ENIGMA DARK

Securing the Shadows



Managed Security Review

Hyperdrive LST

April, 2025

Contents

1. Summary
2. Engagement Overview
3. Risk Classification
4. Vulnerability Summary
5. Findings
6. Disclaimer

Summary

Enigma Dark
Enigma Dark is a web3 security firm leveraging the best talent in the space to secure all kinds of blockchain protocols and decentralized apps. Our team comprises experts who have honed their skills at some of the best auditing companies in the industry. With a proven track record as highly skilled white-hats, they bring a wealth of experience and a deep understanding of the technology and the ecosystem.

Learn more about us at enigmadark.com

Hyperdrive LST
Hyperdrive is an innovative lending protocol deployed in the HyperEVM which offers various yield-strategies integrating with Hyperliquid vaults.

Engagement Overview

Over the course of 1 week, beginning April 30 2025, Enigma Dark coordinated a Managed Security Review of the Hyperdrive LST project, specifically the contracts located at `packages/staking/contracts/protocol`. The review was performed by one Security Researcher: kiki.

The following repositories were reviewed at the specified commits:

Repository	Commit
ambitfi/hyperdrive-contracts	`c04cfce697461450be6c0634156961b9c95a9429`

Risk Classification

Severity	Description
Critical	Vulnerabilities that lead to a loss of a significant portion of funds of the system.
High	Exploitable, causing loss or manipulation of assets or data.
Medium	Risk of future exploits that may or may not impact the smart contract execution.
Low	Minor code errors that may or may not impact the smart contract execution.
Informational	Non-critical observations or suggestions for improving code quality, readability, or best practices.

Vulnerability Summary

Severity	Count	Fixed	Acknowledged
Critical	0	0	0
High	2	2	0
Medium	2	2	0
Low	1	1	0
Informational	6	6	0

Findings

Index	Issue Title	Status
H-01	Fee Yield Can Be Siphoned from Phantom Queue Processing	Fixed
H-02	Possible Fee Extraction & Withdrawal Queue Griefing	Fixed
M-01	Inaccurate Undelegation Amount Leads to Withdrawal DoS	Fixed
M-02	Temporary Share Price Inflation via Refund Logic	Fixed
L-01	Share Price Manipulation in Withdrawal Process	Fixed
I-01	Unreachable Time Condition Check	Fixed
I-02	Incorrect Exchange Rate Calculation	Fixed
I-03	Typo in Code Comment	Fixed
I-04	Inefficient Zero-Amount Undelegation Calls	Fixed
I-05	Unallocated Assets When Last Validator Has Zero-Weight	Fixed
I-06	Unnecessary Conversion in Delegate Function	Fixed

Detailed Findings

High Risk

H-01 - Fee Yield Can Be Siphoned from Phantom Queue Processing

Severity: High Risk

Context:

- WithdrawQueueUpgradeable.sol#L246-L252

Technical Details:
In the `WithdrawQueueUpgradeable` contract, the `executeEpochOperation` function credits the caller with fee shares as long as there is an item in either of the queues. The issue with this is that the fee share is given to the caller regardless of whether any withdrawal requests or pending request are actually processed. This creates a situation where:

1. If there are pending withdrawals but the 7-day waiting period hasn't passed yet
2. Any user can repeatedly call the `execute` function
3. Each call grants them fee shares
4. No actual processing occurs
5. No actual fees are collected

This allows attackers to accumulate fee shares without contributing any value to the protocol. Since fee shares entitle holders to a portion of the revenue, this effectively dilutes the value of legitimate fee shares and redirects legitimate revenue to attackers.

Impact:
Direct theft of unclaimed yield by obtaining fee shares without providing any value or performing legitimate withdrawals.

Recommendation:
Modify the `execute` function to only credit the caller with fee shares when actual requests are successfully processed.

Developer Response:
Fixed at commit `6594553`.

H-02 - Possible Fee Extraction & Withdrawal Queue Griefing

Severity: High Risk

Context:

- StakingVaultUpgradeable.sol#L712

Technical Details:
In the `StakingVaultUpgradeable` contract the cancellation mechanism has no cost beyond gas, which enables several attack vectors:

1. Queue Flooding: An attacker can flood the pending queue with redemption requests and then cancel them, creating a denial-of-service condition since only one order can be executed per block.

2. Yield Theft: The most severe attack occurs when the pending queue is empty. An attacker can:

    - Submit a redemption request
    - Immediately cancel it in the same transaction
    - Also in the same transaction Execute the cancellation request
    - Collect fee shares without contributing any actual fees
    - Repeat this process to accumulate fee shares and steal the yield generated from legitimate fee payments

This effectively allows an attacker to siphon revenue that should go to legitimate participants. And because the queue was initially empty and the attacker can execute this in one transaction they will have no competition for executing the cancellation.

Impact:
Direct theft of unclaimed yield through manipulation of the fee share distribution system.

Recommendation:
Implement a cancellation fee that is charged even when a request is cancelled. This would prevent cost-free manipulation of the queue and ensure that users cannot generate fee shares without contributing actual value to the protocol.

Developer Response:
Fixed at commit `90a9fe3`.

Medium Risk

M-01 - Inaccurate Undelegation Amount Leads to Withdrawal DoS

Severity: Medium Risk

Context:

- StakingVaultUpgradeable.sol#L508

Technical Details:
In the `StakingVaultUpgradeable` contract, there is an issue with how the undelegation total amount is calculated:

```
for (uint256 i; i < amounts.length; i++) {
      Wei amount = amounts[i] == WeiMath.ZERO
        ? WeiMath.ZERO
        : _wei.mulPercent(amounts[i].percentOf(totalAmount),
Math.Rounding.Ceil);

      amounts[i] = amount;
      total = total + amount;
    }
```

The `previewUndelegate` function returns a total that doesn't accurately represent the actual available amount. This inaccuracy can cause the subsequent check to pass even when there are insufficient funds to cover the request. When this happens, the L1 write operation will emit a request to undelegate, but it will silently fail on the L1 side because the requested amount is greater than what's actually available.

This creates a denial-of-service situation in the withdrawal queue:

1. Alice attempts to withdraw 100 HYPE, but the L1 operation fails silently due to only 99 HYPE being currently available.
2. Bob later attempts to withdraw 100 HYPE and his request passes.
3. Only 100 HYPE is available to withdraw, but both Alice and Bob are in the withdrawal queue.
4. One user will be unable to withdraw until more funds are undelegated.
5. This pattern continues, creating a situation where there are more pending withdrawals than available funds.

Impact:
Temporary DoS where users are dependent on the subsequent withdrawer to undelegate more funds so that they access their own funds.

Recommendation:
Before calculating the allocation truncate `_wei` to ensure it is not greater than the available `totalAmount`.

Developer Response:
Fixed at commit `c39f176`.

M-02 - Temporary Share Price Inflation via Refund Logic

Severity: Medium Risk

Context:

- StakingVaultUpgradeable.sol#L312

Technical Details:
In the `deposit` function of the `StakingVaultUpgradeable` contract, there is a vulnerability where an attacker can manipulate the share price by gaining control of execution flow.

When a user sends excess `msg.value` during a deposit, the contract will refund the excess amount, giving control back to the attacker while the contract is in a state where assets have been added but corresponding shares have not yet been minted. During this window:

1. The `totalAssets` function returns an inflated value as it includes the deposited assets
2. The attacker receives control via the refund transaction
3. The attacker can interact with protocols that integrate with this LST (Liquid Staking Token)
4. The attacker can sell their existing tokens at the inflated price to these protocols
5. Once execution returns to the deposit function and completes, shares are minted and the price normalizes

Impact:
An attacker can temporarily inflate the share price of the LST and exploit protocols that use this staking vault in their asset basket, potentially extracting value by selling at artificially high prices.

Recommendation:
Restructure the deposit function to mint shares before processing refunds.

Developer Response:
Fixed at commit `dbcf1a1`.

Low Risk

L-01 - Share Price Manipulation in Withdrawal Process

Severity: Low Risk

Context:

- StakingVaultUpgradeable.sol#L634

Technical Details:
In the `StakingVaultUpgradeable` contract, there's a potential issue with the order of operations during the withdrawal process:

```
    if (fee > 0) {
      payable(msg.sender).sendValue(fee);
    }

    _burn(address(this), shares);
  }
```

The fee is sent to the `msg.sender` before burning the shares, which creates a window where the caller can gain control while the share price is in an inaccurate state. If the WITHDRAW_ROLE is given to an arbitrary contract or is compromised, this could be exploited.

When the external call is made to send the fee, the `msg.sender` receives control while shares are still counted in `totalSupply` but assets have been removed. This temporarily makes the share price undervalued, allowing the caller to potentially:

1. Receive control during the external call
2. Deposit at the artificially low share price
3. Receive more shares than they should
4. Complete the original withdrawal transaction, burning the original shares

Impact:
A compromised account with the WITHDRAW_ROLE could potentially extract additional value by depositing at an artificially low share price during the withdrawal process.

Recommendation:
Reorder the operations to burn shares before sending the fee to the caller.

Developer Response:
Fixed at commit `ccf8387`.

Informational

I-01 - Unreachable Time Condition Check

Severity: Informational

Context:

- WithdrawQueueUpgradeable.sol#L365

Technical Details:
In the `WithdrawQueueUpgradeable` contract, there is a logical issue in the condition check:

```
return entry.eta == 0 || entry.eta > block.timestamp + 12 hours;
```

This condition is checking if either:

1. The entry's estimated time of arrival (eta) is 0, or
2. The entry's eta is more than 12 hours in the future

However, based on how the contract sets the `eta` value, this second condition is impossible to reach. The `eta` is only ever set to either 0 or the current `block.timestamp` at the time of setting, never a future timestamp. This means the `entry.eta > block.timestamp + 12 hours` part of the condition will always evaluate to false, making it unneeded.

Impact: Unneeded condition adds unnecessary complexity and impacts readability.

Recommendation:
Simplify the condition to only check for the reachable cases.

Developer Response:
Fixed at commit `587fa82`.

I-02 - Incorrect Exchange Rate Calculation

Severity: Informational

Context:

- WithdrawQueueUpgradeable.sol#L340

Technical Details:
In the `WithdrawQueueUpgradeable` contract, there is an issue with the exchange rate calculation:

```
uint256 exchangeRate = (entry.assets * 1e18) / entry.shares;
```

The code uses `entry.assets` for calculating the exchange rate, which represents the estimated asset value at the time of request creation. However, it should use the actual amount that was redeemed because in some cases the redeemed amount can be less than what was initially stored in `entry.assets`, which would provide the true exchange rate based on the actual redemption value rather than the initial estimate.

Using the initial estimate rather than the actual redemption amount can lead to inaccurate exchange rate calculations, particularly if market conditions or asset valuations change between the time of request and the time of processing.

Impact:
Slightly inaccurate exchange rate calculations that leads to misleading event emissions.

Recommendation:
Update the exchange rate calculation to use the actual redeemed amount rather than the initially estimated assets.

Developer Response:
Fixed at commit `587fa82`.

I-03 - Typo in Code Comment

Severity: Informational

Context:

- StakingVaultUpgradeable.sol#L676

Technical Details:
In the `StakingVaultUpgradeable` contract, there is a typo in a code comment:

```
// pull the assets form the owner, else pull from the msg.sender
```

The word "form" is incorrectly used instead of "from" in the comment.

Impact:
Purely a documentation issue that may slightly reduce code readability.

Recommendation:
Correct the typo in the comment to improve code documentation.

Developer Response:
Fixed at commit `74ec52d`.

I-04 - Inefficient Zero-Amount Undelegation Calls

Severity: Informational

Context:

- StakingVaultUpgradeable.sol#L508

Technical Details:
In the `StakingVaultUpgradeable` contract, there is an inefficiency in the undelegation process.
When undelegating from validators, the contract may make calls to undelegate zero amounts, particularly when referencing the active proxy. This happens because the code doesn't check if the amount is greater than zero before executing the undelegation call. These zero-amount calls waste gas and provide no functional benefit to the contract.

Impact:
Unnecessary gas consumption due to executing contract calls with zero amounts that have no effect on the system.

Recommendation:
Add a condition to check if the amount is greater than zero before executing the undelegation call.

Developer Response:
Fixed at commit `e663ec5`.

I-05 - Unallocated Assets When Last Validator Has Zero-Weight

Severity: Informational

Context:

- StakingVaultUpgradeable.sol#L467

Technical Details:
In the `StakingVaultUpgradeable` contract, there is an issue with the validator delegation logic:

```
if (configuration.targetWeighting == 0) {
```

The contract has special handling for the last validator in the list, which should receive any remaining assets (dust) after proportional delegation:

```
Wei delegation = i == length - 1 ? WeiMath.toWei(assets) - delegated : ...
```

However, if the last validator has a target weighting of zero, the function will skip it entirely due to an early return, preventing this special case from being executed. As a result, any dust amounts or remaining assets will not be delegated to any validator and will remain idle in the contract.

Impact:
Small amounts of assets remain unallocated and don't generate staking rewards, resulting in slightly lower overall returns for users.

Recommendation:
Because the unallocated amount will be delegated on the next deposit, documenting this behavior and avoiding configurations where the last validator is likely to have a target weight of 0 would be sufficient.

Developer Response:
Fixed at commit `fad54df`.

I-06 - Unnecessary Conversion in Delegate Function

Severity: Informational

Context:

- StakingVaultUpgradeable.sol#L325

Technical Details:
In the `StakingVaultUpgradeable` contract, there is an unnecessary conversion operation in the following line:

```
delegate(address(this).balance - $.totalClaimableAssets).fromWei();
```

The function call converts the result from `delegate` using the `.fromWei` method, but the returned value is never used or stored. This conversion has no effect on the contract's functionality and is redundant.

Impact:
Unnecessary gas consumption due to performing a conversion operation whose result is never used.

Recommendation:
Remove the `.fromWei` conversion since the result is not used, or assign the result to a variable if the value is needed elsewhere in the function. This would reduce gas costs and improve code readability.

Developer Response:
Fixed at commit `070939b`.

Disclaimer

This report does not endorse or critique any specific project or team. It does not assess the economic value or viability of any product or asset developed by parties engaging Enigma Dark for security assessments. We do not provide warranties regarding the bug-free nature of analyzed technology or make judgments on its business model, proprietors, or legal compliance.

This report is not intended for investment decisions or project participation guidance. Enigma Dark aims to improve code quality and mitigate risks associated with blockchain technology and cryptographic tokens through rigorous assessments.

Blockchain technology and cryptographic assets inherently involve significant risks. Each entity is responsible for conducting their own due diligence and maintaining security measures. Our assessments aim to reduce vulnerabilities but do not guarantee the security or functionality of the technologies analyzed.

This security engagement does not guarantee against a hack. It is a review of the codebase during a specific period of time. Enigma Dark makes no warranties regarding the security of the code and does not warrant that the code is free from defects. By deploying or using the code, the project and users of the contracts agree to use the code at their own risk. Any modifications to the code will require a new security review.