Overlay Protocol

Security Assessment (Summary Report)

March 4, 2024

Prepared for:
Adam Kay
Overlay

Prepared by: **Benjamin Samuels and Justin Jacob**

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:

Anne Marie Barry, Project Manager
annemarie.barry@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:

Benjamin Samuels, Consultant
benjamin.samuels@trailofbits.com

Justin Jacob, Consultant
justin.jacob@trailofbits.com

Project Timeline

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

Date	Event
February 1, 2024	Pre-project kickoff call
February 12, 2024	Delivery of report draft
February 12, 2024	Report readout meeting
March 4, 2024	Delivery of summary report

Executive Summary

Engagement Overview

A team of two consultants conducted the review from February 5 to February 9, 2024, 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 changes to Overlay detailed in the coverage section, using automated and manual processes.

Observations and Impact

Coverage

Based on the information provided for this assessment, Trail of Bits has determined that the ideal level of effort for the provided scope is four engineer-weeks. Due to the reduction to two engineer-weeks, we prioritized our review based on discussions with the client throughout the engagement.

Overlay engaged Trail of Bits to review the security of multiple changes made to the Overlay core protocol, focusing on the following changes:

- PR #70 - Fix oiShares Calculation
- PR #162 - Check Stale Price Feed
- PR #133 - Add Chainlink feeds and remove Uniswap V3 feeds
- PR #146 - Fix oiAfterFunding
- PR #141 - Refactor OV Token
- PR #78 - Add Emergency Shutdown and Withdraw
- PR #122 - Add pause and unpause functions to OverlayV1Factory/Market
- PR #171 - Blocktime rescale

These changes were reviewed for high-level logic issues, edge conditions, integration issues with Arbitrum, and re-entrancy. Some limited analysis on rounding directions was performed, but we recommend that the team perform an exhaustive analysis following the guidance in appendix B.

Trail of Bits identified that the following areas would benefit from further review:

- The effectiveness of the system's risk management system and its parameters
- The implementation of the system's risk management controls, specifically the rollers/throttling components
- The system's susceptibility to front-running, back-running, or sandwiching attacks
- Logic outside of the scope of the pull requests listed above
- The arithmetic properties of the modified `LogExp` and `FullMath` libraries
- Rounding issues in the system as a whole
- The system's privileged roles and their impact on the system

In addition, two of the medium-priority pull requests flagged for review could not be reviewed due to time constraints:

- PR #76 - Separate circuit breaker check from slippage
- PR #167 - Reduce calculations on FixedCast operations

Review Summary

Overall, Overlay is a relatively complex perpetual futures protocol with a number of moving parts. Despite this, the implementation is well organized. However, the testing suite has aged since it was first written; we highly recommend implementing stateful fuzzing to test the system's boundary conditions.

Recommendations

Based on the codebase maturity evaluation and findings identified during the security review, Trail of Bits recommends the following:

- **Remediate the findings disclosed in this report**. These findings should be addressed as part of direct remediation or any refactoring that may occur when addressing other recommendations.

- **Add a stateful fuzzing test suite**. Trail of Bits considers stateful fuzzing a baseline requirement for DeFi protocols and applications and **highly** recommends fuzzing the system's basic user flows (`build`, `unwind`, `liquidate`) at the absolute minimum.

  Overlay has a complex state machine with many boundary conditions that are not all individually unit-tested. Even highly unit-tested systems can be hacked due to compound behavior from multiple moving parts or calls occurring in an unexpected order for a specific state space.

  Guidance on introducing stateful fuzzing to a DeFi protocol like Overlay can be found in Trail of Bits' Learn how to fuzz like a pro series on YouTube.

- **Perform a rounding analysis on the protocol's various formulas**. Given the coverage limitations of this engagement, we recommend that Overlay perform a rounding analysis using the rounding guidance provided in appendix B.

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	Overlay uses a large amount of arithmetic to facilitate market-making for perpetuals. However, this arithmetic could not be sufficiently evaluated within the duration of the engagement.	**Further Investigation Required**
Auditing	All state-changing operations in the contract emit events. However, it is unclear if the Overlay team has any monitoring or incident response plan, or whether such a plan is required given the unknown impact of risk parameter changes to the system.	**Moderate**
Authentication / Access Controls	Overlay has several privileged roles with certain risk-setting responsibilities. However, the completeness of access controls and decentralization impact from the use of privileged roles could not be evaluated during the time allocated for the assessment.	**Further Investigation Required**
Complexity Management	Because Overlay is a perpetual futures platform, it is highly complex. The protocol's operations involve many boundary conditions and constraints, but each is separated into easily understandable components. In addition, the various user flows through the codebase are easy to follow and understand at a high level despite many complex state updates and calculations.	**Satisfactory**
Decentralization	Much of the system's power is contained in the Governor, as it can deploy new markets, adjust price feeds and risk parameters, and irreversibly shut down the system. Additionally, the situations in which the Governor will use its power to perform these functionalities are opaque and unspecified. Furthermore, it is unclear if the Overlay	**Moderate**

	team has any plans for decentralizing the system and, if so, how this will be done. Further documentation regarding the above aspects will be crucial for users to garner trust in the system.	
Documentation	The code contains sufficient in-line documentation and NatSpec comments. In addition, we received documentation in the form of the whitepaper and an outline of the user-facing functions. However, much of the documentation in the whitepaper is excessively complex and inaccessible to end users. There is also no documentation regarding admin powers and privileges or a simplified explanation for updating risk parameters.	**Moderate**
Low-Level Manipulation	The contracts do not use any low-level manipulation or inline assembly.	**Strong**
Testing and Verification	Overlay contains a comprehensive unit test suite, but lacks a modern stateful fuzzing suite that would be expected in a protocol of similar complexity.	**Moderate**
Transaction Ordering	Because of the limited time allocated for the review, we could not analyze the impact of transaction reordering on the system.	**Further Investigation Required**

Summary of Findings

The table below summarizes the review's findings, including type and severity details.

ID	Title	Type	Severity
1	Arbitrum sequencer is not checked for uptime	Data Validation	**Medium**
2	Underwater positions have no incentive to be liquidated	Arithmetic	**Medium**
3	Backrun bounder incorrectly assumes Arbitrum's block time	Data Validation	**Undetermined**

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	The system’s resistance to transaction-ordering 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. Rounding Guidance

This appendix provides guidance on how arithmetic calculations must be rounded for protocols to remain solvent. This guidance should be followed for all arithmetic operations relating to functionality such as token deposits, withdrawals, transfers, and fee calculations.

Rounding guidance is necessary because certain arithmetic operations lose precision; this means that the final result of the operation is an approximation instead of the true result. The difference between an approximated result and the true result is called epsilon. Protocols must be aware of where epsilon will manifest in arithmetic calculations and correctly account for epsilon to prevent insolvency.

Trail of Bits maintains roundme, a human-assisted tool for determining rounding directions for individual operations in an arithmetic formula. After reading the sections below, engineers should determine which direction high-level formulas should round in, then use roundme to determine the rounding directions of the individual arithmetic operations that make up the formula.

Recommendations for Determining Rounding Directions

- **Identify where funds are sent from the protocol to users and ensure that the associated operations round down their results.** This will minimize the number of tokens that are sent out to users.
- **Identify where funds are sent from users to the protocol and ensure that the associated operations round up their results.** This will maximize the number of tokens that are kept by the pool.
- **Identify where funds are transferred from the protocol for fees and ensure that the associated operations round up their results.** This will maximize the number of tokens that stay inside the protocol.
- **Trace each variable backward through the project to verify that the rounding directions used to generate it are consistent.** This will help reduce the time it takes to debug more complex rounding issues.

Reasoning about Rounding Directions

To determine whether an operation should round up or down, one must reason about how the outcome of the operation impacts token transfers.

Consider the following equation used by a vault contract to calculate how many `share` tokens a user should be credited for a deposit of `assets`:

$$sharesToMint = \left(\frac{assets * supply}{totalAssets}\right)$$

In order for this calculation to benefit the vault, `sharesToMint` should be rounded down. Variables that need to be rounded down are annotated with (↘):

$$sharesToMint\searrow \ = \ (\frac{assets\searrow * supply\searrow}{totalAssets\nearrow})\searrow$$

These variables must now be traced back through the contract to verify they are rounded correctly. Sometimes a system is designed in such a way that not all variables can be rounded correctly; in cases like this, the protocol must be redesigned to facilitate correct rounding.

Integer Arithmetic Primitives

In integer arithmetic, division operations lose precision due to rounding. Therefore, division operations need two primitive operators to correctly approximate results: one where the quotient is rounded up and one where the quotient is rounded down.

Rounding down is the default behavior:

$$div_{down}(a, b) \ = \ \frac{a}{b}$$

The following is one approximation for rounding up an integer division operation:

$$div_{up}(a, b) \ = \begin{cases} \frac{a}{b}, \ a \bmod b \ = \ 0 \\ \frac{a}{b} + 1, \ a \bmod b \neq 0 \end{cases}$$

div_{down} and div_{up} can then be used as primitives to build the following:

- $mulDiv_{up}$
- $mulDiv_{down}$

C. 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 a comprehensive analysis of the system.

On February 20, 2024, Trail of Bits reviewed the fixes and mitigations implemented by the Overlay team for the issues identified in this report. We reviewed each fix to determine its effectiveness in resolving the associated issue.

In summary, Overlay has resolved all three issues described in this report. For additional information, please see the Detailed Fix Review Results below.

ID	Title	Status
1	Arbitrum sequencer is not checked for uptime	**Resolved**
2	Underwater positions have no incentive to be liquidated	**Resolved**
3	Backrun bounder incorrectly assumes Arbitrum's block time	**Resolved**

Detailed Fix Review Results

TOB-OVERLAY-1: Arbitrum sequencer is not checked for uptime
Resolved in PR #180. Additional logic was added to use Chainlink's sequencer oracle to determine whether the sequencer is operational.

TOB-OVERLAY-2: Underwater positions have no incentive to be liquidated
Finding resolved based on information provided by the client. An off-chain liquidator bot will be run to liquidate underwater positions and mitigate the risk of underwater positions for the protocol.

TOB-OVERLAY-3: Backrun bounder incorrectly assumes Arbitrum's block time
Resolved in PR #181. Additional tooling has been added to detect scenarios where Arbitrum's block time deviates from the mean block time. An additional RISK_MANAGER role was also added to allow the modification of `averageBlockTime` during runtime.

D. 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.