ENIGMA DARK

Securing the Shadows



Managed Security Review

Hyperdrive Subaccounts

August, 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 Subaccounts
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 2 days, beginning August 19 2025, Enigma Dark coordinated a Managed Security Review of the Hyperdrive Subaccounts project, specifically the contracts located at `/lending/contracts/protocol/accounts/**.sol`. The review was performed by one Security Researcher: kiki.

The following repositories were reviewed at the specified commits:

Repository	Commit
ambitfi/hyperdrive-contracts	`d9708959763c516d9ff1c4d0e45c02468df87287`

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	0	0	0
Medium	0	0	0
Low	2	2	0
Informational	8	6	2

Findings

Index	Issue Title	Status
L-01	Excess ETH Can Be Locked in SubAccountFactory	Fixed
L-02	Fee Receiver Zero Address Check Missing	Fixed
I-01	Approvals Not Reset After Function Calls	Acknowledged
I-02	Missing Owner Validation in transferOwner	Fixed
I-03	TransferOwner Event Missing New Key Parameter	Fixed
I-04	Return Variable Name Typo in getDeploymentFee	Fixed
I-05	Missing Revert Reasons Across Functions	Fixed
I-06	Free SubAccount Deployment Until Fee Set	Acknowledged
I-07	Unneeded SLOAD in SetLabel Event Emission	Fixed
I-08	Initial Owner Event Emission Uses Wrong Address	Fixed

Detailed Findings

High Risk

No issues found.

Medium Risk

No issues found.

Low Risk

L-01 - Excess ETH Can Be Locked in SubAccountFactory

Severity: Low Risk

Context:

- SubAccountFactory.sol#L126

Technical Details:
The `deploySubAccount` function in `SubAccountFactory` only validates that `msg.value` is greater than or equal to the deployment fee. When the `actions` array is empty, any excess ETH sent above the deployment fee remains locked in the contract. This occurs because the function forwards excess ETH to the sub-account only when executing actions. Without actions to execute, the excess ETH has no forwarding logic and becomes permanently trapped in the factory contract.

Impact:
Users who accidentally send more ETH than the deployment fee when creating sub-accounts without actions will permanently lose their excess funds.

Recommendation:
Add validation to ensure msg.value equals deploymentFee exactly when the actions array is empty.

Developer Response:
Fixed at commit `a72c85f`.

L-02 - Fee Receiver Zero Address Check Missing

Severity: Low Risk

Context:

- SubAccountFactory.sol#L60

Technical Details:
The `setDeploymentFee` function in `SubAccountFactory` allows setting a non-zero deployment fee with a zero address fee receiver. While the function validates that the fee is less than 100 ether, it does not verify that `feeReceiver` is a valid non-zero address when setting a non-zero fee. This means fees can be configured to be sent to the zero address, effectively burning the collected fees instead of directing them to a valid receiver.

Impact:
When fees are set with a zero address receiver, deployment fees are permanently lost instead of being collected by the protocol.

Recommendation:
Add a validation check to ensure `feeReceiver` is not the zero address when setting a non-zero deployment fee.

Developer Response:
Fixed at commit `a76e98c`.

Informational

I-01 - Approvals Not Reset After Function Calls

Severity: Informational

Context:

- RouterLib.sol#L97

Technical Details:
The `call` function in `RouterLib` executes calls to target contracts without resetting token approvals afterward. Unlike the original router functionality which automatically revoked approvals post-call, this implementation allows approvals to persist. When users execute actions that require token approvals, those approvals if not completely used remain active after the transaction completes.

Impact:
Leftover token approvals.

Recommendation:
Document the new approval behavior to ensure users understand they must explicitly revoke approvals.

Developer Response:
Acknowledged. Behaviour will be documented for implementor.

I-02 - Missing Owner Validation in transferOwner

Severity: Informational

Context:

- SubAccountFactory.sol#L153

Technical Details:
The `transferOwner` function in `SubAccountFactory` lacks two validation checks for the `newOwner` parameter. The function does not verify that the new owner address is non-zero or different from the current owner (`msg.sender`). This allows ownership transfers to the zero address, which effectively burns the sub-account and all funds within it. Transferring the account to the same owner allows the user to change their key which may not be desired functionality.

Impact:
Sub-accounts can be lost through transfers to the zero address.

Recommendation:
Add validation to ensure newOwner is not the zero address and is different from msg.sender.

Developer Response:
Fixed at commit `28545e7`.

I-03 - TransferOwner Event Missing New Key Parameter

Severity: Informational

Context:

- SubAccountFactory.sol#L162

Technical Details:
The `transferOwner` function in `SubAccountFactory` emits a `TransferOwner` event that omits the `newKey` parameter. When ownership is transferred, the sub-account is remapped using a new key for the new owner, but this crucial information is not included in the event emission. This makes it difficult for off-chain systems to track the complete ownership transfer process, as they cannot easily determine the new key associated with the transferred sub-account without additional queries.

Impact:
Limited sub-account ownership transfers tracking.

Recommendation:
Include the newKey parameter in the TransferOwner event emission to provide complete ownership transfer data.

Developer Response:
Fixed at commit `998e391`.

I-04 - Return Variable Name Typo in getDeploymentFee

Severity: Informational

Context:

- SubAccountFactory.sol#L66

Technical Details:
The `getDeploymentFee` function in `SubAccountFactory` contains a typo in its return variable name. The function declares a return variable named `mount` instead of the intended `amount`. While this does not affect functionality since the variable name is only used in the function signature, it impacts code readability and documentation clarity.

Recommendation:
Rename the return variable from `mount` to `amount` to maintain clear and consistent naming conventions.

Developer Response:
Fixed at commit `13298b1`.

I-05 - Missing Revert Reasons Across Functions

Severity: Informational

Context:

- SubAccountFactory.sol#L56
- SubAccountFactory.sol#L122
- SubAccountFactory.sol#L151
- SubAccountFactory.sol#L152
- RouterLib.sol#L33

Technical Details:
Multiple validation checks across the codebase use bare `require` statements without error messages. These checks include deployment fee validation, sub-account key uniqueness verification, ownership validation, and address validation. When these validations fail, transactions revert without providing any context about the failure reason.

Recommendation:
Add revert reasons to all require statements to clearly indicate the reason for transaction failures.

Developer Response:
Fixed at commit `146d6eb`.

I-06 - Free SubAccount Deployment Until Fee Set

Severity: Informational

Context:

- SubAccountFactory.sol#L50

Technical Details:
The `SubAccountFactory` contract's `initialize` function does not set an initial deployment fee or fee receiver. The deployment fee mechanism is designed to control and monetize sub-account creation, but since `deploymentFee` and `deploymentFeeReceiver` are left uninitialized, users can deploy sub-accounts for free until an admin explicitly sets

these values through `setDeploymentFee`. This creates a window where the intended fee collection mechanism is bypassed, allowing unrestricted sub-account creation without any cost.

Impact:
Protocol loses revenue as users can deploy sub-accounts.

Recommendation:
Add deployment fee and fee receiver parameters to the initialize function to ensure fee collection starts immediately upon contract deployment.

Developer Response:
Acknowledged. Fees will not be enabled initially.

I-07 - Unneeded SLOAD in SetLabel Event Emission

Severity: Informational

Context:

- SubAccount.sol#L46

Technical Details:
The `initialize` function in the `SubAccount` contract performs an unnecessary storage load when emitting the `SetLabel` event. The function already has the `owner` parameter in memory, but it loads the same value from storage using `$.owner`. This creates an extra SLOAD operation that increases gas costs without providing any benefit.

Impact:
Each sub-account initialization consumes more gas than necessary due to redundant storage reads, increasing deployment costs for users.

Recommendation:
Use the `owner` parameter directly in the `SetLabel` event emission instead of reading from storage with `$.owner`.

Developer Response:
Fixed at commit `3beab7a`.

I-08 - Initial Owner Event Emission Uses Wrong Address

Severity: Informational

Context:

- SubAccount.sol#L45

Technical Details:
The `initialize` function in the `SubAccount` contract emits a `SetOwner` event with incorrect parameters. When initializing a new sub-account, the event uses `$.owner` as the previous owner parameter, but at this point in the initialization, `$.owner` is the same as the new owner being set since the storage update happens before the event emission. For the first-time owner setting, the previous owner should be `address(0)` to accurately represent that there was no previous owner.

Impact:
Misleading event logs

Recommendation:
Replace `$.owner` with `address(0)` in the `SetOwner` event emission during initialization.

Developer Response:
Fixed at commit `5491c3d`.

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.