UMA Across V2 Audit

OPENZEPPELIN SECURITY | MAY 10, 2022

Security Audits

May 11, 2023

This security assessment was prepared by **OpenZeppelin**.

Table of Contents

Summary

Type
: Bridge

Timeline
: From 2023-02-21
: To 2023-03-06

Languages
: Solidity

Total Issues
: 11 (11 resolved)

Critical Severity Issues
: 0 (0 resolved)

High Severity Issues
: 0 (0 resolved)

Medium Severity Issues
: 0 (0 resolved)

Low Severity Issues
: 4 (4 resolved)

Notes & Additional Information
: 7 (7 resolved)

Scope

In scope were the following contracts:

```
contracts
├── BondToken.sol
├── Optimism_SpokePool.sol
├── SpokePool.sol
├── Succinct_SpokePool.sol
├── chain-adapters
│   ├── Optimism_Adapter.sol
│   └── Succinct_Adapter.sol
└── upgradeable
    └── EIP712CrossChainUpgradeable.sol
```

The following contracts were also in scope, but only regarding their upgradeability:

```
contracts
├── Arbitrum_SpokePool.sol
├── Boba_SpokePool.sol
├── Ethereum_SpokePool.sol
├── Ovm_SpokePool.sol
└── Polygon_SpokePool.sol
```

Update: As part of the fix review process, we also reviewed the following pull requests:

- *PR #249: Emit FundsDeposited event directly*
- *PR #264: Fix handling of refunds on other chains and partial fills*
- *PR #267: Fix potential underpayment in slow fill with negative LP fees*
- *PR #268: Apply rounding consistently to partial and full fills*

System Overview

The Across protocol provides a cross-chain bridge acceleration mechanism for token transfers between Layer 2 (L2) chains and the Ethereum mainnet, as well as between L2 chains. Fast cross-

reside on the destination chain to ensure there is sufficient liquidity for relayer refunds, the liquidity providers receive fees in return for adding liquidity to the protocol.

At the core of the protocol are a single HubPool contract and multiple SpokePool contracts. The HubPool is deployed on the Ethereum mainnet, and a single SpokePool is deployed on each supported chain. SpokePools differ slightly from one another to accommodate the specific requirements of different chains.

To accomplish a cross-chain transfer using Across, a user deposits funds on the origin chain and specifies the destination chain where they want to receive the funds. This emits an event which is monitored by the off-chain relayers. A relayer then fills the deposit with their own funds on the destination chain, effectively transferring user funds from origin to destination. The relayer is later reimbursed by the protocol.

To incentivize relayers and liquidity providers, part of the user-deposited funds are set aside in the protocol in the form of the relayer and liquidity provider fees. There is also a protocol fee which is currently set to zero but reserved for future use.

If a deposit is not filled or only partially filled, the protocol uses its own funds to execute a *slow relay* to fill the request.

The protocol implements a pool rebalancing procedure which ensures that SpokePools have enough funds to refund relayers and execute slow fills. Rebalancing also moves excess funds from SpokePools to the HubPool.

Relayer refunds, slow relays, and pool rebalances are accomplished by the proposal of a root bundle to the Optimistic Oracle by a *dataworker*. The root bundle is a Merkle root constructed in accordance with the UMIP specification(s). Unless disputed, the root bundle is optimistically accepted after a liveness period has passed.

When the root bundle is executed, the HubPool sends a cross-chain message to SpokePools by providing Merkle tree roots of relayer refunds and slow relays along with tokens. When it comes to actually executing relayer refunds, a Merkle proof is provided and verified on the SpokePool's side.

Summary of Changes

- SpokePools have become upgradeable contracts.
- Support for chains without the canonical bridge infrastructure has been added. This is achieved by utilizing the Telepathy protocol of Succinct. Since it is not possible to transfer tokens to Succinct SpokePools, negative fees have been introduced to incentivize users to move funds from high-liquidity SpokePools to low-liquidity ones. A negative fee translates to a reward for a depositor to use the Across protocol. As a counterbalance, the protocol compensates relayers that fill such requests. The computation of these relayer rewards is performed off-chain and integrated into the refund proposal data.
- A custom BondToken has been introduced, which enables the whitelisting of bundle proposers. Disputes, however, can still be submitted by anyone.
- Support for EIP-1271 and EIP-712 signatures has been added. EIP-1271 in particular allows users to invalidate a deposit update after submitting it.
- A non-canonical bridge is now used to transfer SNX tokens on Optimism.
- Pause functionality has been added for deposits and fills.

Security Model & Trust Assumptions

The Across protocol relies on the Optimistic Oracle together with the UMIP specification(s), canonical and non-canonical bridges, and the off-chain software consisting of the relayer and dataworker implementation.

The Optimistic Oracle, which is controlled by UMA token holders, has the ability to resolve disputes on root bundles. This means that if it is compromised, it is possible for disputes to not resolve correctly, and, more importantly, whoever can control the Optimistic Oracle can decide how funds are moved within the system. This is a feature of the greater UMA ecosystem, and incentives exist to keep the Optimistic Oracle honest.

The UMIP specification(s) describe what constitutes a valid root bundle and are a critical component. A loophole or inconsistency in them may lead to consequences comparable to smart contract vulnerabilities. Even if specifications are flawless, there is still a risk that a bug in the off-chain software, which mainly refers to relayer and dataworker implementations, may be abused to achieve similar effects.

There is one administrator for the entire Across Protocol system. This administrator can make decisions regarding which chains have valid SpokePools, which tokens are enabled, and how tokens on one chain map to tokens on another. The administrator also controls parameters such as the system fee percentage, where fees are directed, the bond amount for proposing new root bundles, how disputed root bundles are identified, and which tokens are allowed within the system. Finally, they can cancel or delete root bundles and pause deposits and fills.

With the addition of BondToken, the administrator also manages the proposers' whitelist.

Client-Reported Findings

Incorrect handling of refunds taken on other chains

Client reported: *UMA identified this issue after the audit.*

The `fillCounter` is intended to provide frontrunning protection by tracking anticipated decreases to a SpokePool's balance. For example, in the simple case where a relayer executes a full fill for a 100 token deposit, the chain's `fillCounter` is incremented by 100. If the relayer chooses to take their refund on the same destination chain, depositors can precompute their deposit incentive fees based on the knowledge that 100 tokens will be removed from this SpokePool once the relayer is refunded. Since the pool's running balance has decreased, the incentive to deposit has increased in a predictable way. The logic also works correctly if this relayer were to instead perform a partial fill and a slow relay completes the fill--the net result is still that the SpokePool balance decreases by 100.

However, the logic does not take into account that relayers can request to take their refund on a different chain, i.e. `repaymentChainId != destinationChainId`. Applied to the above example, the `fillCounter` on the destination chain's SpokePool will be increased even though its balance will not change. Likewise, the repayment chain's SpokePool will not be updated to reflect a change in its running balance. As a result, depositors on the repayment chain can be frontrun by relayers in this scenario because the fill counters are not being updated correctly.

relayers can submit subsequent partial fills with rebates on different chains.

Update: *Resolved in pull request #264 at commit ef59e2a.*

Potential underpayment in slow fill with negative LP fees

Client reported: *UMA identified this issue after the audit.*

In the case of a slow relay on a deposit with a negative LP fee, the `maxSendAmount` passed into the `_fillRelay` function could cause `fillAmountPreFees` to be less than the amount, which means it could be less than `amountRemainingInRelay`. In this case, two errors occur: a) the slow fill does not complete the fill, resulting in underpayment of the recipient, and b) the `payoutAdjustmentPct` is ignored, because it is inside the `if` statement comparing `fillAmountPreFees` to `amountRemainingInRelay`.

Update: *Resolved in pull request #267 at commit 01e6ec4.*

Change recipient/message feature does not work

Client reported: *UMA identified this issue after the audit.*

There is a feature that enables changing the recipient and the message. The feature was not working, so if someone deposited funds into an address that was unable to receive them, the funds would become stuck. The bug only occurs if a user makes a mistake - the attacker would not be able exploit this at will to steal money.

Update: *Resolved in pull request #276.*

OpenZeppelin-Reported Findings (after the audit)

Stealing ETH from the dataworker

OpenZeppelin reported: *OpenZeppelin identified this issue after the audit.*

The attacker can create a deposit with a recepient and a message that will not be filled by relayers (e.g. because the relayer fee is set to zero). Eventually, the slow fill process will be triggered. In the

Update: *Resolved in pull request #647.*

Incorrect or misleading documentation

Several instances of docstrings or comments in the codebase were found to be erroneous. In particular:

- In the `SpokePool` contract's `_fillRelay` function, the comment on line 1003 appears unrelated to the `if` statement on the following line. Consider revising the comment.
- The docstring for `pauseDeposits` in the `SpokePool` contract says it "pauses deposit and fill functions", but pausing fill functions is performed by calling `pauseFills`, which does not have a docstring.

Consider revising the `pauseDeposits` docstring and adding a docstring to `pauseFills`.

Update: *Resolved in pull request #251 at commit 21d9ed8.*

Lack of event emission after sensitive action

The `setSuccinctTargetAmb` administrative function does not emit a relevant event after changing the `succinctTargetAmb` address.

Consider always emitting events after sensitive changes take place to facilitate tracking and notify off-chain clients that follow the protocol's contracts' activity.

Update: *Resolved in pull request #252 at commit 10c1190. Additionally, new* `ReceivedMessageFromL1` *events were added to the* `Polygon_SpokePool` *and* `Succinct_SpokePool`*'s message handler functions (*`processMessageFromRoot` *and* `handleTelepathy`*, respectively).*

Missing docstrings

Throughout the codebase, there are several parts that do not have docstrings. In particular:

- `AdapterInterface.sol`, lines 13-20: The interface functions are undocumented.
- `Arbitrum_Adapter.sol`, lines 9-54: The `ArbitrumL1InboxLike` and `ArbitrumL1ERC20GatewayLike` interfaces and their functions are undocumented.

interface and its function are undocumented.

- `Polygon_Adapter.sol`, lines 10-30: The `IRootChainManager`, `IFxStateSender`, and `DepositManager` interfaces and their functions are undocumented.
- `Polygon_SpokePool.sol`, line 11: The `processMessageFromRoot` function is undocumented.
- `SpokePool.sol`, lines 169-196: The `RelayExecution`, `RelayExecutionInfo`, and `DepositUpdate` structs have undocumented members.
- `Succinct_SpokePool.sol`, line 39: The `initialize` function has no docstring.
- `Succinct_SpokePool.sol`, line 57: The `handleTelepathy` function has no docstring.
- `WETH9Interface.sol`, lines 4-11: The interface and its functions are undocumented.
- `ZkSync_Adapter.sol`, lines 13-30: The `ZkSyncLike` and `ZkBridgeLike` interfaces and their functions are undocumented.
- `ZkSync_SpokePool.sol`, lines 6-12: The `ZkBridgeLike` interface and its function are undocumented.

Consider thoroughly documenting all functions (and their parameters) that are part of any contract's public API. Functions implementing sensitive functionality, even if not public, should be clearly documented as well. When writing docstrings, consider following the Ethereum Natural Specification Format (NatSpec).

Update: *Resolved in pull request #253 at commit 5d1090a, pull request #242 at commit 4a8fe1c, and pull request #269 at commit 146f2f2.*

The `ZkSync_Adapter.sol` and `ZkSync_SpokePool.sol` contracts were not changed. The UMA team stated:

> *All suggested comments are implemented except for zkSync contracts which were out of this audit's scope and are still in progress and not live.*

`updatedRelayerFeePct` can be lower than expected

```
require(updatedRelayerFeePct < 0.5e18, "invalid relayer fee");
```

Unlike the equivalent check in the `deposit` function, in this case the `SignedMath.abs` function is not used, which allows the `updatedRelayerFeePct` value to reach or go below the lower limit of `-0.5e18`. An attempt by the relayer to fill this request will be rejected by the `_fillRelay` function's fee check.

Consider using the `SignedMath.abs` function to ensure the `updatedRelayerFeePct` argument provided to `speedUpDeposit` is within the expected limits.

Update: *Resolved in pull request #254 at commit 857e787.*

Notes & Additional Information

Interface files not in `interfaces` directory

The Across Protocol V2 codebase contains the `interfaces` directory, but the `HubPoolInterface.sol` and `SpokePoolInterface.sol` files do not reside in that location. Instead, they are located in the parent directory.

Consider moving all non-external interface files to the `interfaces` directory, so that interfaces can be more easily located.

Update: *Resolved in pull request #255 at commit fa59467.*

Multiple conditions in a single `require` statement

There is a `require` statement with multiple conditions in the `handleTelepathy` function of the `Succinct_SpokePool` contract.

Consider isolating each condition in its own separate require statement with a corresponding error message, to more easily identify the specific condition that failed.

Update: *Resolved in pull request #256 at commit c42ea4f.*

updating the `fillCounter` data when a fill takes place. No update is necessary in the following three cases: when the action is either a slow fill or an initial zero-fill, or when a partial fill for that request has already happened.

However, in the case of a zero-fill the execution flow will not reach this function, which makes the condition `endingFillAmount == 0` redundant.

Consider removing the redundant condition for increased clarity, readability, and gas savings. For extra safety, consider adding a comment where the function returns if there is a zero-fill, denoting that any updates to this part should take the function `_updateCountFromFill` into account.

***Update:** Resolved in pull request #264 at commit ef59e2a. The `endingFillAmount == 0` condition has been removed and a comment has been added to clarify that initial zero-fills will not reach this location in the code.*

Typographical errors

Consider correcting the following typographical errors:

- In `BondToken.sol`:
  - Line 11, "rootBundleProposer()" should be "rootBundleProposal()".
  - Line 21: "permissiong" should be "permissioning".
- In `EIP712CrossChainUpgradeable.sol`:
  - Line 83: "its always" should be "it's always".
- In `MultiCallerUpgradeable.sol`:
  - Line 5: "@title MockSpokePool" should be "@title MultiCallerUpgradeable".
- In `Optimism_Adapter.sol`:
  - Line 23: "its only" should be "it's only".
- In `Polygon_SpokePool.sol`:
  - Line 51: the sentence "See" is incomplete.
  - Line 139: the sentence is malformed.

    - Line 1177: "its always" should be "it's always".
- In `Succinct_Adapter.sol`:
    - Line 27: "destinatipn" should be "destination".
    - Line 27: "the message.." should be "the message.".
- In `Succinct_SpokePool.sol`:
    - Line 22: "callValidated" should be "adminCallValidated".
    - Line 24: "callValidated" should be "adminCallValidated".
    - Line 35: "callValidated" should be "adminCallValidated".

Update: *Resolved in pull request #258 at commit 43a0ea7.*

Unnecessary modulo operation

The `setClaimed1D` function in `MerkleLib.sol` performs a modulo 256 operation on `index` to yield a result in the range [0, 255]. However, `index` is a `uint8` variable, so the operation `index % 256` is equivalent to `index`.

Consider removing the redundant modulo operation.

Update: *Resolved in pull request #259 at commit 04c8488.*

Unused imports

The following imports are unused:

- Import `AdapterInterface.sol` in `HubPoolInterface.sol`
- Import `ECDSA.sol` in `SpokePool.sol`

To improve readability and avoid confusion, consider removing any unused imports.

Update: *Resolved in pull request #260 at commit 4a94713.*

Use of hard-coded values

Consider creating constants to store these values, and assigning them descriptive variable names.

Update: *Resolved in pull request #261 at commit 6f8b6e3.*

Several low-severity issues were identified, as well as some notes to increase the quality of the codebase. Some changes were proposed to ensure smart contract security best practices are followed. The Across protocol, however, heavily relies on relayer and dataworker UMIP(s), their off-chain implementations, and the Optimistic Oracle. As described in the Security Model and Trust Assumptions section, a bug in any of these may lead to a severe loss of funds, and thus a full-system security audit is recommended.

Upgradability Considerations

The following are additions to the process described in the Across draft SpokePool upgrade process to ensure a safe migration.

The migration procedure from non-upgradeable to upgradeable SpokePools should ensure that no funds are lost or locked in the process. Before the migration, it is recommended to ensure that:

- All token routes on the chain where the SpokePool resides are disabled.
- All outstanding deposits on the chain where the SpokePool resides are filled.
- All refund leaves are executed.
- All slow-relay leaves are executed.
- Funds that are left in the SpokePool are returned to the HubPool.
- The root bundle is empty.
- There is no root bundle in transit (i.e., the cross-chain message was sent but not yet executed).

Monitoring Recommendations

While audits help in identifying potential security risks, the UMA team is encouraged to also incorporate automated monitoring of on-chain contract activity into their operations. Ongoing monitoring of deployed contracts helps in identifying potential threats and issues affecting the production environment.

To ensure no unexpected administrative actions are occurring, and to validate that correct values were used, consider monitoring all `SpokePool` administrative events, i.e. events emitted from functions with the `onlyAdmin` or `onlyOwner` modifier. In particular, consider monitoring:

- The occurrence of `PausedDeposits`, `PausedFills`, and `EnabledDepositRoute` events to ensure that a pool is not unintentionally placed in a partially disabled state.

- The addresses and other data emitted in the `SetHubPool`, `SetXDomainAdmin`, `EnabledDepositRoute`, `SetDepositQuoteTimeBuffer`, and `EmergencyDeleteRootBundle` events to validate they match the input values.

Bridge servers experiencing downtime may disrupt the operation of the Across protocol. Consider monitoring the up/down status of bridge nodes, e.g. the availability of the Arbitrum Sequencer.

Consider monitoring for abnormally large or small deposit and fill values, which may indicate an ongoing attack.

Related Posts

Beefy Zap Audit

BeefyZapRouter serves as a versatile intermediary designed to execute users' orders through routes...

Security Audits

OpenBrush Contracts Library Security Review

OpenBrush is an open-source smart contract library written in the Rust programming language and the...

Security Audits

Linea Bridge Audit

Linea is a ZK-rollup deployed on top of Ethereum. It is designed to be EVM-compatible and aims to...

Security Audits

OpenZeppelin