ENIGMA DARK

Securing the Shadows



Security Review

Arrakis Modular

Uniswap V4 Hook Integration

21 November 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

Arrakis Modular: Uniswap V4 Hook Integration
Arrakis Modular is a DEX-agnostic liquidity management framework built around Meta-Vaults and standardized modules. It enables integration with any two-token DEX (like Uniswap V4, Balancer, Ambient) through reusable, and upgradeable components. Designed for flexibility and scalability, it supports both public and private vaults, simplifies liquidity provisioning, and lays the groundwork for advanced strategies across DeFi.

Engagement Overview

Over the course of 3 days, beginning November 21 2025, the Enigma Dark team conducted a security review of the Arrakis Modular: Uni v4 Hook Integration project. The review was performed by two Lead Security Researchers: 0xWeiss & vnmrtz.

The following repositories were reviewed at the specified commits:

Repository	Commit
ArrakisFinance/arrakis-modular	`3df9ac1681aa7e5c4caa14b73737c01321145c85`

Scope

```
src/
├── hooks/
|   └── ArrakisPrivateHook.sol
├── interfaces/
|   ├── IArrakisPrivateHook.sol
|   └── IFantomV4Swap.sol
└── utils/
    └── FantomV4Swap.sol
```

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	1	1	0
Low	3	2	1
Informational	0	0	0

Findings

Index	Issue Title	Status
M-01	Malicious executor can store an arbitrary module inside the fee data mapping	Fixed
L-01	Executor-controlled immediate fee updates enable fee-based front-running	Acknowledged
L-02	Public vault executor can act as an executor in the ArrakisPrivateHook	Fixed
L-03	Unnecessary function to fetch FeesData	Fixed

Detailed Findings

High Risk

No issues found.

Medium Risk

M-01 - Malicious executor can store an arbitrary module inside the fee data mapping

Severity: Medium Risk

Context:

- ArrakisPrivateHook.sol#L79

Technical Details:

In the `setFees` function, the executor can pass a module address inside the FeesData struct. This module is then called to fetch the vault address from the module and it validates that the executor from the vault is the same as msg.sender:

```
function setFees(
 PoolKey calldata poolKey_,
    FeesData calldata data_
) external {
    if (data_.module == address(0)) {
        revert AddressZero();
    }

    address _vault =
        address(IArrakisLPModule(data_.module).metaVault());

    (,,,, address executor,,,) =
        IArrakisStandardManager(manager).vaultInfo(_vault);

    if (msg.sender != executor) {
        revert OnlyVaultExecutor();
    }
```

There is no validation though that the module is active or non-malicious as the executor can create a malicious module that when called it returns a valid `_vault` address in the system that returns their address as the executor.

After, the malicious module will be stored in the FeesData mapping:

```
feesData[poolId] = FeesData({
            module: data_.module,
            zeroForOneFee: zeroForOneFee,
            oneForZeroFee: oneForZeroFee
        });
```

Impact:

The executor is able to introduce a custom module as the module to be used for the specified pool.

Recommendation:

Add a validation to check that the module stored in the vault is the same one as the supplied by the executor:

```
  function setFees(
        PoolKey calldata poolKey_,
        FeesData calldata data_
    ) external {
        if (data_.module == address(0)) {
            revert AddressZero();
        }

        address _vault =
            address(IArrakisLPModule(data_.module).metaVault());

        (,,,, address executor,,,) =
            IArrakisStandardManager(manager).vaultInfo(_vault);

        if (msg.sender != executor) {
            revert OnlyVaultExecutor();
        }

+       if(address(_vault.module()) != data_.module) {
+                revert NotActiveModule();
+            }

        // #region checks if fees are valid.

        data_.zeroForOneFee.validate();
```

```
        data_.oneForZeroFee.validate();

        // #endregion checks if fees are valid.

        PoolId poolId = poolKey_.toId();

        uint24 zeroForOneFee =
            data_.zeroForOneFee | LPFeeLibrary.OVERRIDE_FEE_FLAG;
        uint24 oneForZeroFee =
            data_.oneForZeroFee | LPFeeLibrary.OVERRIDE_FEE_FLAG;

        feesData[poolId] = FeesData({
            module: data_.module,
            zeroForOneFee: zeroForOneFee,
            oneForZeroFee: oneForZeroFee
        });

        emit SetFees(
            poolId,
            data_.module,
            data_.zeroForOneFee,
            data_.oneForZeroFee
        );
    }
```

Developer Response:

Fixed at commit: `e673b07`.

Low Risk

L-01 - Executor-controlled immediate fee updates enable fee-based front-running

Severity: Low Risk

Context:

- ArrakisPrivateHook.sol#L70

Technical Details:

Under the semi-trusted executor assumption, the `setFees` function allows the vault executor to update swap fees immediately and symmetrically for both directions.

Fee updates take effect on the very next swap. This means an executor observing a pending transaction in the mempool could temporarily raise fees just before the trade executes, then revert them immediately afterward.

Impact:

This behavior enables a form of front-running where external traders are subjected to an unexpected fee spike, extracting value at their expense. This creates a griefing and fairness risk.

Recommendation:

Consider introducing a timelock or short delay on fee updates, so changes cannot be applied to the immediately following swap.

Developer Response:

Acknowledged. We view this as a trader-side risk, mitigated by using a minimum amount out. This mechanism is intended for high-frequency, RFQ-style dynamic fees on L2s, and adding a timelock would reduce the required flexibility.

L-02 - Public vault executor can act as an executor in the ArrakisPrivateHook

Severity: Low Risk

Context:

- ArrakisPrivateHook.sol#L81

Technical Details:

The `ArrakisPrivateHook` is meant to be only used with private vaults and modules. In the `setFees` function, there is no validation that the vault from a certain module is a private vault. In the scenario where the executor is a different address for public and private vaults, the executor from a public vault could call `setFees` and update the fee mapping:

```
  address _vault =
            address(IArrakisLPModule(data_.module).metaVault());

        (,,,, address executor,,,) =
            IArrakisStandardManager(manager).vaultInfo(_vault);

        if (msg.sender != executor) {
            revert OnlyVaultExecutor();
        }
```

Impact:

In the scenario where the executor for a private and public vault are different, the public vault executor can update fee data for a private hook.

Recommendation:

Add the following validation:

```
  address _vault =
            address(IArrakisLPModule(data_.module).metaVault());

+    if (!factory.isPrivateVault(_vault)) {
+            revert NotPrivateVault();
+        }

        (,,,, address executor,,,) =
            IArrakisStandardManager(manager).vaultInfo(_vault);

        if (msg.sender != executor) {
            revert OnlyVaultExecutor();
        }
```

Developer Response:

Fixed at commit: `22c1a28`.

L-03 - Unnecessary function to fetch FeesData

Severity: Low Risk

Context:

- ArrakisPrivateHook.sol#L55

Technical Details:

The `feesData` mapping is public which allows any external integration/user to call it to fetch data for a certain pool Id:

```
mapping(PoolId => FeesData) public feesData;
```

this is redundant as there is a specific function that returns the exact same thing:

```
 function getFeesData( PoolId id_ )
        external view returns ( address module, uint24 zeroForOneFee, uint24
oneForZeroFee ) {
        return (
            feesData[id_].module,
            feesData[id_].zeroForOneFee.removeOverrideFlag(),
            feesData[id_].oneForZeroFee.removeOverrideFlag()
        );
    }
```

Impact:

Two different functions with the exact same functionality will be declared in the contract, which cause more deployment costs and maybe be misleading. Additionally, the decoding of the `zeroForOneFee` and `oneForZeroFee` values will be incorrect when being fetch from the `feesData` mapping directly.

Recommendation:

Update the visibility of the mapping:

```
-  mapping(PoolId => FeesData) public feesData;
+ mapping(PoolId => FeesData) private feesData;
```

Developer Response:

Fixed at commit: `948e9b0`.

Informational

No issues found.

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.