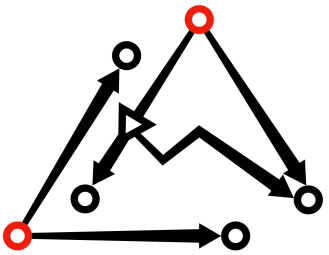

Least Authority

PRIVACY MATTERS

Overlay Protocol V1 Core Smart Contracts

Security Audit Report

Overlay Market

Final Audit Report: 08 June 2022

Table of Contents

Overview

Background

Overlay Market has requested that Least Authority perform a security audit of the Overlay Protocol V1 Core smart contracts.

Overlay is a Decentralized Finance (DeFi) protocol for trading positions based on data streams, enabling markets without the need for traditional counterparties or liquidity pools. The protocol accomplishes this through its native token, OVL.

Project Dates

- **April 4 - May 9**: Initial Code Review *(Completed)*
- **May 11**: Delivery of Initial Audit Report *(Completed)*
- **June 6 - June 7**: Verification Review *(Completed)*
- **June 8**: Delivery of Final Audit Report *(Completed)*

Review Team

- David Braun, Security Researcher and Engineer
- Steven Jung, Security Researcher and Engineer
- Xenofon Mitakidis, Security Researcher and Engineer

Coverage

Target Code and Revision

For this audit, we performed research, investigation, and review of the Overlay Protocol followed by issue reporting, along with mitigation and remediation instructions as outlined in this report.

The following code repository is considered in-scope for the review:

https://github.com/overlay-market/v1-core

Specifically, we examined the Git revision for our initial review:

`c480f6f9af526d4c15f16a3442b2d090197cfb76`

For the verification, we examined the Git revision:

24dffd529068cf3d3e8b3599a06d9aebfd212e37

For the review, this repository was cloned for use during the audit and for reference in this report:

https://github.com/LeastAuthority/Overlay-Protocol-Smart-Contracts

All file references in this document use Unix-style paths relative to the project's root directory.

In addition, any dependency and third-party code, unless specifically mentioned as in-scope, were considered out of scope for this review.

Supporting Documentation

The following documentation was available to the review team:

- Overlay V1 Core.pdf: https://planckcat.mypinata.cloud/ipfs/QmVMX7DH8Kh22kxMyDFGUJcw1a3irNPvyZBtAoqkyJYJEv
- `README.md`: https://github.com/LeastAuthority/Overlay-Protocol-Smart-Contracts/blob/master/README.md
- Docs: https://github.com/LeastAuthority/Overlay-Protocol-Smart-Contracts/tree/master/docs

Areas of Concern

Our investigation focused on the following areas:

- Correctness of the implementation;
- Common and case-specific implementation errors;
- Adherence to the specification and best practices;
- Adversarial actions and other attacks on the smart contracts;
- Potential misuse and gaming of the smart contracts;
- Attacks that impact funds, such as the draining or manipulation of funds;
- Mismanagement of funds via transactions;
- Denial of Service (DoS) and security exploits that would impact the code's intended use or disrupt the execution of the code;
- Vulnerabilities in the code for all features;
- Protection against malicious attacks and other ways to exploit the smart contracts;
- Inappropriate permissions and excess authority;
- Data privacy, data leaking, and information integrity; and
- Anything else as identified during the initial analysis phase.

Findings

General Comments

The Overlay protocol enables users to take long and short positions against values from any stream of data, with V1 focusing on the price of DeFi tokens. A distinguishing feature of the protocol is that it enables trading without counterparties by minting and burning a settlement token (OVL).

OVL's value is determined by the equilibrium of supply and demand for the token. A minting and burning imbalance of OVL would create an inflation risk. An increase in the supply of the OVL token would reduce its value in ETH terms, which would affect the functionality of the system. This risk is mitigated by defensive mechanisms that work to keep the minting and burning of OVL at an optimal level.

Our team attempted to force existing positions to become liquidatable by executing a flash loan attack. The attack attempted to build large subsequent positions that would cause the existing positions to lose value. However, the attack was thwarted by the circuit breaker defense. We observed that the risk parameters `maintenanceMarginFraction` and `liquidationFeeRate` are critical in preventing flash loan attacks and should be set carefully by a secure governance mechanism.

We also attempted to create many small positions as an alternative vector to manipulate the value of existing positions but found that the gas cost was prohibitive. Given that the ability to access the feed

creation mechanism is unguarded, we explored the potential for griefing attacks but did not identify any issues.

Our team found the protocol to be well designed and implemented. However, the project is based on complex finance structures that increased the auditing complexity substantially.

System Design

We found that security has been taken into consideration in the design of the Overlay protocol as demonstrated by a careful delineation of roles and the authority granted to them. Our team noted that sufficient input validation has been implemented where appropriate, in addition to mechanisms that hinder attacks, such as caps, spreads, and market-specific parameters set by protocol governance. However, governance functionality, which is critical to the security of such a system, has not been implemented yet. As such, the protocol cannot be considered ready for launch until a proper governance system is in place.

The current implementation of the Overlay protocol relies on a single oracle, Uniswap's WETH/OVL pool, as a price feed that is used to determine the profitability of trader positions. We recommend that functionality to aggregate additional price data from other liquidity pools be implemented to reduce this single point of failure risk (Suggestion 6).

Code Quality

The Overlay code is well organized and adheres to best practices as demonstrated by optimizing storage space by rationally assigning storage variables, moving logic into libraries to minimize the code, appropriately using the `require` function to ensure the validity of contract state transitions, and also using interfaces to improve readability and facilitate reasoning about the code by abstracting code functionality.

However, we found instances of redundant code, which should be removed to improve the readability and maintainability of the code (Suggestion 2, Suggestion 3). Additionally, we suggest that the `toInt192Bounded` function be simplified to improve readability (Suggestion 4), and that deprecated calls to `_setupRole` be updated to `_grantRole` (Suggestion 5).

Lastly, the implementation imports the `errors` library for the purpose of gas efficiency. We recommend using custom errors, instead, as a better practice (Suggestion 1).

Tests

Overlay has comprehensive test coverage, which significantly aids developers and security researchers in identifying implementation errors that could lead to security vulnerabilities. To test for edge case scenarios, we suggest the creation and running of long-duration simulations in which bots execute millions of trades (Suggestion 7).

Documentation

The project documentation for this review included a README, which describes the general architecture of the system, in addition to a file that lists and describes the core functions implemented in the protocol. Other background documentation was out of date and considered out of scope for this audit, inhibiting our understanding of the functionality of the protocol in a broader context. Furthermore, the project documentation insufficiently describes how the governance of the protocol will be handled. We recommend that the documentation be improved to cover all components of the system (Suggestion 9).

The project documentation also included a whitepaper describing the economic model that the Overlay protocol is governed by, which explains the theory well and describes the security mechanisms that are in place.

Code Comments

The documentation within the code sufficiently describes the intended behavior of security-critical functions and components.

Scope

The in-scope repository for this review included all security critical components of the Overlay protocol. However, key components that are necessary for Overlay to function securely, such as the governance and fee recipient mechanisms, were not in-scope for this review and must be included in a follow-up audit.

Dependencies

The implementation utilizes a few dependencies like the `FullMath.sol` and `TickMath.sol` libraries from the 0.8 branch of Uniswap. We examined the two libraries and did not identify any issues.

Specific Issues & Suggestions

We list the issues and suggestions found during the review in the order we reported them. In most cases, remediation of an issue is preferable, but mitigation is suggested as another option for cases where a trade-off could be required.

ISSUE / SUGGESTION	STATUS
Suggestion 1: Improve Error Handling	Unresolved
Suggestion 2: Remove Unnecessary Code	Resolved
Suggestion 3: Remove Redundant Lines in OverlayV1Token Constructor	Resolved
Suggestion 4: Simplify toInt192Bounded	Resolved
Suggestion 5: Replace _setupRole with _grantRole	Resolved
Suggestion 6: Use Multiple Oracles for Feeds	Unresolved
Suggestion 7: Create A Long-Duration Simulation to Test Properties	Unresolved
Suggestion 8: Make Positions Transferable	Unresolved
Suggestion 9: Improve Documentation	Unresolved

Suggestions

Suggestion 1: Improve Error Handling

Location

`/contracts/utils/Errors.sol`

`/contracts/libraries/LogExpMath.sol`

Synopsis

The Overlay team utilizes the `Errors` library to reduce gas costs in case of `reverts`. However, only 5 out of the 140 defined error codes are used. Custom errors in Solidity provide similar gas efficiencies, in addition to being more featureful by supporting arguments and increased readability, which can make errors easier to understand.

Mitigation

We recommend using custom errors for all error handling.

Status

The Overlay team chose not to address this suggestion.

Verification

Unresolved.

Suggestion 2: Remove Unused Code

Location

`/contracts/feeds/uniswapv3/OverlayV1UniswapV3Feed.sol#L283`

Synopsis

```
uint32 secondsAgo = secondsAgos[i];
```

The variable `secondsAgo` is no longer used, so its declaration can cause confusion for reviewers and maintainers.

Mitigation

We recommend removing the unnecessary line of code.

Status

The Overlay team has removed the unused code as suggested.

Verification

Resolved.

Suggestion 3: Remove Redundant Lines in `OverlayV1Token` Constructor

Location

`/contracts/OverlayV1Token.sol#L13-L15`

Synopsis

The following lines in `OverlayV1Token` are redundant:

```
_setRoleAdmin(MINTER_ROLE, DEFAULT_ADMIN_ROLE);
_setRoleAdmin(BURNER_ROLE, DEFAULT_ADMIN_ROLE);
_setRoleAdmin(GOVERNOR_ROLE, DEFAULT_ADMIN_ROLE);
```

The OpenZeppelin documentation reads:

`AccessControl` includes a special role called `DEFAULT_ADMIN_ROLE`, which acts as the default admin role for all roles. An account with this role will be able to manage any other role, unless `_setRoleAdmin` is used to select a new admin role.

Mitigation

We recommend removing the redundant lines.

Status

The Overlay team removed the redundant lines as suggested.

Verification

Resolved.

Suggestion 4: Simplify `toInt192Bounded`

Location

/contracts/libraries/Cast.sol#L12-L19

Synopsis

There is an opportunity to improve the readability of the `toInt192Bounded` function.

Mitigation

We recommend this minor refactor, which is easier to read and saves a comparison:

```
    function toInt192Bounded(int256 value) internal pure returns (int192) {
        int192 value192 = value < type(int192).min
            ? type(int192).min
            : value > type(int192).max
                ? type(int192).max
                : int192(value);

        return value192;
    }
```

Status

The Overlay team refactored the function as suggested.

Verification

Resolved.

Suggestion 5: Replace `_setupRole` with `_grantRole`

Location

/contracts/OverlayV1Token.sol#L11-L12

Synopsis

In the following code, `_setupRole` is called:

```
        _setupRole(DEFAULT_ADMIN_ROLE, msg.sender);
        _setupRole(MINTER_ROLE, msg.sender);
```

According to OpenZepplin's documentation, `_setupRole` is deprecated in favor of `_grantRole`.

Mitigation

We recommend replacing `_setupRole` with `_grantRole`.

Status

`_grantRole` is now used for `DEFAULT_ADMIN_ROLE` and the `MINTER_ROLE` line has been removed by the Overlay team.

Verification

Resolved.

Suggestion 6: Use Multiple Oracles for Feeds

Location

`/contracts/feeds/OverlayV1Feed.sol`

Synopsis

The current design of Overlay relies on a single oracle for each feed. This creates a single point of failure because individual oracles can become unavailable, supply incorrect data, or be manipulated.

Mitigation

We suggest sampling data from multiple oracles for each feed and combining the results. For the feed implemented in the first Overlay market (OVL/WETH), a second liquidity pool could be created on an automated market maker other than Uniswap.

Status

The Overlay team chose not to address the suggestion.

Verification

Unresolved.

Suggestion 7: Create A Long-Duration Simulation to Test Properties

Synopsis

There are many tunable risk parameters to help manage the risk of the protocol drifting into an undesirable state (e.g. excessive OVL inflation). It is difficult to anticipate the future behavior of the markets and to know whether the risk parameters are set appropriately.

Mitigation

We suggest creating a long-running simulation in which bots place millions of randomly generated trades. The simulation should monitor key systemic properties (such as acceptable OVL inflation) and alert if any of them go out of bounds.

Status

The Overlay team responded they intend to address this suggestion in the future.

Verification

Unresolved.

Suggestion 8: Make Positions Transferable

Location

/contracts/OverlayV1Market.sol#L268

Synopsis

In the current design, trader positions can only be unwound by the same Ethereum account that built them. The Overlay protocol could be more valuable if it were possible for traders to trade the right to unwind a position to other traders. This would make the Overlay protocol a more composable and multipurpose DeFi building block, which could generate more revenue through trading fees.

Mitigation

We recommend making trader positions transferable.

Status

The Overlay team plans to add this feature in a future peripheral library outside the core.

Verification

Unresolved.

Suggestion 9: Improve Documentation

Synopsis

The README provides instructions for running the code, a simple architecture diagram, and some developer documentation. The whitepaper is an excellent deep dive into the financial formulas used in the protocol. What is missing is up-to-date documentation that explains the problem that the Overlay protocol is trying to solve and how it solves it. It is also missing detailed descriptions of key components of the system and how they interact with each other (such as governance, liquidity pools, and the fee recipient).

Mitigation

We recommend documentation be created that introduces the big picture of the Overlay protocol with background context, visual diagrams including system components and interactions, and detailed descriptions of planned but undocumented components.

Status

The Overlay team responded they intend to address this suggestion in the future.

Verification

Unresolved.

About Least Authority

We believe that people have a fundamental right to privacy and that the use of secure solutions enables people to more freely use the Internet and other connected technologies. We provide security consulting services to help others make their solutions more resistant to unauthorized access to data and

unintended manipulation of the system. We support teams from the design phase through the production launch and after.

The Least Authority team has skills for reviewing code in C, C++, Python, Haskell, Rust, Node.js, Solidity, Go, and JavaScript for common security vulnerabilities and specific attack vectors. The team has reviewed implementations of cryptographic protocols and distributed system architecture, including in cryptocurrency, blockchains, payments, and smart contracts. Additionally, the team can utilize various tools to scan code and networks and build custom tools as necessary.

Least Authority was formed in 2011 to create and further empower freedom-compatible technologies. We moved the company to Berlin in 2016 and continue to expand our efforts. Although we are a small team, we believe that we can have a significant impact on the world by being transparent and open about the work we do.

For more information about our security consulting, please visit https://leastauthority.com/security-consulting/.

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis

Our audit techniques included manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's web site to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation. We hypothesize what vulnerabilities may be present, creating Issue entries, and for each, we follow the following Issue Investigation and Remediation process.

Documenting Results

We follow a conservative and transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative and we strive to provide test

code, log captures, or screenshots demonstrating our confirmation. After this, we analyze the feasibility of an attack in a live system.

Suggested Solutions

We search for immediate mitigations that live deployments can take and finally, we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

Responsible Disclosure

Before our report or any details about our findings and suggested solutions are made public, we like to work with your team to find reasonable outcomes that can be addressed as soon as possible without an overly negative impact on pre-existing plans. Although the handling of issues must be done on a case-by-case basis, we always like to agree on a timeline for resolution that balances the impact on the users and the needs of your project team. We take this agreed timeline into account before publishing any reports to avoid the necessity for full disclosure.