CANTINA

Uniswap Protocol Fees

Security Review

Cantina Managed review by:
Sujith Somraaj, Lead Security Researcher
0xWeiss, Security Researcher
Red Swan, Security Researcher Manager

November 25, 2025

Contents

1 Introduction **2**
1.1 About Cantina . . . 2
1.2 Disclaimer . . . 2
1.3 Risk assessment . . . 2
1.3.1 Severity Classification . . . 2

2 Security Review Summary **3**
2.1 Scope . . . 3

3 Findings **4**
3.1 Low Risk . . . 4
3.1.1 Unchecked Fee Parameter in `setDefaultFeeByFeeTier()` Causes Downstream Pool Reverts and Protocol Fee Update DoS . . . 4
3.1.2 Pools of assets with very low volume could make fees sit in the contract indefinitely unless the threshold is updated . . . 4
3.2 Gas Optimization . . . 5
3.2.1 Unnecessary Type Casting in `collect()` Function . . . 5
3.3 Informational . . . 5
3.3.1 Missing Docstrings . . . 5
3.3.2 Missing `address(0)` Validation Across Multiple Critical Functions . . . 6
3.3.3 Residual Tokens May Remain Unclaimed When Vesting Amount Is Updated . . . 6
3.3.4 Missing event emission for key storage changes . . . 7
3.3.5 Single-step ownership transfer . . . 7
3.3.6 Unconfigured default fee values for fee tiers . . . 7

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity level	**Impact: High**	**Impact: Medium**	**Impact: Low**
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings are a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

Uniswap is an open source decentralized exchange that facilitates automated transactions between ERC20 token tokens on various EVM-based chains through the use of liquidity pools and automatic market makers (AMM).

From Nov 16th to Nov 19th the Cantina team conducted a review of protocol-fees on commit hash a35efd21. The team identified a total of **9** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	0	0	0
Low Risk	2	1	1
Gas Optimizations	1	1	0
Informational	6	2	4
Total	**9**	**4**	**5**

2.1 Scope

The security review had the following components in scope for protocol-fees on commit hash a35efd21:

```
src
├── base
│   ├── Nonce.sol
│   └── ResourceManager.sol
├── Deployer.sol
├── feeAdapters
│   └── V3FeeAdapter.sol
├── libraries
│   └── ArrayLib.sol
├── releasers
│   ├── ExchangeReleaser.sol
│   ├── Firepit.sol
│   └── OptimismBridgedResourceFirepit.sol
├── TokenJar.sol
└── UNIVesting.sol
```

3 Findings

3.1 Low Risk

3.1.1 Unchecked Fee Parameter in `setDefaultFeeByFeeTier()` Causes Downstream Pool Reverts and Protocol Fee Update DoS

Severity: Low Risk

Context: V3FeeAdapter.sol#L98

Description: The `V3FeeAdapter.setDefaultFeeByFeeTier()` function accepts any `uint8` value (0-255) for `defaultFeeValue` without validating that it conforms to the constraints required by the Uniswap V3 pool's `setFeeProtocol()` function. When `_setProtocolFee()` is called, the stored `defaultFeeValue` is decomposed into two 4-bit components:

```
IUniswapV3PoolOwnerActions(pool).setFeeProtocol(feeValue % 16, feeValue >> 4);
```

However, the Uniswap V3 pool's `setFeeProtocol()` function enforces strict validation:

```
require(
      (feeProtocol0 == 0 || (feeProtocol0 >= 4 && feeProtocol0 <= 10)) &&
          (feeProtocol1 == 0 || (feeProtocol1 >= 4 && feeProtocol1 <= 10))
  );
```

If an invalid `defaultFeeValue` is set (e.g., 255 → decomposes to 15, 15), all calls to `_setProtocolFee()` for that fee tier will revert. Protocol fee updates are temporarily unavailable for the affected fee tier until corrected.

Recommendation: Add validation in `setDefaultFeeByFeeTier()` to ensure the `defaultFeeValue` will decompose into valid components.

```
  function setDefaultFeeByFeeTier(uint24 feeTier, uint8 defaultFeeValue) external
   ↪  onlyFeeSetter {
        require(_feeTierExists(feeTier), InvalidFeeTier());

+        // Extract the two 4-bit values
+        uint8 feeProtocol0 = defaultFeeValue % 16;
+        uint8 feeProtocol1 = defaultFeeValue >> 4;

+        // Validate both values match pool requirements: must be 0 or in range [4, 10]
+        require(
+            (feeProtocol0 == 0 || (feeProtocol0 >= 4 && feeProtocol0 <= 10)) &&
+            (feeProtocol1 == 0 || (feeProtocol1 >= 4 && feeProtocol1 <= 10)),
+            InvalidFeeValue()
+        );

        defaultFees[feeTier] = defaultFeeValue;
    }
```

Uniswap Labs: Fixed in PR 101.

Cantina Managed: Fix verified.

3.1.2 Pools of assets with very low volume could make fees sit in the contract indefinitely unless the threshold is updated

Severity: Low Risk

Context: ExchangeReleaser.sol#L39

Description: The release function intends to serve as an exchange function where the searchers need to send the `threshold` of the UNI token in exchange for a basket of assets. These assets are the tokens which were claimed as fees and now they are resting in the TokenJar contract.

```
function release(uint256 _nonce, Currency[] calldata assets, address recipient)
    external
```

```
  handleNonce(_nonce)
{
  require(assets.length <= MAX_RELEASE_LENGTH, TooManyAssets());
  RESOURCE.safeTransferFrom(msg.sender, RESOURCE_RECIPIENT, threshold);
  TOKEN_JAR.release(assets, recipient);
  _afterRelease(assets, recipient);
}
```

It is expected to have a very large variety of tokens inside TokenJar. As discussed, the Uniswap team looked at all of the pools from univ3 that have 90% or over market share over other DEXs, which amounts to thousands of pools. These pools should be the ones included inside the merkle tree.

Each pool has its volume and fees from pools with very low volume will be minimal. Given there will be so many assets, and the `MAX_RELEASE_LENGTH = 20` there will be a situation for multiple tokens where its simply not worth release as the value of UNI that needs to be burned in exchange for the tokens released its superior, leaving such tokens with value inside the contract until it is profitable to release them, if that ever happens.

Recommendation: This is mostly an architectural issue. Increasing the value of `MAX_RELEASE_LENGTH` could help but it should not be an overly big value either. We recommend to document and understand that this will happen and take into account that it could need a governance vote at some point to lower the threshold low enough so that its profitable to sweep some of these low value tokens to then increase the threshold again.

Uniswap Labs: Acknowledged. Documented in PR 101.

Cantina Managed: Acknowledged.

3.2 Gas Optimization

3.2.1 Unnecessary Type Casting in `collect()` Function

Severity: Gas Optimization

Context: V3FeeAdapter.sol#L83-L87

Description: The `collect()` function performs unnecessary type conversions that waste gas. The `IUniswapV3PoolOwnerActions.collectProtocol()` function returns (uint128, uint128), but the return values are assigned to uint256 variables and then immediately cast back down to uint128 when populating the Collected struct.

Recommendation: Declare the return variables as uint128 to match the actual return type from `collectProtocol()`, eliminating both casting operations.

Uniswap Labs: Fixed in PR 101.

Cantina Managed: Fix verified.

3.3 Informational

3.3.1 Missing Docstrings

Severity: Informational

Context: Nonce.sol#L6, Deployer.sol#L13, V3FeeAdapter.sol#L71, ArrayLib.sol#L4

Description: Throughout the codebase, multiple instances of missing docstrings were identified:

- In `ArrayLib.sol`, the ArrayLib library is completely undocumented, including the `includes()` function.
- In `Deployer.sol`, the Deployer contract is also completely undocumented, including all the state variables and constants.
- In `Nonce.sol`, the Nonce contract is missing @title and @notice for the contract. Also, the modifier handleNonce is undocumented.

- In `V3Adapter.sol`, the function `setFactoryOwner()` is missing docs alongside most of the internal functions including `_setProtocolFeesForPair()`, `_setProtocolFee()`, `_doubleHash()` and `_feeTierExists()`.

Recommendation: Thoroughly document all functions and the contract mentioned above.

Uniswap Labs: Fixed in PR 101.

Cantina Managed: Fix verified.

3.3.2 Missing `address(0)` Validation Across Multiple Critical Functions

Severity: Informational

Context: ResourceManager.sol#L40, V3FeeAdapter.sol#L71, V3FeeAdapter.sol#L104, ExchangeReleaser.sol#L39, TokenJar.sol#L40, UNIVesting.sol#L73

Description: Multiple functions across the protocol lack `address(0)` validation for critical address parameters. Functions with higher impact *(would recommend fixing)*:

1. `TokenJar.sol`: `setReleaser(address _releaser)`: No validation before setting the releaser address. If the releaser is set to `address(0)`, then funds in the token jar will be temporarily locked.
2. `UNIVesting.sol`: `updateRecipient(address _recipient)`: No validation before updating the vesting recipient. Setting the recipient to `address(0)` is equivalent to burning UNI tokens intended for the Uniswap Labs team.
3. `ExchangeReleaser.sol`: `release(uint256 _nonce, Currency[] calldata assets, address recipi`
   No validation of recipient parameter, could lead to searchers losing funds.
4. `V3FeeAdapter.sol`: `setFactoryOwner(address newOwner)`: No validation before transferring factory ownership, could permanently lock claiming fees from Uni V3.
5. `ResourceManager.sol`: `setThresholdSetter(address _thresholdSetter)`: No validation before setting threshold setter.
6. `V3FeeAdapter.sol`: `setFeeSetter(address newFeeSetter)`: No validation before setting the fee setter.

Recommendation: Consider adding `address(0)` validation to all affected functions.

Uniswap Labs: Acknowledged. Decided not to do this.

Cantina Managed: Acknowledged.

3.3.3 Residual Tokens May Remain Unclaimed When Vesting Amount Is Updated

Severity: Informational

Context: UNIVesting.sol#L98

Description: The `updateVestingAmount()` function allows the owner to change the quarterly vesting amount, but only when `quartersPassed() == 0` (i.e., no quarters are currently available to claim).

However, when the vesting amount is changed, any tokens previously approved by the owner that don't align with the new quarterly amount become difficult to claim and may remain permanently unclaimed without manual intervention.

Consider the following timeline:

1. Owner sets initial quarterlyVestingAmount = 5,000,000 UNI.
2. Owner approves 40,000,000 UNI to the contract.
3. Recipient withdraws 6 quarters: 6 × 5,000,000 = 30,000,000 UNI.
4. Owner updates quarterlyVestingAmount to 3,000,000 UNI (perhaps due to budget changes).
5. Recipient withdraws 3 more quarters: 3 × 3,000,000 = 9,000,000 UNI.
6. Total withdrawn: 39,000,000 UNI.

7. Remaining allowance: 1,000,000 UNI - temporarily stuck unless the owner changes the quarterlyVestingAmount to 1,000,000 UNI (or) approves 2,000,000 more UNI tokens.

Recommendation: Consider documenting this limitation and always ensure that the owner changes the quarterlyVestingAmount carefully to not run into these kinds of edge cases.

Alternatively, add a sweep function, allowing the recipient to sweep any residual approvals.

Uniswap Labs: Acknowledged. Added documentation explaining this behavior in PR 101.

Cantina Managed: Acknowledged.

3.3.4 Missing event emission for key storage changes

Severity: Informational

Context: V3FeeAdapter.sol#L94, V3FeeAdapter.sol#L100, V3FeeAdapter.sol#L105, TokenJar.sol#L41

Description: Multiple functions across the protocol lack event emissions for key state changes:

1. `V3Adapter.sol`: `setMerkleRoot`, `setFeeSetter`, `setDefaultFeeByFeeTier`.
2. `TokenJar.sol`: `setReleaser`.
3. `ResourceManager.sol`: `setThresholdSetter`, `setThreshold`.

Recommendation: Emit an event to reflect the state changes for each one of the above functions.

Uniswap Labs: Acknowledged. We don't think any of these events are necessary, the combination of events we currently have + transfers is sufficient for expected data needs.

Cantina Managed: Acknowledged.

3.3.5 Single-step ownership transfer

Severity: Informational

Context: V3FeeAdapter.sol#L4, V3FeeAdapter.sol#L21, TokenJar.sol#L13, UNIVesting.sol#L17

Description: In the following contracts:

- `V3FeeAdapter.sol`.
- `TokenJar.sol`.
- `UNIVesting.sol`.
- `ResourceManager.sol`.

A single-step ownership transfer pattern is used, where if a mistake is made when transferring ownership there will not be a way of reverting such state.

It is a recommended practice to use a 2-step ownership transfer pattern where the address that has been set as the ”pending owner” needs to accept ownership first, before officially transferring the ownership to that addres.

Recommendation: Use Ownable2Step from Open Zeppelin as your inherited ownership contract.

Uniswap Labs: Acknowledged. Decided not to do this, since governance is the owner of most of these and we dont want a vote to accept ownership from the DAO.

Cantina Managed: Acknowledged.

3.3.6 Unconfigured default fee values for fee tiers

Severity: Informational

Context: Deployer.sol#L60

Description: The `Deployer` contract stores fee tiers in the `V3FeeAdapter` but fails to configure the corresponding default fee values for these tiers before transferring ownership to the DAO. This oversight leaves all protocol fees effectively disabled (set to 0%) until the DAO addresses this through a time-delayed governance process.

Recommendation: Configure default fee values for all stored fee tiers in the `Deployer` constructor before transferring ownership.

Uniswap Labs: Fixed in commit b8d55465.

Cantina Managed: Fix verified.