

ENIGMA DARK

Securing the Shadows



Security Review
Arrakis Modular
Uniswap V4 & PancakeSwap V4 modules

28 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

Arrakis Modular: PancakeSwap V4 & Uniswap V4 Modules

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 weeks, beginning April 28 2025, the Enigma Dark team conducted a security review of the Arrakis Modular: PancakeSwap V4 & Uniswap V4 Modules project. The review was performed by one Lead Security Researcher: 0xWeiss, and one Security Researcher: kiki.

The following repositories were reviewed at the specified commits:

Repository	Commit
ArrakisFinance/arrakis-modular	a0f994c9f79e57468b66cd97f5a9ed37ecef770d

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	4	3	1
Informational	4	4	0

Findings

Index	Issue Title	Status
M-01	Modules are missing storage gaps	Fixed
L-01	Missing event emission in the <code>withdrawEth</code> function	Fixed
L-02	Max Deviation Can Be Exceeded	Fixed
L-03	Executor can make pool swap unfeasible	Acknowledged
L-04	Fee Split Can Be Gamed	Fixed
I-01	All specific PancakeSwap NatSpec its wrong	Fixed
I-02	Redundant check on deposits	Fixed
I-03	Inefficient manager withdrawal functions	Fixed
I-04	Event Emission Does Not Account For Inverse Case	Fixed

Detailed Findings

High Risk

No issues found.

Medium Risk

M-01 - Modules are missing storage gaps

Severity: Medium Risk

Context:

- [UniV4StandardModule.sol#L75](#)
- [PancakeSwapV4StandardModule.sol#L72](#)

Technical Details:

The contracts [UniV4StandardModule.sol](#) and [PancakeSwapV4StandardModule.sol](#) are being inherited by multiple upgradeable contracts.

In upgradeable contracts, it's crucial to include a `gap` variable to ensure that any additional storage variables added in future contract upgrades do not collide with existing storage variables.

Impact:

Corrupted upgradeability.

Recommendation:

Consider adding a [gap variable](#) to future-proof base contract storage changes and be safe against storage collisions.

Very important that the size of the `__gap` array is calculated so that the amount of storage used by the contract always adds up to the same number (in this case 50 storage slots).

Developer Response:

Fixed at commit `d644a99` and `5dd7b69`.

Low Risk

L-01 - Missing event emission in the `withdrawEth` function

Severity: Low Risk

Context:

- [UniV4StandardModule.sol#L299](#)

Technical Details:

The `withdrawEth` function allows for users with a balance in the `ethWithdrawers` mapping over 0 to withdraw ETH from the contract. The issue is that there is no way to index this action properly as there is no event being emitted.

On the other hand, in the `withdraw` function, which withdraws `token0` and `token1`, an event to reflect such withdrawal is being emitted, therefore, it should also be emitted inside the `withdrawEth` function.

Impact:

Missing indexing for ETH withdrawals

Recommendation:

Emit an event at the end of the `withdrawEth` function:

```
function withdrawEth(
    uint256 amount_
) external nonReentrant whenNotPaused {
    if (amount_ == 0) revert AmountZero();
    if (ethWithdrawers[msg.sender] < amount_) {
        revert InsufficientFunds();
    }

    ethWithdrawers[msg.sender] -= amount_;
    payable(msg.sender).sendValue(amount_);
+   emit LogWithdrawETH(msg.sender, amount_);
}
```

Developer Response:

Fixed at commit `d644a99` .

L-02 - Max Deviation Can Be Exceeded

Severity: Low Risk

Context:

- [UniV4StandardModule.sol#L771](#)

Technical Details:

The Arrakis protocol includes a safety mechanism to prevent price manipulation during rebalances. This is implemented through the `validateRebalance()` function which checks if the current pool price deviates too much from the oracle price. The deviation check is for protecting against sandwich attacks and other forms of price manipulation.

The deviation calculation is performed using `FullMath.mulDiv()` which rounds down by default:

```
uint256 deviation = FullMath.mulDiv(
    FullMath.mulDiv(
        poolPrice > oraclePrice
            ? poolPrice - oraclePrice
            : oraclePrice - poolPrice,
        10 ** token1Decimals,
        poolPrice
    ),
    PIPS,
    10 ** token1Decimals
);
```

The issue is that the deviation calculation rounds down, which means that a price deviation that is slightly above the maximum allowed deviation could be rounded down to exactly the maximum deviation, causing the check to pass when it should fail.

Impact:

Rebalances can occur with slightly higher price deviations than what should be allowed.

Recommendation:

When calculating the deviation round up to ensure that rebalances do not exceed the intended max.

Developer Response:

Fixed at commit `d644a99` .

L-03 - Executor can make pool swap unfeasible

Severity: Low Risk

Context:

- [UniV4StandardModule.sol#L508](#)

Technical Details:

The executor has the power to rebalance liquidity across different ranges within the vault. This capability grants the executor access to move all funds in the vault. However, multiple safeguards are in place to mitigate risks from a malicious executor, including checks to ensure both the token amounts stay within specified ranges and the notional value of the vault remains controlled.

One edge case not currently covered involves attacks where the executor renders swapping on the pool infeasible. This can be achieved by pushing all liquidity for token0 and token1 to their respective extremes. This action results in prohibitively high token prices with minimal liquidity available between these extremes.

If such an attack occurs, the pool would become temporarily unusable, requiring the vault owner to assign a new executor and rebalance the pool with accessible liquidity ranges.

Impact:

Temporary Denial of Service.

Recommendation:

Consider allowing the owner control over maximum and minimum tick ranges (in exception for full range LP positions). Otherwise document the behavior to inform users of the full risks associated with a potentially malicious executor.

Developer Response:

Acknowledged, we will inform users through our documentation.

L-04 - Fee Split Can Be Gamed

Severity: Low Risk

Context:

- [UniswapV4.sol#L707](#)
- [PancakeSwapV4.sol#L733](#)

Technical Details:

When fees for providing liquidity are collected, the protocol will take a portion of this through Manager Fees. This is done as follows:

```
uint256 managerFee0 = FullMath.mulDiv(
    withdraw_.fee0, managerFeePIPS, PIPS
);
uint256 managerFee1 = FullMath.mulDiv(
    withdraw_.fee1, managerFeePIPS, PIPS
);
```


The issue, however, is that the manager fee rounds down, which means that fee splits favor an arbitrary vault instead of the Arrakis protocol. In most cases, this would be a small and negligible amount. However, in cases where one of the tokens has low decimals and a high notional value, the current fee split design could be leveraged to redirect a worthwhile amount of funds away from Arrakis.

Take, for example:

- A vault has WBTC as one of its tokens, which has 8 decimals and is worth ~\$100,000.
- `managerFeePIPS` is 1%.

In a case where the vault has earned 0.00000099 WBTC, worth ~\$0.1 this amount of WBTC would round down to zero.

```
fees0 * managerFeePIPS / PIPS
99 * 1_0000 / 1_000_000
990_000 / 1_000_000
0.99 => 0
```

Resulting in the protocol not collecting any revenue. A savvy vault owner could leverage this knowledge and repeatedly claim fees right as they approach 99, earning an extra \$0.1 each time.

Impact:

Loss of protocol revenue.

Recommendation:

Round in favor of the protocol by rounding up when determining manager fees. This will need to be done each time `managerFeePIPS` is used to determine fee amount.

Developer Response:

Fixed at commit `d644a99` .

Informational

I-01 - All specific PancakeSwap NatSpec its wrong

Severity: Informational

Context:

- [PancakeSwapV4StandardModule.sol#L220](#)
- [PancakeSwapV4StandardModule.sol#L225](#)
- [PancakeSwapV4StandardModule.sol#L348](#)

Technical Details:

Most of the NatSpec that should mention PancakeSwap specific functionalities, does mention UniSwap instead, which is incorrect and misleading.

Recommendation:

Grep all the contracts that should reference pancake swap for the word `uni` . Update all those instances to reference pancake swap.

Developer Response:

Fixed at commits `d644a99` and `c5c64a7` .

I-02 - Redundant check on deposits

Severity: Informational

Context:

- [UniV4StandardModulePrivate.sol#L70](#)

Technical Details:

The `fund` function checks whether the depositor is `address(0)`, if so, it reverts:

```
function fund(
    address depositor_,
    uint256 amount0_,
    uint256 amount1_
) external payable onlyMetaVault whenNotPaused nonReentrant {
    // #region checks.

>>    if (depositor_ == address(0)) revert AddressZero();

    if (amount0_ == 0 && amount1_ == 0) revert DepositZero();
```

This is unfeasible as the `depositor_` address is forwarded from the vault contract where is always set to be `msg.sender` :

```
function _deposit(
    uint256 amount0_,
    uint256 amount1_
) internal nonReentrant {

    bytes memory data = abi.encodeWithSelector(
        IArrakisLPModulePrivate.fund.selector,msg.sender,amount0_,amount1_);

    payable(address(module)).functionCallWithValue(data, msg.value );
}
```

Recommendation:

Remove the `address(0)` check:

```
function fund(
    address depositor_,
    uint256 amount0_,
    uint256 amount1_
) external payable onlyMetaVault whenNotPaused nonReentrant {
    // #region checks.

-    if (depositor_ == address(0)) revert AddressZero();

    if (amount0_ == 0 && amount1_ == 0) revert DepositZero();
```

Developer Response:

Fixed at commit `d644a99` .

I-03 - Inefficient manager withdrawal functions

Severity: Informational

Context:

- [UniV4StandardModule.sol#L791](#)

Technical Details:

There are currently two functions (`managerBalance0` and `managerBalance1`) that allow the manager to withdraw their balance of two different tokens, `token0` and `token1`.

Both functions use exactly the same code with exception of the last couple lines of code.

Recommendation:

Merge both functions into one:

```
- function managerBalance1() external view returns (uint256 managerFee1) {
+ function managerBalance(bool token0) external view returns (uint256
managerFee) {
    PoolKey memory _poolKey = poolKey;
    PoolRange[] memory poolRanges =
        UniswapV4._getPoolRanges(_ranges, _poolKey);

    (uint256 leftOver0, uint256 leftOver1) =
        IUniV4StandardModule(this)._getLeftOvers(_poolKey);

    (uint160 sqrtPriceX96_,,,) =
        poolManager.getSlot0(PoolIdLibrary.toId(_poolKey));

    (,, uint256 fee0, uint256 fee1) = UnderlyingV4
        .totalUnderlyingAtPriceWithFees(
        UnderlyingPayload({
            ranges: poolRanges,
            poolManager: poolManager,
            self: address(this),
            leftOver0: leftOver0,
            leftOver1: leftOver1
        })),
        sqrtPriceX96_
    );

+ if (token0){
+     managerFee0 = isInversed ? FullMath.mulDiv(fee1, managerFeePIPS, PIPS)
: FullMath.mulDiv(fee0, managerFeePIPS, PIPS);
+ } else {
+     managerFee1 = isInversed ? FullMath.mulDiv(fee0, managerFeePIPS, PIPS)
: FullMath.mulDiv(fee1, managerFeePIPS, PIPS);
+ }
```

Developer Response:

Fixed at commit `d644a99` .

I-04 - Event Emission Does Not Account For Inverse Case

Severity: Informational

Context:

- [PancakeSwapV4.sol#L364](#)

Technical Details:

The `PancakeSwapV4` library incorrectly emits the manager fee when `isInversed` is true. Unlike in the `UniswapV4` library, where prior to emitting the `LogWithdrawManagerBalance` event, it checks if `isInversed` is true, `PancakeSwapV4` does not have this same check.

```
if (managerFee0 > 0 || managerFee1 > 0) {  
    emit IArrakisLPModule.LogWithdrawManagerBalance(manager, managerFee0,  
managerFee1);  
}
```

This means when the manager claims fees earned from `PancakeSwap`, the event emission will output the wrong amount for each token. This could lead to confusion for 3rd parties.

Impact:

Incorrect event emission.

Recommendation:

Match the `UniswapV4` library by emitting `LogWithdrawManagerBalance` after taking the manager fee and applying the same inverse logic.

Developer Response:

Fixed at commit `d644a99`.

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.