



# Security Assessment Final Report



## M Extension

July 2025

Prepared for M^0 Labs

# Table of content

<b>Project Summary.....</b>	<b>3</b>
Project Scope.....	3
Project Overview.....	3
Protocol Overview.....	3
Findings Summary.....	4
Severity Matrix.....	4
<b>Detailed Findings.....</b>	<b>5</b>
<b>High Severity Issues.....</b>	<b>7</b>
H-01. MYieldFee extension may accrue yield before earning is enabled and after it is disabled.....	7
H-02. wrappedM whitelist enables role bypass for unwrapping M tokens.....	11
<b>Medium Severity Issues.....</b>	<b>14</b>
M-01. swapIn function sends tokens to the wrong recipient in multi-hop swaps.....	14
M-02. Old fee recipient remain whitelisted with zero fee rate after the update in MEarnnerManager.....	16
<b>Low Severity Issues.....</b>	<b>17</b>
L-01. Missing disableInitializers() in upgradeable contracts.....	17
L-02. Hardcoded Uniswap pool fee.....	18
L-03. Accrued yield may be redirected to the wrong recipient when changing claim recipients.....	19
L-04. Possible MExtension yield lock-up in UniswapV3 pools.....	21
L-05. Possible overminting of MExtension tokens due to out-of-sync indexing.....	22
<b>Informational Issues.....</b>	<b>23</b>
I-01. Storage access inconsistency.....	23
I-02. NatSpec comment in _revertIfNotBlacklisted is misleading.....	23
I-03. Remove redundant Interface IUniswapV3SwapCallback.....	24
I-04. Remove or utilize unused events SwappedIn and SwappedOut.....	24
I-05. Unused exactOutput functionality.....	24
<b>Disclaimer.....</b>	<b>26</b>
<b>About Certora.....</b>	<b>26</b>

# Project Summary

## Project Scope

Project Name	Repository (link)	Latest Commit Hash	Platform
M Extension framework	<a href="https://github.com/m0-foundation/m-m-extensions">https://github.com/m0-foundation/m-m-extensions</a>	<a href="#">66eb471</a> (original commit) <a href="#">ba39e69</a> (fix commit)	Solidity

## Project Overview

This document describes the manual code review findings of the **M Extension framework**. The following contract list is included in our scope:

`m0-foundation/m-m-extensions/tree/main/src/*`

The work was undertaken from **June 25, 2025**, to **July 6, 2025**. During this time, Certora's security researchers performed a manual audit of all the Solidity contracts and discovered several bugs in the codebase, which are summarized in the subsequent section.

## Protocol Overview

The M Extension framework provides a system for creating non-rebasing wrapper tokens around the rebasing M token, with three different yield distribution mechanisms: directing all yield to a single account (**YieldToOne**), distributing yield to all holders with fee collection (**YieldToAllWithFee**), and tiered yield splitting for whitelisted accounts (**EarnManager**). The framework includes a **SwapFacility** for seamless conversions between extensions and M tokens, along with a **Uniswap V3 adapter** enabling direct swaps with external tokens like USDC.

# Findings Summary

The table below summarizes the findings of the review, including type and severity details.

Severity	Discovered	Confirmed	Fixed
Critical	-	-	-
High	2	2	1
Medium	2	2	1
Low	5	5	2
Informational	5	5	5
Total	14	14	9

# Severity Matrix

Impact	High	Medium	High	Critical
	Medium	Low	Medium	High
	Low	Low	Low	Medium
		Low	Medium	High
Likelihood				

# Detailed Findings

ID	Title	Severity	Status
<a href="#">H-01</a>	MYieldFee extension may accrue yield before earning is enabled and after it is disabled	High	Fixed
<a href="#">H-02</a>	wrappedM whitelist enables role bypass for unwrapping M tokens	High	Acknowledged
<a href="#">M-01</a>	swapIn function sends tokens to wrong recipient in multi-hop swaps	Medium	Fixed
<a href="#">M-02</a>	Old fee recipient remain whitelisted with zero fee rate after the update in MEarnnerManager	Medium	Acknowledged
<a href="#">L-01</a>	Missing disableInitializers() in upgradeable contracts	Low	Fixed
<a href="#">L-02</a>	Hardcoded Uniswap pool fee	Low	Acknowledged
<a href="#">L-03</a>	Accrued yield may be redirected to wrong recipient when changing claim recipients	Low	Fixed
<a href="#">L-04</a>	Possible MExtension yield lock-up in UniswapV3 pools	Low	Acknowledged
<a href="#">L-05</a>	Possible overminting of MExtension tokens due to out-of-sync indexing	Low	Acknowledged
<a href="#">I-01</a>	Storage access inconsistency	Informational	Fixed
<a href="#">I-02</a>	NatSpec comment in	Informational	Fixed

	<code>_revertIfNotBlacklisted</code> is misleading		
<a href="#">I-03</a>	Remove redundant Interface <code>IUniswapV3SwapCallback</code>	Informational	Fixed
<a href="#">I-04</a>	Remove or utilize unused events <code>SwappedIn</code> and <code>SwappedOut</code>	Informational	Fixed
<a href="#">I-05</a>	Unused <code>exactOutput</code> functionality	Informational	Fixed

## High Severity Issues

### H-01. MYieldFee extension may accrue yield before earning is enabled and after it is disabled

Severity: <b>High</b>	Impact: <b>High</b>	Likelihood: <b>Medium</b>
Files: <a href="#">src/projects/yieldToAllWithFee/MYieldFee.sol</a>	Status: Fixed	

**Description:** The **MYieldFee** extension is designed to accrue yield only after it is added to the TTG Registrar Earners list and the **enableEarning()** function is called. To manage this behavior, the contract provides explicit functions to enable and disable earning. When earning is disabled, yield accrual should stop completely, and when enabled, yield should begin accruing only from that point onward.

However, two issues have been identified in the current implementation that break that intention:

1. **Yield accrual after disabling earning:** When **disableEarning()** is called, the **latestRate** is reset (deleted) to indicate earning has stopped.

JavaScript

```
/// @inheritdoc IMExtension
function disableEarning() external override {
    if (!isEarningEnabled()) revert EarningIsDisabled();

    // NOTE: update the index to store the latest state.
    emit EarningDisabled(updateIndex());

    // NOTE: `latestRate` is set to 0 to indicate that earning is disabled.
    @> delete _getMYieldFeeStorageLocation().latestRate;

    IMTokenLike(mToken()).stopEarning(address(this));
}
```

Despite this, the `updateIndex()` function resets the `latestRate` again regardless of the earning status if called in a later block. This effectively re-enables yield accrual even though earning was meant to be disabled.

JavaScript

```
function updateIndex() public virtual returns (uint128 currentIndex_) {
    // NOTE: Read the current M token rate adjusted by fee rate split.
    uint32 rate_ = earnerRate();

    MYieldFeeStorageStruct storage $ = _getMYieldFeeStorageLocation();

    if ($.latestUpdateTimestamp == block.timestamp && $.latestRate == rate_) return
$.latestIndex;

    // NOTE: `currentIndex()` depends on `_latestRate`, so only update it after this.
    $.latestIndex = currentIndex_ = currentIndex();
    @> $.latestRate = rate_;
    $.latestUpdateTimestamp = _latestEarnerRateAccrualTimestamp();

    emit IndexUpdated(currentIndex_, rate_);
}
```

**2. Yield accrual before enabling earning:** It is also possible for `updateIndex()` to be called before `enableEarning()` is invoked, causing the yield to start accruing prematurely without the explicit activation step.

Both issues break the intended control flow by allowing yield to accrue outside the authorized earning state, which can lead to inconsistent contract behavior and incorrect accounting.

### PoC:

The following two test cases demonstrate the issues described above:

JavaScript

```
function test_resetEarning() public {
    _addToList(EARNERS_LIST, address(mYieldFee));
    mYieldFee.enableEarning();

    vm.warp(vm.getBlockTimestamp() + 10 days);
}
```



```
_removeFomList(EARNERS_LIST, address(mYieldFee));
mYieldFee.disableEarning();
assertTrue(!mYieldFee.isEarningEnabled());
assertEq(mYieldFee.latestRate(), 0);

vm.warp(vm.getBlockTimestamp() + 10 days);
mYieldFee.updateIndex();
assertGt(mYieldFee.latestRate(), 0);
assertTrue(mYieldFee.isEarningEnabled());
}
```

JavaScript

```
function test_enableEarningBeforeStartEarning() public {
    assertTrue(!mYieldFee.isEarningEnabled());
    assertEq(mYieldFee.latestRate(), 0);

    mYieldFee.updateIndex();
    assertGt(mYieldFee.latestRate(), 0);
    assertTrue(mYieldFee.isEarningEnabled());
}
```

**Recommendations:** Update `updateIndex()` to first check whether earning is currently enabled, and skip all yield calculations and state updates if earning is disabled. This ensures yield only accrues during authorized periods.

JavaScript

```
function updateIndex() public virtual returns (uint128 currentIndex_) {
+   MYieldFeeStorageStruct storage $ = _getMYieldFeeStorageLocation();
+   if (!isEarningEnabled()) return $.latestIndex;

    // NOTE: Read the current M token rate adjusted by fee rate split.
    uint32 rate_ = earnerRate();

-   MYieldFeeStorageStruct storage $ = _getMYieldFeeStorageLocation();
}
```

```
    if ($.latestUpdateTimestamp == block.timestamp && $.latestRate == rate_) return  
    $.latestIndex;  
  
    // NOTE: `currentIndex()` depends on `_latestRate`, so only update it after this.  
    $.latestIndex = currentIndex_ = currentIndex();  
    $.latestRate = rate_;  
    $.latestUpdateTimestamp = _latestEarnerRateAccrualTimestamp();  
  
    emit IndexUpdated(currentIndex_, rate_);  
}
```

**Customer's response:** Fixed in [d7df864](#).

## H-02. wrappedM whitelist enables role bypass for unwrapping M tokens

Severity: <b>High</b>	Impact: <b>High</b>	Likelihood: <b>Medium</b>
Files: <a href="#">src/swap/SwapFacility.sol</a> <a href="#">src/swap/UniswapV3SwapAdapter.sol</a>	Status: Acknowledged	

### Description:

The `SwapFacility` contract uses the `M_SWAPPER_ROLE` to restrict unwrapping of M tokens via the `swapOutM()` function. However, this check is not enforced in other swap functions like `swapOutToken()`.

This would normally not be a problem, except that `wrappedM` is whitelisted as a swap token. If `wrappedM` is whitelisted, users can unwrap every extension token and take out the underlying M token.

The flow is as follows:

1. The attacker holds a balance in an MExtension token (e.g., `MEarnerManager`)
2. He calls `swapOutToken` with (`extensionIn = MEarnerManager`, `tokenOut = wrappedM`)
3. Since `extensionIn != baseToken`, the call enters the else branch in `swapOutToken`
4. This triggers the internal `_swap()` function, which unwraps `MEarenerManger` extension token → sends the underlying M tokens to `SwapFacility` and wraps them into `wrappedM`
5. Finally, the attacker performs a multi-hop swap using the following path: `wrappedM` → `USDC` → `wrappedM`

The team confirms their intention to initially whitelist `wrappedM`, `USDC` and `USTD` and this can also be confirmed in the [BaseIntegrationTest](#).

This effectively allows arbitrary users to unwrap M tokens from any extension in which they hold a balance, breaking the intended unwrapping permission model.

### PoC:

The following test demonstrates the issue (paste it in the `test/integration/MEarnerManager.t.sol`):

JavaScript

```
function test_swapOutToken_extentionToWrappedM() public {
    _addToList(EARNERS_LIST, address(mEarnerManager));
    mEarnerManager.enableEarning();

    vm.prank(earnerManager);
    mEarnerManager.setAccountInfo(alice, true, 100);
    vm.prank(earnerManager);
    mEarnerManager.setAccountInfo(bob, true, 100);

    uint256 amount = 10e6;

    vm.startPrank(alice);
    mToken.approve(address(swapFacility), amount);
    swapFacility.swapInM(address(mEarnerManager), amount, alice);

    vm.warp(vm.getBlockTimestamp() + 10 days);
    bytes memory path = abi.encodePacked(
        WRAPPED_M,
        uint24(100),
        USDC,
        uint24(100),
        WRAPPED_M
    );

    swapFacility.swapOutToken(
        address(mEarnerManager),
        mEarnerManager.balanceOf(alice),
        WRAPPED_M,
        0,
        alice,
        path
    );
}
```

**Recommendations:** The mitigation is straightforward. Do not whitelist the `wrappedM` token in `UniswapV3SwapAdapter` as it is the `baseToken`.

Alternatively, if support for `wrappedM` must be retained for other use cases, consider guarding `_swap()` more strictly based on the caller context.

**Customer's response:** Acknowledged.

**Fix Review:** The team plans to upgrade **WrappedM**, after which the described scenario will no longer be possible. This intent was not documented during the audit, which led to the finding being reported.

## Medium Severity Issues

### M-01. swapIn function sends tokens to the wrong recipient in multi-hop swaps

Severity: **Medium**

Impact: **Medium**

Likelihood: **Medium**

Files:

[src/swap/UniswapV3SwapAdapter.sol](#)

Status: Fixed

**Description:** The `swapIn()` function in `UniswapV3SwapAdapter.sol` contains a recipient inconsistency between single-hop and multi-hop swap execution paths. When a custom path is provided (multi-hop swap), tokens are incorrectly sent to `msg.sender` instead of the intended `recipient` parameter.

JavaScript

```
function swapIn(
    address inputToken,
    uint256 inputAmount,
    uint256 minBaseAmount,
    address recipient,
    bytes calldata path
) external returns (uint256 baseAmount) {
    _revertIfNotWhitelistedToken(inputToken);
    _revertIfZeroAmount(inputAmount);
    _revertIfInvalidSwapInPath(inputToken, path);
    _revertIfZeroRecipient(recipient);

    // Transfer token input from sender to this contract
    IERC20(inputToken).safeTransferFrom(msg.sender, address(this), inputAmount);

    address swapRouter_ = swapRouter;

    // Approve the router to spend token input
    IERC20(inputToken).forceApprove(swapRouter_, inputAmount);

    // Swap token input for base token
    if (path.length == 0) {
```

```
IV3SwapRouter.ExactInputSingleParams memory params =
IV3SwapRouter.ExactInputSingleParams({
    tokenIn: inputToken,
    tokenOut: baseToken,
    fee: UNISWAP_V3_FEE,
    recipient: recipient,
    amountIn: inputAmount,
    amountOutMinimum: minBaseAmount,
    sqrtPriceLimitX96: 0
});

baseAmount = IV3SwapRouter(swapRouter_).exactInputSingle(params);
} else {
IV3SwapRouter.ExactInputParams memory params = IV3SwapRouter.ExactInputParams({
    path: path,
    recipient: msg.sender,
    amountIn: inputAmount,
    amountOutMinimum: minBaseAmount
});

baseAmount = IV3SwapRouter(swapRouter_).exactInput(params);
}
}
```

The primary caller `SwapFacility` passes `address(this)` as the recipient, making `msg.sender` and `recipient` identical in practice. But external callers or integrating protocols would experience unexpected behavior where tokens are delivered to the calling contract instead of the intended recipient.

**Recommendations:** Change `recipient: msg.sender` to `recipient: recipient` on line 94.

**Customer's response:** Fixed in [1c07be4](#).

## M-02. Old fee recipient remain whitelisted with zero fee rate after the update in MEarnnerManager

Severity: <b>Medium</b>	Impact: <b>High</b>	Likelihood: <b>Low</b>
Files: <a href="#">src/projects/earnerManager/MEarnnerManager.sol</a>	Status: Acknowledged	

**Description:** When a new fee recipient is set using `setFeeRecipient`, the previous one isn't updated. This means the old recipient might still be earning with a 0% fee, even though they're no longer the active fee recipient.

**Recommendations:** Before updating the fee recipient, reset the old one's status:

JavaScript

```
function _setFeeRecipient(address feeRecipient_) internal {
    if (feeRecipient_ == address(0)) revert ZeroFeeRecipient();

    MEarnnerManagerStorageStruct storage $ = _getMEarnnerManagerStorageLocation();

    if ($.feeRecipient == feeRecipient_) return;

+   _setAccountInfo($.feeRecipient, false, 0);

    // Yield fee recipient does not pay fees.
    _setAccountInfo(feeRecipient_, true, 0);

    $.feeRecipient = feeRecipient_;

    emit FeeRecipientSet(feeRecipient_);
}
```

**Customer's response:** Acknowledged.



## Low Severity Issues

### L-01. Missing `disableInitializers()` in upgradeable contracts

Severity: <b>Low</b>	Impact: <b>Medium</b>	Likelihood: <b>Low</b>
Files: <a href="#">src/projects/yieldToOne/MYieldToOne.sol</a> <a href="#">src/projects/earnerManager/MEarnerManager.sol</a> <a href="#">src/projects/yieldToAllWithFee/MYieldFee.sol</a>	Status: Fixed	

**Description:** `MYieldToOne`, `MEarnerManager`, `MYieldFee` are meant to be upgradeable. It is considered best practice to call `_disableInitializers()` in the constructor to prevent the implementation contract from being initialized directly:

[https://docs.openzeppelin.com/upgrades-plugins/writing-upgradeable#initializing\\_the\\_implementation\\_contract](https://docs.openzeppelin.com/upgrades-plugins/writing-upgradeable#initializing_the_implementation_contract)

**Recommendations:** Add `_disableInitializers` to the constructors of all upgradeable contracts:

```
JavaScript
constructor() {
  _disableInitializers();
}
```

**Customer's response:** Fixed in [7c229fe](#).

## L-02. Hardcoded Uniswap pool fee

Severity: **Low**

Impact: **Medium**

Likelihood: **Low**

Files:  
[src/swap/UniswapV3SwapAdapter.sol](#)

Status: Acknowledged

**Description:** The Uniswap V3 fee tier is hardcoded to **100** (0.01%) in **UniswapV3SwapAdapter**. While this is suitable for stable pairs, it may cause inefficiencies or failed swaps if more liquid pools exist at other fee tiers (e.g. 500 or 3000).

**Recommendations:** Allow the fee tier to be configurable or passed in, to support routing through pools with better liquidity.

**Customer's response:** Acknowledged. At this stage SwapAdapter is meant to work only using WrappedM – USDC pool, which has 0.01% fee.

### L-03. Accrued yield may be redirected to the wrong recipient when changing claim recipients

Severity: **Low**

Impact: **Medium**

Likelihood: **Low**

Files:  
[src/projects/yieldToAllWithFee/MYieldFee.sol](#)

Status: Fixed

**Description:** In `MYieldFee.setClaimRecipient()`, when a claim recipient is changed for an account, any previously accrued yield is not automatically claimed for the current recipient before the change. This means yield that accrued while Alice was the designated recipient could be claimed by Bob if he becomes the new recipient before the yield is claimed.

JavaScript

```
function setClaimRecipient(
    address account,
    address claimRecipient
) external onlyRole(CLAIM_RECIPIENT_MANAGER_ROLE) {
    if (account == address(0)) revert ZeroAccount();
    if (claimRecipient == address(0)) revert ZeroClaimRecipient();

    MYieldFeeStorageStruct storage $ = _getMYieldFeeStorageLocation();

    if ($.claimRecipients[account] == claimRecipient) return;

    // Optionally consider claiming yield for the previous claim recipient.
    // claimYieldFor(account);

    $.claimRecipients[account] = claimRecipient;

    emit ClaimRecipientSet(account, claimRecipient);
}
```

Consider a scenario where Alice is set as the claim recipient for the User's account and the user accrues 100 tokens of yield over 6 months. If an admin then changes the claim recipient to Bob via `setClaimRecipient()`, when `claimYieldFor()` is eventually called, Bob will receive the 100

tokens that accrued while Alice was the designated recipient. This results in previous claim recipients losing yield that logically belonged to them during their tenure and creates potential unfairness in yield distribution based on the timing of administrative actions.

**Recommendations:** Uncomment the `claimYieldFor(account)` call and make `claimYieldFor()` function `public`, to automatically claim accrued yield for the current recipient before changing to a new one.

**Customer's response:** Fixed in [e1e3188](#).

#### L-04. Possible MExtension yield lock-up in UniswapV3 pools

Severity: <b>Low</b>	Impact: <b>Medium</b>	Likelihood: <b>Low</b>
Files: <a href="#">src/projects/yieldToAllWithFee/MYieldFee.sol</a> <a href="#">src/projects/earnerManager/MEarnerManager.sol</a>	Status: Acknowledged	

**Description:** Contracts `MYieldFee.sol` and `MEarnerManager.sol` are MExtensions. Both contracts enable the earning of MTokens using `enableEarning()` functionality. During this period of continuous indexing, all MExtension token holders earn yield in MExtension tokens equivalent to the MToken yield balance generated in the contract.

This yield can be permissionlessly claimed by anyone for any account using the `MYieldFee.claimYieldFor()` or `MEarnerManager.claimFor()` functions. However, when a UniswapV3 pool is created for a MExtension and stablecoin like USDC, any MExtension tokens provided as liquidity to the pool will start earning yield. Since anyone can permissionlessly call `claimYieldFor()` and `claimFor()` on the MExtension for the uniswap pool (`account`), this would lock-up the yield in the pool permanently.

**Recommendations:** It is recommended to have a specific `claimRecipient` in `MYieldFee.sol` for such pools. For `MEarnerManager.sol`, it is recommended to divert 100% of the yield to the fee recipient, which can then be distributed to the LPs separately.

**Customer's response:** Acknowledged. The purpose of `claimRecipient` is to solve this situation.

**L-05. Possible overminting of MExtension tokens due to out-of-sync indexing**Severity: **Low**Impact: **Medium**Likelihood: **Low**

Files:  
[src/projects/yieldToAllWithFee/MYieldFee.sol](#)

Status: Acknowledged

**Description:** Let's take this scenario to understand the edge case here:

1. Alice implements her MExtension contract (for example, [MYieldFee.sol](#)).
2. The contract is removed from the approved earners list by the TTG Registrar at T1 and Alice is notified.
3. As soon as an attacker sees the removal from the approved earners list occur, they can call `stopEarning(account)` directly on the MToken contract to stop any MToken from accruing (at T1 itself).
4. After 10 blocks or more (2 minutes or more), Alice calls `disableEarning()` on her MExtension at T2, which is intended to [1] update the index of the extension [2] call `stopEarning()` on the MToken contract.
5. When the index is updated, the `latestRate` is retrieved but most importantly the time duration is now overestimated. This is because the index calculations in [MYieldFee.sol](#) uses `block.timestamp (T2) - lastUpdatedTimestamp`. Since accrual stopped at T1, T2 - T1 is the overestimated time, which leads to overminting of the yield.

**Recommendations:** It is recommended to disallow index updates when the MExtension is not an approved earner on the MToken contract.

**Customer's response:** Acknowledged.

## Informational Issues

### I-01. Storage access inconsistency

**Description:** The `swapOut` function in the `UniswapV3SwapAdapter.sol` contract reads the `swapRouter` variable multiple times instead of caching it to a local variable, creating inconsistent code patterns between `swapIn` and `swapOut` functions.

**Recommendations:** Cache the `swapRouter` immutable variable at the beginning of the `swapOut` function to match the pattern used in `swapIn`. Add address `swapRouter_ = swapRouter;` after validation checks and use the cached value throughout the function.

**Customer's response:** Fixed in [00981b2](#).

### I-02. NatSpec comment in `_revertIfNotBlacklisted` is misleading

**Description:** In the `Blacklistables.sol` contract the NatSpec comment of the `_revertIfNotBlacklisted()` function does not accurately describe the behavior of the function. It has been duplicated from the `_revertIfBlacklisted()` function and currently says the function "**reverts if an account is blacklisted**", but the logic does the opposite, it reverts if the account is not blacklisted.

JavaScript

```
/**
 * @notice Internal function that reverts if an account is blacklisted.
 * @param $ The storage location of the blacklistable contract.
 * @param account The account to check.
 */
function _revertIfNotBlacklisted(BlacklistableStorageStruct storage $, address account)
internal view {
    if (!$.isBlacklisted[account]) revert AccountNotBlacklisted(account);
}
```

**Recommendations:** Update the NatSpec to match the logic.

**Customer's response:** Fixed in [2b485b3](#).

### I-03. Remove redundant Interface IUniswapV3SwapCallback

**Description:** Interface `IUniswapV3SwapCallback` is never utilized and its `uniswapV3SwapCallback()` function definition is never implemented, introducing redundancy in the codebase.

**Recommendations:** Comment out the interface or remove it.

**Customer's response:** Fixed in [38e291a](#).

### I-04. Remove or utilize unused events SwappedIn and SwappedOut

**Description:** The `IUniswapV3SwapAdapter` interface defines two events that are never emitted:

```
JavaScript
    event SwappedIn(address indexed inputToken, uint256 inputAmount, uint256
baseOutputAmount);

    event SwappedOut(address indexed outputToken, uint256 baseInputAmount, uint256
outputAmount);
```

Unused events increase contract size without providing value.

**Recommendations:** Remove the unused `SwappedIn` and `SwappedOut` events from `IUniswapV3SwapAdapter` interface, or add event emissions in the `swapIn()` and `swapOut()` functions.

**Customer's response:** Fixed in [00981b2](#).

### I-05. Unused exactOutput functionality

**Description:** In the interface `IV3SwapRouter.sol`, the function definitions and structs from the code snippet below are un-implemented and not used in the `UniswapV3SwapAdapter` contract. Since the `UniswapV3SwapAdapter` does not implement this functionality, the `SwapFacility` only offers exact input functionality.



JavaScript

```
struct ExactOutputSingleParams {
    address tokenIn;
    address tokenOut;
    uint24 fee;
    address recipient;
    uint256 amountOut;
    uint256 amountInMaximum;
    uint160 sqrtPriceLimitX96;
}

function exactOutputSingle(ExactOutputSingleParams calldata params) external payable
returns (uint256 amountIn);

struct ExactOutputParams {
    bytes path;
    address recipient;
    uint256 amountOut;
    uint256 amountInMaximum;
}

function exactOutput(ExactOutputParams calldata params) external payable returns (uint256
amountIn);
```

**Recommendations:** Comment out the respective definitions or remove them from the interface.

**Customer's response:** Fixed in [38e291a](#).

# Disclaimer

Even though we hope this information is helpful, we provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the contract is secure in all dimensions. In no event shall Certora or any of its employees be liable for any claim, damages, or other liability, whether in an action of contract, tort, or otherwise, arising from, out of, or in connection with the results reported here.

## About Certora

Certora is a Web3 security company that provides industry-leading formal verification tools and smart contract audits. Certora's flagship security product, Certora Prover, is a unique SaaS product that automatically locates even the most rare & hard-to-find bugs on your smart contracts or mathematically proves their absence. The Certora Prover plugs into your standard deployment pipeline. It is helpful for smart contract developers and security researchers during auditing and bug bounties.

Certora also provides services such as auditing, formal verification projects, and incident response.