Meson Protocol

Design Review

July 15, 2022

Prepared for:

Meson

Prepared by: **Justin Jacob and Josselin Feist**

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 80+ 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

About Trail of Bits **1**

Notices and Remarks **2**

Table of Contents **3**

Executive Summary **4**

Project Summary **5**

Project Targets **6**

Discussion **7**

Introduction 7

Upgradeability 7

Infinite Token Approvals 8

Hard-Coded Bit Shifts 8

Front-Running Risks 9

Signature Malleability 9

LP Misbehavior 10

Documentation 10

A. Incident Response Recommendations **12**

B. Upgradeability Recommendations **14**

Executive Summary

Engagement Overview

Meson engaged Trail of Bits to review the design of its Meson protocol. From June 27 to June 29, 2022, one consultant conducted a design review of the client-provided source code, with three person-days of effort. Details of the project's timeline, test targets, and coverage are provided in subsequent sections of this report.

Project Scope

In conducting this design review, we focused on aspects of the protocol's design that could pose security issues. Our testing efforts were focused on the identification of flaws that could result in a compromise of confidentiality, integrity, or availability of the target system. We conducted this audit with access to documentation describing the system and its functions, as well as the source code of the Meson protocol, written in Solidity.

Project Summary

Contact Information

The following managers were associated with this project:

Dan Guido, Account Manager
dan@trailofbits.com

Mary O'Brien, Project Manager
mary.obrien@trailofbits.com

The following engineers were associated with this project:

Justin Jacob, Consultant
justin.jacob@trailofbits.com

Josselin Feist, Consultant
josselin@trailofbits.com

Project Timeline

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

Date	Event
June 21, 2022	Pre-project kickoff call
June 30, 2022	Delivery of report draft
July 5, 2022	Report readout meeting
July 15, 2022	Delivery of final report

Project Targets

The engagement involved a review and testing of the following target.

Meson Protocol

Repository	https://github.com/MesonFi/meson-contracts-solidity
Version	`b1fe6bc055f670e7b3f4fd14ce4b340dbc480e95`
Type	Solidity
Platform	Ethereum

Discussion

Introduction

The Meson protocol is a cross-chain exchange protocol that facilitates stablecoin swaps. Through the use of signatures, users can avoid paying gas fees for their swaps. Additionally, transactions are broadcast to various liquidity providers (LPs) through both a peer-to-peer (P2P) network and a relayer component.

Throughout our design review, we identified the following areas of concern in the protocol:

- **The use of UUPS proxy contracts to make the Meson contracts upgradeable**
- **The allocation of infinite token approvals**
- **Hard-coded bit shifts throughout the codebase**
- **Risks associated with the possible front-running of user transactions and with denial-of-service (DOS) attacks**
- **Risks associated with the use of ECDSA signatures throughout the codebase**
- **Risks associated with the possibility of misbehaving LPs**
- **Insufficient documentation**

Upgradeability

The Meson protocol's contracts use a UUPS proxy pattern, making them upgradeable. However, we have multiple concerns with regard to the UUPS proxy pattern used. In particular, the UUPS proxy contracts have an `initialize()` function that acts as a constructor. This function could be front-run by a malicious user during deployment to take control over the contracts' ownership and their tokens. Furthermore, UUPS proxies use `delegatecall` to call the implementation contract. The use of `delegatecall` is error-prone and can result in various footguns in the contracts. This Trail of Bits blog post highlights some of these issues.

In appendix B, we provide recommendations for implementing upgradeable contracts in ways that avoid this front-running issue and the use of `delegatecall`.

Response
The Meson team acknowledged this issue and plans to investigate issues surrounding the UUPS proxy pattern and alternative solutions.

Infinite Token Approvals

Another area of concern we have is the use of infinite approvals for ERC20 tokens. If a user does not initially have the appropriate allowance to transfer a token, and if the caller is the `Meson` contract, an infinite approval is allocated to the contract but never revoked. This poses major security risks; since approvals are never revoked, all funds in the `Meson` contract will be at risk in the event of a vulnerability. In fact, there have been numerous incidents in which the use of infinite approvals has been exploited, such as the 2020 Bancor incident and the 2021 Primitive Finance incident.

To mitigate this risk, we recommend allocating only necessary approvals and using `SafeERC20` library functions like `safeIncreaseAllowance()` and `safeDecreaseAllowance()`.

Response
The Meson team acknowledged this issue and indicated that the tokens that receive infinite approvals are used only for event emission and are burned immediately after.

Hard-Coded Bit Shifts

To save gas, most external functions in the codebase take only a single `uint256` parameter and then use helper methods to extract data from this number via bit shifts. For instance, the `amount` parameter used to measure how much of a token to swap is 48 bits, so a bit shift of 208 bits is used to extract the amount from the call data.

We have concerns with the use of these hard-coded bit shifts in the helper methods throughout the codebase. The number of tokens that needs to be swapped is determined by the token's decimals, which are variable depending on the token. Therefore, bit shifts that are hard-coded could be problematic when interacting with different tokens.

In addition to implementing a comprehensive unit test suite, use fuzz testing and integrate Echidna into the test suite to comprehensively test the effects of bit shifts and to account for any edge cases.

Response
The Meson team acknowledged this issue and plans to allow only whitelisted stablecoins to be used in the protocol.

Front-Running Risks

Transactions are broadcast to a public mempool before they are successfully added and executed on-chain. Because the mempool is public, an attacker can monitor the mempool and use the information to make a profit or to grief users by preventing their transactions from going through.

The protocol is susceptible to a number of attacks that involve front-running. In particular, we are concerned that the protocol's use of signatures could be used to conduct DOS attacks. Since users do not pay gas fees to generate and submit signatures, a malicious user could generate cheap signatures (by signing swap transactions for 1 wei) and spam the relayer and the LP, griefing other users.

If the mempool is congested or the relayer is spammed with signatures, a malicious user could front-run the `cancel()` function and prevent users from successfully swapping across chains. Although the `executeSwap()` function can be called by anyone, during periods of high chain congestion, it may be impossible to call the function until after the deadline. At this point, anyone can front-run a call to the function and call `cancel()` instead.

Preventing this attack vector requires deeper knowledge of the system. We recommend that Meson investigate and document the front-running risks in the system and explore mitigation strategies.

Response
The Meson team acknowledged this issue and plans to implement an anti-spamming scheme in the relayer service.

Signature Malleability

Through the use of signatures, users can avoid paying gas fees for their swaps. Users generate signatures with the ECDSA signing algorithm, and the smart contracts verify them on-chain. LPs pass the signature parameters into the contract and thus pay the gas fee for the swap instead of the users. The use of signatures in this manner is known as a meta-transaction.

Our main concern with the use of signatures is that there is no restriction placed on the `s` parameter when signatures are verified, making signatures vulnerable to signature malleability attacks. Consider using a helper library such as OpenZeppelin's ECDSA library to prevent such malleability issues.

Response
The Meson team acknowledged this issue and plans to investigate it further.

LP Misbehavior

The LPs are one of the main components of the Meson protocol. They are responsible for waiving gas fees for users, validating swaps, and providing liquidity for the system.

There are a number of risks associated with misbehaving LPs. For example, a misbehaving LP could refuse to accept transactions, which could lead to a DOS. Another concern we have is that malicious LPs could bond to a swap and refuse to lock their funds. In this situation, users will have no choice but to wait until the swap expires to reclaim their assets.

Preventing these attack vectors requires more specialized knowledge about the Meson protocol. In the short term, we recommend documenting the LPs' privileges and the risks associated with relying on those privileges. In the long term, we recommend reducing the privileges that LPs have within the protocol to reduce the attack surface of the system.

Response

The Meson team acknowledged this issue. There are multiple LPs in the system, and as long as there is one honest LP, user transactions will be accepted. Furthermore, the Meson team plans to enact a penalty for misbehaving LPs.

Documentation

Meson provided us with documentation on the use of the Meson protocol to accompany the source code. We recommend that this documentation be expanded to cover all the system's components:

- Expand the user-facing documentation.
    - Document the risks posed to users if the relayer goes down or suffers a DOS attack.
    - Document the risks associated with DOS and signature phishing attacks.
    - Lastly, document the steps that Meson will take in the event of a hack or exploit.
- In the technical documentation, include diagrams detailing the bridge's state machine and illustrating the steps taken in the event of state changes or system operations. This documentation could also be expanded to explain the actions taken on both sides of the bridge when a cross-chain transfer is initiated.
- Expand the documentation on the P2P component and the C relayer. The APIs of and expected uptime of both components and the process by which the C relayer communicates with the P2P network should be thoroughly documented.
- Document all of the roles of the system (including users, LPs, and the relayer component) and highlight their respective privileges and expected behavior.
- Document all of the system invariants. These are properties that should always hold regardless of the state of the system. Additionally, explain why these invariants hold.

- Document the build and deployment process from start to finish, including any necessary configuration steps.

Response

The Meson team acknowledged this issue and plans to update the documentation for both technical and non-technical users as well as provide diagrams outlining the state machine.

A. Incident Response Recommendations

This section provides recommendations on formulating an incident response plan.

- **Identify the parties (either specific people or roles) responsible for implementing the mitigations when an issue occurs (e.g., deploying smart contracts, pausing contracts, upgrading the front end, etc.).**
- **Document internal processes for addressing situations in which a deployed remedy does not work or introduces a new bug.**
  - Consider documenting a plan of action for handling failed remediations.
- **Clearly describe the intended contract deployment process.**
- **Outline the circumstances under which Meson will compensate users affected by an issue (if any).**
  - Issues that warrant compensation could include an individual or aggregate loss or a loss resulting from user error, a contract flaw, or a third-party contract flaw.
- **Document how the team plans to stay up to date on new issues that could affect the system; awareness of such issues will inform future development work and help the team secure the deployment toolchain and the external on-chain and off-chain services that the system relies on.**
  - Identify sources of vulnerability news for each language and component used in the system, and subscribe to updates from each source. Consider creating a private Discord channel in which a bot will post the latest vulnerability news; this will provide the team with a way to track all updates in one place. Lastly, consider assigning certain team members to track news about vulnerabilities in specific components of the system.
- **Determine when the team will seek assistance from external parties (e.g., auditors, affected users, other protocol developers, etc.) and how it will onboard them.**
  - Effective remediation of certain issues may require collaboration with external parties.
- **Define contract behavior that would be considered abnormal by off-chain monitoring solutions.**

It is best practice to perform periodic dry runs of scenarios outlined in the incident response plan to find omissions and opportunities for improvement and to develop “muscle memory.” Additionally, document the frequency with which the team should perform dry runs of various scenarios, and perform dry runs of more likely scenarios more regularly. Create a template to be filled out with descriptions of any necessary improvements after each dry run.

B. Upgradeability Recommendations

In this appendix, we provide recommendations for developing a strategy for upgrading the contracts using migrations and for designing the contracts to be upgradeable without using the `delegatecall` proxy pattern.

Achieving Upgradeability with Migration

To make the system's contracts upgradeable without using the problematic `delegatecall` proxy pattern, we recommend using a migration procedure. Such a procedure describes what data to migrate and how to execute the migration. In essence, a migration procedure consists of two parts: recovering the data from the old contract, and writing the data to the new contract.

To implement this approach, consider the following questions:

- **Which contracts contain states that need to be migrated before new versions of the contracts are deployed?** In the current implementation, the `Meson` contract does not hold any state besides configuration data. However, the `initialize()` function needs to be initialized with a separate call.

- **Which features/contracts need to be paused when migrating contracts?** When performing a migration, the system will need to be stopped to cleanly extract the data from the old contract and write it to a new storage contract.

- **How will all the different types of data be extracted from a contract being migrated?** Which data can be retrieved through public getters, through events, from private variables by directly reading the contract storage, and from mappings? Each data type requires a different method of extraction.

- **How will Meson ensure that all of the data is completely extracted from each storage contract?** Write tests to ensure that all data is extracted from each storage contract. Develop a set of data extraction steps to completely recover all of the data from each contract, and write tests to ensure that these steps successfully recover all of the data.

- **How will each type of data be written to the new contracts?** Develop a set of steps to write each type of data to the new contracts, and write tests to ensure that all of the data is successfully written.

- **What will be the cost of a migration?** Because of the cost of gas for each transaction and the maximum block gas limit, a migration could be a costly procedure that spans multiple transactions and blocks. By calculating the cost of migrating each of the contracts (with varying amounts of contract storage data) up front, Meson can choose the most efficient plan when a contract needs to be migrated. Doing so will also provide an estimate for how long a migration may take in terms of the number of transactions and blocks.

- **How will new contracts provide a stage in which the migrated data can be written to the contract storage?** Consider setting new contracts into an initial paused state in which the data can be written, after which the contracts will be unpaused.

- **Which references need to be updated when migrating each of the contracts to their new versions?** In the tests ensuring that the process of writing data to the new contracts results in a functioning system, include tests that ensure all of the required references are updated.

- **How does migrating each contract impact external contracts that interact with the purpose-for-profit system?** Communicating with external systems that interact with the purpose-for-profit contracts could result in a better user experience.

References

- Trail of Bits: How contract migration works

Achieving Upgradeability with Data Separation

It is possible to design the contracts in such a way that does not require use of the `delegatecall` proxy pattern. The core idea is to have a separate logic contract and storage contract, as well as an entry point contract so that upgrading the logic and/or storage contract does not change their addresses; therefore, external contracts that interact with the system do not need to point to a new address in the event of an upgrade.

To implement this approach, consider the following questions:

- **How will the appropriate access controls be designed?** Consider which access controls will be in place for upgrading any of the contracts and for using the contracts' internal functionality.

- **Which transaction-related variables need to be adjusted to implement this approach?** For example, in a system that does not use the `delegatecall` proxy pattern, the `msg.sender` is the caller of a given function, not the account that initiated the transaction and then called `delegatecall`. Similarly, in such a system, the Ether sent in each call needs to be passed on to the next contract's function call; it is not automatically sent, as with a `delegatecall`.

- **Should there be one storage contract for the entire system or one for each contract?** Using one storage contract would simplify the storage of the entire system. However, migrating the data would require migrating the data of all the contracts from the one old storage contract to the new storage contract.

- **How will the logic be separated to minimize the amount of logic that needs to be updated during an upgrade?** For example, by separating the business logic from the arithmetic functions, upgrading either of the two would require only the associated contract to be updated. Functionality that is shared between multiple contracts could be placed in a single contract referenced by multiple other contracts.

- **How will the contracts be tested after an upgrade?** Writing tests of the logic and/or storage for each contract after an upgrade will help to uncover unforeseen problems.

- **How will data be stored in the storage contract(s)?** Consider whether the storage of data should be low level, such as storing types of data (e.g., `storeUint256()`) or more high level, such as the use of `storeVestingSchedule()`.

- **During a migration, how will all of the data from the storage contract(s) be efficiently read and written to the new contract(s)?** Design the storage contract's data layout in such a way that reading and writing all of the data is straightforward, allowing for an easier and cheaper migration procedure.

References

- Designing the Gemini dollar, a regulated, upgradeable, transparent stablecoin