Virtuals v4 Hook Contract

Security Review

Cantina Managed review by:

R0bert, Lead Security Researcher
Kaden, Security Researcher

June 10, 2025

Contents

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	Description
Critical	*Must* fix as soon as possible (if already deployed).
High	Leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
Medium	Global losses <10% or losses to only a subset of users, but still unacceptable.
Low	Losses will be annoying but bearable. Applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.
Gas Optimization	Suggestions around gas saving practices.
Informational	Suggestions around best practices or readability.

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 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

Virtuals Protocol is The Wall Street for AI Agents.

From Jun 5th to Jun 6th the Cantina team conducted a review of virtuals-v4-hook-contract on commit hash 8352cc53. The team identified a total of **10** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	1	1	0
High Risk	0	0	0
Medium Risk	2	1	1
Low Risk	2	1	1
Gas Optimizations	3	2	1
Informational	2	0	2
Total	**10**	**5**	**5**

3 Findings

3.1 Critical Risk

3.1.1 HookData can be spoofed to bypass the 1% fee

Severity: Critical Risk

Context: AutoTaxHook.sol#L222-L226

Description: In `_afterSwap` the contract attempts to detect its own "internal-conversion swap" and skip taxation by encoding its address in `hookData`:

```
// Skip taxation if this is our own internal swap
address user = parseHookData(hookData);
if (user == address(this)) {
    return (IHooks.afterSwap.selector, int128(0));
}
```

Because the `hookData` is supplied by any external caller of `PoolManager.swap`, a trader can simply use the same payload:

```
bytes hookData = abi.encode(address(autoTaxHook));  // spoofed
poolManager.swap(key, params, hookData);            // pays 0 % tax
```

The hook believes the call is internal, executes the early-return path shown above, and the fee is never charged. All subsequent logic, including the event emission that would normally reveal a fee, is skipped.

Recommendation: Do not derive privilege from user-controlled calldata. In the `Hooks.sol` library, used by `PoolManager`, we skip the `afterSwap` hook if `msg.sender == address(self)`:

```
/// @notice calls afterSwap hook if permissioned and validates return value
function afterSwap(
    IHooks self,
    PoolKey memory key,
    IPoolManager.SwapParams memory params,
    BalanceDelta swapDelta,
    bytes calldata hookData,
    BeforeSwapDelta beforeSwapHookReturn
) internal returns (BalanceDelta, BalanceDelta) {
    // @audit we skip the afterSwap hook if the sender is the hooks contract
    if (msg.sender == address(self)) return (swapDelta, BalanceDeltaLibrary.ZERO_DELTA);
```


As a result, we don't need any logic to handle this case in `_afterSwap` and can simply remove the provided logic entirely:

```
- // Skip taxation if this is our own internal swap
- address user = parseHookData(hookData);
- if (user == address(this)) {
-     return (IHooks.afterSwap.selector, int128(0));
- }
```

Virtuals: Fixed in commit 7adbcd9.

Cantina Managed: Fix verified.

3.2 Medium Risk

3.2.1 Initialization can be blocked via frontrunning

Severity: Medium Risk

Context: *(No context files were provided by the reviewer)*

Description: In `_afterInitialize`, we check whether a `poolKey` has already been initialized and revert if so:

```
function _afterInitialize(
    address,
    PoolKey calldata key,
    uint160,
    int24
) internal override returns (bytes4) {
    if (poolKeyInitialized) revert PoolKeyAlreadyInitialized();

    // Validate that the pool contains our destination token
    if (!_isPoolSupported(key)) revert UnsupportedPool();

    poolKey = key;
    poolKeyInitialized = true;
    return BaseHook.afterInitialize.selector;
}
```

Since anyone can initialize a pool with this hook, as long as the pool contains the `destinationToken` as one of its tokens, an attacker can initialize a different pool from the intended one before it is initialized with the intended pool, resulting in a denial of service.

Recommendation: We can prevent this either by:

- Initializing the intended pool within the same transaction that we deploy `AutoTaxHook.sol`, or.
- Enforcing that the caller of `PoolManager.initialize` is an authorized account, e.g. the `owner` of this contract, which we can do by checking the `sender` parameter of `afterInitialize`.

Virtuals: Acknowledged.

Cantina Managed: Acknowledged.

3.2.2 Inconsistent destinationToken handling leaves dead code paths

Severity: Medium Risk

Context: AutoTaxHook.sol#L141

Description: The constructor explicitly forbids using native ETH (`address(0)`) as the `destinationToken`:


```
constructor(
    IPoolManager _poolManager,
    address _owner,
    address _destinationToken
) BaseHook(_poolManager) Ownable(_owner) {
    if (_owner == address(0)) revert InvalidOwner();
    if (_destinationToken == address(0)) revert InvalidDestinationToken();  // <-- hard-rejection
    destinationToken = _destinationToken;
}
```

However, two later code paths contain dedicated logic for handling ETH:

```
function _safeSendDestinationTokenToTaxRecipient() internal {
    // ...
    if (destinationToken == address(0)) {                // ETH branch
        (bool success, ) = payable(taxRecipient).call{
            value: balance,
            gas: GAS_LIMIT
        }("");
        if (!success) revert TransferFailed();
    } else {
        IERC20(destinationToken).safeTransfer(taxRecipient, balance);
    }
}

function _safeSwapToDestinationToken(/*...*/) internal {
    // ...
    if (feeCurrency.isAddressZero()) {                   // ETH branch
        poolManager.settle{value: feeAmount}();
    } else {
        IERC20(Currency.unwrap(feeCurrency)).safeTransfer(
            address(poolManager),
            feeAmount
        );
        poolManager.settle();
    }
}
```

Because the constructor refuses `address(0)`, these ETH branches are unreachable as `address(0)` is used to represent native assets in the Uniswap V4 `Currency` library:

```
// @notice A constant to represent the native currency
Currency public constant ADDRESS_ZERO = Currency.wrap(address(0));
```

Recommendation: Consider removing the `if (_destinationToken == address(0)) revert InvalidDestinationToken();` check from the constructor.

Virtuals: Fixed in commit 7adbcd9.

Cantina Managed: Fix verified.

3.3 Low Risk

3.3.1 Skipping small fee amounts may allow for fee avoidance with low-decimal tokens

Severity: Low Risk

Context: *(No context files were provided by the reviewer)*

Description: In `_afterSwap`, in case the `feeAmount` is less than the `MIN_FEE_AMOUNT` (1000), we skip fee payment entirely:

```
// Skip if fee is too small to prevent precision loss attacks
if (feeAmount < MIN_FEE_AMOUNT) {
    return (IHooks.afterSwap.selector, int128(0));
}
```

The risk here is that if the `feeAmount` is not denominated in the `destinationToken`, this may lead us to skipping non-dust amounts of fees if the token has little enough decimals. For example:

- 1000 wei of WBTC (8 decimals) = 0.00001 WBTC ~= $1.
- 1000 wei of GUSD (2 decimals) = 10 GUSD ~= $10.
- 1000 wei of USDC (6 decimals) = 0.001 USDC ~= $0.001.

For the most part, these are relatively small amounts, but in case the gas cost of swapping is less than the fee amount, it can be profitable to split up a larger swap into many smaller ones to avoid paying the fee, resulting in a loss for the protocol.

Recommendation: This can be resolved via computing the `feeAmount` denominated in the `destinationToken` and enforcing a minimum on that amount. Alternatively, it may also be sufficient to only create pools with tokens with sufficiently high decimal amounts such that this attack is not profitable.

Virtuals: Acknowledged.

Cantina Managed: Acknowledged.

3.3.2 Incorrect error

Severity: Low Risk

Context: *(No context files were provided by the reviewer)*

Description: In `emergencyWithdrawERC20`, in case `address(0)` is provided as the `token` to withdraw, we revert with the `InvalidTaxRecipient` error:

```
function emergencyWithdrawERC20(
    address token
) external onlyOwner nonReentrant {
    if (token == address(0)) revert InvalidTaxRecipient();
```

This is likely not the intended error. And should be replaced to better represent the reason for the revert.

Recommendation: Replace the error with one which better describes the reason for the revert:

```
-if (token == address(0)) revert InvalidTaxRecipient();
+if (token == address(0)) revert InvalidToken();
```

Virtuals: Fixed in commit 7adbcd9.

Cantina Managed: Fix verified.

3.4 Gas Optimization

3.4.1 Redundant initialized pool check

Severity: Gas Optimization

Context: *(No context files were provided by the reviewer)*

Description: In `_afterSwap`, we validate that the pool we are swapping through is the initialized pool:

```
// Validate that the swap is happening on our initialized pool
if (!poolKeyInitialized || !_isPoolEqual(key, poolKey)) {
    return (IHooks.afterSwap.selector, int128(0));
}
```

However, since this function can only be called by the `PoolManager` with a pool initialized with this hook contract and only the pool initialized with this hook contract can be the `poolKey`, this `if` statement will always evaluate to false. As a result, this check is redundant.

Recommendation: Remove the check entirely:

```
- // Validate that the swap is happening on our initialized pool
- if (!poolKeyInitialized || !_isPoolEqual(key, poolKey)) {
-     return (IHooks.afterSwap.selector, int128(0));
- }
```

Virtuals: Fixed in 7adbcd9.

Cantina Managed: Fix verified.

3.4.2 Use denominator to calculate fee

Severity: Gas Optimization

Context: *(No context files were provided by the reviewer)*

Description: In `_calculateFee`, we compute the `feeAmount` as follows:

```
feeAmount = (uint128(swapAmount) * TAX_RATE_BIPS) / TOTAL_BIPS;
```

Since we use constants for `TAX_RATE_BIPS` (100) and `TOTAL_BIPS` (10000), this logic is identical to simply dividing the `swapAmount` by 100 directly:

$$x * \frac{100}{10000} = x * 0.01 = \frac{x}{100}$$

Recommendation: Replace the use of `TAX_RATE_BIPS` and `TOTAL_BIPS` with a constant e.g., `TAX_RATE_DENOMINATOR = 100`, to represent a denominator to divide `swapAmount` by directly:

```
- feeAmount = (uint128(swapAmount) * TAX_RATE_BIPS) / TOTAL_BIPS;
+ feeAmount = uint128(swapAmount) / TAX_RATE_DENOMINATOR;
```

Virtuals: Acknowledged.

Cantina Managed: Acknowledged.

3.4.3 Redundant `unlockCallback`

Severity: Gas Optimization

Context: *(No context files were provided by the reviewer)*

Description: We include an `unlockCallback` function in `AutoTaxHook.sol`. However, this function is only necessary for handling executing calls on the `PoolManager` after calling `PoolManager.unlock`. Since we don't call `PoolManager.unlock` at any point, this function is redundant.

Recommendation: Remove the `unlockCallback` function entirely:

```
- /**
-  * @dev Secure unlock callback implementation
-  */
- function unlockCallback(
-     bytes calldata
- ) external view onlyPoolManager returns (bytes memory) {
-     // Validate that this is expected unlock context
-     return "";
- }
```


Virtuals: Fixed in commit 7adbcd9.

Cantina Managed: Fix verified.

3.5 Informational

3.5.1 Lack of a double-step transfer ownership pattern

Severity: Informational

Context: AutoTaxHook.sol#L24

Description: The `AutoTaxHook` contract is using the standard OpenZeppelin's Ownable library. The standard OpenZeppelin's Ownable contract allows transferring the ownership of the contract in a single step:

```
/**
 * @dev Transfers ownership of the contract to a new account (`newOwner`).
 * Can only be called by the current owner.
 */
function transferOwnership(address newOwner) public virtual onlyOwner {
    if (newOwner == address(0)) {
        revert OwnableInvalidOwner(address(0));
    }
    _transferOwnership(newOwner);
}

/**
 * @dev Transfers ownership of the contract to a new account (`newOwner`).
 * Internal function without access restriction.
 */
function _transferOwnership(address newOwner) internal virtual {
    address oldOwner = _owner;
    _owner = newOwner;
    emit OwnershipTransferred(oldOwner, newOwner);
}
```

If the nominated EOA account is not a valid account, it is entirely possible that the owner may accidentally transfer ownership to an uncontrolled account, losing the access to all functions with the `onlyOwner` modifier.

Recommendation: It is recommended to implement a two-step transfer process where the owner nominates an account and the nominated account needs to call an `acceptOwnership()` function for the transfer of the ownership to fully succeed. This ensures the nominated EOA account is a valid and active account. A good code example could be OpenZeppelin's Ownable2Step contract:

```
/**
 * @dev Starts the ownership transfer of the contract to a new account. Replaces the pending transfer if there
 ↪  is one.
 * Can only be called by the current owner.
 *
 * Setting `newOwner` to the zero address is allowed; this can be used to cancel an initiated ownership
 ↪  transfer.
 */
function transferOwnership(address newOwner) public virtual override onlyOwner {
    _pendingOwner = newOwner;
    emit OwnershipTransferStarted(owner(), newOwner);
}

/**
 * @dev Transfers ownership of the contract to a new account (`newOwner`) and deletes any pending owner.
 * Internal function without access restriction.
 */
function _transferOwnership(address newOwner) internal virtual override {
    delete _pendingOwner;
    super._transferOwnership(newOwner);
}

/**
 * @dev The new owner accepts the ownership transfer.
 */
function acceptOwnership() public virtual {
    address sender = _msgSender();
    if (pendingOwner() != sender) {
        revert OwnableUnauthorizedAccount(sender);
    }
    _transferOwnership(sender);
}
```

Virtuals: Acknowledged.

Cantina Managed: Acknowledged.

3.5.2 Missing tests in AutoTaxHook

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: The codebase ships without any unit, integration or fuzz tests. Without an automated test suite, core behaviours (fee charging, internal swaps, pausing, admin actions, etc.) remain unverified, making regressions or hidden logic errors far more likely to reach production.

Recommendation: We have implemented a set of basic tests (exactIn and exactOut swaps) that can be expanded in gist c6135e59.

Virtuals: Acknowledged.

Cantina Managed: Acknowledged.