a safe, reliable, and transparent environment for everyone in Web3 and DeFi. Learn more about us at cyfrin.io.

2 Disclaimer

The Cyfrin team makes every effort to find as many vulnerabilities in the code as possible in the given time but holds no responsibility for the findings in this document. A security audit by the team does not endorse the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the solidity implementation of the contracts.

3 Risk Classification

	Impact: High	**Impact: Medium**	**Impact: Low**
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

4 Protocol Summary

The `AxelarTransceiver` is an upgradeable Transceiver contract that is intended to integrate with the Wormhole Native Token Transfers (NTT) protocol to send and receive messages cross-chain. The Transceiver itself relays messages via the Axelar General Message Passing (GMP) infrastructure. This Transceiver is intended to bridge the wrapped stETH token (wstETH), an NTT token, across chains.

This Transceiver is expected to work along with the Wormhole Transceiver to provide a robust bridging infrastructure for the wrapped version of Lido's staked ETH token.

5 Audit Scope

Cyfrin conducted an audit on `AxelarTransceiver` based on the code present in the repository commit hash fc2dfea.

The following files were included in the scope of the audit:

- src/axelar/AxelarTransceiver.sol
- src/axelar/Structs.sol
- src/axelar/interfaces/IAxelarTransceiver.sol
- test/axelar/AxelarTransceiver.t.sol
- test/axelar/mock/MockGasService.sol
- test/axelar/mock/MockGateway.sol

6 Executive Summary

Over the course of 9 days, the Cyfrin team conducted an audit on the Axelar Transceiver smart contracts provided by Wormhole Foundation. In this period, a total of 4 issues were found.

Our audit of the Wormhole Axelar Transceiver identified a medium-risk issue where updates to the Axelar chain name or transceiver address could potentially disrupt the execution of in-flight messages on the destination chain.

Additionally, we found that the owner-controlled function responsible for updating Axelar chain details lacks necessary input validations and event emissions, which could lead to inconsistencies and tracking issues.

Furthermore, our evaluation revealed inadequacies in the current testing framework. It notably lacks comprehensive end-to-end tests that simulate the full lifecycle of messages passing through the Wormhole NTT Manager and Axelar transceiver infrastructure.

All issues identified during the audit have been addressed and successfully resolved.

Summary

Project Name	Axelar Transceiver
Repository	example-wormhole-axelar-wsteth
Commit	fc2dfea0b16b...
Audit Timeline	Jun 10th - Jun 20th
Methods	Manual Review, Stateful Fuzzing

Issues Found

Critical Risk	0
High Risk	0
Medium Risk	1
Low Risk	1
Informational	2
Gas Optimizations	0
Total Issues	4

Summary of Findings

[M-1] Updating existing Axelar chainId or transceiver address can potentially DoS transaction execution on recipient chain	Resolved
[L-1] Missing input validations in `AxelarTransceiver.setAxelarChainId`	Resolved
[I-1] Missing event emission in `AxelarTransceiver::setAxelarChainId`	Resolved
[I-2] Insufficient test coverage	Resolved

7 Findings

7.1 Medium Risk

7.1.1 Updating existing Axelar chainId or transceiver address can potentially DoS transaction execution on recipient chain

Description: `AxelarTransceiver::setAxelarChainId` allows the contract owner to update Axelar chain name & transceiver address. Updating an existing address can potentially DoS in-flight messages that have already left the source chain and approved by the Axelar gateway on the recipient chain.

Consider the following sequence of events:

1. Alice sends a message on the source chain that uses the existing recipient Axelar transceiver address.
2. The message gets approved on the recipient chain.
3. The owner updates the source chain Axelar transceiver address on the recipient transceiver.

With the above sequence, the in-flight message post-approval cannot be executed because the gateway approval becomes inconsistent with the updated Axelar transceiver address.

Impact: Since the current logic allows an update to the existing Axelar transceiver address, in-flight messages can fail to get executed on the recipient chain in certain edge-case scenarios.

Proof of Concept: Consider the following PoC:

```
contract AxelarTransceiverInflightMessages is Test {
    address constant OWNER = address(1004);
    uint64 constant RATE_LIMIT_DURATION = 0;
    bool constant SKIP_RATE_LIMITING = true;

    uint256 constant DEVNET_GUARDIAN_PK =
        0xcfb12303a19cde580bb4dd771639b0d26bc68353645571a8cff516ab2ee113a0;

    AxelarTransceiver sourceTransceiver;
    IAxelarGateway gateway;
    IAxelarGasService gasService;
    NttManager sourceNttmanager;
    wstETHL2Token sourceToken;
    uint16 sourceChainId;

    AxelarTransceiver recipientTransceiver;
    NttManager recipientNttManager;
    wstETHL2Token recipientToken;
    uint16 recipientChainId;

    function setUp() public {
        gateway = IAxelarGateway(new MockAxelarGateway());
        gasService = IAxelarGasService(address(new MockAxelarGasService()));

        // Setup Source Infrastructure
        sourceChainId = 1;
        sourceToken = new wstETHL2Token("Wrapped StEth Source", "wStEthSrc", OWNER, OWNER);
        address sourceManagerImplementation = address(
            new NttManager(
                address(sourceToken),
                IManagerBase.Mode.LOCKING,
                sourceChainId,
                RATE_LIMIT_DURATION,
                SKIP_RATE_LIMITING
            )
        );
        sourceNttmanager = NttManager(address(new ERC1967Proxy(sourceManagerImplementation, "")));
```

```
sourceNttmanager.initialize();
sourceNttmanager.transferOwnership(OWNER);
address srcTransceiverImplementation =
    address(new AxelarTransceiver(address(gateway), address(gasService),
    ↪  address(sourceNttmanager)));
sourceTransceiver = AxelarTransceiver(address(new ERC1967Proxy(srcTransceiverImplementation,
↪  "")));
sourceTransceiver.initialize();
vm.prank(OWNER);
sourceNttmanager.setTransceiver(address(sourceTransceiver));

// Setup Recipient Infrastructure
recipientChainId = 2;
recipientToken = new wstETHL2Token("Wrapped StEth Recipient", "wStEthRcpt", OWNER, OWNER);
address recipientManagerImplementation = address(
    new NttManager(
        address(recipientToken),
        IManagerBase.Mode.LOCKING,
        recipientChainId,
        RATE_LIMIT_DURATION,
        SKIP_RATE_LIMITING
    )
);
recipientNttManager = NttManager(address(new ERC1967Proxy(recipientManagerImplementation, "")));
recipientNttManager.initialize();
recipientNttManager.transferOwnership(OWNER);
address rcptTransceiverImplementation =
    address(new AxelarTransceiver(address(gateway), address(gasService),
    ↪  address(recipientNttManager)));
recipientTransceiver = AxelarTransceiver(address(new
↪  ERC1967Proxy(rcptTransceiverImplementation, "")));
recipientTransceiver.initialize();
vm.prank(OWNER);
recipientNttManager.setTransceiver(address(recipientTransceiver));


bytes32 sourceNttManagerAddress = bytes32(uint256(uint160(address(sourceNttmanager))));
bytes32 recipientNttManagerAddress = bytes32(uint256(uint160(address(recipientNttManager))));

// set peer ntt manager on source
vm.prank(OWNER);
sourceNttmanager.setPeer(
    recipientChainId,
    recipientNttManagerAddress,
    18,
    100000000
);

// set peer ntt manager on recipient
vm.prank(OWNER);
recipientNttManager.setPeer(
    sourceChainId,
    sourceNttManagerAddress,
    18,
    100000000
);

string memory sourceChainName = "srcChain";
string memory sourceAxelarAddress = "srcAxelar";

string memory recipientChainName = "recipientChain";
string memory recipientAxelarAddress = "recipientAxelar";
```

```
        vm.prank(OWNER);
        sourceTransceiver.setAxelarChainId(recipientChainId, recipientChainName,
        ↪  recipientAxelarAddress);

        vm.prank(OWNER);
        recipientTransceiver.setAxelarChainId(sourceChainId, sourceChainName, sourceAxelarAddress);

        // token mint source
        vm.prank(OWNER);
        sourceToken.mint(address(sourceNttmanager), 10e6 ether);

        vm.prank(OWNER);
        recipientToken.mint(address(recipientNttManager), 10e6 ether);

    }


    function testAxelarInflightMessages() public {
        bytes32 refundAddress = bytes32(uint256(1011));
        TransceiverStructs.TransceiverInstruction memory instruction =
            TransceiverStructs.TransceiverInstruction(0, bytes(""));

        // SEND MESSAGE ON SOURCE CHAIN
        bytes32 to = bytes32(uint256(1234));
        uint64 amount = 12345670000000000;
        bytes memory nttPayload;
        {
            bytes4 prefix = TransceiverStructs.NTT_PREFIX;
            uint8 decimals = 18;
            bytes32 srcToken = bytes32(uint256(uint160(address(sourceToken))));
            uint16 toChain = 2;
            nttPayload = abi.encodePacked(prefix, decimals, amount, srcToken, to, toChain);
        }

        bytes memory nttManagerMessage;
        {
            uint16 length = uint16(nttPayload.length);
            bytes32 messageId = bytes32(uint256(0));
            bytes32 sender = bytes32(uint256(1));
            nttManagerMessage = abi.encodePacked(messageId, sender, length, nttPayload);

        }
        bytes32 sourceNttManagerAddress = bytes32(uint256(uint160(address(sourceNttmanager))));
        bytes32 recipientNttManagerAddress = bytes32(uint256(uint160(address(recipientNttManager))));

        vm.prank(address(sourceNttmanager));
        sourceTransceiver.sendMessage(
            recipientChainId, instruction, nttManagerMessage, recipientNttManagerAddress, refundAddress
        );

        // APPROVE MESSAGE ON AXELAR GATEWAY

        IAxelarGateway.Message[] memory messages = new IAxelarGateway.Message[](1);
        messages[0] = IAxelarGateway.Message(
            "srcChain",
            "0",
            "srcAxelar",
            address(sourceNttmanager),
            keccak256(nttManagerMessage)
        );
        IAxelarGateway.WeightedSigner[] memory signers = new IAxelarGateway.WeightedSigner[](1);
        signers[0] = IAxelarGateway.WeightedSigner({
            signer: address(0xABC),
            weight: 1
        });

        // Dummy data for Proof
        IAxelarGateway.Proof memory proof = IAxelarGateway.Proof({
            signers: IAxelarGateway.WeightedSigners({
                signers: signers,
                threshold: 1,
                nonce: keccak256(abi.encodePacked("nonce"))
            }),
            signatures: new bytes[](1)
        });
        proof.signatures[0] = hex"00";  // Dummy signature


        gateway.approveMessages(messages, proof);

        // CHANGE PEER ADDRESS
        string memory sourceChainName2 = "srcChain2";
        string memory sourceAxelarAddress2 = "srcAxelar2";

        // UPDATE THE PEER ADDRESS ON RECIPIENT CHAIN
        vm.prank(OWNER);
        recipientTransceiver.setAxelarChainId(sourceChainId, sourceChainName2, sourceAxelarAddress2);

        // EXECUTE ON RECIPIENT CHAIN
        bytes memory payload = abi.encode(sourceNttManagerAddress, nttManagerMessage,
        ↪   recipientNttManagerAddress);

        vm.expectRevert(IAxelarGateway.NotApprovedByGateway.selector);
        recipientTransceiver.execute(bytes32(0), sourceChainName2, sourceAxelarAddress2, payload);
    }
}
```

Recommended Mitigation: Since `AxelarTransceiver` is an upgradeable contract, consider allowing the owner to set the Axelar chain name and transceiver address only once via `AxelarTransceiver::setAxelarChainId`. Although they are expected to be extremely rare, consider making any future updates to the chain name or transceiver address via contract upgrades.

Wormhole Foundation: Fixed in PR 22.

Cyfrin: Acknowledged.

7.2 Low Risk

7.2.1 Missing input validations in `AxelarTransceiver.setAxelarChainId`

Description: `AxelarTransceiver::setAxelarChainId` is an `onlyOwner` function responsible for updating the Wormhole chain ID, Axelar chain name, and the transceiver address on destination chains. Unlike the `WormholeTransceiver`, this function lacks the following input validations:

- `chainId == 0`
- empty `chainName`
- empty `transceiverAddress`

Recommended Mitigation: Consider adding input validations to `AxelarTransceiver::setAxelarChainId`.

Wormhole Foundation: Fixed in PR 22.

Cyfrin: Acknowledged.

7.3 Informational

7.3.1 Missing event emission in `AxelarTransceiver::setAxelarChainId`

Description: `AxelarTransceiver::setAxelarChainId` is an `onlyOwner` operation that sets the wormhole chain ID, corresponding Axelar transceiver address and chain name for a given destination chain. However, this operation does not emit any event at present that could enable off-chain tracking.

Recommended Mitigation: Considering the importance of this operation, it is recommended that an event be emitted whenever Axelar chain name and/or transceiver address is updated.

Wormhole Foundation: Fixed in PR 22.

Cyfrin: Acknowledged.

7.3.2 Insufficient test coverage

Description: The following observations were made after reviewing the current test suite:

1. A Sepolia testnet fork is created but not used.
2. Mock Axelar gas service and gateway contracts do not accurately reflect the state changes of deployed contracts.
3. Tests specific to cross-chain attacks such as message replays and gas computations are not handled.
4. Rate limiting is one of the key risk mitigation features of `NttManager`. There are no tests to verify rate limiting with Axelar transceivers.
5. End-to-end tests that capture the full lifecycle of messages from the source `NttManager` to the recipient address are not implemented.
6. Tests such as `testFail_sendMessageChainNotRegisterred` and `testFail_executeNotTrustedAddress` do not check for specific points of failure. In the context of cross-chain transfers with multiple potential points of failure, it is a best practice to write tests that assert specific state changes.

Recommended Mitigation: Consider expanding the current test suite to:

- Use deployed contracts over mocks.
- Use mocks that closely reflect deployed contract functionality.
- Include tests with realistic rate limiters that reflect actual deployment conditions.
- Include end-to-end testing that starts at `NttManager::transfer` and ends at the recipient address.
- Test for specific points of failure relevant to cross-chain transfers.

Wormhole Foundation: Fixed in PR 22.

Cyfrin: Acknowledged.