SPEARBIT

Axiom Remote & Broadcast Contracts Security Review

Auditors

Riley Holterhus, Lead Security Researcher

Blockdev, Security Researcher

Lucas Goiriz, Junior Security Researcher

David Chaparro, Junior Security Researcher

Report prepared by: Lucas Goiriz

February 7, 2024

Contents

1 About Spearbit

Spearbit is a decentralized network of expert security engineers offering reviews and other security related services to Web3 projects with the goal of creating a stronger ecosystem. Our network has experience on every part of the blockchain technology stack, including but not limited to protocol design, smart contracts and the Solidity compiler. Spearbit brings in untapped security talent by enabling expert freelance auditors seeking flexibility to work on interesting projects together.

Learn more about us at spearbit.com

2 Introduction

Axiom gives smart contracts trustless access to the entire history of Ethereum and arbitrary ZK-verified compute over it. Developers can send on-chain queries into Axiom, which are trustlessly fulfilled with ZK-verified results sent in a callback to the developer's smart contract. This allows developers to build rich on-chain applications without additional trust assumptions.

Disclaimer: This security review does not guarantee against a hack. It is a snapshot in time of axiom-v2-contracts-working remote & broadcast according to the specific commit. Any modifications to the code will require a new security review.

3 Risk classification

Severity level	Impact: High	Impact: Medium	Impact: Low
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

3.1 Impact

- High - leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
- Medium - global losses <10% or losses to only a subset of users, but still unacceptable.
- Low - losses will be annoying but bearable--applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.

3.2 Likelihood

- High - almost certain to happen, easy to perform, or not easy but highly incentivized
- Medium - only conditionally possible or incentivized, but still relatively likely
- Low - requires stars to align, or little-to-no incentive

3.3 Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

4 Executive Summary

Over the course of 7 days in total, Axiom engaged with Spearbit to review the axiom-v2-contracts-working remote & broadcast protocol. In this period of time a total of **9** issues were found.

Summary

Project Name	Axiom
Repository	axiom-v2-contracts-working remote & broadcast
Commit	7f8585...df8ba0
Type of Project	Data availability, ZK
Audit Timeline	Jan 15 to Jan 22

Issues Found

Severity	**Count**	**Fixed**	**Acknowledged**
Critical Risk	1	1	0
High Risk	0	0	0
Medium Risk	0	0	0
Low Risk	1	1	0
Gas Optimizations	1	1	0
Informational	6	4	2
Total	**9**	**7**	**2**

5 Findings

5.1 Critical Risk

5.1.1 `AxiomV2GnosisHashiAmbHeaderVerifier` does not whitelist `yaru.adapters()`

Severity: Critical Risk

Context: AxiomV2GnosisHashiAmbHeaderVerifier.sol#L53-L55, AxiomV2GnosisHashiAmbHeaderVerifier.sol#L58-L62

Description: The `_checkL1Broadcaster()` function within the `onlyFromL1Broadcaster()` modifier intends to check that the cross-chain call to `updateLatestPmmr()` originated from the L1 broadcaster. It does so by throwing the `NotBroadcaster()` error in the following scenarios:

- `msg.sender != address(yaru)`
- `yaru.chainId() != SOURCE_CHAIN_ID`
- `yaru.sender() != L1_BROADCASTER`

While these existing checks are indeed necessary, there's an essential check missing for `yaru.adapters()`. This function returns the list of oracles that are confirming the legitimacy of the cross-chain message, and without verifying this value, *any* oracle can be used. As a result, an attacker can validate fake cross-chain messages using their own malicious contracts, which allows them to store arbitrary PMMRs in the system.

Recommendation: Add a check that the `yaru.adapters()` match the ones expected from an actual broadcast. Since the `GnosisHashiAmbBroadcastModule` hardcodes a single `ambAdapter` oracle, this check would be as follows:

```
IOracleAdapter memory oracleAdapters = yaru.adapters();
require(oracleAdapters.length == 1);
// Note: might need to add `ambAdapter` to constructor
require(oracleAdapters[0] == ambAdapter);
```

Additional Note: In the current Hashi bridge, the `yaru` contract has a bug that allows a DoS of any cross-chain message. This is possible because the `yaru` contract does not prevent someone from exhausting real message ids through fake payloads verified by malicious oracles. This bug is being fixed by the Hashi team in their next deployment, so it is recommended that Axiom waits until this external dependency is fixed.

Axiom: For completeness, we've prepared a fix in PR 241. As discussed, we plan to rework this adapter for an updated version of Hashi and will not deploy the Hashi adapter in its current form (i.e. `AxiomV2GnosisHashi` broadcaster and header verifier will not be deployed as is).

Spearbit: Verified.

5.2 Low Risk

5.2.1 Excess ETH sent to `AxiomV2Broadcaster` won't be refunded to users

Severity: Low Risk

Context: BroadcastModule.sol#L32-L47

Description: If the sum of all `bridgePayment` values is strictly less than `msg.value`, the excess ETH won't be returned to the user after the call. Instead, it will be kept in the contract and will sponsor the following users' calls, which results in an unnecessarily expensive call for the naive user.

Recommendation: Consider implementing a refunding mechanism to protect naive users from this scenario. A straightforward approach would be to add such a refunding mechanism in `AxiomV2Broadcaster._sendPmmr()`, once all the modules are executed:

```
  function _sendPmmr(
      uint32 pmmrSize,
      bytes32 pmmrSnapshot,
      Channel[] calldata channels,
      BroadcastParams[] calldata broadcastParamsList
  ) internal {
      uint256 length = channels.length;
      if (length != broadcastParamsList.length) {
          revert ChannelsAndBroadcastParamsDifferInLength();
      }

      for (uint256 i; i < length;) {
          Channel calldata channel = channels[i];
          BroadcastParams calldata params = broadcastParamsList[i];
          address broadcastModule = channelToBroadcastModule[channel.chainId][channel.bridgeId];
          if (broadcastModule == address(0)) {
              revert ChannelNotFound();
          }

          (bool success,) = broadcastModule.delegatecall(
              abi.encodeWithSelector(
                  IBroadcastModule.broadcast.selector,
                  channel.chainId,
                  pmmrSize,
                  pmmrSnapshot,
                  params.bridgeMetadata,
                  params.bridgePayment
              )
          );

          if (!success) {
              revert BroadcastFailed();
          }
          unchecked {
              ++i;
          }
      }
+     if (address(this).balance > 0) {
+         payable(msg.sender).sendValue(address(this).balance);
+     }
  }
```

Axiom: Fixed in PR 238.

Spearbit: Verified.

5.3 Gas Optimization

5.3.1 State variable visibility can be further restricted

Severity: Gas Optimization

Context: AxiomV2PolygonHeaderVerifier.sol#L14, RemoteHeaderVerifier.sol#L18

Description: State variables declared with `public` visibility enforce the creation of corresponding getter functions by the Solidity compiler. In certain cases however, this may be unnecessary and the visibility can be further restricted to `internal` or `private`, resulting in a smaller contract bytecode. For instance:

- AxiomV2PolygonHeaderVerifier.sol#L14: The constant `L2_STATE_SENDER` state variable is solely used for internal operations and it does not seem to require public visibility.
- RemoteHeaderVerifier.sol#L18: The `SOURCE_CHAIN_ID` state variable already has a custom getter, namely `getSourceChainId`. Thus, it is unnecessary for it to have `public` visibility, as it would generate an additional redundant getter.

Recommendation: Consider restricting the visibility of the aforementioned state variables to either `internal` or `private` to reduce contract size and, consequently, deployment gas costs.

Axiom: Changed visibility in `RemoteHeaderVerifier` in PR 237. Decided not to change the `AxiomV2PolygonHeaderVerifier` visibility to make the contract more easily auditable.

Spearbit: Verified.

5.4 Informational

5.4.1 Magic number would lead to `RetryableData` error in Arbitrum's module

Severity: Informational

Context: BroadcastModule.sol#L58

Description: The current implementation in `BroadcastModule` does not include checks for `maxGas` and `gasPriceBid` being non-set to `1`. As per Arbitrum's natspec in `createRetryableTicket`, setting these values to `1` would lead to a revert with error `RetryableData`. This behavior might not be immediately apparent to the users, leading to confusion as this function does not include any mention of this magic number.

```
 * @param gasLimit Max gas deducted from user's L2 balance to cover L2 execution. Should not be set to
↪  1 (magic value used to trigger the RetryableData error)
 * @param maxFeePerGas price bid for L2 execution. Should not be set to 1 (magic value used to trigger
↪  the RetryableData error)
```

Recommendation: By Arbitrum's documentation, it is recommended to add explicit checks to ensure that `maxGas` and `gasPriceBid` are not set to `1` and/or explicit documentation regarding this error message that may appear under abnormal values.

Axiom: Fixed in PR 236 by documenting this behavior.

Spearbit: Verified.

5.4.2 Missing event emission in critical `TIMELOCK_ROLE` actions within `AxiomV2Broadcaster`

Severity: Informational

Context: AxiomV2Broadcaster.sol#L70-L76, AxiomV2Broadcaster.sol#L79-L81

Description: `addChannel()` and `removeChannel()`, critical functions restricted to Axiom's multisig address with `TIMELOCK_ROLE`, are responsible for adding and removing channels for message passing. To enhance transparency for users, it is advisable to emit corresponding events for these changes. This allows off-chain components to monitor the available channels for message passing.

Recommendation: Consider implementing `ChannelAdded` and `ChannelRemoved` events to reflect the updates. As `AxiomV2Broadcaster` implements the `IAxiomV2Broadcaster` interface, these events should be defined there.

Axiom: Fixed in PR 233.

Spearbit: Verified.

5.4.3 Unnecessary `chainId` check on broadcast module constructors

Severity: Informational

Context: ArbitrumBroadcastModule.sol#L32, BaseBroadcastModule.sol#L18, GnosisHashiAmbBroadcastModule.sol#L48, OptimismBroadcastModule.sol#L20, PolygonBroadcastModule.sol#L30, ScrollBroadcastModule.sol#L28

Description: The constructors of various broadcast modules currently include a check to verify that the chainId passed as a constructor argument matches either the mainnet chain id or the testnet chain id. This design allows the same broadcast module to be used for both mainnets and testnets.

However, this check can be circumvented by hardcoding chainId and creating distinct contracts for each chain and test chains.

For instance, let's consider the `OptimismBroadcastModule`. It could be refactored into two separate contracts: `OptimismMainnetBroadcastModule` and `OptimismSepoliaBroadcastModule`. The constructor for OptimismMainnetBroadcastModule would look like this:

```
constructor(uint64 chainId, address _axiomV2OptimismHeaderVerifier, address l1Messenger)
    OpStackBroadcastModule(OPTIMISM_CHAIN_ID, _axiomV2OptimismHeaderVerifier, l1Messenger) { }
```

Similarly, a `OptimismSepoliaBroadcastModule` contract would be defined with the hardcoded `chainId` as `OPTIMISM_SEPOLIA_CHAIN_ID`. Since two contracts need to be deployed in any case (i.e. one for mainnet and the other for the testnet), this approach would save gas on deployment by eliminating the need for the check:

```
if (chainId != OPTIMISM_CHAIN_ID && chainId != OPTIMISM_SEPOLIA_CHAIN_ID) { /*...*/ }
```

This modification comes with the trade-off of having two separate contracts, but it provides gas savings during deployment.

Recommendation: Consider defining a single contract for each chain and test chain with their respective `chainIds` hardcoded for gas savings on deployment.

Axiom: Acknowledged. Our thinking here is that it would be helpful to have the same contract running on Optimism and Optimism Sepolia, just with different `chainId` arguments. Agree this has a bit of deployment overhead, but we think it would be easier for developers.

Spearbit: Acknowledged.

5.4.4 Unnecessary declaration of local variables in `_broadcast` function

Severity: Informational

Context: GnosisHashiAmbBroadcastModule.sol#L75, PolygonBroadcastModule.sol#L51, ScrollBroadcastModule.sol#L47-L49

Description: The `_broadcast` function of a `BroadCastModule` is responsible for broadcasting a PMMR update to the remote chain. To achieve this, it accepts various parameters, which are handled by each specific module for their corresponding bridge. In certain instances, the locally generated variables during this handling are unnecessary. The data can be passed directly to function calls in a more readable manner by utilizing named function parameters. Additionally, in some cases, these variables can be passed directly to struct definitions, resulting in a reduced contract size for specific scenarios. Each case is detailed below:

- Applying the following change in GnosisHashiAmbBroadcastModule.sol:

```
- bytes memory data = abi.encodeWithSelector(IRemoteHeaderVerifier.updateLatestPmmr.selector,
↪ pmmrSize, pmmrHash);

  Message[] memory messages = new Message[](1);
- messages[0] = Message({ to: axiomV2GnosisHashiAmbHeaderVerifier, toChainId: CHAIN_ID, data:
↪ data });
+ messages[0] = Message({
+     to: axiomV2GnosisHashiAmbHeaderVerifier,
+     toChainId: CHAIN_ID,
+     data: /* data */ abi.encodeWithSelector(
+         IRemoteHeaderVerifier.updateLatestPmmr.selector,
+         pmmrSize,
+         pmmrHash
+     )
+ });
  address[] memory relays = new address[](1);
```

  yields the following change of contract size:

  - **Before**:

Contract	Size (kB)	Margin (kB)
GnosisHashiAmbBroadcastModule	3.119	21.457

  - **After**:

Contract	Size (kB)	Margin (kB)
GnosisHashiAmbBroadcastModule	3.113	21.463

- Applying the following change to PolygonBroadcastModule.sol:

```
- bytes memory data = abi.encode(pmmrSize, pmmrSnapshot);

- L1_STATE_SENDER.syncState(axiomV2PolygonHeaderVerifier, data);
+ L1_STATE_SENDER.syncState({
+     receiver: axiomV2PolygonHeaderVerifier,
+     data: /* data */ abi.encode(pmmrSize, pmmrSnapshot)
+ });
```

  yields the following change of contract size:

  - **Before**:

Contract	Size (kB)	Margin (kB)
PolygonBroadcastModule	1.586	22.99

- **After**:

Contract	Size (kB)	Margin (kB)
PolygonBroadcastModule	1.526	23.05

- Applying the following change ScrollBroadcastModule.sol:

```
  - uint256 gasLimit = abi.decode(bridgeMetadata, (uint256));
  - bytes memory data =
  -     abi.encodeWithSelector(IRemoteHeaderVerifier.updateLatestPmmr.selector, pmmrSize,
↪  pmmrSnapshot);

  - L1_MESSENGER.sendMessage{ value: bridgePayment }(axiomV2ScrollHeaderVerifier, 0, data,
↪  gasLimit, msg.sender);
  + L1_MESSENGER.sendMessage{ value: bridgePayment }({
  +     target: axiomV2ScrollHeaderVerifier,
  +     value: 0,
  +     message: /* data */ abi.encodeWithSelector(
  +         IRemoteHeaderVerifier.updateLatestPmmr.selector,
  +         pmmrSize,
  +         pmmrSnapshot
  +     ),
  +     gasLimit: /* gasLimit */ abi.decode(bridgeMetadata, (uint256)),
  +     refundAddress: msg.sender
  + });
```

yields the following change of contract size:

- **Before**:

Contract	Size (kB)	Margin (kB)
ScrollBroadcastModule	1.867	22.709

- **After**:

Contract	Size (kB)	Margin (kB)
ScrollBroadcastModule	1.829	22.747

Recommendation: While the change in contract size is minimal for each case, consider applying the mentioned refactorings to increase code compacity.

Note: while ArbitrumBroadcastModule.sol#L54-L55 may appear suitable for refactoring by removing the `bytes memory data` local variable, attempting to do so results in a stack-too-deep error during compilation. Therefore, the recommended refactoring is not applicable in this case. Nonetheless, it is advisable to refactor the `inbox.createRetryableTicket` function call so that it uses named parameters.

Axiom: We decided not to make these changes because the gas savings are pretty marginal and we think it reduces readability marginally.

Spearbit: Acknowledged.

5.4.5 Alternative implementation of Polygon broadcasting

Severity: Informational

Context: AxiomV2PolygonHeaderVerifier.sol, PolygonBroadcastModule.sol

Description: There is an alternative implementation of Polygon cross-chain messaging that allows bypassing the step of being manually registered in Polygon's `stateSender` system. This alternative implementation is known as "fx-portal", and specifically includes the `FxRoot` contract on Ethereum mainnet and the `FxChild` contract on Polygon.

These contracts have been mapped in the same way that the `AxiomV2Broadcaster` and `AxiomV2PolygonHeaderVerifier` contracts are intended to be mapped (see this transaction). The implementations of both contracts are shown below:

- FxRoot (on Ethereum mainnet):

```
contract FxRoot is IFxStateSender {
    IStateSender public stateSender;
    address public fxChild;

    constructor(address _stateSender) {
        stateSender = IStateSender(_stateSender);
    }

    function setFxChild(address _fxChild) public {
        require(fxChild == address(0x0));
        fxChild = _fxChild;
    }

    function sendMessageToChild(address _receiver, bytes calldata _data) public override {
        bytes memory data = abi.encode(msg.sender, _receiver, _data);
        stateSender.syncState(fxChild, data);
    }
}
```

- FxChild (on Polygon):

```
contract FxChild is IStateReceiver {
    address public fxRoot;

    event NewFxMessage(address rootMessageSender, address receiver, bytes data);

    function setFxRoot(address _fxRoot) public {
        require(fxRoot == address(0x0));
        fxRoot = _fxRoot;
    }

    function onStateReceive(uint256 stateId, bytes calldata _data) external override {
        require(msg.sender == address(0x0000000000000000000000000000000000001001), "Invalid
↪  sender");
        (address rootMessageSender, address receiver, bytes memory data) = abi.decode(_data,
↪  (address, address, bytes));
        emit NewFxMessage(rootMessageSender, receiver, data);
        IFxMessageProcessor(receiver).processMessageFromRoot(stateId, rootMessageSender, data);
    }
}
```

The contracts are essentially wrappers around the `stateSender` contract. Notably, these contracts are not behind proxies and the state variables can't be changed, introducing no additional trust assumptions compared to using the `stateSender` directly (note that the `stateSender` mappings are centralized, posing a potential risk if the bridge changes the mapping; however, this risk is inherent in the current implementation as well).

A notable example of a protocol adopting the fx-portal is Uniswap. They've set their V3 factory owner to be this contract that's compatible with the system. And here is a governance call to `sendMessageToChild()` on the `FxRoot` (this is what the `PolygonBroadcastModule` would need to do).

Recommendation: Consider exploring this potentially faster alternative implementation of Polygon broadcasting.

Axiom: Implemented in PR 228 and PR 240.

Spearbit: Verified.

5.4.6 Trust assumptions regarding `broadcastModule`

Severity: Informational

Context: BroadcastModule.sol, AxiomV2Broadcaster.sol#L178-L187

Description: To send the PMMR, `AxiomV2Broadcaster._sendPmmr()` performs a `delegatecall` to the `broadcastModule` defined by the `channel`. As a general note, using `delegatecall` has a few more trust assumptions than using `call`. In a worst case scenario, a rogue `broadcastModule` could do the following:

1. Include a `selfdestruct`, which technically returns `success == true` and would delete the `AxiomV2Broadcaster`. However EIP-6780 will make this a non-issue in the next hardfork.
2. Modify critical storage variables in `AxiomV2Broadcaster`. For instance, a malicious `broadcastModule` might alter access controls by updating storage during the `delegatecall`.

Although it's reassuring that `addChannel()` is protected by the same permissions as `_authorizeUpgrade()` (which shares similar capabilities), it's crucial to emphasize that `broadcastModule` must be trusted.

Recommendation: Consider documenting this trust assumption for clarity and future reference.

Axiom: Fixed in PR 235.

Spearbit: Verified.