HyperDrive Audit Report

Version 1.0

Conducted by: Kiki

March 2025

Table of Contents

1 Introduction **3**
- 1.1 About Kiki . . . 3
- 1.2 Disclaimer . . . 3
- 1.3 Risk classification . . . 3
  - 1.3.1 Impact . . . 3
  - 1.3.2 Likelihood . . . 3
  - 1.3.3 Actions required by severity level . . . 3

2 Executive Summary **4**
- 2.1 HyperDrive . . . 4
- 2.2 Overview . . . 4
- 2.3 Scope . . . 4
- 2.4 Issues Found . . . 5
- 2.5 Findings & Resolutions . . . 5

3 Findings **6**
- 3.1 Medium Risk . . . 6
  - 3.1.1 WETH Double Counting in Maximum Supply Check . . . 6
- 3.2 Low Risk . . . 6
  - 3.2.1 Flash Loan Enables Max Collateral Limit Bypass . . . 6
  - 3.2.2 Unbounded Collateral Assets Cause Gas Limits . . . 7
  - 3.2.3 Interest Can Accrue During Pause Periods . . . 8
  - 3.2.4 Liquidations Possible During Protocol Pause . . . 9
  - 3.2.5 Unreachable Maximum Supply Limit . . . 9
  - 3.2.6 View Functions Show False Interest During Pause . . . 10

1 Introduction

1.1 About Kiki

Kiki is a Security Researcher who has conducted dozens of security reviews with the top security firm Guardian Audits, as well as through private engagements. View their previous work here, or reach out via Twitter or Telegram.

1.2 Disclaimer

Security Reviews are a time, resource, and expertise bound effort where trained experts evaluate smart contracts using a combination of automated and manual techniques to identify as many vulnerabilities as possible. Audits can show the presence of vulnerabilities **but not their absence**.

1.3 Risk classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

1.3.1 Impact

- **High** - leads to a significant loss of assets in the protocol or significantly harms a group of users.
- **Medium** - only a small amount of funds can be lost or a functionality of the protocol is affected.
- **Low** - any kind of unexpected behaviour that's not so critical.

1.3.2 Likelihood

- **High** - direct attack vector; the cost is relatively low to the amount of funds that can be lost.
- **Medium** - only conditionally incentivized attack vector, but still relatively likely.
- **Low** - too many or too unlikely assumptions; provides little or no incentive.

1.3.3 Actions required by severity level

- **Critical** - client **must** fix the issue.
- **High** - client **must** fix the issue.
- **Medium** - client **should** fix the issue.
- **Low** - client **could** fix the issue.

2 Executive Summary

2.1 HyperDrive

Hyperdrive offers risk mitigated high-yield strategies to those looking to safely park their USDC. Hyperdrive is the premier DeFi hub on Hyperliquid, offering unparalleled yield strategies to all users.

More on the protocol can be found here.

2.2 Overview

Project Name	HyperDrive
Codebase	hyperdrive-contracts
Operating platform	HyperEVM
Language	Solidity
Initial commit	[1] ae86595454d2e92c36f669e221cdd510535286e5
Remediation commit	578a708fd292587588bd29b925e8d6fc1971a032
Audit methodology	Manual Review
Engagement Duration	5 Days

2.3 Scope

Files and folders in scope

- packages/lending/contracts/protocol/markets/CollateralLib.sol
- packages/lending/contracts/protocol/markets/LiquidationLib.sol
- packages/lending/contracts/protocol/markets/Market.sol
- packages/lending/contracts/protocol/markets/MarketHooksLib.sol
- packages/lending/contracts/protocol/markets/MarketLib.sol

2.4 Issues Found

Severity	Total Found	Resolved	Partially Resolved	Acknowledged
Critical risk	0	0	0	0
High risk	0	0	0	0
Medium risk	1	1	0	0
Low risk	6	5	0	1

2.5 Findings & Resolutions

ID	Title	Severity	Status
M-01	WETH Double Counting in Maximum Supply Check	Medium	Resolved
L-01	Flash Loan Enables Max Collateral Limit Bypass	Low	Resolved
L-02	Unbounded Collateral Assets Cause Gas Limits	Low	Acknowledged
L-03	Interest Can Accrue During Pause Periods	Low	Resolved
L-04	Liquidations Possible During Protocol Pause	Low	Resolved
L-05	Unreachable Maximum Supply Limit	Low	Resolved
L-06	View Functions Show False Interest During Pause	Low	Resolved

3 Findings

3.1 Medium Risk

3.1.1 WETH Double Counting in Maximum Supply Check

Severity: *Medium risk (Resolved)*

Context: ***CollateralLib.sol***:*124-127*

Description:

The maximum supply validation in `CollateralLib.sol` incorrectly handles ETH deposits that are wrapped to WETH, leading to double counting of funds.

Issue Breakdown:

1. When ETH is deposited, it is wrapped to WETH, meaning the WETH already exists in the contract balance.
2. The validation check adds the deposit amount again to this balance.
3. This can cause the `MaximumSupplyExceeded` check to fail incorrectly, preventing legitimate deposits.

Example Scenario:

- Max supply: `100 ETH`
- User deposits: `60 ETH`
- Initial contract balance: `0 ETH`
- After wrapping: `Contract holds 60 WETH`
- Check evaluates: `(60 + 60)> 100 => false`
- Reverts with: `MaximumSupplyExceeded`

This issue blocks valid deposits within the supply limit, reducing capital efficiency and worsening user experience.

Recommendation:

Implement a separate validation path for when the `supplyCollateralETH` function is used. This validation should only compares the contract's balance after the deposit to the maximum supply. This ensures WETH is not double-counted, allowing deposits to be correctly validated.

Resolution: Resolved. The recommended fix was implemented in **3ad44d5**.

3.2 Low Risk

3.2.1 Flash Loan Enables Max Collateral Limit Bypass

Severity: *Low risk (Resolved)*

Context: *CollateralLib.sol:124-127*

Description:

The `supplyCollateral` function in `CollateralLib.sol` allows users to bypass the maximum supply limit for a collateral asset when that asset is also the market's lending asset. The vulnerability arises from using the contract's current balance to enforce the maximum supply limit:

```
require(
  IERC20(token).balanceOf(address(this)) + amount < asset.maxSupply,
  IMarket.MaximumSupplyExceeded(asset.maxSupply)
);
```

In markets where the lending asset can also be used as collateral, this check can be bypassed through the following steps:

1. A user calls the `flash loan` function in `Market.sol` to temporarily borrow a large amount of the market's asset.
2. This temporarily reduces the contract's balance of that asset.
3. During the same transaction, the user calls `supplyCollateral` to supply an amount that would normally exceed the maximum limit.
4. The check passes because the contract's current balance is artificially low.
5. The user then repays the flash loan, restoring the contract's balance.
6. The result is a total collateral supply that exceeds the intended maximum.

This vulnerability only affects markets where the lending asset can also be supplied as collateral, but in those cases, it completely undermines the protection provided by the `maxSupply` parameter.

Recommendation:

Modify the validation in `supplyCollateral` to use the tracked total supplied amount instead of the contract's current balance when the asset supplied is the same as the market's lending asset.

This approach is more resistant to manipulation as it relies on internal accounting rather than the current token balance, which can be temporarily altered through flash loans.

Resolution: Resolved. The recommended fix was implemented in **589278c**.

3.2.2 Unbounded Collateral Assets Cause Gas Limits

Severity: *Low risk (Acknowledged)*

Context: *CollateralLib.sol:90*

Description:

The `addOrUpdateAsset` function in `CollateralLib.sol` lacks a limit on the number of collateral assets and hooks that can be added to a market. This creates a significant operational risk on chains with lower gas limits, particularly HyperEVM, which has a 2,000,000 block gas limit for its fast transactions (2-second blocks).

This creates two significant issues:

1. Liquidation operations like `liquidate` must iterate through all collateral assets of a user, resulting in gas costs that scale with the number of assets. With enough assets, these operations could exceed HyperEVM's fast transaction gas limit.
2. When a liquidation exceeds the gas limit of fast blocks (2,000,000), it must wait for HyperEVM's larger blocks, which only occur once per minute. This one-minute delay is particularly dangerous during volatile market conditions where rapid liquidations are essential to maintain protocol solvency.

The absence of an upper bound on collateral assets effectively creates a DoS condition for the liquidation mechanism specifically on HyperEVM, potentially allowing undercollateralized positions to deteriorate further before they can be liquidated.

Recommendation:

With such small block gas limits, it's important to carefully consider and Implement a configurable maximum limits. In this case, the number of collateral assets and hooks that can be added to a market.

HyperEVM is expected to increase block size over time, but for the time being, it's important to be conscientious of gas consumption if the goal is to keep transactions in small blocks.

Resolution: Acknowledged by the team.

3.2.3 Interest Can Accrue During Pause Periods

Severity: *Low risk (Resolved)*

Context: ***Market.sol***:*301-307*

Description:

The current implementation in `Market.sol` lacks a proper mechanism to prevent interest accrual during extended pause periods. While the contract has `beforePause` and `afterResume` functions that call `accrue`, there is a critical issue with how timestamps are managed between pause and resume operations.

The current implementation has the following flow:

1. When paused, `beforePause` calls `accrue`, which updates `$.lastUpdate` to the pause timestamp
2. When resumed, `afterResume` calls `accrue`, which calculates interest based on `block.timestamp - $.lastUpdate`

This calculation spans the entire pause duration, incorrectly accruing interest for a period when the market was explicitly paused. While `accrueLiabilities` does check for the `paused` state and returns 0 when paused, as well as updates `$.lastUpdate`. This protection only works if `accrue` is actually called during the pause period, which cannot be guaranteed.

This issue could lead to:

1. Economically incorrect interest calculations
2. Unexpected debt positions for borrowers upon market resumption
3. Potential liquidity issues or unexpected collateral requirements
4. Undermining user trust in the protocol's pause functionality

Recommendation:

Implement a `beforeResume` function that updates the `lastUpdate` timestamp to the current block timestamp before the market is resumed:

```
function beforeResume() public override {
  MarketStorage storage $ = getMarketStorage();
  $.lastUpdate = block.timestamp.toUint32();
}
```

This ensures that no interest accrues during the pause period, regardless of whether `accrue()` was called during that time, maintaining the economic validity of the pause mechanism.

Resolution: Resolved. The recommended fix was implemented in **b98576f**.

3.2.4 Liquidations Possible During Protocol Pause

Severity: *Low risk (Resolved)*

Context: ***Market.sol***:565-590

Description:

The `liquidate` functions in the `Market` contract are missing the `whenNotPaused` modifier, allowing liquidations to occur even when the protocol is paused. This creates an inconsistency with other critical functions like `borrow`, `repay`, `deposit`, and `withdraw`, which are properly protected with the `whenNotPaused` modifier.

When a protocol is paused, all critical financial operations should be suspended to prevent any potential issues during the pause period. However, the current implementation allows liquidators to continue liquidating positions even during a pause, which:

1. Creates inconsistency with other paused operations
2. Could lead to unfair liquidations when users cannot interact with the protocol to manage their positions

This is particularly concerning because users cannot defend against liquidations during a pause period since they cannot repay their loans or add collateral while the protocol is paused.

Recommendation:

Add the `whenNotPaused` modifier to both `liquidate` functions to ensure they cannot be called during a protocol pause. Or clearly document that liquidations are always possible, even during pause periods.

Resolution: Resolved. The recommended fix was implemented in **23e264d**.

3.2.5 Unreachable Maximum Supply Limit

Severity: *Low risk (Resolved)*

Context: ***CollateralLib.sol***:125

Description:

The maximum supply validation in `CollateralLib.sol` uses an incorrect comparison operator, preventing users from fully utilizing the defined maximum supply limit.

Issue Breakdown:

The current implementation uses a strict less-than (<) operator instead of less-than-or-equal-to (<=):

```
IERC20(token).balanceOf(address(this)) + amount < asset.maxSupply
```

This causes the following problems:

1. Deposits that reach the max supply are rejected instead of allowed.
2. Users cannot fully utilize the intended supply limit.
3. This creates an unintended logical error, making the true max supply unreachable.

Recommendation:

Change the comparison operator from less-than (<) to less-than-or-equal-to (<=) to properly enforce the supply limit.

Resolution: Resolved. The recommended fix was implemented in **92616c5**.

3.2.6 View Functions Show False Interest During Pause

Severity: *Low risk (Resolved)*

Context: ***Market.sol**:504-510*

Description:

The `previewAccrueLiabilities` function in `Market.sol` calculates projected interest accrual without checking if the contract is paused. When the protocol is in a paused state, interest accrual is suspended by the `accrue` function, which correctly returns zero interest when paused. However, the preview function lacks this same check.

This discrepancy creates misleading information in multiple functions that rely on `previewAccrueLiabilities`:

1. The `previewLiabilities` function - Returns inflated debt values for users during a pause
2. The `totalAssets` function - Reports increased assets that include accrued interest which won't actually be collected
3. The `previewSnapshot` function - Returns a snapshot with incorrect utilization and exchange rates

For example, a user checking their position through `previewLiabilities` during a two-week pause might see their debt increase by 1%, when in reality no interest is accruing. Similarly, an integration using `totalAssets` to calculate pool metrics would see growing assets despite interest collection being suspended.

This creates several issues:

- Third-party protocols integrating with this market may make incorrect risk assessments or financial calculations

- The actual behavior when unpaused (no interest for the pause period) will differ from what users and integrations observed during the pause

Recommendation:

Modify the `previewAccrueLiabilities` function to check the paused state of the contract and return zero interest when paused:

```
function previewAccrueLiabilities() private view returns (uint256 accrued) {
    // Return 0 when contract is paused
    if (paused()) {
        return 0;
    }

    MarketStorage storage $ = getMarketStorage();
    uint128 rate = $.interestRateModel.safeRate($.totalSupplyAssets, $.
        totalBorrowAssets);
    accrued = rate == 0 ? 0 : $.previewAccrueLiabilities(rate);
}
```

This ensures consistency between what users and integrations observe during a pause and the actual interest that will be accrued when the contract resumes operation.

Resolution: Resolved. The recommended fix was implemented in **23e264d**.