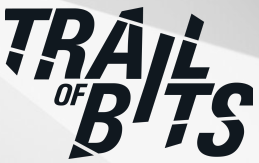

Uniswap MMA

Security Assessment (Summary Report)

November 13, 2023

Prepared for:
Erin Koen
Uniswap Foundation

Prepared by: **Alexander Remie and Nat Chin**

About Trail of Bits

Founded in 2012 and headquartered in New York, Trail of Bits provides technical security assessment and advisory services to some of the world's most targeted organizations. We combine high-end security research with a real-world attacker mentality to reduce risk and fortify code. With 100+ employees around the globe, we've helped secure critical software elements that support billions of end users, including Kubernetes and the Linux kernel.

We maintain an exhaustive list of publications at https://github.com/trailofbits/publications, with links to papers, presentations, public audit reports, and podcast appearances.

In recent years, Trail of Bits consultants have showcased cutting-edge research through presentations at CanSecWest, HCSS, Devcon, Empire Hacking, GrrCon, LangSec, NorthSec, the O'Reilly Security Conference, PyCon, REcon, Security BSides, and SummerCon.

We specialize in software testing and code review projects, supporting client organizations in the technology, defense, and finance industries, as well as government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, and Zoom.

Trail of Bits also operates a center of excellence with regard to blockchain security. Notable projects include audits of Algorand, Bitcoin SV, Chainlink, Compound, Ethereum 2.0, MakerDAO, Matic, Uniswap, Web3, and Zcash.

To keep up to date with our latest news and announcements, please follow @trailofbits on Twitter and explore our public repositories at https://github.com/trailofbits. To engage us directly, visit our "Contact" page at https://www.trailofbits.com/contact, or email us at info@trailofbits.com.

Trail of Bits, Inc.
228 Park Ave S #80688
New York, NY 10003
https://www.trailofbits.com
info@trailofbits.com

Notices and Remarks

Copyright and Distribution

Test Coverage Disclaimer

All activities undertaken by Trail of Bits in association with this project were performed in accordance with a statement of work and agreed upon project plan.

Security assessment projects are time-boxed and often reliant on information that may be provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

Trail of Bits uses automated testing techniques to rapidly test the controls and security properties of software. These techniques augment our manual security review work, but each has its limitations: for example, a tool may not generate a random edge case that violates a property or may not fully complete its analysis during the allotted time. Their use is also limited by the time and resource constraints of a project.

Table of Contents

Project Summary

Contact Information

The following project manager was associated with this project:

Sam Greenup, Project Manager
sam.greenup@trailofbits.com

The following engineering director was associated with this project:

Josselin Feist, Engineering Director, Blockchain
josselin.feist@trailofbits.com

The following consultants were associated with this project:

Alexander Remie, Consultant
alexander.remie@trailofbits.com

Nat Chin, Consultant
natalie.chin@trailofbits.com

Project Timeline

The significant events and milestones of the project are listed below.

Date	Event
September 25, 2023	Pre-project kickoff call
October 2, 2023	Delivery of report draft and report readout meeting
October 13, 2023	Delivery of summary report
October 18, 2023	Fix review of the Uniswap MMA issues
November 13, 2023	Delivery of summary report with fix review

Executive Summary

Engagement Overview

Uniswap engaged Trail of Bits to review the security of the cross-chain MMA multibridge on commit hash `dd0ea1e`.

A team of two consultants conducted the review from September 25 to September 29, 2023, for a total of two engineer-weeks of effort. With full access to source code and documentation, we performed static and dynamic testing of the codebase, using automated and manual processes.

Observations and Impact

During this review, we focused on all the contracts in the repository. This includes the two receiver and sender contracts and those contracts specific to the Wormhole and Axelar bridges. We did not consider the actual bridges or any off-chain components outside the contracts to be in scope.

We identified several findings related to lack of data validation that stemmed from not properly considering return values on functions, which would result in failing actions not being detected. We discovered another issue related to the incorrect order of operations, which prevents functions from executing correctly. We also found issues where events were missing from owner-controlled functions and assumptions were made around the "successful" execution of a proposal.

Recommendations

We recommend that the Uniswap MMA team take the following steps:

- Address the findings identified during this audit.
- Identify system invariants that are expected to hold on bridges and test them using property-based testing (e.g., Echidna). This will help keep the implementation of all bridge adapters working as expected as more bridge adapters are added.
- Simplify the implementation according to the recommendations outlined in appendix E. This will improve the overall readability of the code.
- Improve the documentation according to the recommendations outlined in appendix D.
- Perform another audit of new adapters whenever they are developed. The Uniswap MMA team intends to add more bridge adapters in the future. We recommend that each of these be reviewed before being used in production.

Codebase Maturity Evaluation

Trail of Bits uses a traffic-light protocol to provide each client with a clear understanding of the areas in which its codebase is mature, immature, or underdeveloped. Deficiencies identified here often stem from root causes within the software development life cycle that should be addressed through standardization measures (e.g., the use of common libraries, functions, or frameworks) or training and awareness programs.

Category	Summary	Result
Arithmetic	The Uniswap multibridge contracts use Solidity v0.8.9, which has built-in overflow and underflow protections. Arithmetic calculations in this codebase are kept to a minimum. The majority of the unchecked blocks in the codebase are used for looping and iterating over arrays.	**Satisfactory**
Auditing	Most of the functions correctly emit events that aid in the off-chain monitoring of the system. However, we identified two configuration functions that are missing an event emission. There is an elaborate incident response plan in the user-facing documentation. We recommend performing regular test runs of the various scenarios outlined in the plan to learn how to handle such incidents and to uncover any gaps or otherwise incorrect or sub-optimal steps in the plan. We also recommend looking into an off-chain monitoring system to track events. Create a monitoring plan that describes which events and values to detect and how to react to such events.	**Moderate**
Authentication / Access Controls	All functions have appropriate access controls. The user-facing documentation includes an overview of the access controls on some of the contracts. We recommend extending this by also including the access controls on the actual adapter contracts (as we did in appendix B).	**Satisfactory**
Complexity Management	The overall complexity of the protocol is low. All the functions perform a single task and are easy to understand. However, we did find numerous areas where code quality improvements would further simplify the	**Satisfactory**

	implementation (appendix E). The user-facing documentation is very helpful for understanding the protocol.	
Configuration	No deployment scripts were provided.	**Further Investigation Required**
Decentralization	The MMA protocol is controlled by Uniswap Governance and, as such, is decentralized.	**Satisfactory**
Documentation	The Uniswap MMA team provided significant documentation outlining the system's expected behavior. While we have some minor recommendations for areas that would benefit from clarification (see appendix D), we recommend that the Uniswap MMA team continue to document system invariants expected to hold on each bridge as more bridges are added. Additionally, some of the files are missing NatSpec comments (e.g., `MessageSenderGAC`, `MessageReceiverGAC`, and `StringAddressConversion`), and some functions' NatSpec comments do not include the function arguments (e.g., `updateQuorumAndReceiverAdapter`).	**Satisfactory**
Low-Level Manipulation	The codebase uses limited assembly. We did identify instances where low-level calls were lacking contract size checks, which could result in code not executing despite a "successful" transaction; however, this was considered expected behavior.	**Satisfactory**
Testing and Verification	The contracts have adequate tests on functionality. However, we recommend spending additional effort documenting system invariants and assumptions on all bridges. We also recommend that all tests check internal state changes before and after execution. Additionally, we recommend including information in the repository `README` to explain how to execute the tests locally. Lastly, we recommend using Foundry fuzzing tests. These tests might have uncovered at least one of the issues found during this audit and, in general, are a very useful enhancement to regular unit tests.	**Moderate**

Transaction Ordering	We did not identify any front-running vulnerabilities that affect the system.	**Satisfactory**

A. Code Maturity Categories

The following tables describe the code maturity categories and rating criteria used in this document.

Code Maturity Categories	
Category	**Description**
Arithmetic	The proper use of mathematical operations and semantics
Auditing	The use of event auditing and logging to support monitoring
Authentication / Access Controls	The use of robust access controls to handle identification and authorization and to ensure safe interactions with the system
Complexity Management	The presence of clear structures designed to manage system complexity, including the separation of system logic into clearly defined functions
Cryptography and Key Management	The safe use of cryptographic primitives and functions, along with the presence of robust mechanisms for key generation and distribution
Decentralization	The presence of a decentralized governance structure for mitigating insider threats and managing risks posed by contract upgrades
Documentation	The presence of comprehensive and readable codebase documentation
Low-Level Manipulation	The justified use of inline assembly and low-level calls
Testing and Verification	The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage
Transaction ordering risks	The system's resistance to front-running attacks

Rating Criteria	
Rating	**Description**
Strong	No issues were found, and the system exceeds industry standards.
Satisfactory	Minor issues were found, but the system is compliant with best practices.
Moderate	Some issues that may affect system safety were found.

Weak	Many issues that affect system safety were found.
Missing	A required component is missing, significantly affecting system safety.
Not Applicable	The category is not applicable to this review.
Not Considered	The category was not considered in this review.
Further Investigation Required	Further investigation is required to reach a meaningful conclusion.

B. Role Privileges

This appendix outlines the expected caller for each of the functions within the listed contracts.

MultiBridgeMessageSender

Function	Intended caller
`remoteCall`	GAC Authorized Caller
`remoteCall` with additional `excludedAdapters` argument	GAC Authorized Caller
`addSenderAdapters`	GAC Global Owner
`removeSenderAdapters`	GAC Global Owner

According to the documentation, the GAC Global Owner is intended to be set to the Uniswap `Timelock` contract. We use the terminology and role names used in the Uniswap MMA smart contract in this table.

MultiBridgeMessageReceiver

Function	Intended caller
`receiveMessage`	Receiver adapter
`executeMessage`	Anyone
`updateGovernanceTimelock`	GAC Global Owner
`updateReceiverAdapters`	GAC Global Owner
`updateQuorumAndReceiverAdapter`	GAC Global Owner
`updateQuorum`	GAC Global Owner

According to the documentation, the owner of these contracts should be set to the Uniswap `Timelock` contract on the destination chain. The receiver adapter is expected to be any address in the “authorized receiver adapters” set that the admins approve.

BaseReceiverAdapter

Function	Intended caller
`updateSenderAdapter`	GAC Global Owner

AxelarSenderAdapter

Function	Intended caller
`dispatchMessage`	Multibridge message receiver
`setChainIdMap`	GAC Global Owner

C. Axelar vs. Wormhole Adapter Differences

There are minor deviations between the Axelar Adapter and the Wormhole Adapter. The following table identifies the differences between these two adapter implementations.

Behavior	Axelar Adapter	Wormhole Adapter
Entry point for cross-chain messaging	Uses the `execute` function	Uses the `receiveWormholeMessages` function
`msg.sender` check	No validation	Checks that `msg.sender` is the relayer contract
Validates incoming chain ID	Hashes using Keccak-256 due to incoming argument being a `string`	Compares `bytes` directly due to incoming argument being `bytes32`
Validates that source address is not `senderAdapter`	Converts `string => address` for comparison	Converts `bytes => address` for comparison
Validates contract call	Calls `AxelarGateway.validateContractCall()`	No validation
Validates duplicate messages	Checks that `commandIdStatus[commandId]` and `isMessageExecuted[msgId]` flags are set	Checks whether `isMessageExecuted[msgId]` or `deliveryHashStatus[deliveryHash]` flags are set
Validates receiver adapter	Checks that the `payload.receiverAdapter` is set to `address(this)`	Checks that the `payload.receiverAdapter` is set to `address(this)`
Validates destination address	Checks that `payload.finishDestination` is `multiBridgeMsgReceiver`	Checks that `payload.finishDestination` is set to `multiBridgeMsgReceiver`

D. Documentation Improvements

The Uniswap team provided in-depth documentation outlining the system's expected behavior. During our review, we identified areas that would benefit from additional elaboration and documentation of known assumptions and risks. This appendix details our recommendations in these areas.

Elaborate Current Documentation

- **Change the documentation on the `MessageExecuted` event to specify that the event is emitted when the transaction is scheduled for execution.** The name of the event suggests that the transaction has already been executed, which is misleading.
- **Add instructions for how to run the test suite for the codebase.** The test suite relies on API keys to be set in the .env file. We recommend adding instructions for how to run these tests.
- **Document all design decisions in the codebase.** This will help provide context for the product and detail the reasons for specific decisions.

Document Known Assumptions

- **Contracts do not impose restrictions on whether the target address where code is executed is a contract.** This is a known assumption made by the Uniswap team about external calls because these low-level calls may return "success" despite not having code run. The success of the transaction execution is inferred by the return value of the low-level call.

E. Code Quality Recommendations

- **Consider simplifying the adapter data structure.** Currently, the adapters being added to the infrastructure are submitted in a list of addresses, which must be ordered. This adds complexity to the codebase when adapters are added or deleted by requiring the code to iterate over the entire list of adapters. While this behavior may not be very costly for two to three adapters, the level of complexity may increase as more adapters are added to the system.

- **Fix the NatSpec comment on `src/libraries/Message.sol#L14` to remove the mention of "zero."** When this value is set in `MultiBridgeMessageSender._remoteCall#L281`, it uses `block.timestamp + expiration`, which means a zero timestamp is not realistic.

- **Remove the version of the `remoteCall` function without the `_excludedAdapters` argument.** The system currently implements two separate versions of `remoteCall`, where the only difference is the `address[] calldata _excludedAdapters` argument. Instead of introducing two separate functions, allow the caller to specify an empty adapters list if they do not want to exclude adapters.

- **Use the `onlyGlobalOwners` modifier consistently across the codebase.** Currently, some contracts use `onlyOwner` in place of `onlyGlobalOwner`. We recommend adding specificity in these naming conventions.

- **Remove the `BaseSenderAdapter.getReceiverAdapter` function since the `receiverAdapters` mapping is already a public variable.** By default, Solidity provides getters for public variables, which makes this code redundant.

- **Change the `multiBridgeMsgReceiver` variable to `public` and remove the `MessageReceiverGAC.getMultiBridgeMessageReceiver()` function.** Solidity can provide these getters by default for public variables.

- **Simplify the `isGlobalOwner` modifier to `return _caller == owner()` instead of using an `if` statement.** This will enhance code readability.

- **Add an underscore to the function argument for `MessageSenderGAC.setAuthorizedCaller(address newMMSCaller)`.** The codebase introduces underscores for all other incoming variables and should be kept consistent.

- **Emit the old value for the `MessageSenderGAC.setMultiBridgeSender` and `MessageSenderGAC.setAuthorisedCaller` functions.** All other configuration functions emit the old value and the new value.

- **Remove the `MessageSenderGAC` contract's `getGlobalMsgDeliveryGasLimit`, `getMultiBridgeMessageSender`, `getRemoteMultiBridgeMessageReceiver`, and `getAuthorisedCaller` functions.** These functions return values of public variables that already have getters by default.

- **Add a `DelayUpdated` event to the `GovernanceTimelock` contract's constructor.** The contracts emit the `AdminUpdated` event when the admin address is updated and should do the same for the delay being updated.

- **Move the `ExecutorAware` contract out of the `src/interfaces/` directory since the contract is not an interface.** This directory should hold only interfaces.

- **Rename the `MessageExecuted` event to `MessageScheduledForExecution`.** The event is emitted only when scheduled for the timelock, not when the message occurs.

- **Change the visibility of the `WormholeSenderAdapter.relayer` variable from private to public.** The `WormholeReceiverAdapter.relayer` is public.

- **Add a zero-value check to the `WormholeSenderAdapter` constructor to ensure that the `relayer` cannot be `address(0)`.** Most other functions have a zero-value check, but it is missing in this instance.

- **Standardize the capitalization of names throughout the codebase to ensure that both receivers and adapters use the same spelling.**

    - The `WormholeSenderAdapter` contract uses `wormhole`, while the `WormholeReceiverAdapter` contract uses `WORMHOLE`.

    - Similarly, the `AxelarSenderAdapter` contract uses `axelar`, while the `AxelarReceiverAdapter` contract uses `AXELAR`.

- **Rename the `senderChain` variable to `senderChainId` and the `_toChainId` variable to `receiverChain` in the adapter contracts.** This will standardize the references to chain IDs and make the code more readable.

- **Add a zero-value check for the `_gasService`, `gateway`, and `_gac` arguments provided to the `AxelarSenderAdapter.constructor`.** Almost all other functions check to ensure that addresses sent are nonzero.

- **Compare a `uint` against zero using `== 0` instead of `<= 0`.** Checking the case where a `uint` is less than zero is not meaningful.
- **Remove the return value of the adapter's `dispatchMessage` function.** The return value of the `dispatchMessage` function is not used by the `MultiBridgeMessageSender` contract.

F. Invariant Testing

This section describes system invariants that are worth implementing and testing using Echidna, Medusa, or Foundry invariant testing. Due to the time constraints of this audit, we were unable to implement these invariants.

Adapter List

- **Adding an adapter should always increase the length of the adapter list.** After an adapter is added, the `senderAdapters` array length should be one greater than the pre-execution run.
- **Removing an adapter should always decrease the length of the adapter list.** After an adapter is removed, the `senderAdapters` array length should be one less than the pre-execution run.
- **Once a trusted executor is removed, it should not persist in the adapter list.**
- **Once a trusted executor is added, its entry should be in the adapter list.**
- **The adapter list should never contain duplicates.** This is an assumption made throughout the code.
- **The adapter list should always be in order.** The code contains logic to ensure that the list is ordered properly.

Access Controls

- **Only authorized callers can execute calls.** Any calls not made by external callers should fail.
- **The `Timelock` contract should be the only address allowed to update the quorum and `timelock` variables.** Any calls not made by this address should revert.
- **Only the GAC Global Owner should be able to add an adapter to the allowed list.** Any calls not made by the GAC Global Owner should revert.

Wormhole Adapter

- **Receiving a message always sets the `isMessageExecuted` and `deliveryHashStatus` mapping entries to `true` for the message.** Receiving messages should be registered correctly.

Axelar Adapter

- **Receiving a message always sets the `isMessageExecuted` and `commandIdStatus` mapping entries to `true` for the message.** Receiving messages should be registered correctly.

G. Fix Review Results

When undertaking a fix review, Trail of Bits reviews the fixes implemented for issues identified in the original report. This work involves a review of specific areas of the source code and system configuration, not comprehensive analysis of the system.

On October 18, 2023, Trail of Bits reviewed the fixes and mitigations implemented by the Uniswap MMA team for the issues identified in this report. We reviewed each fix to determine its effectiveness in resolving the associated issue.

In summary, of the four issues described in this report, the Uniswap MMA team has resolved all four issues. The current scope of the review also included fixes for the identified Code Quality Recommendations. For additional information, please see the Detailed Fix Review Results and Code Quality Fix Review Results below.

ID	Title	Status
1	remoteCall does not revert if dispatchMessage fails for all the AMBs	**Resolved**
2	_updateReceiverAdapter does not check Boolean return value of EnumerableSet operations	**Resolved**
3	updateQuorumAndReceiverAdapter can unexpectedly revert when increasing or decreasing both the quorum and trusted executors	**Resolved**
4	Missing events in critical operations	**Resolved**

Detailed Fix Review Results

TOB-MMA-1: remoteCall does not revert if dispatchMessage fails for all the AMBs
Resolved in PR #97. The fix introduces a new variable that specifies a minimum quorum of remote chains required for the call to be successful. If the total quorum of all bridges is less than the threshold, the `remoteCall` function reverts.

TOB-MMA-2: _updateReceiverAdapter does not check Boolean return value of EnumerableSet operations
Resolved in PR #102. The fix checks the success value from both the `addTrustedExecutor` and `removeTrustedExecutor` functions. If one of them fails, it reverts with the expected error.

TOB-MMA-3: updateQuorumAndReceiverAdapter can unexpectedly revert when increasing or decreasing both the quorum and trusted executors
Resolved in PR #99. The fix now supports the simultaneous changing of receiver adapter and quorum values.

TOB-MMA-4: Missing events in critical operations
Resolved in PR #92. The fix adds events for the `GovernanceTimelock` contract and the chain ID mapping.

Code Quality Fix Review Results

See the following list for fixes that address the Code Quality Recommendations identified in this report. Resolved issues were fixed in PR #96 unless otherwise noted.

- **Consider simplifying the adapter data structure.**
  Unresolved. The Uniswap MMA team stated the following:

  > *The current design requires that we retain the use of arrays for this purpose because we need the caller to be able to predict the ordered set of bridges that will be used when dispatching a message, taking into account an exclusion list. This is necessary for a caller to know so they can specify a list of fees for each corresponding bridge when sending a message. Although we understand that this may add cost, we believe that the number of bridge adapters will remain small, so we do not expect this to be an issue. This may be revised in the future.*

- **Fix the NatSpec comment on `src/libraries/Message.sol#L14` to remove the mention of "zero."**
  Resolved.

- **Remove the version of the `remoteCall` function without the `_excludedAdapters` argument.**
  Resolved.

- **Use the `onlyGlobalOwner` modifier consistently across the codebase.**
  Resolved.

- **Remove the `BaseSenderAdapter.getReceiverAdapter` function since the `receiverAdapters` mapping is already a public variable.**
  Resolved.

- **Change the `multiBridgeMsgReceiver` variable to `public` and remove the `MessageReceiverGAC.getMultiBridgeMessageReceiver()` function.**
  Resolved.

- **Simplify the `isGlobalOwner` modifier to `return _caller == owner()` instead of using an `if` statement.**
  Resolved.

- **Add an underscore to the function argument for `MessageSenderGAC.setAuthorizedCaller(address newMMSCaller)`.**
  Resolved.

- **Emit the old value for the `MessageSenderGAC.setMultiBridgeSender` and `MessageSenderGAC.setAuthorisedCaller` functions.**
  Resolved.

- **Remove the `MessageSenderGAC` contract's `getGlobalMsgDeliveryGasLimit`, `getMultiBridgeMessageSender`, `getRemoteMultiBridgeMessageReceiver`, and `getAuthorisedCaller` functions.**
  Resolved.

- **Add a `DelayUpdated` event to the `GovernanceTimelock` contract's constructor.**
  Resolved.

- **Move the `ExecutorAware` contract out of the `src/interfaces/` directory since the contract is not an interface.**
  Resolved in commit 1b19216.

- **Rename the `MessageExecuted` event to `MessageScheduledForExecution`.**
  Resolved in PR #100.

- **Change the visibility of the `WormholeSenderAdapter.relayer` variable from private to public.**
  Resolved.

- **Add a zero-value check to the `WormholeSenderAdapter` constructor to ensure that the `relayer` cannot be `address(0)`.**
  Resolved in PR #98.

- **Standardize the capitalization of names throughout the codebase to ensure that both receivers and adapters use the same spelling.**

  - The `WormholeSenderAdapter` contract uses `wormhole`, while the `WormholeReceiverAdapter` contract uses `WORMHOLE`.
    Resolved.

  - Similarly, the `AxelarSenderAdapter` contract uses `axelar`, while the `AxelarReceiverAdapter` contract uses `AXELAR`.
    Resolved.

- **Rename the `senderChain` variable to `senderChainId` and the `_toChainId` variable to `receiverChain` in the adapter contracts.**
  Resolved in lines 30–31 and 38–41.

- **Add a zero-value check for the `_gasService`, `gateway`, and `_gac` arguments provided to the `AxelarSenderAdapter.constructor`.**
  Unresolved.

- **Compare a `uint` against zero using `== 0` instead of `<= 0`.**
  Resolved.

- **Remove the return value of the adapter's `dispatchMessage` function.**
  Unresolved. The Uniswap MMA team stated the following:

  > *One recommendation that we have not addressed is the suggestion to remove the return value from dispatchMessage() because it is currently unused. However, according to the ERC5164 standard specification, the function must return a message ID. Therefore, in order to adhere to the interface, we need to keep the current behavior for our adapters.*

H. Fix Review Status Categories

The following table describes the statuses used to indicate whether an issue has been sufficiently addressed.

Fix Status	
Status	**Description**
Undetermined	The status of the issue was not determined during this engagement.
Unresolved	The issue persists and has not been resolved.
Partially Resolved	The issue persists but has been partially resolved.
Resolved	The issue has been sufficiently resolved.