SHERLOCK

Security Review For M0

Collaborative Audit Prepared For:	M0
Lead Security Expert(s):	hildingr
	Obsidian
Date Audited:	November 10 - November 13, 2025

Introduction

The `JMIExtension` is an upgradeable ERC20 token contract designed to wrap the `\$M` token into a non-rebasing equivalent. It incorporates a "Just Mint It" (JMI) backing model, allowing users to mint the extension token by depositing either `\$M` or other approved collateral assets. The contract assumes a 1:1 peg between the deposited assets and `\$M`. All yield generated from the underlying `\$M` is consolidated and can be claimed by a single, designated yield recipient. The contract includes mechanisms for pausing functionality, managing asset freezes, and setting caps on the amount of non-`\$M` collateral.

Scope

Repository: m0-foundation/common

Audited Commit: 613d2d95a476612270324ecbe92317eb5e9bc98f

Final Commit: 613d2d95a476612270324ecbe92317eb5e9bc98f

Files:

- src/libs/TransferHelper.sol

Repository: m0-foundation/m0-evm-m-extensions

Audited Commit: 49cc37acbb21d414dd64a8116504f4f747cc1c84

Final Commit: 7609e5e9151589e2078d91d0d95e1d0fef4278ce

Files:

- src/components/pausable/IPausable.sol
- src/components/pausable/Pausable.sol
- src/projects/jmi/IJMIExtension.sol
- src/projects/jmi/JMIExtension.sol
- src/swap/interfaces/ISwapFacility.sol
- src/swap/SwapFacility.sol

Findings

Each issue has an assigned severity:

- High issues are directly exploitable security vulnerabilities that need to be fixed.
- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.

- Low/Info issues are non-exploitable, informational findings that do not pose a security risk or impact the system's integrity. These issues are typically cosmetic or related to compliance requirements, and are not considered a priority for remediation.

Issues Found

High	Medium	Low/Info
0	3	4

Issues Not Fixed and Not Acknowledged

High	Medium	Low/Info
0	0	0

Issue M-1: Using USDT as an asset in the JMIExtension will fail [RESOLVED]

Source: https://github.com/sherlock-audit/2025-11-m-zero-extension-nov-10th/issues/8

Summary

The SwapFacility::_swapInJMI() function uses `IERC20.transferFrom` to obtain the `asset` from the user. The issue is that `USDT` does not follow the `IERC20` spec (as it doesn't have a return value) causing the `transferFrom` call to fail with an `EvmError`.

Vulnerability Detail

The JMIExtension enables an extension token to be minted by depositing either M or an allowed `asset` token. This occurs in the `wrap()` function which can only be called by the `SwapFacility` contract.

The `SwapFacility::_swapInJMI()` function uses the following logic to obtain tokens from the user:

```
IERC20(asset).transferFrom(msg.sender, address(this), amount);
IERC20(asset).approve(extensionOut, amount);
```

This will revert for USDT since it does not return any values upon `transferFrom`, while the IERC20 interface expects a bool to be returned.

Impact

USDT (on Ethereum mainnet) will not be usable as one of the assets for any JMI extensions.

Code Snippet

https://github.com/sherlock-audit/2025-11-m-zero-extension-nov-10th/blob/819013aacec661ebc7f32057b37897807acc0e3a/m0-evm-m-extensions/src/swap/SwapFacility.sol#L364-L365

Tool Used

Manual Review

Recommendation

Consider using the `safeTransferFrom` and `safeApprove` functions from the TransferHelper library, this will ensure USDT can be transferred succesfully.

Issue M-2: Fee on transfer tokens will lead to incorrect accounting [RESOLVED]

Source: https://github.com/sherlock-audit/2025-11-m-zero-extension-nov-10th/issues/9

Summary

Using a fee on transfer token would lead to incorrect accounting. USDT is an example of a token with fee on transfer functionality that is currently turned off.

Vulnerability Detail

When `_wrap()` is called we calculate amount of JMI to mint with the following code

https://github.com/sherlock-audit/2025-11-m-zero-extension-nov-10th/blob/819013aacec661ebc7f32057b37897807acc0e3a/m0-evm-m-extensions/src/projects/jmi/JMIExtension.sol#L264-L266

The the token has a fee on transfer the actual amount will be `<amount` as a fee has been taken.

Impact

We mint more jmi tokens than expected creating a small unbalance with each wrap.

Code Snippet

Tool Used

Manual Review

Recommendation

Mint exactly the amount received during the transfer to support transfer on fee tokens

Issue M-3: Pause functionality in the upgraded `SwapFacility` will not work [RESOLVED]

Source: https://github.com/sherlock-audit/2025-11-m-zero-extension-nov-10th/issues/11

Summary

The currently deployed proxy of the `SwapFacility` is at 0xB6807116b3B1B321a390594e31ECD6e0076f6278.

The newer implementation of the `SwapFacility` incorrectly introduces the `__Pausable_init()` in the `initialize()` function, which will not be callable.

Vulnerability Detail

The new `__Pausable_init()` logic was added to the original `initialize()` function:

```
    function initialize(address admin, address pauser) external initializer {
+        __Pausable_init(pauser);
        __ReentrancyLock_init(admin);
    }
```

However since this function has the `initializer` modifier, it can only be called once (and already has been called in the live contract).

Impact

Since the `initialize()` function can't be called again, the pausable functionality cannot be initialized, so the `SwapFacility` cannot be paused.

Code Snippet

https://github.com/sherlock-audit/2025-11-m-zero-extension-nov-10th/blob/976faa39aa3b5a5ab8240e58a8cf2866aecf64a3/SwapFacility.sol#L83

Tool Used

Manual Review

Recommendation

Consider using another function with the `reinitializer` modifier, where the `__Pausable_init(pauser)` logic is included.

Issue L-1: Dust amount left in the contract [ACKNOWLEDGED]

Source:
https://github.com/sherlock-audit/2025-11-m-zero-extension-nov-10th/issues/10

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

Dust amount is locked in the contract due due to rounding all assets to 6 decimals.

Vulnerability Detail

All assets we wrap an asset with more than 6 decimals it will be rounded down

https://github.com/sherlock-audit/2025-11-m-zero-extension-nov-10th/blob/819013aacec661ebc7f32057b37897807acc0e3a/m0-evm-m-extensions/src/projects/jmi/JMIExtension.sol#L266

https://github.com/sherlock-audit/2025-11-m-zero-extension-nov-10th/blob/819013aacec661ebc7f32057b37897807acc0e3a/m0-evm-m-extensions/src/projects/jmi/JMIExtension.sol#L354-L356

https://github.com/sherlock-audit/2025-11-m-zero-extension-nov-10th/blob/819013aacec661ebc7f32057b37897807acc0e3a/m0-evm-m-extensions/src/projects/jmi/JMIExtension.sol#L377-L385

when we replace assets with M tokens we will convert back to the original decimals

https://github.com/sherlock-audit/2025-11-m-zero-extension-nov-10th/blob/819013aacec661ebc7f32057b37897807acc0e3a/m0-evm-m-extensions/src/projects/jmi/JMIExtension.sol#L293C31-L293C58

This will leave dust amount in the contract

Impact

Dust amount left in contract

Tool Used

Manual Review

Recommendation

Add a function for admin to take dust amounts but since the current plan is to use only stablecoins the dust amount will be insignificant

Issue L-2: The `TransferHelper` library doesn't check if the `token` has code [RESOLVED]

Source:
https://github.com/sherlock-audit/2025-11-m-zero-extension-nov-10th/issues/12

Summary

The `TransferHelper` library doesn't check if the `token` has code, which causes calls to non-existent tokens to succeed.

Vulnerability Detail

If the `token` has no code, `safeTransferFrom` will still succeed even though no transfers occurred. This is because low level calls return `success==true` if the callee has no code.

Impact

There is no impact currently (since managers would have to whitelist the asset in the JMI extension, and are trusted to not whitelist undeployed contracts), but can have an impact in future use cases of this library.

Here is an example vulnerability that can arise due to this root cause.

Code Snippet

https://github.com/sherlock-audit/2025-11-m-zero-extension-nov-10th/blob/819013aacec661ebc7f32057b37897807acc0e3a/common/src/libs/TransferHelper.sol#L16-L21

Tool Used

Manual Review

Recommendation

Consider using OpenZeppelin's `SafeERC20` library which mitigates this vulnerability.

Issue L-3: Incorrect natspec in `JMIExtension::_beforeWrap()` [RESOLVED]

Source:
https://github.com/sherlock-audit/2025-11-m-zero-extension-nov-10th/issues/13

Summary

The natspec in `JMIExtension::_beforeWrap()` states:

```
* @param amount    The amount of extension tokens to mint.
```

The issue is that the `amount` parameter actually represents the amount of assets being deposited, which can be different to the amount of extension tokens to mint (due to differences in decimals).

Code Snippet

https://github.com/sherlock-audit/2025-11-m-zero-extension-nov-10th/blob/819013aacec661ebc7f32057b37897807acc0e3a/m0-evm-m-extensions/src/projects/jmi/JMIExtension.sol#L218

Tool Used

Manual Review

Recommendation

Consider changing the natspec to ”The amount of asset tokens to be deposited”, or similar.

Issue L-4: canSwapViaPath ignores pause state, returning true even when swaps are paused [RESOLVED]

Source:
https://github.com/sherlock-audit/2025-11-m-zero-extension-nov-10th/issues/14

Summary

`canSwapViaPath` is a view function that returns whether a given `swapper` can execute a swap from `tokenIn` to `tokenOut` via `SwapFacility`.

The issue is that the function will return true, even when the SwapFacility or tokenIn/tokenOut extension contracts are paused.

Code Snippet

https://github.com/sherlock-audit/2025-11-m-zero-extension-nov-10th/blob/819013aacec661ebc7f32057b37897807acc0e3a/m0-evm-m-extensions/src/swap/SwapFacility.sol#L231

Tool Used

Manual Review

Recommendation

Consider checking if the contracts are paused, and returning false in that case

Disclaimers

Sherlock does not provide guarantees nor warranties relating to the security of the project.

Usage of all smart contract software is at the respective users’ sole risk and is the users’ responsibility.